Design of High-Speed Digital Readout System for Use in Proton Computed Tomography

A thesis by

Ola Slettevoll Grøttvik

for the degree of

Master of Science in Physics

Department of Physics and Technology

University of Bergen

June 2017

Abstract

A team at the University of Bergen is developing a proton computed tomography (CT) scanner prototype. The proton CT (pCT) scanner will introduce fewer insecurities than conventional photon CT when analyzing the location of the tumor and the particle stopping power in the tissue in front of the tumor. The pCT will consist of multiple CERN-developed particle detector chips.

This work covers the design and development of a readout system for the pCT scanner. The requirements for the different parts of the system are formulated. Particular emphasis is put on the clock data recovery method employed for handling the 1.2 Gb/s data transmitted from the detector chips. Because of the high number of detector chips in each scanner layer, transceivers cannot be employed for this purpose. This constraint results in a custom design employing the I/O primitives to achieve phase and word alignment automatically.

A complete verification system is realized around a simulatable model of the particle detector chip. This system is used for verification of multiple readout procedures, as well as validation of detector chip communication and firmware functionality. The simulations are further verified in a physical hardware setup. The results of the I/O primitive approach are positive, but some errors are produced when employing longer cables.

Preface

This work was carried out at the Department of Physics and Technology at the University of Bergen between August 2016 and June 2017. The pCT project was just initiated, and little documentation or information existed. Unraveling the design requirements and solving the challenges that turned up was done without much prior knowledge of a large system design.

Acknowledgments

This work was largely influenced by my two great supervisors, *Professor Kjetil Ullaland* and *Associate Professor Johan Alme*. Thank you very much for all your help, guidance and patience. Working with something as important as cancer therapy, although indirectly, has been deeply rewarding. Thanks to *Professor Dieter Röhrich* for heading the pCT group and letting me be a part of it.

A huge thanks to *Attiq Ur Rehman*, *Simon Voigt Nesbø* and *Shiming Yang* which provided invaluable insight at several stages of the process. I would also like to express my gratitude to my fellow students at room 503. It has been two gratifying years and I hope I will encounter you all in the future.

Finally, I would like to thank my friends and family. A special thanks to Ingrid for comments and suggestions. Finally, I would like to especially thank Ida and my parents. Thank you for enduring everything and always being there.

Ola Slettevoll Grøttvik
Bergen, June 2017

Contents

Acronyms

ADC Analog-to-Digital Converter
ALICE A Large Ion Collider Experiment
ALPIDE ALICE Pixel Detector
ALWM ALPIDE Lightweight Model
AMBA Advanced Microcontroller Bus Architecture
ASIC Application Specific Integrated Circuit
AXI Advanced eXtensible Interface

BER Bit Error Rate
BFM Bus Functional Model

CDR Clock Data Recovery
CERN Conseil Europen pour la Recherchè Nuclaire
CMOS Complementary Metal-Oxide Semiconductor
Glossary: CMOS Process
CPU Central Processing Unit
CT Computed Tomography

DDR Double Data Rate
DMA Direct Memory Access
DTC Digital Tracking Calorimeter
DUT Device Under Test

FIFO First In First Out
FPGA Field Programmable Gate Array
FROMU Framing and Management Unit

GBT GigaBit Transceiver
Glossary: GBT

HDL Hardware Description Language
HP High Performance
HR High Range

IC Integrated Circuit
ILA Integrated Logic Analyzer
IP Intellectual Property
ITS Inner Tracking System

JTAG Joint Test Action Group

LET Linear Energy Transfer
LHC Large Hadron Collider
LSB Least Significant Bit
LVDS Low-Voltage Differential Signaling

MAC Media Access Control
MEB Multi-Event Buffer
MGT Multi-Gigabit Transceiver
MMCM Mixed-Mode Clock Manager
MSB Most Significant Bit

PCB Printed Circuit Board
pCT Proton CT
PHY Physical Layer
PLL Phase Locked Loop
pRG Proton Radiography
PRUdense pCT Readout Unit

RAM Random Access Memory
RISC Reduced Instruction Set Computing
RO Read Only
ROM Read Only Memory

RTL Register Transfer Level
RW Read/Write

SCU System Control Unit
SEL Single Event Latch-up
SERDES Serializer/Deserializer
SEU Single Event Upset
SOBP Spread-Out Bragg Peak
Glossary: Bragg peak
SoC System-On-Chip
SRAM Static Random Access Memory

UART Universal Asynchronous Receiver Transceiver
USB Universal Serial Bus
UVVM Universal VHDL Verification Methodology

VHDL Very High Speed Integrated Circuit Hardware Description Language
Glossary: VHDL

WO Write Only

Glossary

8B/10B
: An encoding algorithm that encodes a 8-bit word into a 10-bit word to achieve DC-balance.

Bragg peak
: The region of high dose at the end of a charged particle's range.

CMOS Process
: A specialized manufacturing process technology that are implemented to create transistors on silicon wafers to make ICs. The technology is often described by the length of a single transistor channel.

Combinational Design
: A system in which outputs are a function of only its present input values.

Fault
: Failure of a subsystem. May and may not produce an observable error.

GBT
: Bidirectional optical link aimed at being radiation hard. Developed at CERN for use in Large Hadron Collider (LHC) upgrades.

Half-Duplex
: In communications, a half-duplex system can communicate in both directions, but not simultaneously.

IP Core
: An Intellectual Property core is collection of logic or code, that is specific to do a certain task in an IC often represented as a behavioral module. IP cores are an integral part of design reuse in electronic design automation industry.

Monolithic Active Pixel Sensor
: Image sensor where all pixel detectors are implemented with individual amplifiers on IC with readout electronics included.

Rise Time
: The amount of time required for a signal to transition from 10 percent to 90 percent of its final steady value.

Synthesis
: The process of generating a representation of function in a lower level based on a more abstract level. Usually refers to when an FPGA design is transformed from a RTL level to logic gate level.

Twinax
: Type of cable similar to a coaxial cable, but with two conductors instead of one. Becoming common in short-range high-speed applications.

VHDL
: A language for describing digital electronic systems, and widely used for developing FPGA firmware.

Vivado Design Suite
: Software suite for synthesis and analysis of HDL designs targetting Xilinx FPGAs.

Wrapper
: A module which converts the signals of an interface to another interface. E.g., used to convert signals from a Wishbone compatible slave to an AXI compatible master.

CHAPTER 1

Introduction

1.1 Background and Motivation

The most common form of radiation therapy of cancer in Scandinavia has, until recently, been utilizing high energetic photons. Lately, greater focus has been given to radiation therapy with charged particles. This method benefits the patient due to the reduced irradiated volume of the patient during treatment and by maximizing the ratio between dose to tumor and dose to healthy tissue [1]. This reduces side effects like radiation-induced secondary cancers. In Norway, focus on proton treatment was given when the government in 2013 funded a new report targeting the possibilities for new radiation therapy centers [2].

137 000 patients were by January 2015 already treated with charged particles around the world, in which 86% were treated with protons [3]. However, the current technique for calculating how the particles will be absorbed by the body is using X-ray photons [4]. This method introduces uncertainties that can be reduced by employing proton computed tomography [5]. The University of Bergen has been granted project funding to develop a prototype proton CT scanner.

1.2 About this Thesis

The primary goal of this thesis is to start the design and development process of a readout system to be used in the proposed proton CT scanner. The project plans to use several of the CERN-developed particle detector chip called ALPIDE. These will be organized in several layers of multiple sensor chips, where all chips will be connected to FPGAs. The readout system must, therefore, work in a highly pipelined fashion in which each data channel operates independently of each other. Furthermore, since the data speed of every detector chip is 1.2 GB/s, the data must be processed in a very efficient way to prevent bottlenecks during readout.

The readout system must also handle the control of all the chips. Each detector chip includes a control channel which serves two purposes [6]:

1. provide read and write access to internal chip registers, commands, configuration, and memories
2. distribution of trigger and other broadcast commands

The readout system must be able to perform all the necessary configurations of the chips, and also broadcast trigger commands synchronously to all detectors.

This work builds on the effort done by the ITS upgrade team at CERN, who are using the same sensor chip. Some time was spent on setting up a simulation testbench environment in which a simulatable model of the ALPIDE chip was incorporated. This testbench environment enabled modification of the firmware to represent the Proton CT (pCT) requirements. Later a physical test environment was set up using an ITS-developed PCB and a single ALPIDE chip.

A quite substantial amount of effort was put into the design of the pCT readout system concept and especially formulating requirements for the different parts of the system. This required both insights in the different ways ALPIDE chips operate, as well as how the triggering and timing should work to produce consistent data.

Different readout procedures were developed and tested in both the simulation environment and physically in hardware. In particular, a major challenge was to develop a reliable sampling method that employed regular I/O pins without the help of transceiver features.

A significant effort was put into the communication system and particularly defining a register handling format for the bus interface. This also included writing software to automatically produce the required register handling firmware, although this task is not entirely completed at the end of this work.

Software development was required to test the hardware designs thoroughly. This work involved using the Python programming language.

The majority of this work was done while interfacing a test board which was poorly documented. This brought an extra challenge to the process.

A strong emphasis was put on documentation and version control to ease future development. All code and documentation are collected in a git repository, and all changes done are thoroughly logged with commit messages. Notable milestones are saved with tags. All VHDL code written complies to the department's guidelines. All simulation, synthesis, and testing are accomplished by employing scripts and automated software.

1.3 Thesis Outline

This thesis is divided into the following chapters:

Chapter 2: Radiation and Cancer Therapy This chapter starts with theory regarding radiation and cancer therapy. This theory is required to understand some of the benefits of proton cancer treatment and a proton CT scanner, which is explained in the second part of the chapter.

Chapter 3: ITS and the ALPIDE Chip This chapter describes the ALICE detector and the ITS upgrade, and is included due to the considerable similarities between this system and the proton CT scanner. An overview of the ALPIDE chip's operation modes and interfaces is given to help the reader understand some of the design choices made in later chapters. Ultimately, some of the ITS readout unit's modules are introduced and explained. These modules constitute the basis of the work done with the pCT readout unit.

Chapter 4: Readout System This chapter contains a comprehensive discussion of the complete pCT readout system. The system as a whole is introduced, followed by a more detailed discussion about the different units. Particular emphasis is put on the various stages of the readout process where data on the high-speed data interface on the ALPIDE is sampled by the readout unit. After that, the communication and control parts of the system are described before a discussion about the timing requirements and the trigger generation. Finally, the power control system is briefly outlined.

Chapter 5: Development and Testing This chapter reviews the development of the readout unit and the simulation and testing procedures.

Chapter 6: Discussion and Conclusion This chapter summarizes the work for this thesis, and then discusses the further tasks required to complete the system. Finally, there is a brief section that concludes the thesis and the results of the work.

Appendices The last section of the thesis includes multiple appendix chapters with too many details to be included in the main thesis. It includes both description of registers in the system, the procedures and functions of the ALPIDE BFM, as well as firmware and testbench overviews. The section also includes a summary of the test software, and an overview of the code repository. The two last appendices are about the problems that may arise with high-speed digital signals and the methods needed when designing complex hardware. Specifically, the last chapter outlines Weste and Harris' [7] design strategies. Subsequently, a summary of techniques for detecting errors in a simulation environment is presented before some hardware testing principles are described.

1.4 Citation Principles

This thesis is using the principle that citations listed after the ending punctuation of a paragraph may refer to several statements in the section. Citations listed before any punctuation will always refer to the last statements.

CHAPTER 2

Radiation Therapy and Proton CT

This chapter outlines some of the differences between the behavior of electromagnetic radiation and charged particles. Further, it explains how these differences affect the patient when employing radiation for cancer therapy. Finally, the benefits of proton CT is presented.

2.1 Radiation

By definition, radiation is the emission and propagation of energy through space or a material medium. Radiation is characterized in various ways; including ionizing and non-ionizing radiation or charged particles and electromagnetic waves. [8, Chapter 1][9, Chapter 1]

Radiation is also characterized by the energy of the beam particle. If the energy is high enough to completely remove an electron from an atom by transferring energy to the medium, it is known as ionizing radiation. Some radiation is known as directly ionizing radiation because of how the particles directly can interfere with atoms. These include α and β particles,[1] as well as ions like protons and ^{12}C. Uncharged particles like neutrons and photons are indirectly ionizing radiation, however, and they achieve ionization by creating by-products that can interact with atoms. [9, Chapter 5]

2.2 Charged Particles Interaction with Matter

Particles will be losing velocity and energy while traveling through matter. The energy, charge, and mass of the particles, and the type of matter it is moving in

[1] α-particles are ^4_2He-nuclei, and β-particles are charged particles indistinguishable from an ordinary electron, but may have positive charge [10].

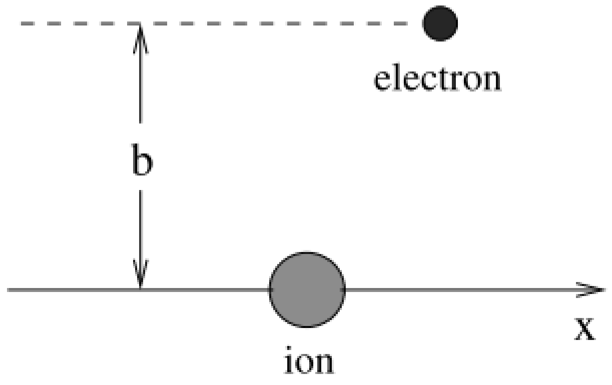


Figure 2.1: A charged particle is moving in the electron's electric field [8].

determine the type of processes that will be involved and the scale of the effects. For charged particles, several mechanisms will contribute to energy loss. Nonetheless, inelastic scattering with orbital electrons in the matter will often be the dominating mechanism due to the high electron density in matter.[2] This interaction can be modeled as in Figure 2.1. As the ion passes the orbital electron, the transfer of energy is caused by the Coulomb force between the charges. The amount of energy the ion loses depends on the distance between the particles, the amount of charge and the amount of time the ion uses in passing the electron. The latter means that the ion will lose more energy if it moves more slowly. The less kinetic energy, the more energy will be lost. [11, Chapter 4][10, Chapter 5]

An important parameter is specific ionization, which expresses the linear rate of energy loss due to ionization and excitation. For a charged particle, specific ionization is the number of ion pairs formed per unit distance traveled. Figure 2.2 shows the relationship between the specific ionization and the Bethe-Bloch formula energy for a β-particle. Relativistic effects cause the somewhat increasing specific ionization at higher energy. Specific ionization is used when one is interested in the radiated particles' energy loss. If one is more focused on the absorber material, like in radiation therapy and radiation protection, it is common to use the term *Linear Energy Transfer (LET).* This term quantifies the linear rate of energy absorbed by the medium and is defined by

$$LET \leq \tfrac{dE_L}{dl}$$

where $dE_L$is the average energy transferred to the medium traveling a distance dl. The need for two separate definitions implies that the energy locally absorbed by the medium is not equal to the energy lost by the moving particle. In fact, some of

[2]A 10 MeV proton will lose most of its energy after moving only 0.25 mm in copper.

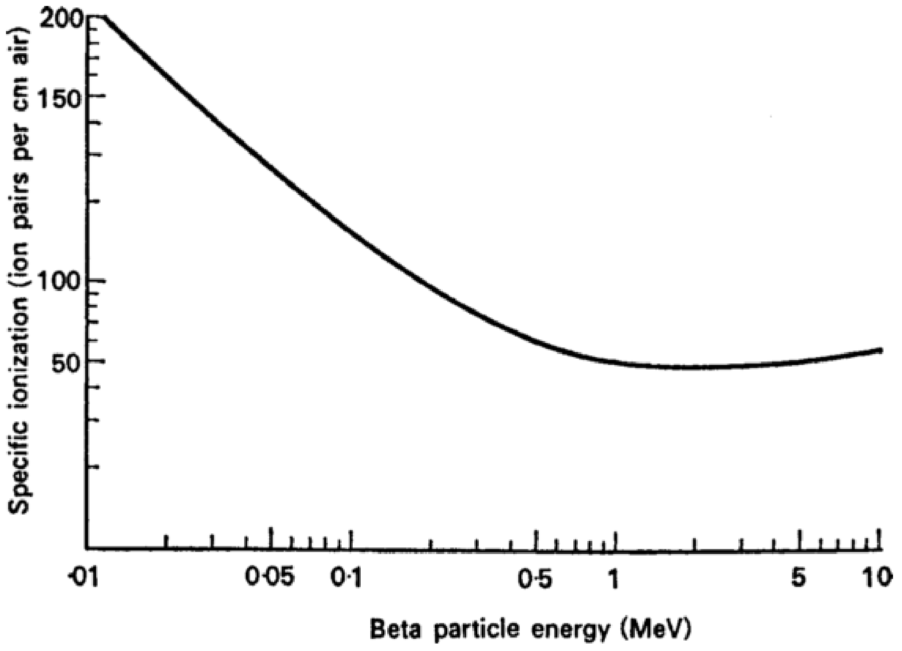


Figure 2.2: Relationship between particle energy and specific ionization in air [10, Figure 5-7].

the energy lost is carried away by energetic secondary electrons. [10, Chapter 5][11, Chapter 4][9, Chapter 14]

2.3 Photons Interaction with Matter

In 1925 de Broglie introduced the idea of the duality of nature and that electromagnetic waves could behave like particles [9, Chapter 1]. Electromagnetic radiation,[3] a beam of photon particles, will gradually lose intensity as it moves through matter. While α and β particles may be completely absorbed, electromagnetic radiation can only be reduced in intensity. This attenuation is partly due to scattering of the photons, and partly due to absorption of the photons. Figure 2.3 shows how the intensity is exponentially attenuated by increasing the absorber thickness. Note the logarithmic nature of the graph, which entails that the intensity will never be reduced to zero. [11, Chapter 5][10, Chapter 5]

The main contributors to the attenuation of electromagnetic radiation are three processes; photoelectric effect, Compton scattering, and pair production. Photoelectric absorption occurs when a photon hits an orbital electron with a binding

[3]High energy electromagnetic radiation may, depending on the energy level, also be called gamma (γ) or X-ray radiation.

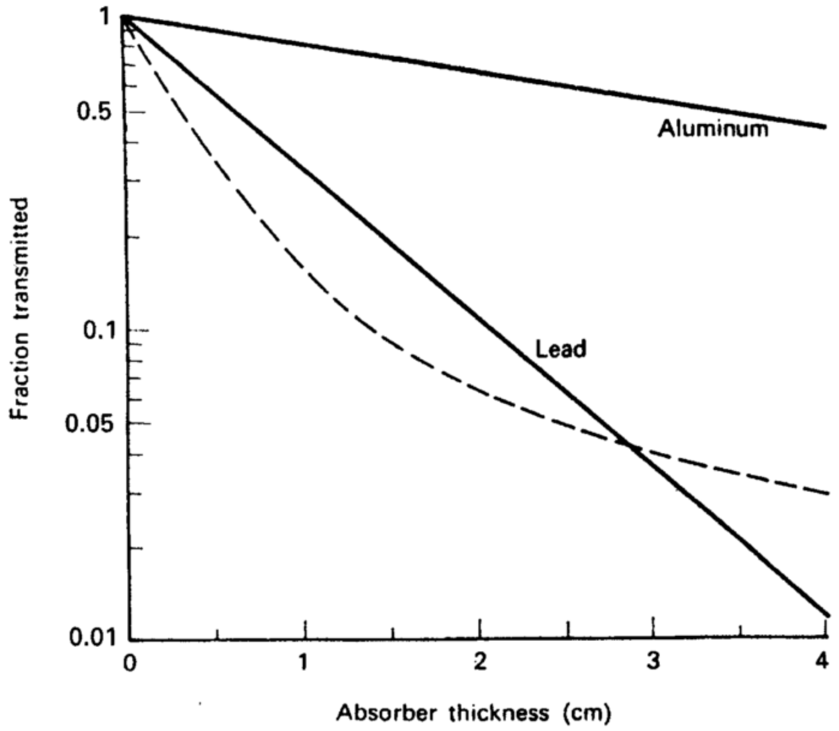


Figure 2.3: Attenuation of gamma rays. Solid lines represent attenuation for monoenergetic gamma rays, and the dotted line represents a heterochromatic beam [10, Figure 5-13].

energy equal to or less than the photon energy. The photon disappears, and the resulting particle, the photoelectron, transfers its energy[4] by excitation and ionization. Compton scattering occurs when there is an elastic collision between a photon and an electron in the outer shell of the atom. These electrons can be considered “free” in the sense that the binding energy can be radically smaller than the energy of the photon. This collision causes the photon to change course, and lose some energy. Further, the hit electron has obtained the energy difference between the original and the scattered photon, moving out of the atom in an angle that represents the change in wavelength. [10, Chapter 5][11, Chapter 5]

Photoelectric effect and Compton scattering are dominant at energies below 1.02 MeV, consequently the energy equal to the rest mass of two electrons. While at higher energies, a photon can convert into an electron/positron pair, a so-called pair production. This conversion of electromagnet energy into mass can only take place when a photon is passing close to a particle, such as a nucleus, so that momentum is conserved. [10, Chapter 5]

[4] $E_{pe} = hf - \phi$ where hf is the energy of the photon and ϕ is the binding energy of the orbital electron.

The attenuation also varies widely with the type of absorber material. The difference between lead and aluminum is shown in Figure 2.3. The difference is due to the atomic number. The Compton effect, however, is almost entirely independent of the atomic number of the absorber. This is because the Compton effect deals with electrons in the outer shell, so-called "free" electrons, which are independent of atomic number. [10, Chapter 5]

2.4 Range and Biological Effects

For a beam of particles, the range is defined as the distance or thickness of absorber material needed for there to be no more observed particles than the background radiation [10, Chapter 4]. The fundamental difference in how charged particles and photons interact with matter is the reason why the distance traveled in matter is different. The distance any charged particle can move in matter is decided by a number of discrete collisions with a statistical distribution [11, Chapter 4]. The relation between energy and deposited energy shows that a charged particle will lose more and more energy before it eventually stops. Whereas a beam of photons has no explicit range because only the intensity of the beam is reduced. This can be observed in Figure 2.3, where the intensity of the beam never equals zero.

As we have seen, the energy absorbed by radiation may cause ionization and excitation of atoms in the absorbing material. This ionization may break up molecular bonds in the absorbing material. It follows that this may be extremely hazardous for biological tissue; as molecules break, living cells and genetic material may be severely damaged. The cells and even whole organisms may die [12, Chapter 32]. There are both direct and indirect effects of radiation that may cause harm to the cells. One direct effect is point mutations, a change in a single gene [10, Chapter 7]. The ionization or excitation may cause a molecular lesion that inflicts damage to normal function. Such damages are often repaired, but sometimes they prevail and are transferred to new generations of DNA cells. An example of indirect radiation effects is the creation of toxic and hazardous chemicals by nuclear reactions. Since most of the human body is water, one possible reaction is the creation of hydrogen peroxide which is a powerful oxidizing agent [10, Chapter 7]. The consequences of over-exposure may be acute and deterministic, or delayed and stochastic, like genetic effects and cancer.

Measuring the damage done to living tissue is extremely complicated. Consequently, significant attempts have been made to control the exposure to living organisms, and radiation dosimetry is the effort to quantify the effect of radiation on living tissue. The absorbed dose is the energy deposited per unit mass.

$$D = \frac{\Delta E}{\Delta m}$$

where the unit is called Gray(Gy). $1Gy = 1J/1kg$. The absorbed dose may be used to measure all types of radiation, independent of the fundamental processes in which the energy is transferred. Even though the absorbed dose is not considered a sufficient measure of the biological effects of radiation, it is adequate to understand the benefits of cancer treatment with protons versus photons. [13, Chapter 43][10, Chapter 6]

2.5 Cancer and Radiation Therapy

The effects of ionizing radiation on biological tissue may be employed for health purposes, and are used on a large scale both for diagnostics and therapy. Until recently, the conventional form of radiation therapy utilized photons. Although protons and heavier ions were introduced as possible candidates for cancer treatment as early as in 1946 [14], the high complexity and cost of technical realization may have slowed the development compared to conventional radiation [15]. Therefore, proton therapy has been in clinical use only since the 1970's [16]. As discussed, a charged particle, like protons and carbons, will lose more energy as they go further into the matter before eventually reaching the Bragg peak. The Bragg peak is the region where the charged particle will lose the majority of its remaining energy, and a close-to-zero dose will be absorbed by the material beyond this peak [14]. This behavior differs fundamentally from photons, which after a short build-up region, will decreasingly deposit energy as they move in the matter [17]. Thus, as shown in Figure 2.4, radiation therapy which utilizes protons or heavy ions may be more precise in delivering the dose to the tumor.

The particle beam may be tuned so that the Bragg peak is placed at any depth into the patient's tissue [5], and therefore able to hit tumors at any depth. The main benefit in using protons and heavier ions for cancer therapy is a reduction of the total dose absorbed by the patient. As can be observed in Figure 2.5, the dose distribution of a photon beam covers a much larger region than the proton beam. This reduction in absorbed dose entails that secondary radiation effects also may be reduced [2]. Long-term perspective is of particular importance when treating cancer in children. The reduced risk for radiation-induced cancer and other hazards, brought about by dose reduction, makes proton treatment very promising for child treatment. Concurrently, while minimizing the irradiated area, proton treatment allow for an increase in dose delivered to the tumor, and thus increasing the potential for controlling the tumor [17]. To summarize, proton therapy will theoretically both reduce side effects of the treatment, and also increase the chances of controlling the tumor. More research is needed, but clinical evidence indicates that tumors close to critical structures are benefiting the most from proton treatment [16].

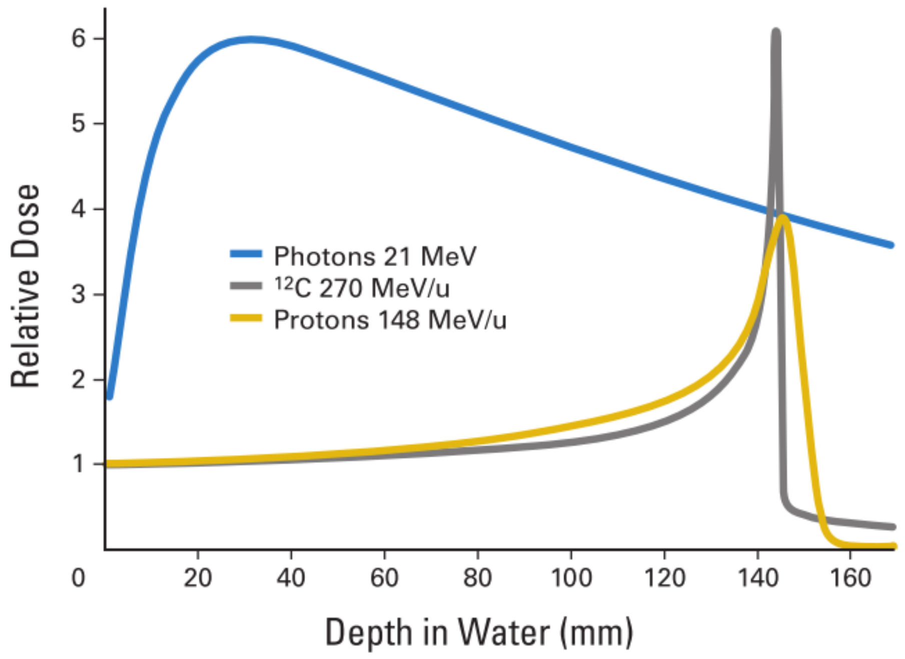


Figure 2.4: Relative dose distribution profiles of photons, protons and carbon ions [14, Fig 1]

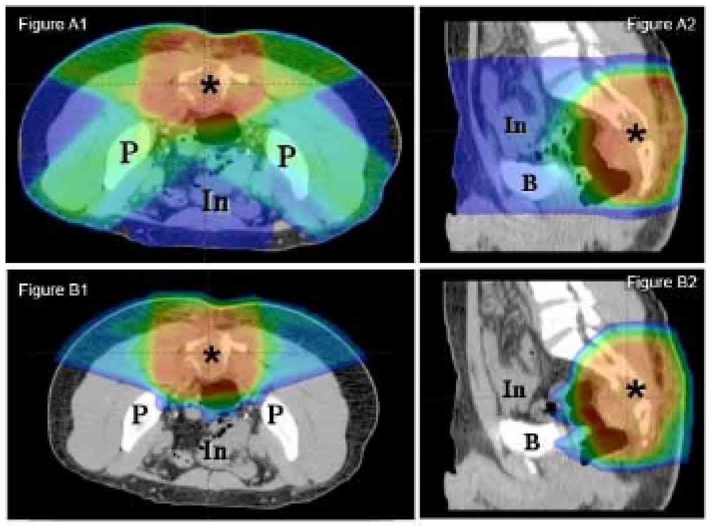


Figure 2.5: X-ray dose distribution (A1&A2) and proton dose distributions (B1&B2) [18].

As the Bragg peak is seldom wide enough, a set of beams with different energies are often needed in order to tune the proton beam to place the Bragg peak in the tumor, and cover the entire tumor [19]. The multiple beams are weighted with different energies to create the so-called Spread-Out Bragg Peak (SOBP), which covers the entire treatment volume. To achieve this, precise information regarding the tumor location and the particle stopping power in tissue preceding the tumor is needed. This information, however, is currently obtained mainly through X-ray photon CT [4]. The conversion between the behavior of photons in the CT and the protons in the therapeutic beam is associated with uncertainties up to several millimeters [5].

2.6 Proton CT (pCT)

The use of protons in CT is an imaging technique which will provide a low dose to the patient, and present range predictions which decrease the uncertainties introduced by X-ray photon CT [5]. While proton therapy requires protons to deposit their energy and stop in the tumor, proton CT requires particles to pass through the patient [20]. This is due to the particle's energy loss in the medium being the primary mechanism for image contrast and must be measured on the other side of the patient. Therefore, a proton CT scanner requires a high-energy proton beam. Figure 2.6 shows an example 3D rendering of pCT-reconstructed relative stopping power map.

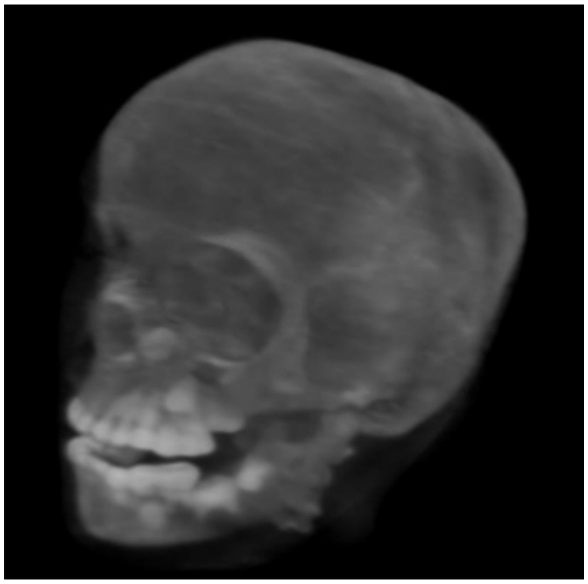


Figure 2.6: Example of 3D rendering of pCT-reconstructed relative stopping power map [5, Fig. 14].

Some proposed Proton Radiography (pRG) and pCT systems use a proton integrating approach, that calibrates a signal in a detector with the length traversed, averaged over many protons. This approach has some limitations; the proton integrating systems developed have produced degradation in spatial resolution and the presence of severe edge artifacts [20]. Proton-tracking is an alternative approach to obtain the required data. This technique measures each individual proton's residual energy after emerging from the patient. Figure 2.7 shows how a proton-tracking pCT system may be set up. It consists of multiple position-sensitive detectors and a detector measuring the residual energy-range. Multiple pCT prototypes follows this approach, including the project at the Loma Linda University [21] and the Italian project called PRIMA. Furthermore, these projects employ calorimeters or scintillators that require only one proton per element during a readout for valid measurements. [20]

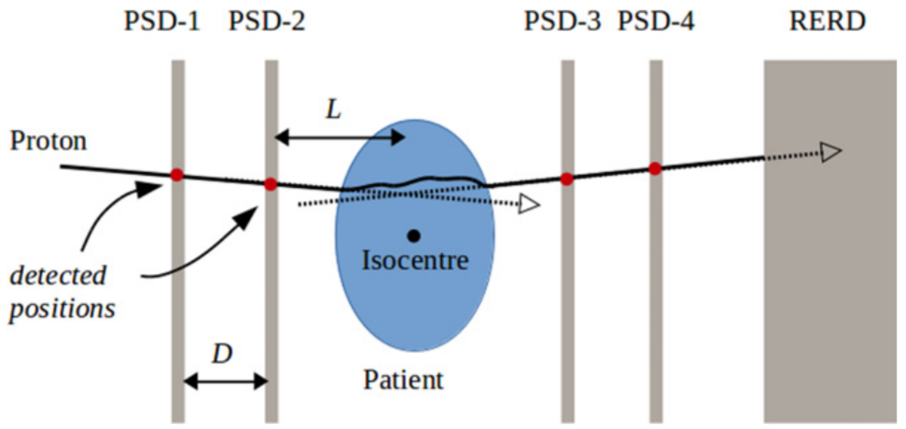


Figure 2.7: A proton-tracking proton radiography/proton CT system [20, Figure 5]. Consists of four position-sensitive detectors (PSD) and a residual energy-range detector (RERD).

2.6.1 Bergen pCT Prototype

The aim of the pCT prototype which is being developed at the Department of Physics and Technology at the University of Bergen is rapid and precise reconstruction of tracks of the protons traversed through the patient. The measurements are done with a high-granularity semiconductor calorimeter, also called the Digital Tracking Calorimeter (DTC). A high granularity detector is realized by structuring a large number of detector chips which consist of 1024x512 pixels each. Each sensor pixel can detect deposition of energy above a given threshold. The DTC is constructed by creating multiple layers of multiple detector chips. The DTC will use the CERN-developed particle detector chip ALPIDE. This chip is described in Chapter 3.

The setup differs from other proton tracking approaches in that both position and energy deposition is obtained by a single detector instrument, the DTC. The proton tracks measured will be used to calculate the deposited energy of the individual proton. The duality of this technique, calculating both the angle and position as well as the deposited energy of the protons, simplifies and reduces the cost of the pCT [4].

Furthermore, due to the pixelated nature of the CMOS detector, the DTC allows for measuring multiple protons per frame time. The high granularity of the sensors makes it possible to track several individual protons at the same time. This increases the proton rate capability of the detector and reduces the time required to produce a high-resolution image.

The structure of the pCT is not yet completely settled, and it follows that the task of designing a readout system must be highly adaptive and modular. A visualization of the structure and setup is presented in Section 4.1.

CHAPTER 3

ALICE ITS and the ALPIDE Chip

This chapter offers a brief presentation of the ALICE ITS detector upgrade, and explains the ALPIDE chip's operation modes and interfaces. It also presents a concise discussion of the ITS readout firmware modules. This design forms the basis of the pCT firmware development.

3.1 ALICE ITS

The ALICE experiment at CERN is one of four large detector experiments that are registering collision event data at the LHC. ALICE is focused on heavy-ion collisions and the strong interaction sector of the Standard Model [22]. ITS is a multiple layer, silicon vertex detector at the heart of ALICE. An upgrade of the system will place the first detection layer closer to the beam line, reduce the material budget and improve the maximum readout rate [23]. The upgrade is scheduled during the second LHC shutdown (LS2) in the years $2019 - 2020$.

