

Development of Digital Signal Processing with FPGAs for the Readout of the ATLAS Liquid Argon Calorimeter at HL-LHC

Diplomarbeit
zur Erlangung des wissenschaftlichen Grades
Diplom-Physiker

vorgelegt von

Steffen Stärz

geboren am 14.09.1985 in Hoyerswerda

TECHNISCHE UNIVERSITÄT DRESDEN

INSTITUT FÜR KERN- UND TEILCHENPHYSIK
FACHRICHTUNG PHYSIK

FAKULTÄT MATHEMATIK UND NATURWISSENSCHAFTEN

2010

Eingereicht am 14. Dezember 2010

1. Gutachter: Jun.-Prof. Dr. Arno Straessner
2. Gutachter: Prof. Dr. Kai Zuber

Abstract

The Liquid Argon calorimeter of the ATLAS detector at CERN in Geneva is supposed to be equipped with advanced readout electronics for the operation at High Luminosity LHC. In this diploma thesis the aspect of fast serial data transmission and data processing to be used for the communication between different readout modules and data storage buffers of the trigger shall be further developed.

Furthermore, the main focus is put on first preparation of the detector raw data with regard to a signal correction using a FIR filter. It is aimed at a most efficient, most resource economising and minimal latency causing solution that allows to process the huge amount of upcoming detector raw data in real time.

Therefore a via UDP/IP reconfigurable prototype of a 5-stage FIR filter with Gigabit Ethernet Interface was implemented in a Xilinx[®] Virtex-5 FPGA.

The performance reached is fully within the the requirements for the upgraded calorimeter readout of ATLAS.

Kurzdarstellung

Das Flüssig-Argon-Kalorimeter des ATLAS-Detektors am CERN in Genf soll für den Betrieb am High Luminosity LHC mit einer verbesserten Ausleseelektronik ausgestattet werden. In dieser Diplomarbeit wird der Aspekt der schnellen seriellen Datenübertragung und -verarbeitung weiter entwickelt, welche für die Kommunikation zwischen den Modulen der Ausleseelektronik und zum Datenspeicher des Triggers eingesetzt werden soll.

Desweiteren steht die Erstaufbereitung der digitalisierten Detektordaten hinsichtlich einer Signalkorrektur im Sinne eines FIR-Filters im Vordergrund. Ziel ist eine möglichst effiziente, also ressourcensparende und auf minimale Verarbeitungsdauer optimierte Lösung, die es erlaubt, die riesigen anfallenden Detektordaten in Echtzeit zu verarbeiten.

Dazu wurde ein per UDP/IP rekalkulierbarer Prototyp eines 5-stufigen FIR-Filters mit Gigabit-Ethernet-Anbindung in einem Xilinx[®] Virtex-5 FPGA implementiert.

Contents

Abstract	v
1 Introduction	1
2 The ATLAS Liquid Argon Readout System	3
2.1 The LHC Experiment	3
2.2 The ATLAS Detector	5
2.3 The Liquid Argon Calorimeters	6
2.3.1 Front End Boards and Signal Processing	9
2.3.2 Readout Driver Boards	14
2.3.3 Data Acquisition System	14
2.4 Detector Upgrade Motivation	15
2.4.1 A Possible Readout Scheme for the HL-LHC	15
2.4.2 Requirements of the Digital FIR Filter in the ROD FPGA	16
2.5 Energy Calculation	17
2.5.1 FIR Filter	17
2.5.2 Incoming Raw Data	17
2.5.3 Offset: Pedestal	18
2.5.4 Filter Coefficients	19
2.5.5 Output Data Format	20
3 Technical Introduction	23
3.1 Field Programmable Gate Arrays	23
3.1.1 FPGA Components and Termini	24
3.1.2 Clocking	26
3.2 ML507 and Virtex-5	29
3.3 VHDL: A Description Language for FPGAs	31
3.3.1 Verilog	31
3.3.2 VHDL	32
3.3.3 User Constraints	32
3.3.4 State machines	33
3.3.5 Block RAM	34
3.3.6 DSP48E Units	36
3.4 Ethernet	39
3.4.1 UDP/IP	40
3.5 Auxiliary Tools	44
3.5.1 ISE: A Xilinx Design Suite	44
3.5.2 Wireshark	44
3.5.3 Packet Builder	46

