Institute of High Frequency Technology
Prof. Dr.-Ing. Dirk Heberling

Diploma Thesis

FPGA-based Coincidence-Unit for Digital PET

Gökçe Aydos
Matr.-Nr. 266917

6 Januar 2012

Supervised by:

Prof. Dr.-Ing. Dirk Heberling(IHF)

Dipl.-Ing. Daniel Basten (IHF)

Dipl.-Ing. Pierre Gebhardt (Philips)

Dr. Christoph Lerche (Philips)

I hereby declare that I have created this work completely on my own and used no other sources or tools – except for the offical attendance by the Institute of High Frequency Technology – than the ones listed, and that I have marked any citations accordingly.

Also I give permission to the RWTH Aachen University to view, store, print, reproduce, and distribute this work as a whole or in part.

Aachen, 6 Januar 2012

(Gökçe Aydos)

Assignment of Tasks

In Positron Emission Tomography, a radioactive tracer is administered to the patient, which accumulates in certain parts of the body depending on the type of the tracer. After every radioactive decay, a positron is released, which annihilates with an atomic electron along its path. During this process, both particles are annihilated and subsequently two high-energy photons are emitted in opposite directions. These photons -so-called *Singles*- are detected with PET-Detectors, digitized and forwarded to the *Coincidence Unit*.

The task of the Coincidence Unit is to detect these Singles resulting from annihilations within a specific time window (*coincident* Singles), to forward these coincident Singles and to reject anything else. In the current non-clinical PET-system prototype, the Singles data from ten detector modules is transferred via ten 1 Gbit/s UDP/IP links with up to 700 Mbit/s throughput each. The resulting total amount of detected coincidences will be in the range of only about 650 Mbit/s and thus processable on the image reconstruction computer in an acceptable processing time, which is required for measurements for longer periods.

In this thesis, an infrastructure for a Coincidence Unit with twelve fiber-optical Gbit-Ethernet ports based on a Xilinx ML605 evaluation board shall be developed for the current PET-prototype. The particular tasks are:

- Design of a data-acquisition and -transmission architecture for the FPGA based on the existing system-architecture with own communication protocol.
- Implementation of the design with VHDL, Mentor HDL Designer and Xilinx ISE. Simulation of the design with Mentor ModelSim.
- Extension of the existing Control-Software in C# for controlling and monitoring of the Coincidence Unit.
- Commissioning of the Coincidence Unit and data transmission tests with detector modules of the current PET-prototype connected as data sources to the Coincidence Unit and workstation as data-dumper.
- Optional: Implementation of a *Processing Chain* for coincident Singles search for data reduction. The commissioning test will be carried out at least with emulated sensor data.

The thesis is done in cooperation with Philips Research Laboratories in Aachen. The experimental and practical part of the thesis will be done at Philips because of the needed laboratory infrastructure.

Summary

FPGA-based Coincidence-Unit for Digital PET

Positron Emission Tomography (PET) is a nuclear imaging method that can visualize the distribution of positron-emitting substances. This modality can provide images of metabolic processes in tissue, e.g. glucose intake of cells, and is mainly applied in oncology, to localize cancer cells, which have a high glucose intake.

The images in PET are reconstructed by detecting gamma photons pairs, which are generated by the annihilation of a positron with an electron in the surrounding tissue. The two gamma photons travel in opposite directions and are detected with a small time difference. Detection of such gamma-pair, which belong to an annihilation, is also called *Coincidence*. The processing unit, which searches for such gamma-pairs is called *Coincidence Unit*.

A current research project called *HYPERImage* conducted in Philips Research Labs Aachen focuses on the development of a novel PET-scanner for research purposes, which can be integrated in an Magnet Resonance Imaging-device (MRI) and allows simultaneous PET and MRI images. Compared to other hybrid imaging techniques used in the assessment of metabolic processes, the simultaneous PET/MRI could produce better images.

In this thesis, a Field-Programmable Gate Array-based (FPGA) infrastructure for a Coincidence Unit for this PET-scanner is developed.

The thesis includes a compact introduction to PET and description of communication infrastructure the mentioned PET-scanner, which are relevant for the thesis. Afterwards, the concept of the FPGA-architecture is introduced.

Most of the conceptual architecture were implemented and tested, which are documented in this thesis. Additionally, general FPGA-development principles based-on Xilinx FPGA-tools are discussed.

An important part of the implementation was the simultaneous-clocking of twelve Xilinx GTX-transceivers with a single clock source as a fallback solution. A feasible solution is presented.

The developed design is operable in the communication hierarchy of the PET-Scanner in terms of routing small data packets. However, at higher rates inacceptable data corruption occurs. This problem could not be solved due to time reasons.

Kurzfassung

FPGA-basierte Coincidence-Unit für Digital PET

Positronen-Emissions-Tomographie (PET) ist ein Bildgebungsverfahren im Nuklearmedizin, in dem Verteilungen von Materialien visualiziert werden können, die Positronen strahlen. Mit diesem Verfahren lassen sich metabolische Prozesse im Körper untersuchen, wie z.B. Glukosenaufnahme von Zellen und es wird meistens in der Onkologie eingesetzt, um Krebszellen zu lokalisieren, die wegen hoher Wachsrate viel Energie benötigen.

Für die Bild-Rekonstruktion werden Gamma-Photonen-Paare detektiert, die nach einer Annihilation von einem Positron mit einem Elektron in seiner Umgebung entstehen. Die zwei Gammas fliegen in umgekehrte Richtungen und werden mit kleiner Zeitabstand von mindestens zwei Gamma-Detektoren registriert. Wenn so ein Gamma-Paar detektiert wird, das zu einer Annihilation gehören, dann wird dieses Ereignis auch *Coincidence* genannt. Die Einheit, die nach solchen Gamma-Paaren sucht, wird *Coincidence Unit* genannt.

Ein aktuelles Forschungsprojekt im Philips Forschungszentrum Aachen namens *HYPERImage* beschäftigt sich mit der Entwicklung eines neuartigen PET-Scanners für Forschungszwecke, der in ein Magnet-Resonanz-Tomographie-Gerät (MRT) reingeschoben werden kann und simultane PET- und MRT-Bilder ermöglicht. Dieses simultane PET/MRT-Bildgebungsverfahren könnte bessere Bilder als andere hybride Bildgebungsverfahren erzielen, die bei der Untersuchung von metabolischen Prozessen verwendet werden.

In dieser Diplomarbeit wird eine Infrastruktur für eine FPGA-basierte Coincidence Unit speziell für diesen PET-Scanner entwickelt.

Die Arbeit enthält eine kompakte Einführung ins Thema PET und eine Beschreibung der Kommunikations-Infrastruktur des beschriebenen PET-Scanners, die relevant für das allgemeine Verständnis der Diplomarbeit sind. Danach wird das Konzept für die FPGA-Infrastruktur vorgestellt.

Der größte Anteil von der konzipierten Architektur wurde implementiert und getestet, die auch in dieser Arbeit dokumentiert sind. Es werden auch allgemeine FPGA-Entwicklungsverfahren erläutert, die auf Xilinx FPGA-Entwicklungswerkzeugen basieren.

Ein wichtiger Teil der Arbeit war die simultane Taktung von zwölf Xilinx GTX-Transceivern mit nur einer Taktquelle als Rückfalllösung. Eine akzeptable Lösung wird präsentiert.

Das entwickelte Design kann in der Kommunikations-Infrastruktur von dem PET-Scanner betrieben werden und ist in der Lage kleine Datenpakete erfolgreich zu routen. Bei höheren Sensordatenraten werden Daten teilweise korrumpiert. Dieses Problem könnte aus Zeitgründen nicht behoben werden.

Contents

1 Introduction

According to the global cancer statistics, estimated 12.7 million people were newly diagnosed with cancer and about 7.6 million deaths were caused from cancer in 2008 worldwide [1].

The diagnosis of cancer at its earlier stages increases the chance of cure, because it is easier to treat the malignant lesions, when they are not spread to a neighbor tissues due to metastasis.

Some symptoms of cancer are persistent fatigue, pain [2] and unexpected weight loss [3]. However, these symptoms can also have other reasons than cancer, which requires more costly and aggressive therapy for the patient. Therefore, it is important to identify the cause of the symptoms correctly.

An early diagnosis and correct identification of cancer requires imaging modalities which are sensitive enough to differentiate the cancerous cells from others. One of the imaging methods to evaluate cancer symptoms is Computed Tomography (CT), which can be used to determine and follow the progress of the cancer cells [4].

For more sensitive imaging of cancer cells a functional imaging method is needed, which can be used to identify the high amount of metabolic activity caused by the cancer cells. A possible method is nuclear imaging, where radionuclides are combined with specific substances, the latter being consumed by specific tissues, e.g. cancer cells. After this compound is administered to the patient, the radiation can be detected with special detectors and these specific tissues can be localized.

One of the imaging methods in nuclear medicine is Single-Photon Emission Computed Tomography (SPECT). In SPECT, the gamma radiation is detected by planar gamma detectors, which can be rotated around the patient to obtain 2-D images of the radiated gamma photons. The result is a 3-D image of the radionuclide distribution in the tissue.

Another nuclear imaging method with a better image quality is Positron Emission Tomography (PET). In PET, a positron is emitted by the tracer which annihilates with an electron of the surrounding tissue, resulting in two gamma photons which travel in opposite directions. These coincident photons are detected by two opposing gamma detectors. Compared to the detection of single gamma photons directly emitted by the radionuclides in SPECT, detection of coincident gamma pairs leads to a better localisation of radionuclides and hence to a better image quality.

Because of its higher sensitivity for assessing early metabolic changes compared to CT, PET is a more preferred method in oncology [5]. However, due to the lack of anatomical information in PET, both modalities are used in PET/CT scans, offering fused PET and CT scans. The same also applies to SPECT. Most of modern SPECT devices have a built-in CT.

Simultaneous MRI (Magnetic Resonance Imaging) and PET -so-called PET/MRI- is a newer hybrid imaging modality, which has several advantages over PET/CT. The main advantages are variable soft-tissue contrast and the capability to acquire simultaneous functional and anatomic information compared to sequential[1] PET/CT systems [6]. There are also evidences which show to an increased risk of cancer due to high dose of radiation, when a CT-scan is used for the diagnosis [7].

There are many technical challenges in the fusion of PET and MRI due to the magnetic fields present in the MRI. The main reason is the common detector used in PET, i.e. the Photomultiplier Tube (PMT), does not work in a magnetic field.

Fortunately, recent advancements in semiconductor technology led to the use of Silicon-Photomultipliers (SiPM) in PET [8]. Their compact size makes the integration in an MR-device possible and they can be operated in an magnetic field.

Compared to the older PET-devices with PMTs and analog front-ends, the newer systems with SiPMs allow early digitization of the sensor signals, which leads to more compact detectors with less pins and possibly less interference to noise.

Another advantage of the early digitization is the availability of raw sensor data at early stages and thus the possibility of early processing of the signals with dedicated digital electronics. Older PET-devices lack this advantage due to the loss of information caused by the analog signal corrections done by the front-end electronics.

On newer PET systems, the first stage of processing after the digitalization of sensor signals is mostly done on processors, e.g., searching for gamma photons with enough energy, i.e., *Singles*. Next step would be the search for Singles happened in a specific time window, i.e., *Coincidences*. These processing steps are called *Single* and *Coincidence Processing* respectively.

These processing steps can be done on reconstruction computers, which reconstruct the radionuclide distribution in the tissue.

The computers for preclinical[2] purposes can be high performance clusters, with significant amount of memory and processing power.

These computer clusters provide a versatile platform for the research and evaluation of reconstruction algorithms, however they have also some disadvantages. Because of their size they are not easy not handle and take too much space. Therefore, it makes sense to have dedicated processing unit for the system, e.g., Field-Programmable Devices (FPDs).

[1]Two different imaging modalities are used one after the other by shuffling the patient
[2]Testing with animals for research purposes

FPD is an integrated circuit used for implementing digital hardware that allows the end user to configure the chip to realize field-specific designs. An example for an FPD with a high logic capacity is Field-Programmable Gate Array (FPGA) [9].

The dedicated electronics populated with FPGAs would be much more compact compared to a computer cluster.

Not only the the size but also the processing power makes the FPGAs interesting for research. At the time of writing, an FPGA-based processing unit capable of processing 100 million Single/s exists [10], but a search of a computer cluster with a similar performance revealed no results.

The high processing capabilites of the FPGA-based processing units make also online-processing possible, which allows the reconstruction of an image at the time of a scan. This also makes the need for the storage of the detector-data unnecessary.

A current research project conducted in Philips Research Laboratories Aachen focuses on the development of a novel SiPM-based preclinical PET-scanner, which can be integrated in a 3T MR-scanner [6]. The SiPMs in this PET-scanner allow very early digitization of the detector-signals and hence early digital-processing of Singles or Coincidences.

In this thesis, an FPGA-based infrastructure for a Coincidence-processing unit[3] for this PET-device[4] will be defined, developed and tested.

Later, this infrastructure could be used as a test-platform for hardware-processing algorithms to show and proof the advantages of an FPGA-based Coincidence processing system for PET.

The remaining structure of the thesis is as follows:

Chapter 2 deals with the theoretical background and general working principles of PET. Additionally, technical terms important for the understanding of this work will be introduced in this chapter.

The work in this diploma thesis relies on constraints specific to a preclinical PET-system. The hardware-processing-infrastructure of this PET-system and the FPGA-based evaluation hardware for the Coincidence Unit will be described in chapter 3.

After these prerequisites, the concept for the FPGA-based CU-infrastructure is introduced in chapter 4. This includes the overall architecture and particular modules for the infrastructure. At the end of this chapter, a commissioning-test system is described, which can be used as a basic data generator for the CU.

Chapter 5 explains the implementation phase of this project. First, the software tools and FPGA-design flow used in this project are introduced. Afterwards, the design strategy for the defined modules and the differences between the realisation and implementation of these modules are explained. At the end, an example for a design challenge in terms of FPGA-development is given.

[3]Also called Coincidence Unit (CU)

[4]To be introduced in chapter 3

Verification is an important part of the FPGA-development flow. Chapter 6 gives a small example for a simulator-based visual verification, which was used throughout the development phase.

Chapter 7 describes the commissioning-tests made with the developed design and their results. Finally, chapter 8 gives a discussion about the results, overall summary and outlook for this project.

2 Positron Emission Tomography

General principles of Positron Emission Tomography (PET) are important to understand the purpose of the Coincidence Unit (CU). Therefore, the basic working principle of PET is briefly described in this chapter.

2.1 Principle and Applications

PET is a nuclear-imaging method that can visualize metabolic processes in tissue. This is achieved by injecting radioactive labeled substances in the body of the subject. This material can be a molecule that is metabolized in a specific process e.g. glucose. The choice of the material depends on the metabolic process to be monitored.

Some application areas of PET are identification of cancer cells in cancer diagnosis, monitoring of the heart muscle function in cardiology and visualization of brain functions in neurology. [11, 2]

For example, if Fluorodeoxyglucose[1] is injected into a patient, it is taken up by all living tissues, especially by those that need a lot of energy. Cancer cells in particular fall into this category because they are growing faster compared to other cells.

These cells can be located by detecting the gamma rays created during the radioactive decay of radionuclides. The detection of these gamma rays is described in the next section.

2.2 Physics and Image Acquisition

Unstable radionuclides used in PET undergo a β^+ decay to reach a stable state and during a β^+ decay, a proton is converted to a neutron, a positron, a neutrino and kinetic energy.[11, 21]

$$p^+ \rightarrow n + e^+ + \nu + \text{kinetic energy} \tag{2.1}$$

[1]a glucose analog carrying radioactive isotope fluorine-18[12]

The resulting positron travels a short distance in tissue till it interacts with other nuclei. Predominantly, this interaction is an inelastic collision with an atomic electron along its path. As a result, the positron and the electron annihilate and electromagnetic radiation in form of two gamma rays is created. The energy of these two gamma rays can be calculated with Einstein's Special Theory of Relativity which says that mass and energy are equivalent. Given that an electron and a positron have a rest mass of $9.11 \times 10^{-31}\, kg$ each, the energy of the gamma rays can be calculated to:

$$\begin{aligned} E &= mc^2 \\ &= 9.11 \times 10^{-31}\, kg \times (3 \times 10^8\, m \cdot s^{-1})^2 \\ &= 8.199 \times 10^{-14}\, kg \cdot m^2 \cdot s^{-2} \\ &= 8.199 \times 10^{-14}\, J \\ &= 8.199 \times 10^{-14} \times \frac{1\, eV}{1.602 \times 10^{-19}} \\ &= 511\, keV \end{aligned} \tag{2.2}$$

These two rays ideally travel in opposite directions. But, it is also possible that they scatter due to interaction with atomic electrons on their path. This leads to a partial loss of energy and change of the ray direction.

Figure 2.1 summarizes the positron and gamma ray generation process.

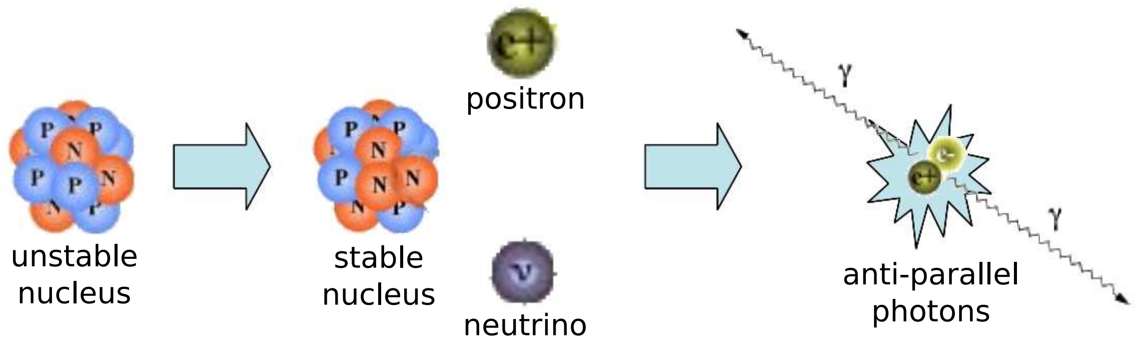

Figure 2.1: Positron and gamma creation process

The very small wavelength of the gamma rays and their massless nature makes it possible to easily travel through the atom nuclei and thus surrounding tissue with less interaction, but it also makes it difficult to stop them and measure its energy [11, 23]. It is shown that the interaction probability of gamma photons with materials decreases with increasing energy of the photons in range of 1 keV to 1 MeV [13]. Therefore, a material with a high atomic number (i.e. LYSO[2] crystal) is used as a scintillator to stop these rays and convert them to visible light.