Figure 3.1 shows how the ALPIDE chips will be placed in the ITS. There are seven layers of detector chips and the two outer and middle chip layers will be configured in the Outer Barrel mode. The output data rate of these chips will be reduced compared to the chips in the three inner layers, which are set up in Inner Barrel mode. The ALPIDE configuration modes are explained in Section 3.2.1. The different configurations and speed of the data streams are critical in the readout electronic's point of view [24]. It is challenging to accomplish clock data recovery with multiple data speeds. High-speed data recovery is often accomplished by utilizing transceivers (see Section 4.5.1) that require input data with a fixed rate.

The readout system makes use of 192 identical readout units which are interfaced to the ALPIDE chips, as illustrated in Figure 3.2. These units cover the full ITS and provide the clock, control, and data links. An external device, the Central Trigger

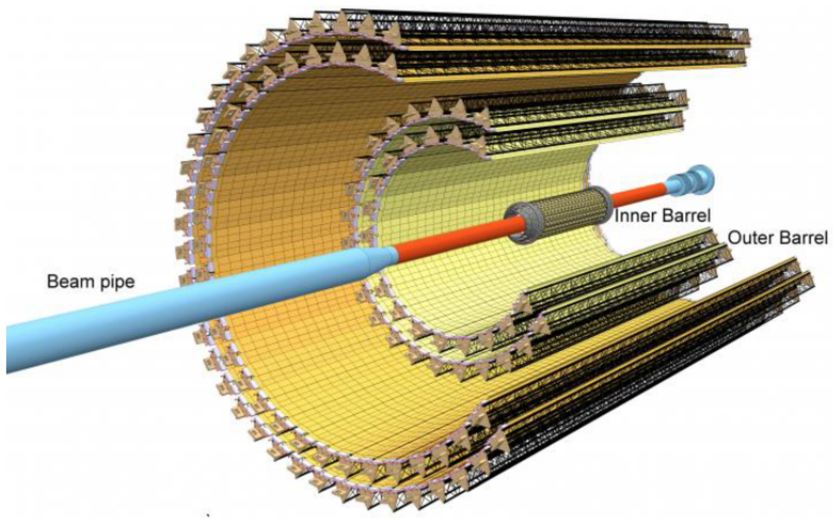


Figure 3.1: ALICE ITS seven layers layout [24, Figure 1].

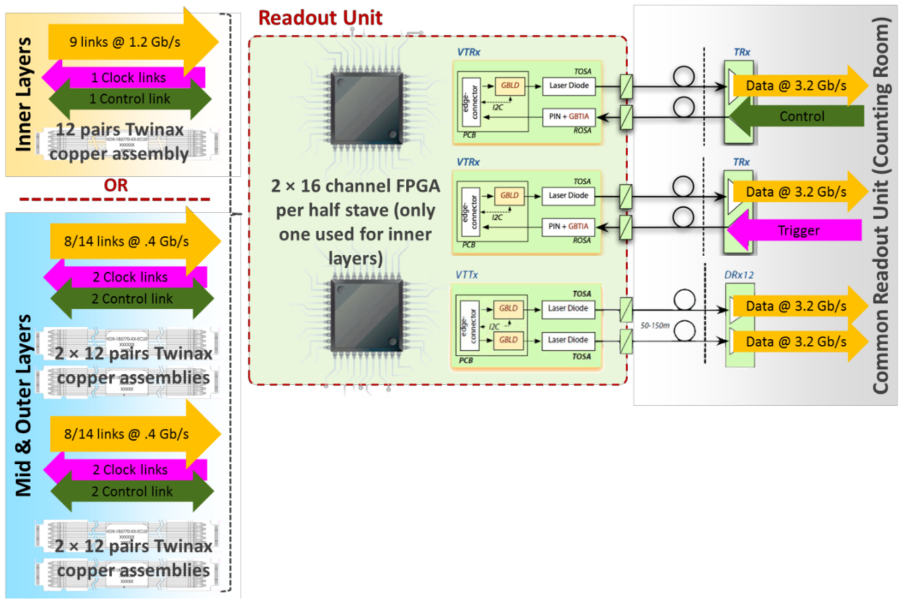


Figure 3.2: ITS Readout Unit implementation [24, Figure 37].

Processor, provides the trigger signal to the readout units which distribute the signal to the ALPIDE chips. Further links are connected to the Common Readout Unit, which is interfacing all readout units. This device handles the communication with the ALPIDE chips and receives the data for further processing.

3.2 ALPIDE Chip

The ALPIDE chip was developed for the ALICE ITS upgrade at CERN. The ALPIDE is a particle detector based on monolithic active pixel sensor, and is implemented in 180 nm CMOS technology process that is enhanced for imaging sensors [6]. The chip contains a matrix of $512x1024 pixels$, which individually can detect an ionization energy over a common threshold level. A pixel hit can then be latched onto an in-pixel memory position if certain conditions are met. The information about pixel hits during a certain time period can be read out over a high-speed serial link.

3.2.1 Operation Modes

The ALPIDE can be configured to operate in three different modes; Inner Barrel Chip, Outer Barrel Master and Outer Barrel Slave. These modes make the ALPIDE work in separate ways related to where they will be placed in the ALICE ITS detector. However, in the proton CT, there may be a high rate of particle hits in all areas of the detector. Hence, it makes sense to focus on the Inner Barrel mode which provides the highest data transfer rate. A differential signaling 1.2 Gb/s serial data port is the Inner Barrel mode's data readout interface, where 8B/10B encoding makes the maximum data throughput 960 Mb/s. The control interface is also a differential signaling serial port, but at a significantly slower rate, 40 Mb/s. Consequently, its called the *slow* control interface. The interface supports bi-directional half-duplex signaling and multi-point connection. This means that several ALPIDE chips can be connected to the same control signal, but the chips require unique IDs to differentiate them from each other. The ID will also set the chip operation mode, as presented in Section 3.3.

3.3 ALPIDE Control Interface

The ALPIDE chips are integrated on staves. The Inner Barrel stave contains nine ALPIDE chips stitched together, adding up to a total length of 27 cm. The IDs are interfaced to the chips through a dedicated 7-bit chip port, CHIPID. The three MSBs are used to set the operating mode of the chip. All zeroes indicate Inner Barrel mode.

Table 3.1: ALPIDE interface ports relevant for Inner Barrel mode operation

Port	Description	Direction	Type
DCLK_P	Main clock	IN	Differential (M-LVDS[a])
DCLK_N	Main clock	IN	Differential (M-LVDS[a])
CHIPID [6:0]	Chip address and mode operation	IN	CMOS, internal pull-down
HSDATA_P	High-speed serial data	OUT	Differential (LVDS[b])
HSDATA_N	High-speed serial data	OUT	Differential (LVDS[b])
DCTRL_P	Control interface	IN/OUT	Differential (M-LVDS[a])
DCTRL_N	Control interface	IN/OUT	Differential (M-LVDS[a])
RST_N	Global chip reset	IN	CMOS, internal pull-up
POR_DIS_N	Power-On Reset disable	IN	CMOS, internal pull-up

[a] Designed concerning standard TIA/EIA-899 Electrical Characteristics of Multipoint-LVDS, but are not standard regarding the range of input common voltage.
[b] Not standard regarding the range of input common voltage.

The four LSBs provide the chip with an address ID unique to that stave. The IDs are used both to control a particular chip and to identify the data from a specific chip.

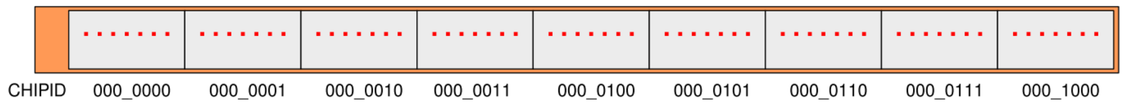


Figure 3.3: The chip ID scheme for Inner Barrel Mode [6].

The control bus, DCTRL, enables read and write access to all ALPIDE control and status registers, as well as providing a command distribution channel. The ALPIDE chip contains a large set of local registers for configuration of operation. These registers control everything from PLL settings to data readout speed. They also serve as the configuration registers for multiple test features, e.g. test vectors to be transmitted over the high-speed serial link. Communication via the control bus is accomplished by sending pre-defined opcodes on the bus. The ALPIDE chips interpret the received opcode and then perform the requested action. Some opcodes are tailored for a single chip, e.g. unicast read, which is used to read the data stored in a register on a specified chip. Other opcodes, like multicast write, involve communication with all the chips connected to the same control port. There are also broadcast commands that trigger actions from all the chips but, in contrast to multicast commands, are without subsequent communication. One special broadcast command is the TRIGGER command. The opcodes[1] for this command are decoded with much lower latency in the internal decoding by the ALPIDE chips than the other broadcast commands. The list of the predefined opcodes is available in the ALPIDE Operations Manual [6, Table 3.3].

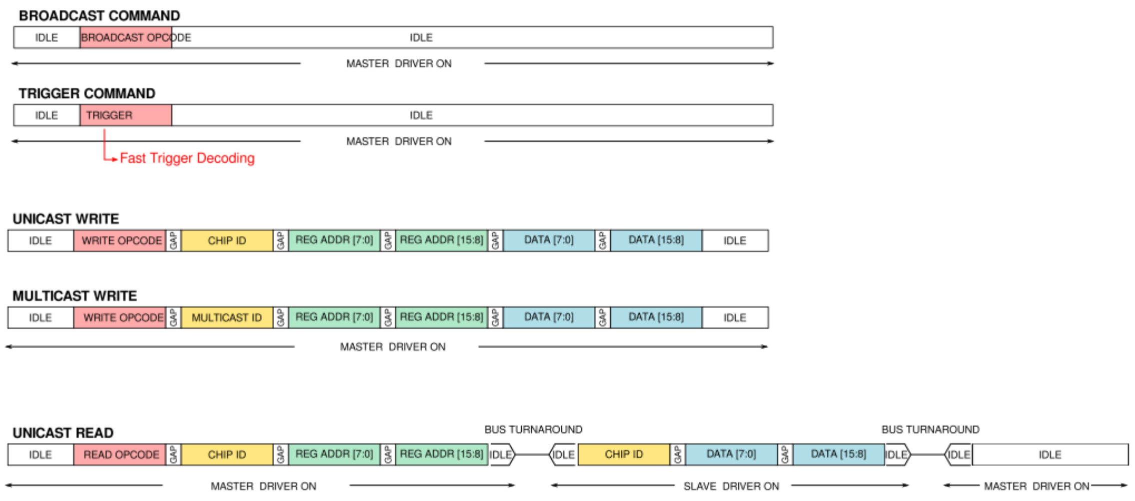


Figure 3.4: Valid transaction formats [6].

The various opcodes require different transaction formats that fit the purpose of the opcode. E.g., if a single chip is to send data from one of its registers, the unique chip ID must be provided so that the intended chip can respond. Figure 3.4 illustrates the different transaction formats required for control bus communication.

3.4 ALPIDE Data Interface

The data interface when the chip is in the Inner Barrel mode is utilized by the differential serial port (HSDATA_P and HSDATA_N). By default, the data is encoded by the 8B/10B algorithm. 8B/10B is a coding method to achieve DC-balance, which is a requirement of some transmission media for the receiver to reliably distinguish between a high or low bit [25, 26, Chapter 2]. The ALPIDE will transmit a comma word when not transmitting data. Since no clock is transferred along the data from the ALPIDE, the comma word can be used for clock recovery and to achieve byte synchronization [25], and even phase alignment as proposed in Section 4.6.

[1]There are four opcodes for the TRIGGER command which reflect the low latency nature of the opcode decoding. The decoding is based on the two LSBs of the command which is the same for all the four opcodes.

Data from the ALPIDE is formatted according to a predefined list of valid data words, as observed in Table 3.2. E.g., all data packets will begin with a 16-bit header that includes the chip ID and the value of the bunch crossing counter when the trigger corresponding to the data was received. An important data word is BUSY ON, which is transmitted on an assertion of the BUSY status. This is a state which is activated when the chip approaches saturation of its data processing capabilities.

Table 3.2: Valid data words [6, Table 3.37].

Data Word	**Length (bits)**	**Value (binary)**
IDLE	8	`1111_1111`
CHIP HEADER	16	`1010<chip_id[3:0]><BUNCH_COUNTER_FOR_FRAME[10:3]>`
CHIP TRAILER	8	`1011<readout_flags[3:0]`
CHIP EMPTY FRAME	16	`1110<chip_id[3:0]><BUNCH_COUNTER_FOR_FRAME[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_<bit_map[6:0]>`
BUSY ON	8	`1111_0001`
BUSY OFF	8	`1111_0000`

3.5 ALPIDE Operation

3.5.1 Front-end

Each sensor pixel in the ALPIDE matrix has a width of 29.24 µm and height of 26.80 µm. An analog front-end circuit transforms the physical effects of a particle hit into a digital signal which is sent to the digital section of the pixel. The signal may set the hit state register of the pixel if the corresponding STROBE signal (see Section 3.5.3) is asserted simultaneously. Each pixel feature three state registers or buffers referred to as a Multi-Event Buffer (MEB) that store hit information. This means that each pixel may store a maximum of three hits at the same time. Depending on the readout mode (see Section 3.5.3 and 3.5.4), the BUSY status is asserted based on the number of filled registers. These registers can also be programmed digitally for testing purposes.

Figure 3.5 shows a simplified pixel logic schematic, including a timing diagram of a pixel hit. The analog front-end electronics are always on and can create a shaped signal at any time. The shaped signal output from the amplifier is in the order of µs. These design choices were made to save power [24]. The shaping signal is fed to a comparator and will produce a logic 1 if it exceeds the threshold value. The

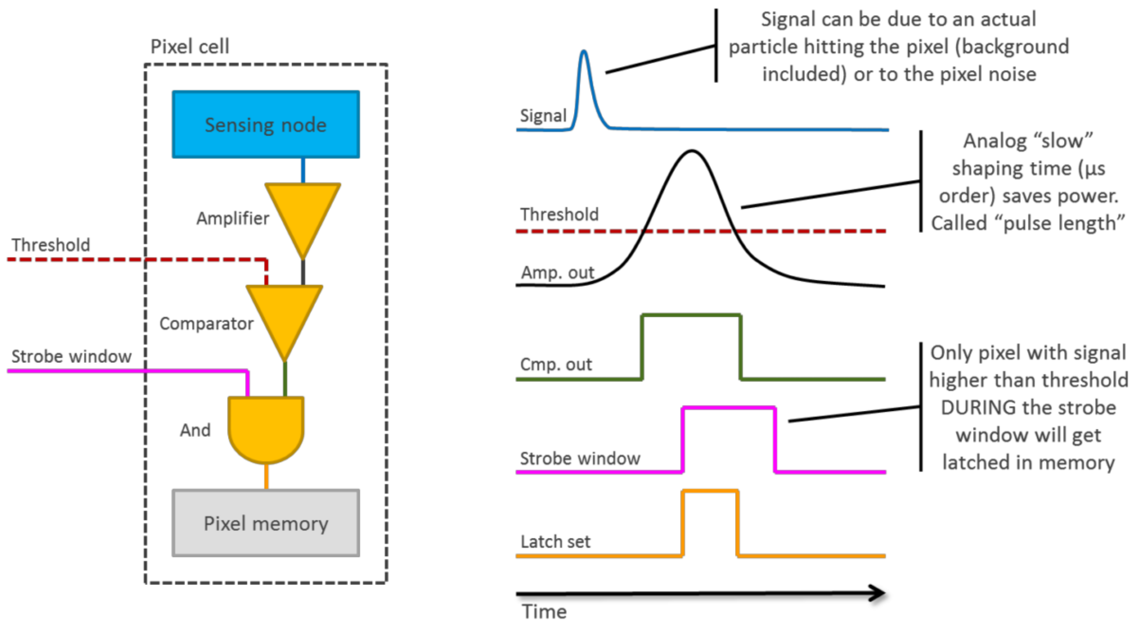

Pixel cell
Sensing node
Amplifier
Threshold
Comparator
Strobe window
And
Pixel memory
Signal can be due to an actual particle hitting the pixel (background included) or to the pixel noise
Signal
Analog "slow" shaping time (µs order) saves power. Called "pulse length"
Threshold
Amp. out
Cmp. out
Only pixel with signal higher than threshold DURING the strobe window will get latched in memory
Strobe window
Latch set
Time

Figure 3.5: Pixel logic schematic and timing diagram [24, Figure 15].

threshold value is variable and can be set via the control interface.[2] As discussed, if the comparator outputs a logic 1 while the strobe signal is asserted, the value is latched to the pixel MEB. Note that if two strobe windows are sufficiently close together, a shaping signal can extend between both windows, and as a result, a pixel hit may be registered twice.

3.5.2 Priority Encoders

The pixels are split into 32 regions containing 16 double columns of pixels. A single region is observed in Figure 3.6. The data is read out from 512 priority encoder blocks, one for each double-column.

3.5.3 Triggering and Framing

To capture information about particle hits, the ALPIDE will store the collection of all pixel MEBs at a particular time in a frame or snapshot. Following a TRIGGER command, the chip generates a frame and subsequently transmits data from the chip. The TRIGGER command may be received on the control interface or generated internally by a sequencer. The TRIGGER command is handled by the Framing

[2]Note that the threshold is set on a chip basis, and the threshold will naturally fluctuate between the pixels because of process variations, inter alia.

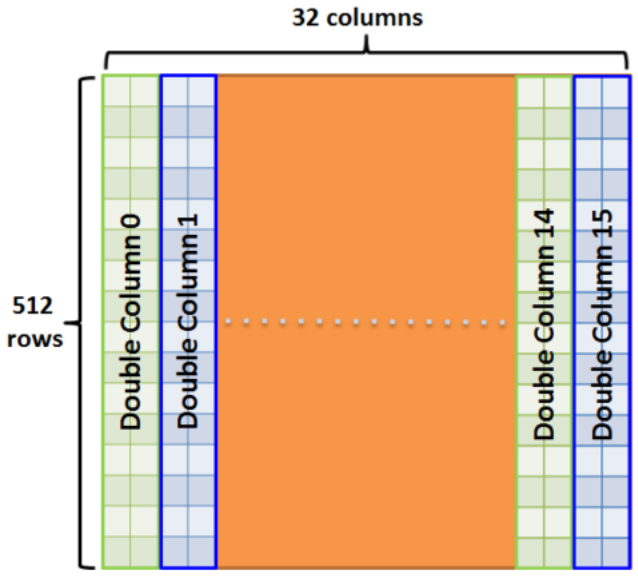


Figure 3.6: Double columns of data pixels in a single region [6, Figure 4.4].

and Management Unit (FROMU), which then generates the STROBE signal that is connected to the digital section of the pixels (see Section 3.5.1). The window in which STROBE is asserted is called the framing window. The MEMSEL signal is also generated by the FROMU and indicates which register bank to be read out. As seen in Figure 3.7 the data is readout through the Region Readout Units and then to the Top Readout Unit which transmits the data off the chip.

The length of the framing window can be customized and is subjectable to which readout mode the chip is configured in. In *triggered* mode, the intended framing window is relatively short (hundreds of ns), and the start of the window is typically controlled by an external trigger source. In continuous mode, the framing window is intended to be longer (a few µs). The framing window is meant to be periodically initiated, and the intervals have a duration equal to the period between two consecutive framing windows. The internal sequencer can generate the trigger, with register control of strobe and gap length.

The two modes are in principle similar, except from the logic handling. *Triggered* mode will prioritize events that are already stored in the MEBs, and a TRIGGER command received when the buffers are full will not generate a new framing window. *Continuous* mode, however, will prioritize newly received frame requests over already stored data. In this mode, there must always be one buffer free of data.

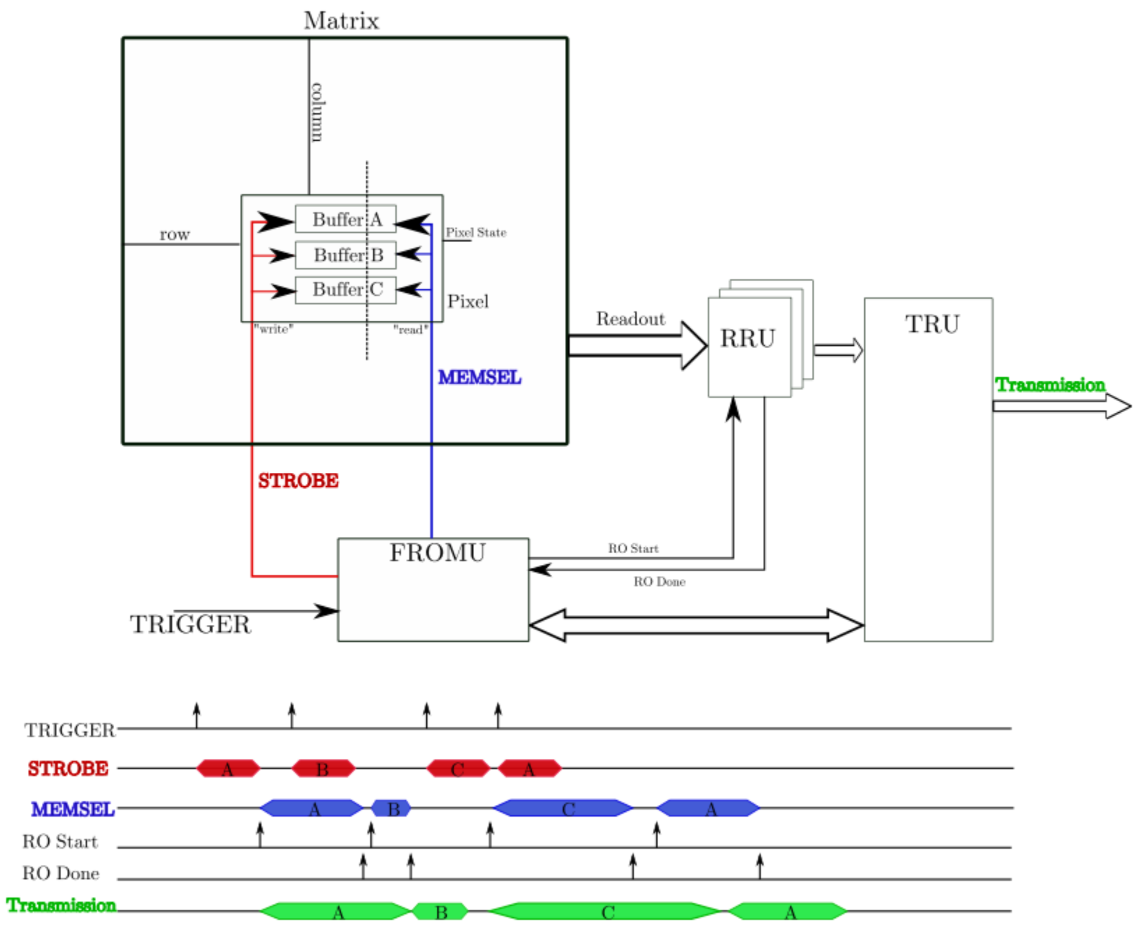


Figure 3.7: Pixel MEB Management Scheme [6, Figure 3.7].

3.5.4 BUSY State

It takes time to read out the data from the MEBs, and when the chip is near saturation of its data processing capabilities, the BUSY state is asserted. In an Inner Barrel configuration, the communication of the BUSY state to the outside world is done through the data link. In *triggered* mode, the chip is BUSY at the beginning of a framing window that addresses the last free buffer, when there is no more available storage for another framing window. All further TRIGGERs will be ignored until the BUSY state is over. The BUSY is done as soon as one of the buffers are completely read out. In *continuous* mode, BUSY is asserted at the beginning of a framing window targeting the second buffer, when there is only one free buffer left. A TRIGGER received in the BUSY state will initiate a framing window if the last buffer is available, whilst interrupting the ongoing readout to ensure that there is always a free buffer. This is signaled to the outside world with a flag on the data chip trailer. BUSY is deactivated once two buffers are available.

The BUSY ON data word transferred while the chip is in the BUSY state does not contain any information about what caused the situation to occur. Note that a BUSY state may also occur when internal FIFOs in the readout units of the chip are saturated.

3.6 ITS Readout Unit

The ITS readout unit provides 32 high-speed I/Os, and is interfacing a various amount of chips depending on which detector layer it is affiliated with. For the inner layers, one readout unit interfaces only one stave, and must therefore handle just nine data pairs, one clock, and one control line. A middle layer and an outer layer stave consists of 8 and 14 modules, respectively. A module consists of two master chips which interfaces the readout unit. Thus, for the outer layers, the readout units must handle 28 data pairs. For the middle layers the readout unit must handle 16 data pairs. This is illustrated in Figure 3.2.

The readout unit itself will consist of a PCB with a state-of-the-art FPGA chip that handles the majority of tasks in the programmable internal circuits. Data and control connections to the Central Trigger Processor and the Common Readout Unit are implemented optically with the CERN-developed GBT link.

3.6.1 Modules Overview

The readout unit consists of three main modules that are interfaced to the ALPIDE chips; the control module, the data readout module, and the voltage module.

3.6.2 Control Module

In order to communicate with the ALPIDE chips, the readout unit requires a control module that performs operations on the stave-shared communication line. The control module[3] handles all the slow control operations discussed in Section 3.3 and ensures that the bus turnaround operation is performed correctly. The module is operating on the same 40 MHz clock that is provided to the ALPIDE chips. The slow control output is serialized by this clock. However, to avoid metastability issues, the slow control data from the ALPIDE is sampled with a 160 MHz clock. The input is then deserialized by choosing a fixed point in the 160 MHz clock period. The phase

[3]The pCT readout unit design can inherit the module developed by Matteo Lupi (matteo.lupi@cern.ch) at CERN.

position is set by a 2-bit register and is not obtained automatically. Hence, there are four positions of sample points to choose from. This method is tested successfully at CERN [27].

To avoid sending TRIGGER commands to the ALPIDEs when a chip is busy, the module monitors a busy signal. This busy signal may be connected to the output of a protocol checker (see Section 3.6.3). Note that the effect of this feature is that no other chips on the stave will receive a TRIGGER command if a single chip reports the busy state. However, the feature can be disabled by setting a register.

The communication to the ALPIDE chips is mainly controlled by the readout unit's bus interface and the control module registers. When writing to the CTRL register, the communication state machine is initialized and is employing the data stored in the DATA and OPCODE registers to determine which operation to perform. However, operations may also be initialized by asserting the TRIGGER or PULSE input signals. This means that these commands may be performed without interacting with the bus interface, e.g. by an external device that controls the TRIGGER operation for all the readout units (see Section 4.10.1). Note that the control module does *not* reject these external commands whenever a busy state has been detected.

3.6.3 Data Module

The data recovery for the ITS must handle the different operation modes of the detector chips in the different layers. This complicates the use of built-in FPGA transceivers to achieve byte synchronization because the transceivers rely on a given data rate. However, it is still possible to use the transceiver to sample each data bit. The ALPIDE has three data rates; 1.2 Gb/s, 600 Mb/s and 400 Mb/s. Fortunately, these rates are fractions of each other. The transceiver can sample the data at a fixed rate of 1.2 Gb/s. For the lower speed rates, this just means oversampling. For given settings, the data can be downsampled to reflect the actual data rate. The downsampling module excludes the bits that are sampled multiple times by the transceiver, and creates a 10-bit word. It is fairly trivial to activate and deactivate this kind of module by using a configuration register.

This design, shown in Figure 3.8, was developed by engineers at CERN. Comma detection and alignment, as well as 8B10B decoding, are realized by RTL modules. The design also involves the data chain until read out over a USB interface.

Protocol Checker

The protocol checker module is not included in the block diagram but is connected to the data output from the 8B10B decoder. Its purpose is to analyze the data transmitted from the ALPIDE and detect errors in the protocol. It will assert various

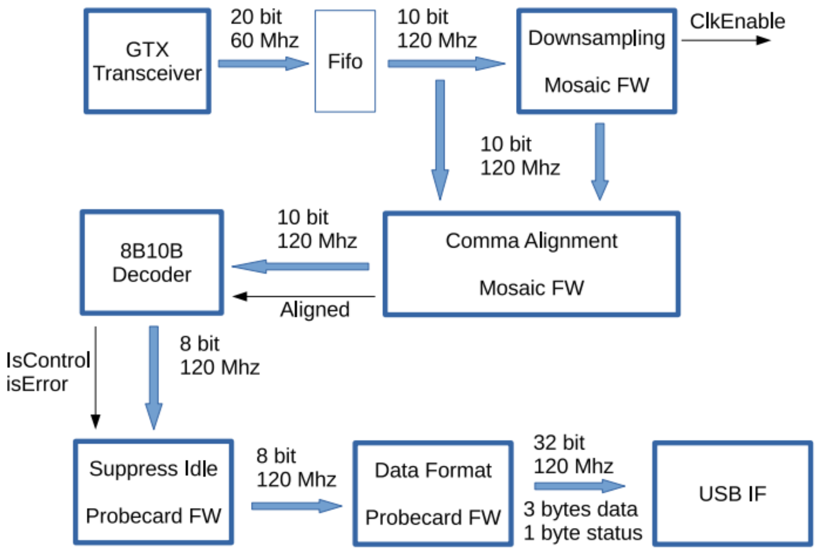


Figure 3.8: Block diagram of ITS readout data recovery system [27].

signals under certain conditions, for example when a chip is busy or a region is empty. The protocol checker module is described more thoroughly in Section 4.6.4. This module was later modified for pCT to be used for testing of test vectors (see Section 5.3.1), and then to aid with filtering, tagging, and buffer data (see Section 5.8).

3.6.4 Voltage Module

The purpose of the voltage module is to enable the voltage regulators that provide the ALPIDE with its required voltage and current. It is reused during the development phase for the pCT, but must be modified according to requirements discussed in Section 4.11.

CHAPTER 4

pCT Readout System

This chapter offers a discussion of the pCT system requirements and proposes a readout system design. Further, a readout unit and the different stages of the readout process are discussed in detail. The following sections present clock data recovery methods. This includes Section 4.5.2 which describes a data recovery method utilizing a transceiver for sampling data, comma alignment and 8B/10B decoding. This design is implemented and tested as discussed in Section 5.4. Section 4.6 portrays an automated approach to achieving phase and word alignment. This design is fully implemented and tested in Section 5.6 and 5.7.

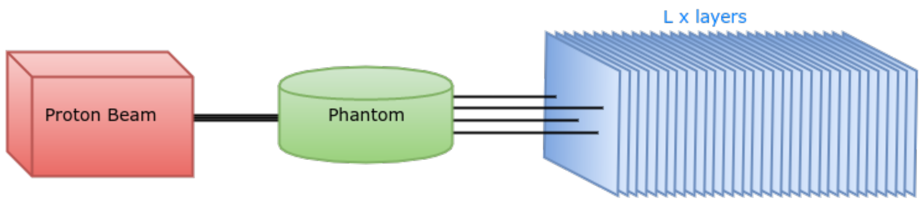
Subsequently Section 4.7 is about data filtering, tagging and buffering and presents a design proposal that will handle these tasks. Parts of this design are realized and simulated in Section 5.8. However, further analysis is needed to develop a complete design.

The following section offers a discussion on communication between modules. This involves the choice of bus interface. Arguments are made as to which bus interface best suits the pCT design. The implementation and verification is described in Section 5.5.

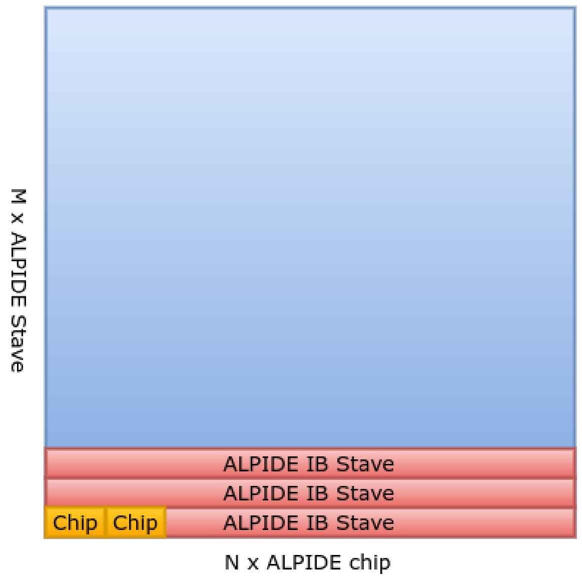
The next part presents a discussion of how the pCT system should operate to accommodate the timing requirements of the scanner. Finally, the system's power control and monitoring features are briefly drafted. The last two sections are presented without specific design implementations.

4.1 pCT Requirements

The pCT scanner will utilize the ALPIDE chips in a layer-like fashion, but the details were not completely specified during the work of this thesis. The final specifications of the detector layout will depend on data simulations that will calculate the size needed to create a sufficiently detailed picture of the phantom. The leading idea is that a detector with 40 layers with a 4 mm gap, consisting of aluminum and air, will be adequate to fulfill the requirements. One also expects that the cost of ALPIDE production will limit the detector size. A visualization of a pCT setup is shown in Figure 4.1a.



(a) Visualization of a pCT scanner with layers of detector chips.



(b) Layer layout.

Figure 4.1: Assumed detector layout. Size and distance are arbitrarily chosen and not to scale.

However, it can temporarily be assumed that a single layer will be constructed like a square made up of multiple Inner Barrel ALPIDE staves. As mentioned in Section 3.3, these staves contain nine ALPIDE chips that are bonded together. A layer will then require eighteen staves to create a square. Further, we assume that all layers will be equal in size and that there will be between 20 and 40 layers. We must, however, keep in mind that the specifications may change in the future and that the proposed system design must be modular in order to accommodate for updates. Figure 4.1b shows the tentative layout of a layer.

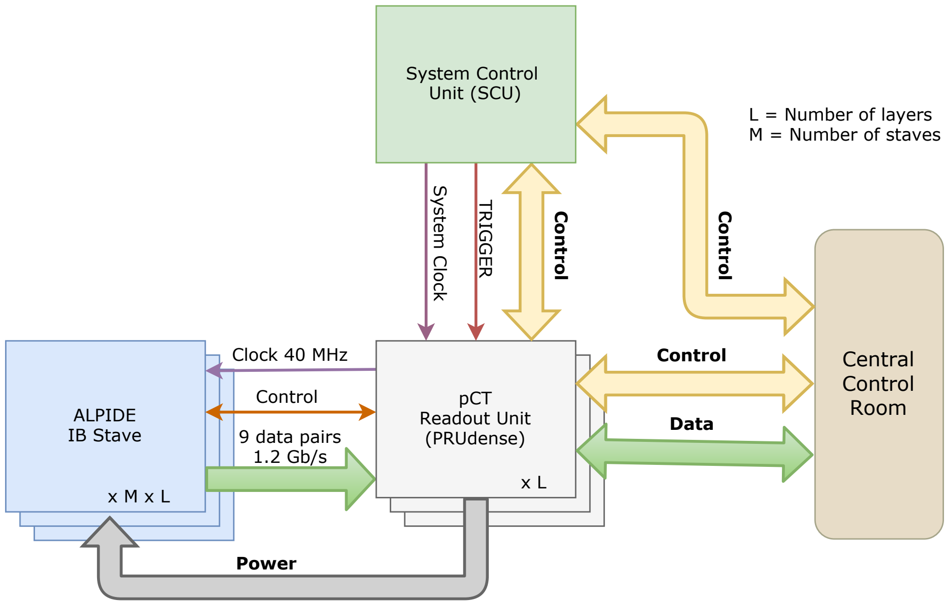


Figure 4.2: Block diagram of the proposed pCT Readout System.

The block diagram of the proposed pCT readout system (see Figure 4.2) consists of two main units; the readout unit (PRUdense) and the System Control Unit (SCU). PRUdense is interfacing the ALPIDE chips' data channel and controls the communication channels. The SCU is responsible for distributing the system clock and monitoring of the status of the staves and the readout units. Additionally, the SCU may also distribute a trigger signal to the readout units. Both main system units can be controlled from the control room.