4	Implementation	47
4.1	Implementation of UDP/IP	47
4.1.1	UDP Transmitting Module	50
4.1.2	UDP Receiving Module	52
4.1.3	Resource Utilisation	53
4.2	Design Overview	54
4.3	The Filter Algorithm	56
4.3.1	The First Filter Design	56
4.3.2	The Second Filter Design	58
4.3.3	The Third Filter Design	61
4.4	Filter Calibration	63
4.5	RAM Management	66
4.6	High Clocking of the Multi channel Filter	69
4.7	Output Data Encoding	72
5	Results	75
5.1	FPGA Resource Estimation	75
5.2	Latency Estimation	77
5.3	Data Rates	79
6	Conclusion and Outlook	81
6.1	Data Transfer	81
6.2	Filter Development	81
6.3	HL-LHC and Physics Measurements	82
	List of Figures	83
	List of Tables	85
	List of Code Examples	87
A	Appendix	89
A.1	Design Simulation	89
A.2	Schematic of the Output Data Encoding Module	89
A.3	Global Design Parameters	91
A.4	Data and Calibration sample packets for 4 channels	94
	Bibliography	99

1 Introduction

The goal of the Large Hadron Collider (LHC) [1] is to record and analyse high energy particle collisions in order to get a deeper understanding of the fundamental physics laws of nature. That is, in fact, a very challenging and complex task as many different pieces have to fit together perfectly. That starts from the creation of a particle beam, includes the detection of collisions, and finally end with the analysis of events and their interpretation. The goal is to test existing theories like the Standard Model of particle physics or to confirm more complex, yet unproven theories.

Reliability plays a key role at all layers of any such investigation. Particularly the readout system of a detector has to be able to satisfy high standards to allow a maximum of data capture at a minimum dead time and measurement uncertainty.

The current ATLAS Liquid Argon (LAr) calorimeter readout at the LHC is designed for operation with a nominal luminosity of $10^{34} \text{ cm}^{-2}\text{s}^{-1}$. The Front End detector electronics is expected to resist the radiation of ten years of LHC operation in these conditions which refers to about 700 to 1000 fb^{-1} of integrated luminosity.

As an upgrade of the LHC to the High Luminosity Large Hadron Collider (HL-LHC) is foreseen in the years around 2020 where the luminosity is increased by one order of magnitude, the Front End electronics is expected to fail due to the expected, more than five times higher radiation levels and due to normal ageing.

In case the readout electronics is replaced, also the signal triggering is planned to be improved. Since higher luminosity is equivalent to a larger pile-up of proton-proton collisions, the very good granularity of the detector readout is foreseen to be also explored at the event trigger level. That puts a high demand on the performance of the readout to be of extremely large data bandwidth and with fast signal processing. Furthermore, the current on-detector data buffer size limits the maximum trigger latency as well as the data bandwidth. That opposes an update of the complex trigger system which is also foreseen. It is thus the task to investigate on a new readout system being able to cope with all future requirements.

In this work the idea of direct processing of the incoming data from the LAr calorimeter without buffering is studied, assuming functioning on-detector electronics for the LAr calorimeter that provides digital signal samples for its 182468 channels at the LHC bunch-crossing frequency of 40 MHz.

A Field Programmable Gate Array (FPGA) implementation of a digital Finite Impulse Response (FIR) filter is presented. It processes Analog-Digital Converter (ADC) data samples with three different gain factors sent by the on-detector electronics for

each channel in real time. It is aimed at a minimal total filter latency and minimal resource utilisation. At the same time an optimal data throughput shall be reached, in particular a parallel processing of multiple input channels.

In the current readout design five data samples are processed with conjoint gain. In contrast to that and due to the expected much higher occupancy of the upgraded system, each data sample will be treated separately with its own gain factor.

The presented FIR filter implementation includes an interface for online recalibration to easily update all filter parameters without reconfiguration of the entire FPGA. This is another design requirement because the filter parameters are adapted to the effective LHC luminosity, respectively to the expected pile-up ratio.

Additionally a first prototype of a high speed Ethernet link for data transfer is developed.

This document will start with a brief introduction of the ATLAS detector and a presentation of the current readout system in section 2. It includes a detailed motivation for the upgrade (section 2.4) and theoretical background on optimal filtering (section 2.5).