[2]Cerium doped Lutetium Yttrium Orthosilicate

The visible light can be converted to electrical current by using Photomultiplier Tubes (PMT) or Avalanche Photo Diodes (APD). The output signal of these sensors is used to extract energy, position and time information of the encountered photons. A detected gamma photon in the context of PET is called *Single*.

To gather the three-dimensional location of the positron-electron annihilation, a second Single must be detected which belongs to the same annihilation as the first one. If such a pair is found this event is called a *Coincidence*.

Mostly, a ring sensor geometry is used to detect the coincidences. Figure 2.2 shows the ring geometry for PET sensors.

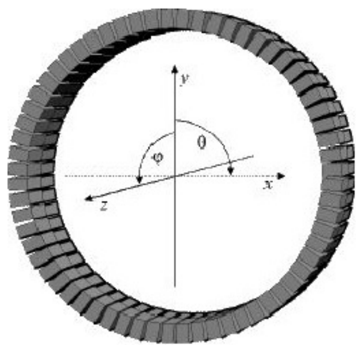

Figure 2.2: Commonly used PET sensor placement: ring geometry. Each of the grey boxes visualizes a photon-detector sensor

Data measured via detector modules is sent to a central processing unit which is called *Coincidence Processing Unit*. It is responsible for finding coincident Singles.

From a temporal point of view, Coincidence means that the Single events must have occurred in the same time window. This so-called *Coincidence window* depends amongst others on the scanner size and is approx. 3-4 ns for a full-body scanner. [11, 35] Figure 2.3 shows PET sensors placed in a ring geometry and the coincidence detection principle.

If a Coincidence is detected, a line of response (LOR) can be drawn which reduces the set of possible annihilation locations to a line. Therefore, at least two coincident events must be detected to calculate an annihilation point. If the timestamp resolution of the detectors is fine enough then the annihilation can be located more precisely. This is achieved by calculating the timestamp difference and travel distance difference of two coincident singles. Subsequently, a probability graph can be drawn on the LOR considering that the sensors have a limited time resolution. This method is called Time of Flight (TOF) and visualized in figure 2.4.

After the successful detection of the coincidences the image of the radionuclide distribution in the tissue can be reconstructed.

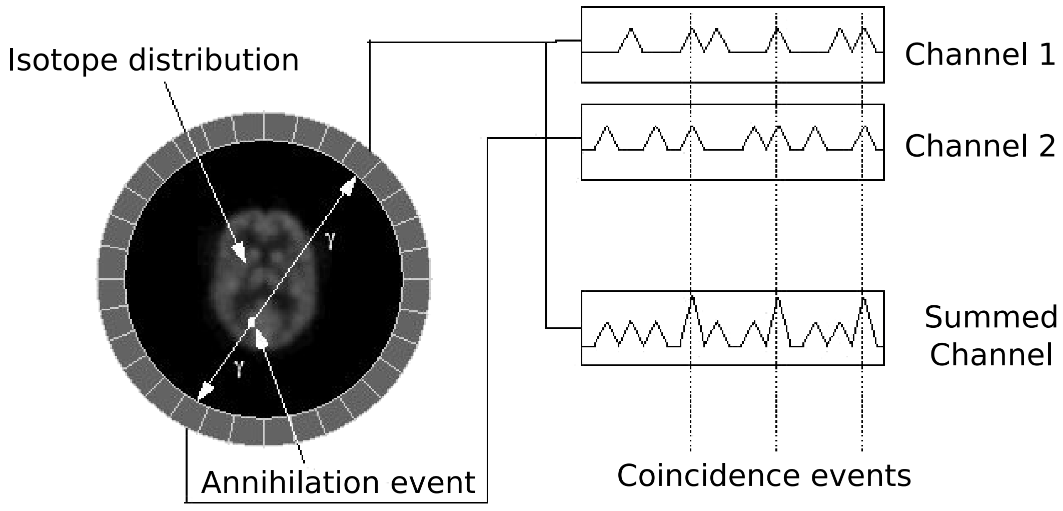

Figure 2.3: The two signals *Channel 1* and *Channel 2* are added to search for Coincidences. Note that the signal amplitudes are not to scale. In this figure, the signal peaks in the summed signal that have the amplitude of a gamma ray are not selected as coincidence events. However, due to scattering the gamma rays can lose half of their energy and the summed signal can only reach 511 keV, which is also a coincidence event and this information can be used in a research environment.

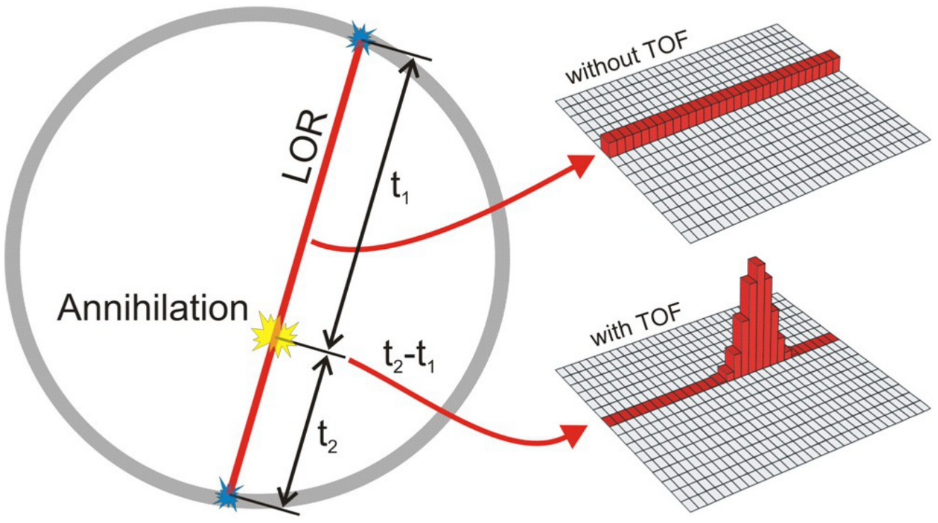

Figure 2.4: The TOF method allows to determine the annihilation position with a single Coincidence event only. The bar graphs show the annihilation probability at different sections on the line of response with and without the TOF method.

3 Existing Infrastructure

The CU is designed for a PET system which is developed as part of a major project called *HYPERImage*. Part of the Hyperimage hardware architecture, that is relevant for the CU design is described in this chapter. Additionally, the FPGA-based processing platform is introduced which the CU is developed on.

3.1 HYPERImage Project

HYPERImage is a partial European Union funded research project which is implemented by various research institutes and Philips. The aim is the development of a proof-of-concept system for simultaneous whole-body Positron Emission-Magnet Resonance Tomography (PET-MR) imaging for humans. [14]

As part of this research project, a preclinical PET device for small animals is developed that can be inserted in an MR device for experimental purposes. The communication infrastructure of this insert is introduced in the following subsection. From this point on the name *Hyperimage* will be used to refer to the communication infrastructure of this insert.

3.1.1 Communication Infrastructure

The core of the image acquisition system consists of five types of devices which are connected to each other in a tree-hierarchy. These are:

- Backbone
- Sensor Stack
- Single Processing Unit (SPU)
- CU PC
- Control PC

Figure 3.1 shows the Hyperimage communication infrastructure. Because of the tree hierarchy, every packet can flow in two directions only. These are called *downstream* for packets in direction of the Control PC and *upstream* for packets in direction of the sensor stacks.

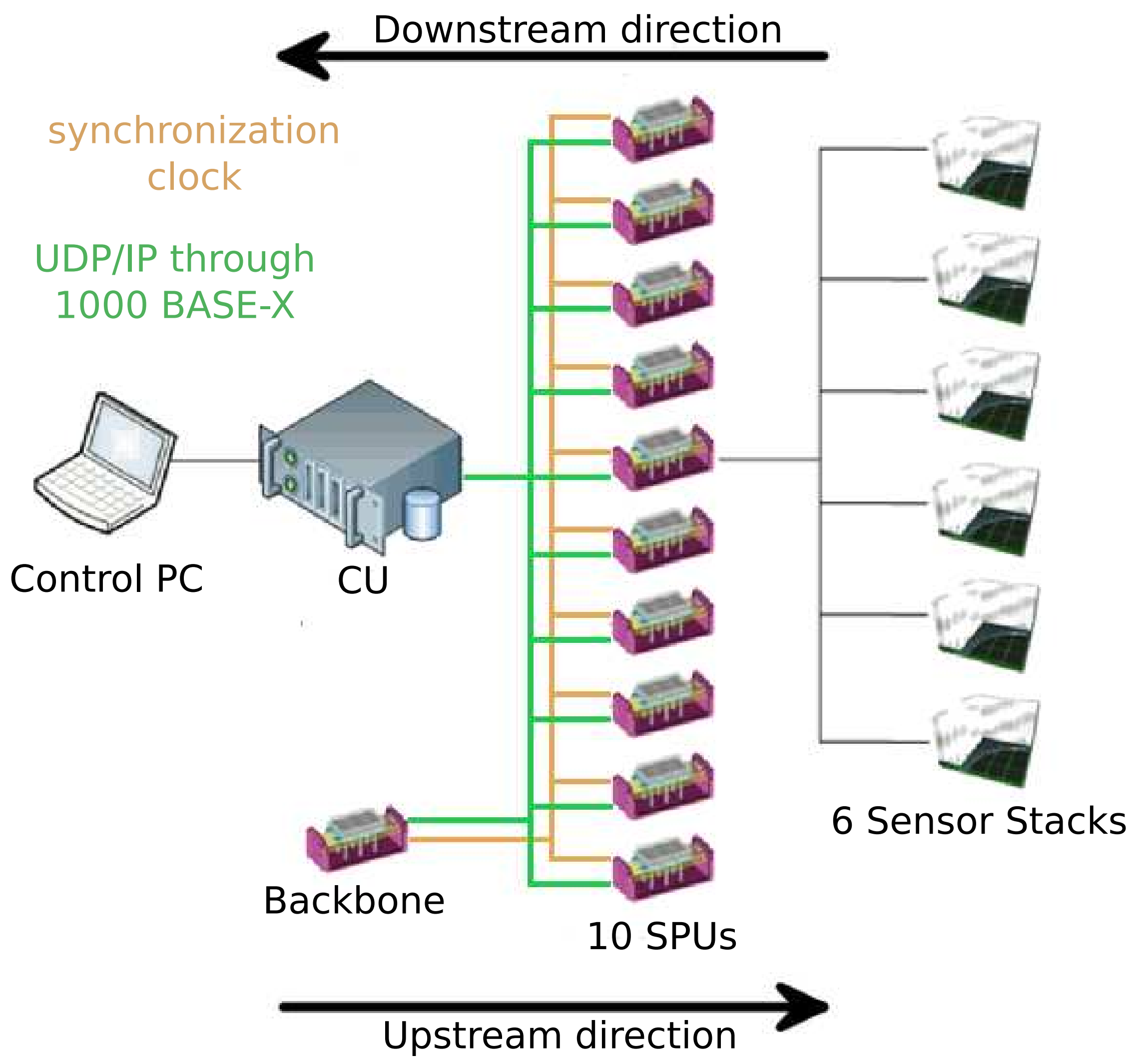

Figure 3.1: Hyperimage communication infrastructure

3.1.1.1 Backbone

As mentioned in section 2.2, Coincidence means that the Single events must have occurred in the same time window. Technically, this means that the Single timestamps from different detector modules are compared to each other to find out if their time difference is lower than a specific limit.

For the precision of the Coincidence calculations in Hyperimage, the synchronisation of the clocks on different Sensor Stacks is important, because the timestamp counters on the Sensor Stacks are also incremented with this clock signal. Therefore a central unit is needed for synchronisation purposes. This unit is called *Backbone* and is an SPU module with a slightly altered design which incorporates additional clocking elements like clock distributors and dividers. It is connected to all SPUs in a star geometry.

The Backbone's main functions regarding the detector data readout are:

- Generation of the central 625 MHz clock signal for the Sensor Stack ASICs.
- Generation of the central 156.25 MHz clock signal for the Interface FPGAs by dividing 625 MHz clock signal.
- Generation of the ASIC timestamp counter synchronisation signal, which triggers a synthetic *Hit*[1] event on the ASICs. Subsequently, the timestamp counters on all ASICs are read out. These read-out values are used as timestamp offset values, which are subtracted from the respective channel timestamp counters. At the time of writing this correction is carried out first at image reconstruction phase.

  This correction is needed even if there is a central clock and reset for the ASICs, because PLLs on different ASICs require different times to lock on the input clock signal. Different lock durations again lead to different initialisation times of the ASICs, thus leading to different counter start times.
- Triggering of the Hit read-out from the ASICs.

3.1.1.2 Sensor Stack

A Sensor Stack consists of a scintillator array and three stacked boards: Silicon Photomultiplier (SiPM) board, SiPM ASIC board and Interface FPGA board.

As described in chapter 2, gamma rays are hard to detect because of their high energy. The scintillator arrays convert incoming highly energetic gamma rays into visible light photons with lower energy. A single gamma photon is converted to the number of visible light photons which corresponds to the energy of the gamma quantum.

The SiPM board consists of an array of SiPMs. Each SiPM has an array of avalanche photodiodes which converts the hitting light photons into electrons and amplifies them to get a measurable electrical current.

The SiPM ASIC board is attached to the SiPM board from the bottom side and acts as analog-to-digital converter (ADC) and time-to-digital (TDC) converter. The ADC measures the released charge which is proportional to the number of scintillation photons and consequently cumulated photons' energy. The TDC captures a timestamp of the first detected photon in terms of a photons hit in a specific time window.

At the bottom side of the stack is the Interface FPGA board. It configures the ASICs, reads out sensor data, receives commands from the SPU and sends sensor

[1]In Hyperimage, a set of detected light photons in a time window that have a total energy greater than a specific threshold qualifies to a Hit. It is the most basic event in Hyperimage which is timestamped.

or status data in direction of the SPUs. Figure 3.2 shows a sensor stack with its layers.

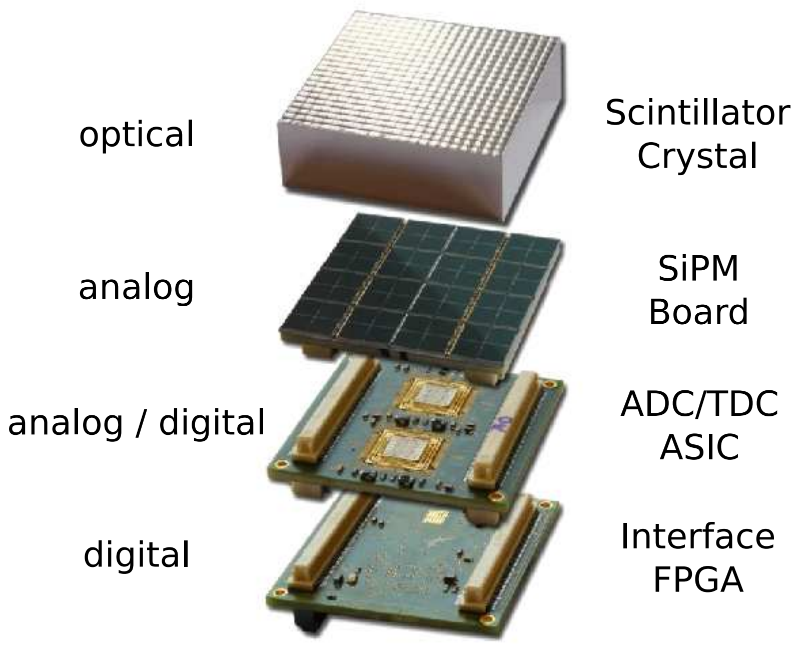

Figure 3.2: Sensor Stack layers and signal types at respective layers

3.1.1.3 Singles Processing Unit (SPU)

The SPU is an FPGA-based board which performs various tasks like processing and forwarding the data coming from the Sensor Stacks, routing the command packets coming from the Control-PC, setting the threshold voltages of the sensor stacks and measuring signals like temperature of the stacks against over-temperature, supply voltages against over-voltage as well as acceleration to detect vibration caused by magnetic fields of the MRI. Up to six sensor stacks can be attached to one SPU board.

Due to the conversion of one gamma photon to a set of light photons and the propagatiion of these photons to neighbor crystal arrays, not only one SiPM but many are triggered and subsequently Hit events are generated. In most cases the triggered SiPMs are neighboring.

The processing power of the SPU makes it possible to perform Single calculations that determine the real hit point of the gamma particle. The SPU has the infrastructure to perform Singles processing, which explains the unit's name. But, at the time of writing, this feature is not implemented and Sensor Stack measurement data (Hits) is merely forwarded.

The SPUs are connected to the Coincidence Unit via fiber optical cables which operate on IEEE 1000BASE-X standard. Each link is capable of 1 Gbit/s bidirectional data transfer.

The left picture in figure 3.3 shows an SPU board in a plastic enclosure with a whole and two partial sensor stacks attached.

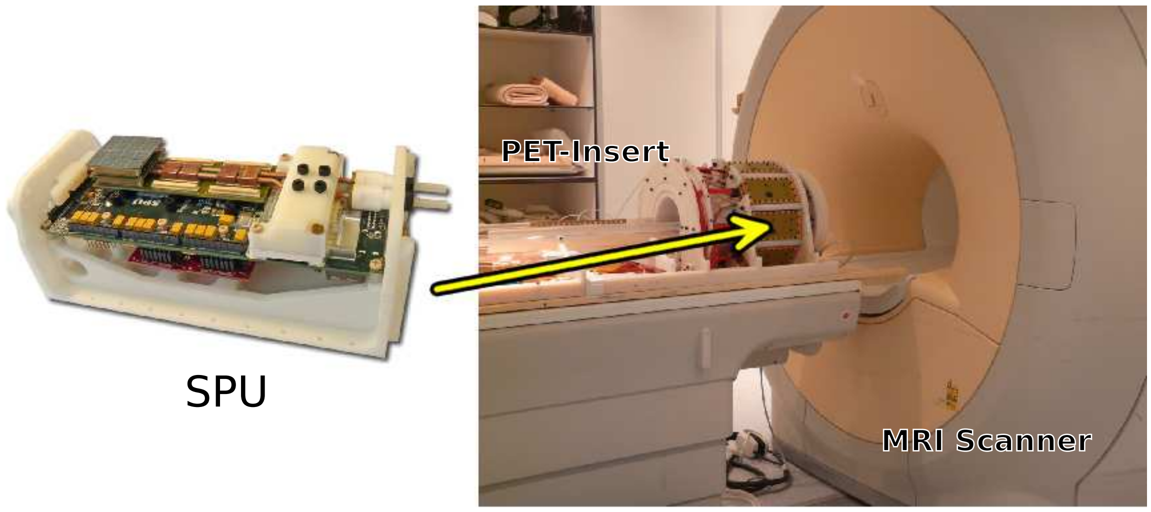

Figure 3.3: An SPU board with one complete sensor stack attached and the Hyperimage PET insert which fits in a clinical MR device.