The design of the pCT scanner involves one major advantage compared to the design of the ITS: All layers will consist of inner barrel staves, and hence, the readout units need only handle a single data rate of 1.2 Gb/s. This simplifies the clock recovery and data sampling and also reduces the need for different configurations of the readout units.

The proposed layer size incorporates 162 ALPIDE chips that each sends data over its own LVDS line. However, the control and clock lines are shared between the chips on one stave. To achieve regularity (see Section H.2.2), it will be beneficial to create identical readout units. A further aim would be to design readout units that can handle one whole layer each. Though not critical, this would significantly reduce cost and complexity of the remaining system. The major point is that the readout units handle equal amounts of data links and that the origin of the data is recoverable when the data is transmitted to the data center.

The pCT scanner will in principle require less time than ITS to collect enough data to achieve the desired results, and a guessed estimate is between 1 and 10 seconds.

This is beneficial when considering data transfer from the readout units as it may enable buffering data before offloading. Another benefit is that critical chip failure, like single event latch-up (SEL), may not cause a significant amount of data loss, and reconfiguration of the chip can wait until the next data-taking procedure. Although, this must be verified by simulations.

A further difference between the ITS and pCT readout systems is the expected radiation levels in the environment in which the readout units are located. The assumption is that the pCT readout units will be placed with a certain distance to the proton beam, and that the baseline radiation will be low enough to avoid adverse interaction with the electronics. This must be verified by simulations and measurements, which falls beyond the scope of this thesis. Regardless, one must evaluate taking precautions in the design process and choosing alternatives that historically reduces the single event upset (SEU) rate.

In the ITS setup, the distance between detector chips and readout units requires data cables of 5 meters [28]. It is assumed that this will be reduced to between 1 and 2 meters for the pCT setup. This reduction is relevant for data readout testing where different possible cable lengths must be tested.

4.2 pCT Readout Unit (PRUdense)

The readout unit will interface an unknown number of ALPIDE sensor chips. The unit is named based on the density of ALPIDE chips it is interfaced to. It will be responsible for the following tasks:

- Receive and decode multiple high-speed data streams (1.2 Gb/s)
- Handle communication with multiple ALPIDE staves (40 Mb/s)
- Filter and distribute TRIGGER signals to the ALPIDE chips
- Combine multiple data streams into a convenient data format
- Tag the data streams with time and detector placement
- Detect BUSY ALPIDE chip
- Perform error checking
- Transmit data to central
- Monitor and control voltage and current
- Reconfigure ALPIDE chip in the event of SEL, or similar critical error

Potentially, if found beneficial:

- Buffer data on onboard RAM

However, it is beyond the scope of this thesis to cover all of these features in detail. Emphasis is put on finding the most efficient clock and data recovery method, as well as outlining all of the readout stages discussed in Section 4.4. The following sections will discuss and examine the different design and implementation choices for the readout unit.

4.2.1 Top Level Design

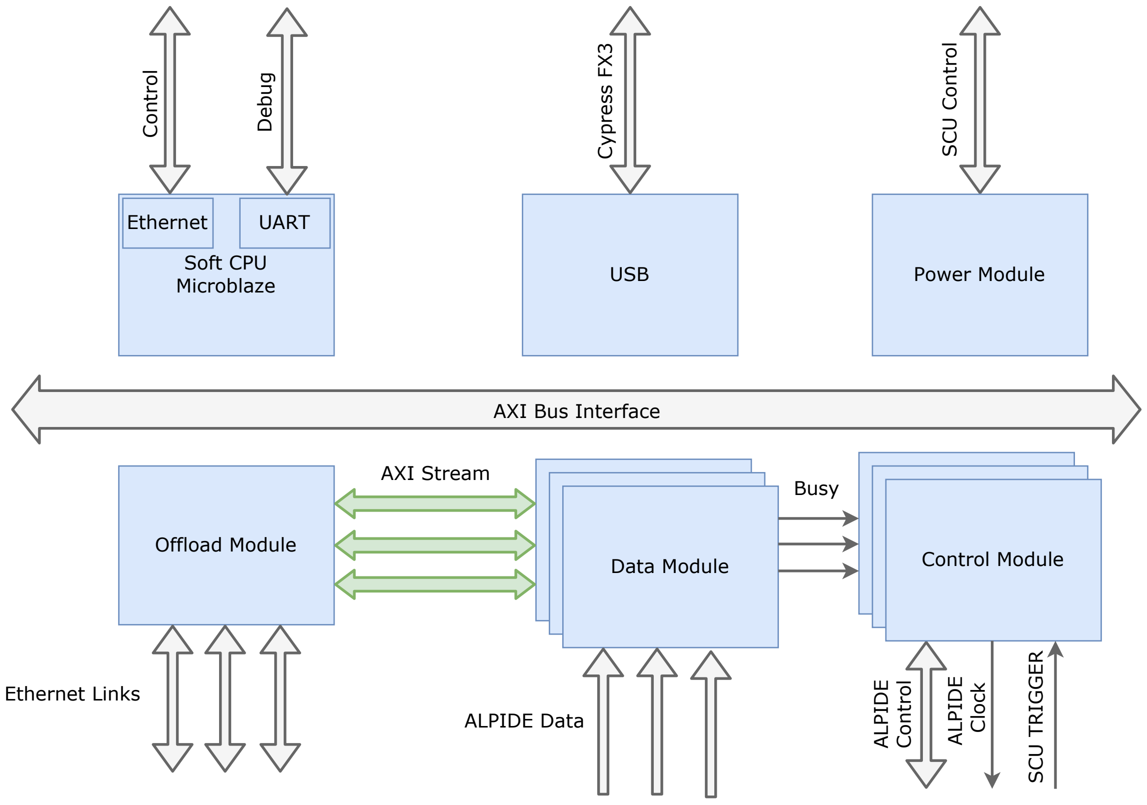


Figure 4.3: Top level design of the readout unit (PRUdense).

The top level design in Figure 4.3 shows the readout unit modules and how they are connected to the AXI bus interface. Bus interface discussion is covered in Section 4.9.

USB is used for communication and debugging until the soft processing system is implemented and tested. The UART channel is intended to be used as a debug channel and can indirectly communicate with the AXI bus via software running on the CPU. Ultimately, the processing system's main communication channel will be via one of the Ethernet links.

There are an equal number of data modules as there are ALPIDE chips connected to the readout unit. This module's task is to recover and sample the data correctly (see Section 4.5 and 4.6), filter redundant data words (see Section 4.7.1), and tag the data and buffer it (see Section 4.7.3), until it is read out on the AXI Stream interface to the offload module. Each module directly interface one ALPIDE data channel. The data module can also analyze the data and alert the corresponding stave's control module if the ALPIDE is busy.

The control modules are controlling the ALPIDE control channel. Each module control one stave of ALPIDE chips. The module is also interfacing the SCU's triggering signal and must respond to an assertion of that.

The power module (see Section 4.11) is controlling the voltage and current regulators on the readout board, and monitors the power drawn by the ALPIDE chips. The module is interfacing the AXI bus and is mainly controlled and monitored by the SCU. Whether the SCU is interfacing the module via the AXI bus or via another interface is yet to be determined.

4.3 Electronics

Reliable handling of many high-speed links will require parallel processing to avoid bottle-necking. Developing a custom ASIC would provide the performance needed, but this usually requires a high production quantity to be justified economically [7, Chapter 13]. It is, therefore, more common to use FPGAs in highly technical demanding scientific projects where the non-recurring engineering cost must be kept low, and design resources and time are scarce. An FPGA also provides a higher degree of flexibility in the design process, and this is especially useful in the development in the pCT due to the unknown or unclear specifications.

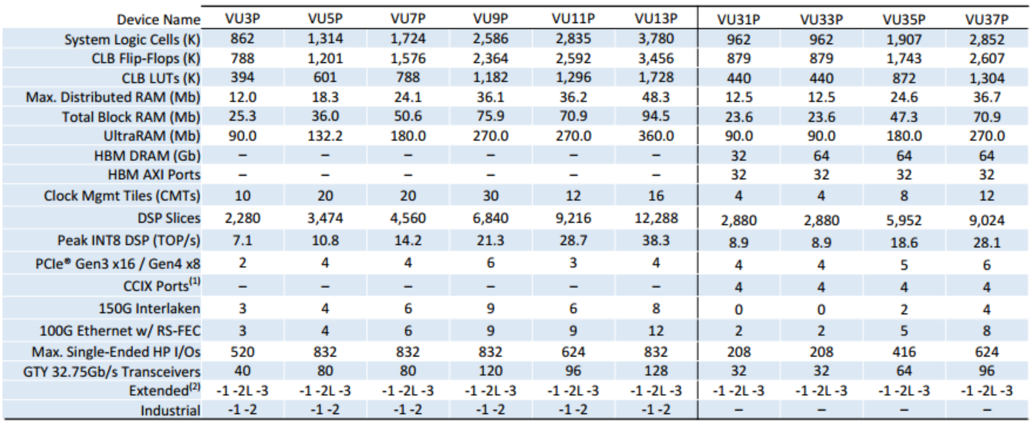
Two companies share the majority of the commercial FPGA market; Xilinx and Altera.[1] Both companies provide a variety of SRAM-based FPGA products with and without System-On-Chip (SoC) features. The chips vary in the number of programmable logic blocks, on-chip RAM blocks, high-speed transceivers, and more. Xilinx claims that their products are designed to have an inherently low susceptibility to SEUs [29]. Although it is assumed that the baseline radiation level will be low in the pCT readout environment, it will be higher than the natural background levels. Thus, a greater degree of resistance to radiation-induced upsets may be beneficial.

Changing FPGA provider, and thus tools, design flow, and much more, during a design process, is usually time-consuming and must be thoroughly evaluated. The design team for the ITS upgrade chose Xilinx chips as the design target early in the design process. Thus, large parts of the development that the pCT firmware will be based on are targeting Xilinx technology. Finding a Xilinx chip that fits the pCT requirements will be highly beneficial.

The newest Xilinx product family is called UltraScale+™. The Virtex devices are the largest of the product family. They provide the highest quantity of programmable

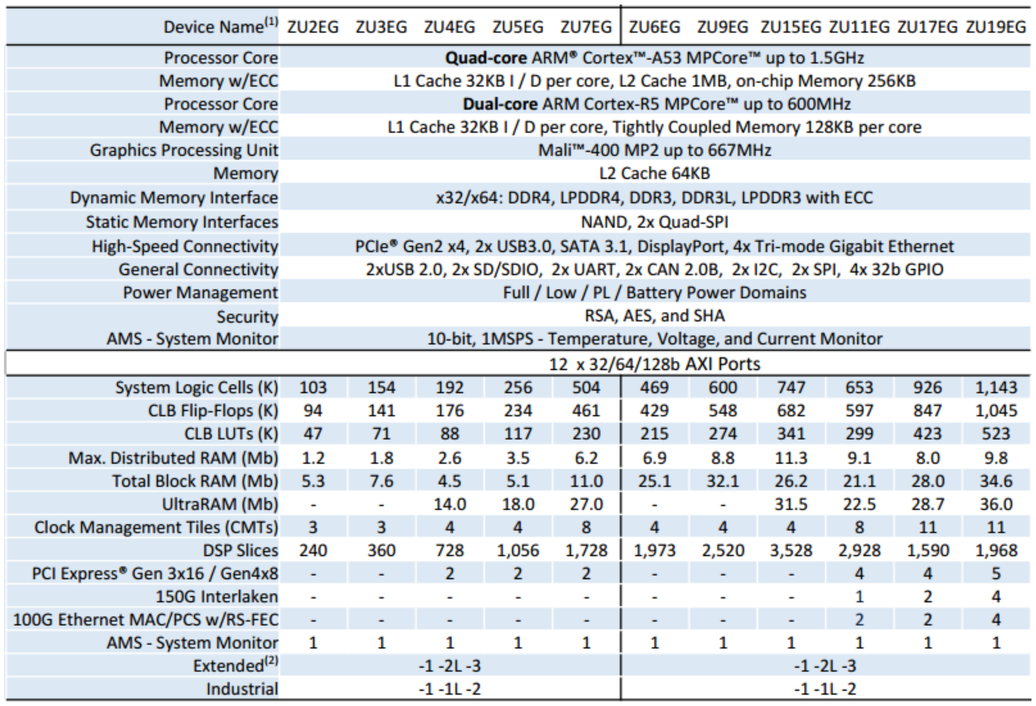
[1]Altera was recently acquired by Intel.

logic resources, but do not provide SoC features. The Zync devices, however, provide a powerful hard processor system, but significantly lower programmable resources. Figure 4.4a and 4.4b show product overviews of Virtex and Zync devices.



Device Name	VU3P	VU5P	VU7P	VU9P	VU11P	VU13P	VU31P	VU33P	VU35P	VU37P
System Logic Cells (K)	862	1,314	1,724	2,586	2,835	3,780	962	962	1,907	2,852
CLB Flip-Flops (K)	788	1,201	1,576	2,364	2,592	3,456	879	879	1,743	2,607
CLB LUTs (K)	394	601	788	1,182	1,296	1,728	440	440	872	1,304
Max. Distributed RAM (Mb)	12.0	18.3	24.1	36.1	36.2	48.3	12.5	12.5	24.6	36.7
Total Block RAM (Mb)	25.3	36.0	50.6	75.9	70.9	94.5	23.6	23.6	47.3	70.9
UltraRAM (Mb)	90.0	132.2	180.0	270.0	270.0	360.0	90.0	90.0	180.0	270.0
HBM DRAM (Gb)	–	–	–	–	–	–	32	64	64	64
HBM AXI Ports	–	–	–	–	–	–	32	32	32	32
Clock Mgmt Tiles (CMTs)	10	20	20	30	12	16	4	4	8	12
DSP Slices	2,280	3,474	4,560	6,840	9,216	12,288	2,880	2,880	5,952	9,024
Peak INT8 DSP (TOP/s)	7.1	10.8	14.2	21.3	28.7	38.3	8.9	8.9	18.6	28.1
PCIe® Gen3 x16 / Gen4 x8	2	4	4	6	3	4	4	4	5	6
CCIX Ports(1)	–	–	–	–	–	–	4	4	4	4
150G Interlaken	3	4	6	9	6	8	0	0	2	4
100G Ethernet w/ RS-FEC	3	4	6	9	9	12	2	2	5	8
Max. Single-Ended HP I/Os	520	832	832	832	624	832	208	208	416	624
GTY 32.75Gb/s Transceivers	40	80	80	120	96	128	32	32	64	96
Extended(2)	-1 -2L -3	-1 -2L -3	-1 -2L -3	-1 -2L -3	-1 -2L -3	-1 -2L -3	-1 -2L -3	-1 -2L -3	-1 -2L -3	-1 -2L -3
Industrial	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	–	–	–	–

(a) Virtex Ultrascale+™product overview [30].



Device Name(1)	ZU2EG	ZU3EG	ZU4EG	ZU5EG	ZU7EG	ZU6EG	ZU9EG	ZU15EG	ZU11EG	ZU17EG	ZU19EG
Processor Core	**Quad-core** ARM® Cortex™-A53 MPCore™ up to 1.5GHz										
Memory w/ECC	L1 Cache 32KB I / D per core, L2 Cache 1MB, on-chip Memory 256KB										
Processor Core	**Dual-core** ARM Cortex-R5 MPCore™ up to 600MHz										
Memory w/ECC	L1 Cache 32KB I / D per core, Tightly Coupled Memory 128KB per core										
Graphics Processing Unit	Mali™-400 MP2 up to 667MHz										
Memory	L2 Cache 64KB										
Dynamic Memory Interface	x32/x64: DDR4, LPDDR4, DDR3, DDR3L, LPDDR3 with ECC										
Static Memory Interfaces	NAND, 2x Quad-SPI										
High-Speed Connectivity	PCIe® Gen2 x4, 2x USB3.0, SATA 3.1, DisplayPort, 4x Tri-mode Gigabit Ethernet										
General Connectivity	2xUSB 2.0, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO										
Power Management	Full / Low / PL / Battery Power Domains										
Security	RSA, AES, and SHA										
AMS - System Monitor	10-bit, 1MSPS - Temperature, Voltage, and Current Monitor										
	12 x 32/64/128b AXI Ports										
System Logic Cells (K)	103	154	192	256	504	469	600	747	653	926	1,143
CLB Flip-Flops (K)	94	141	176	234	461	429	548	682	597	847	1,045
CLB LUTs (K)	47	71	88	117	230	215	274	341	299	423	523
Max. Distributed RAM (Mb)	1.2	1.8	2.6	3.5	6.2	6.9	8.8	11.3	9.1	8.0	9.8
Total Block RAM (Mb)	5.3	7.6	4.5	5.1	11.0	25.1	32.1	26.2	21.1	28.0	34.6
UltraRAM (Mb)	-	-	14.0	18.0	27.0	-	-	31.5	22.5	28.7	36.0
Clock Management Tiles (CMTs)	3	3	4	4	8	4	4	4	8	11	11
DSP Slices	240	360	728	1,056	1,728	1,973	2,520	3,528	2,928	1,590	1,968
PCI Express® Gen 3x16 / Gen4x8	-	-	2	2	2	-	-	-	4	4	5
150G Interlaken	-	-	-	-	-	-	-	-	1	2	4
100G Ethernet MAC/PCS w/RS-FEC	-	-	-	-	-	-	-	-	2	2	4
AMS - System Monitor	1	1	1	1	1	1	1	1	1	1	1
Extended(2)			-1 -2L -3					-1 -2L -3			
Industrial			-1 -1L -2					-1 -1L -2			

(b) Zynz Ultrascale+™product overview [31].

Figure 4.4: Comparison of Virtex and Zync products.

While a hard processing system may be convenient for handling the output data stream, the loss of both programmable logic and on-chip memory is significant. A soft core processor will also use a quite high proportion of the programmable logic, so the trade-off between these two models is significant. According to Xilinx, the Microblaze system will consume between 600 and 1100 LUTs and 300 to 1300 flipflops depending on the configuration of the system [32]. The default configuration consumes 619 LUTs and 272 flipflops on the Virtex Ultrascale FPGA. Comparing these numbers to the loss of resources when employing the hard processing system on the Zync chips tells us that the soft processing system leaves far more programmable resources available to be used elsewhere.

Note that there is a finite number of transceivers available on the FPGAs. None of the alternatives have as many as 162 transceivers. Therefore, if the PRUdense is to be interfaced to that many ALPIDEs, it has to use regular I/O pins. This challenge is discussed further in Section 4.6.1.

During the start of the development and testing phases, the firmware will target a Xilinx Kintex 7. This chip has lower general performance, and a lower amount of I/O pins and transceivers than the UltraScale+ chips. However, this chip will be targeted to reuse a test board developed by the ITS development team, and thus avoid losing time by developing a new test board. The UltraScale+ chips were not available at the time the development of the test board was started. The Kintex 7, however, has high bandwidth I/O pins, high-performance transceivers and enough programmable logic to develop and evaluate parts of the system. The Multi-Gigabit Transceiver (MGT) supports data rates between 500 Mb/s and 12.5 Gb/s.

4.3.1 ITS RUv0a Test Board

The main features of the RUv0a test board[2] are as follows:

Kintex 7 FPGA: The specific chip has a speed grade of 2, and it follows that it has I/O pins with 1.25 Gb/s bandwidth capabilities. It also has a number of MGT.

JTAG: Enables FPGA programming and debugging.

Cypress FX3: This chip enables USB communication between the board and an external computer.

Samtec FireFly: Multiple FireFly connections are connected to various inputs on the FPGA. The connections are organized to be used for the different ALPIDE configurations. The pCT project has employed two of these connections where the second connector is lined up with the FPGA MGTs and the fifth that is lined up with the ordinary I/Os. For the fifth connector, the data channel is differentially and commonly terminated, as well as DC-blocked with a capacitor. This termination network is illustrated in the schematic in Figure 4.5.

160 MHz Oscillator: Provides a clock signal that can be used to create all necessary clock signals.

[2]Developed at CERN by K. Sielewicz [33].

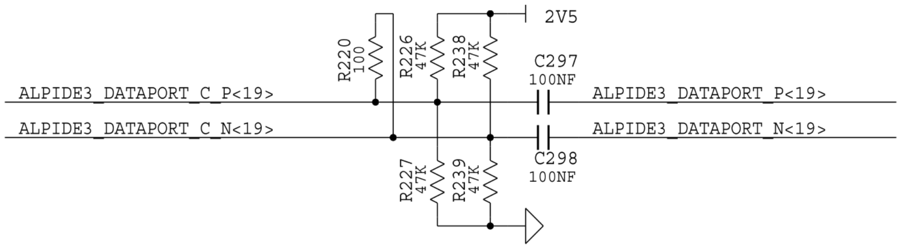


Figure 4.5: Termination network for the data channel connected to regular FPGA I/Os [33].

SMA Connectors: Enables connection of external clock signals and also transmission of created clock signals out of the board to be analyzed.

Voltage regulators: Provides power to the ALPIDE chips. Are controllable by the FPGA.

4.3.2 ALPIDE Carrier Card and Adaptor

During the early stages of development, the ALPIDE chips are only physically available attached to a carrier card.[3] The carrier card couples the pins of the ALPIDE chip to a PCIe connector. The pixels themselves are protected with a glass shield.

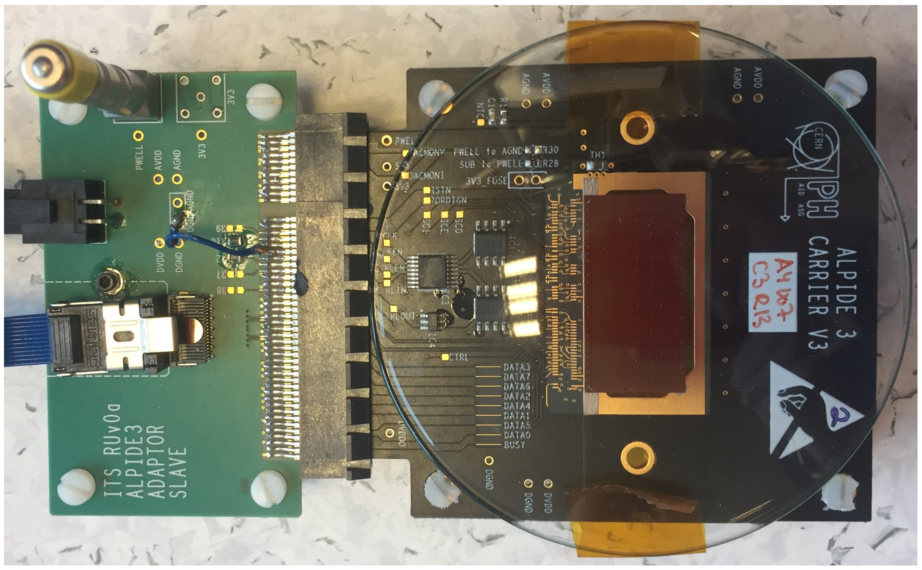


Figure 4.6: On the right: The ALPIDE carrier card. On the left: the ALPIDE adaptor slave.

[3]Developed at CERN by A. Sanchez [34].

To connect the ALPIDE to the RUV0a, the connection must be converted to Samtec FireFly. An ALPIDE adaptor slave performs this operation.[4] The card also supplies the power to the chip and serves as a back bias connection via the PWELL terminal.

Note that the high-speed data channel is neither differentially nor commonly terminated on any of the two cards [35, 34]. This is probably contributing to a poorer signal integrity than necessary.

4.3.3 Samtec FireFly

Samtec's FireFly system is a set of cables and connectors that promises high-speed data transfer with minimal signal integrity problems. The cable chosen for ITS has 12 pairs of twinax conductors and is usually used for short-range applications. Its attenuation characteristics are only specified for a cable length up to 1 meter. Thus, longer cables must be tested for performance. Samtec also provides cables with higher performance for a longer range that use the same connector, but the cost will probably disqualify them for utilization in the pCT project.

The reason for working with FireFly is that the ALPIDE staves will be delivered with a FireFly connector bonded to the chip ports. However, there are still discussions whether it may be possible to change this connector. One suggestion is to replace the FireFly system with flexible PCB. Evaluation of different types of connections falls beyond the scope of the thesis, and further discussion will assume that FireFly is used between the ALPIDEs and the readout unit.

4.4 Readout Process

The readout process consists of three main functions:

1. Sampling input data and filtering
   The stage where data from the 162 ALPIDE chips are sampled into the FPGA fabric. Data words that do not contain ALPIDE event data are discarded.
2. Data tagging and buffering
   The ALPIDE data contain information about which chip that is transferring the data, and when the strobe window trigger occurred. The chip ID, however, only indicates a specific chip on an implicit stave. When combining data from multiple staves, there is no identification of which stave the specific chip is bound to in the ALPIDE data format. The trigger time information is also lacking as it is based on the timing of the CERN LHC, and the maximum value is matching the duration of one LHC orbit [6]. More detailed tagging of the data is therefore required (see Section 4.7.2).

[4]Developed at CERN by A. Sanchez [35].

Buffering is required to adjust and convert the data of the input and the output stage. E.g. decoupling from a frame based data stream to a packet based data stream. Some memory will be required at this stage. This is discussed further in Section 4.7.3.

3. Data offloading
This stage involves retrieving data from multiple memory locations, combining the data and offload it via one or more data links. This is discussed in Section 4.8.

The first and second stage can be combined into one data module as depicted in the top level diagram of the readout unit (see Figure 4.3).

4.5 Clock and Data Recovery

Data transmitted from the sensor chips are serially transferred without an associated clock signal. The data rate and phase are decided by the internal ALPIDE PLL, and cannot be sampled at a fixed clock rate without adjusting the frequency and phase of the FPGA sampling clock without expecting bit errors. To sample the signal, the clock must be recovered from the incoming data [36]. Recovery is often achieved by phase aligning a reference clock with an approximately equal frequency to the transitions on the incoming data stream [37]. Using the phase aligned reference clock to sample the input stream is then possible, and this technique is called Clock Data Recovery (CDR). 8B/10B encoding ensures that there are never more than five ones or zeroes in a row and that there, in a string of 20 bits, never is a larger difference than two between the counts of ones and zeroes [25]. In other words, the code has a maximum disparity of two. This helps reduce glitches in clock recovery by providing a sufficient amount of state changes and produces a DC-balanced signal.

4.5.1 Transceivers

A common method of clock recovery is to detect the edges of the incoming data stream [36]. This is how the MGT, provided in Xilinx FPGA architectures, accomplish the task [38, Chapter 4]. Transceivers are, in FPGAs, I/O pins with extended features, specifically designed to handle both incoming and outgoing high-speed serial data. Some of the key Xilinx MGT receiver features are an analog front end, digital equalizing, a CDR block, byte and word alignment and 8B/10B decoding [38, Chapter 4]. The analog front end has configurable termination voltage and calibrated termination resistors. The equalizer includes several filters aimed at improving and repairing eventual distortion and attenuation that have affected the signal traveling through a channel. It is intended to attenuate the low-frequency components of a signal to match them to the high-frequency components that have been diluted in a lossy line [39]. The CDR samples both the edges and data and uses this to determine the phase of the incoming data and control phase interpolators [40]. The phase of the edge samples is locked to the transition region, and the data sampler is locked to the middle of the data eye. The phase interpolators are controlled by PLL-provided reference clocks.

4.5.2 Transceiver Only Design

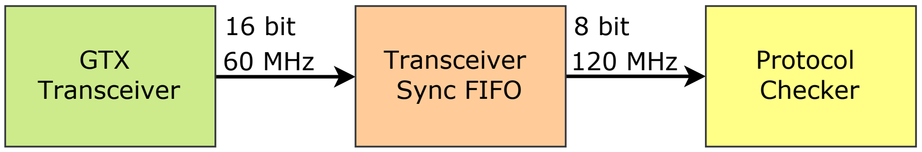


Figure 4.7: Block diagram of readout scheme utilizing transceiver for receiving data with a fixed rate.

The pCT readout system must only handle one data rate; the maximum 1.2 Gb/s. Consequently, byte synchronization is simplified compared to the ITS design, and can be achieved by only employing an MGT. When receiving data at a fixed rate, the MGT can perform both CDR, word alignment and 8B/10B decoding. The ITS data module design was, therefore, simplified and tested (see Section 5.4). The fabric modules of the ITS design (see Section 3.6.3) that perform these tasks were removed and replaced with the design in Figure 4.7. The design was realized by employing the 7 Series FPGAs Transceiver Wizard, which automates the task of creating HDL wrappers to configure the MGT. The MGT aligns the data to comma and decodes the resulting 20-bit word, to 16 bit. The FIFO converts the 16-bit word output from the transceiver to the 8-bit word which was originally encoded and transmitted from the ALPIDE. The FIFO is generated by the FIFO Generator and employs an independent clock block RAM with a depth of 16. This means that the read and write channels operate on differing clocks. The read enable signal provided to the FIFO is the complement of the FIFO empty signal, which prevents the FIFO from filling up. The protocol checker is reused from the ITS design, but is modified to allow for testing (see Section 5.3.1).

This CDR method, however, requires a transceiver for every single data link. As discussed in Section 4.3, the number of transceivers and PLLs is limited even in the largest FPGAs. Hence, the goal of handling up to 162 data links will not be met using this technique. Another constraint is that the transceivers may be required to transfer the output data to the common data central.

4.6 Automatic Phase and Word Alignment Design

4.6.1 I/O Features

A potential solution for achieving CDR without transceivers is to use the regular differential-compatible I/O pins and develop firmware that is implemented in programmable logic that controls the phase of the incoming data. This implementation, however, depends on the bandwidth of the I/Os, and that the frequency of the ALPIDE data is reliable and will not drift during readout. Regular I/O pins also lack equalizing filters. The transmission medium must therefore not introduce too much attenuation and distortion.

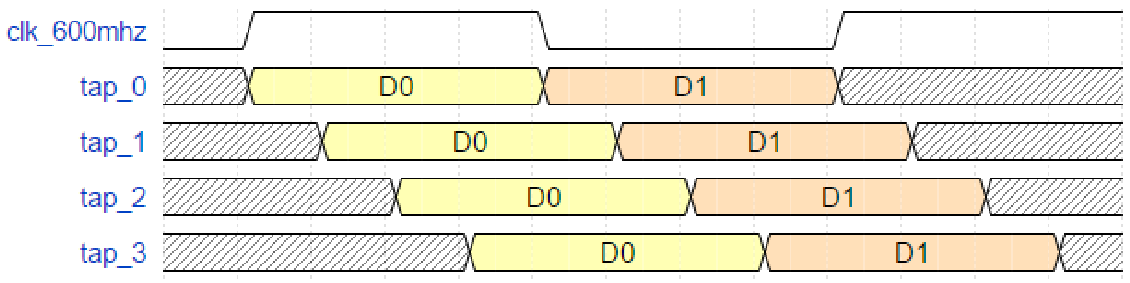


Figure 4.8: Illustration of the concept of data delay chain. The data in delay tap_2 will be sampled more reliably than the data in the other taps.

The main idea is that the frequency of the high-speed data is given at 1.2 GHz. This frequency is controlled by a PLL internal to the ALPIDE which multiplies the provided 40 MHz clock by 15 and serialize the data with DDR. The readout unit should be able to reproduce the exact same frequency by applying a PLL. Even so, the phase will not necessarily be equal, as it is influenced by a number of factors like the length of the data cable, temperature, and startup time. To adjust for the phase difference, the data can be delayed by fractions of the clock period. Figure 4.8 shows how the data may be delayed and how this influences sample reliability. On the falling and rising sample edges of the clock, the data of `tap_2` is right in the middle of two transitions. Hence, this delay tap will give the most reliable data sampling if the frequency is stable.

SelectIO Resources

The I/O pins on Xilinx devices generally have high bandwidth,[5] and also provide input delay resources which allow incoming signals to be delayed on an individual input pin basis [43]. Additionally, every single pin embodies SERDES features. These will be not only useful to create data words from the serial data, but also contain bitslip functionality which can be beneficial for word alignment. Figure 4.9 shows the schematic of a pin in a Xilinx 7 Series FPGA. UltraScale+ chips have the same I/O resources with some modifications [44]. These differences are discussed in Section 6.1.

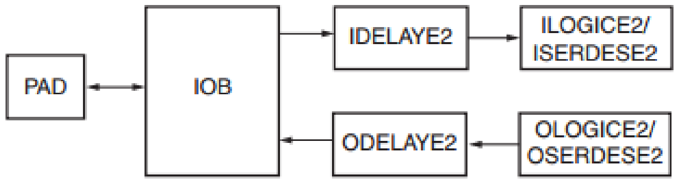


Figure 4.9: Xilinx 7 Series FPGA HP Bank I/O Tile [43, Figure 2-1].

The IDELAYE2 primitive is a 32 tap delay chain with a calibrated tap resolution. For a given reference clock, the delay taps are distributed over half the clock period and allow accurate placement of a sample point. The chain delay resolution is given by $1/(32 * 2 * F_{REF})$ [41, Table 29]. However, the individual tap delay values are averaged to this formula and will fluctuate. What makes the IDELAYE2 primitive so useful is that the delay tap value can be incremented or decremented dynamically without any glitch. Additionally, it is possible to directly load a certain delay tap value without incrementing through all the taps between the current and the desired tap.

The IDELAYE2 primitive is calibrated by IDELAYCTRL. The reference clock is fed to this primitive, and it will continuously calibrate the individual delay taps [43]. It not only controls the delay resolution from the reference clock, but will also reduce the effects of process variations, as well as temperature and voltage fluctuations.

The ISERDESE2 primitive converts the serial data into a word. Hence, it is a serial-to-parallel converter with customizable data width. The module also features bitslip functionality. By asserting the bitslip input signal, the ISERDESE2 will reorder the bits going into the FPGA fabric. This can be used for both word alignment and drift adjustment.

The amount of delay that will give the most reliable sample point must be determined by an automated process. This process should be a part of the initialization phase

[5]Kintex 7 HR and HP pins perform up to 1250 Mb/s for chips with speed grade 2 [41, Table 17]. All Virtex Ultrascale+ pins that support differential signaling have a performance of 1250 Mb/s independent of speed grade [42, Table 19].

of the pCT detector. The delay tap value will be locked the entire period while obtaining data after first achieving lock. The delay is only susceptible to change via continuous calibration by the IDELAYCTRL primitive. Confirmation that phase and word alignment has successfully been obtained by all data channels has to be checked by the central before enabling the radiation beam. The check must be done to prevent unnecessary radiation of the patient. Thus, each data channel must have an assigned data register for checking phase lock status, word alignment status as well as other metrics.

The following design employs Xilinx I/O primitives, and are set up as follows: The IDELAYCTRL reference clock is set to 600 MHz and controlled by a PLL. This gives an average tap delay resolution of 26 ps. ISERDESE2 is also provided with the 600 MHz clock, and a 120 MHz clock which serves as the divider clock. DDR-mode is enabled. IDELAYE2 is set in variable loadable mode. This enables both glitch-free incrementing of the delay taps, as well as manual loading of a delay tap value. All primitives are instantiated as part of the SelectIO Interface IP.

4.6.2 Phase Aligner Module

By enabling the ALPIDE readout whilst refraining from sending TRIGGER commands, the ALPIDE chips will send a constant stream of comma words. This can be exploited to achieve both phase and word alignment. The phase aligner module does the following:

1. Shifts 10 bits of data in from ISERDESE2. At all times hold 20 bits of data in an array.
2. Searches for 8B/10B encoded comma, either $0x17C$ or $0x283$ depending on the running disparity, positioned anywhere in the 20-bit string. This is done by employing a *for* loop which increments the array index and checks the resulting data string.