Section 3 will present FPGAs, its peculiarity and methods as well as tools to have them operating. Section 3.4 will introduce the Ethernet standard and the data transfer protocol used.

In section 4 consecutively the implementation of the FIR filter in an embedded Ethernet environment is shown, guiding through different stages of optimisation.

The obtained results will be discussed in section 5 with regard to reaching the high aims and requirements of the HL-LHC.

Finally an outlook will be given in section 6.

2 The ATLAS Liquid Argon Readout System

This chapter will present the current ATLAS Liquid Argon readout system and its components with impact on this work. A motivation for an upgrade of the current detector system will be given and the first processing of detector raw data will be explained.

2.1 The LHC Experiment

The largest human-built particle accelerator is the Large Hadron Collider (LHC) [1] near Geneva, Switzerland, at the European Organisation for Nuclear Research (CERN)¹. Not only the size of the circular proton-proton collider of 26.7 kilometers of circumference, its location of 70 to 100 meters under ground and its center-of-mass energy of 14 TeV are unique, but also the technological know-how and effort to control and bend the particle beam inside the LHC ring are remarkable. A complex magnet system with 1232 superconducting dipole and several thousand multi-pole magnets for focusing and defocusing is unbeaten. A magnetic dipole field strength of up to 8.3 T is required to achieve this task at full beam energy.

To provide the planned nominal instantaneous luminosity of $10^{34} \text{ cm}^{-2}\text{s}^{-1}$ an interaction rate of 40 MHz of 2808 bunches, each containing up to 1.15×10^{11} protons, is needed. This particularly high luminosity is required to be statistically sensitive to the rare processes of particle physics aimed to be studied. It is in the same view a never seen challenge to continuously detect and measure collision events with a time spacing of 25 ns.

After seven years of construction until the first operation, it was amazingly only a question of hours until the first proton beams were circulated in the LHC ring successfully on 10 September 2008. After an incident on 19 September, where serious damage was caused by a malfunctioning magnet [2], the machine could be taken into operation again on 20 November 2009. The first collisions at 450 GeV were detected just three days later [3]. This is a great result in comparison to the bringing into service of the Large Electron-Positron Collider (LEP) experiment that was situated in the same underground tunnel before its shutdown in November 2000. For this experiment it took one month to get the first collisions [4].

¹Conseil Européen pour la Recherche Nucléaire

There are four main experiments at the LHC: ALICE, ATLAS, CMS and LHCb.

ALICE (A Large Ion Collider Experiment) [5] is designed to study collisions of heavy ions. One centre of interest is the analysis of lead-lead nuclei collisions at 2.76 TeV nucleon energy of the lead beam to explore the quark-gluon plasma.

The LHCb ('b' represents the bottom quark) [6] aims at measuring CP violation and the interactions of the bottom quarks and is thus a specialized b-physics experiment.

The CMS (Compact Muon Solenoid) [7] and ATLAS (A Toroidal LHC Apparatus) [8] experiments are multi-purpose detectors. Main tasks are the search for the Higgs boson, the validation and measurement of parameters of the Standard Model and the exploration of physics at the TeV scale.

Furthermore, there are LHCf and TOTEM:

LHCf (Large Hadron Collider forward) [9] as the smallest experiment is a special purpose experiment for astroparticle physics intended to measure the energy and numbers of neutral pions produced by the collider to explain the origin of ultra-high energy cosmic rays.

TOTEM (Total Cross Section, Elastic Scattering and Diffraction Dissociation) [10] aims at measurement of total cross section, diffractive processes and elastic scattering.

2.2 The ATLAS Detector

The ATLAS detector [8] (figure 2.1) is a multi-purpose detector intended to cover a wide range of physics measurements [11]. It is designed in a nominally forward-backward symmetric cylindrical geometry and thus covers almost the complete solid angle (4π) around the interaction point².