3.1.1.4 Coincidence Unit (CU)

The CU is a processing platform for coincidence search. Its main function is to receive, store and process the sensor read-out data coming from the SPUs.

During a measurement every SPU can generate data of up to 700 Mbit/s. Therefore, storing and processing the Singles data coming from ten SPUs requires high amount of RAM and extensive computing power respectively. For this purpose a Dell PowerEdge R815 is used.

The server has four AMD Opteron 6174 processors running at 2200 MHz and 64 GB RAM. It is capable of processing approximately 10 million Singles per second. At the time of writing, processing is done offline: First, the sensor read-out data from the SPUs is stored on the harddisk. After the measurement, the coincidence processing is started.

3.1.1.5 Control-PC

The Control-PC is a workstation which acts as a user interface. It is responsible for commanding the whole PET system. The software running on the system is called *Hyperion*.

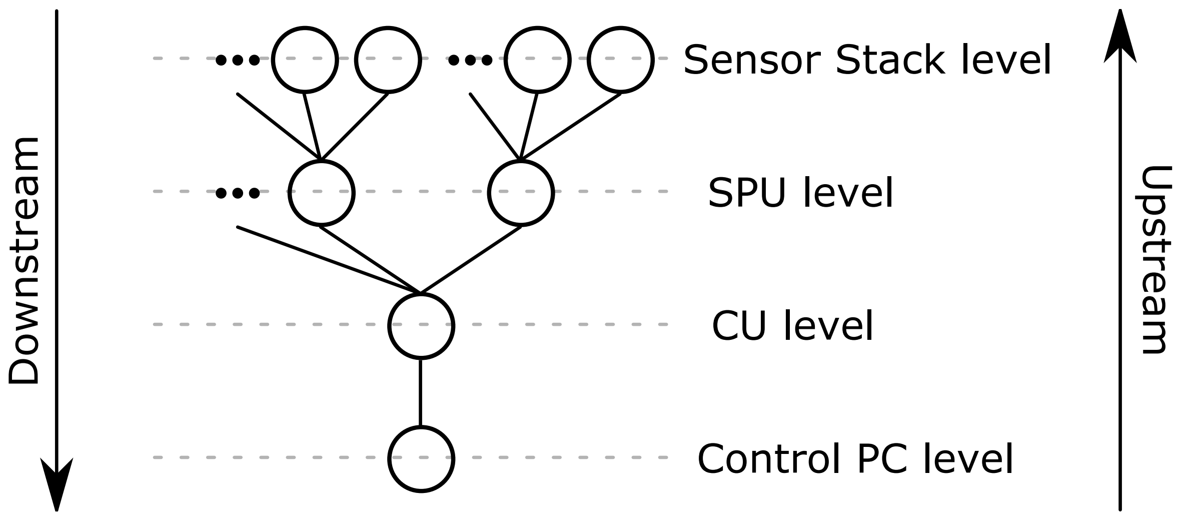

Figure 3.4: Hyperimage communication hierarchy as tree diagram with various levels.

Hyperion is the control software for the PET system with a GUI frontend. Its main purpose is to configure the devices, showing device statistics and to start a measurement.

Control-PC is directly connected to the CU-PC which routes the packets to the SPUs.

3.1.2 Communication Protocol

In this subsection the communication hierarchy and structure of a Hyperimage communication packet is explained. Figure 3.4 shows the communication hierarchy as a tree diagram.

In Hyperimage there are two kinds of data:

- Read-out
- Control/Status

Read-out data packets contain measurement data read out from the sensor stacks for image reconstruction. This is the datatype which causes most of the bandwidth. It only flows in downstream direction.

The rest of the packets are control packets for sending commands to the modules and status packets generated as an answer to a command packet or in case of a sudden error.

Figure 3.5 shows the Hyperimage packet structure.

The packet header comprises of the Preamble, Source or Destination Address, Sequence Number, Length and CRC fields.

	Header							Payload	(Trailer)
	Preamble	Source or Destination Address			Seq. Nr.	Length	CRC		
		(SPU)	(Stack)	Module					
field length	3	(1)	(1)	1	1	2	2	up to 2^{16} - 1	(3)

Figure 3.5: Hyperimage packet structure. Existence of the fields in brackets depend on where the packet currently travels.

The address fields *SPU*, *Stack* and *Module* indicate the source or destination address of the packet depending on in which direction the packet currently travels. Either source or destination address is sufficient. This is possible by two properties of the communication architecture. Firstly, every addressable entity is connected to another entity via point-to-point connection and knows what kind of entity it is connected to in the hierarchy. Secondly, the entities are only controlled only by lower entities. In other words, no control packets travel in downstream direction but only status packets and read-out packets. As status packets and read-out packets can be used by every higher entity they do not need a destination address. It is similar in the upstream direction, where a higher entity does not send any status or read-out packets to a lower entity but only control packets and a lower entity does not need the information from which level exactly a packet is originating. It forwards it to another module or runs the command depending on the destination address.

In summary, the address field contains the source address in downstream and the destination address in upstream direction.

The addresses for the SPUs, Sensor Stacks and modules start with number 1. The address *zero* has a special meaning regarding the address and the type of the packet.

A zero in the *Module* field means that the payload contains read-out data. Read-out packets travel only in downstream direction and can be created by all entites in the hierarchy: by Sensor Stacks, SPU or CU.

Read-out data has a slightly different meaning depending on which device it is created on. Sensor Stacks generate raw sensor data. SPUs can (after processing of a set of hit-data packets) generate a single *Singles packet.* The CU can in turn generate a single Coincidence packet after processing a set of Singles data packets. All this data originating from the detectors is called read-out data and therefore marked with the Module field being zero.

A zero in the Stack or SPU field has different meanings in upstream and downstream direction.

In downstream direction it means that this packet is not generated by a device at this entity level but at a higher entity in the communication hierarchy. For example, if there is a zero in the *SPU* field of a downstream packet then this packet doesn't originate from an SPU.

In upstream direction it means that this entity is not addressed. For example if the CU receives an upstream packet with a zero in the SPU field then it forwards it to a CU-internal module.

Another aspect of the address field is its dynamic size. It grows in downstream direction and shrinks in upstream direction. This is made possible by the tree hierarchy of the communication architecture. If a packet travels down in the tree hierarchy then fields belonging to higher entities are not needed, so they are cut off from the address field. For example, if the CU forwards an upstream packet to an SPU, it removes the SPU field, because, when the packet arrives at the SPU level of the hierarchy there is only one SPU at this level.

Dynamic size of the address field results in smaller packets in upper levels, especially at Sensor Stack level, where most of the bandwidth is generated by the detected Hits. This feature reduces the maximum needed bandwidth in bottom levels.

The packet size grows again when packets travel in downstream direction due to address field additions, but this increase in packet size is compensated again via Coincidence and Single processing. This is achieved e.g. by deleting packets at the SPU level that belong to hits with not enough energy or Singles that do not belong to a Coincidence event.

Table 3.1 gives some packet examples with their source, destination and type.

The Sequence Number is incremented every time when a device generates a new packet. This eases the debugging process in case of lost or defect packets.

The two byte wide Length field is the total payload length without trailer.

The last two bytes of the header are CRC-data over the header which ensures packet integrity.

The Payload is variable in size and can hold up to 65535 bytes. It contains read-out, status or control data.

The Payload is appended by a Trailer. It eases the synchronisation of the state machines, which are responsible for packet parsing. The trailer only exists in downstream direction, because in this direction most of the data packets are generated, which leads to packet loss and corruption.

3.2 Platform for the FPGA-based Coincidence Unit

As the coincidence unit has to deal with data rates up to 7 Gbit/s (in case of unprocessed Hit data) during a PET acquisition, high demand is placed on the coincidence processing unit. An investigation on PET processing platforms shows that FPGA based processing platforms based on Xilinx Virtex-2 or Virtex-5 FPGAs exist that can process up to 100 million Singles per second [15][16]. Consequently, a similar

Direction	SPU	Stack	Module	Tree Branch	Explanation
Upstream	0	0	1	Control PC - CU	Command packet for module 1 in CU
Upstream	1	0	5	Control PC - CU	Command packet for module 5 in SPU-1
Upstream	-	0	5	CU - SPU	Command packet for module 5 in SPU-1
Upstream	2	3	10	Control PC - CU	Command packet for module 10 in Stack-3 of SPU-2
Upstream	-	-	10	SPU - Sensor Stack	Command packet for module 10 in the Stack of current branch
Downstream	0	0	0	CU - Control PC	Read-out packet generated by CU
Downstream	-	0	0	SPU - CU	Read-out packet generated by the SPU of current branch
Downstream	3	0	0	CU - Control PC	Read-out packet generated by SPU-3
Downstream	2	1	0	CU - Control PC	Read-out packet generated by Stack-1 of SPU-2
Downstream	-	-	7	Sensor Stack - SPU	Status packet generated by Module-7 in the Stack of current branch
Downstream	2	1	7	CU - Control PC	Status packet generated by Stack-1 of SPU-2

Table 3.1: Some Hyperimage packet examples. All upstream packets are generated by the Control PC. Tree Branch shows on which branch the packet currently travels.

	Virtex-6 XC6VLX240T
Logic Cells	241,152
Slices	37,680
Block RAM	14,976 Kb
Ethernet MACs (TEMAC)	4
GTX transceivers	24

Table 3.2: Virtex-6 XC6VLX240T specifications

platform based on a new generation of Virtex is chosen to implement the processing infrastructure.

The FPGA-based coincidence unit is based on the ML605 Virtex-6 evaluation board from Xilinx. The board incorporates a Virtex-6 XC6VLX240T FPGA. Table 3.2 shows some specifications [17][18].

For the 1000BASE-X gigabit connectivity to the SPUs, two custom daughter boards with total of twelve Stratos RJS-ST31 fiber optical transceiver modules are used. Ten of the ports are connected to ten SPUs, one to the Backbone and the last one to the Control PC. These are connected directly to the GTX transceiver ports of the Virtex-6 FPGA. Figure 3.6 shows the ML605 board with the two custom daughter boards.

A GTX transceiver is an integrated circuit on the Virtex-6 FPGA to meet the analog and digital signaling requirements of many communication standards including Gigabit Ethernet, which is also used for the connectivity in Hyperimage.

The Virtex-6 silicon also embeds Tri-Mode Ethernet MAC (TEMAC) blocks which implement the Media Access Control (MAC) sublayer specifications of the ISO OSI Layer 2. These hard-wired integrated blocks can be used for ethernet connectivity and allow to save generic logic resources on the FPGA.

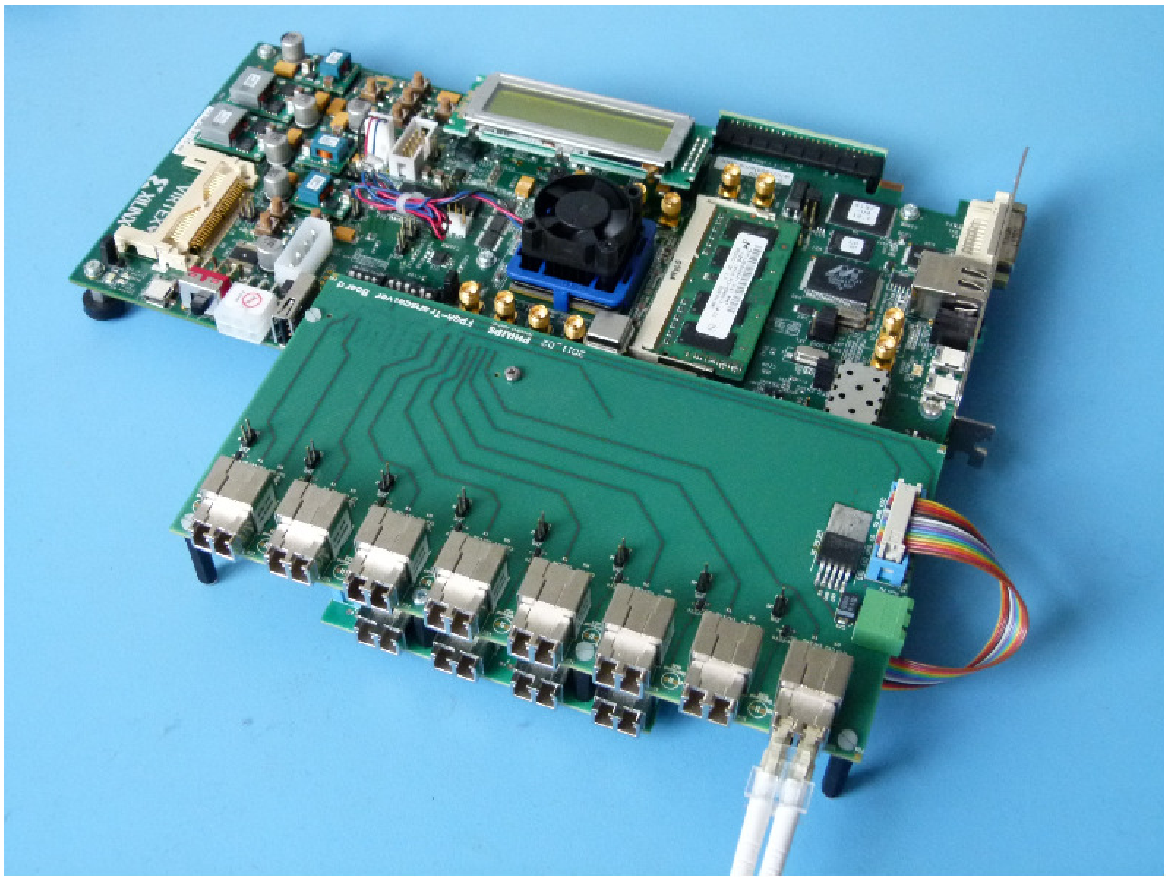

Figure 3.6: FPGA-based Coincidence Unit

4 Concept

After the analysis of the existing communication-hardware and -architecture, a concept for the CU was thought up, which fits to the communication hierarchy in Hyperimage and provides infrastructure for Coincidence-processing. Due to the existing FPGA-architecture of the SPU, most of the architecture concept is based on the SPU-FPGA-architecture.

During the implementation phase, SPUs for testing the CU were not available. Therefore an SPU Emulator was designed as temporary testing device.

This chapter presents the proposed CU-architecture with its modules in detail. At the end the SPU-Emulator is introduced.

4.1 CU Architecture

As a part of the hardware chain in the Hyperimage, the CU provides blocks to deal with the Hyperimage communication protocol, for Coincidence-processing, for user-interaction and controlling as well as querying the CU.

In this section, first an overview of the proposed CU architecture is given, then the CU-modules are described more in detail.

4.1.1 Overview

The analysis of the Hyperimage hardware infrastructure leads to the conclusion that it is necessary for the CU to implement these functions at minimum:

- Packet routing between Control-PC and SPUs
- Infrastructure for Coincidence processing

These requirements lead to the following architecture shown in figure 4.1, which is going to be explained in the next sections.

Gigabit Ethernet modules on both sides handle the UDP/IP communication based on 1000BASE-X. These modules are able to pack Hyperimage packets in an ethernet frame and vice versa. There is one Gigabit Ethernet block for each SPU, the Backbone and the Control-PC.

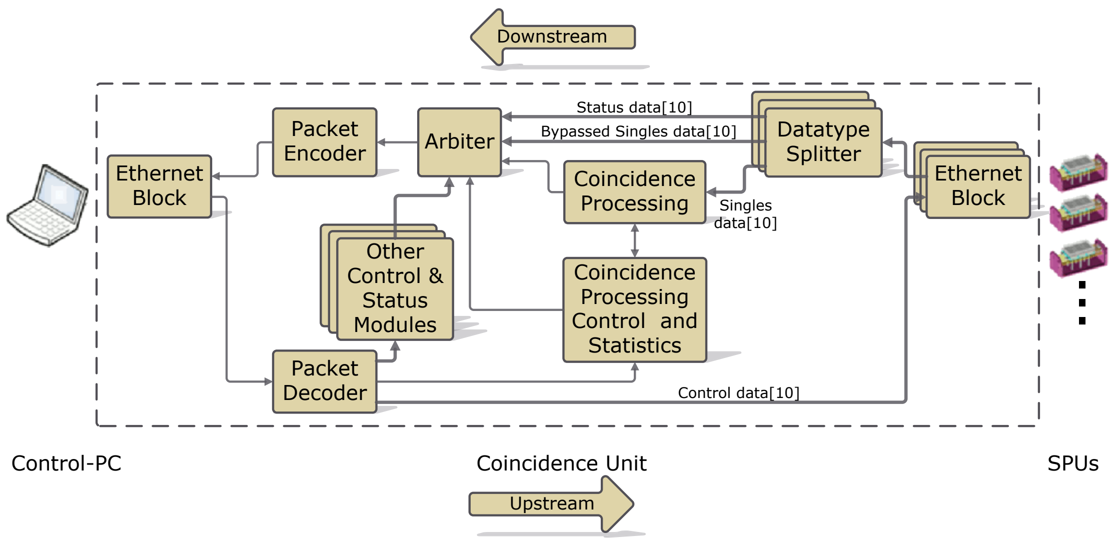

Figure 4.1: Overview of the CU architecture

Next to the Gigabit Ethernet module on the upstream side, there is the Packet Decoder which is responsible for routing the command packets coming from the Control-PC and stripping unneeded header parts. The destination can be an internal module or an external module in the SPU, Backbone or Interface Board.

The downstream datapath carries read-out and status data. Before doing any coincidence processing, Singles data and status data are separated by the Datatype Splitter. This way, Singles data can be sent to the Coincidence Processing module and status data to the Control-PC with least delay. This separation also has the advantage to implement a priorisation between read-out and status data, as status data can be very crucial on emergency situations like over-temperature in an SPU.