This operation can be done on any of the delay taps to confirm if the delay produces a valid sample point. To make sure a delay tap produces valid data samples, a given number of consecutive commas must be observed. In the module it is assumed that 255 consecutive commas are enough to verify that a delay tap gives a sample point that is within the valid range. However, to minimize bit errors it is necessary to obtain the sample point that is in the middle of the transitions. Therefore, every tap must be evaluated and the delay tap chosen must be the middle of the longest consecutive row of valid taps.

Figure 4.10 shows the state machine diagram of the module. It is assumed that the ALPIDE readout has been enabled, and is thus currently sending 8B/10B decoded comma words on the data line. When the run register is enabled, the state machine will loop until the IDELAYCTRL primitive is calibrated and locked. This step is added mainly as a last check that the calibration is achieved, and should have been accomplished within a couple of clock cycles of the reference clock. No timeout condition is added to this loop, as a fail to lock implies a fatal error that requires a reset of the I/O primitives. This task must therefore be handled by software in the power-on sequence (see 4.11.1).

Whenever the calibration check is done, `search_1` is initiated and the process starts to probe for commas in the incoming data stream. Note that the incoming data is independent of the current state of the state machine and is accomplished in a separate process. If no comma is found, the delay tap is incremented the next clock cycle. However, if a comma is found, a counter is incremented. When or if the counter reaches 255, the delay tap is considered valid. The number 255 is chosen based on the bit-length and the assumption that it is "good enough." Remember; the goal is to find the middle of the data eye, not to identify individual delay taps as usable. However, if the design produces incorrect results, one may consider increasing the requirement. The higher the number required, the larger the counter will be, and the more resources will be used per data channel.

As noted earlier, whenever no comma is detected, the current delay tap is considered invalid. A vector keeps track of the result of each delay tap. Zero equals valid, and one means invalid. When all delay taps have been controlled, the state machine uses the vector to choose the next operation. If no valid taps are identified, the next state is `tap_overflow_error`; an error signal is asserted, and the state machine is returned to idle. If some valid taps are found, however, the next state will be `search_2`. This state calls the `calc_index` procedure. This procedure uses the delay tap result vector to calculate the longest series of consecutive zeroes and outputs the start and stop index of this series. VHDL code for the `calc_index` procedure is presented in Figure B.2. The next state, init_serdes, is active the next clock cycle. To load a specific delay tap, the SelectIO Interface IP requires a reset pulse. The state asserts the reset pin, and also outputs the chosen delay tap to the input of the IP. The selected delay tap is defined as $dp = \frac{index1+index2}{2}$, where $index1$ and $index2$ are returned from the calc_index procedure.

The next state is then `wait_until_valid` which only waits for four clock cycles. The SelectIO Interface IP uses three clock cycles before it reliably outputs sample data. Two more clock cycles are needed before data is sampled into the phase aligner module. The phase aligner will in the next state, locked, assert the `locked_in_eye` signal which will be used by other modules. Furthermore, now the input data will be inspected for an aligned comma. When an aligned comma is detected, a counter will increment until it reaches 255 consecutive aligned commas. The phase aligner will then switch to the aligned state. This is the final state and asserts both the `locked_in_eye` and `aligned_to_comma` signals. The state machine will return to idle if the run register is disabled.

The state machine is developed as a Mealy state machine. This means that the

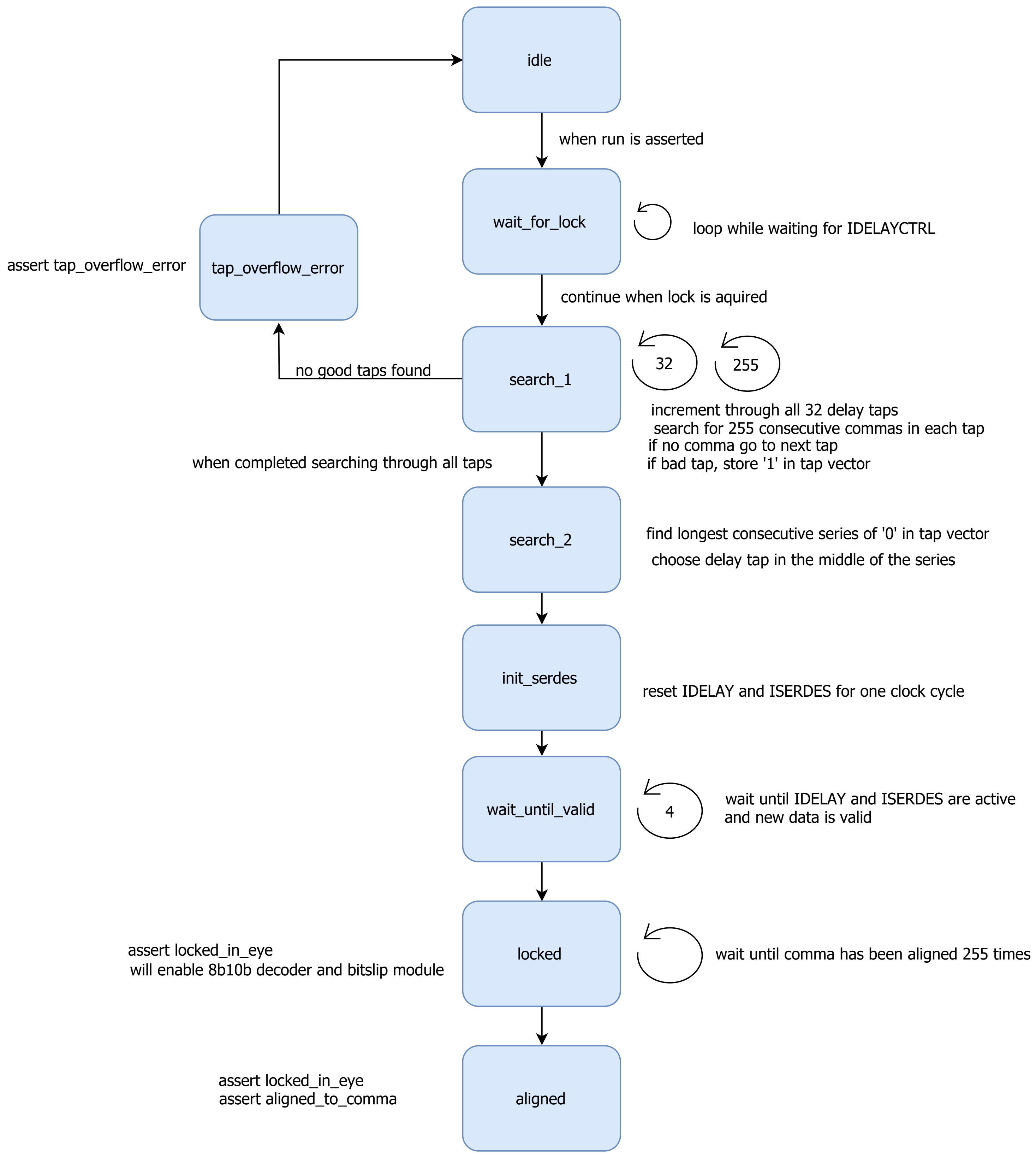

idle
when run is asserted
wait_for_lock
loop while waiting for IDELAYCTRL
continue when lock is aquired
assert tap_overflow_error
tap_overflow_error
no good taps found
search_1
32
255
increment through all 32 delay taps
search for 255 consecutive commas in each tap
if no comma go to next tap
if bad tap, store '1' in tap vector
when completed searching through all taps
search_2
find longest consecutive series of '0' in tap vector
choose delay tap in the middle of the series
init_serdes
reset IDELAY and ISERDES for one clock cycle
wait_until_valid
4
wait until IDELAY and ISERDES are active
and new data is valid
assert locked_in_eye
will enable 8b10b decoder and bitslip module
locked
wait until comma has been aligned 255 times
assert locked_in_eye
assert aligned_to_comma
aligned

Figure 4.10: State machine diagram of automatic phase alignment scheme.

outputs depend not only on what the current state is, but also on the inputs and internal logic [45, Chapter 10]. E.g., the increment tap delay signal is asserted several times in the `search_1` state.

It is crucial that, once the delay tap has been chosen, nothing may cause the sample point to become ambiguous. The phase may drift over time due to external factors like temperature, or there may be minor differences in the frequency of the data and the sample clock. This should be tested before the final implementation, or the external factors must be controlled.

4.6.3 8B/10B Decoding Module

The 8B/10B decoding module[6] used in this design is purely a combinational design and uses the mathematical algorithm developed by the 8B/10B patent writers [25]. However, one feature is added to benefit the pCT design. An enable signal must now be asserted for the module to produce error signals. The enable signal input is connected to the phase aligner module's `locked_in_eye` signal. The 8B/10B decoding module will therefore not produce error signals before the phase alignment process is in a locked state. This prevents assertion of bitslip signals to the ISERDES primitive during the phase alignment stage.

4.6.4 Protocol Checker Module

The protocol checker[7] is used for testing the ALPIDE data protocol. The phase aligner provides the `comma_aligned` signal which the protocol checker module uses as an enable signal. The module checks the incoming data stream for valid ALPIDE protocol data words. Errors will strobe specific output signals according to the type of error that has occurred. The module has been further upgraded with a test mode feature for the pCT design (see Section 5.3.1). This mode can be activated by enabling the added test mode register. When the test mode is enabled, the data input to the protocol checker is checked against three defined test vectors. This adheres to the ALPIDE test functionality, where three test vectors can be deployed to transmit consecutively followed by 8B/10B decoding. The test vectors are set in writable registers in the data module. Whenever the input data do not correspond with the test vectors or the predefined comma, the module strobes an error signal.

[6]Developed by Chuck Benz and is a Verilog design released open source [46].

[7]Originally designed by Matthias Bonora (matthias.bonora@cern.ch) at CERN, but modified for pCT usage as mentioned in Section 3.6.3.

4.6.5 Bitslip Module

The bitslip module is responsible for controlling when to assert the bitslip signal to the ISERDES primitive. As previously noted, by asserting this signal the output bits from the ISERDES will be reordered.

Obtaining Word Alignment

Whenever the phase aligner module is in the locked state, the bitslip module will produce a bitslip strobe when the 8B/10B decoding module produces an error for three consecutive clock cycles. The bitslip module requires this amount of errors to prevent the bitslip signal from being asserted too often. While the SERDES produces the reordered data bits the next clock cycle, the 8B/10B decoding module is not able to decode the data in an instant. The bitslip module will, therefore, assure that the 8B/10B decoding module error is a result of the latest data. The result of strobing bitslip until the 8B/10B decoding module no longer outputs an error is that the SERDES reorders the bits until the data is word aligned. The 8B/10B encoding of the comma ensures that no other order of the bits will produce a correct code. This is confirmed by the phase aligner module which eventually asserts the `aligned_to_comma` signal which is connected to the bitslip module. Consequently, the bitslip module will no longer strobe bitslips from 8B/10B decoding module errors.

Fixing Bitslip Errors

While it is assumed that the phase and frequency should be locked during an entire readout, the bitslip module will allow the assertion of bitslips while the phase aligner is in the aligned state if the protocol checker strobes an error. In the current design, this feature is only enabled if the protocol checker is in test mode, and the error is produced by data not corresponding to test vectors. If multiple bitslips are strobed in this state while testing, the feature must be considered to be enabled while the test mode is disabled. However, if more errors are observed when this feature is enabled than there are when it is disabled, this feature may be considered removed altogether.

4.6.6 Module Overview and Test Process

Figure 4.11 shows the schematic of the phase and word alignment design. The I/O primitives are instantiated as a SelectIO Interface IP. The data out from the ISERDESE2 is directly connected to both the phase aligner module and the 8B/10B decoding module. As noted above, the phase aligner will not start to operate before the run register is enabled and the IDELAYCTRL is locked. The phase aligner will then increment through the delay taps, before eventually choosing a delay tap for optimal sampling reliability. The 8B/10B decoding module will be enabled and asserting errors. These errors will result in bitslip strobes from the bitslip module to the ISERDESE2, until the phase aligner module confirms that the data is word aligned.

These modules are all instantiated as entities in the data module. The data module must be regular to ensure that all data channels will reuse this design. Within this module, there is also a test process that can be used to confirm that the design works in the development process, that the phase is locked and that data is aligned before the pCT beam is activated.

The process counts bitslips and also compares the incoming data, after 8B/10B decoding, to predefined test vectors. Checking starts when the phase aligner asserts that the data is aligned. The test vectors are set by the same registers that connect to the protocol checker. The process counts both correct and incorrect words. Both the correct and incorrect word counters are connected to two 48-bit registers and incoming data can, therefore, be checked continuously for 12 hours with a data rate of 1.2 Gb/s without risking overflow.

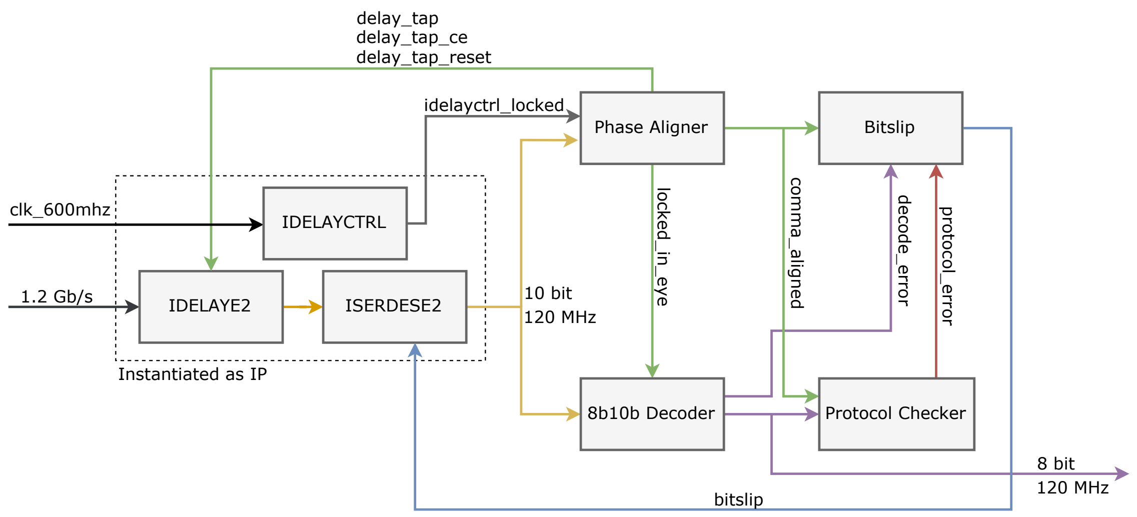


Figure 4.11: Schematic of phase and word alignment design.

Both the 8B/10B decoding module and the protocol checker module are written in other languages than VHDL; Verilog and SystemVerilog, respectively. It is beneficial for a design to have all modules in a given language and with given code design rules. Since the rest of the firmware design is written in VHDL, translating these modules should be prioritized.

4.6.7 Registers

The design requires and benefits from a few registers. Being able to control and monitor the process via a communication channel is crucial to verify the design. A list of all the registers and their functions is found in Appendix C.

4.7 Filtering, Tagging, and Buffering Design

4.7.1 Data Filtering

The ALPIDE transmits IDLE words when data is not ready to be transmitted. These data words are redundant and must be filtered out from the data stream. However, the IDLE word can also be observed when it is part of data. E.g. a hit in the pixel row 127, 377 or 511 in any chip region will produce the same word and will resemble IDLE. These data are not redundant and must not be filtered out. Therefore, the filtering step must require that the IDLE word occurs consecutively. The ITS design has designed an idle suppress module which buffers the data and looks directly on this data to examine if the idle is occurring consecutively. However, a more efficient solution will be to employ the protocol checker (see Section 4.6.4) to indicate whether an observed IDLE word is part of the data or not. Since the protocol checker employs a state machine to control where in the protocol the current data word belongs no buffering is required. This will require an expanding of the protocol checker module.

Another task for the data filter stage is to ignore data that is sampled during the startup test phase. Since this is indicated by the test mode register in the data module, this can be input as an active low enable signal in the filter design. The design can also require that the phase alignment process has indicated that the data is aligned before enabling the buffer FIFO. These are features that are not implemented in the ITS' idle suppress module, and therefore another reason for redesigning the filtering stage.

4.7.2 Data Format

The data received by the readout units must be formatted in such a way that it is evident when and where the data is coming from. The exact data format is to be defined by the pCT software group, but essentially what is required in addition to the ALPIDE data format is the following:

- Readout Unit ID
- Stave ID
- Strobe/Trigger ID
- Timestamp
- Chip operation mode
- Strobe length
- Strobe gap
- Error flags

Identifying and labeling the data with a strobe/trigger ID is not as trivial as it may seem. The ALPIDE may still be transmitting data from a TRIGGER #1 some time after TRIGGER #2 (or even TRIGGER #3) has occurred. The readout unit may therefore not simply tag it with a counter of external trigger signals without looking at the data. Moreover, this method is not suitable if the internal trigger sequencer is to be employed.

By employing the external TRIGGER-initiating signal, the control module will *not* filter TRIGGER commands from chips in the BUSY state. In continuous mode, this causes an interruption of the current data transmission, while in triggered mode, the chip will ignore the command. In any case, there will still be produced event headers, or simply an empty chip data packet in the case of triggered mode. These data format words can be used to indicate which TRIGGER ID the data belong to. Therefore, the readout unit can count the number of chip headers and chip empty frames for each separate ALPIDE channel, and tag the data accordingly.

4.7.3 Data Tagging and Buffering

As an event is sampled and the redundant data is dumped, the rest of the data must be temporarily stored until the whole event is offloaded from the ALPIDE. A FIFO may be used for this purpose. However, the data tags must also be stored with this data. There are two ways to accomplish this:

1. Two separate FIFOs.
   One for ALPIDE data, and one for data tags. The data tag FIFO will then be extracted by the next stage before the data FIFO. This method requires some extra logic in the offloading stage in order to combine the data from two FIFOs correctly.
2. Delaying the ALPIDE data.
   Buffer the data in registers for the amount of clock cycles required to write the tag data. This, however, is complicated if two data events are transmitted back to back. The logic will be intricate in order to prevent erasing parts of the last event.

Because of the issues that may occur if two events are transmitted back to back, the most convenient solution is the former. Furthermore, the combining of data of two FIFOs is made simple by creating FIFOs with AXI Stream master interface and employing the AXI Stream Interconnect core (see Section 4.7.3).

If the data width of the FIFO is decided to be larger than the original data width (8 bits), the data must be delayed, or the FIFO must support partial-word writing. The width and depth of the buffering FIFOs will depend not only on the rate and size of events per ALPIDE, but also the size of the data format tagging and the method for extracting the data from the FIFO and the offloading protocol.

The buffering step must also include an event counter that sets a flag that indicates to the offloading step that there exist, at least, one whole event in the FIFOs. This part will analyze the incoming data and react to empty data headers and data footers that indicate that complete events have been stored. One could consider creation of several flags to indicate priority if several events were fully stored in the FIFO. However, there is no guarantee that two events will occupy more memory space than a single large event.

Additionally, the logic must identify when a read sequence by the offload stage is completed. One possible solution is to also analyze the FIFO output data for empty data headers and data footers.

AXI Stream FIFO

Employing a FIFO with an AXI Stream interface will not only simplify the combing of data from two FIFOs but will also aid in constructing the offloading stage because it is the most used interface by Xilinx' IP cores for networking.

A write is initiated by asserting the slave valid signal. Whatever the value is on the data in port will be stored in the FIFO on the next rising edge. This is illustrated in the timing diagram in Figure 4.12. Therefore, the filter module may use this signal to indicate whether a word is a valid data word assigned for storing.

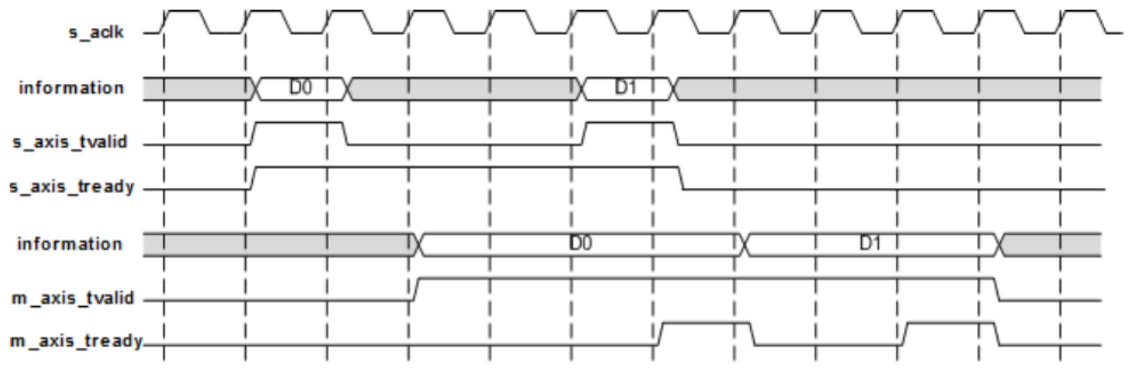


Figure 4.12: AXI4-Stream FIFO Timing Diagram [47, Figure 1-3].

4.7.4 Design Overview

An enhanced protocol checker must provide information on whether the incoming data contains IDLE words that are *not* part of the pixel hit data. Furthermore, to aid the filtering process, the protocol checker must also indicate whenever an event is completed. That is, all the data for the events have been sampled and checked by the protocol checker. This can be implemented by asserting a signal whenever a chip trailer word or empty chip header occurs. Also, the protocol checker can also indicate to the data tagger module that either a chip header or empty chip header has occurred and that a data header should be produced and stored in the tag FIFO.

Figure 4.13 shows how the enhanced protocol checker should operate acting on a series of sampled ALPIDE data words. Note that, for space reasons, the diagram underestimates the bit length of chip header and chip trailer data words. The data must be buffered since the protocol checker outputs the results of the inputs on the next rising clock edge. The `idle` signal is asserted as a result of incoming comma and idle words, depending on the state of the protocol checker. Whenever the data is aligned, and the test mode is inactive, the write enable signal (`wr_en`) is the complement of the `idle` signal. The write enable signal can be connected to the `tvalid` input on an AXI Stream FIFO, and as a consequence, trigger a store action on the FIFO as illustrated in Figure 4.12. The `event_rdy` signal indicates that a complete event has been sampled. In the diagram, it is asserted as a consequence of the chip trailer word. Note that it should also be asserted whenever an empty chip header occurs. The `header` signal indicates that a chip header has been observed. As the `event_rdy` signal, this signal will also be asserted whenever an empty chip header occurs. The signal is used to indicate that the data tagger module should produce and store data. The timing diagram does not illustrate the situation where an IDLE word is part of the data, but this is discussed in Section 5.8.

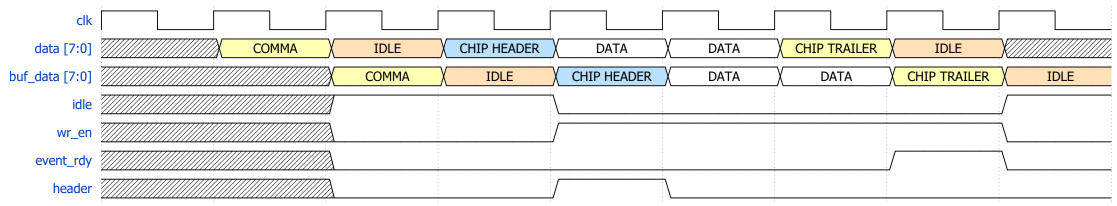


Figure 4.13: Timing diagram of enhanced protocol checker outputs based on ALPIDE data words. Assuming that data is aligned and that the test mode is not active so that `wr_en` is the complement of the `idle` signal.

Figure 4.14 shows a proposed design implementation for data filtering, as well as data format tagging and buffering. The incoming data and the `comma_aligned` signal are output from the design implementation in Figure 4.11. The `test_mode` signal is a connected register. The block uses the outputs of the enhanced protocol checker as described above. The `idle` signal produces the write signal connected to the data FIFO. As discussed in Section 4.7.1, the data that is sampled during testing should not be stored in the FIFOs. This is accomplished by ANDing the complement of test mode register with both the `comma_aligned` signal and the complement of the `idle` signal. The `wr_en` signal is connected to `tvalid` input on the FIFO. This prevents the FIFO from filling up with data that is not relevant.

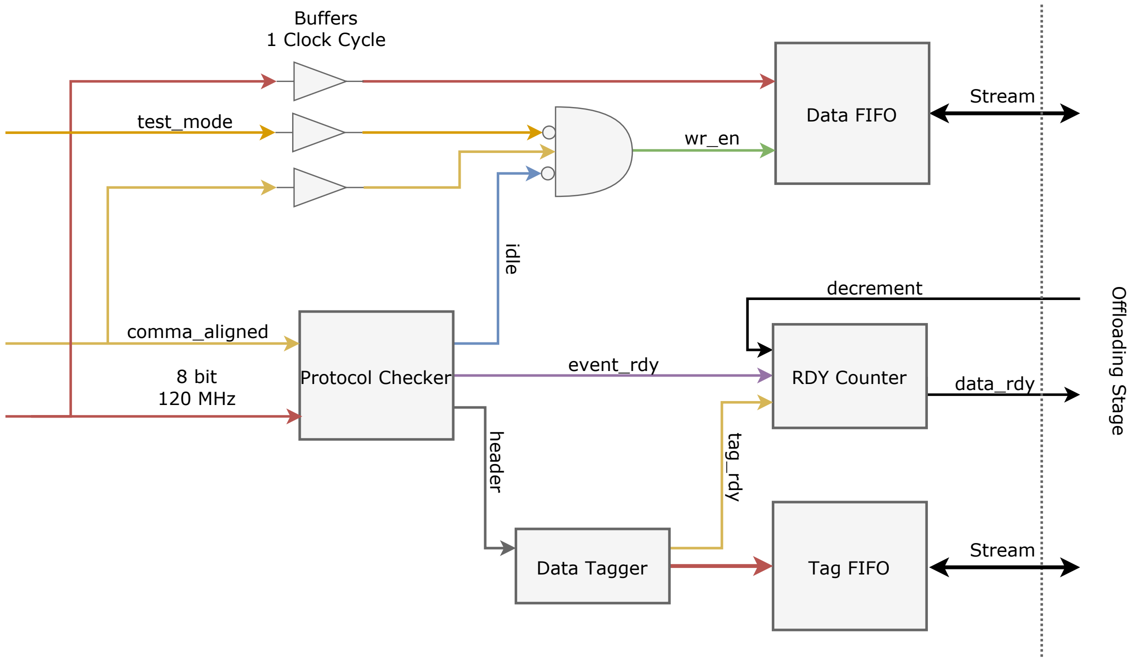


Figure 4.14: Data filtering and buffering design proposition.

Whenever a header signal is asserted, the data tagger module initiates the process of creating and storing the data tag. When this process is completed, the `tag_rdy` signal is asserted. Since the final data format is not yet defined, the data tagger module is a black box in this stage of the development. The signals from the data tagger to the tag FIFO contain both tag data and the required FIFO protocol signals. The test mode and aligned signals are also implicitly connected to the data tag FIFO, in the same way as the data FIFO.

The RDY counter module's task is to indicate to the offloading stage that both the data and tag FIFO contain *at least* one complete event. The `event_rdy` signal indicates that the event data has been fully stored in the data FIFO, and the `tag_rdy` signal suggests that a full data tag has been stored. The RDY counter counts these events in two separate counters and asserts the `data_rdy` signal whenever *both* of these counters are above zero. The counters are both decremented whenever the offload stage has completely read a whole event from the FIFOs and asserts the `decrement` signal.

This design is not yet implemented, but the protocol checker is enhanced to perform the actions as discussed. This is presented in Section 5.8.

4.8 Offloading

The offloading stage is responsible for combining the data that are temporarily stored in multiple buffer FIFOs, to one or more high-speed data streams. The decision regarding use of transport layer protocol (like UDP or TCP/IP) falls beyond the scope of this thesis, and will be evaluated by the network engineers responsible for the data server. However, it is assumed that the physical layer of the output may be Ethernet or optical. The number of output links will depend on physics simulations of event rates and sizes.

To combine the data, logic must be developed that, for a group of data modules, look for the flag that indicates a whole event is buffered. When a flag is detected, the event (and tagging data) is streamed out to the data link. Scanning of flags can be executed as a round-robin scheme, or implemented as interrupts to the CPU which handles the task with software routines. Furthermore, the offload stage must also detect whenever a whole event is read and assert the `decrement` signal to the data module. This implies using a protocol checker to affirm where the empty chip header or the chip trailer data words occur.

4.9 Communication and Control

4.9.1 Bus Interface

The bus interface is a connection for communication between the different modules on the FPGA. It is common to utilize bus interfaces that employ a master module and multiple slave modules that are controlled by the master. Some interfaces also enable utilization of several master modules which share control of the interface. When choosing the type of bus interface to be implemented on the pCT readout unit, the following must be considered:

- Robustness and reliability
- Ease of setup
- Ease of modification (modularity)
- Handling of multiple clock domains
- IP support
- Dependence on proprietary tools

Two bus interfaces were considered for use in this design; Wishbone and AXI.

Wishbone

Wishbone is a very straightforward but flexible bus interface developed to standardize the communication between IP cores [48]. The bus interface itself is very simple, and beyond the clock, reset, data and address signals, only consists of four compulsory signals. Figure 4.15 shows how the master module asserts `STB_O` when it is ready to transfer data. The valid data will remain on the data output until the slave asserts `ACK_I` or one of the error signals. A similar process occurs in a read cycle.

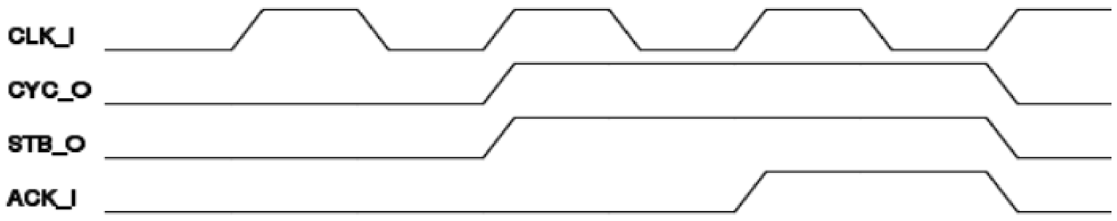


Figure 4.15: Wishbone Handshaking Protocol [48, Illustration 3-3].

Wishbone was already implemented in the ITS firmware design. The Wishbone bus is also connected to the Cypress FX3 USB chip, for communication with external devices, via a wrapper implemented on the FPGA. Thus, Wishbone was a natural choice for the start of the development process.

Several tools automate the creation of a Wishbone interface, like the `wbgen2` and `wb_builder` utilities [49, 50]. These tools are not standardized with warranty but are open source licensed. The `wbgen2` utility creates VHDL source files and C headers, as well as documentation, based on a standardized input file. `wb_builder` creates VHDL source files from an input file as well, but also provides a GUI. The ITS developers may have used these tools when building the system, but have manually edited the results to comply to the multiple clock domain crossings in the design. Hence, any modification of the interface must be done manually.

Another problem is the multiple clock domains in the design. Wishbone has no standardized method of handling clock domain crossings. Every register must, therefore, be synchronized manually to avoid meta-stability issues. This has resulted in a quite intricate design in the ITS firmware, which requires significant labor to maintain or expand.

Wishbone is widely supported by the open hardware movement and was designed and developed by contributors at OpenCores. OpenCores also has a collection of a wide range of IPs for different purposes. Many of these have Wishbone interface support. No special tools or specific hardware target are required to work with Wishbone, as is one of the main goals of the developers [48].

AXI

AXI is part of the ARM AMBA microcontroller bus family [51]. The latest version is called AXI4. There are three variants; AXI4, AXI4-Lite, and AXI4-Stream. The

Stream version is developed for high-speed streaming data where the concept of address is not present or required, but the other two versions are targeted for memory-mapped communication. However, the full version supports data bursts, that is a transaction that consists of more than one transfer.[8] The AXI bus is deemed to be an industry standard, and implementation of it is considered to be highly reliable and well tested. It is also possible to automatically enable a protocol checker to validate the operation of the interface.

The AXI interface consists of multiple channels:

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

Each of these channels contain multiple signals to handle both the data and the control signals. Figure 4.16 shows the process where the master module writes to the slave. The channel operation is similar to the control signals of the Wishbone bus. Although, the AXI interface has far more control signals as each channel has its own valid and ready signals. The valid signal indicates that the data/address can be sampled by the slave peripheral, while the ready signal indicates that the slave is ready to receive data/address. The Write Data channel also contains a strobe signal that indicates which bytes of the data bus are valid and should be read by the slave. The Read Data and Write Response channels contain a response status signal that indicates whether the transfer was completed successfully. [52]

The connection of multiple master and slave modules are achieved by employing a Interconnect IP. This IP can be automatically generated by using Xilinx Vivado Design Suite. The AXI Interconnect module can be enabled to take care of clock domain synchronization and ensures that no meta-stability issues occur. This is achieved by AXI FIFOs inside the interconnect module, and is a major benefit for the pCT system due to the multiple modules that operate in different clock domains. The layout of the connections is designed in Vivado by utilizing the Block Design feature (see Section 5.5.1). This function makes it possible to modify the design. However, the development process is entirely dependent on Xilinx' tools. There are no tools that can automatically create a complete AXI bus interface as there are for Wishbone, but Vivado can create a template of an AXI slave based on the user's input. The Block Design feature simplifies the process of defining the modules offset addresses and their range.

Xilinx has adopted the AXI bus protocol for their IP cores [51]. This simplifies the incorporation of these IP cores as the communication interface is predefined,

[8] A transfer refers to a single clock cycle where information is communicated, validated by a handshake. Also known to as a data beat [51].

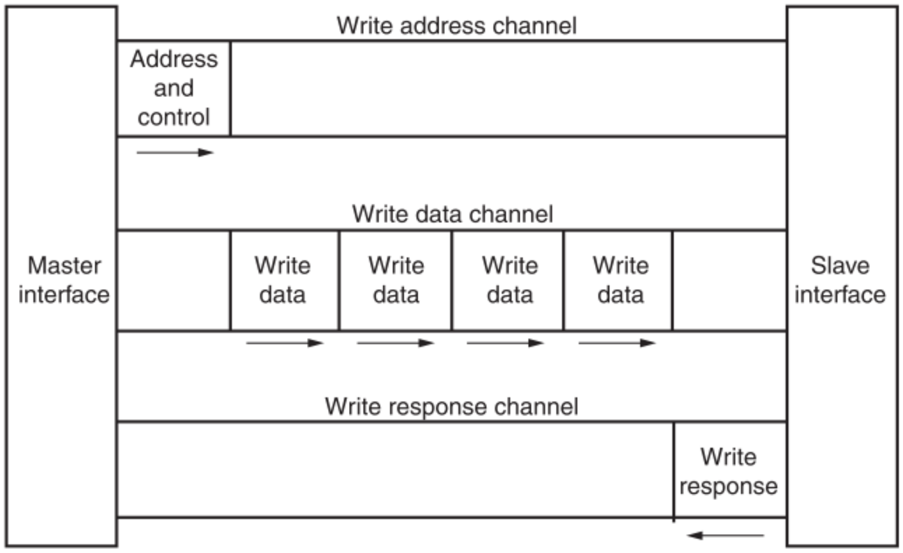


Figure 4.16: AXI Architecture of write process [51, Figure 1-2].

especially when employing a processor system on the Xilinx FPGAs (see Section 4.9.3). The AXI interface can also combine memory-mapped modules with streaming, and may, therefore, prove useful if Direct Memory Access (DMA) or similar is to be implemented.