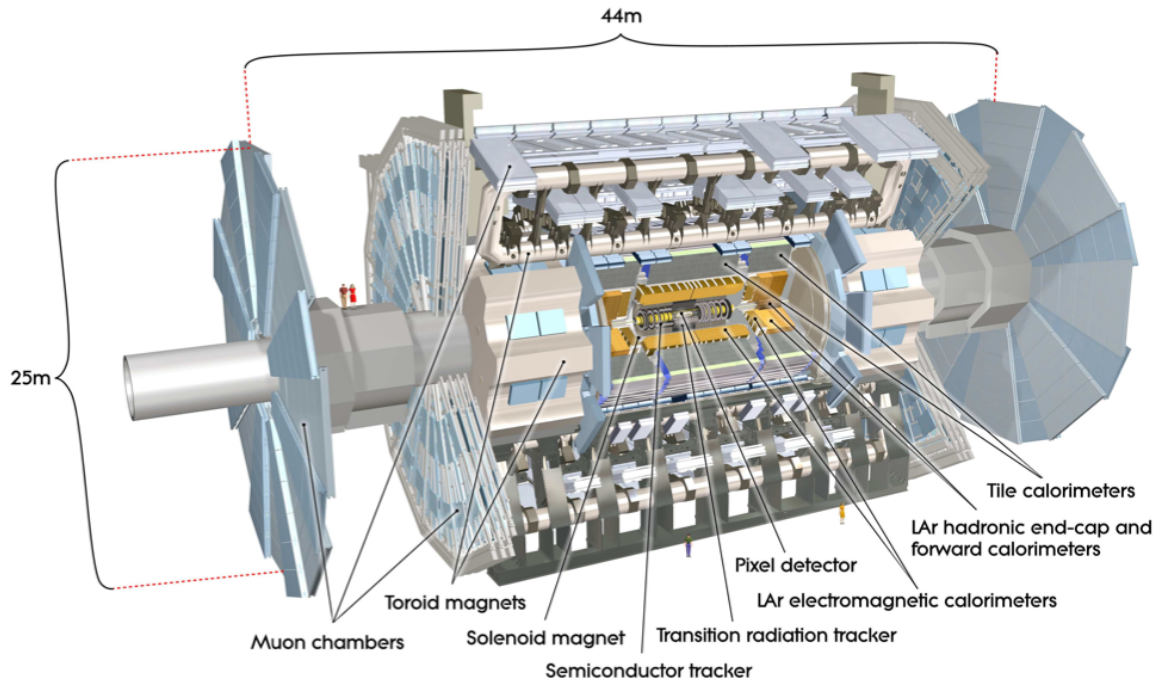


Figure 2.1: Outline of the ATLAS detector at the LHC [8].

The ATLAS detector consists of three overall detector systems: inner tracking detectors, calorimeters and outer muon spectrometers.

One of the major ATLAS detector systems is formed by a complex system of Liquid Argon (LAr) calorimeters.

²The coordinate system of the ATLAS detector is a Cartesian right-handed one. The nominal interaction point is the origin, the x-axis points to the centre of the LHC ring, the y-axis is directed upwards and the z-direction points in anti-clockwise beam direction to complete the right-handed coordinate system. To make use of the symmetries of the ATLAS detector, a polar angle θ is defined with respect to the z-axis and an azimuthal angle φ is defined in the transverse x-y plane around the beam axis.

2.3 The Liquid Argon Calorimeters

The Liquid Argon (LAr) calorimeter [12] plays a key role in ATLAS. It is dedicated to trigger on and to provide precision measurements of photons, electrons, jets, and missing transverse energy.

The LAr calorimeters (see figure 2.2) include the electromagnetic barrel calorimeter (EMB) in the center and two endcap calorimeter systems (EC) on the front and the backside. Both EC cryostats include a forward calorimeter (FCAL), an electromagnetic endcap calorimeter (EMEC) and a hadronic endcap calorimeter (HEC).

