Scalable Readout for Proton CT

A thesis by

Øistein Jelmert Skjolddal

for the degree of

Master of Software Engineering

Department of Computing, Mathematics and Physics,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

June 2020

Abstract

A collaboration centred around University of Bergen, Western Norway University of Applied Sciences and Haukeland University Hospital is developing a pixel based proton computed tomography(pCT) prototype. In clinical use, this technology can help reduce the uncertainties and errors associated with the conversion of the tissues stopping power from the photon radiation used in the dosage planing to the proton radiation used in treatment. Additionally, the use of the treatment device can help remove uncertainties connected to the location of the tumor accumulated since the dosage plan was created.

This thesis covers the work done to create two software modules processing a data stream received from the detector. The modules are developed in C++ 17 and runs on a 64 bit CentOS system. This work is done as a part of a larger effort to improve the readout software and make it more flexible, for this, it utilises a new template based control module. This allows easy reconfiguration of the software's functionality by changing the included modules.

This thesis takes a look at the performance requirements and compares this to the measured performance of the software in various configurations. The tests shows that with binary output, the simulated data for a 1 second beam would be processed and written to disk within 1,4 seconds. Using the root writer module to enable analysis of the output,this increases to 4.8 seconds.

The memory usage is also measured, showing a 3,5 increase in memory use when the root writer is employed and a 1,88 increase when the output is to a binary file giving an indication on how much buffer memory is required in the prototype system.

The software is designed to scale with the detector, and the software model provide a total processing speed designed to follow the speed of the layer receiving the most data.

Preface

The work on this thesis have largely been performed at the Department of Physics and Technology at the University of Bergen between August 2019 and until the closure of the institute in May 2020. There has been an active research collaboration working on the project driving it forwards.

Acknowledgements

Firstly I must thank my two supervisors Professor Håvard Helstrup and Professor Johan Alme for their help and guidance. I would also like to add Matthias Richter who have overseen the software section of the project and provided valuable insight and guidance.

I would also like to thank my fellow students Thea Bodova and Alf Herland who have been working on the project along me and for providing help and ideas. And the PhD students Ola Gøtvik and Viljar Eikeland who have provided insights and ideas. And Kristin for putting up with my absence.

Øistein Jelmert Skjolddal
Bergen, Juni 2020

Contents

List of Figures

List of Tables

Glossary

active-pixel In an active-pixel sensor, each pixel sensor cell has a photodetector and one or more active transistors.

ASCII American Standard Code for Information Interchange is a 7 bit, now extended to 8 bits (UTF-8), standard for encoding text.

attenuation In physics: The gradual loss of energy by passing through a medium.

Bragg peak The region of high energy deposition and delivered dose at the end of a charged particles range.

calorimeter In physics: A devise measuring the energy of a particle.

Ethernet A common family of computer networking technologies used in local area networks (LAN) and wide area networks (WAN).

Inner Barrel mode A operating mode for the ALPIDE chips designed for the high intensity environment in the innermost layer of the ALICE Inner Tracking system detector.

K28.5 COMMA word A control symbol in 8b/10b encoding. It is in the class of "comma symbols" which are used for synchronization.

LVDS Low-voltage differential signaling is a physical layer specification that enables operations at low power and that can run at very high speeds using inexpensive twisted-pair copper cables.

Monolithic Monolithic sensor contains its own readout electronics.

offline analysis Data analysis preformed on a stored file.

on-board imaging systems A radiation therapy device capable of producing CT images for patient positioning.

online analysis Data analysis preformed directly on the data from a detector.

pencil beam Used to describe a steerable beam of radiation or charged particles.

phantom Is a conceptual representation of a patient containing elements to simulate soft tissue, bone, metal implants etc..

scintillator A scintillator is a material that exhibits the property of luminescence when excited by ionizing radiation.

slab A 1/4 of a layer in the Bergen pCT detector, There is a top slab and a bottom with slight differences in their backing plate.

string 9 ALPIDE chips mounted on a FLEX cable. This unit forms the basis for the Bergen pCT detector.

strobe In the ALPIDE setting, A data readout performed at a regular interval.

subcutaneous Beneath the skin.

Acronyms

ALICE A Large Ion Collider Experiment.
ALPIDE Alice Pixel Detector.
ATLAS A Toroidal LHC ApparatuS.

CI Continuous Integration.
CT Computed Tomography.

FoCal Forward Calorimeter.

HU Hounsfield Units.

ITS Inner Tracking System.

LHC Large Hadron Collider.

OBI on-board imaging.

pCT proton Computed Tomography.
pDTP pCT Data Transfer Protocol.
PRaVDA Proton Radiotherapy Verification and Dosimetry Applications.
pRU Proton Readout Unit.

RSP Relative Stopping Power.
RU Readout Unit.

SOBP Spread Out Bragg Peak.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

CHAPTER 1

Introduction

For the past 20 years, particle therapy has seen increasing use as a treatment modality within radiation therapy cancer treatment [7, p.2]. This has been followed up by the Norwegian parliament and government who, through bipartisan agreements, have worked to create national centres for particle therapy [18, p.7, p.15] [37]. Currently, treatment is purchased from foreign treatment centers and there is a desire to nationalise this capability to increase availability [37].

In conjunction with this, University of Bergen, Western Norway University of Applied Sciences, Haukeland University Hospital and others, have an ongoing research project to develop a proton Computed Tomography (pCT) detector suite. The project have looked at different kinds of treatment types, sensor arrangements and available technologies [29]. The project has opted for the ALPIDE pixel sensor developed for the Inner Tracking System (ITS) at the ALICE experiment [9, Section 1] upgrade along with custom readout and control hardware and software. There are currently other projects using different sensor technologies. The two most notable is the proof of concept detector developed by a US based research group centering around Santa Cruz Institute for Particle Physics and Loma Linda University and the experimental detector by the British PRaVDA consortium [40] [4].

The US based research group uses a combination of silicon-strips detectors for the front and back tracking sensors, one on each side of the object to be scanned, followed by a scintillator calorimeter to determine the particles energy loss. The use of a scintillator based calorimeter greatly reduces cost and complexity as this technology is well understood, mature and cheap. However, each scintillator element can only read out a single particle at a time greatly limiting the spatial resolution of the detector. This limits the potential particle rate of the beam, thus increasing the time required to collect the necessary particle tracks for reconstruction. It is expected that this detector can potentially support data capture speeds in the 5 to 10 minute range [33, section.1,

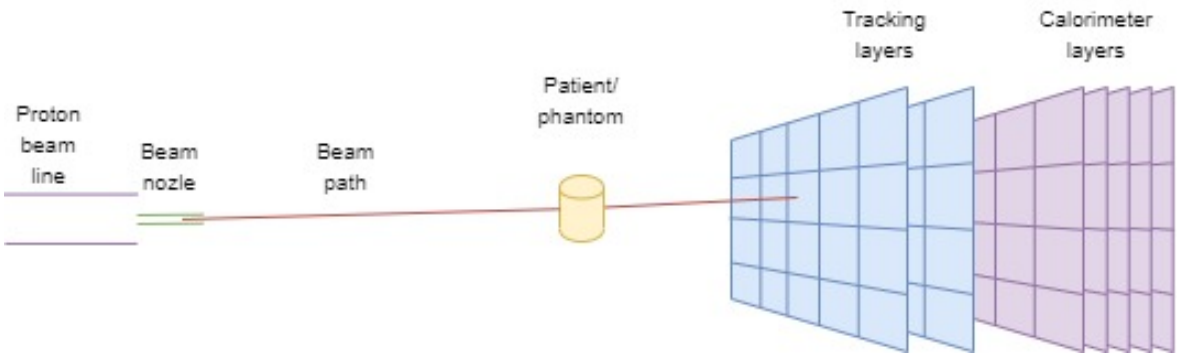


Figure 1.1: The novel approach of the Bergen pCT setup using a pencil beam as the beam origin and only a rear tracker before the calorimeter.

2, 3].

The PRaVDA project in the UK is using only silicon strip detectors for both the front tracers and calorimeter. The detector is based on sensor technology developed for the CERN Large Hadron Collider (LHC) ATLAS experiment [26, section 1]. The detector has an effective sensor area of 93 mm x 96 mm containing 2048 detection strips [26, section 3.1]. Each tracker contains 3 silicon strip detectors offset at 60 degrees to create virtual pixels by correlating the readout. [26, section 3.2]. Using a silicon-strip based calorimeter greatly increases the spatial resolution in the detector. This makes it possible to track far more particles per readout cycle and a higher intensity beam can be used. The higher readout rates provides a more effective detector reducing the time required to gather the necessary particle tracks for reconstruction. This detector has been used used to generate a pCT image of a phantom. A series of 180 projections were taken, each projection required 1 second to take [26, section 4]. This give the detector a theoretical capture speed of 3 minutes.

The Bergen pCT detector is using the ALPIDE sensor detectors developed for the LHC ALICE experiment. The ALPIDE chip is a small, active pixel sensor. The chip provides excellent spatial and temporal resolution for the detector. This enables the detector to track a large number of particles in a short time interval. The potential for a large number of simultaneous particles coupled with short readout cycles will reduce the time necessary to collect sufficient data for image reconstruction.

Another novel feature of this detector is the use of a pencil beam as a fixed position for the particle origin, instead of using a front tracker like the other two designs. This concept is shown in figure 1.1.

1.1 Background and Motivation

proton Computed Tomography and the combination of an imaging and treatment device offers several advantages over photon based CT where the the CT scanner and the treatment device is separated. Since protons deposit most of their energy right before the particle comes to rest, most of the radiation energy will be deposited in the calorimeter instead of in the patient. Compared to conventional photon CT, this will result in a lower radiation dose for the patient. Using the treatment machine to generate a CT image will have several benefits.

The proton Computed Tomography capability will enable in situ and on demand imaging. This will limit the errors and uncertainties associated with translating the spatial coordinates from the patient CT to those of the treatment machine. This removes the errors associated with the translation, erroneous measurement and changing patient position, in both space and time. An additional benefit from this is that dosage planning can be easily verified, removing spatial uncertainty of the tumour location. [35]

The use of different radiation types for image generation and treatment necessitates a conversion introducing both errors and uncertainties. Photon based CT images uses Hounsfield Units (HU) showing the stopping power of photons in the imaged tissue. Protons has a different interaction with the tissue and the description of this interaction uses Relative Relative Stopping Power (RSP) in water equivalents. This introduces the necessity a HU to RSP conversion introducing additional uncertainties as well as errors based on the different tissue/bone interaction of photons and protons [13, section 3.3].

The Bergen pCT is currently in the process of verifying the 9 ALPIDE chip string (string) and associated production processes, concepts, hardware, software, procedures and readout chain. Concurrently, other groups are running simulations and tweaking of the detector design. Once the full 9 chip string and associated processes, hardware and software has been verified, the project can move to start building and testing the first full 12 string detector layer.

The excellent spatial resolution of the planed detector will provide a large amount of data to be offloaded and processed. A concurrent master thesis by Alf Herland is concerned with the network and data transfer. While the master thesis of Ola Slettevoll Grøttvik [31], his concurrent pHD work and a concurrent master thesis by Tea Bodova is working on the initial detector data offload.

This thesis is a part of the Bergen pCT collaborations effort to develop software that can handle readout of the necessary data volumes generated by the detector. The capabilities of the ALPIDE sensor will allow the detector created by the Bergen pCT collaboration to reduce the time necessary to generate a CT image in comparison to competing technologies.

1.2 Problem Description

This thesis is a continuation of the work done by Håkon Andreas Underdal [14] and Annar Eivindplass Hilde [2] in 2018-2019 and the proof of concept software they created. This thesis tackles the challenge of transforming that prototype into two more effective software modules in the context of a new control structure. The Processing Module organises the incoming data stream and removes faulty data in preparation of further processing. The Output Module extracts the information in the data stream and stores it in a usable format. The setup can be seen in the lower half of figure 1.2.

The two new modules will function as the preliminary step in in the reconstruction process. However, as the reconstruction software is still being prototyped, the most important function of the two modules is to help locate bugs in the system during development and convert the data stream into a format more suitable for analysis and human interaction.

This thesis also addresses the processing speed requirements for the modules with regards to both online and offline data processing and recourse use. During data taking, each detector layer can potentially generate a significant amount of data and the hardware available for processing must be scaled accordingly. The detector is based on a layered stack of sensor planes, each layer having dedicated readout hardware. The goal is to have the software scale in the same manner, by having a dedicated software instance for each detector layer. This is illustrated in the upper half of figure 1.2.

The processing and output modules receives data from file or from a Network Module. The Network Module receives its data from a 10 Gb/s Ethernet connection from the readout electronics of a single detector layer. The Processing Module and Output Module should be able to process this data in a reasonable time frame and must process it before any buffer overruns occur.

The work in this thesis has followed an agile approach, focusing on finishing small and discrete sub modules encompassing a specific functionality. This helps development and is conducive to changes, present and future as well as potential code reuse. This development strategy is important since project is still being developed with changing hardware and software requirements, so all software will also have to support changes. The two most important considerations are changes in the physical detector layout and the reconstruction software currently in the concept phase.

For this reason, the software modules will, at least conceptually, be divided in to sub modules with a discrete function aiding development and simplifying changes in the functionality of the overall software.

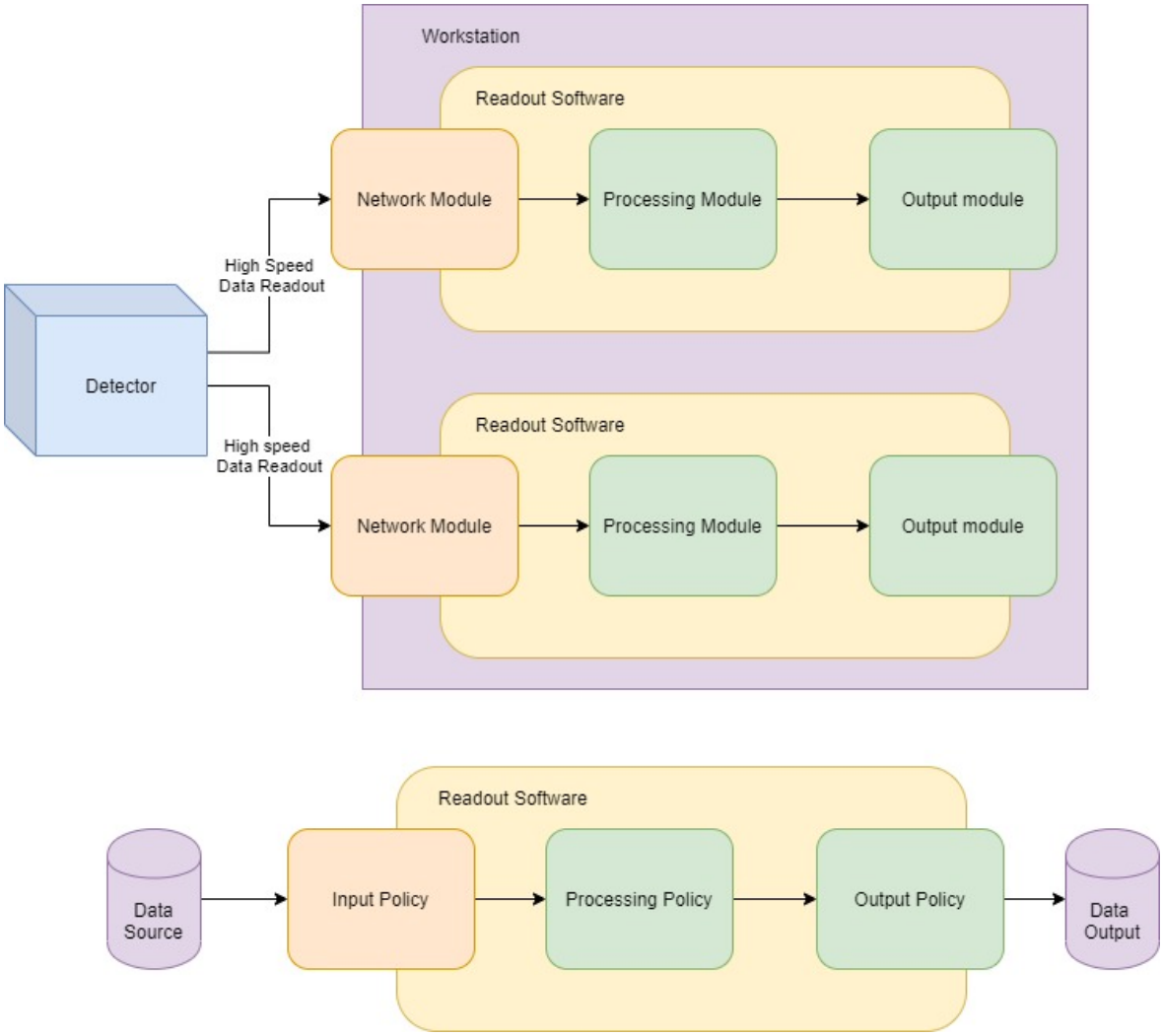


Figure 1.2: General overview of the software.The Top figure shows system scaling with multiple software instances running in parallel. The bottom illustrates the subdivision of a software instance and the 3 templates in the readout chain. The modules relevant for this thesis is marked in green. The network module is a separate Masters Thesis.