AXI to AXI Connector

The AXI Interconnect IP core has a limit of 16 masters and 16 slaves. The pCT readout design requires a minimum amount of 180 AXI slaves (162 data modules and 18 control modules). To expand the number of supported masters and slaves, Xilinx provides an AXI to AXI connector [53]. This module simply enables the designer to connect multiple AXI Interconnect cores in a cascade.

Bus Interface Selection

As the modules of the pCT readout unit will operate on different clock domains (see Section 3.6.2 and 4.10), it is important that the clock domain crossings are handled properly. Although this is possible to achieve with the Wishbone interface, it is not done automatically. Furthermore, it requires more modification if registers are to be added, removed or changed. AXI, however, achieves clock domain crossing by automatically inserting FIFOs in the data chain.

While the process of developing firmware that targets the Xilinx' FPGAs depends on using their development tool Vivado, it is not a drawback that AXI bus interface requires the same tool for developing the interconnect block. Rather, it is an advantage as the tool makes it easier to modify the design after changes in requirements.

Especially because of the probability of a high number of modules in the pCT readout unit and the ease provided by Vivado to define offset addresses and change connections it was concluded that the AXI bus interface was the appropriate bus interface for the design.

4.9.2 AXI Slave Module Template

To obtain a high degree of modularity, it was decided to define a fixed methodology for implementation of the AXI bus interface. The design would be based on the AXI4-Lite Slave template provided by Vivado. Any changes in the number of registers, or the names or modes of the registers, would be done manually. Later in the design process, development of an automated software for creating and modifying the register handler firmware was started (see Section 5.5.3).

To make the design as clean and precise as possible, VHDL records would be implemented to group every block of signals that belonged together. All handling of registers, read, write and reset, is controlled by a module separated from the rest of the logic. Naturally, registers that hold values controlled by the logic will not be writable by the bus interface.

Figure 4.17 shows how the register handling module is instantiated by the logic block. AXI protocol signals are transferred in two defined record groups from the top-level to the register handler; one group of signals that are transferred to the module, and one for signals that are transferred from the module to the interconnect block. Two records are defined for the registers; one for RW registers and one for RO registers. If found beneficial, another record can be defined for registers that are WO; registers that only hold their value for one clock cycle. The read-only registers are holding values based on the logic and are therefore transferred from the logic to the register handler. The writable registers are transferred in the other direction. Note that the read-only record is also defined as an internal signal in the logic, and may also be read by this module. Since the AXI Interconnect block handles the clock domain crossing, each register handler module can operate in the same clock domain as the logic module. Thus, no clock domain crossings are needed inside the module.

The use of records makes this methodology immaculate. When scaling up the numbers of modules in the top-level module, there are only two signals for the AXI bus interface per module. Hiding the handling of registers in another module also makes the logic module much clearer. When new developers are introduced to the project, there are no need to be distracted by how the AXI bus interface works, as this is predefined and pretested, and hence, can be hidden away. This is also an example of locality

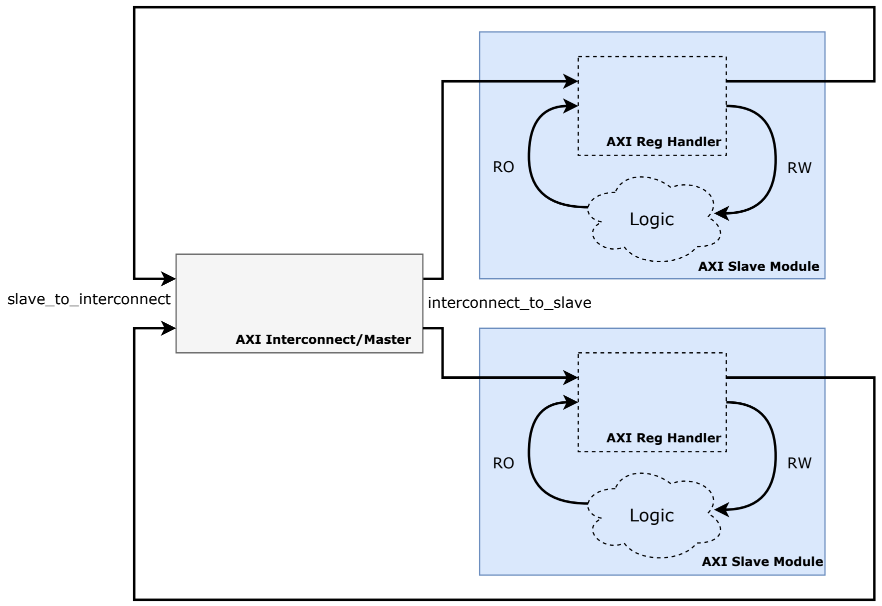


Figure 4.17: Overview of AXI Slave Module Template.

as discussed in Section H.2.4. To free the developer from considering clock domain crossings and possible meta-stability issues are also highly beneficial.

4.9.3 Soft Processor System

As discussed in Section 4.3, the Virtex UltraScale+ does not include a hard processor system. However, a Microblaze soft processor core may be implemented in the FPGA fabric. The Microblaze core natively supports all the AXI4 bus types, and it consequently simplifies the connection of peripherals to the processor system. As the AXI interconnect supports multiple master modules, the processor may serve as one of these modules. The Microblaze core is a 32-bit RISC processor and is developed to satisfy a diverse set of embedded applications [54]. The processor system on the readout unit can potentially handle the following tasks:

- Communication between the control room and the ALPIDE chips, and other peripherals on the readout unit, as well as all the modules on the FPGA
- Automatic configuration of the ALPIDE chips
- Control of the phase alignment process, e.g. reset after error occurs within fixed amount of time (see Section 4.11.1)
- Timestamp creation
- Control and monitoring of the data offload
  - Configuring hardware devices (e.g. PHY and MAC)
  - Implementation of transport protocol if required

External communication with the processor can be accomplished with Ethernet via the AXI bus. However, in the early development phase, it is more convenient that the processor is controlled by UART.

4.10 Timing Requirements and TRIGGER Generation

The pCT scanner will, as discussed, contain a large number of ALPIDE detector chips and subsequently, the corresponding number of readout units to handle these chips. It is important that all units are synchronized to a single clock for the entire system. This is because the readout units supply the clock to the ALPIDE chips, which must all operate on the same clock edge to ensure that all strobe windows occur at the same time. The firmware modules on the readout units operate on different frequencies. The clock rates required are listed in Table 4.1. The control module operates on the same 40 MHz clock as the ALPIDE chip, but also require a 160 MHz clock for data sampling (see Section 3.6.2). The data module requires a clock that is one tenth of the ALPIDE data rate, therefore 120 MHz. This clock will be synced to each data word received by the module. 600 MHz is required by the ISERDES primitive for DDR sampling of the data stream. The ALPIDE voltage module (see Section 4.11) may operate on any clock frequency compatible with the external voltage regulators it communicates with. The Wishbone wrapper operates on the same frequency as the Cypress FX3 USB chip.

Table 4.1: Clock rates required by each firmware module.

	160 MHz	**120 MHz**	**80 MHz**	**600 MHz**	**60 MHz**	**40 MHz**
ALPIDE Control	×					×
ALPIDE Data[a]	(×)	×		×	(×)	
ALPIDE Voltage		×				
Wishbone Wrapper/Bridge			×			

[a] Design proposals that included transceivers also required a 160 MHz reference clock for the transceiver, and a 60 MHz output clock for each 20-bit word received.

All clocks may be produced externally and delivered individually to each readout unit. However, this will take up I/O resources and wiring. The UltraScale+ FPGA family has many clocking resources for generating and handling clocks. Therefore, the readout unit may be provided with just a single clock, and each of the clock frequencies listed can be generated internally. As the jitter on the provided system clock is distributed further down the hierarchy of clocks, it must be minimized. A high-precision clock generation system available from vendors like Silicon Labs may

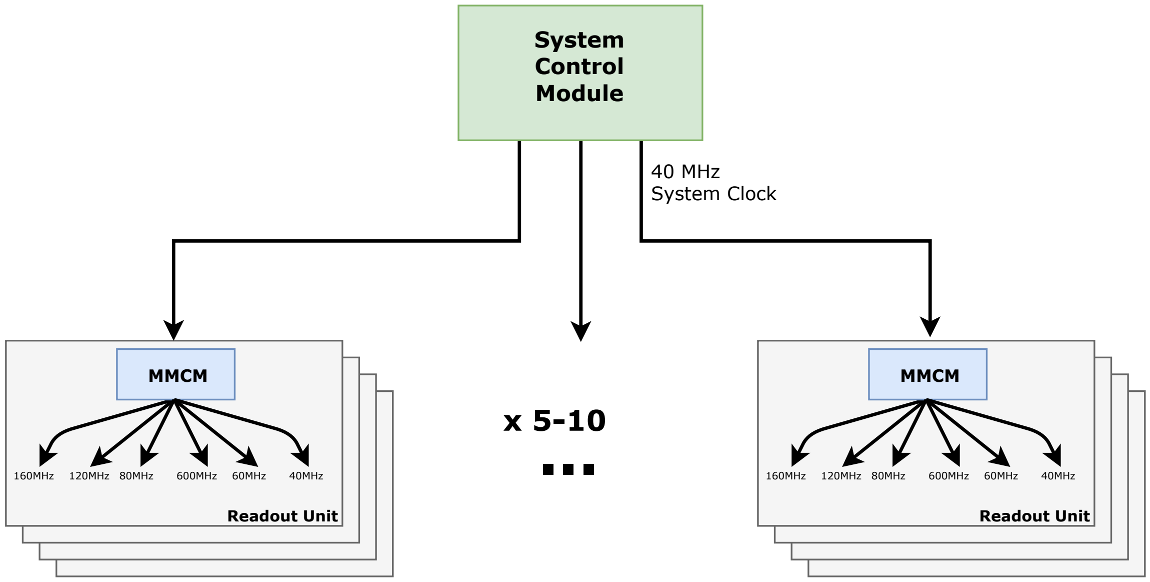


Figure 4.18: Clock generation scheme and jitter minimization.

be responsible for generating the reference clock. The distribution to readout units can be done in a group of four with a wire of the same length. This way the fanout, and thus, the delay, will be roughly equal to each readout unit. Each wire stub must be terminated correctly to avoid reflections and jitter. The clock signals should be provided on a differential line to avoid common mode interference. The clock may be distributed by an external control module (as proposed in Section 4.10.1). Furthermore, the UltraScale+ Mixed-Mode Clock Manager (MMCM), which is an advanced PLL-type primitive, has jitter filtering features which may help to reduce the amount of jitter.

The most important design guideline for preventing signal integrity problems is to always use the longest rise time possible [39]. As the rise time of a signal typically is between 7-10% of the period, the clock frequency should be as low as possible. The lowest clock frequency in the design is the 40 MHz clock that is interfaced to the ALPIDE chips, and this will also produce the longest rise time. Accordingly, there are few arguments for choosing a fast clock signal as the main system clock. Transmitting the clock from an external module will produce the least problems, and will also contribute to the closest possible synchronization of the ALPIDE operations.

Figure 4.18 shows how an external module is distributing the system clock to the readout units in pairs of four. Five to ten differential clock lines are needed depending on the number of layers in the final design.

4.10.1 TRIGGER and STROBE Scheme

As discussed in Section 3.5.3, the STROBE signal is generated following a TRIGGER command. Moreover, there are multiple ways to initiate a TRIGGER command. The pCT scanner relies on data that can be referred to a given time. As a consequence, it is vital that all the ALPIDE chips produce the STROBE signal at the same time.

As mentioned, the control module may start transmitting a TRIGGER command by asserting an external signal (see Section 3.6.2). To ensure that the strobe timing requirement is fulfilled, an external module may send this TRIGGER-initiating signal to all readout units at the same time. To ensure that there are no damaging jitter or delay, the signal should be transferred the same way as the global system clock signal, as a group of four with a wire of the same length. As the SCU already handles the 40 MHz system clock and the ALPIDE chips also operate on this frequency, it is natural that the external trigger signal is synchronized to this clock. This signal frequency will, consequently, produce less signal integrity problems.

When the pCT scanner is initializing, the SCU will receive a signal from the control room or the beam instrument that starts the triggering process.

There are two ways this can work:

1. The SCU initiates all the TRIGGER commands, based on the first start-up signal from the beam, and hence all the STROBE windows.
2. The SCU initiates just the first TRIGGER command, and then the ALPIDE's internal sequencer continues the process.

As all ALPIDE chips are synchronized to the same clock, the internal sequencer should produce the same framing window on all ALPIDEs if the TRIGGER command is dispatched from all the readout units concurrently. Thus, there should in principle be no differences between these two methods. Nevertheless, the ALPIDE chips have built-in registers that simplify changing the TRIGGER signal interval and STROBE length. If the SCU would be responsible for this, there had to be developed more extensive firmware for this unit as well. One particular advantage of employing the external TRIGGER-initiating signal is that it keeps the AXI bus free for other types of communication. In particular, communication for power control and monitoring. Hence, the preferred method is to only use the SCU to initiate the first TRIGGER command.

4.11 Power Control and Monitoring

The readout unit is responsible for controlling and monitoring the power usage of the ALPIDEs. Individual power lines are distributed to each stave. The voltage and current should be controlled by regulators on the pCT readout unit. The regulators must be chosen so that the FPGA can control and monitor these. The interface between the FPGA and the regulators should be as simple, and use as few pins as possible. There is also no need for the communication to be very fast. I^2C and SPI are common interface options when communicating between ICs on a PCB.

All ALPIDE chips contain ADCs with a 10-bit dynamic range that makes it possible to measure each single chip's current, voltage and temperature [6]. The procedure involves writing and reading the ALPIDE registers, and therefore requires the use of the slow control bus. Consequently, monitoring the status of each individual ALPIDE must be done by one of the AXI master modules; the CPU or the SCU.

The main purpose of the power control module on the readout unit is to transmit information about the power status of the regulators to the control room to control that the data capture procedure is only run when the ALPIDE chips are functioning. This can be done in two ways:

1. Via the main AXI bus
2. Via another bus interface

Employing the AXI bus for continuous polling of the status of the individual ALPIDE chips may cause unnecessary traffic on the bus. This, in turn, may cause conflicts when configuring the ALPIDE chips for readout. To prevent conflicts on the AXI bus during readout and initialization, the SCU may be directly connected to the power module on each readout unit with UART.

However, it can be argued that directly monitoring each ALPIDE chip is unnecessary during other stages of communication activity on the AXI bus. Preventing polling the ALPIDE during other activity will solve the potential conflicts on the bus. Employing AXI for the main communication bus for the SCU to the readout units will greatly simplify the design of this unit. Regardless of communication method, the SCU should be the main source of power information directed to the control room.

Another feature of the power control module should be detection of and protection from the effects of latch-up or other SEUs in the ALPIDEs. Latch-up is still considered a potential source of failure [55]. A latch-up usually requires a power cycle to eliminate the low-impedance path that is created in the device in question. If this occurs during readout, the power control module should only turn off the power of the stave with the affected chip. The module must also store information about which staves that have been switched off during readout. If an error is detected at other times, the power control module must perform a power cycle of the stave in question and also indicate that these chips need to be reconfigured. This indication may be delivered as an interrupt to the soft processor, which will then initiate chip reconfiguration.

How the power control module should detect these errors falls beyond the scope of this thesis.

4.11.1 Power-On Sequence

The power-on sequence of the pCT readout system ought to set up and prepare all units for data taking. The following steps are required:

- Power-on readout unit and SCU
  - The process of turning on power for the readout units. The power is delivered from an external power unit which delivers power to the entire system.
- Program FPGAs
  - Controlled by the external SCU or via an on-board physical ROM chip.
- ALPIDE power-on
  - Controlled by control room software, but accomplished via the readout unit's power control module.
  - Wait until stable.
- Enable driving of readout channel on ALPIDEs
  - This will start the transmission of comma words on the high-speed data channel.

If employing the I/O features for data capture:

- Enable readout unit automatic phase and word alignment
  - Check whether the IDELAYCTRL is locked. Wait a specified time before resetting the I/O primitives and the data module.
  - Wait until data is aligned.
  - Wait for 60 seconds to check if the data remain correct. If errors are reported, repeat the phase alignment process until the check succeeds.
  - The 60 seconds are chosen without thorough consideration and should be judged by tests of when the first error occur (if any). See Section 5.7.2 for test results.
  - This initialization procedure should be controlled by software running on the readout unit soft processor to avoid too much communication to and from the control room. The software is enabled, however, by calls from the control room software.

If employing transceivers for data capture:

- Start the transceiver start-up procedure and wait the amount of time required for data capture.

CHAPTER 5

Development and Testing

This chapter deals with the following steps of the development and testing process:

1. Setup and development of an ALPIDE emulation model testbench environment.
2. Setup and development of a readout unit testbench environment.
3. Testing of CDR with ITS firmware.
4. Development and testing of CDR with transceivers only.
5. Replacement of the Wishbone bus interface with the AXI bus interface.
6. Testing of CDR with I/O primitives.
7. Development and testing of the automatic phase and word alignment firmware. Analysis of results.
8. Development and testing of data filtering firmware.

5.1 ALPIDE Lightweight Model Testbench Environment

At the beginning of the work of this thesis, it was a goal to develop a simulation model of the ALPIDE to aid and validate the development of the readout unit firmware and software. However, quite early in the process, the ALPIDE Lightweight Model (ALWM) was made available to the pCT group. The ALWM is developed by engineers of the ITS upgrade team. It is a behavioral model and is not suitable for synthesis. The model is based on the actual RTL code for the ALPIDE chip but where the pixel matrix is replaced with a simple internal data generator. Whenever a trigger command occurs, the model generates a random data set which mimics the data of an actual event.

One major advantage of this model is that, besides from the pixel matrix, all other functionality remain. Consequently, the model can be employed to learn the behavior of the ALPIDE where the documentation lacks details or clarity. A simulation

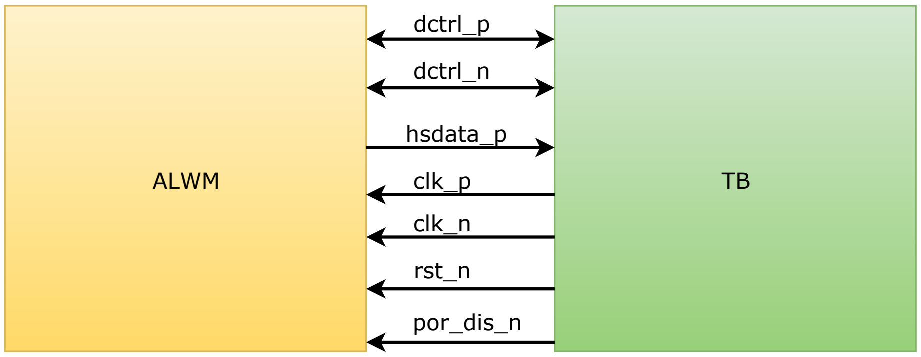


Figure 5.1: Test setup with all connected signals.

environment was set up employing Bitvis Universal VHDL Verification Methodology (UVVM) to study the ALPIDE behavior, and to confirm that the model could be used for testing the developed readout firmware (see Section H.3.2).

5.1.1 ALPIDE BFM

Developing a Bus Functional Model (BFM) for the ALWM simplified building of a test environment. Another benefit was that writing the BFM procedures meant studying in detail the operation of the ALPIDE chip. All procedures are described in Appendix A.

5.1.2 Weaknesses of the ALWM

During the several months of employing the model for development and testing, two major weaknesses were identified:

Lack of matrix: Although the model provides a simple data generator, the lack of the pixel matrix logic prevents several critical and potentially beneficial operations. E.g., it is not possible to pulse a masked set of pixels to enable output of a given set of data.

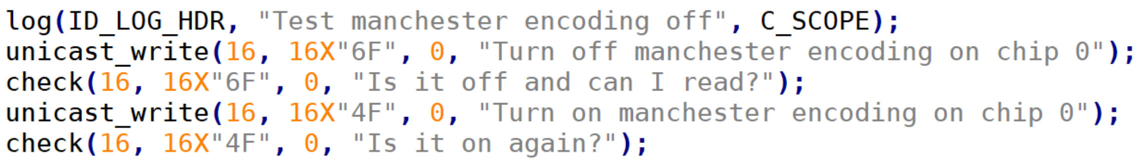
PLL accuracy: When simulating the model in conjunction with the ITS firmware, several periodical glitches were identified in the data sampling. The problem was identified to correlate with the clock generation in the model. A PLL is behaviorally modeled in SystemVerilog and uses a real variable to hold the clock period. For the clock frequency of 600 MHz (which is used to serialize the data with DDR on the data output), the period will be an infinite decimal. This number cannot be stored in a real variable, and consequently, the model will not produce the correct frequency. This issue was solved by increasing the time resolution of the simulation. With the transceiver design, this workaround caused the glitch to only appear with a significant interval between each case and could be ignored. However, when the design was changed to employ I/O delay features, the glitch appeared frequently (see Section 5.2.3).

5.1.3 Testbench Sequence

As the ALWM model was assumed to work, the motivation behind the testbench was rather to explore the limitations of the model. The test sequencer performs the following operations:

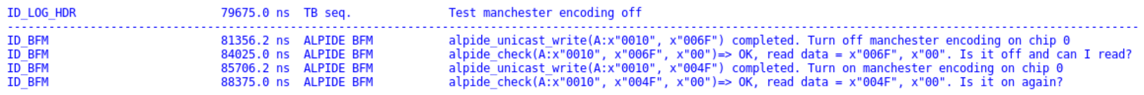
1. Establishes that there exists communication over the slow control communication channel.
2. Validates that the model has instantiated the registers with the default values specified in the manual.
3. Validates read and write to registers.
4. Turns off Manchester encoding on the slow control channel, validates that communication is still functioning.
5. Enables the readout channel, by following predefined procedures in the manual. Validates that the model transfers comma words.
6. Starts the internal trigger sequencer and validates that the model creates data and transfers that data following each trigger.

Figure 5.2 shows an excerpt of procedure calls processing step 4 of the test sequence and the resulting transcript.



```
log(ID_LOG_HDR, "Test manchester encoding off", C_SCOPE);
unicast_write(16, 16X"6F", 0, "Turn off manchester encoding on chip 0");
check(16, 16X"6F", 0, "Is it off and can I read?");
unicast_write(16, 16X"4F", 0, "Turn on manchester encoding on chip 0");
check(16, 16X"4F", 0, "Is it on again?");
```

(a) Procedure calls for turning Manchester encoding off and on, and test if data still can be read.



```
ID_LOG_HDR                    79675.0 ns  TB seq.             Test manchester encoding off
------------------------------------------------------------------------------------------------------------------------------------------
ID_BFM                        81356.2 ns  ALPIDE BFM          alpide_unicast_write(A:x"0010", x"006F") completed. Turn off manchester encoding on chip 0
ID_BFM                        84025.0 ns  ALPIDE BFM          alpide_check(A:x"0010", x"006F", x"00")=> OK, read data = x"006F", x"00". Is it off and can I read?
ID_BFM                        85706.2 ns  ALPIDE BFM          alpide_unicast_write(A:x"0010", x"004F") completed. Turn on manchester encoding on chip 0
ID_BFM                        88375.0 ns  ALPIDE BFM          alpide_check(A:x"0010", x"004F", x"00")=> OK, read data = x"004F", x"00". Is it on again?
```

(b) Transcript excerpt from testbench resulting from the above procedure calls.

Figure 5.2

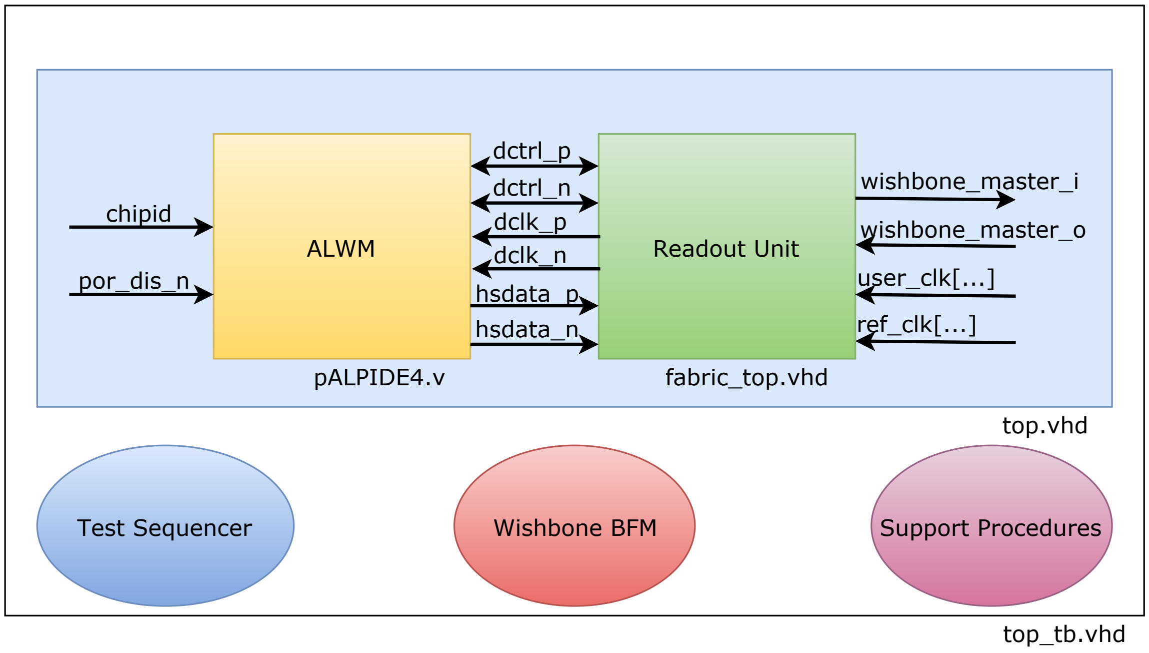


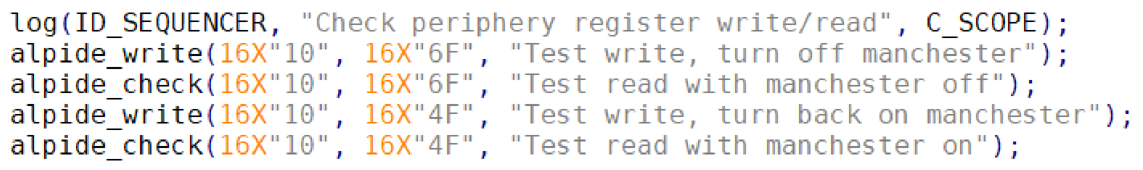
Figure 5.3: Readout Unit Testbench Environment. The readout unit is continuously updated during the development.

5.2 Readout Unit Testbench Environment

To simulate the system a testbench environment combining the ALWM and the readout unit was set up. A top-level file instantiates both the ALWM and the readout unit, and handles the signals to and from the modules. Figure 5.3 illustrates the environment. Notice that the ALPIDE chip ID is provided by the testbench top module, as well as the power-on-reset signal. This mimics how the chip will interface the readout unit when mounted on a stave. On a stave, the chip ID will be set by connecting pull-down resistors to the corresponding inputs, and the power-on-reset is disconnected by default. The readout unit provides the ALPIDE clock. The readout units are controlled via the Wishbone bus interface. The `user_clk` signals are the clocks used for the FPGA logic, and the `ref_clk` signals are used as a reference for the transceivers.

Instantiating `fabric_top.vhd` is not done directly, but by instantiating a component. The component is defined in the package-file `component_pkg.vhd`. This is done to avoid creating dummy signals for all the in/out-ports that are not in use, and prevent the instantiation from being more verbose than necessary.

The test sequencer is modified for each test setup, with a few generic startup tests that are run for every configuration. The generic tests include read and write access to the data registers of all instantiated modules in the readout unit, as well as the registers of the ALPIDE model. Figure 5.4 shows an example of support procedure calls in the test sequence. Notice that the calls perform the same checks as the ALPIDE BFM procedures in Figure 5.2a, but the BFM procedures are replaced. All support procedures are described in Appendix D.



```
log(ID_SEQUENCER, "Check periphery register write/read", C_SCOPE);
alpide_write(16X"10", 16X"6F", "Test write, turn off manchester");
alpide_check(16X"10", 16X"6F", "Test read with manchester off");
alpide_write(16X"10", 16X"4F", "Test write, turn back on manchester");
alpide_check(16X"10", 16X"4F", "Test read with manchester on");
```

Figure 5.4: Support procedure calls in test sequencer.

5.2.1 Wishbone BFM

The ITS RUv0a readout unit employs a Cypress FX3 USB chip for communication between the FPGA and a computer. In the simulation, this connection is replaced by a Wishbone BFM. This is accomplished by a `generate` statement in `fabric_top.vhd` that instantiates either the BFM master or the USB module depending on the generic `simulate` variable. The generic variable is provided in the instantiation of `fabric_top`. The Wishbone BFM is provided by Bitvis.

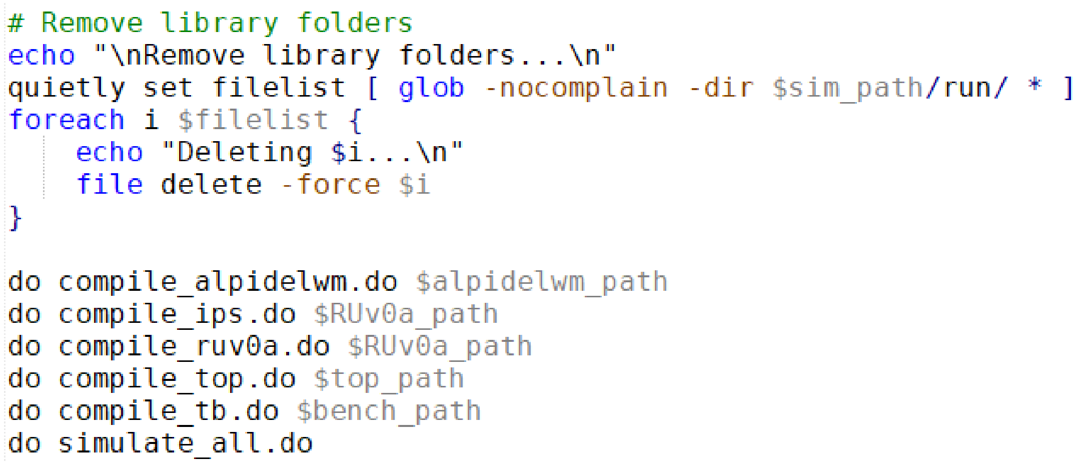
When the bus interface was changed to AXI, the interface between the computer and the FPGA still remained Wishbone. Therefore, the simulation of communication to and from the readout unit still had to be done with Wishbone. When employing another PCB for readout or employing a soft processor for communication, it will be necessary to replace the Wishbone BFM with an AXI BFM. Luckily, this BFM is also provided by Bitvis for free.

5.2.2 Scripts

Several Tcl-scripts were developed to automate the simulation process. They are responsible for a number of tasks:

- Setting up the project
- Compilation of all design IP source files
- Compilation of all design RTL source files
- Compilation of all testbench dependencies
- Compilation of testbench top-level and sequencer
- Starting simulation procedure with correct arguments
- Add relevant components to waveguide for inspection

This is done for several reasons. First of all, to make sure that the simulation is performing the same tests every time it is run. Another effect of these scripts is a significant time efficiency improvement. All scripts were also developed to be used on multiple systems. An excerpt of a script is illustrated in Figure 5.5.



```
# Remove library folders
echo "\nRemove library folders...\n"
quietly set filelist [ glob -nocomplain -dir $sim_path/run/ * ]
foreach i $filelist {
    echo "Deleting $i...\n"
    file delete -force $i
}

do compile_alpidelwm.do $alpidelwm_path
do compile_ips.do $RUv0a_path
do compile_ruv0a.do $RUv0a_path
do compile_top.do $top_path
do compile_tb.do $bench_path
do simulate_all.do
```

Figure 5.5: Excerpt of top-level Tcl-script.

5.2.3 Glitch Issue

Several of the design clocks have a period defined as an infinite decimal. As the simulation does not have an infinite time resolution, this caused several glitches, especially when the firmware employed ISERDESE2 to deserialize the incoming data. This had the effect that the deserialized data frequently featured bitslips, and the readout, therefore, appeared not to work properly. Figure 5.6 illustrates the issue.

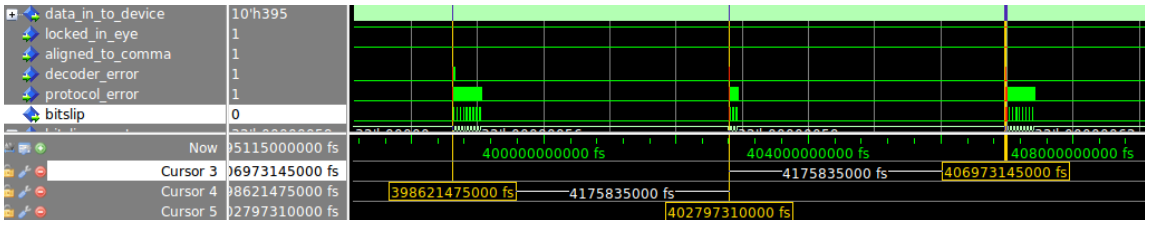


Figure 5.6: Bitslip occurs with a fixed interval caused by clock periods with infinite decimals.

To workaround this issue, the frequencies of all clocks in the design were modified by the same factor so that none had a period of infinite decimals. This caused the simulation to function properly. However, the time units of the simulation will not correctly refer to the real design. Table 5.1 shows the relationship between the design clocks and the simulation clocks.

Table 5.1: Relationship between design clocks and simulation clocks.

Design Clocks		**Simulation Clocks**	
Frequency (MHz)	**Period (ns)**	**Frequency (MHz)**	**Period (ns)**
160	6.25	266.666	3.75
120	8.333	200	5
80	12.5	133.333	7.5
40	25	66.666	15
600	1.666	1000	1

5.3 Test of Readout: Transceiver with Fabric Modules

The ITS firmware was reportedly already tested at CERN. However, it was decided that this test should be replicated at Bergen to confirm the test setup. It was not clear from any documentation how the ITS upgrade team performed its tests, so a new testing scheme had to be developed.

5.3.1 Firmware Modification

Although the firmware was already developed by ITS, some changes were needed to be able to verify the readout. Several methods of testing were evaluated before implementation:

- Masking and enabling certain pixels in the ALPIDE to produce a given data set. To verify the readout, the data must be compared to a fixed data set stored on the FPGA or transferred to a computer. In this stage of the design development, the only communication between the FPGA and the computer was by employing the Wishbone bus. To read out large quantities of data would be slow and not entirely error-proof. Therefore, the only feasible option would be to compare data on the FPGA itself. This method was considered impractical as it would use a large quantity of the memory resources, and the time of implementation would be significant. Also, the ALWM does only provide a simplified data generator and not the pixel matrix. Thus, this method would not be simulatable.
- Sending test vectors from the ALPIDE. The ALPIDE features three registers that can be set to be utilized as test vectors. The ALPIDE can either encode the vectors with 8B10B or bypass this feature. To validate the data, the options

were the same as the previously suggested method; compare the data to a fixed data set on the FPGA or transfer the data to a computer. Sending data to a computer for testing would be slower than the readout from the ALPIDE. Any longer test would therefore create a bottle-neck and require data buffering. Therefore, the preferred method was to test the readout data on the FPGA itself.

The test was then implemented by modifying the protocol checker module developed by the ITS upgrade team. As adding new registers was complicated and time demanding, it was implemented without adding new registers, but by utilizing unused bits in already existing ones. One bit for setting the protocol checker in test mode, one bit for asserting an erroneous data word, and 8 bit for the last word that caused the error. Once an error occurs, the protocol checker stays in the error state until it is reset. The firmware is neither counting the number of correct words nor the number of errors. Consequently, the test software must perform the counting. As an error will put the protocol checker in the error state until it is reset, no counting of consecutive errors will be conducted and there will be a gap in time between each check. It was evaluated that, as long as it was not observed a significant number of errors during testing, this method would be good enough to validate the readout.