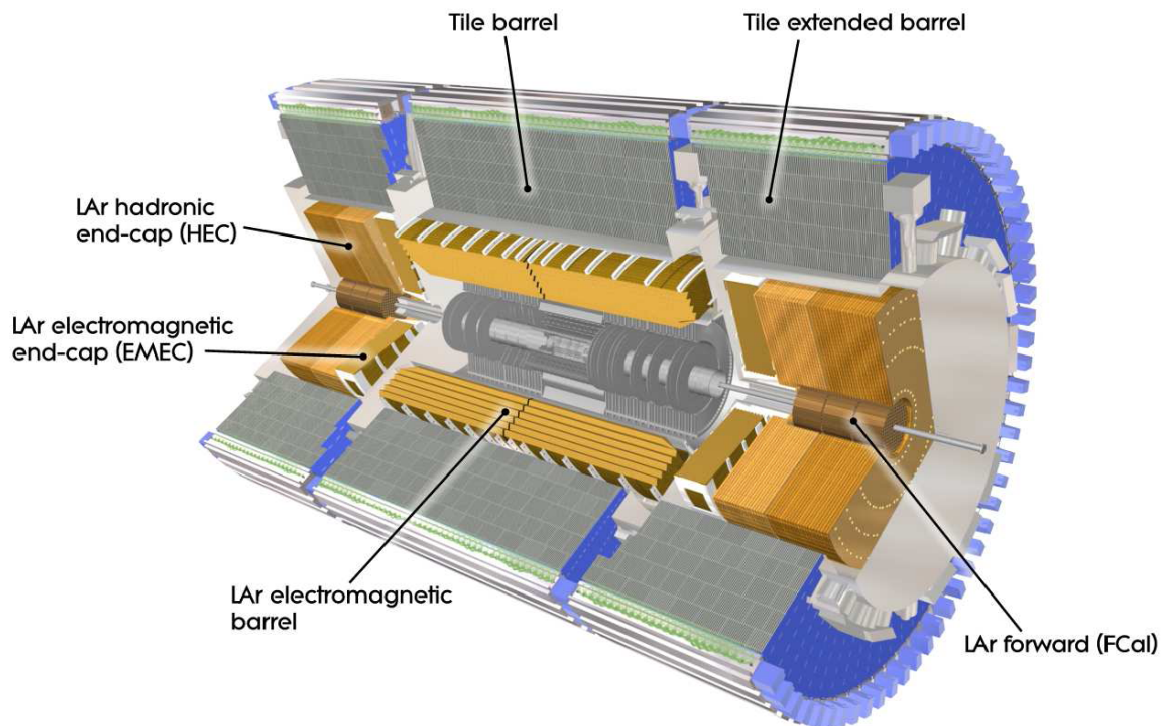


Figure 2.2: The ATLAS calorimeters are shown. The LAr calorimeters (inner yellowish part) are inside the tile hadronic calorimeters [13].

All in all the LAr calorimeters consist of 182468 detector cells [14]. Their electronic signals, being proportional to the energy deposit in each cell, need to be read out, digitised and processed. Figure 2.3 illustrates the arrangement of the different cells throughout the detector. They are separated into three layers, each with different geometric and spatial resolution³.

The electronic readout of the LAr calorimeters is separated into a Front-End system mounted directly on the cryostat feedthroughs in the radiation environment, and a Back-End system located in an off-detector underground counting house.

³The pseudorapidity η is defined as $\eta = -\ln(\tan(\frac{\theta}{2}))$ using the angle θ of a particle relative to the beam axis.