1.3 Thesis Outline

The thesis is organised as follows:

Theoretical Background

Provides an overview of the technologies used by this project and provides the reader with a basic and brief summary of the general physics behind the project. This is followed by a general explanation of the ALPIDE chip with a focus on the software elements relevant to this thesis. The chapter ends with a brief summary of the most significant related works.

The Bergen Proton CT System

This chapter provides a more detailed description of the Bergen pCT system. This chapter provides the reader with a description how the system is built up and of how the hardware and software interacts.

Readout Software

This chapter describes the readout software in detail. The focus of this thesis and chapter is on the data processing and preparation for analysis, data visualisation or use by a future reconstruction algorithm.

Analysis and Assessment

This chapter presents the testing methodology, the test results and an assessment of the implemented software.

Conclusion

This chapter presents a summary of the work undertaken and compares this with the thesis goals. This is followed by a a summary of the work recommended in order to improve on the work presented in this thesis.

CHAPTER 2

Theoretical Background

The use of photons in the form of Roentgen radiation was acknowledged already by the turn of the 20th century. First, as an important tool for imaging internal structures but also quickly as an important treatment modality for cancer patients. Radiation therapy using photons has over the past 100 years evolved into today's advanced treatments where multiple beams are used to accurately irradiate the tumor volume while minimizing the exposure of surrounding tissue [23, p.779].

Photons deposit most of the energy right after entering the tissue, thereafter, the energy deposition gradually falls of greatly irradiating the tissue in front of the target and slightly irradiating the tissue behind the target. While all radiation treatments carries the risk of secondary tumors and damage to surrounding tissue, the photons deposition profile creates a large tissue section at risk, both in front of and behind the target volume for any given ray. For high intensity beams, this energy deposit distribution may also result in radiation burns and additional damage to the skin and subcutaneous layers. In dosage planning, it is therefore a primary concern to minimize the dose to healthy tissue while maximizing the dose to the target volume.

The advantageous properties of ionising particles when used in cancer radiation therapy has been known since at least 1946 where the use of both protons and carbon atoms in the treatment of tumors were proposed [36, p.490]. Research on and use of ionising particles in cancer radiation therapy continued [24] but it has only recently started to be widely used and accepted. Per March 2020, there is currently at least 102 particle centers around the world in operation and providing therapy [11]. 32 centers are currently under construction, around 20 of these have a planned startup in 2020 and 29 centres are currently in the planing stages [12] [10].

Especially in the past 10 years, there have been an enormous development in the field. This is demonstrated by the fact, that of the 102 operational centers, only 29

centres are 10 years or older [11].

2.1 Physics Background

The theoretical basis for the project is the properties of proton and heavy ion radiation and their applications in medical imaging and cancer treatment. As part of ongoing research to improve the current technology in medical particle imaging and radiation treatment, bergen pCT collaboration attempts at creating a pCT detector. To this end, a new particle sensor and energy detector is being developed along with dedicated hardware and software for readout, control and system monitoring.

2.1.1 Radiation Therapy

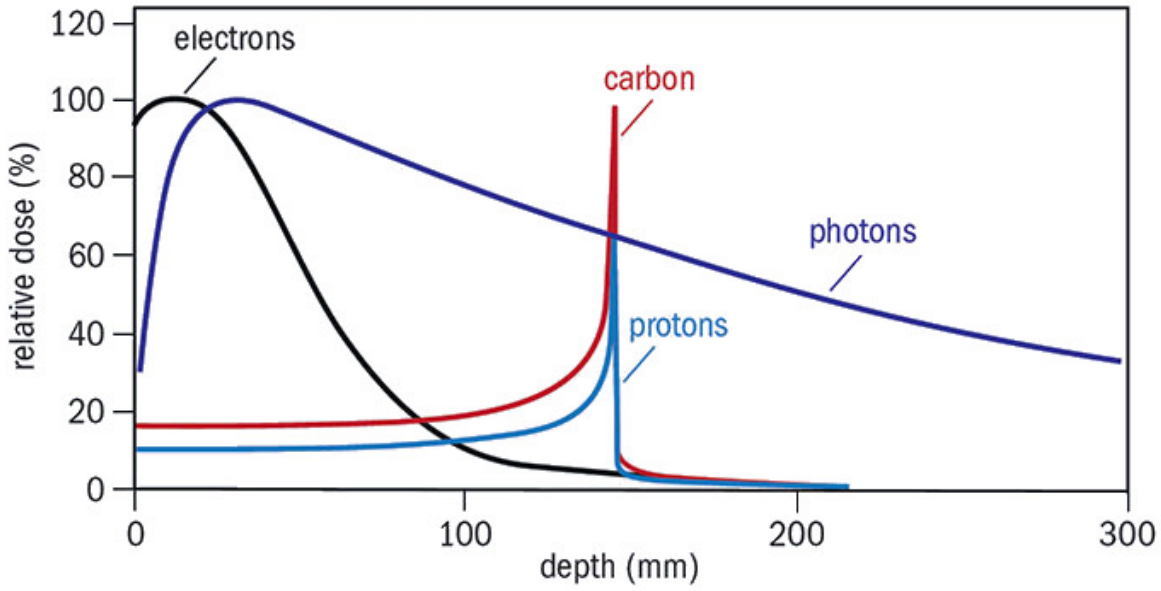


Figure 2.1: Bragg curves for the most common medical radiation types [6]

In radiation therapy, the delivered dosage is dependent on how much energy is deposited in the desired and unhealthy tissue. At the same time, there will be some undesired energy deposition in healthy tissue. The goal is to achieve the minimal lethal dose in the tumor to spare the surrounding tissue unnecessary exposure.

Currently, most radiation treatments use photon radiation. The lower cost and complexity of photon based machines along with the more developed procedures provides a cost effective treatment. Compared to the relatively higher cost of proton treatment,

this ensures that the majority of radiation therapy will continue to be dominated by photon treatment [6].

The properties of energy deposition of heavy ions and protons are shown in figure 2.1. As shown, photons and electrons deposit most of their energy right after entry and gradually loose energy as they go deeper. Protons and heavy ions on the other had, deposit most of their energy right before the particle comes to rest in what is known as the Bragg peak [36]. Conversely, photons and electrons will deposit most of their energy at, or shortly after entry and then gradually lose the remaining energy.

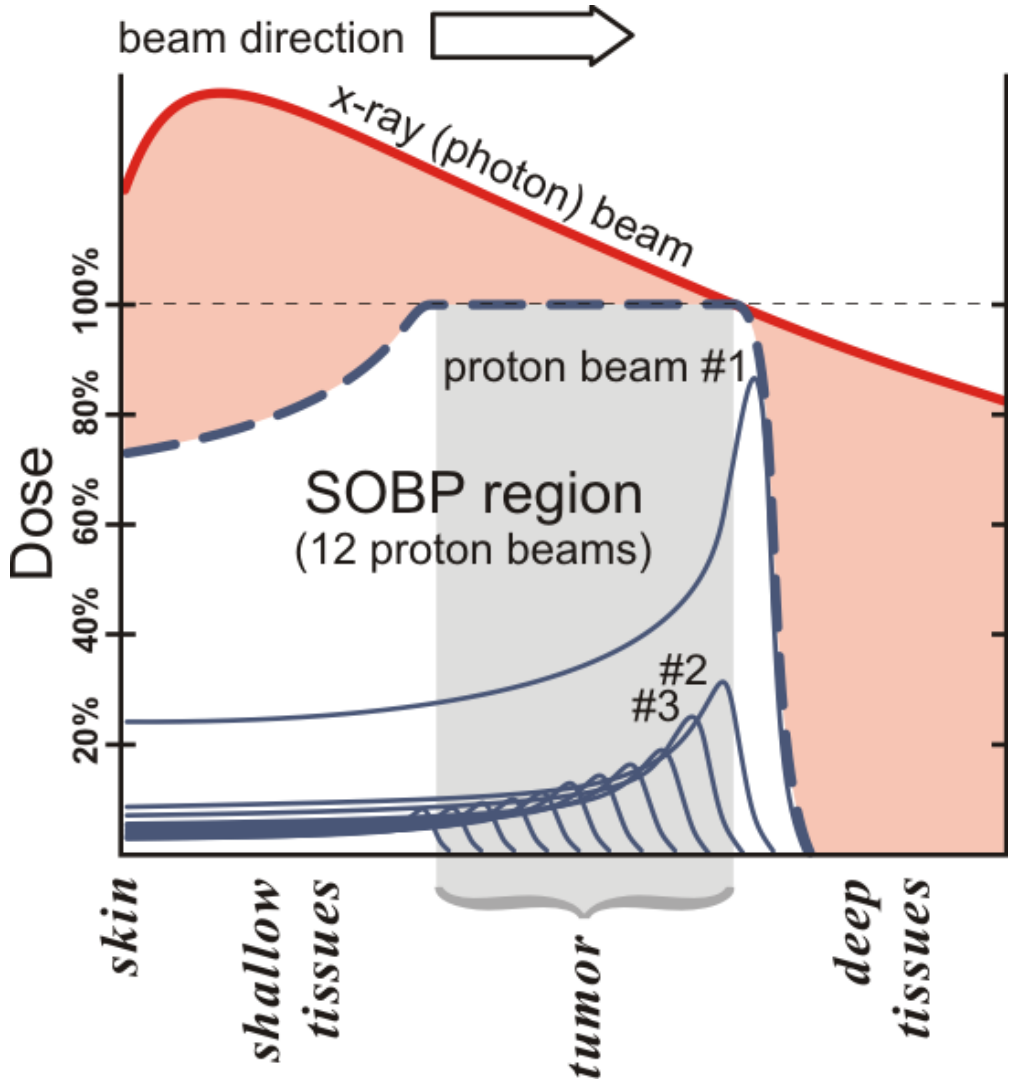


Figure 2.2: Brag curves for photons and range modulated protons (Spread Out Bragg Peak) [27].

Using protons and heavy ions allows for more energy deposited in the desired tissue and less in the surrounding tissue compared to photons. When used in treatment scenarios the beam is range modulated creating a Spread Out Bragg Peak (SOBP).

This effect is shown in figure 2.2 where the SOBP in an ideal case delivers just enough radiation to the tumorous area to be lethal. The area between the photon beam graph and the SOBP graph in figure 2.2 is excess radiation to healthy tissue.

The reduced radiation of healthy tissue can reduce the side effects of the treatment, and it can enable the treatment of more sensitive areas like the eye, brain, spinal cord etc. Especially the sharp drop of in radiation after the peak dose facilitates the treatment of tumors near vital areas.

2.1.2 CT

Computed Tomography gives volumetric models by taking many 1 or 2 dimensional sections of an object. These are then combined to a 3D model of the object. These models have many diagnostical and analytical uses amongst other, the resulting models are used in dosage planning and post treatment tumor control.

Current CT scanners used in treatment planning uses photon radiation for image generation. These CT scanners provides an accurate volumetric map of the tissues photon absorption rate. The attenuation properties in tissue and bone structures are different for the protons used in treatment, relative to the photons used in the image generation. The use of this map (the CT image) in dosage planning will therefore mandate a conversion, introducing errors and uncertainties. Imaging with protons will provide a more accurate map of the tissues proton absorption rate. This image will provide confirmation or correction from the CT model used in the dosage planning. This will reduce the conversion error and uncertainties in the plan.

Due to the dose distribution of protons, pCT has the potential to generate images with a lower imaging dose. This is a limitation in current photon based on-board imaging (OBI) systems where the total radiation dose prohibits the use of CT for positioning and control on a daily basis. This can then be used to reduce the error margins, or at least reduce the potential positioning variations.

2.1.3 Conversion Uncertainties

CT scans provide a volumetric map of the tissue's ability to attenuate photon radiation in Hounsfield Units (HU). Proton radiation has a slightly different attenuation interaction with the physiological elements measured in Relative Stopping Power (RSP). To use the CT scan for proton treatment planning, a conversion from HU to RSP is necessary. This has the effect of introducing both errors and uncertainties as the conversion model is not perfect. By using protons in the imaging process, the resulting image will show the attenuation directly in RSP reducing the errors and uncertainties.

A proton Computed Tomography (pCT) capable on-board imaging systems will be able to augment the

2.1.4 Patient Positioning

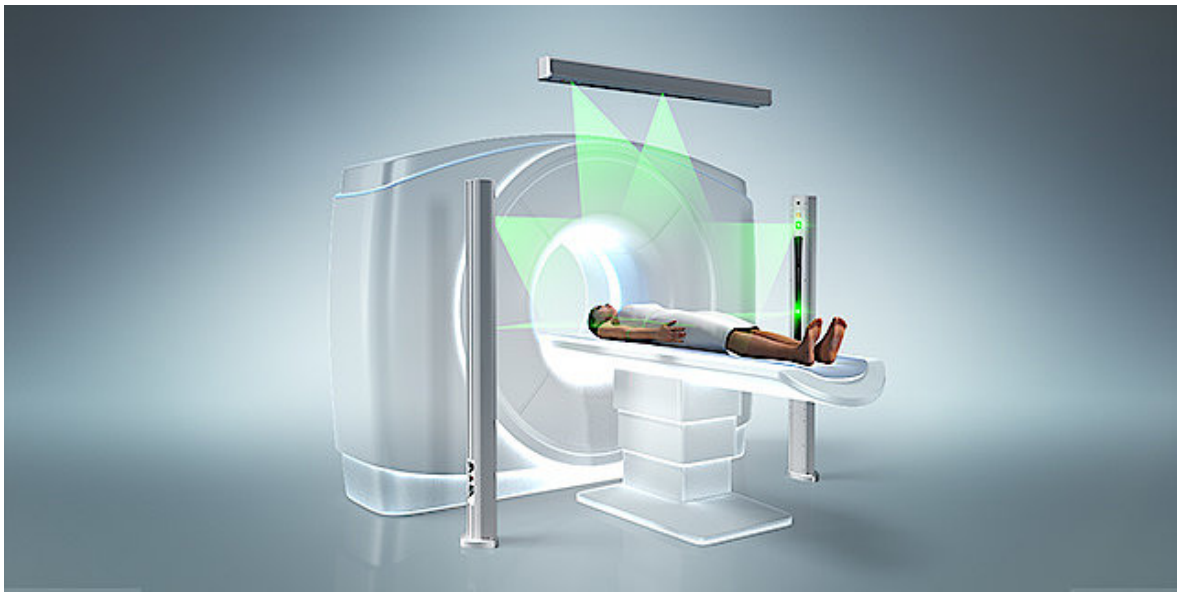


Figure 2.3: Patient positioning using the DORADOnova patient marking system [25]

Since the CT scanner is a separate machine in a separate location, it uses a separate reference frame that needs to be converted and unified. In conventional proton therapy, it is therefore necessary to align the dosage plan and CT model with the patients position in the treatment apparatus. The most common method is surface alignment, most commonly by laser ranging [35, p.599] as shown if figure 2.3 or by cameras. With proton CT treatment, the treatment apparatus can be used to provide CT feedback and positioning during treatment session reducing errors stemming from the uncertainties created by the spatial and temporal distances from the dosage planning device [24, p798-799] [35, p.604] [25].

2.2 Technical Background

The main component of the detector stack is the ALPIDE chip, and while the work conducted in this thesis does not interface directly with the chip, the data stream it generates is acted upon by the processing module. The design of the readout electronics is also required for the extraction of information contained in the data stream.