The Datatype Splitter can output the read-out data in two versions: as bypassed- or normal Singles-data. The latter contains only the payload of the read-out packet, which can be immediately processed by the Coincidence Processing module. The bypassed Singles-data output is for Coincidence-processing in the Control-PC.In this case the header of the packet is not stripped off for easy forwarding.

Data destined for the Control-PC is bundled in the Arbiter. It collects data from internal as well as external modules and handles the access to the Control-PC path.

The packet encoder generates a valid Hyperimage packet from the incoming datastream. It adds header data according to the addressing mechanism described in subsection 3.1.2.

The Ethernet Block, Packet Encoder, Packet Decoder, Arbiter, Datatype Splitter are called *core modules*. There are also other infrastructure blocks to control or query the core modules. One example is the Coincidence Processing Control and Statistics block which can alter parameters or get statistics about coincidence processing.

4.1.2 Core Modules

The core modules deal with the communication, which includes Ethernet and UDP/IP communication with other entities in the hardware hierarchy and also internal modules, which encode, decode and prioritize the Hyperimage packets.

This subsection describes the function of the core modules more in detail.

4.1.2.1 Packet Decoder

The main task of the Packet Decoder is to parse the incoming packets coming from the Control-PC and route them in appropriate locations. The destination can be an internal module, such as Control and Status modules, or an external module, in case of a module in an external entity, i.e. the SPU.

The destination is determined by reading the header data. After that, the unnecessary parts of the header are stripped off (see subsection 3.1.2).

There are three kinds of destinations, which the header stripping is dependent on:

- In case of an external destination only the SPU field is stripped off.
- In case of basic internal modules all the packet fields but the payload are thrown away.
- More complex internal modules with larger data input may use FIFOs for their input. In this case an additional two byte length field is sent.

The output of the Packet Decoder is connected to the Ethernet blocks on the SPU-side and to the internal control and status blocks. There is an enable signal for every of these blocks which signalizes that the enabled module shall accept the decoder data.

Figure 4.2 visualizes how the packets are processed in the Packet Decoder.

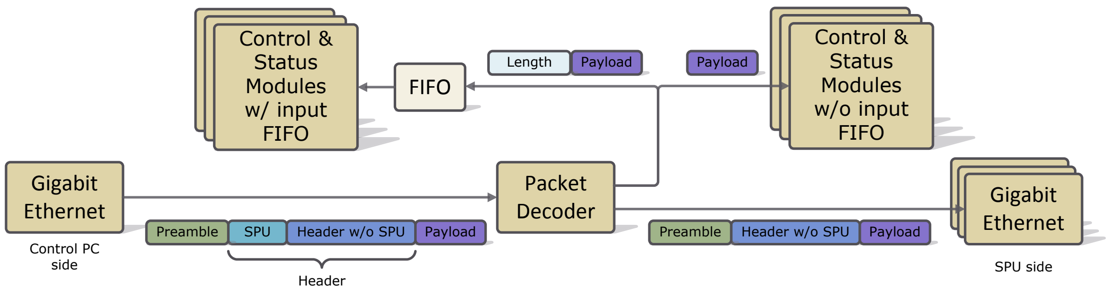

Figure 4.2: Packet processing in Packet Decoder

Another task is assuring the integrity of the received packets. If a packet is corrupted during transmission then it is not routed. This check is done firstly by checking the valid header structure like correct preamble or a correct device-id in according header fields. Secondly, the CRC must be valid for the header. If these two conditions are valid, the packet qualifies to be forwarded to a next destination. If they are not met, the state machine searches for a new packet header in the incoming datastream.

The payload to be forwarded is determined from the flowing datastream by decrementing the payload counter, which was initialized with the Length field. When the counter is fully decremented, then the according module enable signal is deasserted and state machine waits for a preamble. In other words, the packet boundaries are determined by valid header structure, correct CRC and Length field.

In case of a corrupt header the Error Handler module (described in 4.1.2.6) is activated and the Control-PC is informed about the corrupt packet.

4.1.2.2 Packet Encoder

The Packet Encoder is responsible for adding a valid header to the data or modifying existing headers of the data coming from the Arbiter.

If the data originates from an internal module then a complete header including trailer is added according to the source of the data supplied by Arbiter. Figure 4.3 shows packet processing in Packet Encoder in case of data originating from internal modules.

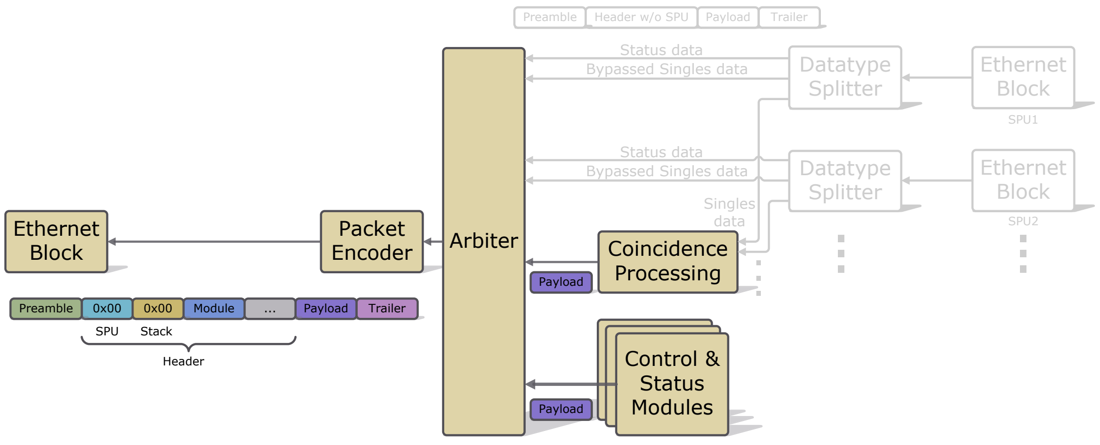

Figure 4.3: Packet processing in Packet Encoder in case of data originating from internal modules. SPU and Stack fields are zero, because the payload originates from a CU-internal module.

If the data is coming from an SPU then only the address of the SPU is added and the CRC recalculated. Figure 4.4 shows packet processing in Packet Encoder in case of data originating from an external module.

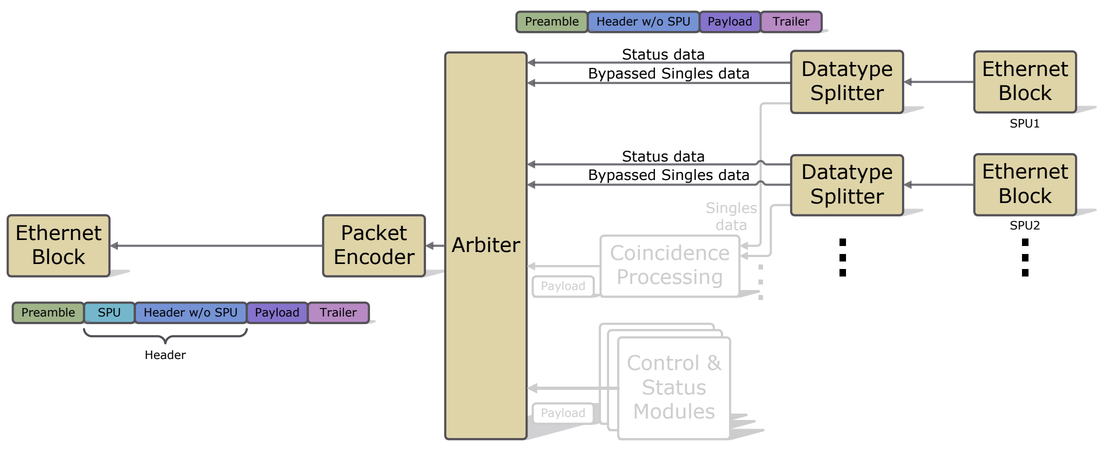

Figure 4.4: Packet processing in Packet Encoder in case of data originating from an external module

Incoming data is not checked for integrity because the data coming from the Arbiter is expected to be valid.

4.1.2.3 Arbiter

The Arbiter controls access to the the Gigabit Ethernet and Packet Encoder data path on the Control-PC side, because this path is a shared resource. It is used by all modules that forward data to Control-PC.

If a module wants to send data, it asserts a request and waits for an acknowledge from the Arbiter.

During a measurement, most of the data in Control-PC direction is created by the Stacks and therefore consist of read-out data. Thus, most of the time, Datatype Splitter or Coincidence Processing block shall raise a send request. This would lead to a delay on status data which originates from internal or external modules and cause a delayed reaction by the Control-PC on emergency cases like high voltage or temperature. Therefore the Arbiter prioritizes status data higher than read-out data when processing the requests.

4.1.2.4 Datatype Splitter

Data coming from the SPUs contain read-out as well as status data. Therefore the datastream must be separated into two paths and only read-out data is fed to Coincidence Processing module.

Another task of this module is the data integrity, as packets being sent over UDP/IP can be corrupted or fragmented. Packets with wrong header or CRC are thrown away and the Error Handler is activated.

The Ethernet Block can generate fragmented Hyperimage packets. Assuming unfragmented packets for data processing infrastructure modules greatly reduces design efforts for these modules, as in this case additional checks on packets or waiting for the rest of a packet don't have to be implemented. Therefore the Datatype Splitter ensures data integrity and that no fragmented packets are forwarded to the destination modules. This can be done by buffering a packet and waiting for the fragmented part and checking for integrity before forwarding it to the next module.

4.1.2.5 Ethernet Block

The Ethernet Block handles the data exchange between CU and external devices like SPUs or Control-PC. In transmit direction, it wraps the Hyperimage packets with a UDP/IP and MAC header and generates signaling for a glass fiber transceiver. In receive direction, the output of the transceiver is read, the Hyperimage packets are extracted from ethernet frames and stored in a FIFO for further processing. The data in this FIFO can then be consumed by other modules like the Frame Decoder or the Dataype Splitter.

It comprises of two main modules:

Media Access Control and Physical Layer

The Media Access Control and the Physical Layer implements the requirements of the data-link and physical layer of the OSI-Model layers.

These modules are available as IP-Cores from Xilinx. Therefore, this section contains only a short description about these modules.

Figure 4.5 shows an overview of these layers and their placement in OSI-Model.

The Physical Medium Dependent (PMD) layer performs signaling on the physical medium and is implemented by the fiber-optical transceiver.

The Physical Medium Attachment (PMA) layer is responsible for octet level synchronisation and scrambling/descrambling.

The Physical Coding Sublayer (PCS) performs auto-negotiation and 8b/10b coding and decoding.

The physical layer is implemented by the PMD, PMA and PCS modules. The MAC layer resides in the data-link layer and is responsible for addressing in a subnet.

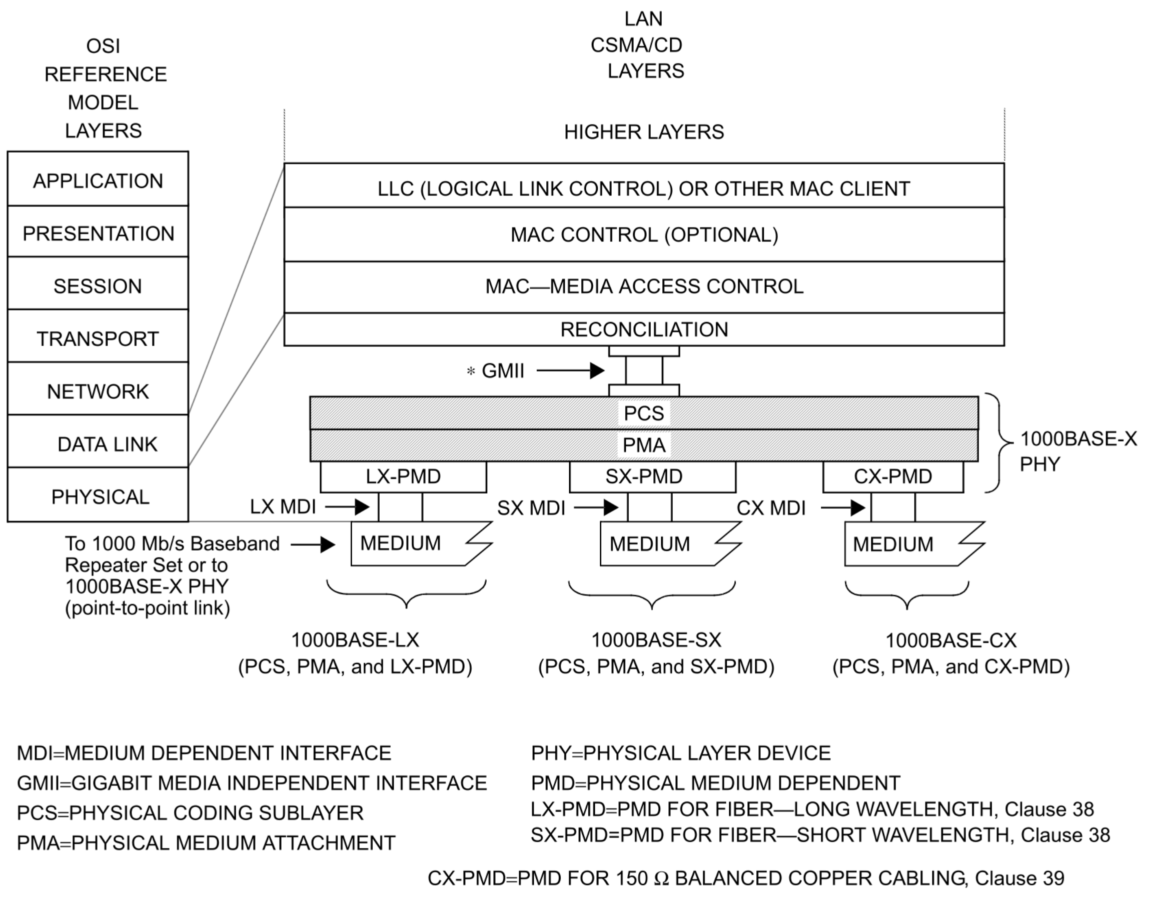
OSI REFERENCE MODEL LAYERS
APPLICATION
PRESENTATION
SESSION
TRANSPORT
NETWORK
DATA LINK
PHYSICAL
LAN CSMA/CD LAYERS
HIGHER LAYERS
LLC (LOGICAL LINK CONTROL) OR OTHER MAC CLIENT
MAC CONTROL (OPTIONAL)
MAC—MEDIA ACCESS CONTROL
RECONCILIATION
∗ GMII
PCS
PMA
1000BASE-X PHY
LX-PMD
SX-PMD
CX-PMD
LX MDI
SX MDI
CX MDI
MEDIUM
MEDIUM
MEDIUM
To 1000 Mb/s Baseband Repeater Set or to 1000BASE-X PHY (point-to-point link)
1000BASE-LX (PCS, PMA, and LX-PMD)
1000BASE-SX (PCS, PMA, and SX-PMD)
1000BASE-CX (PCS, PMA, and CX-PMD)
MDI=MEDIUM DEPENDENT INTERFACE
GMII=GIGABIT MEDIA INDEPENDENT INTERFACE
PCS=PHYSICAL CODING SUBLAYER
PMA=PHYSICAL MEDIUM ATTACHMENT
PHY=PHYSICAL LAYER DEVICE
PMD=PHYSICAL MEDIUM DEPENDENT
LX-PMD=PMD FOR FIBER—LONG WAVELENGTH, Clause 38
SX-PMD=PMD FOR FIBER—SHORT WAVELENGTH, Clause 38
CX-PMD=PMD FOR 150 Ω BALANCED COPPER CABLING, Clause 39

Figure 4.5: An excerpt from the IEEE 802.3-2008 Section 3. It shows the placement of MAC, PCS, PMA and PMD layers [19].

UDP/IP and ARP Modules

UDP/IP and ARP Modules implement a small UDP/IP stack for ethernet communication.

Actually, the communication protocol in Hyperimage hardware infrastructure does not need any layers higher than data-link layer in OSI model, because there is only one application that receives and sends data. However, as the SPU or CU can directly be connected to the Control-PC, which uses an ordinary UDP/IP Stack, a light UDP/IP stack is required, which implements Transport and Network layers.

In Transport layer, the UDP is used because of its much lower implementation effort and communication overhead compared to the TCP. It uses only one configurable port number, because there is only one application in the devices. UDP-CRC is also not used because of implementation effort.

In Internet layer, IPv4 is used. Like the configurable UDP-port, IP-address can also be remotely configured.

ARP (Address Resolution Protocol) module answers ARP-requests and tells the network partner about the MAC-address of the device. This feature is also not needed in a network topology with point-to-point connections and permanently connected network nodes, where every node knows its parents and children. But the Control-PC has a standard network interface, which follows the IEEE 802.3 Ethernet standard. Therefore ARP functionality is also required in the SPU and CU.

In summary, TCP/IP modules extract Hyperimage packets from an Ethernet frame and store them in a FIFO in receive direction. In transmit direction the data is read from the Ethernet Block input FIFO and is sent to the MAC module for further processing.

4.1.2.6 Other Infrastructure Modules

Gbit Ethernet Control

This modules sets the parameters for UDP/IP, ARP and Ethernet modules. These parameters include UDP port, IP address and MAC address of the CU Ethernet interface.

LCD Control

During a measurement it is useful to see some Coincidence statistics on a screen. This module consists of two submodules, *LCD* and *device specific driver module*.

The LCD module takes bytes with their type as input and converts them to ASCII characters which can be interpreted by an LCD driver and printed on an LCD. For example if the byte *0xAE* ($\cong$ 174) is given as an unsigned paramater, it is converted to ASCII characters *1*, *7*, *4*. The output is then forwarded to the underlying module.

The device specific driver module acts as a physical interface between the FPGA application and LCD. It generates the signals required for the communication with the LCD chip.

This module accepts its input as control packets from the Packet Decoder, but can also be connected to a standalone module to show module specific data like Coincidence statistics.

User Input/Output

This module is an user-interface module. It reads the state of the user inputs like buttons on the CU board and stores it in a register. It also implements also a writable register which can control user outputs on the board like LEDs.

This module receives its input from the Packet Decoder as control packets.

Coincidence Processing Control and Statistics

As the CU performs Coincidence processing it is helpful to store the actual state of the module and other statistics like detected Coincidences or filtered Singles. This data can then be sent to the Control-PC or can be shown on the LCD screen. Therefore it implements logic for statistics and a register, which stores status data regarding the Coincidence Processing Chain.

Additionally it sets parameters regarding the Coincidence Processing Chain.

This module receives its input from the Packet Decoder as control packets.

Coincidence Processing Chain

This module is an optional part of the thesis and could not be implemented due to project time constraints.