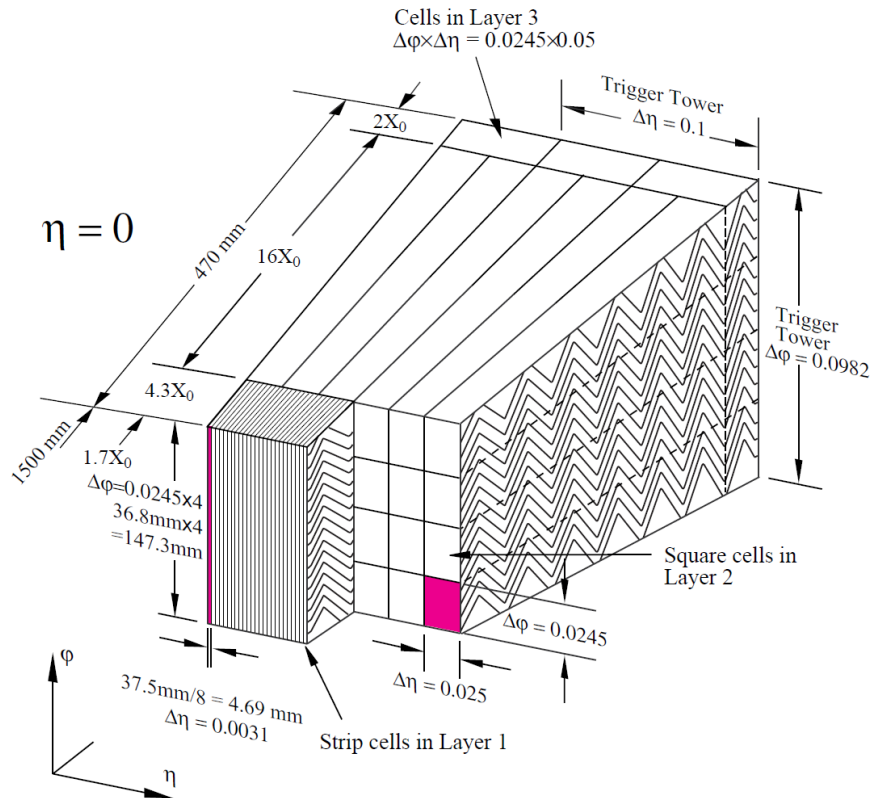


Figure 2.3: Sketch of a LAr calorimeter barrel module where the different layers and their arrangement in single cells is shown. The granularity in η and φ for the cells and the trigger towers is given [8].

The Front-End system is composed of Front End Boards (FEBs), which perform the readout and signal processing of the calorimeter cells, of calibration boards to inject precision calibration signals, of boards to provide analog sums for the Level 1 (L1) trigger, and of control and monitoring boards. The FEBs will be presented in more detail in the next section. Up to 128 channels are combined in a group and processed by one of a total of 1524 FEBs.

The Back-End system includes Readout Driver (ROD) boards to perform digital filtering of the signals based on Digital Signal Processors (DSPs), being presented in section 2.3.2.

Figure 2.4 gives an overview of the data path of the current Read Out system from the detector through the FEB and the ROD to the Data Acquisition System (DAQ). The different stages will be explained in the following sections.