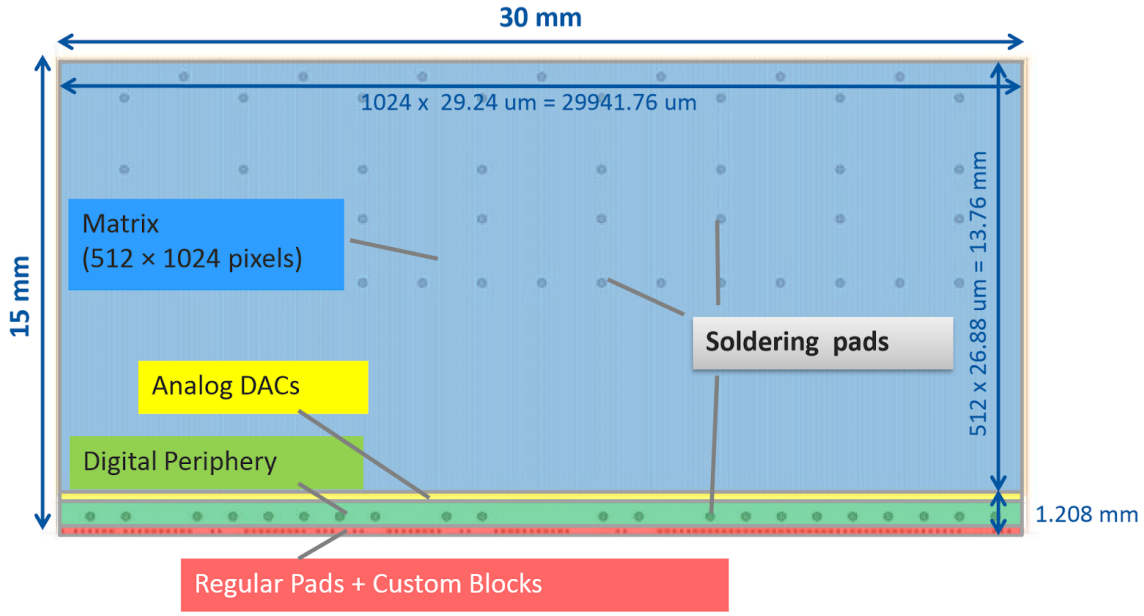


Figure 2.4: ALPIDE Chip general structure and soldering points [8, p. 10]

2.2.1 ALPIDE

The detector uses the ALPIDE sensor, a newly developed Monolithic active-pixel sensor for ionizing particles. The sensor is developed at CERN for the upgrade of the ITS of the ALICE experiment [28]. The pCT detector plan is also very similar to the proposed ALICE Forward Calorimeter (FoCal) [3, section 3.2.1 and section 6.1] and there is synergy effects between the two projects. The ALPIDE sensor provides high accuracy, a low error detection rate, low power consumption and readout only in the case of a hit, which lowers the readout process data rate. These elements makes the sensor useful in a medical setting, with high demands of efficiency and accuracy to ensure patient safety and accurate treatment. ALPIDE sensor clusters can track particle paths and energy deposition with high precision. This forms the basis for more accurate treatment and more information in the treatment setting than was available with previous technology.

The project aims to develop a custom ALPIDE readout pipeline and acompanying hardware that is more suited to the requirements of a pCT scanner in stead of using the existing ITS system. The ITS system is designed create snapshots of an event with a global triggering system base of quick triggering photon detectors to register events. For pCT use, the beam setup is known and controllable so a system based on a fixed strobes is desirable.

Each ALPIDE chip is 30mm * 15 mm (X * Y) and contains 1024 * 512 pixels. Since it is a Monolithic sensor, it has a 1.2mm * 30 mm readout circuit region in the chip

Table 2.1: This table shows the valid ALPIDE words constituting the data format [5, section 3.4.1].

Data Word	Bits	Value (binary)
IDLE	8	1111 1111
CHIP HEADER	16	1010 chip_id[3:0] BUNCH_COUNTER[10:3]
CHIP TRAILER	8	1011 readout_flags[3:0]
CHIP EMPTY FRAME	16	1110 chip_id[3:0] BUNCH_COUNTER[10:3]
REGION HEADER	8	110 region_id[4:0]
DATA SHORT	16	01 encoder_id[3:0] addr[9:0]
DATA LONG	24	00 encoder_id[3:0] addr[9:0] 0 hit_map[6:0]
BUSY ON	8	1111 0001
BUSY OFF	8	1111 0000

periphery providing the chip functionality illustrated in figure 2.4. Each chip is capable of masking noisy or damaged pixels and adjusting the pixel trigger energy levels once these settings have been provided. Due to the radiation environments, these values will need to be regularly updated.

The ALPIDE can be connected and configured in several different ways to meet the needs of for the ALICE experiment. For pCT, the chip will be operating in the Inner Barrel mode. This mode requires that each chip has a dedicated readout link, providing each chip a discrete, programmable 1.2 Gb/s link to the Readout Unit.

The ALPIDE is designed to operate in one of two modes, triggered and continuous. In the triggered mode, a readout of the currently active pixels is performed and this provides a snapshot of the hit status. This readout is performed based on an external trigger signal. In continuous mode, readout is performed during repeating, fixed, intervals (strobes) forgoing the external triggers [5, section 3.3.1 Readout modes].

The chip has two data interfaces, a serial and a parallel data port which both utilizes the same 8 bit oriented protocol. In Inner Barrel mode, only the serial port is utilized. The serial port is 8b/10b encoded using the K28.5 COMMA word control word for clock recovery and data stream synchronisation [5, section 3.3.1 Readout modes]. The COMMA is filtered by the RU and currently not used, but if tests shows that clock recovery will be necessary in the operational prototype it can be included [32, section 1.2].

The ALPIDE data extraction

The ALPIDE chip organises the hit position information in the data in the readout stream after the internal readout hardware. The pixels on the ALPIDE chip is organised into double columns with a shared readout bus extending from the chips digital

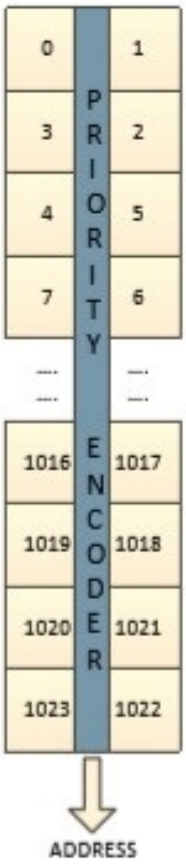


Figure 2.5: ALPIDE pixel indexing of a double column [5, fig 4.5]

periphery. The double columns are numbered left to right with the readout periphery on the bottom. Each of the 512 double columns has a dedicated Priority Encoder circuit responsible for data offload. The priority encoders are in turn organised in 32 regions with a dedicated Region Readout Unit. The whole system with the Region Readout Units containing the 1024x512 pixel matrix is supported by a Top Readout Unit module organising the system. Each double column is read sequentially by the Region Readout Unit while the Region Readout Units are read in parallel by the Top Readout Unit module [5, p.7 & p.9].

All indexing is done with chips digital periphery along the bottom axis. [5, Chapter 4] The pixels in each double column are indexed starting with the top left pixel as 0, moving to top right as 1, 2nd right as 2, 2nd left as 3, 3rd left as 4 and so on, thus moving in an s pattern down the central buss, see figure 2.5.

These 1024 addressable pixels represents two 512 double pixel columns in the hard-

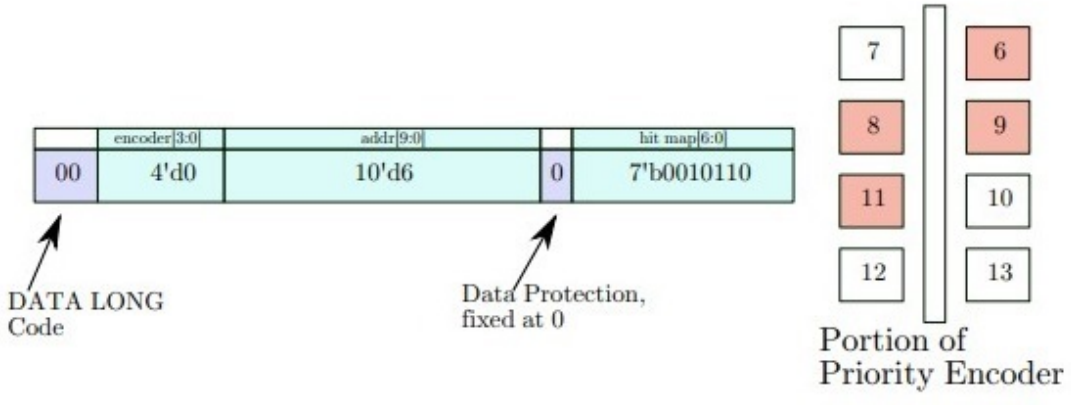


Figure 2.6: An example of a data long with the corresponding pixel cluster [5, fig 3.11].

ware and is the 10 bit address field (addr[9:0]) in the ALPIDE Data Short and Data Long seen in Table 2.1.

The hit map in the Data Long specifies the relative position of other activated pixels immediately following the pixel specified by the 10 bit address. The pattern follows the same s shaped pixel addressing scheme. For instance a hit in the pixels 6, 8, 9 and 11 is shown in figure 2.6. The value 10'd6 is simply pixel number six and would be represented by the binary 10b00'00000110 (number six), the binary 07b0010110 (number 22) is a bit map represents hits at the 2nd, 3rd and 5th pixel after pixel number 6.

The 16 Priority Encoders for each double column are indexed left to right in each Region Readout Unit. The priority encoders are the 4 bit encoder_id value in the Data Short and Data Long as seen in table 2.1. The 32 Region Readout Units are also indexed from left to right and are the 5 bit region_id in the Region Header word as seen in table 2.1.

The Chop Trailer word consists of a 4 bit header and a 4, 1 bit flags. The flags, in bit wise order is:

- BUSY VIOLATION – indication that the chip is replying with an empty data packet due to saturation of data processing capabilities.
- FLUSHED INCOMPLETE – indication that a MEB slice was flushed in order to ensure that the MATRIX always has a free memory bank for storing new events. Observed in Continuous mode only.
- STROBE EXTENDED – indication that the framing window for the event of question was extended due to the reception of an external trigger.

- BUSY TRANSITION – indication that the BUSY was asserted during the readout of the frame in question [5, section 3.4.1].

Of these, only the 2 most significant bits are deemed fatal. This means the bit patterns 0000 (no errors), 0001 (Busy Transition), 0010 (Strobe Extended) and 0011 (Busy transition and Strobe Extended) will not trigger the removal of the frame but any other combination will cause the frame to be removed.

2.3 Related Work

This thesis is a part of a larger effort to create a proton Computed Tomography proof of concept detector to be tested at the planned Norwegian proton centre in Bergen. As parts of an ongoing project, there is a number of student projects, PhD's and other work being undertaken, see the project wiki [29].

There are several finished works that will not be listed here but are available from the project wiki. It is also worth noticing that the project wiki does not contain references for all related work.

The most relevant works for this thesis is:

The concept prototype paper "Proton tracking in a high-granularity Digital Tracking Calorimeter for pCT purposes" by Pettersen et al. This paper provides details on a prototype pixel detector using the Monolithic Active Pixel Sensor chip PHASE2/MIMOSA23. This paper provides a proof of concept and details the rough and approximate construction of the current detector design [17].

The article in review "Design Optimization of a Pixel Based Range Telescope for Proton Computed Tomography" by Pettersen H.E.S. et al [16]. and the 2018 PhD thesis "A Digital Tracking Calorimeter for Proton Computed Tomography" by Pettersen H.E.S [15] introduces the initial concepts around the ALPIDE based detector setup.

Karl Emil Sandvik Bohne - Ethernet-Based Control System and Data Readout for a Proton [22] and Ola Slettevoll Grøttvik [31] - Design of High-Speed Digital Readout System for Use in Proton Computed Tomography describes the initial concepts of the readout hardware, firmware and protocols used by the project.

The work on the proof-of-concept readout and control software created by Håkon Andreas Underdal and Annar Eivindplass Hilde [14] [2]. This software formed the basis for much of the work described in this thesis and it was used for a 2018 beam test.[14, section 6.2]. A test preformed showed that this software had a processing speed of 0,2 MB/s or 1,6 Mb/s.

The Network module being developed concurrently by Alf Herland meant to interface with the software described in this thesis.

CHAPTER 3

The Bergen Proton CT System

3.1 General Overview

The proton computed tomography system consists of a single post-patient particle detector as shown in figure 1.1. The detector is designed for use with a pencil beam as the particle origin, removing the need for a pre-patient detector. Compared to other systems, this simplifies the detector design and reduces cost and complexity.

The detector stack is based on the ALPIDE string, a prototype 9 chip version in a protective cover can be seen in 3.1. The string is composed of a flex cable containing the data and power supply lines, supporting 9 ALPIDE chips. Each chip on the string has a dedicated 1,2 Gb/s data link and a dedicated power conduit.

Because the flex cable takes up nearly as much area as the ALPIDE chips, there is not enough room on a single plane for the necessary coverage of ALPIDE sensors. To have the required sensor coverage, a layer is made up of two half layers. Each half layer contains 6 strings and is shown in figure 3.2.

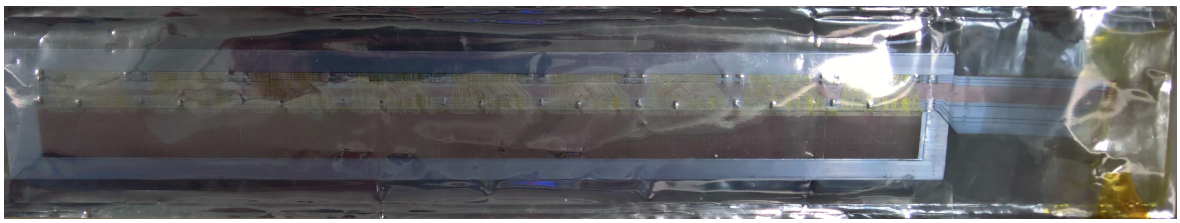


Figure 3.1: A prototype 9 chip string fixed to an aluminium backing plate and covered in anti static protection.

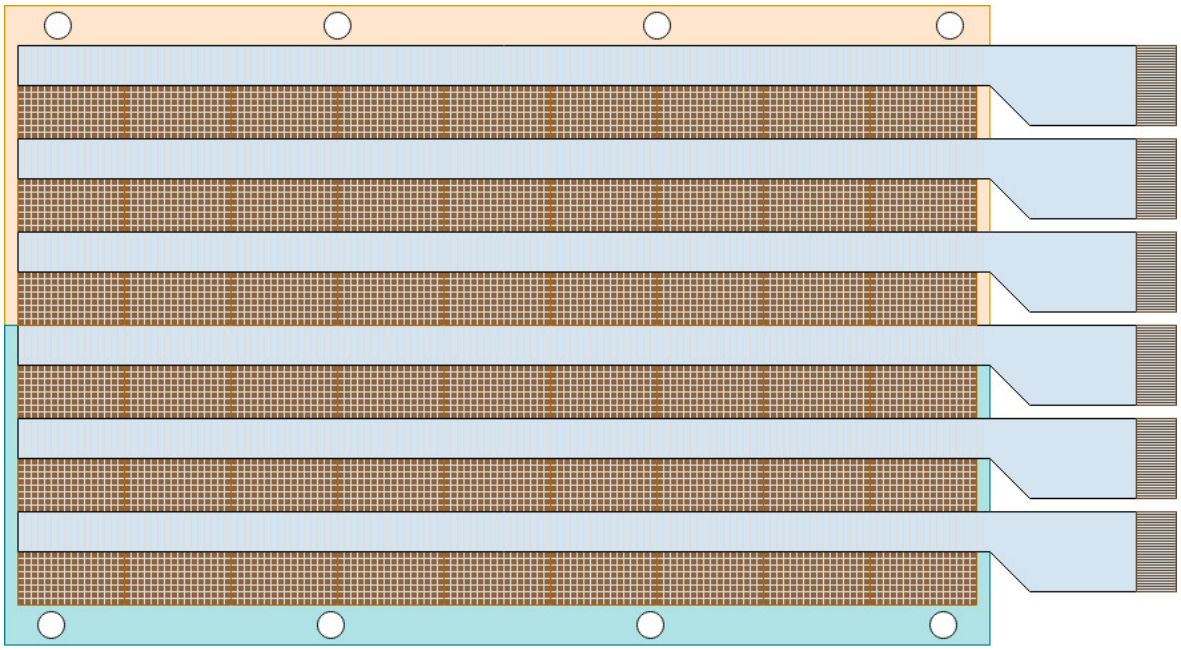


Figure 3.2: A half layer with the calorimeters top slab in orange and bottom slab in green. The tracking layers have a single piece backing plate

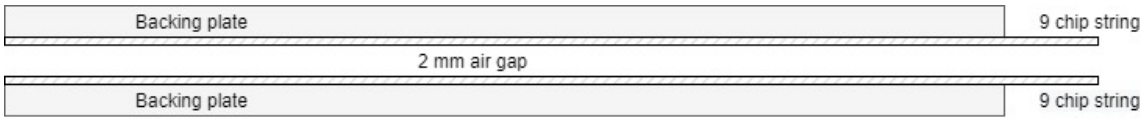


Figure 3.3: A conceptual sketch of a layer. 2 half layers are mounted with the ALPIDEs facing each other and with a spacer supplying a 2mm air gap to prevent damage.