5.3.2 Simulation

The test sequencer performs the generic tests described in Appendix D, and then does the following:

- Starts the data module readout initialization. Enables the transceiver by setting and clearing the transceiver reset register. The MGT requires a period of 100 µs after startup before it is ready to handle the data stream, and thus the sequencer waits this amount of time.
- Starts the ALPIDE readout procedure.
  - Sets up PLL
  - Sets up the charge pump
  - Turns on PLL
  - Forces reset
  - Clears reset
  - Writes to the mode control register
  - Activates driving of the local bus, by broadcasting global reset
  - Waits for 2 us for the ALPIDE to send stable comma words
- Activates transmission of constants test pattern from the ALPIDE
- Sets the protocol checker in test mode
- Polls the test registers to confirm that data is read out properly
- Activates transmission of constant erroneous test pattern from the ALPIDE

- Confirms that the protocol checker detects the error and is locked in the error state
- Activates transmission of constant correct test pattern from the ALPIDE
- Resets the protocol checker
- Polls the test registers to confirm that data is read out properly

The simulation therefore confirms not only that readout is successful, but also that the modified protocol checker works as expected. The tests returned no errors.

5.3.3 Hardware Test

The test software, written in Python and communicating over USB, is performing continuous polling of the registers after readout of the test vectors is initialized. The software checks if the data is word aligned, and then if the data is correct. When encountering an error, it resets the protocol checker by turning the test mode off and then on again. The only counting performed is of the number of errors occurred. As discussed, this number will not necessarily be correct, but will rather indicate how often the protocol checker is stuck in the error state. The software will run and continuously poll the registers until the user cancels the operation. If errors have occurred, the number will be indicated together with test duration. If no errors are detected, only the test run time will be presented. All test software classes and functions are listed and described in Appendix E.

The test functionality was evaluated by provoking an error by transmitting erroneous test vectors from the ALPIDE.

Test Setup and Results

The test employed the ITS RUv0a PCB, and thus the Xilinx Kintex 7 FPGA. The FireFly cable is connected to the second connector on the readout unit, in which for the data lines corresponds to one of the Kintex 7's MGTs. The physical setup is equal to that in Figure 5.7, except for the FireFly port employed. Two tests were run for 12 hours: One with a FireFly cable length of 0.3 meters, and one with a 2 meters cable. No errors were detected. The BER was calculated to be less than 19.2×10^{-15}.

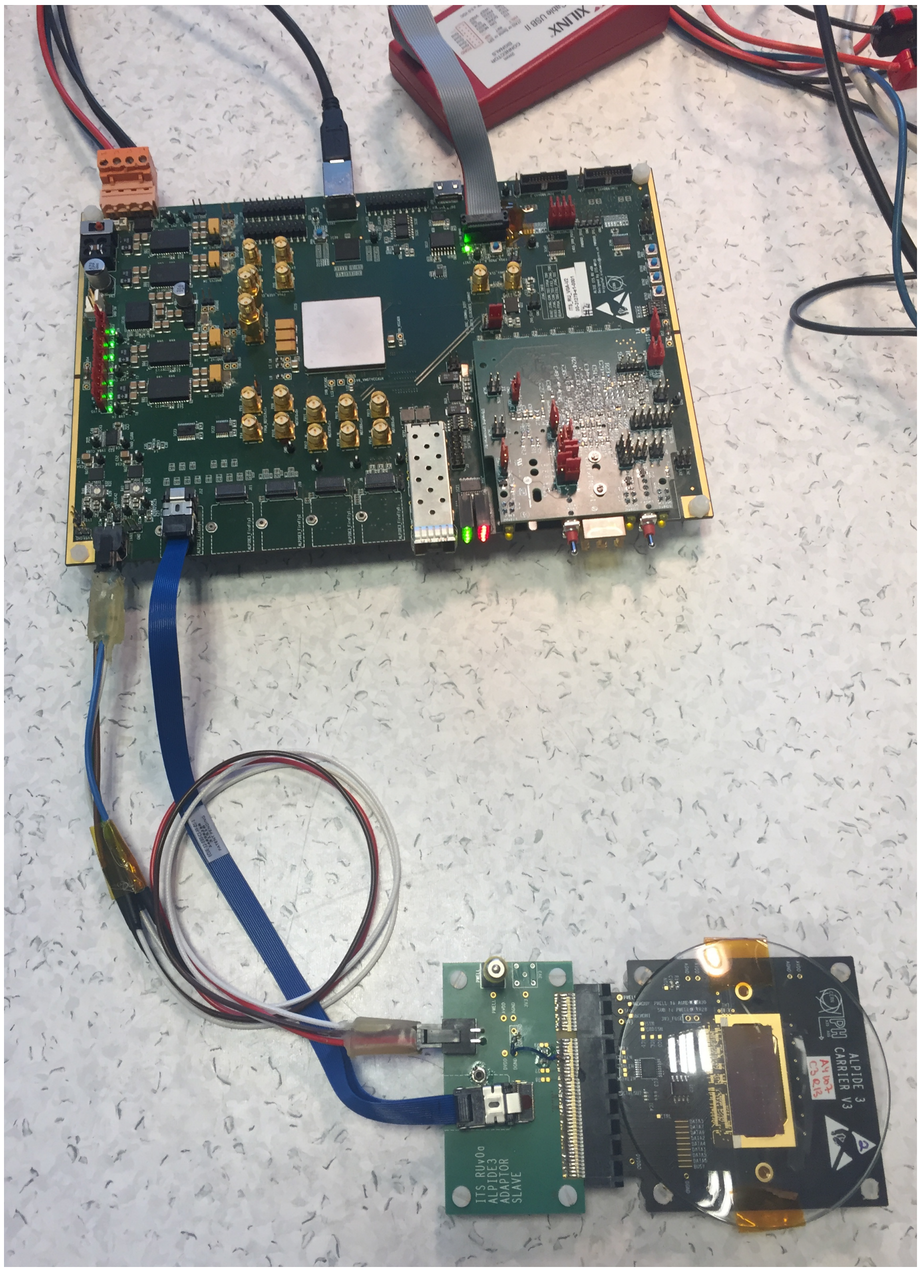


Figure 5.7: ITS Readout Unit card connected to ALPIDE in Inner Barrel mode via adapter to FireFly.

5.4 Test of Readout: Transceiver Only

To simulate and test the modified transceiver approach (discussed in Section 4.5.2), the same procedure as the previous test case was used. The protocol checker is thus connected to the output of the Transceiver Sync FIFO, as illustrated in Figure 4.7, and compares the incoming data to predefined test vectors.

5.4.1 Setup

The transceiver is customized in Vivado by using an MGT wizard found in the IP catalog. The wizard sets the following:

- data line rate
- reference clock frequency
- the specific PLL to employ
- type of decoding
- comma alignment specifications

The wizard can also enable or disable signals to be used for debugging or functionality. E.g., the signal `rxbyteisaligned_out` was enabled and connected to the protocol checker as an enable signal.

The transceiver synchronization FIFO was also generated from the IP catalog in Vivado. Its purpose is to cross from the 60 MHz clock domain which the transceiver output data is referenced, to the 120 MHz clock domain in which the rest of the module is operating. The FIFO was implemented as a block RAM, rather than distributed RAM or built-in FIFO, as this implementation is recommended for different read and write widths. The write width is 16 bit, as is the transceiver output data width, and the read width is 8 bit, as is the required data width of the protocol checker. The write depth was set as the minimum 16. The FIFO is instantiated with an always on read enable signal (except when the FIFO is empty), and therefore, there is no reason to have any higher depth.

5.4.2 Simulation

The test sequence is identical to the previous test setup. The simulation tests returned without errors.

5.4.3 Hardware Test

The test software is identical to the previous test setup. Two tests of 12 hours were run with the same two cables as the previous trials. No errors were detected. The BER was calculated to be less than 19.2×10^{-15}.

5.5 Bus Replacement

Before the bus interface was replaced on all modules, two generic slave modules were generated according to the template sketched out in Section 4.9.2. The purpose of these was to confirm that the AXI bus interface is functioning properly. If at a later stage there are errors after modifying the preexisting modules, it can be ruled out that it is the bus interface itself that constitutes the problem. Two modules were designed to validate the communication to one module without affecting the other.

5.5.1 AXI Interconnect and Wishbone Bridge

During development targeting the RUv0a test board, communication over USB employing the Cypress FX3 chip must be maintained. The Cypress FX3 chip is complex, and the task of developing a new wrapper module to convert the communication to AXI was considered demanding. Therefore, a solution was to convert the Wishbone interface signaling to AXI signaling instead. This type of module is already developed and is accessible as an open source IP.[1] The Wishbone master signals, from the BFM in simulation and the Cypress FX3 wrapper in hardware, were connected to the Wishbone bridge which produces the AXI master signals that can be connected to the AXI interconnect.

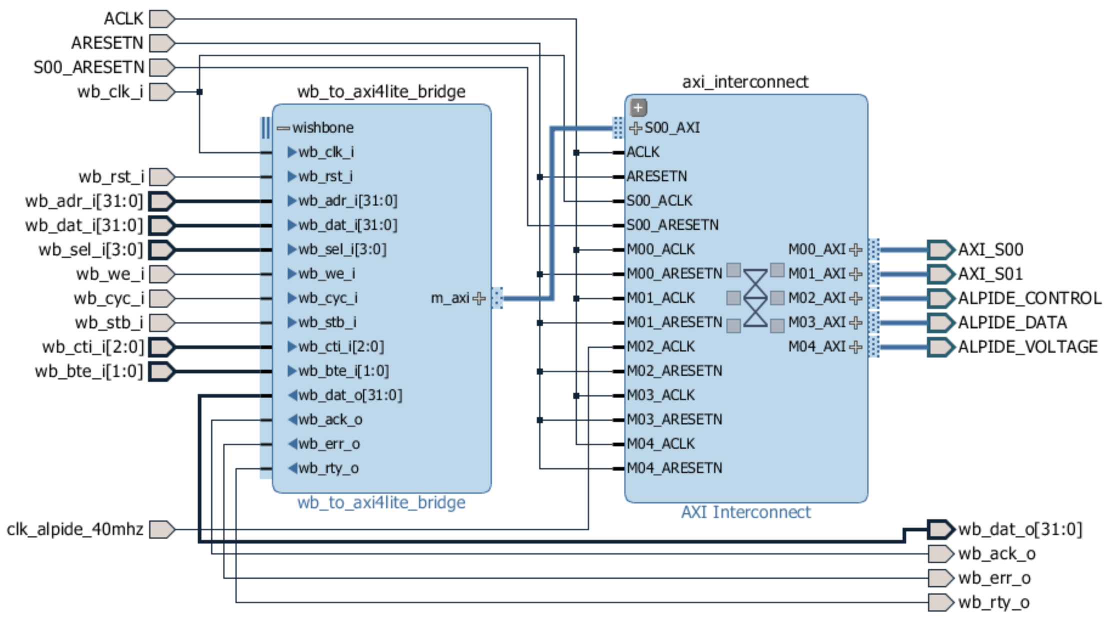


Figure 5.8: Block design diagram of the connection between the Wishbone, AXI and the modules.

[1]The unmodified source for the IP chosen is available at github.com/bluecmd/wb-axi.

Figure 5.8 shows how the Wishbone master bus is connected to the bridge and converted to an AXI interface. The diagram illustrates the situation when all modules had been converted to AXI. The AXI bus signals are shown as a single group of signals. The figure also illustrates the different clock inputs which control the modules. The Wishbone bus operates on `wb_clk_i` and therefore also the AXI master, while ALPIDE control module operates on `clk_alpide_40mhz`. All other modules, including the interconnect block, operates on `ACLK`, which is the 120 MHz clock.

Byte-addressed Memory

The Xilinx AXI Slave module template employs byte-addressed memory, i.e. that each address refers to a byte of data. As the data is stored in a 32-bit register, each register has an address offset of 4. The Cypress FX3 only employs an address vector of 12 bits, where 4 bits refers to a specific module. This will result in 8 register address bits and thus, 256 individual addresses per module. When employing a byte-address memory scheme, the number of registers are reduced to 64. Furthermore, as the testbench environment and the test software targets Wishbone modules with a word-addressed memory scheme (an offset of 1 between each register), the connection between the Wishbone wrapper and the Wishbone bridge left shifts the address bits by two. Consequently, the AXI slave registers will have an offset of 1 between each register and have 256 register addresses available. When the Wishbone bus is entirely eliminated from the system, this workaround should be removed since a typical AXI bus employs a 32-bit address vector, and byte-addressed memory is the standard practice in Xilinx Processor Systems [51].

5.5.2 Verification

The Wishbone-to-AXI scheme was first simulated and verified by using the same simulation environment as the previous tests. The test sequencer was modified to only perform communication with the two generic AXI slaves. The hardware tests were carried out with the software described in Appendix E. When the modules were amended to employ AXI, the same test sequencer and software as the previous tests were employed.

5.5.3 Software

Changing registers in a module requires time and effort, as well as testing to confirm that it has been done properly. To avoid this, a process was started to develop software that automatically produces the required register handler firmware for a given register specification.[2] The work with this software is currently not completed, but the software is partly functional. This means that any changes to the registers will be easily and quickly solved by this software.

5.6 Test of Readout: CDR with I/O Delay Features

To check whether the concept of locking the phase with the I/O delay features was feasible, a test was set up to monitor sample data with a loadable delay tap. The firmware was modified to check for a fixed pattern on the SERDES primitive data output port, and the delay tap was connected to a writable AXI register. Software was written to start a test sequence with a chosen delay tap and to look for a specified data pattern. Data from several delay taps were manually monitored before selecting a delay tap considered to be roughly in the middle of the data eye. The tests were all executed with a FireFly cable of 0.3 meters as this was the only cable length available at the time of execution. Later, when longer cables were acquired (1.0 and 2.0 meters), the eye-diagram measurements were executed.

5.6.1 Test Setup

The test employed the ITS RUv0a PCB, and thus the Xilinx Kintex 7 FPGA. The FireFly cable is connected to the fifth connector (J5) on the readout unit. The data line is connected to pin B17 and C17 on the FPGA. These pins are in a High Range (HR) I/O bank. The speed grade for the given FPGA chip is 2, and the HR I/Os should, therefore, be capable of correctly sampling data with DDR up to a frequency of 1.25 GHz [41, Table 17]. To avoid reflections on the data line, the DIFF_TERM property is set to true for the data I/O pins.[3] This constraint enables a differential termination of 100 Ω on the I/O pin [43]. One weakness with the test setup is that the ALPIDE's data channel is not terminated until the connection with the RUv0a board.

5.6.2 Test Result 600 MHz Clock Signal

A 600 MHz clock signal was generated by the ALPIDE by employing the test vector registers and turning off 8B/10B encoding. The pattern was validated by the firmware.

[2]This work is inspired by similar software products like Progbit's AutoReg and Bitvis' Register Wizard. These products, however, do not support the AXI bus interface.

[3]Tests were also performed with DIFF_TERM set to false which resulted in errors exclusively caused by the deteriorated signal integrity.

After 12 hours of testing, no errors were observed. The BER was calculated to be less than 19.29×10^{-15}.

5.6.3 Test Result 1.2 Gb/s Comma Word

The 1.2 Gb/s comma word is transmitted from the ALPIDE when driving of the output is enabled without enabling test functionality or transmitting TRIGGER commands. The pattern was validated by the firmware. No errors were observed after 12 hours of testing. The BER was calculated to be less than 19.29×10^{-15}.

These results confirmed that there is a negligible drift in the ALPIDE data output compared to the reference clock of the system, and that it was possible to sample high-speed data without employing transceivers. The development of the automatic phase alignment module could therefore continue.

5.6.4 Eye Diagram

An eye diagram measurement is a method to determine the electric quality of a signal [56]. The eye diagrams on page 79 measure the electric quality of the high-speed data line from the ALPIDE on the RUv0a PCB. The data signal is probed with a 1 GHz differential probe on the termination resistors connected to the FireFly input connectors on the readout unit (as seen lower left corner of the readout unit in Figure 5.7[4]). The ALPIDE is outputting a pseudo-random bit pattern, and the oscilloscope is triggering on rising edges of the data. The horizontal delay is set to 10.5 ns, so the produced eye period is the 12th unit interval after the edge being triggered. The oscilloscope is measuring several values where the following are relevant:[5]

Crossing %: The percentage between high or low the signal is crossing. It is, therefore, a comparison of the rise and fall time, and also a duty cycle distortion measurement.

Eye height: The vertical opening of the eye in the center of the unit interval. Measured in volts.

Eye width: The measure of the horizontal opening of the eye in ps. Calculated by measuring the difference between the statistical mean of the crossing points of the eye.

[4]The resistors are labeled R227 and R239 on the RUv0a schematic [33].

[5]Eye top is only used in extinction ratio measurements. This measurement is visible as the line and the two arrows in the figures.

Table 5.2: Eye diagram measurements.

Cable Length (m)	**Cross %**	**Eye Height (mV)**	**Eye Width (ps)**
0.3	51.49 ± 0.3	309.8 ± 0.2	405.1 ± 0.6
1.0	50.720 ± 0.006	298.9 ± 0.1	434.5 ± 3
2.0	51.06 ± 0.01	190.3 ± 0.2	443.7 ± 0.3

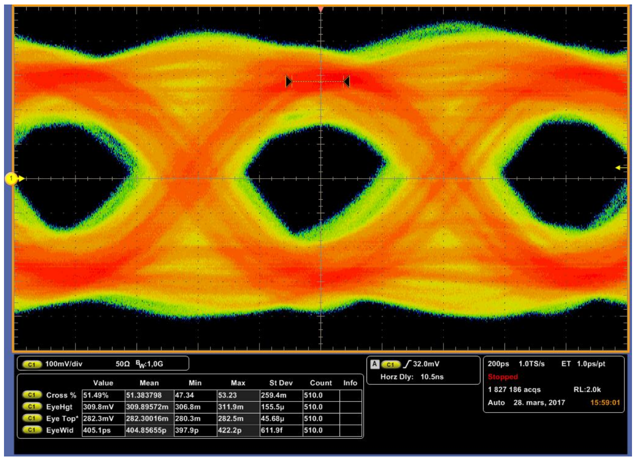
These measurements show how a longer cable attenuates the signal. This is to be expected, as a longer cable inevitably has more series and shunt-resistive impedances, and a lossy line is the biggest problem for high-speed data transfer on long transmission lines (see Section G.0.1). This affects the eye-height, and thus degrades the quality of the eye. The eye width, however, is actually increasing with increasing cable lengths. This may be due to measurement weaknesses, and not due to an actual enhancement of the signal integrity.

Measurement Weaknesses

The signal was measured with a differential probe with a bandwidth of 1 GHz. The bandwidth, the highest significant sine-wave-frequency component, of a clock signal can be estimated to be roughly five times the clock repeat frequency [39]. Therefore, the highest component can be assumed to be approximately 6 GHz. The result may be that the probe operates as a low-pass filter and that the higher frequency components are not sampled correctly. This may cause the oscilloscope to inaccurately illustrate the signal as less square than it is, and in effect render an eye that is smaller than in reality. Moreover, the measurements may not explain the reasons for errors at longer cable lengths (see Table 5.3). When the signal is illustrated as less square than in reality, the measured eye width will be incorrect. In fact, an attenuation of the signal can result in an increasing eye width because the difference between a high and a low bit is smaller. The amount of time spent in transition will, therefore, be reduced as well, resulting in a wider eye.

The effect that is not seen is that higher frequency components are usually attenuated more than lower frequency components [39]. The shorter cable should have produced a more square-like signal than the measurement shows. Furthermore, the ringing due to reflections usually consists of higher frequency components. If this type of ringing occurs, it is not detected by the measurement.

However, the measurements still prove useful for comparisons of parts of the signal quality with different cable lengths, and, as discussed, illustrates the attenuation of the signal with longer cables, although not the frequency dependent attenuation.

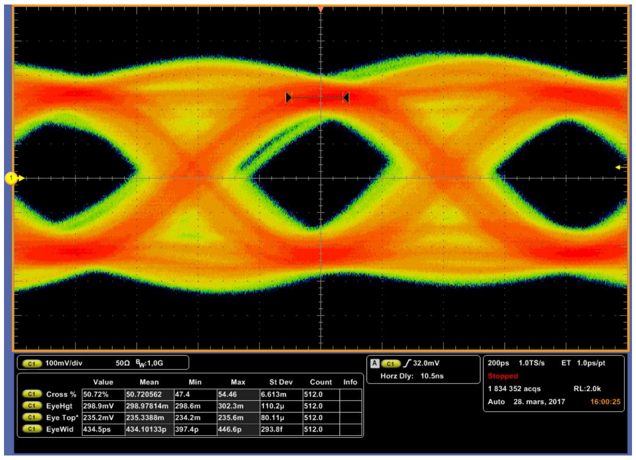

C1 100mV/div 50Ω BW:1,0G

	Value	Mean	Min	Max	St Dev	Count	Info
C1 Cross %	51.49%	51.383798	47.34	53.23	259.4m	510.0	
C1 EyeHgt	309.8mV	309.89572m	306.8m	311.9m	155.5µ	510.0	
C1 Eye Top*	282.3mV	282.30016m	280.3m	282.5m	45.68µ	510.0	
C1 EyeWid	405.1ps	404.85655p	397.9p	422.2p	611.9f	510.0	

A C1 32.0mV
Horz Dly: 10.5ns

200ps 1.0TS/s ET 1.0ps/pt
Stopped
1 827 186 acqs RL:2.0k
Auto 28. mars, 2017 15:59:01

Figure 5.9: The eye diagram of the signal with 0.3m cable.

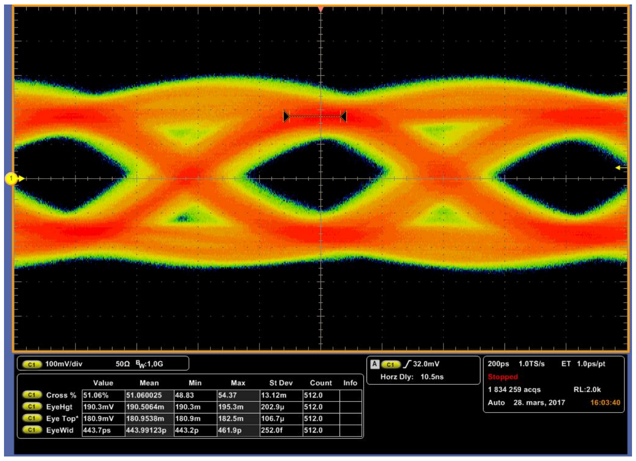

C1 100mV/div 50Ω BW:1,0G

	Value	Mean	Min	Max	St Dev	Count	Info
C1 Cross %	50.72%	50.720562	47.4	54.46	6.613m	512.0	
C1 EyeHgt	298.9mV	298.97814m	298.6m	302.3m	110.2µ	512.0	
C1 Eye Top*	235.2mV	235.3388m	234.2m	235.6m	80.11µ	512.0	
C1 EyeWid	434.5ps	434.10133p	397.4p	446.6p	293.8f	512.0	

A C1 32.0mV
Horz Dly: 10.5ns

200ps 1.0TS/s ET 1.0ps/pt
Stopped
1 834 352 acqs RL:2.0k
Auto 28. mars, 2017 16:00:25

Figure 5.10: The eye diagram of the signal with 1.0m cable.


C1 100mV/div 50Ω BW:1,0G

	Value	Mean	Min	Max	St Dev	Count	Info
C1 Cross %	51.06%	51.060025	48.83	54.37	13.12m	512.0	
C1 EyeHgt	190.3mV	190.5064m	190.3m	195.3m	202.9µ	512.0	
C1 Eye Top*	180.9mV	180.9538m	180.9m	182.5m	106.7µ	512.0	
C1 EyeWid	443.7ps	443.99123p	443.2p	461.9p	252.0f	512.0	

A C1 32.0mV
Horz Dly: 10.5ns

200ps 1.0TS/s ET 1.0ps/pt
Stopped
1 834 259 acqs RL:2.0k
Auto 28. mars, 2017 16:03:40

Figure 5.11: The eye diagram of the signal with 2.0m cable.

5.7 Test of Readout: Automatic Phase Alignment

5.7.1 Simulation

To check the automatic phase alignment behavior from Section 4.6, the test sequencer was expanded to:

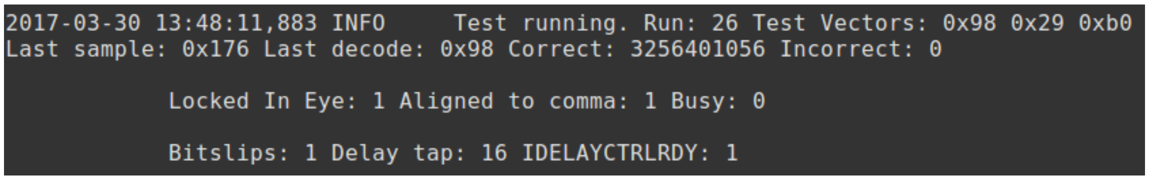
1. Enable the ALPIDE readout channel. This initiates the COMMA word transmission.
2. Start the phase alignment process by setting the `run` register.
3. After a delay of 50 µs, confirm that `locked_in_eye` and `aligned` registers have been set.
4. Control that `incorrect_cnt` is zero and that `correct_cnt` is higher than zero.

To provoke errors, the phase alignment process was also initiated without enabling the ALPIDE readout. As there are no commas on the data line, the expected behavior of the phase aligner is to never lock to a delay tap. This was the exact behavior that was observed.

The design was also validated by observing the waveguide in Figure 5.12. Note that, in the simulation environment there is only one single delay tap that is considered invalid. This happens because the change of this particular delay tap causes a bitslip.

5.7.2 Hardware Test

New software was developed for the hardware tests. The software is controlling the same registers as the test sequencer, as well as the last sampled and decoded data, the chosen delay tap and bitslip count. All available information is output to the screen and to log file. A sample log entry is shown in Figure 5.13. See Appendix E for more details.



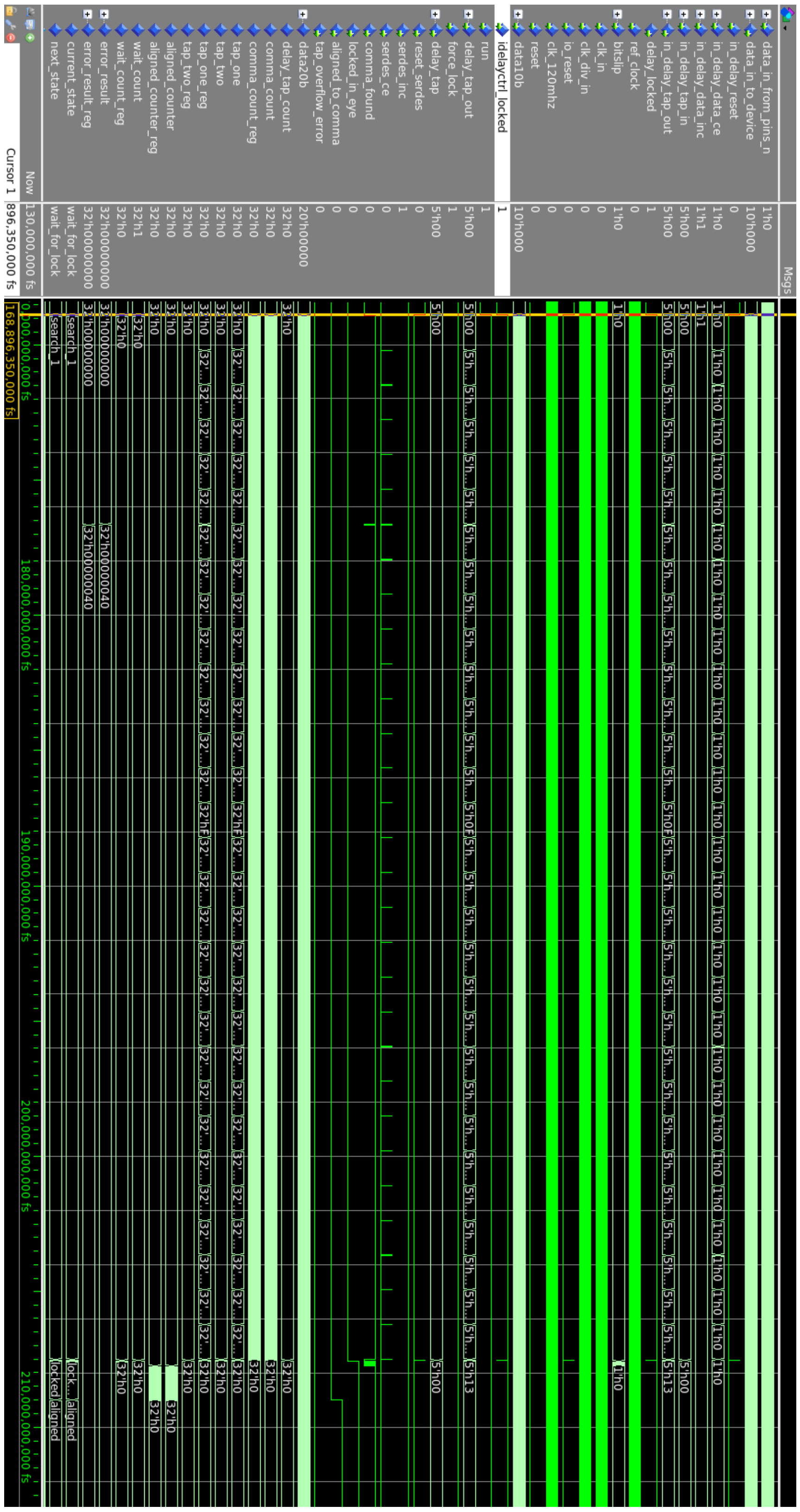
```
2017-03-30 13:48:11,883 INFO     Test running. Run: 26 Test Vectors: 0x98 0x29 0xb0
Last sample: 0x176 Last decode: 0x98 Correct: 3256401056 Incorrect: 0

           Locked In Eye: 1 Aligned to comma: 1 Busy: 0

           Bitslips: 1 Delay tap: 16 IDELAYCTRLRDY: 1
```

Figure 5.13: Sample software log entry during testing.

Testing was performed with a randomized time between each reset and initialization phase. The ALPIDE was power cycled between each run, and also full reset procedure



```
                       Msgs
data_in_from_pins_n    1'h0
data_in_to_device      10'h000
in_delay_reset         0
in_delay_data_ce       1'h0
in_delay_data_inc      1'h1
in_delay_tap_in        5'h00
in_delay_tap_out       5'h00
delay_locked           1
ref_clock              0
bitslip                1'h0
clk_in                 0
clk_div_in             0
io_reset               0
clk_120mhz             0
reset                  0
data10b                10'h000
idelayctrl_locked      1
run                    1
delay_tap_out          5'h00
force_lock             1
delay_tap              5'h00
reset_serdes           0
serdes_inc             1
serdes_ce              0
comma_found            0
locked_in_eye          0
aligned_to_comma       0
tap_overflow_error     0
data20b                20'h00000
delay_tap_count        32'h0
comma_count            32'h0
comma_count_reg        32'h0
tap_one                32'h0
tap_two                32'h0
tap_one_reg            32'h0
tap_two_reg            32'h0
aligned_counter        32'h0
aligned_counter_reg    32'h0
wait_count             32'h1
wait_count_reg         32'h0
error_result           32'h00000000
error_result_reg       32'h00000000
current_state          wait_for_lock
next_state             wait_for_lock
Now                    130,000,000 fs
Cursor 1               896,350,000 fs
```

Figure 5.12: Simulation of the automatic phase alignment.

Table 5.3: Automatic phase alignment test results.

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**
Test runs	1000	10	1000	10	1000	10
Length of each test (s)	60	21 600	60	21 600	60	21 600
Total time (s)	60 000	216 000	60 000	216 000	60 000	216 000
Cable length (m)	0.3	0.3	1.0	1.0	2.0	2.0
Temperature	Not supervised	Not supervised	Not supervised	Not supervised	Not supervised	Not supervised
Correct words	7.2×10^{12}	25.9×10^{12}	7.2×10^{12}	25.9×10^{12}	7.2×10^{12}	25.9×10^{12}
Incorrect words	0	0	0	0	**8306**	0
Runs with errors	0	0	0	0	14	0
Bitslips	9000[a]	82[a]	9059[a]	12[a]	1000[b]	10[a]
BER [c]	$< 13.0 \times 10^{-15}$	$< 3.9 \times 10^{-15}$	$< 13.0 \times 10^{-15}$	$< 3.9 \times 10^{-15}$	1.2×10^{-11}	$< 3.9 \times 10^{-15}$

[a] All bitslips occurred during word alignment and do not indicate an error.
[b] The connection between protocol checker and bitslip module is removed since it produced a lot more errors.
[c] Since the check is done on words, not bits; it is assumed that each incorrect word consists of a single bit error.

was performed. Each run has a fixed set of test vectors sent from the ALPIDE, but they are randomized between each run. Tests were run without supervision of temperature and humidity. Further tests should be conducted where these factors are considered.

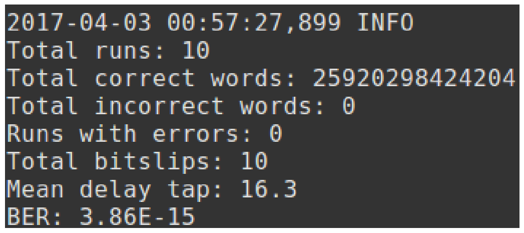

```
2017-04-03 00:57:27,899 INFO
Total runs: 10
Total correct words: 25920298424204
Total incorrect words: 0
Runs with errors: 0
Total bitslips: 10
Mean delay tap: 16.3
BER: 3.86E-15
```

Figure 5.14: Summary of test results of Test 6.

The test results in Table 5.3 clearly prove the concept of phase and word alignment with the developed firmware. The bitslips are, as expected, only asserted in the word alignment process and are never needed later. The internal signals can be analyzed by employing an Integrated Logic Analyzer (ILA), which is described in Section H.4.1. By viewing the internal signals (see Figure 5.15), we see that the consecutive valid delay taps are divided into two groups with the invalid delay taps arranged in the middle. Therefore, choosing a delay tap in the middle of the longest consecutive streak may not be the best choice. An improved algorithm which also looks for

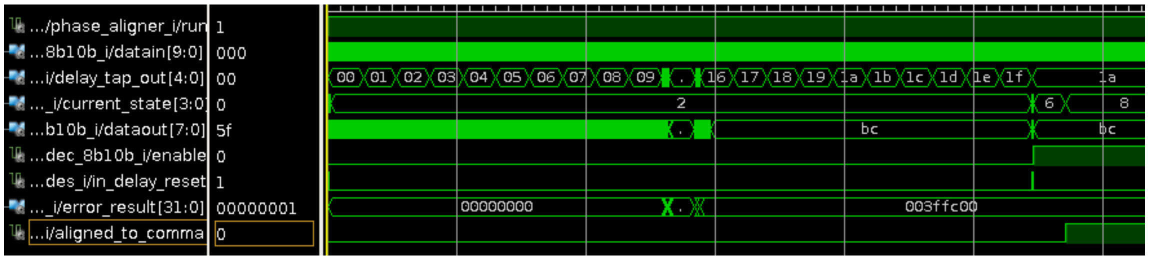

.../phase_aligner_i/run 1
...8b10b_i/datain[9:0] 000
...i/delay_tap_out[4:0] 00
..._i/current_state[3:0] 0
...b10b_i/dataout[7:0] 5f
...dec_8b10b_i/enable 0
...des_i/in_delay_reset 1
..._i/error_result[31:0] 00000001
...i/aligned_to_comma 0
00 01 02 03 04 05 06 07 08 09 16 17 18 19 1a 1b 1c 1d 1e 1f 1a
2 6 8
bc bc
00000000 003ffc00

Figure 5.15: Internal signals during the automatic phase and word alignment with a 0.3m long cable. Measurement is done *before* clock phase shift. Obtained by ILA (see Section H.4.1).