Error Handler Unit

This module sends a packet to the Control-PC in case of an error with the error reason. An example case is, if a packet with a wrong CRC is received.

Firmware Revision Unit

This module sends a packet with the revision number of the current design, if it is requested by Hyperion.

4.2 SPU Emulator

The SPU Emulator is an artificial SPU packet generator for CU testing. It is designed for the commissioning tests, when there were no SPUs available. It comprises of two main modules: The packet generator module and the Gigabit Ethernet module.

The packet generator generates constant-sized read-out packets with a variable interpacket timer. Only the sequence number is incremented in the header for debugging purposes, but the payload itself stays the same.

It sends the generated data on ten Ethernet ports. The interpacket timer is controlled via a Chipscope[1] Virtual Input module, which can be set via the Chipscope GUI.

[1]Chipscope is a JTAG debugging tool for Xilinx FPGAs [20].

The Gigabit Ethernet module is the same module used in the CU, which is described in section 4.1.2.5.

5 Implementation

The FPGA-development is done on the similar principles of software engineering. After the concept phase the developed software is divided in functional blocks, which are first coded and tested separately in a coding-verification cycle. Afterwards, they are connected together and the top design is developed in the similar manner.

Compared to high-level-programming, the FPGA-development has more stages in terms of generating and testing the program. These additional stages include mapping of the written code to logic elements, routing between the functional blocks in the FPGA, and downloading the firmware.

For better unterstanding of the FPGA-development, the tools and development flow are introduced in next sections.

Another important part of the implementation phase was the implementation strategy. Due to project time constraints, the design-reuse strategy was selected. The infrastructure modules from the SPU that could fit to the CU-design were used and altered where necessary. Again due to time reasons, some proposed modules could not be implemented at all. Section 5.3 lists the implemented modules in the CU-design and changes made during migration.

One important task was the clocking of the FPGA-internal-transceiver with the existing hardware, which was the most time consuming task during the implementation phase. Therefore the problem and its solution is explained in section 5.4.

5.1 Design Tools

5.1.1 HDL Designer

HDL Designer from *Mentor Graphics* is a graphical design environment for FPGA and ASIC development. Compared to text-based development environments, it provides additional design methods like graphical block editor for structural designs and graphical state machine creators for functional modules. These features ease the development and saves time especially on large designs.

The tool is a code generator and debugging front-end. In other words, it generates VHDL or Verilog code which is then synthesized by device-native tools and can invoke various design simulators to verify the design.

Version 2010.3 of HDL Designer was used for the development.

5.1.2 Xilinx ISE

Xilinx ISE (Integrated Software Environment) is a software pack for development of Xilinx logic devices. The environment provides text editor, code synthesis, simulation and debugging tools. During development it was used only for synthesis and debugging as HDL Designer was primarily used as front-end.

Major version 13 of Xilinx ISE was used for development.

5.1.3 Modelsim

Modelsim from *Mentor Graphics* is an FPGA and ASIC simulation and verification tool.

During FPGA development, the simulation of the design is very important, because of the design complexity of hardware logic design. Instead of synthetizing the design for debugging, a simulator is preferred. This way, the internal signals can be observed and time is saved compared to hardware debugging due to long synthesis time of FPGA designs.

Modelsim SE 10 was used during development.

5.2 Design Flow

The design entry is done by a text editor of the user's choice, or by HDL Designer's graphical design tools, Block Diagram Editor and State Machine Editor.

All functional blocks in VHDL can be modeled with a black box whose output is dependent on the input and/or the former inputs. This behavior is typically implemented with sequential state machines. Therefore state machine design is a commonly used design method for functional blocks in FPGAs.

HDL Designer provides a graphical State Machine Editor to assist the user during implementation. The user creates the states, transitions, transition conditions and internal variables on a GUI which are then converted to VHDL-code.

HDL Designer Block Diagram Editor is a graphical editor for structural VHDL modules. The user can instantiate implemented modules and connect them to each other with signals. The structural diagram is then converted to VHDL-code.

When used properly, the desribed graphical design entry methods in HDL Designer provide better results than the text-editor-based design entry in terms of design time and readability.

A crucial part of the design flow is verification. Verification is a process used to demonstrate that the intent of a design is preserved in its implementation [21, 1]. It can be done at various stage of the development, for example at the concept phase as plausability check or during the implementation phase in form of *testbench*es[1].

Compared to code writing, verification can be more time-consuming task. It can consume 70% of the overall design effort [21, 2]. Because of these reasons and the limited time of this project, the design was verified with limited amount of input stimuli on functional level. An example is shown in chapter 6, where a top level design is verified visually with a waveform viewer in terms of processing an input data packet. A complete verification involves much more implementation and testing effort. However, this project involves developing a proof-of-concept system, therefore a complete verification is not part of this project.

A part of the verification is simulation. One reason is the long duration of synthetizing, mapping, placing and routing of the design. Table 5.2 shows an example of implementation tool runtimes during the compilation of the CU design with only three Ethernet interfaces. It is clear that a direct in-circuit verification of an FPGA design is not feasible after every small design change. Therefore a comprehensive simulation is needed which covers most of the possible input stimuli to ensure the design to work as intended, before testing the design on the FPGA.

Tool	Runtime
Synthesis (xst)	55 s
Netlist Merge (ngdbuild)	23 s
Mapper (map)	186 s
Place & Route (par)	104 s
Static Timing Check (trce)	33 s
Bitfile Generation (bitgen)	76 s
Total Runtime	7 min 17 s

Table 5.2: FPGA implementation tool runtimes during compilation of the CU design with only three Ethernet blocks on a quad-core processor with multithreading enabled. The names in brackets are program-tool-names from the Xilinx ISE.

The second reason is the convenience in terms of generating the input stimuli signals. A comprehensive simulation makes it possible to cover most of the possible input stimuli to ensure the design to work as intended, when the it is commissioned.

Another reason is the debug capabilities of a simulator. It gives the design engineer the possibility to watch every single signal in the design and to stop the simulation at different conditions to check the state of the signal states as needed.

[1]Simulation code used to create a predetermined input sequence to a design, then optionally to observe the response [21, 1].

Unfortunately, there are also downsides of simulating large FPGA designs as a whole. A top-level simulation can take very long time, as the simulation is approximately one million times slower than in-circuit execution [22]. Therefore, it makes sense to first ensure the intended function of smaller blocks before simulating the top-level design.

The simulation of the design is performed with Modelsim. First, a testbench with input stimuli is prepared, which is also a VHDL module itself. Then the VHDL modules are compiled and the output signals are checked, if they are behaving as intended. The check is either done by the testbench or visually on the simulator GUI itself. An example visual verification is shown in chapter 6.

If the design is simulated only in terms of logic, then it is called a *functional simulation*. This type of simulation sees the design as a big boolean function with ideal logic gate delays. This allows the user to check the functional correctness of the design.

Large designs that incorporate many functional blocks or many levels of combinatorial logic are also likely to cause timing errors, especially in sequential designs. The reason is the increasing propagation delay with every logic gate connected in sequence. A long delay in turn makes it difficult to meet the setup time of the next combinatorial level, which in turn leads to undeterministic behavior or temperature-dependent errors in the design, although the functional simulation was successful.

There are also type of errors, which happen due to the complexity of synthesis tools. Synthesis tools are large software systems, which rely on the algorithms and libraries they are based on [21, 9]. These can also produce wrong synthesis results, based on the input. Such an error was also experienced during the implementation phase of this project. A generated and downloaded design was showing another behavior than in the functional simulation in specific cases. Switching from the Xilinx ISE version 12.4 to 13.1 had solved the problem. Due to time constraints no investigations on the cause were made. Probably the code was synthetized to wrong logic gates at some points, which behaved differently than in the code described process.

This type of errors can be simulated after the routing phase of the FPGA design, when the interconnect between logic blocks is implemented. At this stage block and net delays can be approximately calculated for best and worst cases and considered in the simulation. This type of simulation is called *Timing Simulation* and is the closest verification type to the in-circuit verification.

After the design is routed, a bitstream is generated for the FPGA. This bitstream is downloaded to the FPGA and it configures the logic blocks and the interconnect.

The fact that the design was verified with functional and timing simulations doesn't mean that the design will also work on the FPGA. There are also error-sources that cannot be simulated during the functional or timing simulations. For these type of errors, Chipscope [20] can be used. It is a synthetizable logic analyzer with online configurable triggers which store the captured data directly in the memory blocks on the fabric. This solution gives the debugging user more flexibility compared to an external logic analyzer in terms of the amount and depth of probed signals. Despite the advantages of this tool, attention must be paid to over-instantiation of the Chipscope cores. Because the Chipscope itself is also a logic block, it affects the timing of the whole design and can result to errors that do not happen on the design without Chipscope cores.

Figure 5.1 shows a simplified diagram of the FPGA implementation and verification flow and tools which were used during this project.

5.3 Design Reuse and Changes

As the SPU has similar functions like decoding, encoding and routing the packets, the basic infrastructure can also be used in the CU design. Due to the limited implementation time for this project, design reuse was the preferred implementation strategy. Therefore, modules from the SPU design were ported to the CU design whenever it was applicable and others were implemented from scratch. This chapter describes which modules were ported and gives an overview of changes.

Modules in CU fall into three main categories in terms of implementation strategy.

- Ported modules
- Copied modules
- Modules which have been written from scratch

Ported modules include modules which are available in the SPU design, but need to be altered to fit the CU's requirements as well as modules which need to be partially implemented from scratch because of hardware-dependent submodules. These submodules include wrappers for GTX transceivers or Embedded-MAC modules, whose logic is implemented on the fabric and therefore different for every FPGA-line.

Modules which are copied from the SPU design without any design change are Copied modules.

The rest of the modules are CU-exclusive modules which need to be written from scratch.

Figure 5.2 gives an overview of the CU implementation and the types of modules in terms of how they were implemented. In next subsections the design changes and reasons are described in more detail.

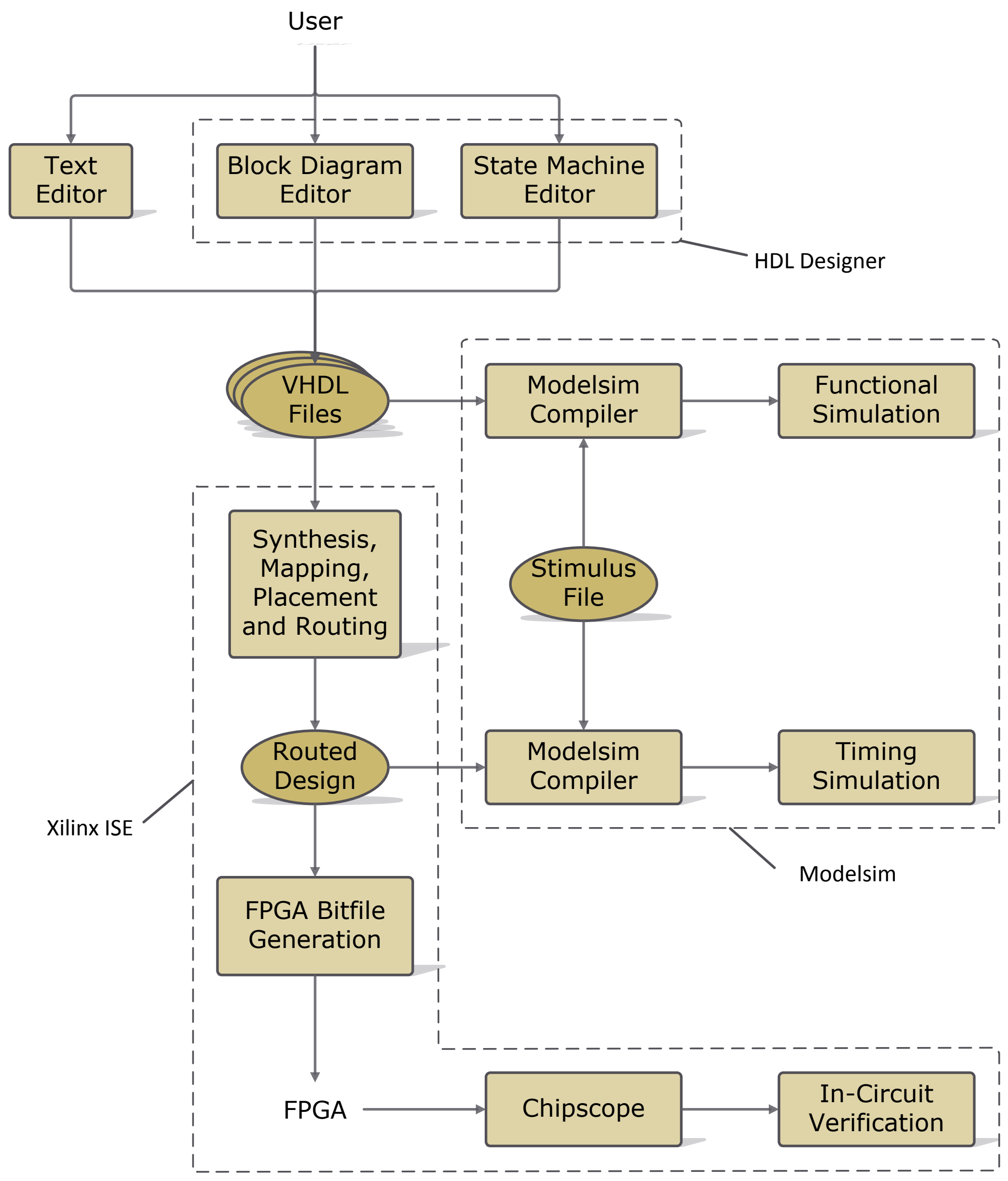

Figure 5.1: Simplified FPGA implementation and verification flow

5.3.1 Ported Modules

This subsection decribes the modules which were ported from the SPU and gives an overview of the changes done.

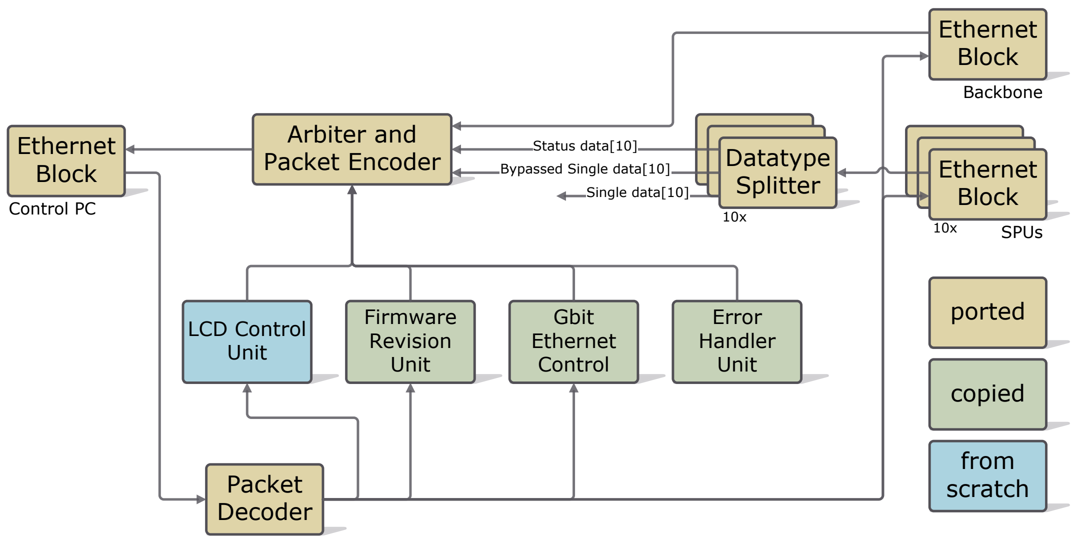

Figure 5.2: CU architecture implementation

5.3.1.1 Ethernet Block

As described in section 4.1.2.5, the Ethernet Block consists of two submodules, *UDP/IP and ARP modules* and *MAC and Physical Layer*. The latter consists again of MAC and GTX transceiver modules. Figure 5.3 shows the Ethernet block in detail.

UDP/IP and ARP modules had to be fixed only at the receive state machine. The state machine was expecting exactly seven bytes of ethernet preamble, but it turned out that the Xilinx MAC module is sending out six or seven preamble bytes randomly. Currently, UDP/IP and ARP modules accept both preamble lengths.

MAC and Physical Layer were designed from scratch. The main reason is that MAC and GTX transceivers are device-dependent, in other words, they had to be regenerated with Xilinx Tools for the current FPGA. Secondly, the CU incorporates twelve Ethernet blocks compared to one in the SPU. Therefore the Ethernet Block had to be altered that it supports the use of Soft MACs additional to the Embedded MACs[2].

5.3.1.2 Packet Decoder

The difference between the Packet Decoder in the CU and the SPU is the handling of the header. The module in the SPU accepts packets without the SPU field in the header, therefore the state machine in the CU must be changed accordingly.

[2]Virtex-6 FPGAs include four MAC blocks on the fabric which are called *Embedded MAC*s. Additionally, there is also a MAC core from Xilinx which is synthesizable on FPGAs and is called *Soft MAC* in this project.

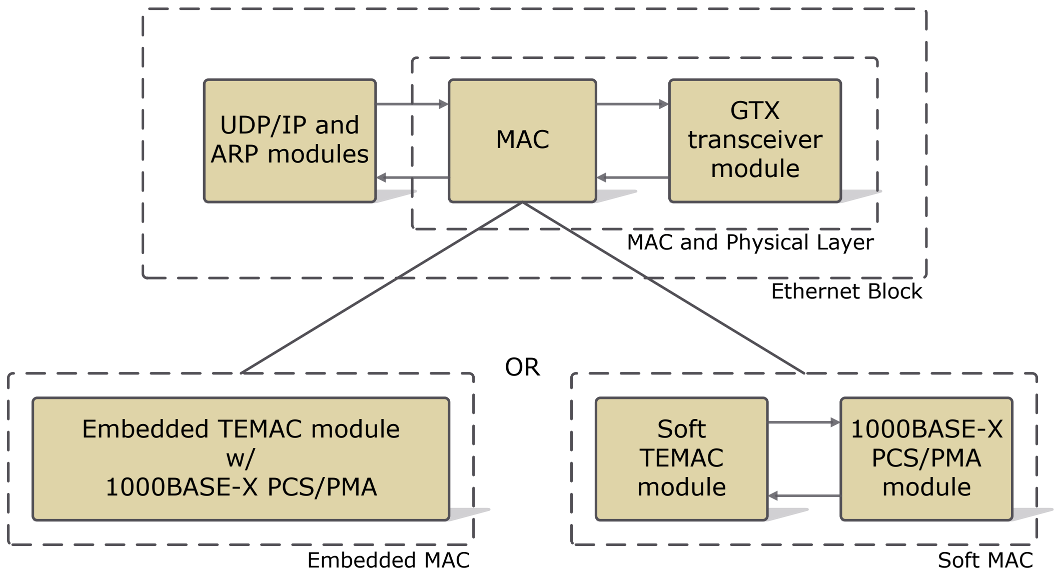

Figure 5.3: Ethernet Block detail

The Packet Decoder was ready to use with a CU, but was not tested with CU settings. Some small fixes in the state machine were conducted.