2 layer is put against each other so they create a top half layer and a bottom half layer forming a sandwich structure, the ALPIDE chips facing each other on the inside. A spacer creates an air gap in between to prevent contact and damage 3.3.

This creates a robust unit protecting the fragile chips while minimizing the material difference for the top and bottom. The significant section is the flex cable and chips of the top layer, the air gap has little effect. Each of the half layer consists of 6, ALPIDE strings. With the flex cable taking up slightly less space than the ALPIDE chips, this provides a slight over coverage [34, sections 2.2.3 and 2.2.4]

Each detector layers supported by an outer framework. The framework fixes the 43 sensor layers in their correct alignment and provides support for the initial read-out electronics. Each full layer alternates so that the transition electronics can be distributed along both sides of the detector providing more vertical space for this component. The outer frame also contains the cooling so that each layer can easily be removed for maintenance. The propose of this design is to simplify the construction of

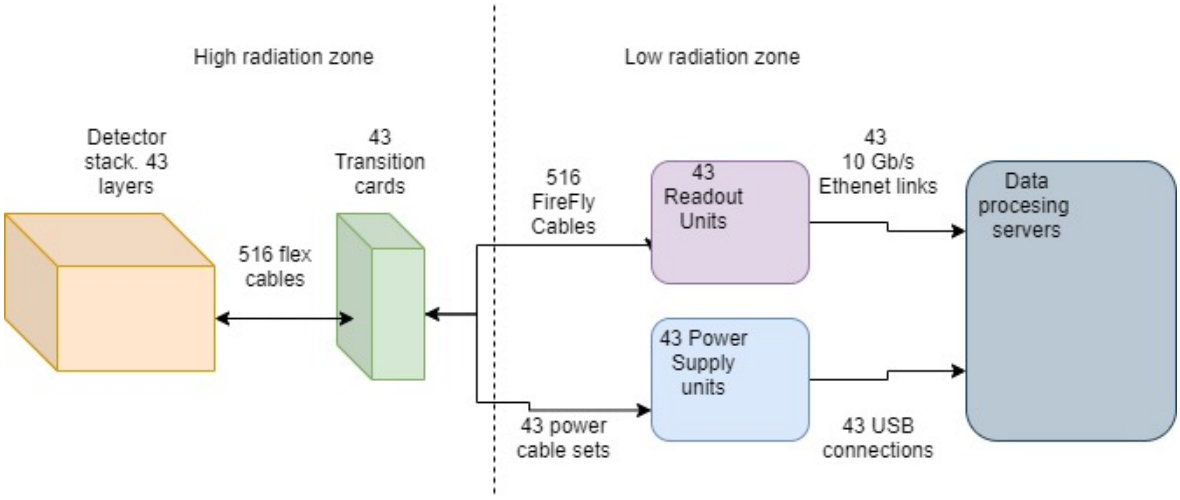


Figure 3.4: General overview of the radiation exposure of the hardware.

the string with only a single design leading to reduction in cost and production time and increased reliability.

The design is based around the detector stack with only simple re transmission and power routing electronics in the radiation area. This lets the more delicate power supply units and readout modules be placed away from the radiation environment removing the need for expensive, radiation hardened electronics 3.4.

3.2 The Detector Stack

The stack consists of 43 layers and the 2 first layers of the detector are the front trackers. These 2 tracking layers have a single piece, carbon fiber backing plate, for each half layer. The carbon fiber allows the particles to penetrate the layer without the loss of much energy giving more data points for tracking and have heat transfer properties similar to aluminium for cooling. The remaining 41 layers have aluminum backing plates functioning as energy absorbers to force the particles to come stop inside the calorimeter section of the detector. For these layers, the half layers are made up of two slabs of 3 strings each, a top slab and a bottom slab, with guide holes for the frame fixture on opposite sides as can be shown in figure 3.2.

Each detector plane is a self contained unit including the transition electronics that can be removed as a single piece. Each plane contains 108 ALPIDE chips set in a 2x6x9 matrix. The flex cables from each layer ends in a radiation hardened transition card [38]. The transition card separates the power and data links and provides power to the ALPIDE chips. The card also provides fuses for the power line to prevent catastrophic failure in the ALPIDE chips if there were to be a short circuit or other serious failure

Figure 3.5: The current Xilinx FPGA test board used in prototyping.

that the control software is unable to respond to. The transition card provides a Samtec FireFly link for the upstream data connection for each string.

The transition card provides a rigid mounting point for the fragile flex cables. The transition card is mounted to the layer and can be removed with the rest of a layer without the need unplug the strings reducing wear on the ALPIDEs.

The power is supplied by a standard, remotely controllable, power supply unit. This unit will regulate the power for each transition card while providing feedback to the control system. The transition card uses the MIC29302 power regulator providing emergency protection against over-current faults, reversed input polarity, reversed lead insertion, over-temperature operation, and positive and negative transient voltage spikes [20].

3.3 The Readout Units

Each layer has a dedicated Readout Unit (RU) handling control of the layer and initial processing of the data stream. The RU is based on an Xilinx FPGA with custom firmware. the Readout Unit is connected directly to the transition card via Fire Fly cables and is placed away from the radiation area so that the electronics does not need to be radiation hardened as per Figure 3.4. A 10 Gb/s Ethernet link provides the upstream interface. The finalised Readout unit will be a simplified and scaled down

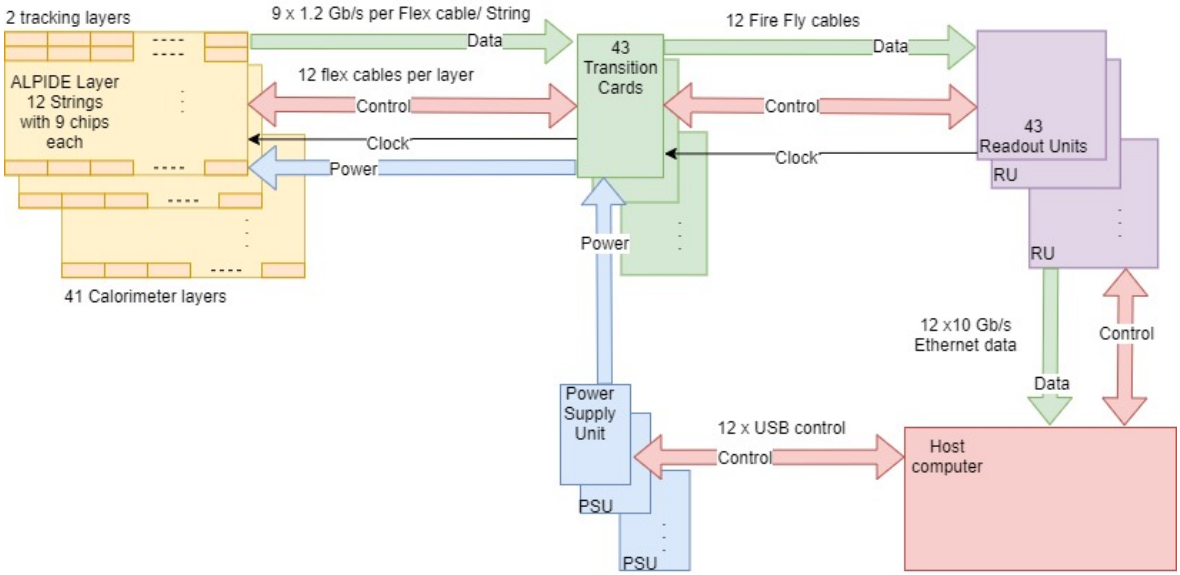


Figure 3.6: The current pCT readout and control scheme. Although one interface in this image, the control is conceptually different from the high speed readout.

version of the test setup in figure 3.5.

The RU provides a unified control interface for an entire layer handling clock synchronisation, data offload and control. The Readout Unit provides a control interface for both the individual chips and a broadcast command functionality for a simplified control of an entire layer. This simplifies control since it is no longer necessary to send general control instructions to each individual ALPIDE chip. Instruction sets only meant for individual chips, like pixel masking sets, will still have to be addressed individually.

A sketch of the whole pCT setup is shown in 3.6. With the detector stack, the transition cards, the readout units and the host computer.

3.3.1 The pRU Format

To handle and differentiate the data from the different ALPIDEs in the different sensor layers, Ola Grøtvik has defined a wrapper format encapsulating the ALPIDE data and adding additional detector specific information called the Proton Readout Unit (pRU) format [32]. This is a 128 bit data format encapsulating the ALPIDE data and adding additional information about time, Readout Unit (RU), string(This was previously called stave, hence the STAVE tag) and errors. In the format, all words follow the general format: WORD TYPE(2 bit), RU(6 bit), STAVE(4 bit), CHIPID(4 bit) and DATA(112 bit).

The format contains four words:

1. DATA WORD Contains the ALPIDE data words from the readout of a chip.
2. TAG HEADER WORD marks the beginning of a chip readout and contains information about the RU and string the chip comes from as well as busy flags and timers.
3. TAG TRAILER WORD, marks the end of a chip readout and contains error flags described in table 3.1
4. TAG EMPTY WORD is a bandwidth saving measure collecting a number of ALPIDE EMPTY FRAME words.
5. DELIMITER WORD which is a special tag marking a need to flush the offload stage of the RU.

When an ALPIDE Chip header is received, a TAG HEADER word is generated. The ALPIDE data fills consecutive DATA WORDs, ALPIDE words larger than 8 bits might be split between subsequent DATA WORDs. When a ALPIDE chip trailer is received, the remaining bits in the data word is padded with all "1". The last data word is followed by a TAG TRAILER WORD.

When receiving ALPIDE data the RU performs basic checks on the received data to check for transmission errors, buffer overflow situations and other faults as can be seen in table 3.1 as well as the BUSY_ON and BUSY_OFF flags in the pRU Header word.

The detector set up is using a server client architecture. The Readout Unit acts as a server simplifying the connection process as it will be listening for connections by a workstation. This moves all initialisation to the workstation client as the Readout Unit will be listening for connections from workstation clients once it is turned on.

To make the most effective use of the available bandwidth and reduce latency in the data transferring process it is based on the User Datagram Protocol (UDP). UDP forgoes the reliability of protocols like Transmission Control Protocol (TCP) for higher latency and throughput making it potentially unreliable. To mitigate this problem Ola Grøtvik has defined the pCT Data Transfer Protocol (pDTP). This protocol ads transmission modalities, reliability and flow control. The pDTP has different datagram headers for the server and the client as the communication is largely one way. The server will primarily send data packets and acknowledgements while the client will send commands to the server.

The format specifies 3 transmission modes that the client can requests from the server:

Table 3.1: This table shows the error flags that can be set in the pRU Trailer word [32].

pRU Error Flags	
0 Decode/Protocol Error	Asserted whenever the 8B10B Decoder has been unable to decode a byte during processing of the frame, but processing may continue. Is also asserted whenever other protocol errors are observed.
1 Frame Error	Asserted whenever a fatal error occurred during 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 comes directly after it.
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 has overflown.
6 Max Size Error	Asserted whenever the data tagger has been waiting for more than 100 consecutive clock cycles (120MHz) for valid data during the frame. Causes cancellation of the frame and forces a trailer word. The maximum wait time may be edited by setting the proper register on the RU.
7 Max Wait Time Error	Asserted whenever the data tagger has been waiting for more than 100 consecutive clock cycles (120MHz) for valid data during the frame. Causes cancellation of the frame and forces a trailer word. The maximum wait time may be edited by setting the proper register on the RU.

1. PULL mode: The client pulls a single packet of a specified size from the server if the data is available, if no data is available an error will be sent instead. Acknowledgements and re-transmission is possible.

2. SEMI-PUSH: The server sends a specific number of packets of a specified size if the data is available. Acknowledgements are sent after the last packet(End Of Stream).

3. FULL-PUSH: The server will continuously push available data to the client. The server will continue in this mode until the client sends an abort.

3.3.2 pRU Data Stream

The RU will generate the appropriate pRU elements as the ALPIDE words are received. Once an ALPIDE chip header is received a corresponding pRU Header word is generated. ALPIDE Region Headers, Data Short and and Data Long elements are used to create pRU Data words once 112 bits of data or an ALPIDE Trailer Word is received. The receipt of an ALPIDE Trailer word will also generate a pRU trailer.

By generating the pRU words as the data arrives, the output will be a semi-ordered list. The data from any single ALPIDE chip will arrive in order as specified by readout module topology [5, section 1] over a dedicated () data line [31, p.29]. So for each ALPIDE Chip: An ALPIDE Chip Header will be followed by that ALPIDEs Region Header followed by data etc. The ALPIDEs in a layer will each be read out in parallel by separate data modules, each interfacing with a single ALPIDE link [31, Section 4.2.1].

The resulting list will have the data from any one ALPIDE in order, but similarly ordered data from other chips might be in between the different data elements of that chip.

By the nature of the underlying UDP protocol, it is expected that the list will be split and a pRU Header might arrive separately from it's data and Trailer element.

3.4 Final Offload and Data Processing

Connecting to the Readout Unit via the 10 Gb/s interface is a workstation running the client software. The number of workstations needed will depend on the data volume received and the speed of the finished software, the network card and the available Linux kernel [1]. Each Readout Unit is connected to a dedicated client software responsible for processing the data stream supporting organic scaling of the detector. Adding any

number of layers to the calorimeter will increase the particle energy range the detector can handle and, if desired, a pre-patient tracker could be added for a conventional design in much the same way.

Currently a single workstation is used in conjunction with the test Readout Unit shown in figure 3.5. The final hardware for the readout software has not been selected, but it is expected that the concept will carry on, with a virtual machine running in a server environment supporting a single readout unit connected to a 10 Gb/s network interface.

Depending on the reconstruction software's requirements, it is preferred that this will share this hardware and run either in between or after data takings.

CHAPTER 4

Software

This chapter describes the two modules that have been created for the pCT readout chain as a part of this thesis. The first module is the pRU parser. The role of this module is to organize and error check the pRU data stream from the Readout Unit connected to this software instance. The second module is the Root Writer module extracting the hit information stored in the ALPIDE data and storing this in a root tree.

The software is written in C++ 17 and designed to run in a 64 bit version of the Community Enterprise Operating System (CentOS) environment. The software have been tested on a custom built workstation containing 64 GB system memory, an Intel Xenon W-2133 CPU running at 3.60GHz , an 512 GB Samsung MZVLB512HAJQ-000H2 SSD and a Nividea Quadro P400 graphics card running on a HP 81C5 motherboard.

The work undertaken here is based upon the work done by Håkon Andreas Underdal and Annar Eivindplass Hilde taking the software they developed and improving it by using pure binaries and adapting it to a new control structure. The old readout software was written in C++ 11 and provided a proof of concept of reading data from the ALPIDEs with the pCT setup. Most of the code have had to have been rewritten as the previous concept used ASCII binary numbers, but much of the structure still remains.

4.1 Software Goals and Functionality

The focus of the pct-online software is the timely, efficient and correct readout and processing of the ALPIDE data stream from the Readout Units. For proton based

CT to be viable, the particle information captured by the detector must be offloaded in a timely manner to avoid data loss by buffer overflow. To reduce the hardware requirements of the final system, it is desirable that the modules have a low resource footprint.

The software must be conducive to change and be easily adaptable to changes in requirements. An example is that hardware requirements, such as the physical detector layout, is still subject to change. Likewise, data elements and their significance is also subject to change as an error might be moved fro a non fatal event to a fatal event requiring the frame in question to be dropped.

The software should be capable of both offline analysis and online analysis of a data stream. By splitting the concerns, the software developed does not distinguish between data sources. The major challenge is instead the nature of the input. An offline process will read the data in it's entirety from disk and provide this as a single data package for processing. In contrast, online processing will receive smaller data packages when they become available. Online processing thus requires that status information about the current processing is not lost when a data package has been processed.

For testing purposes all data must be collected for further analysis and error checking of the system.

The software has been divided into modules containing the overall functionality. The input defining the data extraction is not a part of this thesis so data is assumed to be available. The focus her is the processing and output of the data.

4.1.1 pRU Processing Specifications

A summary of the specifications described in chapter chapter3.3: Since a pRU word is 128 bits, any incoming data package must be modulo 128 to ensure it is possible to separate it in to individual pRU words.

The pRU words specified in chapter 3.3 will be collected and the corresponding pRU Header word, pRU Data words and pRU Trailer word will be put in to a pRU Frame. Each Frame will contain a single pRU Header word, one or more pRU Data words and a single pRU Trailer word with matching RU, STAVE and Chip ID. If any of these elements are missing, the Frame is rejected.

BUSY ON and BUSY OFF flags in the pRU Header are stored for further processing while all of the errors in the pRU Trailer are currently treated as fatal and the Frame is rejected.