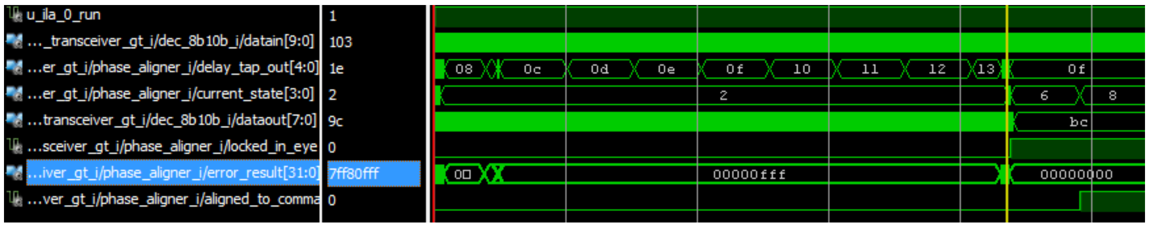

u_ila_0_run 1
..._transceiver_gt_i/dec_8b10b_i/datain[9:0] 103
...er_gt_i/phase_aligner_i/delay_tap_out[4:0] 1e
...er_gt_i/phase_aligner_i/current_state[3:0] 2
...transceiver_gt_i/dec_8b10b_i/dataout[7:0] 9c
...sceiver_gt_i/phase_aligner_i/locked_in_eye 0
...iver_gt_i/phase_aligner_i/error_result[31:0] 7ff80fff
...ver_gt_i/phase_aligner_i/aligned_to_comma 0
08 0c 0d 0e 0f 10 11 12 13 0f
2 6 8
bc
00 00000fff 00000000

Figure 5.16: Internal signals during the automatic phase and word alignment with a 2.0m long cable. Measurement is done *after* clock phase shift. Obtained by ILA.

a possibility of a combination of the two groups should be considered developed. However, by phase shifting the 600 MHz clock by 90 degrees, the valid taps were moved to the middle of the `error_result` vector. This effect was observed with all cable lengths, and can thus be considered a solved problem. Figure 5.16 shows the internal signals after the phase shift of the clock.

To further explore the effects of different cable lengths, many observations were made of the internal signals of the phase aligner module. The number of valid taps was drastically reduced by increasing the cable length from 0.3 m to 2.0 m. This alone tells us that there is a limit to the length of cables that can be handled by the I/O primitives alone. The results are also in agreement with the eye diagram measures performed earlier. Ten measurements of each cable gave the following results:

- 0.3 m cable: 16 ± 1 valid taps
- 1.0 m cable: 14 ± 1 valid taps
- 2.0 m cable: 8 ± 2 valid taps

Lessons:

- The protocol checker may only need access to the bitslip module in test mode.
- An improved algorithm to combine two groups of valid delay taps should be developed.
- A 90 degree phase shift of the 600 MHz clock centers the valid delay taps for all cable lengths tested.

- Longer cables distort the eye diagram and decreases the number of valid taps.
- Errors are introduced at a cable length of 2 m.

Further testing

As errors were introduced when employing a 2 m cable, a longer test was performed to measure the exact rate of errors occurred, and especially the number of test runs with errors. The previous tests consisted of either 1000 test runs of 60 seconds data streaming, or 10 test runs of 60 hours data streaming. A new test was performed with 7500 test runs of 30 seconds of data streaming, a total test time of 62.5 hours. The percentage of test runs that contained errors were 2.01 %. The BER was calculated to be less than 3.18×10^{-9}.

The software was modified to log the time of the first error occurred to explore the concept of using software to remove a portion of the errors, as discussed in Section 4.11.1. 10 000 test runs of 120 seconds were executed, equivalent to two weeks of running time. That amount of test time runs was required to generate a sufficient amount of error entries. This produced the histogram of detected errors in Figure 5.17.

Qualitatively the results show that errors are often detected quite early in the process. By integrating the histogram from 0 to 60 and dividing by the number of entries, we observe that over 85 % of the errors occur in this interval. This tells us that there may be possibilities for designing a start-up procedure that can prevent a high portion of the bad initializations as discussed in Section 4.11.1.

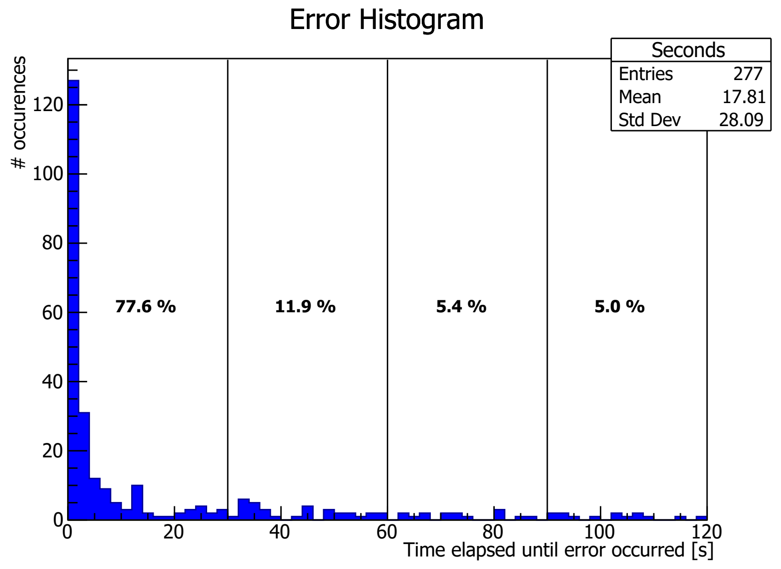


Figure 5.17: Histogram of detected errors.

5.7.3 Resource Usage

Table 5.4 shows us that the required LUTs for all data modules on the entire pCT readout unit is only 16 % of the total number of LUTs available on the biggest UltraScale+. Even the smallest UltraScale+ would have enough LUTs if only the data modules were to be implemented. Observe that an even lower fraction of registers is needed to implement the design. These numbers are only crude calculations, however, and an actual implementation may require far more resources.

Note also that the measured data module did not have the required buffer FIFOs implemented, and thus the memory usage of this critical part of the design is not considered.

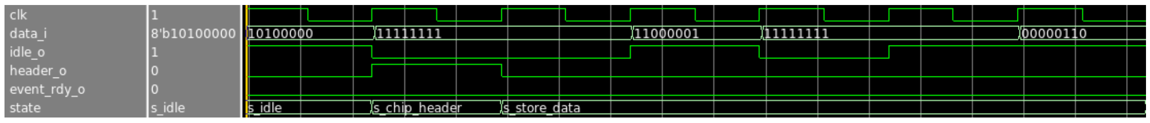
Table 5.4: Resource utilization of data sampling modules.

	Slice LUTs	**Slice Registers**
AXI Register Module	271	303
Bitslip	41	37
8B10B Decoder	21	0
Phase Aligner	918	120
Protocol Checker	138	62
Total	1724	701
×162[a]	279 K	114 K
Percentage of largest UltraScale+[a]	16%	3%

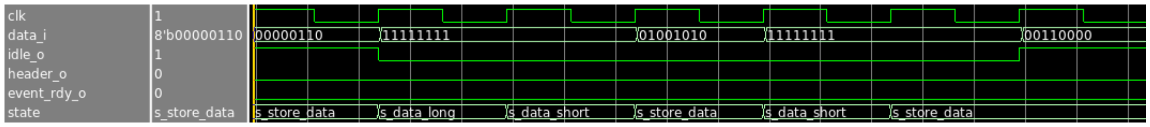
[a] Calculated value only. Implementation necessary to obtain accurate number.

5.8 Data Filtering Design Simulation

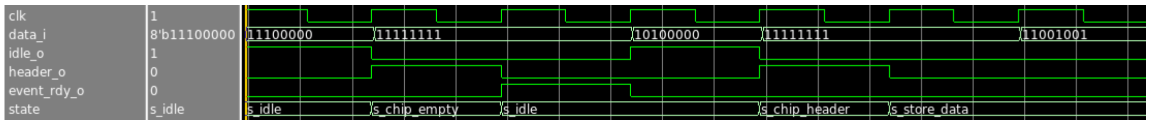
The protocol checker was translated from System Verilog to VHDL to make it consistent with the rest of the design files. Later it was expanded to include the features described in Section 4.7.4. To check whether this design behaved as intended, the design was simulated and inspected. The ALWM was provoked to output IDLE words as part of the chip header, data long, and data short words. This was done to investigate whether the protocol checker managed to avoid asserting the `idle` signal in these situations. Figure 5.18 shows the waveguide for this simulation during different circumstances.



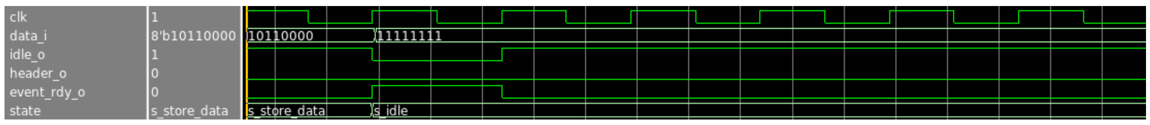
(a) Chip header and region header.



(b) Data long and data short.



(c) Empty chip header.



(d) Chip trailer.

Figure 5.18: Simulation of Protocol Checker behavior. Shows provoked IDLE words as part of event data.

Figure 5.18a shows the behavior of the protocol checker when observing a chip header and a region header word. The `header` signal is asserted as a consequence of the chip header word. The IDLE word occurring directly after the chip header word is ignored, because it may be part of event data. However, we can observe that the `idle` signal is asserted after the next idle word. This happens because the chip header is only 16 bit long, and the next idle word does not contain any data. The idle word occurring directly after the region header also produces a high `idle` signal.

Figure 5.18b shows the behavior after data long and data short. Notice that `idle` is not asserted for the two IDLE words after data long, and not until the second idle word after data short.

Figure 5.18c shows how `header` is asserted directly after the empty chip header word, and `idle` is not asserted until the second idle word. The empty chip header also causes `event_rdy` to go high. Figure 5.18d shows how a chip trailer word effectively raises the `event_rdy` signal. The simulation, therefore, confirms that our design is correct.

CHAPTER 6

Discussion and Conclusion

This chapter starts with a discussion of the results of Chapter 5. Then it offers an outline of the topics that needs to be addressed in the future to create a complete working readout system.

6.1 Readout Evaluation

Three clock data recovery methods were tested in Chapter 5. All tests of the systems that employed MGTs resulted in zero errors. However, when employing I/O features, errors are introduced when using 2-meter cables. The eye diagram measurements also showed an attenuation of the signal with longer cables. The number of errors is not alarmingly high, but may be sufficient to represent a problem in the final pCT prototype. The exact reasons for the errors have not been identified, but there are reasons to believe that the increased attenuation of longer cables sufficiently distorts the data eye. What is probably due to the equalizing stage of the MGTs, this attenuation does only result in errors when employing regular I/Os.

Some errors may be removed by implementing a power-on sequence where software controls that all data channels produce correct data. This could be observed in the histogram in Figure 5.17 where most of, but not all of the bad delay tap initializations, were detected before 60 seconds had elapsed. The distribution indicated that it was somewhat random when the first error word occurred.

The test system itself may have some weaknesses that are the main cause of the number of errors. One particular weakness that can be improved is the lack of termination of the differential data line on the ALPIDE carrier card. New and improved versions of this design are currently being developed at CERN and should become available for testing in Bergen. This new card also has a direct connection to FireFly, so no adaptor card is needed, which will further improve the signal integrity of the channel.

The I/O pins of the FPGA has a DDR bandwidth limit of 1.25 Gb/s, which means our device is operating close to this limit. This may not be the case of the FPGA employed in the final design.

The UltraScale+ devices use a new architecture for the I/O ports. The component mode is compatible with older designs from the 7-series and uses primitives like ISERDES and IDELAY, but the primitives have significant feature reductions. In this mode, the UltraScale+ devices only support the same sampling bandwidth as the 7-series FPGAs. However, the component mode IDELAY primitive has a total of 512 delay taps compared to the 32 delay taps of the 7-series IDELAY. This allows the logic to choose a far more accurate sampling point. On the other hand, in native mode, the Virtex UltraScale+ supports DDR sampling up to 1600 Mb/s. Native mode is part of the new UltraScale architecture and refers to a sampling of data with native primitives like RX_BITSLICE instead of IDELAY and ISERDES. That means that a future design may operate with a larger safety margin from the bandwidth limit. However, this may require redesigning the phase alignment system because of the usage of other primitives than the current design. The feature reduction of the component mode primitives may be a problem when employing the present design on the new FPGA chip.

When all this is said, the errors were only observed when using a 2-meter long FireFly cable. The final design may not need a cable of this length, and may even use another interconnect approach entirely (as mentioned in Section 4.3.3). The work done has still confirmed the possibility of using regular I/Os for sampling data from the ALPIDEs.

6.2 Design Evaluation

The ITS developed module for communication with the ALPIDE works as expected, also after being modified to use the AXI interface instead of the Wishbone interface.

Section 4.10.1 discussed the various ways to initiate an event window on the ALPIDEs, and it was argued that an initial TRIGGER command should be controlled from the SCU. However, it was not concluded whether the SCU should produce all the following TRIGGER commands as well. That being said, there are no real reasons for not using the ALPIDEs themselves for this purpose. The length of a strobe window must be programmed in an ALPIDE register, and it is trusted that this will produce the same length on all chips in the system. Thus, one may be confident that all strobe windows will start at the same time when employing the ALPIDEs' internal trigger sequence, in contrast, to externally provide trigger signals.

One technique of handling the busy state of an ALPIDE chip implemented by the ITS firmware is to filter TRIGGER commands. When the ALPIDEs themselves control the event window, this feature will cease to have an effect. That being said, the busy state is an unwanted effect of either too many event windows in a too small time interval or a too active beam source. Therefore, the busy state should rather be handled by setting an appropriate strobe length and gap or by changing the nature of the beam. If already stored data is regarded as more important than new data the ALPIDE chips should be set in *triggered* mode, rather than *continuous* mode. The mode selected will not have any damaging effects on the proposed filtering, tagging and buffering design.

6.3 Future Work

There are quite a lot of tasks that need to be addressed before a fully functional pCT readout system is realized.

The automated phase and word alignment scheme must be tested with the UltraScale+ FPGA. This includes performing the changes necessary to employ the current design in the component mode for the I/O ports. If errors are still observed with 2-meter cable, the system can be updated to the new native mode which will provide a larger safety margin. To test the new chip, a new physical test setup is required. The VCU118 evaluation kit provides a UltraScale+ chip, as well as several other features that may prove useful in a test setup, including a Samtec FireFly interface. However, that interface is directly connected to MGTs on the FPGA. To test the I/O pins, it is necessary to develop a breakout board that can be connected to the FMC connector on the evaluation board.

When the readout functionality is working on the UltraScale+ chip, the tests should also be performed in an environment where the temperature and humidity are controlled. A test should also be completed in a radiation environment to test susceptibility to SEUs on both the ALPIDEs and the readout unit.

To complete the design of the tagging and buffering stage it may be helpful to extend the simulation environment with multiple instances of the ALWM. This is also required for the building of the offloading stage where data from multiple modules are being combined. A complete tagging stage will also require a complete definition of the data format.

The final design may not employ all available MGTs in the offloading stage. The remaining transceivers should be put to use in the data recovery stage. The data module should, therefore, be expanded to have the possibilities of employing both I/O ports and transceivers. This functionality should be easily configurable with a VHDL generic.

Section 4.9.3 outlined different tasks a soft processing system could potentially handle. This system must be implemented on the UltraScale+ chip. It must be explored whether this system should be powered by an operating system, with or without RTOS (Real-Time Operating System) features. E.g., the timestamp creation for the data tagging process could potentially be associated with a given time requirement. Software for the tasks run on the processor must also be developed. This includes software that controls the phase alignment process and ensures that errors detected within a fixed amount of time (e.g. 60 seconds), disqualifies the delay tap and reruns the initialization.

An important part of the proposed pCT readout system is the SCU. This unit must, therefore, be entirely defined and developed. One aspect of this is the power control and monitoring. This will also require the development of firmware and hardware on the readout unit.

To help aid in future creation and modification of AXI modules, it will be helpful to finalize the AXI register wizard software. Several features can be added, e.g. automated creation of C header files and Python packages for software communication.

Finally, the task of designing the PRUdense and SCU PCBs must be accomplished. Prior to this, all firmware must be verified in both simulation and hardware setups. Furthermore, it must be concluded whether the Ethernet links can offload data in real-time or if onboard RAM blocks are required for buffering the data.

6.4 Conclusion

This thesis has presented a potential design solution for the pCT readout system. The stages of the readout process are defined, and the need for two system units has been identified; the PRUdense and the SCU. Parts of the PRUdense have been developed and tested when employing a 7 Series Kintex on the ITS RUv0a test board.

The testing of the readout shows no errors when employing I/O features with a shorter FireFly connection cable, but some errors are introduced with a longer cable. However, the results overall are arguably positive and prove the concept of automated phase and word alignment with use of Xilinx I/O primitives. Suggestions for preventing a portion of the errors are also discussed, as well as improving the signal integrity of the readout channel itself.

APPENDIX A

ALPIDE BFM

Complete documentation is available in HTML format.[1]

Procedures

Write

`alpide_multicast_write` Write to all chips connected to slow control. Grabs control over bus then sends opcode, multicast ID, reg addr H and L, and data H and L. Sends random gap between each word.

`alpide_unicast_write` Write to a single chip by employing chip ID. Grabs control over the bus, then sends opcode, chip id, reg addr H and L, and data H and L. Sends random gap between each word.

`alpide_broadcast_grst` Sends global reset command to all chips. Grabs control over the bus and sends GRST opcode. Ends with a random gap.

`alpide_broadcast_prst` Sends pixel matrix reset to all chips. Grabs control over the bus and sends PRST opcode. Ends with a random gap.

`alpide_broadcast_pulse` Sends pixel matrix pulse to all chips. Grabs control over the bus and sends PULSE opcode. Ends with a random gap.

`alpide_broadcast_rorst` Sends readout (RRU/TRU/DMU) reset. Grabs control over the bus and sends RORST opcode. Ends with a random gap.

Read/check

`alpide_unicast_read` Reads data from a specific ALPIDE chip. Grabs control over the bus and sends RDOP opcode. The chip ID, reg addr L and H are transmitted with random gaps between each word. The bus is turnaround

[1] github.com/olagrottvik/alpidebfm

before receiving the chip id and the data. The bus is then turnaround once again.

`alpide_check` Uses alpide_unicast_read and then compares the received data with a provided input word.

Support

`send_char` Sends a 8-bit character on DCTRL. Grabs the bus, waits for 1/4 clock period after rising edge, and then asserts data bits every clock cycle. Then releases bus.

`send_reset` Assert the reset signal for time_reset. Wait for time_idle before deasserting.

`send_gap` Grab the bus for 1 clock cycle.

`send_random_gap` Grab the bus between min and max clock cycles.

`bus_turnaround` Gives the bus control to/from the ALPIDE. When giving the control to ALPIDE: take control of the bus for 5 clock cycles, deassert the drive signal, assert the read signal. Chooses sampling point. Checks that chip drives the bus and wait for 5 clock cycles. When taking the control back from ALPIDE: deassert the reading signal and wait for 5 clock cycles. Assert the drive signal and wait for 5 clock cycles.

`receive_char` Receive a 8-bit character on DCTRL. Wait until rising_edge, check if dctrl_in is START_BIT. Otherwise loop until this occurs. Then wait for 1 clock period and sample dctrl_in 8 times while shifting the data into variable. Confirm that last bit is STOP_BIT.

`alpide_init_continuous` Set up the ALPIDE in continuous mode.

`alpide_init_triggered` Set up the ALPIDE in triggered mode.

Types

```
type t_alpide_bfm_config is   record
        max_wait_cycles           : integer;
        max_wait_cycles_severity : t_alert_level;
        clock_period              : time;
        id_for_bfm                : t_msg_id;
        id_for_bfm_wait           : t_msg_id;
        id_for_bfm_poll           : t_msg_id;
end record;

type t_alpide_if is record
        rst_n   : std_logic;
        dctrl   : std_logic;
        chipid  : std_logic_vector(6 downto 0);
        drive   : std_logic;
        reading : std_logic;
end record t_alpide_if;
```

APPENDIX B

Firmware

This chapter briefly outlines selected modules, processes and procedures developed in this work. Complete documentation is available in HTML and PDF format in the pCT repository.

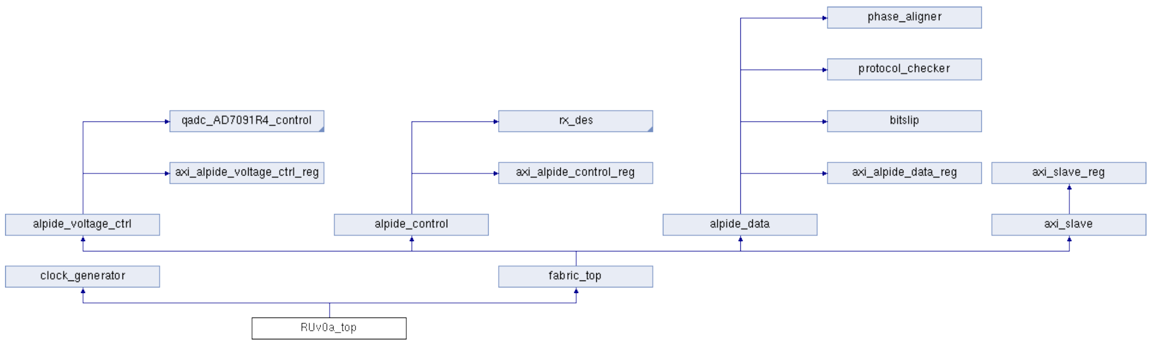


Figure B.1: Instantiation diagram of selected modules in the firmware design. Generated by Doxygen. Note that only VHDL-modules are parsed, and not IP nor Verilog instantiations. AXI, Wishbone and USB interconnect and bridge modules are excluded.

alpide_data

The data module depicted in Figure 4.3. CDR, filtering, tagging and buffering of ALPIDE data.

Processes

`count_error_p ( clk_120mhz , reset_serdes )` Process for counting errors. Compares the incoming data stream to comma or predefined testvectors. Also assign last sampled and last decoded words to registers.

`upcounter_p ( clk_120mhz , reset )` Counts events indicated by Protocol Checker.

Instantiations

`io_serdes_i` Instantiate the I/O primitives for data sampling.
`phase_aligner_i` For accomplishing phase and word alignment.
`dec_8b10b_i` Decodes the 10-bit output from the SERDES into a 8-bit word. Only active when locked_in_eye is asserted.
`protocol_checker_i` Controls that the data adheres to the ALPIDE protocol. Is also used to for data filtering, tagging and buffering.
`bitslip_i` Controls the bitslip pulse based on 8b10decoder and protocol checker. Makes sure that the bitslip is not asserted too often.
`axi_alpide_data_reg_i` AXI Register Handler.

phase_aligner

Search for comma and increment throguh IDELAY taps to determine data eye.

Types

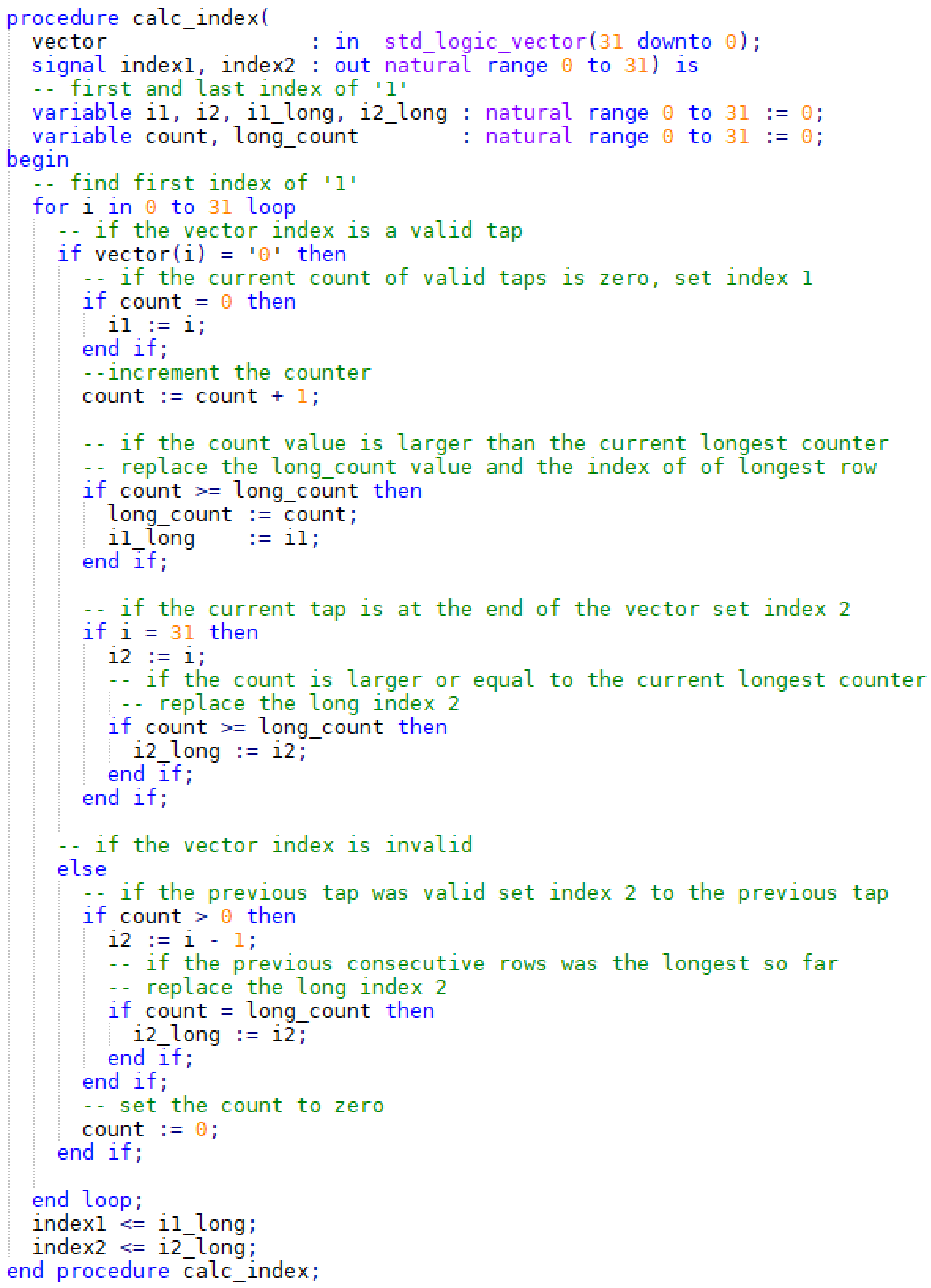
```
type t_state is (idle, wait_for_lock, search_1, search_2,
                 init_serdes, wait_until_valid, locked,
                 search_error, aligned);
```

Processes

`nxt_state_p ( all )` Chooses the next state. Scans all delay taps for comma, determines "good" and "bad" taps before choosing an optimal delay tap with calc_index(). Combinational logic.
`outputs_p ( current_state , tap_one_reg , tap_two_reg , delay_tap_out )` Controls the outputs of the state machine. Combinational logic.
`shift_data_p ( clk_120mhz , reset )` Shifts data into 20-bit buffer. Enables the search for a comma that is not word-aligned.
`state_reg_p ( clk_120mhz )` Stores signals in registers, resets when run is not enabled.

Procedures

`calc_index` Calculates the index of the longest interval of '0'. Used for finding the optimal delay tap. Code and comments in Figure B.2.



```vhdl
procedure calc_index(
  vector                : in  std_logic_vector(31 downto 0);
  signal index1, index2 : out natural range 0 to 31) is
  -- first and last index of '1'
  variable i1, i2, i1_long, i2_long : natural range 0 to 31 := 0;
  variable count, long_count        : natural range 0 to 31 := 0;
begin
  -- find first index of '1'
  for i in 0 to 31 loop
    -- if the vector index is a valid tap
    if vector(i) = '0' then
      -- if the current count of valid taps is zero, set index 1
      if count = 0 then
        i1 := i;
      end if;
      --increment the counter
      count := count + 1;

      -- if the count value is larger than the current longest counter
      -- replace the long_count value and the index of of longest row
      if count >= long_count then
        long_count := count;
        i1_long    := i1;
      end if;

      -- if the current tap is at the end of the vector set index 2
      if i = 31 then
        i2 := i;
        -- if the count is larger or equal to the current longest counter
        -- replace the long index 2
        if count >= long_count then
          i2_long := i2;
        end if;
      end if;

    -- if the vector index is invalid
    else
      -- if the previous tap was valid set index 2 to the previous tap
      if count > 0 then
        i2 := i - 1;
        -- if the previous consecutive rows was the longest so far
        -- replace the long index 2
        if count = long_count then
          i2_long := i2;
        end if;
      end if;
      -- set the count to zero
      count := 0;
    end if;

  end loop;
  index1 <= i1_long;
  index2 <= i2_long;
end procedure calc_index;
```

Figure B.2: VHDL code for calc_index procedure.

protocol_checker

Monitor received data stream and check ALPIDE protocol. Were translated from System Verilog to VHDL and expanded in this work.

Types

```
 type t_code is record
        idle           : std_logic_vector(7 downto 0);
        busy_off       : std_logic_vector(7 downto 0);
        busy_on        : std_logic_vector(7 downto 0);
        chip_empty     : std_logic_vector(7 downto 0);
        chip_trailer   : std_logic_vector(7 downto 0);
        chip_header    : std_logic_vector(7 downto 0);
        region_header  : std_logic_vector(7 downto 0);
        data_short     : std_logic_vector(7 downto 0);
        data_long      : std_logic_vector(7 downto 0);
end record;

type t_state is (s_idle, s_chip_header, s_store_data,
                s_data_short, s_data_long, s_chip_empty,
                s_test_error);
```

Constants

```
constant code : t_code := (
        idle          => 8b"1111_1111",
        busy_off      => 8b"1111_0000",
        busy_on       => 8b"1111_0001",
        chip_empty    => 8b"1110_----",
        chip_trailer  => 8b"1011_----",
        chip_header   => 8b"1010_----",
        region_header => 8b"110-_----",
        data_short    => 8b"01--_----",
        data_long     => 8b"00--_----");
```

Processes

fsm_p (clk) Scans the incoming data, and controls that it adheres the ALPIDE protocol.

Procedures

`store_event ( void t_void )` Procedure for storing an event. Dummy parameter to make the procedure calls less ambigious.

bitslip

Processes

`bitslip_p ( clk_120mhz , areset )` Prevents that bitslip is asserted too often. Three decoder errors are required before bitslip is asserted.

axi_alpide_data_reg

Handles write and read of AXI register. Generated by Vivado[1] but expanded to reflect the AXI Slave Module template methodology.

Processes

`PROCESS_0 ( s_axi_aclk )` Implement axi_awready generation. axi_awready is asserted for one s_axi_aclk clock cycle when both s_axi_awvalid and s_axi_wvalid are asserted. axi_awready is de-asserted when reset is low.

`PROCESS_1 ( s_axi_aclk )` Implement axi_awaddr latching. This process is used to latch the address when both s_axi_awvalid and s_axi_wvalid are valid.

`PROCESS_2 ( s_axi_aclk )` Implement axi_wready generation. axi_wready is asserted for one s_axi_aclk clock cycle when both s_axi_awvalid and s_axi_wvalid are asserted. axi_wready is de-asserted when reset is low.

`PROCESS_3 ( s_axi_aclk )` Implement memory mapped register select and write logic generation. The write data is accepted and written to memory mapped registers when axi_awready, s_axi_wvalid, axi_wready and s_axi_wvalid are asserted. Write strobes are used to select byte enables of slave registers while writing. These registers are cleared when reset (active low) is applied. Slave register write enable is asserted when valid address and data are available and the slave is ready to accept the write address and write data.

`PROCESS_4 ( s_axi_aclk )` Implement write response logic generation. The write response and response valid signals are asserted by the slave when axi_wready, s_axi_wvalid, axi_wready and s_axi_wvalid are asserted. This marks the acceptance of address and indicates the status of write transaction.

`PROCESS_5 ( s_axi_aclk )` Implement axi_arready generation. axi_arready is asserted for one s_axi_aclk clock cycle when s_axi_arvalid is asserted. axi_awready

[1]The process descriptions are also adopted from the generated template.

is de-asserted when reset (active low) is asserted. The read address is also latched when s_axi_arvalid is asserted. axi_araddr is reset to zero on reset assertion.

`PROCESS_6 ( s_axi_aclk )` Implement axi_arvalid generation. axi_rvalid is asserted for one s_axi_aclk clock cycle when both s_axi_arvalid and axi_arready are asserted. The slave registers data are available on the axi_rdata bus at this instance. The assertion of axi_rvalid marks the validity of read data on the bus and axi_rresp indicates the status of read transaction. axi_rvalid is deasserted on reset (active low). axi_rresp and axi_rdata are cleared to zero on reset (active low).

`PROCESS_7 ( all )` Implement memory mapped register select and read logic generation. Slave register read enable is asserted when valid address is available and the slave is ready to accept the read address.

axi_pkg

Defines the types required by the AXI bus interface.

Types

```
type axi_interconnect_to_slave is record
        araddr  : std_logic_vector(AXI_ADDR_WIDTH-1
                downto 0);
        arprot  : std_logic_vector(2 downto 0);
        arvalid : std_logic;
        awaddr  : std_logic_vector(AXI_ADDR_WIDTH-1
                downto 0);
        awprot  : std_logic_vector(2 downto 0);
        awvalid : std_logic;
        bready  : std_logic;
        rready  : std_logic;
        wdata   : std_logic_vector(AXI_DATA_WIDTH-1
                downto 0);
        wstrb   : std_logic_vector((AXI_DATA_WIDTH/8)-1
                downto 0);
        wvalid  : std_logic;
end record;

type axi_slave_to_interconnect is record
        arready : std_logic;
        awready : std_logic;
        bresp   : std_logic_vector(1 downto 0);
        bvalid  : std_logic;
        rdata   : std_logic_vector(AXI_DATA_WIDTH-1
                downto 0);
        rresp   : std_logic_vector(1 downto 0);
        rvalid  : std_logic;
        wready  : std_logic;
end record;
```

APPENDIX C

Data Readout Registers

Table C.1 provides a subset of registers in the data module. More registers are defined but are only used in the transceiver design. The width of the registers is constrained by the use of Cypress FX3 and Wishbone. All registers are defined in records in `alpide_data_pkg` and documented in the pCT repository.

Table C.1: Subset of registers in the data module.