5.3.1.3 Arbiter and Packet Encoder

In the SPU design the Arbiter and Packet Encoder were in a single block compared to the independent modules conceived for the CU. The block was also ready to use with a CU, but some bugs were found in the Packet Encoder state machine and parts of it were rewritten.

The priorisation concept described in 4.1.2.3 is currently not implemented in the Arbiter due to time constraints.

5.3.1.4 Datatype Splitter

During testing of this module, it was discovered that the module's state machine was not able to process fragmented data packets. This behavior is reasonable, because for the communication inside the SPU, the packets are sent as a whole and there are no fragmenting modules except the TCP/IP module in the Ethernet Block. Therefore, there is no need to deal with fragmented packets for the Datatype Splitter in the SPU.

The UDP/IP module sends the MAC packet either after the Ethernet transmit timer times out or when the maximum payload size of 1500 bytes are reached. The latter can cause fragmented packets.

The Datatype Splitter was designed to reject the first part of a packet, if the rest does not come during the pre-set wait time. Therefore many packets get lost in downstream direction.

Parts of this module were reimplemented to support fragmented packets.

5.3.2 Copied Modules

Firmware Revision Unit, Gbit Ethernet Control and Error Handler Unit were copied to the CU design and no changes or fixes had to be done. All these mentioned modules but the Gbit Ethernet Control fit to the concept explained in section 4.1.2.6.

5.3.2.1 Gbit Ethernet Control

The Gbit Ethernet Control distinguishes itself from the module described in the concept (see section 4.1.2.6) at some points. The module not only sets the UDP port, IP address and MAC address of the CU Ethernet interfaces but also the addresses of the link partner on the other side of the line (destination).

At the time of writing, the destination UDP port, IP address and MAC address must also be configured, because the UDP/IP modules are not programmed in a generic manner but with only one link partner on one interface in mind. Thus, an extraction of the destination address out of a received packet is not possible. Every UDP packet is sent with this configured data.

This module only sets the parameters of the Control-PC interface, because the Control-PC running on a generic Ethernet interface card and TCP/Stack must receive packets with correct destination address. Otherwise they would not get forwarded to the Hyperion.

In contrast to the Control-PC, the CU Ethernet interface accepts every Ethernet packet regardless of the wrong destination MAC or IP address, which is also the reason why the network parameters of the interfaces for the SPUs do not need to be set[3]. Only the destination UDP port must be correct, otherwise the packet is rejected.

5.3.3 Modules written from scratch

LCD Control is a CU-exclusive module, therefore it was written from scratch.

[3]As the UDP/IP modules are migrated from the SPUs, the SPUs also process the Ethernet packets in a similar fashion

5.3.4 Unimplemented Modules

The User Input/Output, Coincidence Processing Chain and Coincidence Processing Control and Statistics could not be implemented due to time constraints.

5.4 Transceiver Clocking

During the implementation phase, many design challenges were confronted regarding FPGA design. Most time-consuming problem was the simultaneous operation of twelve GTX transceivers with one clock source. Therefore, this problem and its solution are explained in the next subsection.

5.4.1 Transceiver Clocking Constraints

The CU hardware incorporates twelve optical fibre transceivers, which operate at 1250 MHz. To achieve such frequencies for data exchange, dedicated multi-gigabit transceivers are used on Xilinx FPGAs. These transceivers are called *GTX transceiver*s [23].

These blocks can achieve line rates of 6.6 Gb/s and need clock sources with low jitter. Therefore, they have their own reference clock inputs compared to the global clock inputs of the FPGA logic. The exclusive reference clock inputs for the GTX ensure that the clock signal is directly routed to the GTX blocks, which reduces the jitter creation, which becomes a critical aspect especially at higher frequencies.

If the GTX reference clock inputs are used, there is a design constraint which limits the set of usable GTX transceivers to a particular area.

The GTX transceivers are grouped as quads, which share the same clocking resources. For example, if an oscillator is connected to the external clocking input of a GTX quad, then it can supply all of the four transceivers. Additionally there are also clocking lines to the neighboring quads, which enable the quads to use the clocking resources of their neighboring quads. In summary, three neighboring quads can be supplied with one oscillator, which is connected to the middle quad. Figure 5.4 shows the clocking relation between quads more in detail.

*MGTREFCLK*s are dedicated external oscillator inputs for clocking the GTX transceiver fabric. Because of their direct connection to the transceiver, these are the recommended clock inputs by Xilinx. This clock line is routed to the upper or lower quad by the router, if one transceiver is used in the upper or lower quad respectively[4].

[4]The router does not route transceiver clock lines automatically, if two GTX clock inputs are used in the FPGA design. In this case the clocking input of every transceiver must be connected manually.

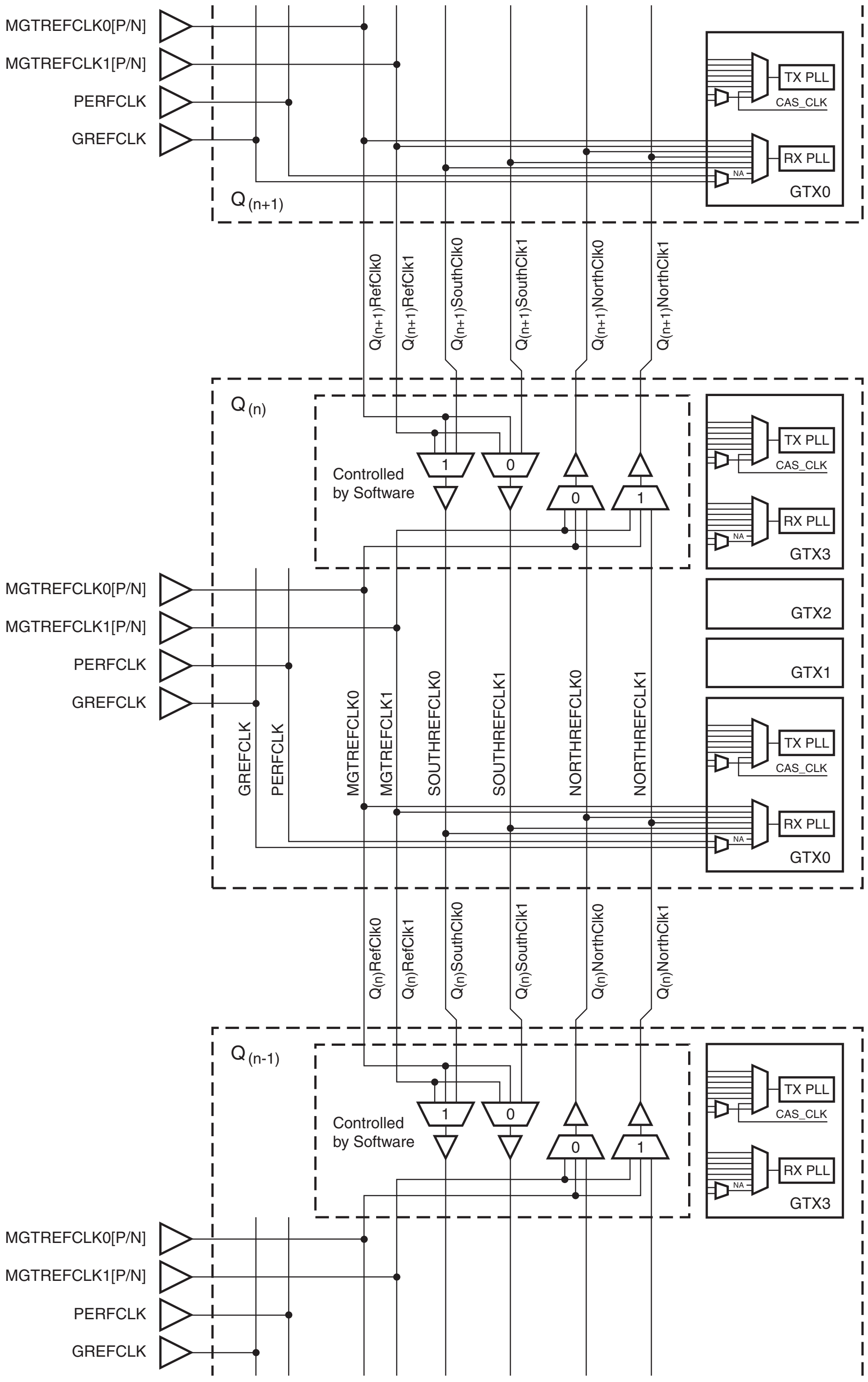

Figure 5.4: This excerpt from the Xilinx User Guide 366 shows the GTX transceiver reference clocking. $Q_{(n)}$ is a single quad. The *MGTREFCLK0* and *MGTREFCLK1* inputs are external oscillator inputs. They can be routed to the north ($Q_{(n+1)}$) or south ($Q_{(n-1)}$) quad. *PERFCLK* and *GREFCLK* are reference clocks that can be sourced from the FPGA logic.

Alternatively, the GREF input can be used, which can be connected to the normal FPGA logic clocking resources. In other words, the GREF input of every GTX transceiver can be routed to the global clocking lines of the fabric. As these global clocking lines have enough fan-out, there are no constraints in terms of clocking area as in case of clocking the quads, if this input is used.

But there is a disadvantage of this clocking source. Because this clock has more jitter and delay than a dedicated GTX clock due to internal routing, sourcing the transceiver clock from the FPGA logic is not recommended by Xilinx.

5.4.2 ML605 and Transceiver Daughter Board Clocking

The daughter boards were designed with the assumption that the 125 MHz oscillator on *REFCLK0* pin from the *BANK_116* (orange labeled in figure 5.5) could clock all the used GTX transceivers (yellow labeled). Unfortunately, this design mistake was not discovered before the implementation phase of the CU.

The daughter board designs could have been altered to incorporate one external oscillator on one board, which is connected to the *REFCLK0* pin on the *BANK_113*. In this case the twelve transceivers on the quads *BANK_112*, *BANK_113* and *BANK_114* could have been used with dedicated clocking.

Due to long time loss that would be caused by the redesign of the daughter board PCB, a solution based on the FPGA configuration was investigated.

After manually editing the Coregen[5] generated GTX transceiver wrapper files, it was found that disconnecting the *BUFR*[6] component out of the GTX transceivers' wrapper module and using only the *MGTREFCLK* pin on all transceivers was accepted by the FPGA router. The resulting design could successfully establish an Ethernet link and was operable.

After having analyzed the routed design with the FPGA Editor[7], it was discovered, that the external clock signal was routed to a global clock buffer (BUFG) and the output of the buffer was connected to the *GREF* clocking inputs of the transceivers.

The use of *GREF* as clocking source for GTX is not recommended by Xilinx, because FPGA logic clocking resources can introduce jitter for operation at high data rate [23, 106]. Nevertheless this solution was accepted, because of acceptable operability of the system and time constraints of this project.

In the future, for the reliability of the GTX transceivers at high data rates, the external oscillator solution should be used.

[5]A tool for generating Xilinx-FPGA-specific IPs

[6]Regional clock buffer. It is a clocking driver component which drives only a region of the FPGA compared to a *BUFG*, which can drive global clock lines on a Xilinx FPGA.

[7]A graphical application by Xilinx for displaying and manually configuring FPGAs [24]

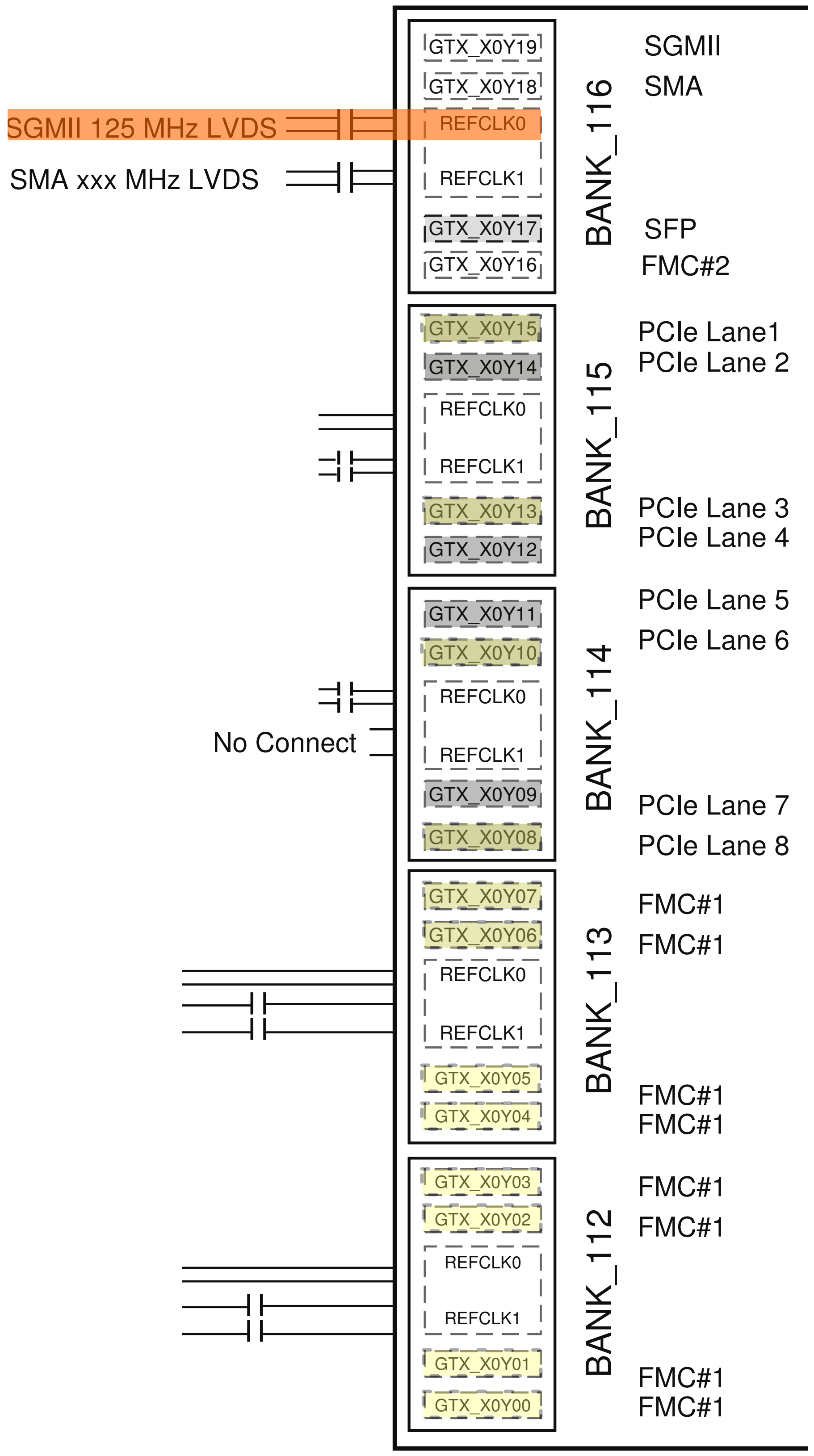

Figure 5.5: An excerpt from the Xilinx UG534 ML605 User-Guide. Only the GTX blocks, which are marked with *FMC#1* or *PCIe Lane* can be used in a custom design, e.g. a daughter board. The orange marking shows the 125 MHz Ethernet oscillator. The yellow markings show the GTX transceivers, which were connected to the optical fibre transceivers on the daughter boards.

6 A Verification Example

The importance of an FPGA design verification was discussed in section 5.2. In this chapter the verification of the design in upstream direction is demonstrated as an example to show how the parts of the design were verified throughout the implementation phase.

Verification in upstream direction means that the packets are injected from the Control PC side and that they get processed by the Command Decoder (the main upstream processing module). The Command Decoder can route the packets either in SPU direction or to an internal infrastructure module. In this example, only the routing to an SPU is shown.

6.1 Upstream Verification Example

To verify the design in upstream direction two tester-modules were used in the verification testbench: The *Stimulus Parser* and the *Ethernet BASE-X Modulator*.

The Stimulus Parser reads a stimulus file, which contains the Ethernet packets that should be modulated by the Ethernet modulator module. The output of the parser is read by the Ethernet modulator.

The Ethernet modulator module wraps the Ethernet packets with preamble and CRC and generates the differential signals for 1 Gbit BASE-X communication. The output of the modulator is connected to the Control-PC port of the CU on the testbench.

The simulated CU top design consists of one Control-PC port and only two SPU ports to speed up the simulation. It is clocked by a 125 MHz clock and the SPU ports are connected in a loopback scheme.

Figure 6.1 shows the schematic of the upstream verification testbench.

Part of the wave output after the simulation of the design in Modelsim is shown in figure 6.2. The verification will be explained with the help of this figure.

Due to excessive length of the waveform, unimportant parts of the waveform are cut away and seen as whitespaces in figure 6.2.