The FRAME Size must match the received size of ALPIDE words, a mismatch is

a protocol error or loss of a number of pRU Data words and is considered fatal. The frame would then be rejected.

4.1.2 ALPIDE Processing Specifications

A summary of the specifications described in chapter 2.2:

The Readout Unit will by default filter so all ALPIDE COMMA, IDLE; CHIP EMPTY FRAME, BUSY ON and BUSY OFF are checked for in the PRU Data Words.

Any ALPIDE data stream must start with a ALPIDE Chip Header followed by an ALPIDE Region Header.

Any ALPIDE Region Header must be followed by an ALPIDE Data Word.

Only the BUSY VIOLATION and FLUSHED INCOMPLETE error flags in the ALPIDE Trailer is Fatal.

4.2 System Overview

The basis for the readout software, is a control module developed by Mathias Richter. The control module uses lambda expressions to define easily replaceable template modules called policies. The control module define the system workflow and is split in to 3 different policies encapsulating the input, the data processing and the output of the readout chain. The control module accepts the policies to be used during it's creation and only functions as a hosts for the policies and dictates the data flow.

Each policy is thereby a completely separate entity that neither the control module or the other other policies knows what it is or what it does. Only the compiler enforces that the input and return types matches. The type requirements are as follows: The Input Policy must take no argument and return something of type T. The Processing policy must take an argument of type T and return something of type B. The Output Policy must take an argument of type B and not return anything. This loose coupling leaves to the programmer to ensure that the different modules truly match.

From this, an executable is created defining the policies to be used in an executable as shown in 4.1. The compiler will only accept modules with matching function calls but the developer must ensure they provide output the following module can process. The template base of the system provides both flexibility and robustness by decoupling the program control structure from the system function and by decoupling the individual policies/ software modules.

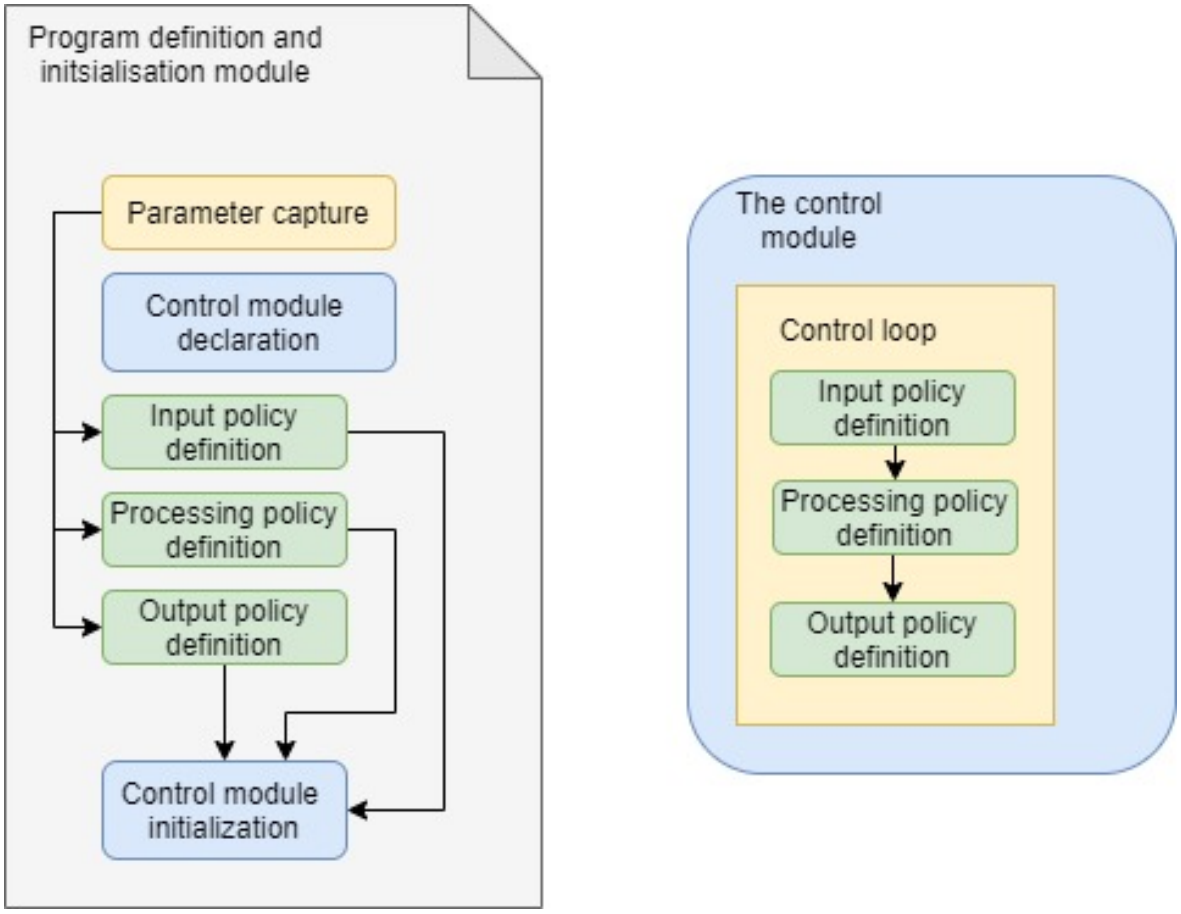


Figure 4.1: General overview of how an executable is defined using policies and the control module.

To ease development, the software is primarily divided by discrete functionality and rely primarily on binaries of common interfaces like the ALPIDE data, pDTP and pRU definitions for communication between the different modules. This allows the various pieces of software to be completely disjoint relying on the common interfaces instead. This also facilitates conversion to and from text representations of the binaries or hexadecimal conversions, for ease of human inspection or data generation by python script.

The control module does not know anything about the policies but the C++ type system will require that the input-output pairs have corresponding types at compile time. By making the control class a template class, the specific return value types will be decided at compile time. The class constructor then utilises initialisation list, binding the policies (lambda expressions) received to create static reference binding at run time, making the reference a class member.

4.2.1 Data Processing

The processing/filter policy requires a software module that can take a incoming pRU data stream and prepare it for further processing. The module accepts the binary pRU Data in the form of $vector < char >$. The data stream might be separated in to multiple vectors, each containing one or more of the 128 bit pRU Words. The data stream can come from any source as long as the pRU Stream requirements specified in chapter 3.3 are met.

The parsing of an pRU stream will generate a number of Frames. Each valid Frame consists of a number of 128 bit pRU words. It will always start with a pRU Header Word followed by one or more pRU Data Words and ending with a pRU Trailer Word as defined in section 3.3. The data stream can be expected to be semi ordered where each pRU word will keep it's relative position in regards to the rest of it's own frame but with elements from other frames mixed in.

A pRU Stream could look like the following: header 1, data 1.1, header 2, data 1.2, data 2.1 trailer 2, data 1.3, trailer 1.

There is fundamentally two approaches. The first approach is to extract the data in to new elements while traversing the stream, performing the extraction and error checking at each element. With this approach, it is only necessary to traverse the list once at the cost of more processing at each step. Because of the unsorted nature if the data stream, the potential for data races and access conflicts it is much harder to implement parallel processing with this approach. The relatively small size of the objects compared to the pointer (128 bit data word vs 64 bit pointer) the reduction in data volume with a list of pointers would only be 50% compared to a complete deep copy. for the added complexity at relatively meager gains, this approach has not been

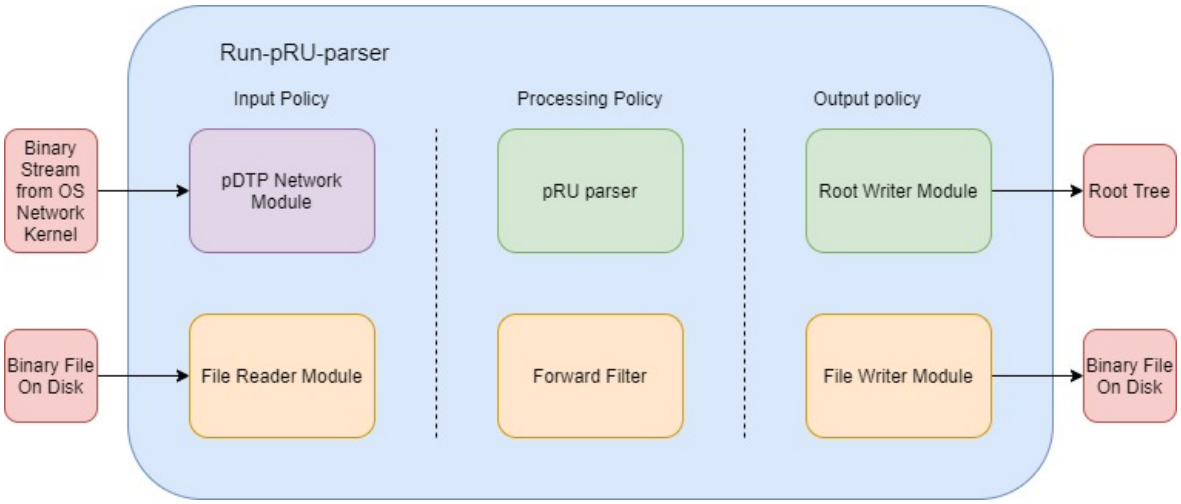


Figure 4.2: General overview of the current software modules. The module in green is covered by this thesis.

considered further.

The approach used is to sort the data stream (extract the pRU frames), check each pRU frame for errors and lastly extract the data in to a new element. This will remove any faulty data before the data extraction and the binary stream can easily be stored as is or the stream of faulty frames can be separated and stored individually for later processing or inspection. The disadvantage of this method is that it can potentially use a a lot of memory, short of sorting the the data structure directly, most approaches would increase the data volume by 50% or more. For instance a list of pointers to the pRU words of a frame would give about a 50% increase in data volume since each pRU word is only 128 bits and all relevant architectures operates with 64 bit pointers [19, The Linux 64-bit architecture]. The advantage is that each step is discrete and conducive to modular and parallel processes and it is a just in time approach only doing the necessary processing on an element before moving on so that data extraction is not attempted on faulty frames.

4.3 The Control Module

The control module contains a set of separate sub processes called policies. The control module accepts the policies to be used during it's creation and has only function as a hosts for the policies while it controls the data workflow. Each policy is thereby a completely separate entity that neither the control module or the other other policies know what it is or what it does. Only the compiler enforces that the input and return types matches. This loose coupling leaves to the programmer to ensure that the different modules truly match.

The control module requires 3 policies defined at the time of creation, defining the behaviour of that control instance. The first policy is the input policy. The role of this policy is to acquire data from some source, it takes no argument and returns the data it has gathered. The second policy is the processing policy. This policy takes the output from the input policy, processes it and returns the result. Lastly, there is the output policy, its takes some input and does not returning anything.

The control module is set up to run each policy in turn until it receives an abort signal. Depending on the module calling the control module this might be anything from a fixed loop to a command input.

The project has currently developed 6 modules for the initial testing of the prototype hardware. The 2 input policies, pDTP parser for network interface and RU data offload and the file reader policy for basic file reading. The 2 processing policies the forward filter passing any input directly to output and the pRU parser sorting and checking the pRU data stream from the RU and pDTP parser or a file. Lastly the 2 output policies uses the $std :: ifwriter$ for writing a binary pRU stream to disk and the root writer for writing the ALPIDE pixel hit coordinates stored in the pRU stream to a CERN root tree for statistical analysis and visualization using the root library.

The pDTP Network Module is a separate project and will not be discussed here. the File Reader and File Writer modules are simple stream reader/writer and the forward filter will just forward any input.

All the modules currently interface with a $vector < char >$ containing a binary pRU stream. All modules currently developed can easily be switched and still interface with the other policies. They are not entirely compatible as the Root Writer Module requires its input to have been parsed first. So the network module followed by the forward filter is incompatible with the Root Writer Module. A binary file can be read and fed through the forward filer to the Root Writer Module provided it has been run through the parser first.

4.3.1 Sorting and Error Checking

The pRU parser processing module has been developed by conceptually, treating the sorting of the pRU stream in to pRU Frames and the error checking of each complete Frame as separate processes. This approach was chosen based on a desire for a modular and parallelizable software architecture and the possibility for developing modules with discrete functionality.

The pRU parser module itself is still a monolithic module as the error checking lacks a separate control structure due to the legacy code structure. This is shown in 4.3 as the purple paths. A separate control structure analysing the finished Frames

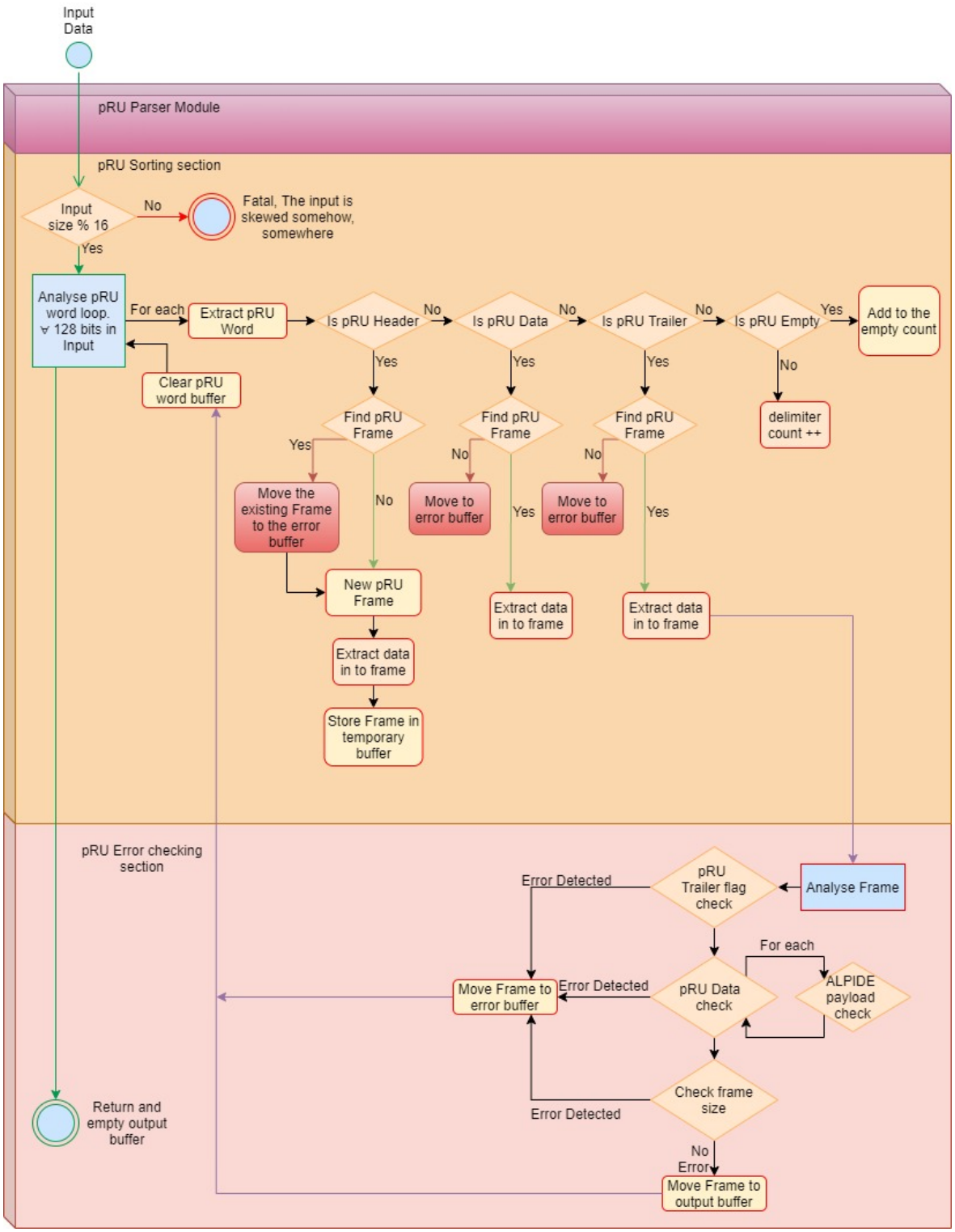


Figure 4.3: A detailed overview of the decision process in the pRU Parser.

in the temporary buffer or a distinct buffer is necessary to completely separate the functionality. This process has been postponed to be implemented with the inclusion of threads.

The general approach to the processing module is to have 2 discrete processes designed to operate as separate sub modules. The goal of this approach is:
- To facilitate parallel processing in the software.
- To have shorter development cycles with useful and working software.
- Simplify testing by separating concerns.