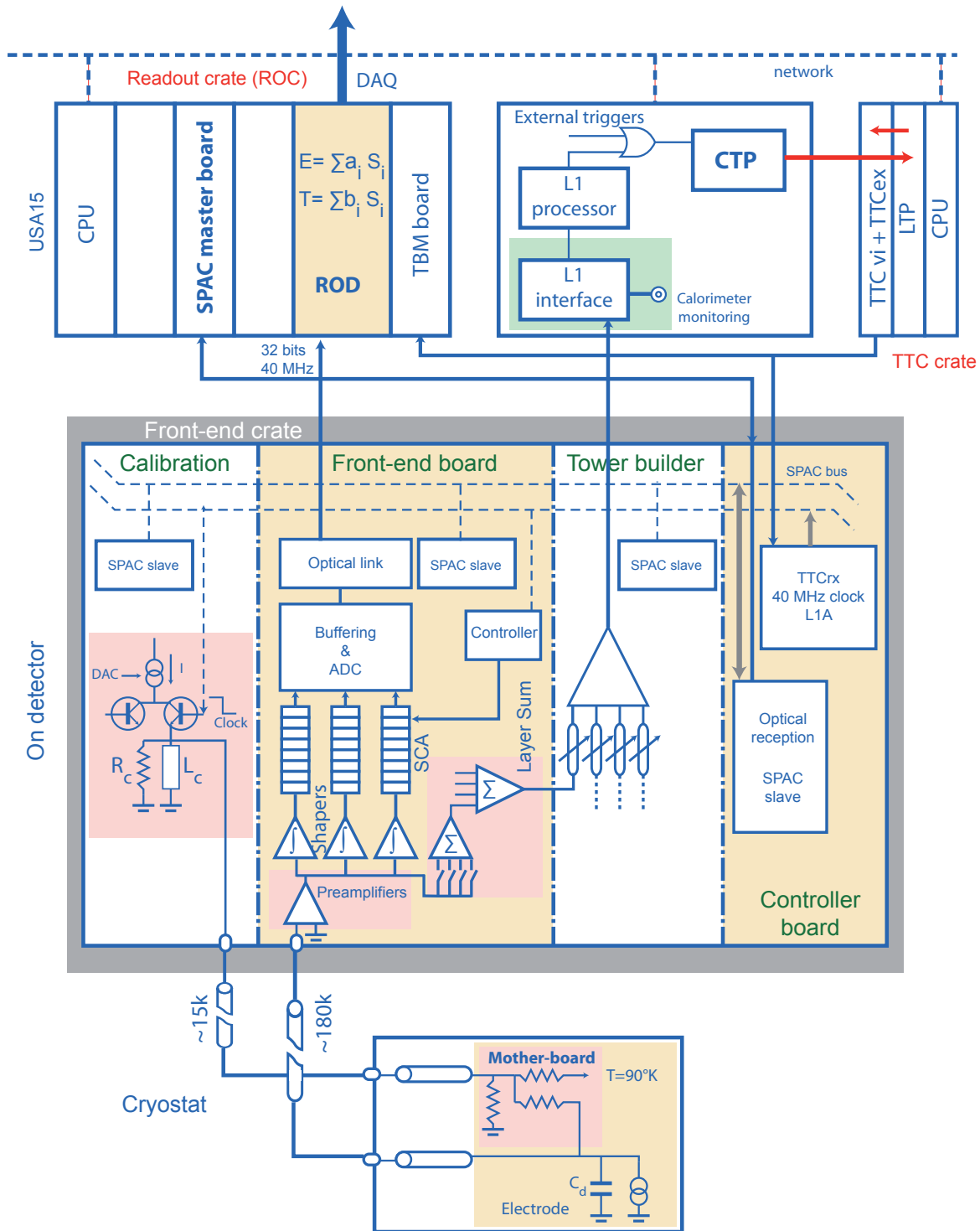


Figure 2.4: The current LAr calorimeter readout scheme illustrates the different stages of signal capturing (cryostat), pre-processing (front end crate), processing (ROD) and data storage (DAQ) [14]. The combined electronic signals of several channels for the first level trigger are routed on a parallel data path to the ROD system.

2.3.1 Front End Boards and Signal Processing

The signal processing on the Front End Boards (FEBs) includes pre-amplification, shaping, analog buffering, digitisation and gain-selection of the detector input signal. In front of that sequence an analog signal summing is performed to feed the L1 trigger.

A detailed description of the FEBs is given in [13].

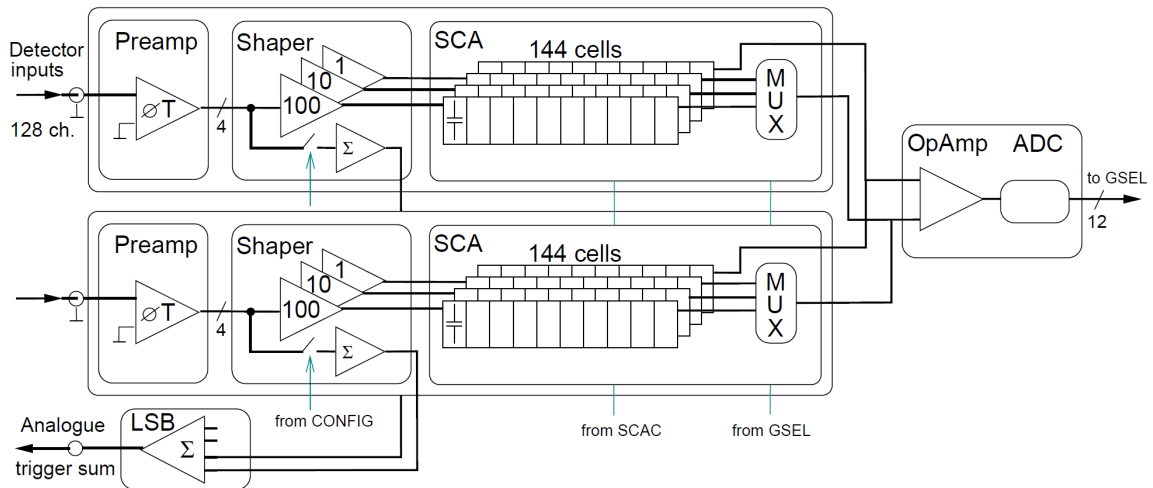


Figure 2.5: The analog signal processing chain of the LAr calorimeter FEB is shown including the pre-amplifier, shaper, analogue buffer (SCA) and digitisation. Note that there is one ADC for each amplification stage and the gain selection for one channel is done by an ASIC (not shown) after digitisation [13].