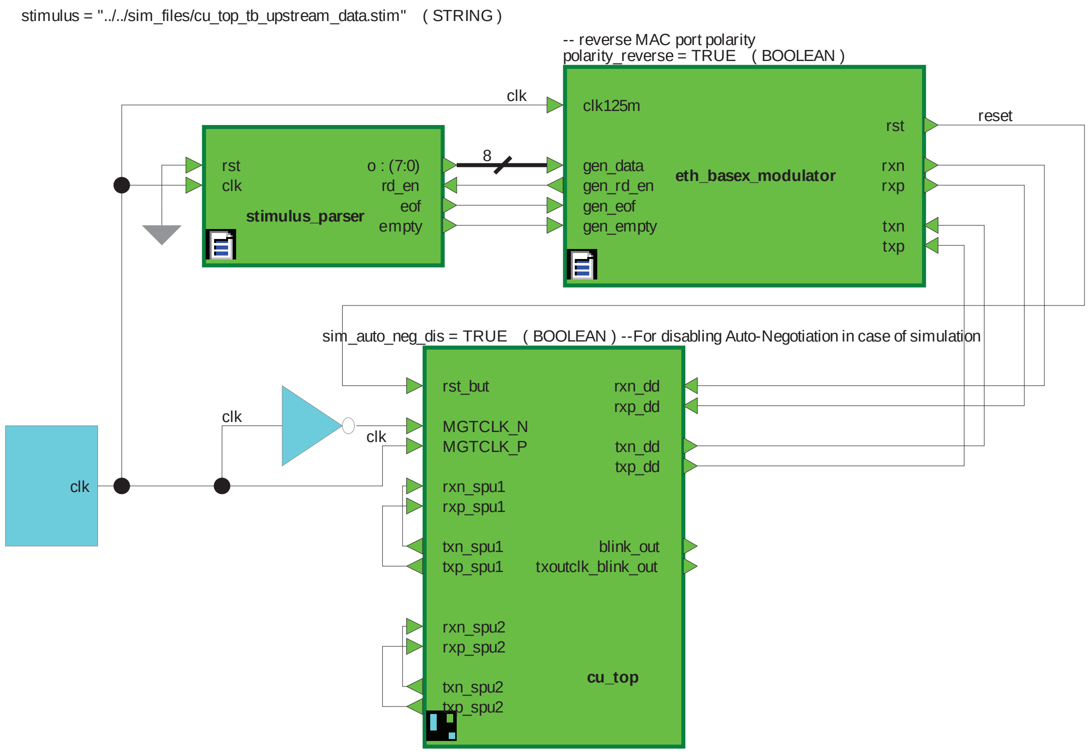

Figure 6.1: Upstream verification testbench schematic. The Control PC side Ethernet port names are suffixed with *_dd*, which is an abbreviation for *Data Dumper*. *cu_top* is the top VHDL entity of the CU. *blink_out* and *txoutclk_blink_out* signals are debug signals and are not used in this testbench.

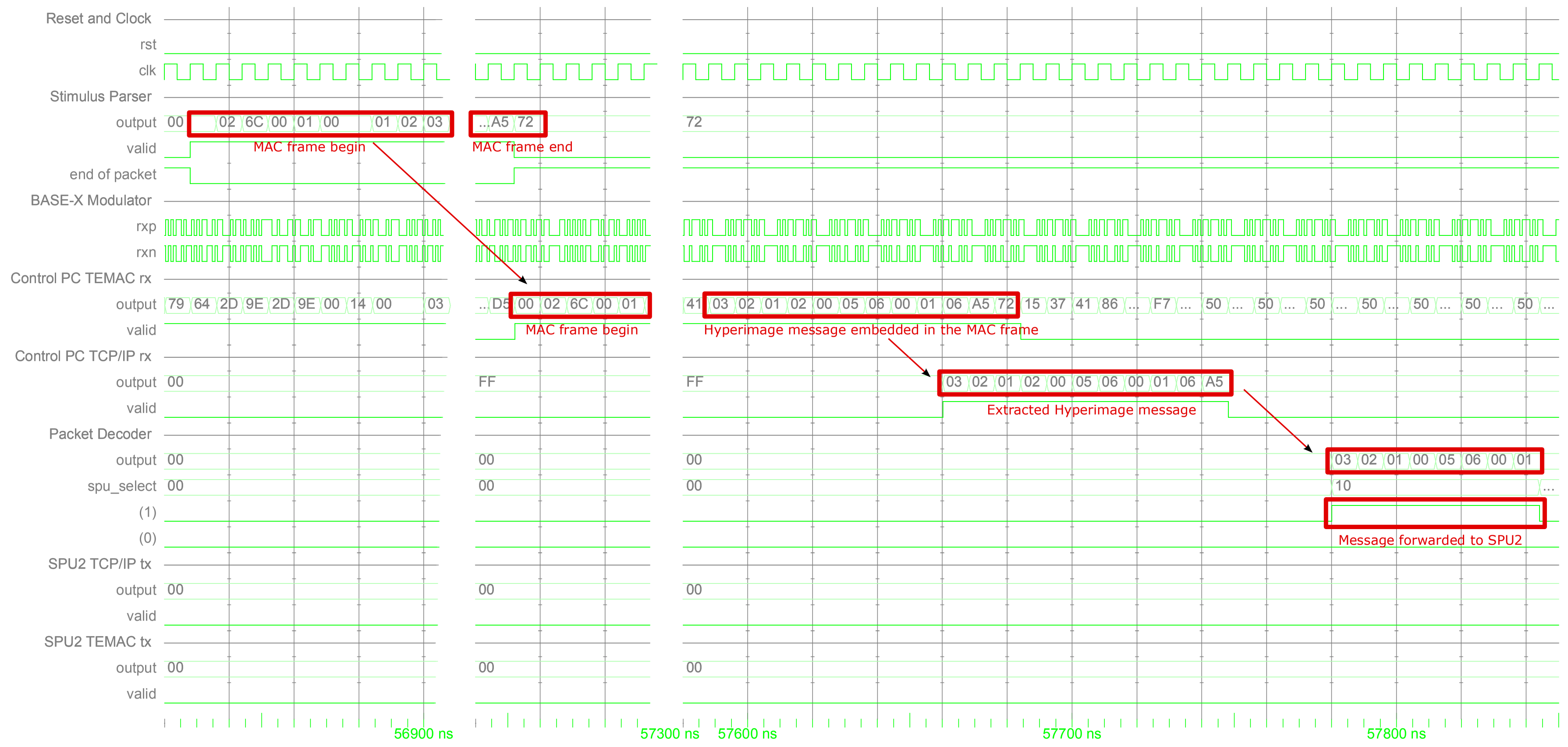

Figure 6.2: Upstream verification testbench waveform. Due to the excessive length of the byte sequences, only the processing until the output of the Packet Decoder is shown.

For better differentiating the text from the signal names in the diagram, the signal names are written in italic.

At the top of the waveform there are global reset (*rst*) and clock (*clk*) signals for the whole design.

The modules are arranged in the wavelist according to the processing order of the data packets. First, the stimulus is created by the *Stimulus Parser* and modulated by the *BASE-X Modulator* on the Control-PC-side receive differential lines (*rxp*, *rxn*). The signal is then demodulated by the TEMAC block (*Control-PC TEMAC rx*) and the Ethernet packet is disassembled by the UDP/IP module (*Control-PC UDP/IP rx*). Afterwards, the *Packet Decoder* routes the resulting Hyperimage packet to the SPU2 in this particular case, which is processed by UDP/IP and TEMAC modules (*SPU2 UDP/IP tx and SPU2 TEMAC tx*) and finally sent to the SPU2.

The signals at the bottom of each title are the respective output (*output*) signals and output-valid (*valid*) signals of each module. Data of an output module is read by the destination module only if the valid signal is asserted. The Packet Decoder has two valid signals (signal group *spu_select*), which are connected to the two Ethernet Blocks for two SPUs.

The stimulated data is a command packet destined for the module 6 in SPU2 with one byte payload (*0x03, 0x02, 0x01, 0x02, 0x00, 0x05, 0x06, 0x00, 0x01, 0x06, 0xA5, 0x72*), which is packed in an ethernet frame and a UDP/IP packet.

The injection of the packet begins at approx. 56835 ns. The modulated data is not easily readable by looking at the differential signals. It is added to the wave list only to check the presence of the modulated signal. The beginning of the demodulated ethernet frame can be seen at 57250 ns. The embedded Hyperimage packet is correctly extracted by the UDP/IP module (from 57660 ns to 57750 ns[1]) and subsequently decoded by the Packet Decoder by stripping the SPU field out (from 57780 ns to 57890 ns). Also, the SPU2-valid signal is asserted, which also acts as a write-enable signal for the Ethernet Block connected to the SPU2 port.

At 57890 ns the Hyperimage packet is successfully written to the FIFO of the UDP/IP block. Because of the overlength of the simulation waveform, the processing is shown only until this point. After this point, *SPU2 TCP/IP tx* packs the data in the FIFO in an UDP/IP packet and prepends MAC header fields. Subsequently, the Ethernet Frame is wrapped with an Ethernet preamble and CRC by *SPU2 TEMAC tx* and finally modulated on the SPU2 glass fiber transceiver lines.

[1]The last byte (in this case also the only payload byte) of the packet (*0x72*) is out of the scope of this diagram and comes approximately 15 cycles later. This is due to the delay caused by the CRC calculations done in the Packet Decoder.

7 Commissioning and Results

There were two important milestones during this project:

- Successful communication of the CU with the Control-PC software Hyperion
- Successful communication of the CU with the SPUs in transmit and receive direction

A *successful communication* in the first milestone means that both entities, CU and Control-PC, are able to send messages to each other and to decode these as valid command or status packets.

Similar to the first milestone, the second one involves ensuring a communication between the CU and SPUs by exchanging command and status packets. But, a more important goal is the successful routing of the read-out data coming from the SPUs, which additionally involves commissioning tests with read-out data at different bandwidths.

To assess the extent to which these goals have been achieved, various commissioning tests were conducted. This chapter describes these commissioning tests and their respective results.

7.1 Communication Test between the CU and the Control-PC

The aim of this commissioning test is to show that the CU can receive packets from the Control-PC and respond with a valid packet. This test can be easily accomplished by sending a command packet to a module, that can respond, i.e. Firmware Revision module.

For the test setup, the CU was directly connected to the Control-PC with a fiber-optical cable.

For the commissioning, Hyperion is started and a configuration file is loaded, which describes the current setup (in this case merely a CU). Then, a command packet for Gbit Ethernet Control is sent to the CU to configure the UDP/IP modules with the MAC, IP address and port number of the Control-PC. Then a request to the Firmware Revision Module is sent, which responds the command with a valid status packet. The firmware revision is then shown in Hyperion.

This test was successful.

7.2 Communication Test with the SPU Emulator

As described in section 4.2, the SPU Emulator was designed for the phase, in case there were no SPUs available for testing.

For the test setup, the CU and SPU Emulator designs were generated with only two SPU ports. The SPU ports were connected to each other and the CU was connected to a workstation PC, which is responsible for receiving and storing the data coming from the CU. Figure 7.1 shows the commissioning test setup.

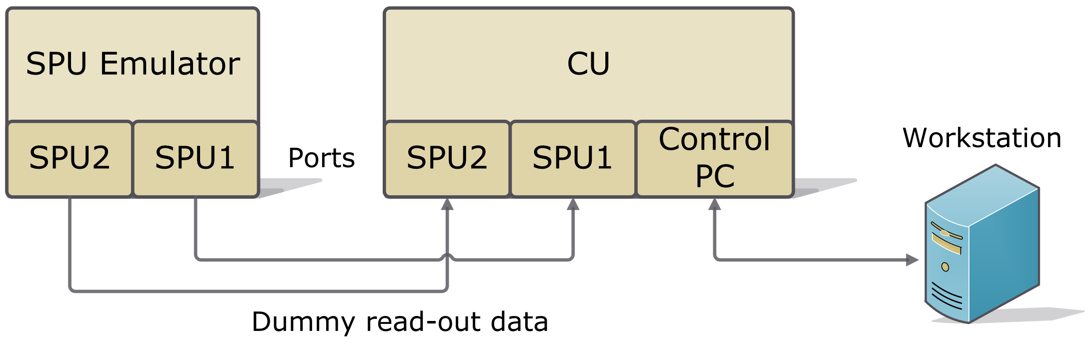

Figure 7.1: Setup for the communication test with the SPU Emulator

Hyperion was not used in this test, because the SPU Emulator is controlled via Chipscope and the CU does not require any commands to forward the data coming from the SPUs in current configuration.

In order to store the data, a Perl script is used, which listens to UDP packets coming from the CU and prints the bytes in a text file by separating every Hyperimage packet with a linefeed character. Regarding data-analysis, the program *grep* was used, which simply compares every line of the dumped data with a pattern, in this case the Hyperimage packet generated by the SPU Emulator. When there are lines which do not fit to the pattern, they are outputted by the program.

For the commissioning, first the interpacket delay in the SPU Emulator is set via Chipscope, which determines the amount of wait cycles between generated Hyperimage packets. Afterwards, the UDP-data-dumper script is started on the dumper-workstation. Subsequently, the CU and the SPU Emulator are reset in sequence.

The incoming datastream from the CU was stored for one minute. Afterwards, the UDP-data-dumper-script was stopped and the analyzer-script was run on the stored data.

The test procedure was to decrement the interpacket delay by a specific amount after every test, until a corrupted line was detected by grep. The test results showed that the CU could process the data up to an input bandwidth of 2.3 MBytes/s without any data corruption, which is far away from the target input data bandwidth of 700 Mbit/s.

This test was conducted to determine the data bandwidth, at which a data corruption occurs, when the SPUs were not available for testing. This data bandwidth could be simulated in a testbench to determine the bottlenecks of the design.

It helped to find and correct some bugs in the design, but due to time constraints not all of the data corruption sources could be determined.

For thorough testing, the SPUs were used, because SPU Emulator does not model the read-out packet behavior. It only sends constant payloads with changing sequence numbers.

7.3 Communication Test with two SPUs

The aim of this commissioning test is to prove that the CU is able to communicate with the SPUs and to determine the packet loss at different bandwidths of input data. The test was done at two main stages:

- Commissioning test with the SPUs in terms of control and status data routing
- Performance test with the SPUs in terms of processing (header processing and routing) the read-out data at different bandwidths

7.3.1 Control/Status Data Routing Test

This test is done with the same goal as the test between the CU and the Control-PC described in section 7.1.

The test setup is also similar to the mentioned test, but this time additionally with one SPU connected to the CU.

For the commissioning, the same procedure in section 7.1 is followed to establish a communication to the CU. Then a request to the Firmware Revision Module in the SPU is sent, which responds with the firmware version of the SPU.

This test was successful.

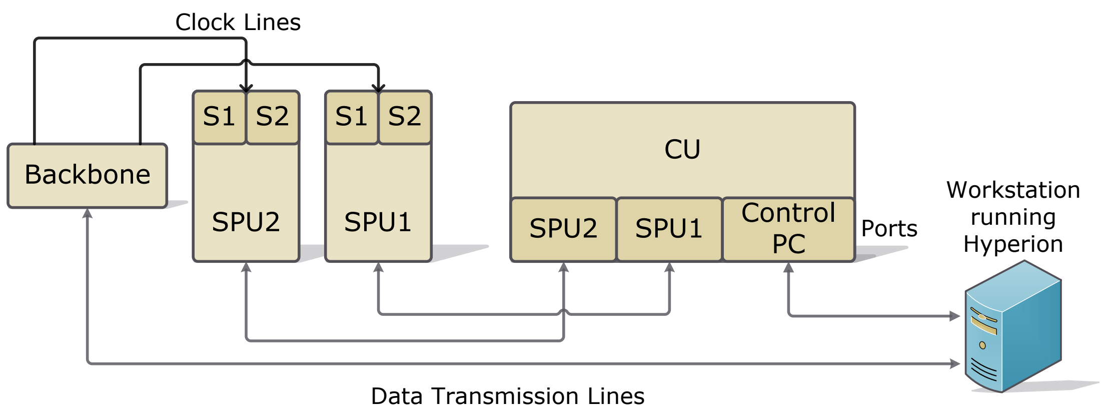

Figure 7.2: Setup for the communication test with two SPUs. *S1* and *S2* are the Sensor Stacks, which generate the read-out data.

7.3.2 Read-out Data Processing Test

This test involves the evaluation of the performance of the CU in terms of header processing and routing. The performance is determined by calculating the percentage of lost or invalid Hit-data in measurement data at different data input bandwidths.

The input bandwidth is calculated at the input of the Datatype Splitter of the CU, because this is the first processing point after the extraction of the Hyperimage packets from the Ethernet frames.

For the test setup, two SPUs with two Sensor Stacks each are connected to the CU. The CU is connected to the Conrol-PC, which is running Hyperion. The Sensor Stacks need a reference clock for operation, therefore the Backbone is connected to the SPUs. The Backbone is also connected to the Control-PC, because at the time of commissioning test the CU design did not include the Ethernet interface for the Backbone. Figure 7.2 shows the test setup diagram. At the end of this chapter there is also a photo of the test setup on figure 7.6.

The test was carried out only with two SPUs, because there were only two available at the time of this test.

For the test, Sensor Stacks with only Interface FPGA board were used, which were configured as artificial sensor data generators. There are five different configurations, which generate readout packets with 1, 2, 4, 20, 40 or 80 Hits per time slot. There is one FPGA-firmware for each configuration.

One time slot is approximately 52,42 μs. At the end of every time slot, the Interface FPGA generates a *Frame* and increments the *Frame Counter*. A Frame contains the Hits that are generated by different measurement channels of an ASIC during that time slot.

This Frame contains a 2 byte header, a 4 byte Frame Counter value (*Frame*) and the Hits in 12 byte chunks. These chunks contain detailed information about the Hit, which are irrelevant for the scope of this measurement. The Frame structure is shown in figure 7.3. The total length of a Frame is $12 \cdot n_{Hits} + 6$ bytes, where n_{Hits} is the number of Hits generated by one Stack in a time slot.

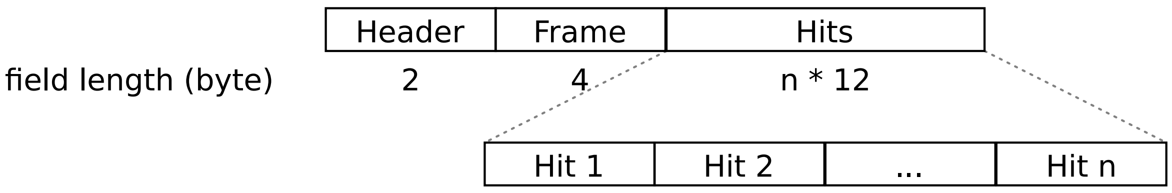

Figure 7.3: Frame structure generated by the Interface FPGAs

The Interface FPGA wraps the generated Frame in a Hyperimage packet and forwards the Frame in downstream direction. The length of a Hyperimage packet at the input of the Packet Encoder is:

$$\begin{aligned} length_{total} &= length_{header} + length_{payload} + length_{trailer} \\ &= 10 + (12 \cdot n_{Hits} + 6) + 3 = \\ &= 12 \cdot n_{Hits} + 19 \end{aligned} \tag{7.1}$$

With the Frame generation period t_{Frame}, number of Stacks n_{Stacks} and the number of Hits per frame n_{Hits}, the average bandwidth at the input of the CU can be calculated:

$$\overline{bandwidth} = \frac{n_{Stacks} \cdot length_{total}}{t_{Frame}} \tag{7.2}$$

For the commissioning, Hyperion is loaded with the setup configuration including the Backbone, the two SPUs and the four Sensor Stacks. After the connection to the CU is established (see section 7.1), the firmwares for the Interface FPGAs are sent to the Sensor Stacks via Hyperion. Then, the Backbone is configured to supply the reference clock for the Sensor Stacks.