Name	**Access**	**Width**	**Default**	**Description**
run	R/W	1	0x0	Enables the phase aligner module
test_mode	R/W	1	0x0	Enables the test mode functionality of the protocol checker
test_vector1_2	R/W	16	0x0	Register to store two of the three test vectors
test_vector3	R/W	8	0x0	Register to store the third test vector
aligned	R/O	1	0x0	Phase aligner has confirmed that data is aligned.
busy	R/O	1	0x0	The protocol checker asserts that the ALPIDE is busy
locked_in_eye	R/O	1	0x0	Phase aligner has found a delay tap and locked to it
delay_tap_out	R/O	5	0x0	The delay tap position IDELAYE2 is currently locked to.
correct_cnt	R/O	16	0x0	The number of consecutive correct words, bit 15:0.
correct_cnt2	R/O	16	0x0	The number of consecutive correct words, bit 31:16
correct_cnt3	R/O	16	0x0	The number of consecutive correct words, bit 47:32
incorrect_cnt	R/O	16	0x0	The number of incorrect words, bit 15:0.
incorrect_cnt2	R/O	16	0x0	The number of incorrect words, bit 31:16.
incorrect_cnt3	R/O	16	0x0	The number of incorrect words, bit 47:32.
last_sample_word	R/O	10	0x0	The last word that was output directly from ISERDESE2
last_decoded_word	R/O	8	0x0	The last word that was output directly from 8B10B decoder.
bitslip_counter	R/O	16	0x0	The number of bitslip strobes since startup.
idelayctrl_locked	R/O	1	0x0	The state of IDELAYCTRL.

APPENDIX D

Testbench Procedures and Functions

alpide_pkg

Package for simplifying Wishbone communication to ALPIDE. Uses Wishbone BFM procedures to write and read from the ALPIDE.

Procedures

`alpide_broadcast ( )` Broadcast opcode to ALPIDE(s).
`alpide_check ( )` Read alpide register and confirm expected value. Uses wb_check to check if the data is correct. Returns failure if differing value.
`alpide_init_test_pattern ( )` Initialize sending of test vectors from ALPIDE. Test vectors are defined in test_pkg.
`alpide_stop_test_pattern ( )` Stop sending of test vectors from ALPIDE. Does not deactivate the data channel, so test vectors are replaced by comma words.
`alpide_write ( )` Write data to a ALPIDE register.

transceiver_pkg

Package for simplifying Wishbone communication to readout unit's data module. Must be updated for I/O primitive usage. Uses Wishbone BFM procedures to write and read from the readout unit.

Procedures

`startup_sequence ()` Startup sequence for the MGT.
`transceiver_read ()` Read data module register.
`transceiver_test_pattern_check_on ()` Sets the registers that initialize the test vector check feature.

`transceiver_test_pattern_check_off ()` Resets the registers that enables the test vector check feature.

`check_errors ()` Check that register that indicates errors are not set.

APPENDIX E

Test Software

The test software is written on the basis on the following preexisting packages:[1]

`communication` Communication classes, used to communicate with a board, using the FX3 chip and usb_if firmware. The provided communication classes are either direct via libusb, over the usb_comm server, or via simulation. They all share the common base class Communication.

`wishbone_module` Abstraction of a wishbone accessed module. Provides Read/Write functionality to a module with given module_id on construction time.

`dctrl` Contains classes to implement the latest capabilities of the dctrl/ctrl block relative to wait/trigger commands with busy management.

`userdefinedexceptions` Contains user-defined exceptions.

All packages, except the communication package, have been modified and expanded during the pCT development. What follows is what has been added to the software, both to the existing packages and to new packages. Complete documentation of all code is found in the pCT repository.

wishbone_module

`WishboneModule.pureReadback (self, addr, data)` Write to a specific address of the module, then reads back to confirm R/W functionality.

`WishboneModule.check (self, addr, data, standalone=True)` Read from specific address from the module, and confirm that it contains the specified data.

dctrl

`Dctrl.broadcast (self, opcode, chipid)` Broadcast a specific opcode to a chip.

[1]These packages was copied from the ITS repository. The author is unknown.

`Dctrl.check_reg (self, address, data, chipid)` Read from a specific chip register, and confirm that it contains the specified data.

`Dctrl.pureReadback (self, address, data, chipid)` Write to a specific chip register, and then read it to confirm functionality.

`dctrlTest ()` Test the ALPIDE slow control communication channel. Test the default values of the ALPIDE registers, and then tests read and write functionality. Employs the functions above.

userdefinedexceptions

`class AlpideCheckError(Exception)` Exception raised if mismatch between expected and read data from ALPIDE.

init_python

Initialize environment from where multiple tests can be run.

`powerOnAlpide ()` Turns on ALPIDE power.

`powerOffAlpide ()` Turns off ALPIDE power.

`setupClockOut600Mhz ()` Set up ALPIDE to output clock signal on the data channel. Utilizes ALPIDE test vector functionality to output clock signal. Frequency is 600 MHz.

`setupClockOut200Mhz ()` Set up ALPIDE to output clock signal on the data channel. Utilizes ALPIDE test vector functionality to output clock signal. Frequency is 200 MHz.

`initPRBS ()` Set up ALPIDE to output PRBS. Utilizes ALPIDE PRBS test feature.

`initComma ()` Set up ALPIDE to output comma word. Enable ALPIDE data channel without test functionality. Will result in transmission of consecutive comma words.

`setupTestVector (tv1, tv2, tv3)` Set up ALPIDE to transmit given test vectors and data module to expect the same vectors. Set test vectors in data module first, before sending the test vectors to the ALPIDE registers.

`resetAlpide ()` Reset ALPIDE to initial state. Goes through all ALPIDE registers and set them to manual specification. Finally, broadcast all the reset commands. Turn off power. Then turn power on again.

`resetReadout ()` Reset data module run register.

`enableReadout ()` Set data module run register.

`enableReadoutTestMode ()` Set data module run and test mode registers.

`readAligned ()` Read the aligned register.

`readBusy ()` Read the busy register.

`readLockedInEye ()` Read locked_in_eye register.

`readDelayTapOut ()` Read delay_tap_out register.

`readCorrectCounter ()` Read correct_cnt register.
`readIncorrCounter ()` Read inncorrect_cnt register.
`readLastSampleWord ()` Read last_sample_word register.
`readLastDecodeWord ()` Read last_decoded_word register.
`readBitslipCounter ()` Read bitslip_counter register.
`readSerdesDelayLocked ()` Read idelayctrl_locked register.
`readTest ()` Performs test of the readout. Let the user specify the number of test runs, and the lenght of each test run. A run is initialized with resetting the ALPIDE, turning the power off/on, and the resetting the readout. Comma is initialized, and the data module is set in test mode. All steps are done with random time intervals. Further, random test vectors are produces, and then sent to ALPIDE and data module. During test run, all relevant data are polled from registers and logged to screen and file. When all test runs are completed, the results are printed.
`measureErrorTime ()` Get a text file of the times for the first error in a test run. Perform readout test to measure time of first error detection. Set up test runs the same way as readTest(). When encountering error, log time to text file and start next test run.

readout_test

The readout test functions employed for MGT testing.

`readout_test.readout_test ()` Test that MGT correctly samples and encodes data.

`readout_test.testDisplay(stdscr,time,noErrorCounter, errorCounter,errorWord)`
Displays test runtime statistics during test run.

check_fx3

Checks if correctly flashed Cypress FX3 is connected to USB.

alpide_power_on

Turns on ALPIDE power.

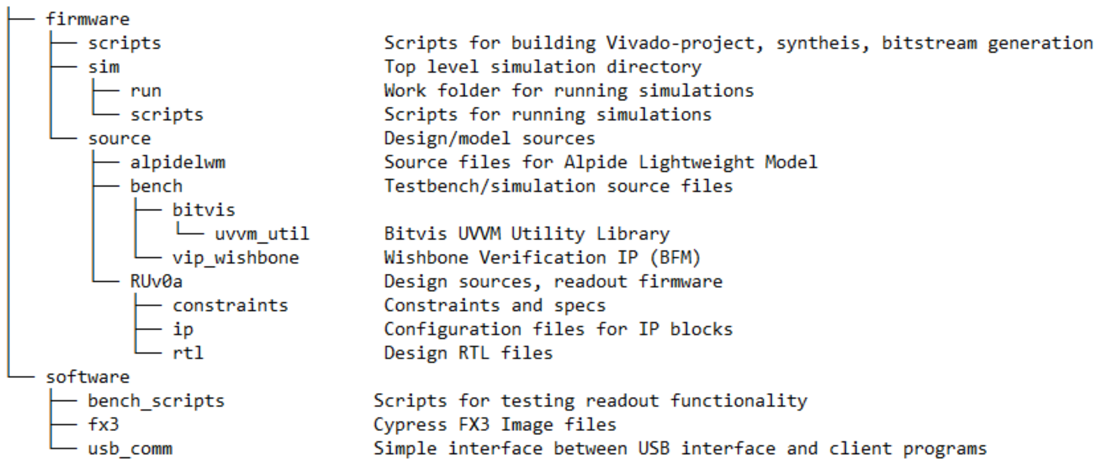
program_fpga

Program connected FPGA with given bitfile.

APPENDIX F

Repository Structure

All the steps of the development process has been carefully tracked by employing `git`, a version control system. Below is the outline of the structure of the code repository.



```
├── firmware
│   ├── scripts                         Scripts for building Vivado-project, syntheis, bitstream generation
│   ├── sim                             Top level simulation directory
│   │   ├── run                         Work folder for running simulations
│   │   └── scripts                     Scripts for running simulations
│   └── source                          Design/model sources
│       ├── alpidelwm                   Source files for Alpide Lightweight Model
│       ├── bench                       Testbench/simulation source files
│       │   ├── bitvis
│       │   │   └── uvvm_util           Bitvis UVVM Utility Library
│       │   └── vip_wishbone            Wishbone Verification IP (BFM)
│       └── RUv0a                       Design sources, readout firmware
│           ├── constraints             Constraints and specs
│           ├── ip                      Configuration files for IP blocks
│           └── rtl                     Design RTL files
└── software
    ├── bench_scripts                  Scripts for testing readout functionality
    ├── fx3                            Cypress FX3 Image files
    └── usb_comm                       Simple interface between USB interface and client programs
```

Figure F.1: The structure of the pCT readout code repository.

APPENDIX G

Signal Integrity

Signal integrity refers to all problems that arise in high-speed devices due to interconnects, but are often referring to noise issues. Signal integrity effects are getting significant whenever a signal is operating in the high-frequency regime and the propagation delay along the conductor are comparable to the signal rise/fall times [7, Chapter 12]. All signal integrity problems can be referred to as one of four families of problems:

- the signal quality of one net including reflections at impedance changes, and high-frequency attenuation
- cross-talk between multiple nets
- weak power distribution network
- electromagnetic interference and radiation to/from the entire system

However, as a general rule, all of the problems above will get worse with a decreasing rise-time of a signal. Furthermore, all of these problems may be described based on impedance, which is the key electrical property of an interconnect. [39]

G.0.1 Attenuation

Bogatin [39] refers to loss in transmission lines as the main problem for signals with clocks higher than 1 GHz with a distance traveled more than 25 cm. As the signal frequency increases, the impedance is dominated by inductance and causes the effective thickness of the conductor to decrease [7, Chapter 5]. This is called the skin effect and causes increasing conductor resistance and higher attenuation. However, both conductor loss and dielectric loss contribute to attenuate the high-frequency components more than the low-frequency components of a signal [39].

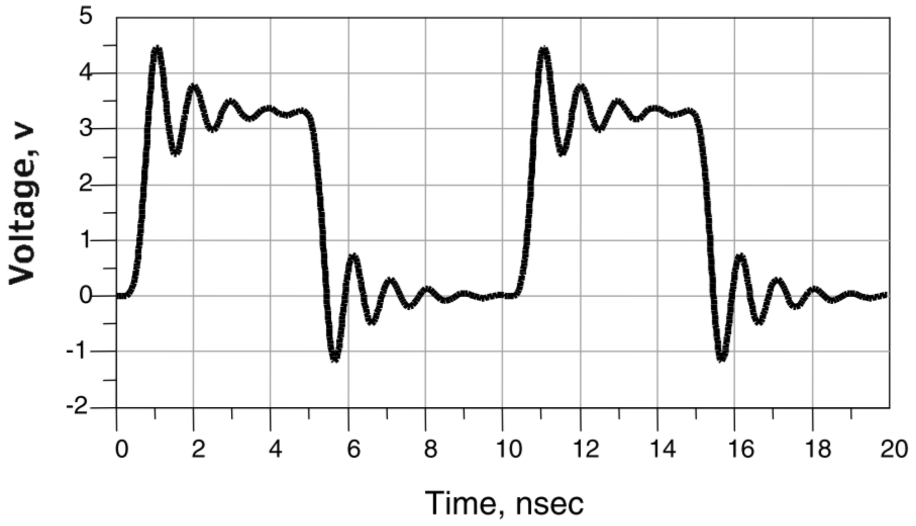


Figure G.1: Ringing noise at the receiver end of a 1-inch-long controlled-impedance interconnect created because of impedance mismatches and multiple reflections at the ends of the line [39, Figure 8-1].

G.0.2 Reflections

Reflections on a electrical interconnect are the result of impedance discontinuities. Often the effect looks like ringing, as is illustrated in Figure G.1, and may cause false triggering and sampling. Often the designer can reduce the fluctuation in instantaneous impedance in an interconnect by avoiding branching and minimizing stub lengths. Furthermore, a good strategy for termination at the ends of the transmission lines is critical. This is because the impedance will always deviate at the ends of the transmission lines [39]. Bogatin's rule number 8 of signal integrity is to place the terminating resistors as close to the package pads as possible.

G.0.3 Jitter

The quality of a clock signal is dependent on the amount of phase jitter [58]. It refers to the time domain instability of the clock signal and is typically expressed in picoseconds (ps). It is also expressed as the signal's random phase variation from its ideal waveform.

G.0.4 Reliable Data Sampling

To sample data reliably, one has to make sure that the sample point is between data transitions. A typical high-speed signal is illustrated in Figure G.2. The eye diagram

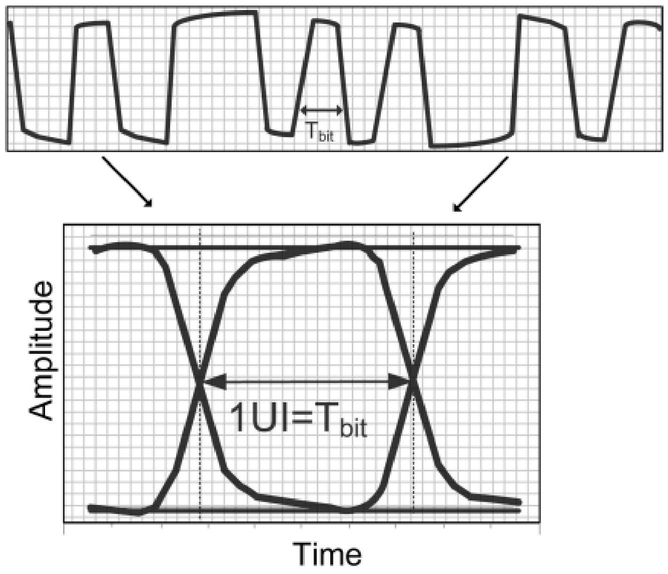


Figure G.2: Typical high speed digital signal with eye diagram [56, Figure 2].

illustrates the electrical quality of the signal, and phenomena like jitter and reflections will distort the eye, making it harder to reliably find a point where the data value is not ambiguous. A signal with poor integrity will produce an eye diagram where it is impossible or difficult to find a point where the value is always clear.

APPENDIX H

Methodology

When starting a design of hardware, software or any other complex design, some methodology is essential to accomplish the desired result. First and foremost, it is required to know what the design should do and what the requirements that pose the critical operation are. Moreover, subsequently, the methods and techniques to be implemented to achieve this operation must be identified. Lastly, the timeline and the steps of development should be roughly defined and updated when required. The pCT project involved a large number of unknowns when the work with this thesis was started. Therefore, creating a detailed technical specification of the entire readout system proved difficult. However, some methods and principles were considered to make sure the effort and time dedicated to the project resulted in a favorable outcome.

H.1 Information Gathering

A typical design flow will usually start with time devoted to information gathering. Especially information that relates to interfaces between components, timing, and bus protocols are crucial when developing readout electronics. The information available at the time of the start of the thesis was limited and mainly involved the chosen sensor chip for the project. Equally important, FPGA had been selected to be the preferred design method to implement the readout circuit. The selected sensor chip ALPIDE are described in Chapter 3. The ALPIDE Operation Manual provided information about the chips interfaces, timing and bus protocol. Also provided was the code base for a readout unit under development which involved the ALPIDE chip. This constituted a starting point for setting up a simulation environment for development and testing.

H.2 Design Strategies

During development of a sophisticated design, several strategies can be enforced to utilize maintainable structure and provide efficiency in converting from concept to implementation. During design, there are continuous trade-offs between parameters such as performance, time to design and ease of verification [7, Chapter 13]. When designing for programmable logic and FPGAs, some parameters, such as cost, changes dramatically from ASIC development. However, since various FPGAs involve variable costs, it is still important to evaluate factors like the size of the design, the number of I/Os needed and the bandwidth required. Weste and Harris identify four design principles for IC design [7, Chapter 13].

H.2.1 Hierarchy

Hierarchy refers to the division of the system into modules and then repeating the process to further divide the modules into smaller submodules [7, Chapter 13]. Eventually, the whole system can be represented by a tree of modules and submodules. The complexity of the system is then simplified to the degree that every function and submodule are easily comprehended. Creating an architecture in which modules and submodules are clearly defined and separated from each other, significantly increases effectiveness in designing the individual modules. A hierarchy approach also enables the possibility to use IP Cores, previously established and designed modules (see Section H.2.3). These may be available internally at a company or an institution, or FPGA manufacturers like Xilinx and Altera. In fact, there has been increasing interest in creating an open source community for hardware design, and this has made several advanced IP Cores available for free online.[1]

H.2.2 Regularity

Sometimes it is possible to define modules that enable reuse in other parts of the architecture. Regularity is the principle to define similar modules for reuse [7, Chapter 13]. This simplifies the design process, and minimizes the effort needed to verify the functionality. Regularity will reduce the total number of unique components in the design, and thus, the number of components to be tested. E.g., when the pCT has not defined the number of sensor chips to be connected to a single readout interface, regularity of the design of one connection is important. The regularity will simplify the scalability of the design by making it possible to duplicate entities previously defined.

[1]For instance, the site opencores.org provides hosting of source code for an extensive collection of IP Cores.

H.2.3 Modularity

When creating a complex system, it is often necessary to divide the design effort between different persons or teams and delegate. Specific modules may be assigned to a particular team, and they must be able to interface correctly with another module assigned to another team. For this process to be manageable, it is important to define the modules properly. Modularity refers to that modules must have clearly defined functions and interfaces. In HDL it is usually important to define the ports as either digital or analog and also determine the direction of the signal [7, Chapter 13]. It is equally important to define clocking specifications, potential bus protocols and all other specifics that may refer to the behavior of module observable to the outside environment.

H.2.4 Locality

Locality refers to information hiding [7, Chapter 13], and is a principle that complements the modularity principle. In software design, it often relates to the axiom to not create global variables when not needed. When specifying a module's outside characteristics, there should also be given a high priority to hide the complexity that is redundant to the module's user. Having said that, locality often also means that all signals are referenced to a particular clock [7, Chapter 13].

H.2.5 Scalability

Scalability is a design principle which is not identified by Weste and Harris, but it is implied by the results from the other principles. When the knowledge of size and scale of the end product is unknown, it is important that design can both scale up and down to handle the required size when it ultimately is defined. Besides, this principle can also be critical when updating a design to a new version. The scalability precept is naturally bound to both hierarchy and regularity, which are both required to achieve this flexibility.

H.3 Simulation and Verification

The most common form of verification of electronic design is a computer-assisted simulation. With modern HDL simulators, it is possible to simulate sophisticated designs and view all the involved signals with time precision down to 10^{-15} seconds (picoseconds). This technique is superior to testing modules and designs as physical prototypes as it greatly simplifies debugging and reduces time spent on unnecessary synthesis and hardware setup. The simulation may also reveal bugs that are quite hard or impossible to detect in a lab setup, and the bugs may even be caught by a short simulation [59]. Every time a change is done in the design, there is a chance that faults are introduced. This is why both simulation and verification are needed at every increment in the design and development process. This process is called regression testing.

H.3.1 Testbench

By creating a set of expected behaviors that a model should execute with a given set of inputs, it is possible to verify that a module performs correctly. This method involves the use of a verification testbench. The name is an analogy to a real hardware testbench where one uses signal generators as input stimuli and watch and confirm behavior by looking at output signals with probes [57, Chapter 1]. In an HDL testbench, the module or device under test (DUT) is instantiated as a component instance, and processes generate a sequence of signal values that are connected to the inputs to the DUT. It is also common to include processes that verify that the outputs of the DUT correspond correctly to the input stimuli.

H.3.2 Bitvis UVVM

It is possible to construct testbenches from pure HDLs like VHDL or Verilog, as well as use higher lever languages like SystemVerilog or SystemC. These higher level languages are similar to C ++ but contain concurrent time handling which makes them appropriate for testing HDLs. However, there are benefits of using the same language as is used in the model design. Specifically, it is easier to detect typos and errors which involve falsely matching types between languages. Unfortunately, VHDL does not provide extensive testbench features, like expressing design intent, without using a separate language [57, Chapter 18].

Nevertheless, Bitvis UVVM presents a complete VHDL verification environment that considerably assists in verifying complex designs without using a separate language. One of the main features of the UVVM approach is that it provides a starting point for creating a simple testbench architecture. For a simple device, a testbench architecture should contain a test sequencer and support processes. Support procedures and functions are provided by UVVM. Figure H.1 shows an example testbench architecture for a simple interrupt controller. The test sequencer contains all the tests to be run and should be readable and manageable. A test sequencer is effortlessly built by using built-in UVVM features. [60]

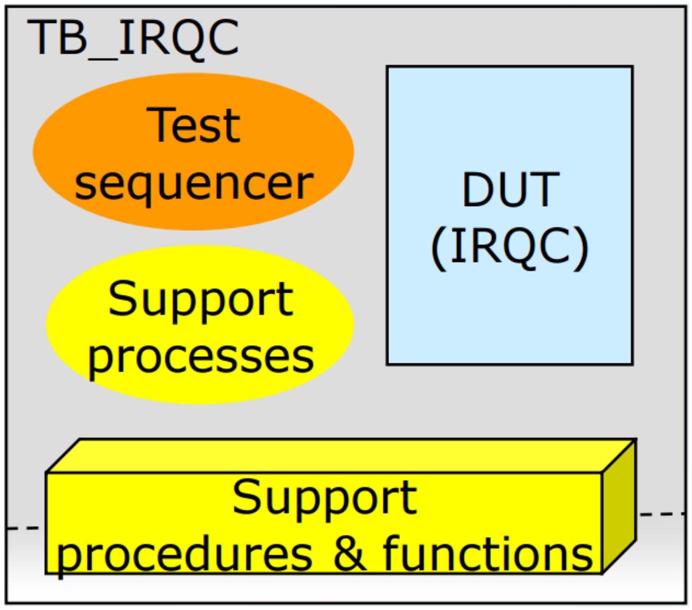


Figure H.1: A simple testbench architecture [59]

H.3.3 BFM

A BFM is a set of procedures and functions which emulate the behavior of the referred bus protocol. A BFM simplifies bus communication in the simulation environment where it replaces an often long sequence of commands with a simple procedure call. Figure H.2 shows how a write procedure may be substituted by a simple procedure call. The most basic BFM only contain simple read, write and check transaction procedures. More advanced BFMs also feature synchronization to the respective bus clock, normalization of inputs, sanity checks, etc. Most important they feature logging capabilities and different severity levels. These features enable the designer to turn information about the process on or off as needed.

H.4 Hardware Testbench

When a design has been completely verified by simulation at different abstraction layers, it must be confirmed in a physical setup. FPGA development simplifies this step by avoiding fabrication of complex mask sets for microchips. It is often even possible to use a vendor-provided development board where the chosen FPGA is already mounted and there exist useful ports and connections that comply to the desired usage. However, if custom connections are to be used, it is necessary to develop a custom PCB. This development will involve several steps that may introduce faults and errors.

```
cs     <= '1';
we     <= '1';
addr   <= "00100010";
data   <= "11110000";
wait until rising_edge(clk);
wait until falling_edge(clk);
cs     <= '0';
we     <= '0';
```

(a) Write procedure without using BFM [59].

```
write(x"22", x"F0");
```

(b) BFM replacement [59].

Figure H.2

A crucial part of the physical test setup is that it is possible to inject test stimuli to the DUT and control that the outputs are correct. This can be achieved by manually inject input stimuli with signal generators and check the output signals with probes. This is, without question, complicated, time-consuming, as well as prone to errors. However, it is possible to attain a test setup that is highly automated in comparison. By implementing a communication channel between the test PCB and the outside world, e.g. USB connection to a computer, the inputs and outputs can be completely controlled by a high-level computer language. To recreate the tests performed in the simulation environment is feasible with a language like Python. Unfortunately, there does not currently exist any automated way to translate between VHDL and Python, and this has to be executed manually.

Another physical test technique may involve implementing built-in self-test modules in the FPGA device itself. This technique allows the module to implement and perform tests upon themselves but will add to the area used on the chip [7, Chapter 14]. However, it may contribute to reducing testing time and effort, and thus the cost involved in testing.

H.4.1 Integrated Logic Analyzer

Many FPGA vendors offer a kind of logic analyzer that can be implemented in the circuit. Altera offers SignalTap, while Xilinx has the Integrated Logic Analyzer (ILA) IP core. These cores can be used to monitor the internal signals of the design. This is very useful since there are no possibilities to access probe points in FPGA circuits physically. The core can be activated by setting up a trigger condition, e.g. a rising edge of a particular signal. The signal states are then collected over a given number of clock cycles and stored in the on-chip block RAM. The information about the signal states is transmitted to a computer via an auto-instantiated core that connects to JTAG.

Bibliography

[1] S. M. Macdonald, T. F. Delaney, and J. S. Loeffler, “Proton Beam Radiation Therapy,” *Cancer Investigation*, vol. 24, no. 2, pp. 199–208, 2006.

[2] H. Lynnebakken, “Jubler for partikkelterapi,” 2012. [Online]. Available: http://www.mn.uio.no/fysikk/forskning/aktuelt/aktuelle-saker/2012/jubler-for-partikkelterapi.html

[3] M. Jermann, “Particle Therapy Statistics in 2014,” *International Journal of Particle Therapy*, vol. 2, no. 1, pp. 50–54, 2015. [Online]. Available: http://theijpt.org

[4] H. Pettersen, J. Alme, A. Biegun, A. van den Brink, M. Chaar, D. Fehlker, I. Meric, O. Odland, T. Peitzmann, E. Rocco, K. Ullaland, H. Wang, S. Yang, C. Zhang, and D. Röhrich, “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*, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900217301882

[5] V. A. Bashkirov, R. P. Johnson, H. F.-W. Sadrozinski, and R. W. Schulte, “Development of proton computed tomography detectors for applications in hadron therapy,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 809, pp. 120–129, 2016.

[6] ALICE ITS ALPIDE development team, “ALPIDE Operations Manual,” Tech. Rep., 2016.

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

[8] S. Naeem Ahmed, *Physics and Engineering of Radiation Detection.* Academic Press, 2007.

[9] F. M. Khan, *Physics of Radiation Therapy*, 3rd ed. Philadelphia, PA: Lippincott Williams & Wilkins, 2003.

[10] H. Cember, *Introduction to Health Physics*, 4th ed. New York: McGraw-Hill Medical, 2009.

[11] T. F. Thorsteinsen, *Kompendium i Strålingsfysikk*. Institutt for fysikk og teknologi - Universitetet i Bergen, 2014.

[12] J. D. Cutnell and K. W. Johnson, *Physics*, 9th ed. John Wiley and Sons, Inc, 2012.

[13] H. D. Young and R. A. Freedman, *Sears and Zemansky's University Physics: with Modern Physics*, 13th ed. San Francisco: Addison-Wesley, 2011.

[14] D. Schulz-Ertner and H. Tsujii, "Particle Radiation Therapy Using Proton and Heavier Ion Beams," *Journal of Clinical Oncology*, vol. 25, no. 8, pp. 953–964, 2007.

[15] P. P. Connell and S. Hellman, "Advances in Radiotherapy and Implications for the Next Century: A Historical Perspective," *Cancer Research*, vol. 69, no. 2, 2009.

[16] M. Brada, M. Pijls-Johannesma, and D. De Ruysscher, "Proton Therapy in Clinical Practice: Current Clinical Evidence," *Journal of Clinical Oncology*, vol. 25, no. 8, pp. 965–970, mar 2007. [Online]. Available: http://www.jco.org/cgi/doi/10.1200/JCO.2006.10.0131

[17] T. Bortfeld, H. Paganetti, and H. Kooy, "Proton Beam Radiotherapy - The State of the Art," 2005. [Online]. Available: http://www.aapm.org/meetings/05AM/pdf/18-4016-65735-22.pdf

[18] P-Cure Clinics, "P-Cure Clinics." [Online]. Available: http://www.p-cure.com/index.aspx?id=4013

[19] D. Schulz-Ertner, O. Jäkel, and W. Schlegel, "Radiation Therapy With Charged Particles," *Seminars in Radiation Oncology*, vol. 16, no. 4, pp. 249–259, 2006.

[20] G. Poludniowski, N. M. Allinson, and P. M. Evans, "Proton radiography and tomography with application to proton therapy," *The British Journal of Radiology*, vol. 88, no. 1053, p. 20150134, sep 2015. [Online]. Available: http://www.birpublications.org/doi/10.1259/bjr.20150134

[21] R. P. Johnson, V. Bashkirov, L. Dewitt, V. Giacometti, R. F. Hurley, P. Piersimoni, T. E. Plautz, H. F. W. Sadrozinski, K. Schubert, R. Schulte, B. Schultze, and A. Zatserklyaniy, "A Fast Experimental Scanner for Proton CT: Technical

Performance and First Experience with Phantom Scans," *IEEE Transactions on Nuclear Science*, 2016.

[22] The ALICE Collaboration, "The ALICE experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, no. 08, pp. S08 002–S08 002, aug 2008. [Online]. Available: http://stacks.iop.org/1748-0221/3/i=08/a=S08002?key=crossref.4a430fa328e181a89b6c10b850640204

[23] R. Lemmon, "The ALICE Inner Tracking System Upgrade," *Nuclear Physics A*, vol. 904-905, pp. 937c–940c, may 2013. [Online]. Available: http://arxiv.org/abs/1211.4494

[24] ALICE ITS Upgrade Team, "Alice ITS Upgrade: Readout Electronic - WP10," CERN, Tech. Rep., 2016.

[25] P. A. Franaszek and A. X. Widmer, "Byte oriented DC balanced (0,4) 8B/10B partitioned block transmission code," 1984. [Online]. Available: http://www.google.no/patents/US4486739

[26] J. F. Wakerly, *Digital Design: Principles and Practices*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2000.

[27] M. Bonora, "Readout Module Progress," CERN, Tech. Rep., 2016.

[28] ALICE ITS Upgrade Team, "Readout Electronics Overview," Tech. Rep. October, 2015.

[29] Xilinx Inc., "Single Event Upsets," 2017. [Online]. Available: https://www.xilinx.com/support/quality/single-event-upsets.html

[30] Xilinx Inc., "UltraScale+ FPGA Product Tables and Product Selection Guide," 2017. [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf

[31] Xilinx Inc., "Zynq UltraScale+ MPSoC Product Tables and Product Selection Guide," 2016. [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf

[32] Xilinx Inc, "Performance and Resource Utilization for MicroBlaze MCS v3.0," 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ru/microblaze-mcs.html

[33] K. Sielewicz, "ITS Prototype Readout Board Schematic," Tech. Rep., 2015.

[34] A. Sanchez, "pALPIDE3 Carrier V3 Schematic," Tech. Rep., 2015.

[35] A. Sanchez, “ALPIDE3 ITS Readout Adaptor Schematic,” Tech. Rep., 2015.

[36] H. Twyman, “Digital Clock Recovery.” [Online]. Available: http://www.twyman.org.uk/clock_recovery/

[37] D. Nagaria and A. Mitta, “Beginner’s Guide To Clock Data Recovery.” [Online]. Available: http://www.arrowdevices.com/blog/beginners-guide-to-clock-data-recovery/

[38] S. Churiwala and I. Hyderabad, *Designing with Xilinx® FPGAs*, S. Churiwala, Ed. Cham: Springer International Publishing, 2017. [Online]. Available: http://link.springer.com/10.1007/978-3-319-42438-5

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

[40] Xilinx Inc., “7 Series FPGAs GTX/GTH Transceivers - User Guide,” 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf

[41] Xilinx Inc., “Kintex-7 FPGAs Data Sheet: DC and Switching Characteristics (DS182),” 2015. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds182_Kintex_7_Data_Sheet.pdf

[42] Xilinx Inc., “Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics (DS923),” 2016. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds923-virtex-ultrascale-plus.pdf

[43] Xilinx Inc., “7 Series FPGAs SelectIO Resources User Guide (UG471),” 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf

[44] Xilinx Inc., “UltraScale Architecture SelectIO Resources User Guide (UG571),” 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug571-ultrascale-selectio.pdf

[45] K. L. Short, *VHDL for Engineers.* Essex: Pearson Education Limited, 2014.

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

[47] Xilinx Inc., “FIFO Generator v13.0 - LogiCORE IP Product Guide,” 2015. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v13_1/pg057-fifo-generator.pdf

[48] OpenCores, *Wishbone B4 - WISHBONE System-on-Chip (SoC)Interconnection Architecturefor Portable IP Cores.* OpenCores, 2010. [Online]. Available: https://opencores.org/cdn/downloads/wbspec_b4.pdf

[49] Open Hardware Repository, "Wishbone slave generator." [Online]. Available: http://www.ohwr.org/projects/wishbone-gen

[50] OpenCores, "WISHBONE Builder," 2014. [Online]. Available: http://opencores.org/project,wb_builder

[51] Xilinx Inc., "AXI Reference Guide," 2011. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

[52] R. Griffin, "Designing a Custom AXI-lite Slave Peripheral," 2014. [Online]. Available: https://forums.xilinx.com/xlnx/attachments/xlnx/NewUser/34911/1/designing_a_custom_axi_slave_rev1.pdf

[53] Xilinx Inc., "LogiCORE IP AXI to AXI Connector (v1.00.a): Product Specification," 2010. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ds803_axi2axi_connector.pdf

[54] Xilinx Inc., "MicroBlaze Soft Processor Core." [Online]. Available: https://www.xilinx.com/products/design-tools/microblaze.html

[55] M. Johnson, R. Cline, S. Ward, and J. Schichl, "Latch-Up (White Paper)," Texas Instruments, Tech. Rep., 2015. [Online]. Available: http://www.ti.com/lit/wp/scaa124/scaa124.pdf

[56] Semiconductor Components Industries, "AND9075/D Understanding Data Eye Diagram Methodology for Analyzing High Speed Digital Signals," 2015. [Online]. Available: http://www.onsemi.com/pub/Collateral/AND9075-D.PDF

[57] P. J. Ashenden, *The Designer's Guide to VHDL.* Morgan Kaufmann, 2010, vol. 3.

[58] Silicon Laboratories, "Jitter Attenuation - Choosing The Right Phase-Locked Loop Bandwidth," 2010. [Online]. Available: https://www.silabs.com/documents/public/application-notes/AN513.pdf

[59] E. Tallaksen, "Making a Simple, Structured and Efficient Testbench Step-by-step." [Online]. Available: http://www.bitvis.no/media/15295/Simple_TB_step_by_step.pdf

[60] Bitvis, "What is UVVM?" [Online]. Available: http://www.bitvis.no/products/uvvm-vvc-framework/what-is-uvvm/