The first sub module is the sorting module. The purpose of this sub module is to organises the data stream in to pRU Frames and reject any ill formed Frames as described in section 4.1. As can be seen in figure 4.3, the parser will check the size of the input data to ensure it can be broken up in to individual 128 bit pRU Words. Then it will enter the main control structure analysing each 128 bit pRU Word. Any pRU Words and partial Frames rejected will be put in to a dedicated error buffer that can be extracted or written to disk for further analysis.

The second sub module checks the pRU Frames content for errors, if any errors are found, the frame will be rejected and put in to the dedicated error buffer. This module will first check the pRU Trailer word for any error flags and reject the frame if any flags are set. It will then go through the ALPIDE Data in the each pRU Data Word and check that it is well formed, as described in chapter 2.2.1 and 3.3, and reject the frame if it is not. It will also count the number of ALPIDE words and their total size. Any error flags in the ALPIDE Trailer Word is checked and if they violate the constraints in chapter 2.2 the frame is rejected. The padding in the last pRU Data Word is checked and the frame is rejected if there is something else than padding after the ALPIDE Chip Trailer. Finally the count of the size of ALPIDE Data received is compared to the registered sum in the pRU Trailer Word.

This will result in a sorted list of binary pRU Frames containing the accepted frames being returned to be accepted by the output policy. Additionally the error buffer can be extracted and stored separately. Unfortunately, this file has to be inspected manually. A module to convert this buffer to a more flexible format will hopeful be developed in the future.

4.3.2 Data Extraction

For further processing like statistical analysis, plotting and reconstruction, it is necessary to extract the particle hit information from each ALPIDE chip. Each ALPIDE is treated as a Cartesian plane of 1024 by 512 pixels (x,y) with the readout periphery oriented in a strip along the boom of the x axis.

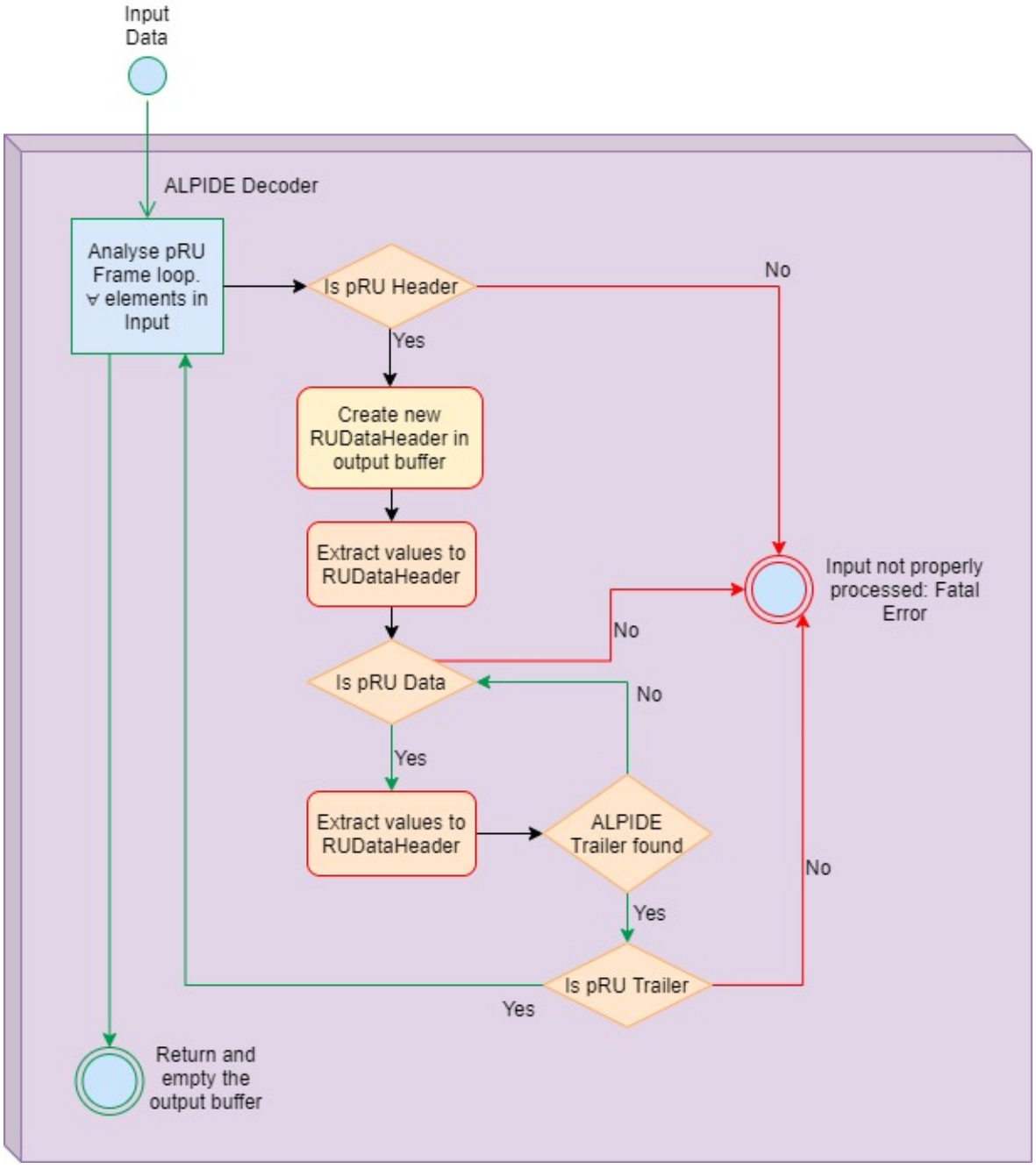


Figure 4.4: A detailed overview of the decision process in the ALPIDE Decoder.

The extraction of hit information is performed within the 1024x512 hit matrix of each individual ALPIDE chip. The ALPIDE chip number and Readout Unit number is stored for downstream software to determine that ALPIDE chip's place in the detector stack. This avoids the need for the current software to address the physical detector layout.

For software flexibility, ease of development and to respond to future change in the hardware, since the detector layout has not been completely finalised, the hit information is extracted within the 1024x512 hit matrix of each individual ALPIDE chip. The ALPIDE chip number and Readout Unit number is stored with the hit information for downstream software to determine that ALPIDE chip's place in the detector stack. This avoids the need for the current software to address the physical detector layout and changes in the detector stack layout are deferred to the reconstruction software.

This separation of concerns enables analysis and plotting of the data received from the current test setups and should be feasible up til a complete layer of 108 chips. Giving the reconstruction software the responsibility of knowing the physical detector layout incurs a very small cost since it is likely this module will have to contain this information regardless. An additional benefit is that is simplifies the use of the software for different detector stacks. Most notably, the current planed prototype, will at least be smaller and also possibly different in other ways than a clinical prototype facilitating adaptation of the current software for future sensors.

The ALPIDE Decoder functions by looping through every pRU Frame. At the pRU Header position, it will create a new RUDataHeader and extract the Readout Unit number, the string ID, the chip ID, the Frame ID, and the Readout Unit clock. It will then go through every pRU Data Word in the Frame saving the ALPIDE Chip Header and the current ALPIDE Region Header and decode each Data Short or Data Long in to their respective Cartesian hit coordinates and storing them in a hit list until the ALPIDE trailer is found. When a pRU Trailer is found as the next pRU Word the next Frame is decoded until the input buffer is empty and the list of RUDataHeaders are returned.

The buffers persist in the pRU parser module enabling the module to receive data in separate inputs. If a part of a readout is split by a network frame size limit or any other reason, as long as the other stream requirements previously discussed are not broken, this will not affect the parser. The error buffer will also continuously accumulate any rejected data.

For use with reconstruction and for ease of analysis, some separate data types are being created. The RUDataHeader and AlpideHit data types have currently undergone 2 iterations and have seen significant change. The RUDataHeader represents a single readout (fixed in time) from a single ALPIDE (fixed in space) containing a number of (x,y) hits represented by the AlpideHit type. The requirements for the type are chiefly

that they must contain the relevant extracted data in "regular" variables to cater to the need of both statistical analysis and performance checking of the prototype hardware and future reconstruction algorithms. The type should be serializeable for writing to disk or network transfer if the reconstruction takes place on separate hardware. The type must be space efficient. A lot of data is expected so careful use of types is necessary in order not use more space than necessary. Additionally, since some of the data field in the pRU format are smaller than 8 bits, there is an option of using unions of combining smaller variables in a larger type. For instance storing two 4 bit variables can be stored in an 8 bit type.

```
struct UnionExample \{
  union \{
    uint8_t container;
    struct \{
      uint8_t 4BitVariableA : 4
      uint8_t 4BitVariableB : 4
    };
  };
};
```

Currently, the RUDataHeader format is using a non-serializable vector for the storage of ALPIDE hits for ease of implementation and testing. This have to be converted back to the serializable C style array.

Data Extraction In The Readout Chain

There are valid arguments to be made to have the data extraction as part of the processing module. The data extraction can be seen as integral to the process of preparing the data stream for further use. The RUDataHeader format would have to be serialised before it is returned to be compatible with the existing modules. This would, however, create a conflict where seemingly similar files would not be in compatible format as any attempts at re parsing a file containing the RUDataHeader format by the parser would fail. The other option would be to introduce a module that can deduct the formatting of an incoming stream greatly increasing complexity or force a different naming scheme on the output. It is therefore more practical, by following the existing modules, to set the extraction of hit coordinates in to the RUDataHeader format, as a sub module in an output policy or, if the control structure is change as a separate second processing module.

The primary use for the data extraction module is to support analysis, graphical plotting and reconstruction. By making the data hit coordinate extraction software a sub module accepting the sorted pRU Frame binaries from the standard output policy ensures compatibility, allows the binary files generated by the standard output policy

to be valid input for the parser and can still be used as a sub module in a future reconstruction policy.

ALPIDE Hit Position Calculations

As described in chapter 2.2, the ALPIDE pixel numeration is is based on a 1024 pixel address space of tow 512 pixels in 16 double columns in to 32 readout regions. The hit position calculation code has been taken from the CERN and adapted and improved to meet the requirements of its user module.

The code treats every ALPIDE Data Word as a Data Long, a data short simply having a cluster of 0. The 8 bit hit region is masked put in to an int, the encoder ID is extracted and the encoder ID and 8 least significant bits of the address is extracted. At this point, every Data Word is treated as a short so for the original hit and every hit in the cluster hit map (data short is treated as having a 0 cluster), the pixel address is the extracted encoder ID and 8 least significant bits incremented by the cluster position. The row of this pixel is half the pixel address while the column is calculated by the formula:

$$(region * 32 + encoder\ ID * 2) + (0\ or\ 1) \text{ for left or right column}$$

The column is the left one if

$$\text{address } mod\ 4 = 0\ or\ \text{address } mod\ 4 = 3$$

Se pixel pattern in figure 2.5.

CERN Root Tree Module

Root is a data analysis toolkit developed at CERN [39].The framework is used by physicists and provides tools and functionalities to perform big data processing, statistical analysis, visualisation and storage. It was a desire from the user group to have the readout stored in a format processable by root. The desire to use the root framework greatly affected the ALPIDE Decoder format in order to get the data in to analysable root Trees.

The root framework is primarily a scripting tool working very well when performing tasks inside the framework. It has its own c++ interpreter to run c++ code directly, easily serializing custom c++ classes in to root files. On the other hand, using this functionality outside root, is not particularly easy as the serialisation uses some ”behind the scenes” functionality in root.

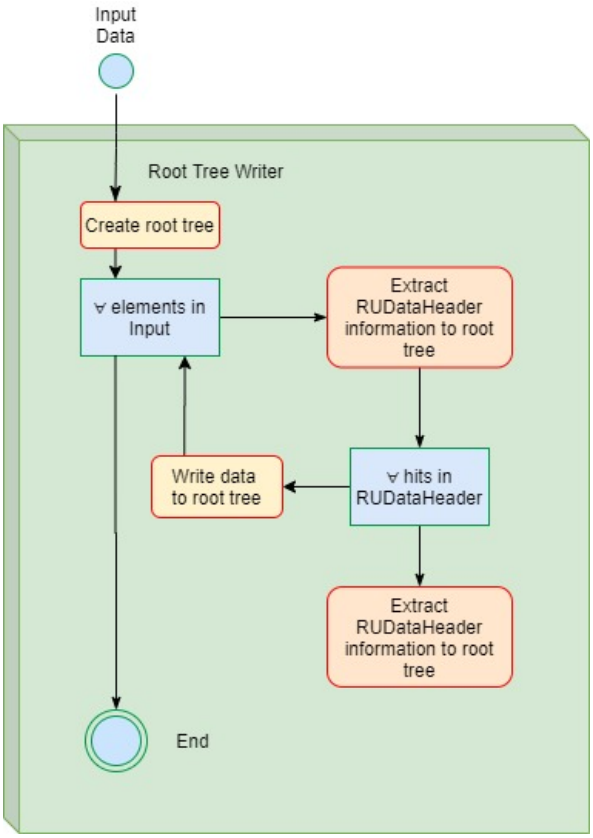


Figure 4.5: A detailed overview of the decision process in the Root Writer module.

The root framework has a number of file options but on of the more common formats is the root TTree used for this project. A TTree represents a columnar data set and is filled by fundamental types. The following elements are defined:

$$ReadoutUnitID$$
$$FrameID$$
$$StaveID$$
$$ChipID$$
$$ReadoutUnitClock$$
$$SpillID$$
$$BunchCounter(ALPIDEClock)$$
$$Column$$
$$Row$$

The root tree is created by taking each RUDataHeader element and filling in all but the column and row fields, then looping through the ALPIDE hit list and writing each row column element. The Frame is then added to the TTree by calling:

```
// Write frame info to tree
tTree->Fill();
// Resett frame
frame.column={};
frame.row={};
```

This ads the Frame as a new element to the TTree. Once all Frames have been processed the control loop exits.

As the pRU Parser module, the root tree module is reset on initialisation and will over writer any existing trees with the same name. In a production environment, date and time might be added to the output file to guarantee uniqueness and prevent accidental data loss leaving the user to remove undesired files manually.

CHAPTER 5

Analysis and Assessment

5.1 General Overview

The modular nature of the software has lead to the testing focus being on the individual modules, with incremental improvements of the test suite, as new functionality has been added or bugs found and fixes provided. During development a Continuous Integration (CI) environment has been enabled for the repository. The CI is configured to build the project and run the tests it can find. The CI system therefore only provides the build and execution environment while it is up to each developer to provide test cases for new functionality or bug fixes.

The repository is set up so that each area of concern in the repository has its own separate testing section for testing of the different elements.

The primary focus of these tests are on the correctness of their accompanying module. Test on speed and throughput are performed separately and manually. The reason for this is that the required data sets needed are far to large to store in the repository and the tests are most easily performed by inserting benchmarking code around the code bloc under testing.

Testing of memory issues is similarly done by using the Valgrind profiling suite.

5.2 Future Development Consideration

As the pCT project is active, significant development work still remains. The software is therefor required to consider not only the current needs of the project but also future

needs.

The designed pipeline is starting to become well developed for the initial readout. The RU firmware is currently being used to readout the ALPIDE test strings and is functioning well. Updates are being delivered to the firmware and the design of the production hardware has begun.

For the software, a network module is being used to offload data from the RU and along with the control module, the pRU Parser and the Root Writer, it provides a complete pipeline. The software was originally based on the simple file reader module still in use as the input policy, an largely outdated forwarding filter just returning the input as the processing policy and the simple file writer still in use as the output policy. The pDTP Client module has been added providing the ability to connect and receive the data stream from the RU. The pRU Parser has been added providing sorting and error checking of the data stream from the pDTP module and finally a root tree writer module has been added extracting the pixel hit information and storing it in the root tree format for analysis and visualization.

This demonstrates that the system architecture provides ample opportunity to easily add and change the functionality of the system just by switching or developing new modules to expand the systems functionality when the necessity arises.

The most significant consideration is the architecture of the final server setup and the reconstruction software. It is not certain how or where reconstruction will happen and how it will access the processed data. A separate serializable format has been developed and is used internally in the root writer module by the ALPIDE decoder. If it should be necessary, the ALPIDE Decoder can be coupled to a network module for of site data processing.

Lastly, by splitting the modules into smaller sub modules each with a distinct task, like the ALPIDE Decoder extracting the pixel hit information in to the PRUDataHeader format, it may be possible to re use this element or easily replace or modify it for more efficiency. This also applies to the pRU Parser, as it has been separated in to a sorting section and an error checking section.

This is demonstrated by the 3 executables made for testing. These executables are made by changing the modules to suit the desired purpose. These executables are primarily used for testing the more complicated Network module. Further modifications are made by changing the Processing Policy and Output Policy modules between compilations to alter the behaviour when required.