Hyperion must be run in data-dumper mode, because online processing allocates processor time, which can cause packet loss during the measurement. In data-dumper mode, Hyperion simply stores the incoming read-out datastream in a database, which can be processed after the measurement.

After this configuration, the *Inhibit* signal for Sensor Stacks is deasserted via Hyperion, which causes the generation of read-out data by the Stacks with number of Hits according to the loaded firmware.

After Hyperion has received at least one million Hits, the measurement is stopped by asserting the Inhibit signal for the Sensor Stacks.

After the measurement, Hyperion can load the database file and parse the dumped Hyperimage packets. During the parsing of the data, also the integrity of the Hyperimage packets is verified in terms of validity of the field-values. (e.g. preamble, trailer and CRC). At the end, Hyperion shows the count of preamble, trailer and CRC errors. However, this data is not enough to determine the percentage of the lost packets, because Hyperion does not output the total number of processed Hyperimage packets at the time of writing. Therefore, a data-analyzer program for the assessment of the measurement data was implemented.

The parsed measurement-data-dump is used as data input for the analyzer. This is a comma-separated values (CSV) file, which contains one extracted Hit per line with the following data structure shown in figure 7.4.

Frame	Source				TDC-Value		ADC-Value
	SPU	Stack	ASIC	Channel	Coarse	Fine	

Figure 7.4: Parsed measurement-data structure

The *Frame* field contains the frame counter value. The *Source* fields indicate the source of the Hit. The TDC- and ADC-Value give information about the timestamp and energy of the Hit, respectively.

The emulated read-out data contains valid source information depending on the SPU and Sensor Stack configuration and Frame-field is incremented after every time slot. The TDC- and ADC-Value stay constant throughout the measurement.

The lost and invalid Hit-data rate (err_{hit}) of the measured datastream is determined for every Stack ($err_{hit}(spu, stack)$), because every Stack has different Frame Counter start value at the beginning of the measurement. Therefore the first and last Frame Counter values have to be determined for every Stack in order to calculate the total amount of Frames sent by a Stack and the ($err_{hit}(spu, stack)$).

$$\begin{aligned} err_{hit}(spu, stack) &= 1 - \frac{\text{total Hits received from Stack(spu, stack)}}{\text{total amount of Hits generated by Stack(spu, stack)}} \\ &= 1 - \frac{\sum_{asic}\sum_{channel} hit(channel)}{(\text{Frame Counter}_{end} - \text{Frame Counter}_{start} + 1) \cdot n_{Hits}} \end{aligned} \tag{7.3}$$

$$hit(channel) = \begin{cases} 1 & \text{if a valid hit is detected on the } channel \\ 0 & \text{if zero or invalid hit is detected on the } channel \end{cases}$$

The total error rate can also be determined:

$$err_{hit,total} = 1 - \frac{\sum_{spu} \sum_{stack} err_{hit}(spu, stack)}{n_{Stacks}} \tag{7.4}$$

The analyzer-program first determines the Frame Counter intervals for each Stack by parsing the beginning and the end of the CSV-data. This helps to reject the Hits with the Frame Counter values, that are out of the determined interval due to data corruption during measurement. During parsing, also Hits with wrong field-values are rejected (e.g. wrong TDC- and ADC-Value).

The valid Hit data is stored in a tree[1], where every leaf is a Hit. The data structure is shown in figure 7.5.

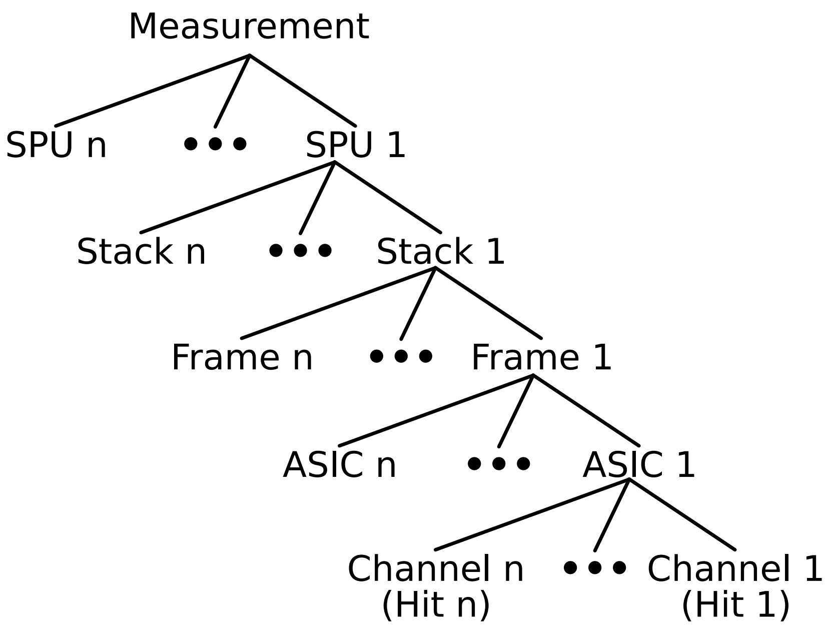

Figure 7.5: Parsed Hit data in a hierarchial tree structure

Transforming the data to the shown tree structure eases the search for the lost Hits. The number of leafs under a Stack are counted to determine the total Hits received from a Stack.

The results of the measurement are shown in table 7.3.2.

The commissioning-test showed that the CU works successfully in upstream direction, because the configuration of the Stacks involves sending of packets with the size in the order of 1 Mbyte.

However, the results clearly show that significant percentage of the detector data packets are corrupted during routing, even at lower rates compared to the target input data bandwidth of 700 Mbit/s.

[1]A data-structure with a hierarchial tree structure

n_{SPU}	n_{Stacks}	n_{Hits}	$\overline{bandwidth}$	$err_{hit,total}$
1	1	1	578 kB/s	0.01 %
1	2	2	1602 kB/s	3.30 %
1	2	4	2496 kB/s	3.85 %
1	2	20	9650 kB/s	5.69 %
1	2	40	18592 kB/s	8.05 %
1	2	80	36477 kB/s	N/R
2	3	1	1732 kB/s	0.07 %
2	4	2	3204 kB/s	7.63 %
2	4	4	4992 kB/s	11.34 %
2	4	20	19300 kB/s	27.84 %
2	4	40	37184 kB/s	39.86 %
2	4	80	72953 kB/s	N/R

Table 7.2: The table shows the measurement results. n_{SPU} stands for the number of the SPUs, n_{Stacks} for the total number of Stacks on the SPUs, n_{Hits} for the number of generated Hits by a Stack in a time slot and $err_{hit,total}$ for invalid and lost Hit rate. *N/R* stands for no response from the CU.

It is also apparent from the results that the corruption is higher, when the input bandwidth is halved into two SPU ports (compare the $err_{hit,total}$ for n_{SPU} =1, n_{Hits} =40 and n_{SPU} =2, n_{Hits} =20).

The reason for the corruption is probably that the Datatype Splitter is still not able to process fragmented Hyperimage packets at aimed data rates. However, the higher corruption rate with two SPUs indicate that the Arbiter could also some problems in case of more the one requester.

The results also show that the data is corrupted at input bandwidth of 1732 kB/s with two SPUs, whereas the CU could route the packets up to 2.3 MByte/s without errors. The reason could be the different measuring chain. The SPU Emulator test was in Linux and without Hyperion, whereas this test was conducted with Hyperion. As the TCP/IP Stacks are different, this could have led to different results.

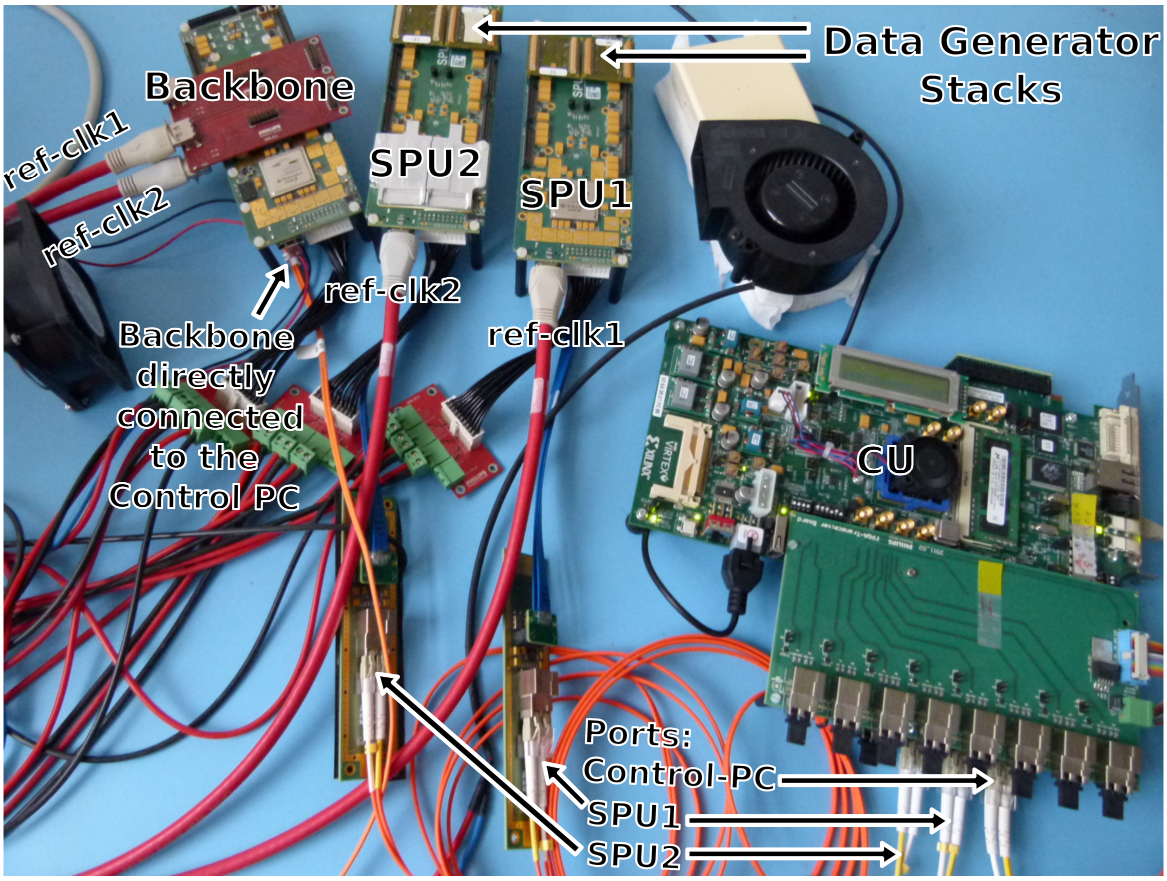

Figure 7.6: The photo shows the commissioning test setup with two SPUs. The SPUs are connected with blue cables to an external optical-fiber transceiver board, which are connected to the CU. The three red PCBs in the middle-left area of the picture are intermediate connector-boards for power supply of the SPUs.

8 Summary and Outlook

The goal of this thesis were conception, development and testing of an infrastructure for evaluation of hardware-based coincidence processing algorithms for PET image reconstruction.

A significant time of the project was spent for clocking-implementation of twelve transceivers simultaneously with a single clock source and an acceptable solution was presented in section 5.4.2.

Significant parts of the defined FPGA-architecture could be implemented and tested. The commissioning-tests show that the design works successfully in upstream direction.

Unfortunately, some migrated modules from the SPU design were not designed to process fragmented Hyperimage packets, which led to data corruption at higher input data rates in downstream direction. Due to time reasons, it was not possible to debug or replace the error-causing modules.

As a next step, the packet corruption problem in the downstream modules should be solved. Possible solutions are:

- The output behavior of the UDP/IP module could be altered. Buffering extracted Hyperimage packets and forwarding them unfragmentedly, could solve the data corruption problem.
- The Datatype Splitter could be rewritten to support fragmented packets.
- The commissioning-tests with two SPUs show that the Arbiter probably cannot cope with two frequent requesters. This could be solved by redesigning the arbiter.

Bibliography

[1] A. Jemal, F. Bray, M. Center, J. Ferlay, E. Ward, and D. Forman, “Global cancer statistics,” *CA: a cancer journal for clinicians*, 2011.

[2] M. Dodd, C. Miaskowski, S. Paul, *et al.*, “Symptom clusters and their effect on the functional status of patients with cancer.,” in *Oncology Nursing Forum*, vol. 28, p. 465, 2001.

[3] D. Berger, M. Engelhardt, and H. Henß, *Concise manual of hematology and oncology.* Springer Verlag, 2008.

[4] R. Thoeni, A. Moss, P. Schnyder, and A. Margulis, “Detection and staging of primary rectal and rectosigmoid cancer by computed tomography.,” *Radiology*, vol. 141, no. 1, p. 135, 1981.

[5] S. Gambhir, J. Czernin, J. Schwimmer, D. Silverman, R. Coleman, and M. Phelps, “A tabulated summary of the fdg pet literature,” *Journal of Nuclear Medicine*, vol. 42, no. 5 suppl, pp. 1S–93S, 2001.

[6] V. Schulz, T. Solf, B. Weissler, P. Gebhardt, P. Fischer, M. Ritzert, V. Mlotok, C. Piemonte, N. Zorzi, M. Melchiorri, *et al.*, “A preclinical pet/mr insert for a human 3t mr scanner,” in *Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE*, pp. 2577–2579, IEEE, 2009.

[7] A. Einstein, M. Henzlova, and S. Rajagopalan, “Estimating risk of cancer associated with radiation exposure from 64-slice computed tomography coronary angiography,” *JAMA: the journal of the American Medical Association*, vol. 298, no. 3, p. 317, 2007.

[8] M. Ritzert, P. Fischer, V. Mlotok, I. Peri, C. Piemonte, N. Zorzi, V. Schulz, T. Solf, and A. Thon, “Compact sipm based detector module for time-of-flight pet/mr,” in *Real Time Conference, 2009. RT’09. 16th IEEE-NPSS*, pp. 163–166, IEEE, 2009.

[9] S. Brown and J. Rose, “Fpga and cpld architectures: A tutorial,” *Design & Test of Computers, IEEE*, vol. 13, no. 2, pp. 42–57, 1996.

[10] M. Tetrault, J. Oliver, M. Bergeron, R. Lecomte, and R. Fontaine, “Real time coincidence detection engine for high count rate timestamp based pet,” *Nuclear Science, IEEE Transactions on*, vol. 57, no. 1, pp. 117–124, 2010.

[11] Bailey, *Positron Emission Tomography Basic Sciences.* New York: Springer, 2005.

[12] M. Reivich, D. Kuhl, A. Wolf, J. Greenberg, M. Phelps, T. Ido, V. Casella, J. Fowler, E. Hoffman, A. Alavi, *et al.*, "The [18f] fluorodeoxyglucose method for the measurement of local cerebral glucose utilization in man," *Circulation Research*, vol. 44, no. 1, pp. 127–137, 1979.

[13] J. Hubbell, "Review of photon interaction cross section data in the medical and biological context," *Physics in medicine and biology*, vol. 44, p. R1, 1999.

[14] Hybrid PET-MR system for concurrent ultra-sensitive imaging (HYPER-IMAGE). `http://cordis.europa.eu/fetch?CALLER=FP7_PROJ_EN&ACTION=D&DOC=1&CAT=PROJ&RCN=87568`, November 2011.

[15] E. Kim, P. Olcott, and C. Levin, "Optical network-based PET DAQ system: One fiber optical connection," in *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*, pp. 2020–2025, IEEE, 2010.

[16] A. McFarland, D. Newport, B. Atkins, D. Pressley, S. Siegel, and M. Lenox, "A CompactPCI Based Event Routing Subsystem for PET and SPECT Data Acquisition," in *Nuclear Science Symposium Conference Record, 2006. IEEE*, vol. 5, pp. 3091–3093, IEEE, 2006.

[17] Xilinx, *UG534 ML605 Hardware User Guide.*

[18] Xilinx, *DS150 Virtex-6 Family Overview.*

[19] IEEE Std. 802.3-2008, *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access method and Physical Layer specifications.*

[20] Xilinx, *UG029 Chipscope Pro Software and Cores.*

[21] J. Bergeron, *Writing testbenches: functional verification of HDL models.* Springer Netherlands, 2003.

[22] T. Wenisch, R. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. Hoe, "Simflex: statistical sampling of computer system simulation," *Micro, IEEE*, vol. 26, no. 4, pp. 18–31, 2006.

[23] Xilinx, *UG366 Virtex-6 FPGA GTX Transceivers Guide.*

[24] FPGA Editor Overview. `http://www.xilinx.com/support/documentation/sw_manuals/help/iseguide/mergedProjects/fpga_editor/fpga_editor.htm`, December 2011.

Nomenclature

ADC	Analog to Digital Converter
APD	Avalanche Photo Diodes
ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange, a character encoding scheme
ASIC	Application Specific Integrated Circuit
CRC	Cyclic Redundancy Check
CSV	Comma-Separated Values
CT	Computed Tomography
CU	Coincidence Unit
FIFO	First In First Out buffer
FPD	Field-Programmable Device
FPGA	Field-Programmable Gate Array
GTX	A marketing name for Xilinx gigabit-capable transceivers
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IP	Internet Protocol
ISE	Integrated Software Environment
ISO	International Organisation for Standardization
LCD	Liquid Crystal Display

LYSO	Cerium doped Lutetium Yttrium Orthosilicate
MAC	Media Access Controller
MR	Magnetic Resonance
MRI	Magnetic Resonance Imaging
OSI	Open System Interconnection
PCS	Physical Coding Sublayer in OSI-Model
PET	Positron Emission Tomography
PLL	Phase Locked Loop
PMA	Physical Medium Attachment Layer in OSI-Model
PMD	Physical Medium Dependent Layer in OSI-Model
PMT	Photo-Multiplier Tube
PMT	Photomultiplier Tube
RAM	Random Access Memory
SiPM	Silicon Photomultiplier
SPU	Singles Processing Unit
TDC	Time to Digital Converter
TOF	Time of Flight
UDP	User Datagram Protocol
VHDL	Very-high-speed integrated circuits Hardware Description Language