5.3 Theoretical Requirements

The expected data rates from a Readout Unit will vary with beam intensity, the layer position in the sensor stack and the length of the data taking window (strobe). The front layer will generate the most data and has simulated outputs of about 650 Mb/s, 700 Mb/s, 800 Mb/s, 950 Mb/s and 1 400 Mb/s for 30 microseconds, 20 microseconds, 15 microseconds, 10 microseconds and 5 microseconds strobe respectively. A minimal gap of 25 nanoseconds is used between each strobe [30, p.1-2] see figure 5.1.

If the parser keeps pace with or outpaces the data stream, the processing will run parallel to the readout and finish at roughly the same time as the data stream providing instantaneous feedback. This is the, desired solution, however, it is not given that the parser will reach these speeds so the question that needs to be answered is: "how slow can we go?".

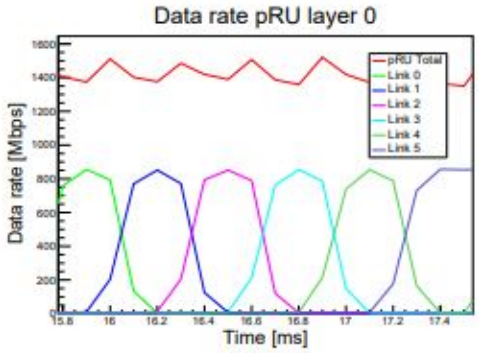
Assuming a readout measured in seconds and the processing runs concurrent with the data readout, the data processing time should not continue for any additional period than the time it took to do the readout. In order to prevent the user form feeling that the software is slow: a 2 second readout should therefore, as a goal, not continue for more than 2 seconds after the readout has finished.

It is not quite known how a data take will be performed but as long as the beam time is measured in seconds, before realigning, either the sensor or patient for the next slice, this provides a time gap for the data processor to finish without slowing down the overall process. It one take is all that is necessary, the processor should finish without creating to much of a wait for the user.

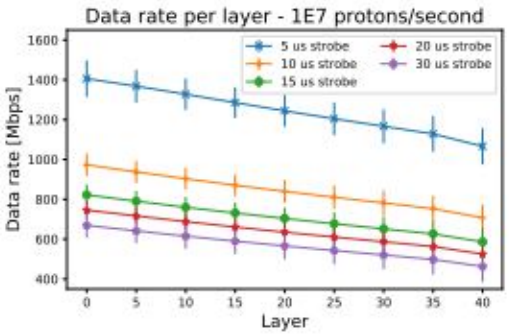
When running software instances in parallel, the total speed is dictated by the layer producing the most data, since this will take the most time to process. Keeping the parse time below double the data acquisition time requires the parser to to have a processing speed no less than half the speed of the data readout for that layer. From a data rate somewhere in the 650 – 1400 Mb/s range, it is desirable to reach a processing speed in the 325 - 700 Mb/s range or greater. These figures includes the time it takes to write the data to disk.

5.4 Processing Speed

The focus of this thesis is processing of the pRU data so speed testing is performed with a test file and the file reader module. The processing speed is measured by inserting benchmarking code around the code block under test and running the software. The primary metric is throughput measured in megabits per seconds(Mb/s) average



(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 5.1: Figure from [30, Fig 1] showing simulated data rates from a 230 MeV proton scanning beam with a beam intensity of 1E7 protons/second. The scans takes 65 ms.

processing speed. The use of bits in stead of bytes is because it is desirable to have a metric that directly comparable to the network.

Since the file reader is used, there is only one, compete input and the size of the input forms the base for the calculations. The tests uses the std::chrono library and it's high resolution clock and time point utility functions. The code used for all the tests is the same to ensure they are comparable.

Code block for the test setup:

```
// ---------- Define benchmarking tools ---------- //
std::chrono::time_point< std::chrono::high_resolution_clock >
    startTimePoint;
std::chrono::time_point< std::chrono::high_resolution_clock >
    endTimePoint;
```

Code block for the test:

```
// ---------- Start benchmarking ---------- //
startTimePoint = std::chrono::high_resolution_clock::now();
.....
Code under test
......
// ---------- End benchmarking ---------- //
endTimePoint = std::chrono::high_resolution_clock::now();
```

Code block for the test result:

```
%// ---------- print result ---------- //
auto start = std::chrono::time_point_cast<std::chrono::microseconds>
    (startTimePoint).time_since_epoch().count();
auto end =std::chrono::time_point_cast<std::chrono::microseconds>
    (endTimePoint).time_since_epoch().count();

auto duration = end - start;  // milliseconds

// Should give number of bits(but is system dependent)
long double Megabits =
        (data.size() * 8)/(long double)1'000'000;
long double seconds = duration/(long double)1000'000;

std::cout<<"This parse had a speed of: "<<
Megabits/seconds<<"Mb/s"<<std::endl;
```

The test setup and code is tailored for single input runs, ie. input from the file reader. To expand the testing to multiple input, it is possible to just run everything multiple times to simulate the input. This would increase the control overhead and perhaps give a more realistic picture of the processing speed but these tests have not been performed.

During each of the four tests for the overall system, the pRU parser and root writer modules and the ALPIDE Decoder pRU Exporter (root tree writer) submodule and the same file was used for each run. Since no data take of sufficient size with real detector data exists, The data used could either be a composite file made from a data take or a generated test file.

The test file used was a file from a data taking session performed 10.01.2020. The file contains real word data in an idle environment, no beam or radiation source present. The file therefore contains relatively small Frames as there is little radiation present to trigger the pixels, giving few hits per readout cycle/Frame. This may affect performance of the tested modules.

The data is then copied 1000 times to create a 3.14 GB test file used in all speed test.

5.4.1 pRU Parser

The pRU Parser module is currently a monolithic module instead of discrete sorting and error checking sub modules. The code will sort the incoming data and as soon

as a pRU Frame is completed it will will proceed to perform error checking on that frame before the sorting is continued. It is therefore not practically possible to test the sorting and error checking independently as the testing on one would impact the performance of the other. The parser has been tested as a whole by setting the initial time point as the first element in the pRU Parser parse method and the end time point right before the return statement. The test code is contained in the class under test and output is written to the terminal, the code inserted between the end time point and the return statement when it goes out of scope. The software is compiled on the host machine and the executable is run with the input file and the results are manually registered and processed.

5.4.2 Root Writer

The Root Writer module consists of two completely discrete modules designed to be run in parallel. Since it is not a monolithic unit like the pRU Parser, it is easy to test each sub module independently. To test the whole root writer module, it it necessary to place the testing code around the calls to the modules in the run-pru-parsers output policy. The output is placed after the code executes and before it goes out of scope by the end of the lambda expression. Since this test is performed in the entity creator, it is not necessary to perform the output right away however, it is good practice to keep related elements together and have the same layout and output for all test cases. The output for the test of the root writer and it's sub modules is the same as the output for the pRU Parser. The output is written to the terminal then registered and processed manually.

For the ALPIDE Decoder, the test is performed similarly to the pRU Parser with the initial time point as the first element in the decoder and the end time point just before the return statement. The scope is moved to the sub module and everything is set up as the pRU Parser with the output between the end time point.

The Root Writer module was similarly tested the initial time point as the first element and the end time point with the output as the final elements of the method scope.

in the pRU Parser parse method, the end time point right before the return statement and the output is inserted between the end time point and the return statement. The software is compiled on the host machine and the executable is run with the input file and the result is manually registered and processed.

Table 5.1: This table shows a summary of the testing results. For full details, see apendix A. *The data size is unknown but it should be comparable to the ALPIDE Decoder

Module	Data size [Mb]	Mean time [s]	Mean speed [Mb/s]	std dev speed [Mb/s]
Offline to root tree	25 169.92	86.564	291	3
Offline to binary	25 169.92	26.092	965	8
Online to root tree	25 169.92	85.561	294	3
Online to binary	25 169.92	25.395	991	6
pRU Parser,	25 169.92	24.593	1023	6
Root Writer Module	17 979.136	61.253	294	4
ALPIDE Decoder	17 979.136	12.721	1413	4
pRU Exporter*	17 979.136	46.633	386	5

5.4.3 Results

The data size used for the speed calculation is primarily the data read from disk. This is 3 146 240 kilobytes or 25 169,92 megabit. The exception is the Root Writer module which uses the size of the output from the pRU Parse module. Also, the input for the pRU Exporter sub-module is incorrect since the ALPIDE Decoder transform the binary input in to the RUDataHeader format. This format conceals the size of the input. since no data is removed, and there is a little overhead in both formats, the input size to the ALPIDE Decoder is used as a reasonable substitute.

Notes: For offline processing, pRU Parser and Root Writer module, the time is a calculated from the speed and data size. Most tests had 20 runs with the exception of Offline processing to binary and Online processing to binary. The

5.5 Resource Use

The primary tool used throughout development is the Memcheck tool in Valgrind. Memcheck performs run time analysis of the compiled code. This way, Memcheck can cover the entire code, even dynamically linked libraries and There is no need to insert test code in the software being tested. Memcheck will only analyse the code that is actually run and it will not be able to catch faults in execution paths that is not used [21, section 2.9 and 2.10].

Memcheck dynamically tests the addressability of every byte of memory detecting all accesses to unaddressable memory. Memcheck tracks all heap blocks allocated using malloc() and new. This enables Memcheck to detect bad or repeated frees of

heap blocks, and leaks at program termination. Memcheck looks for overlapping blocs of memory supplied as arguments to functions. Most importantly it tracks every bit of data in registers and memory enduring that they are well defined. With this, Memcheck has bit level precision in tracking undefined value errors [21, Section 1.1].

During the processing speed measurements, the memory usage was also monitored. The process peaks at 10.9 GB memory during the root tree creation process. When the parsing is preformed with the binary file writer as the output policy, the memory consumption peaks at 5.9 GB at the very end of the file writing process.

5.6 Correctness

Testing have for the most part been performed alongside the development of the module under test. The constraints, requirements and existing code base for the different modules have resulted in completely different testing strategies for the two module. The general goal was to have a comprehensive, automated test suite to aid and guide development and catch errors early.

The pRU Parser module have followed this strategy and has an automated test suite developed along the software. The test suite have BEEN developed using the BOOST::TEST framework, creating an automated test suite run at compile time.

The Root Writer module on the other hand have applied the diametrically opposed strategy of manual testing for the software development.

5.6.1 pRU Parser

The pRU Parser test suite is based on the Boost framework's Test library. The tests have been developed along side the software and is based on simple hard coded sets of pRU words, one set for the input, and one set for the correct output. Automated generation of test data is the most desirable as it is easy to test the whole range of possible combinations.

Making a software suite for automatic generation of test data, is time consuming and testing suite would need some testing. In stead of having a comprehensive test suite the pRU parser uses a just in time strategy of incrementally adding smaller blocks of hard coded data when a new function is added or a bug discovered. The narrow scope ensures simplicity and and by having hard coded values, the values are easily tailored to narrowly test a single or limited number of features and include the points most likely to fail. The reason this strategy was chosen was the initial code base. When adapting the old legacy parser to the new system, it was necessary to have a simple

test suite available immediately. Once this system is in place, and the initial tests cleared, this setup provides simple and accurate incremental additions to the test suite complementing the features of the software developments.

Each test is completely independent and automated by the Boost framework. The framework provides automated response and is recognised by the CI environment so all tests are run during integration.

The first tests is focused on the sorter, providing input with missing or miss aligned pRU words. These test are created by duplicating, moving or removing elements of the original test for correctness. Further tests are mead by making Small changes some pRU words. A good example is size error input to check if all pRU Data words have been received, only changes the last two bits of the pRU Trailer word as shown in the example below.

Example of a test created by flipping two bits:

```
// Original and correct pRU trailer
0b10000010, 0b00010011, 0b00000000, .., 0b00000001, .., 0b00100100
// Modified incorrect pRU trailer
0b10000010, 0b00010011, 0b00000000, .., 0b00000001, .., 0b00100111
```

Each test consists of a named Boost test case denoting what is being tested and terminal output with test number and feature being tested. Each test initialises its own pRU Parser module which goes out of scope at the tests end. The module is then feed the test input and the output is compared with the expected result using Boost check.

5.6.2 Root Writer

The Root Writer module have used a diametrically opposite strategy of purely manual testing. The choice of manual testing was based on several factors.

Firstly, unlike the pRU parser, it was not as easy to test the module until the software was relatively complete and when it was complete, the root formatted output was harder to test automatically but was easily human readable. There exists code for reading .root files and the root framework provides some tools to create files with serialized c++ objects to make the files readable by a test suite or code for reading a root file directly. However the primary blockers faced in development were memory related and functional resulting in insufficient time to develop an automated test suite during development.

Secondly generation of test data was not as easy as for the pRU parser as each

ALPIDE data element now had to contain data. Additionally, the way the ALPIDE chips reads out the pixel hits and distributes the information over a number of data fields make creation of test data more time consuming as it is necessary to know the hit coordinates of the ALPIDE input to verify the output.

Lastly, the existing pRU Frame generator in WP3 could be used. This generates pRU frames with an accompanying hit list so the input and output can be compared. The test scripts in WP3 unfortunately generated data sets in incompatible hexadecimal text files. The labour of converting these files and checking the result is less labour intensive and overall faster than creating a test suite considering the additional technical challenges that would have to be solved.

For these reasons, the testing of the Root Writer have been performed manually using the data from the wp3 software simulation package and web based hex to binary converters.

The reliance on these web based tools might seem questionable at first glance, however, at least 3 different tools were used, and for all 3 to have the exact same fault, is highly unlikely. Likewise; the chance that an error in the hexadecimal conversion would exactly mask a fault in the ALPIDE data extractor and the Root Writer is not likely either. It is therefore reasonable to accept this test method as sufficient.

CHAPTER 6

Discussion and Future Work

The software modules being described in this thesis is now being integrated in to the project development to replace the prof of concept software. Some features have still not been implemented but it is not the less preforming well.

6.1 Performance

6.1.1 Processing Speed

The testing done in chapter 5 and using the 3.14 GB file, shows the feasibility of using the current software to analyse test data. The top layer is expected to generate the most data and simulations shows it will generate about 1 400 Mb/s. The processing of the data generated buy a 1 second beam exposure (1 400 Mb) would be 4,8 seconds while the processing of a 10 GB file would take 4,6 minutes at the current pares speed mean of 294 Mb/s. This compares quite favorably to the 1,6 Mb/s parse speed of the proof of concept software but does not quite reach the desired minimum parse speeds set forth in the requirements.

However a switch to the binary output processor increases the parse speed mean to 991 MB/s well in excess of the 700 Mb/s desired but still less than the goal of 1 400 Mb/s. This setup would process 1 400 Mb of data in 1,4 seconds and 10 GB in 1 minute and 21 seconds.

The most notable result is the comparison between the 1 023 Mb/s mean processing speed speed of the pRU Parser module and the 293 Mb/s mean processing speed of the Root Writer module. This clearly shows that the Root Writer module is the current

bottleneck as the mean processing speed of 293 Mb/s is within the margin of error for "online" processing, when the time it takes to read from disk is excluded.

Of additional interest is the comparison of the Alpide Decoder submodule and the pRU Exporter sub module. The relatively simple pRU Exporter submodule is taking significantly longer than the ALPIDE Decoder submodule which reaches the desired processing speed of 1 400 Mb/s.

6.1.2 Memory Usage

The software currently has a memory consumption of 3,5 times the data input when using the rot writer module. Going by the data rates in figure 5.1, the highest data rate is provided by the first layer and the beam stops at layer 40. At this point the data rate is roughly 1 100 Mb/s. The decrease is roughly linear which gives an average output of 1 250 Mb/s (0,156 GB/s) per layer. If the 40 layers then captured 1 s of data at this rate, the detector would generate:

$$40\ layers * 0,156\ GB/s * 1\ s = 6,24\ GB \text{ of data}$$

The final setup would then require a minimum on board memory of:

$$6.24\ GB * 3,5\ scale\ factor = 21,84\ GB$$

This shows that a minimum of 22 GB of memory is required to complete this process in memory and write to disk, for the whole detector, with the current setup. This compares favourably to the 64 GB of memory installed on the test machine.

With the binary file writer, the memory consumption scales by a factor 1,88.

$$6.24\ GB * 1\ s * 1,88\ scale\ factor = 11,73\ GB$$

This reduces the memory requirement to 12 GB of memory for a complete data take.

6.1.3 Scaling

The important part of this thesis is the scaling. This is achieved by providing each layer/Readout Unit with a dedicated software instance for processing. Going by the example above, the processing of these 6.24 GB will still only take 4.8 seconds.

Because each layer has it's own dedicated software instance, the processing is of the data stream from each layer is an independent process. The total processing time will therefor equal the processing time of the layer generating the most data.

This requires that no other hardware bottleneck exists. If the software is to run in virtual machines in a server, it is feasible the the hard drive write capacity might be exceed as this scaling will create 43 separate root tree files, one for the software instance of each layers. It is also possible that other unknown bottlenecks might be found.

Each root tree file can be opened and viewed immediately upon completion but to open all, the entire process will have to be completed. The unification of these files for further processing or reconstruction has not been solved. It is also unknown if the root tree is the format that is going to be used as a basis for the reconstruction software. The RUDataHeader format creates the possibility to replace the pRU Exporter submodule with a network interface instead.

6.1.4 Summary

The current processing speed does not allow for proper online monitoring of a high intensity beam. However, it is sufficiently fast to quickly verify a test setup and process reasonable amounts of data in a reasonable time frame.

Currently the biggest blocker for additional increases in the processing speed is the root tree creation. The rest of the the two software modules are already close to or at the speeds predicted by the simulations and clear improvement paths exists. The root module is very simple in its current form so might need a fundamentally different approach.

6.2 Design evaluation

The design has shown itself to be quite adaptable as the current modules can be easily changed to alter the software's function. This is demonstrated both by how easy it is to create new executables and how easy it is to modify existing executables by replacing some modules. There is some issues with the control loop but this is outside the scope of this thesis.

The for the goal to facilitate further development, the biggest remaining obstacle is that the pRU Parser is still a monolithic module. Future changes will therefor have to be done on the whole module increasing complexity and requires a developer to digest more coder in order to change module.

The ALPIDE Decoder in the Root Writer module has some sections outside the control loops where know elements are. This can be somewhat confusing and it is a

performance consideration if these sections should be compressed and moved into an expanded control structure.

6.2.1 Temporary Buffers

Currently the rejected data is only stored in a temporary buffer, the buffer persists with the parsing module so it is necessary to manually extract this data along with any partially completed frames remaining. If this is not done the data is lost when the module goes out of scope. A plan to more permanently deal with this necessary and the approach is different depending on what software element is chosen to have this responsibility.

6.2.2 Correctness

For the default, filtered data stream from the Readout units, the parser has been functioning well. However, the parser does not handle and unfiltered stream and the inclusion of these ALPIDE words will lead to a mislabeling of the element. Even if the event should not be fatal, any error generated will not be indicative of the underlying cause.

6.3 Future work

Although a step in the right direction, there is still work that needs to be done. Better Unit Testing for coverage from both modules is chief among them. This work should also be used to set up and clearly define the desired functionality of the module under testing. This should include functionality where it is desirable to have the ability to turn it on and off.

Since the filtering of these ALPIDE Words can be turned on and off in the Readout Unit, this functionality should be included in the pRU parser as well. If thes ALPIDE Words are being filtered by the RU, it is not necessary to spend CPU resources loading this code and checking for these elements.

More functionality is desired, especially from the parser module. The most important upgrade is improved event handling. It would be a significant improvement in correctness if the pRU parser module could handle events that are currently not covered. Currently only the pRU words have full coverage while ALPIDE Words like the COMMA, BUSY etc are not checked for.

The pRU Parser's error buffer currently have to be inspected manually. A module that could read and present this content in a more human friendly format would ease debugging.

The monolithic pRU Parser module should be divided into at least two separate submodules. This would make the code easyer to modify and it would enable the threading of the error checking section of the parser. Threading this section has the potential to increase the processing speed significantly.

Implement threading and parallel processing in the pRU error checking and in the ALPIDE Decoder. If a sufficiently fast buffer solution can be found, this could greatly increase the processing speed of the software instance. Most importantly, decoupling the Root Writer module, I believe significant gains can be made by allowing it to work in parallel with the rest of the readout chain instead of forcing the slowest element to wait for the completion of the previous steps.

Improvements can still be made by removing unnecessary copying present in the software modules. This will have the additional benefit of improve memory footprint of each software instance. A similar improvement is to change location of some elements to let them go out of scope sooner, This will free up additional memory reducing the overall memory footprint of the application.

Bibliography

[1] Herland A. private communication, 2020.

[2] Hilde A.E. private communication, 2019.

[3] ALICE Collaboration. A Forward Calorimeter (FoCal) in the ALICE experiment. *CERN Document Server*, October 2019.

[4] The PRaVDA Consortium. Proton radiotherapy verification and dosimetry applications.

[5] ALICE ITS ALPIDE development team. Alpide operations manual, July 2016.

[6] Manjit Dosanjh. The changing landscape of cancer therapy, 2020. Accessed: 2020-06-15.

[7] Dale E. and Waldeland E. Proton therapy – a reality in Norway from 2023. *Tidsskriftet - Den Norske Legeforening*, 138, September 2018.

[8] Rinella G.A. The alpide pixel sensor chip for the upgrade of the alice inner tracking system. In *Vienna Conference on Instrumentation*, VCI, Viena, Austria, 2016. indico.

[9] Rinella G.A. 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*, 845:583–587, February 2017.

[10] Particle Therapy Co-Operative Group. Particle therapy facilities in a planning stage, 2020. Accessed: 2020-04-11.

[11] Particle Therapy Co-Operative Group. Particle therapy facilities in clinical operation, 2020. Accessed: 2020-04-11.

[12] Particle Therapy Co-Operative Group. Particle therapy facilities under construction, 2020. Accessed: 2020-04-11.

[13] Paganetti H. Range uncertainties in proton therapy and the role of monte carlo simulations. *Physics in medicine and biology*, 57:R99–117, June 2012.

[14] Underdal H.A. Design of high-speed digital readout system for use in proton computed tomography, 2019.

[15] Pettersen H.E.S. *A Digital Tracking Calorimeter for Proton Computed Tomography*. PhD thesis, The University of Bergen, 2018.

[16] Pettersen H.E.S. and et al. Design optimization of a pixel based range telescope for proton computed tomography. Submitted to Physica Medica Special Issue: Advances in Geant4 for medicine.

[17] Pettersen H.E.S. and et al. Proton tracking in a high-granularity digital tracking calorimeter for proton ct purposes. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 860:51–61, July 2017.

[18] Sykehusbygg HF. Konseptfase – etablering av protonbehandling, sluttrapport, 06 2016. Avalible at: https://www.helse-sorost.no/Documents/Store utviklingsprosjekter/OUS/Radiumhospitalet/Konseptrapport - 0etablering av protonbehandling.pdf.

[19] Adiga H.S. Porting linux applications to 64-bit systems.

[20] http://ww1.microchip.com/downloads/en/DeviceDoc/MIC2915x-30x-50x-75x High-Current-Low-Dropout-Regulators-DS20005685B.pdf. Mic2915x/30x/50x/75x high-current low dropout regulators, 2020.

[21] Seward J. and Nethercote N. Using valgrind to detect undefined value errors with bit-precision. In *Proceedings of the USENIX'05 Annual Technical Conference*, USENIX ATC 05, pages 17–30, Anaheim, California, 2005. USENIX.

[22] Bohne K.E.S. Ethernet-based control system and data readout for a proton computed tomography prototype, 2018. Available at: http://dspace.uib.no/handle/1956/18466.

[23] Leksell L. The stereotaxic method and radiosurgery of the brain. *Acta chirurgica Scandinavica*, 102(4):316–319, December 1951.

[24] Leksell L. Stereotactic radiosurgery. *Journal of Neurology, Neurosurgery, and Psychiatry*, 46(9)::797–803, September 1983.

[25] LAP. Positioning at ct/pet-ct, 2020. Accessed: 2020-06-07.

[26] Esposito M. and et al. Pravda: The first solid-state system for proton computed tomography. *Physica Medica*, 55:p149–p154, November 2018.

[27] MarkFilipak. Thematic diagram showing dose as a function of depth for overlay of proton radiotherapy and x-ray radiotherapy to facilitate a comparison of the two radiotherapy methods., 2020. Accessed: 2020-06-15.

[28] P. Martinengo. The new inner tracking system of the alice experiment. *Nuclear Physics A*, 967:900 – 903, 2017. The 26th International Conference on Ultra-relativistic Nucleus-Nucleus Collisions: Quark Matter 2017.

[29] University of Bergen. Wiki for the bergen proton ct project. Available at: https://wiki.uib.no/pct/index.php/Main_Page.

[30] Groettvik O.S. and etal.. Development of Readout Electronics for a Digital Tracking Calorimeter. In *Proceedings of Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2019)*, volume 370, page 090, 2020.

[31] Grøttvik O.S. Design of high-speed digital readout system for use in proton computed tomography, 2017.

[32] Grøttvik O.S. pru data format specification, 2018.

[33] Johnson R. P. and et al.. A fast experimental scanner for proton ct: Technical performance and first experience with phantom scans. *Transactions on Nuclear Science*, 63(1):p52–p60, February 2016.

[34] P. Piersimoni and et al.. A High-Granularity Digital Tracking Calorimeter Optimized for Proton CT. Submitted to Frontiers in Physics. Currently under review.

[35] Wu Q.J. and et al. On-board patient positioning for head-and-neck imrt: Comparing digital tomosynthesis to kilovoltage radiography and cone-beam computed tomography. *International Journal of Radiation Oncology*Biology*Physics*, 69:598–606, 2007.

[36] Wilson R.R. Radiological use of fast protons. *Radiology*, 47(5):p487–p491, November 1946.

[37] Stortinget. Stortinget - møte onsdag den 13. november 2013 kl. 10 spørsmål 5. Available at: https://www.stortinget.no/no/Saker-og-publikasjoner/Publikasjoner/Referater/Stortinget/2013-2014/131113/ordinarsporretime/5/.

[38] Bodova T. private communication, 2020.

[39] The ROOT Development team. Root a data analysis framework. Accessed: 2020-006-10.

[40] Santa Cruz' Institute for Particle Physics University of California, Loma Linda University, and San Bernadino California State University. The pct collaboration.

APPENDIX A

Testing Tesults

The data size used for the speed calculation is primarily the data read from disk. This is 3 146 240 kilobytes or 25 169,92 megabit. The exception is the Root Writer module which uses the size of the input to the module. Also, the input for the pRU Exporter sub-module is incorrect since the ALPIDE Decoder transform the binary input in to the RUDataHeader header format. This format conceals the size of the input. since no data is removed, and there is a little overhead in both formats, the input size to the ALPIDE Decoder is used as a reasonable substitute.

The tables are color coded so that measured value have a blue background while calculated values have a white background.

Table A.1: Offline processing to a root tree. This includes both file read and file write.

Data volume (Mb)	Time (s)	Speed Mb/s
25 169.92	86.640	290.511
25 169.92	88.256	285.191
25 169.92	86.684	290.365
25 169.92	87.955	286.169
25 169.92	86.587	290.691
25 169.92	86.612	290.606
25 169.92	87.604	287.315
25 169.92	85.225	295.334
25 169.92	86.145	292.182
25 169.92	87.649	287.168
25 169.92	85.978	292.748
25 169.92	85.146	295.609
25 169.92	86.861	289.771
25 169.92	85.979	292.745
25 169.92	85.976	292.756
25 169.92	86.197	292.006
25 169.92	85.750	293.527
25 169.92	87.352	288.145
25 169.92	86.757	290.121
25 169.92	85.925	292.928
Mean	86.564	290.794
Standard deviation	2.880	

Table A.2: Offline processing to a binary file. This includes both file read and file write.

Offline processing to binary Data volume (Mb)	Time (s)	Speed Mb/s
25 169.92	25.996	968.206
25 169.92	25.970	969.175
25 169.92	25.929	970.725
25 169.92	25.942	970.233
25 169.92	26.500	949.825
25 169.92	25.953	969.833
25 169.92	26.034	966.804
25 169.92	26.049	966.236
25 169.92	26.087	964.848
25 169.92	26.455	951.429
Mean	26.092	964.731
Standard deviation	7.672	

Table A.3: Online processing to a root tree. This does not includes file read.

Data volume (Mb)	Time (s)	Speed Mb/s
25 169.92	84.858	296.613
25 169.92	84.903	296.457
25 169.92	85.968	292.782
25 169.92	85.1475	295.604
25 169.92	86.472	291.078
25 169.92	86.637	290.521
25 169.92	87.179	288.717
25 169.92	85.375	294.815
25 169.92	86.049	292.506
25 169.92	85.282	295.137
25 169.92	85.921	292.943
25 169.92	86.310	291.622
25 169.92	85.058	295.916
25 169.92	84.266	298.696
25 169.92	84.120	299.214
25 169.92	84.968	296.227
25 169.92	86.149	292.167
25 169.92	85.705	293.680
25 169.92	85.194	295.443
25 169.92	85.654	293.856
Mean	85.561	294.200
Standard deviation	2.705	

Table A.4: Online processing to a binary file. This does not includes file read.

Data volume (Mb)	Time (s)	Speed Mb/s
25 169.92	25.392	991.245
25 169.92	25.668	980.585
25 169.92	25.333	993.546
25 169.92	25.487	987.557
25 169.92	25.172	999.936
25 169.92	25.438	989.448
25 169.92	25.506	986.839
25 169.92	25.344	993.115
25 169.92	25.351	992.846
25 169.92	25.253	996.705
Mean	25.395	991.182
Standard deviation	5.442	

Table A.5: pRU Parser Module using the input file size.

Data volume (Mb)	Time (s)	Speed Mb/s
25 169.92	24.499	1 027.400
25 169.92	24.603	1 023.050
25 169.92	24.473	1 028.490
25 169.92	24.494	1 027.610
25 169.92	24.499	1 027.390
25 169.92	24.471	1 028.580
25 169.92	24.522	1 026.410
25 169.92	24.532	1 025.990
25 169.92	24.918	1 010.130
25 169.92	24.628	1 021.990
25 169.92	24.552	1 025.160
25 169.92	24.469	1 028.640
25 169.92	24.524	1 026.350
25 169.92	24.597	1 023.290
25 169.92	24.468	1 028.700
25 169.92	24.736	1 017.560
25 169.92	24.477	1 028.320
25 169.92	24.838	1 013.370
25 169.92	24.997	1 006.930
25 169.92	24.572	1 024.350
Mean	24.593	1 023.486
Standard deviation	6.467	

Table A.6: Complete root writer Module using the input file size.

Data volume (Mb)	Time (s)	Speed Mb/s
17 979.136	60.746	295.972
17 979.136	60.519	297.083
17 979.136	63.265	284.188
17 979.136	61.325	293.176
17 979.136	61.018	294.651
17 979.136	60.840	295.513
17 979.136	61.096	294.277
17 979.136	61.498	292.352
17 979.136	61.130	294.113
17 979.136	61.250	293.536
17 979.136	61.216	293.702
17 979.136	61.774	291.048
17 979.136	60.721	296.093
17 979.136	60.312	298.102
17 979.136	61.034	294.578
17 979.136	61.366	292.981
17 979.136	61.729	291.257
17 979.136	60.171	298.801
17 979.136	63.274	284.148
17 979.136	60.785	295.784
Mean	61.253	293.568
Standard deviation	3.801	

Table A.7: ALPIDE decoder submodule using the input file size.

Data volume (Mb)	Time (s)	Speed Mb/s
17 979.136	12.711	1 414.410
17 979.136	12.755	1 409.553
17 979.136	12.729	1 412.499
17 979.136	12.738	1 411.412
17 979.136	12.742	1 411.047
17 979.136	12.705	1 415.156
17 979.136	12.739	1 411.368
17 979.136	12.719	1 413.576
17 979.136	12.678	1 418.114
17 979.136	12.733	1 412.055
17 979.136	12.711	1 414.455
17 979.136	12.738	1 411.479
17 979.136	12.720	1 413.465
17 979.136	12.618	1 424.891
17 979.136	12.744	1 410.803
17 979.136	12.732	1 412.100
17 979.136	12.739	1 411.390
17 979.136	12.724	1 412.988
17 979.136	12.769	1 407.986
17 979.136	12.681	1 417.790
Mean	12.721	1 413.327
Standard deviation	3.666	

Table A.8: pRU Exporter (root tree) submodule using the input file size from the alpide decoder.

Data volume (Mb)	Time (s)	Speed Mb/s
17 979.136	46.411	387.391
17 979.136	47.813	376.027
17 979.136	46.207	389.101
17 979.136	47.137	381.425
17 979.136	47.726	376.718
17 979.136	47.191	380.987
17 979.136	46.682	385.140
17 979.136	46.416	387.349
17 979.136	46.896	383.387
17 979.136	46.622	385.633
17 979.136	45.691	393.491
17 979.136	47.217	380.781
17 979.136	46.862	383.660
17 979.136	46.360	387.815
17 979.136	46.134	389.715
17 979.136	46.298	388.333
17 979.136	47.017	382.398
17 979.136	46.306	388.268
17 979.136	45.574	394.506
17 979.136	46.100	390.001
Mean	46.633	385.606
Standard deviation	4.960	