Pre-amplification

The raw signals of the ATLAS barrel and endcap calorimeters are cabled out of the detector and sent to the FEBs. The pulse shape of a calorimeter signal is illustrated in figure 2.6. It is an approximately triangular current pulse with a sharp rise of some few nanoseconds and an extended falling edge of about 400 ns due to drifting electrons in the LAr calorimeter gaps.

The pre-amplification serves as a first stage of signal amplification of the specific detection current which has a typical value of about 2 to 3 $\mu\text{A}/\text{GeV}$. That leads to a range of interest of nA up to several mA. The pre-amplification circuit is designed to achieve excellent linearity and noise performance. For the precision readout, each channel is calibrated separately.

Shaping

The shaper applies an $\text{CR}-(\text{RC})^2$ analog filter to the signals in order to further optimise the signal-to-noise ratio and to remove the long tail of the detector response. Noise

is reduced by two integrations that limit the bandwidth.

The shaping function uses a RC time constant of 13 ns, representing a compromise between minimizing thermal noise, which decreases for slower shaping, and pile-up noise, which increases for slower shaping.

The resulting bipolar, zero-integrated pulse has a peaking time of about 35 to 40 ns as depicted in figure 2.6. That peaking time is optimised for the LHC design luminosity of $10^{34} \text{ cm}^{-2}\text{s}^{-1}$ as shown in figure 2.7.

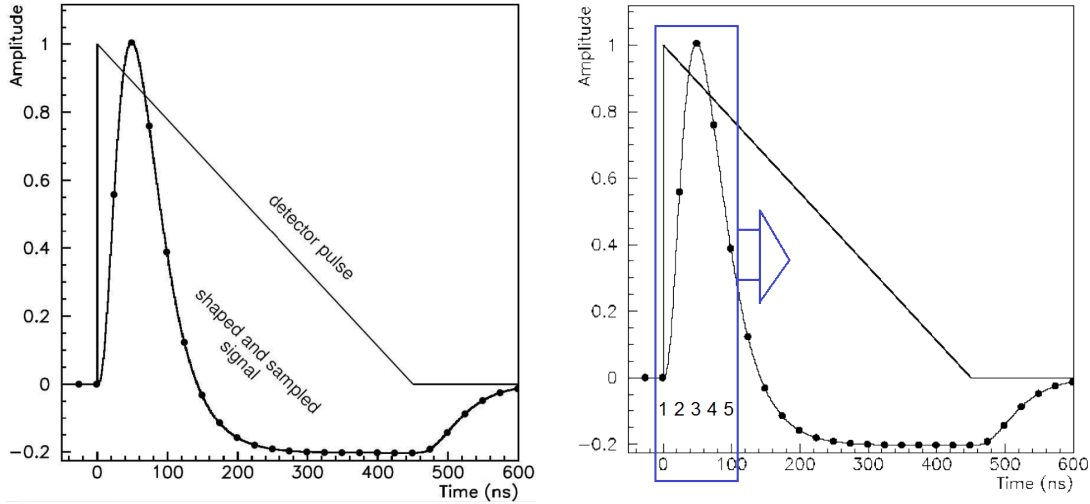


Figure 2.6: Left: The triangular positive pulse generated by the front end electronics is being reshaped to a bipolar, zero-integrated pulse. It is sampled every 25 ns. Right: A sliding window of five samples, numerated as can be seen, illustrates the sliding window of the filtering [13].

The shaped signal is further amplified and split into three overlapping linear gain scales to achieve the full dynamic range. The absolute gain values are 0.8 (LO gain), 8.4 (MED gain) and 82 (HI gain).

Besides this amplification, the shaper also contains a preliminary analog summation of four input channels to feed the L1 trigger system.

The shaper with all its described features is implemented in a four-channel Application Specific Integrated Circuit (ASIC).

Analog Buffering

A sampling frequency of 40 MHz is used to read out the shaper output and to store it in analog form by a Switched-Capacitor Array (SCA) analog pipeline. All three gain scales for each of four calorimeter channels are processed by the SCA.

A depth of 144 cells of the SCA is chosen in order to provide at least an eight-event de-randomising buffer based on the assumption of a maximum L1 trigger latency of 100 bunch crossings ($2.5 \mu\text{s}$) and five signal samples per event.

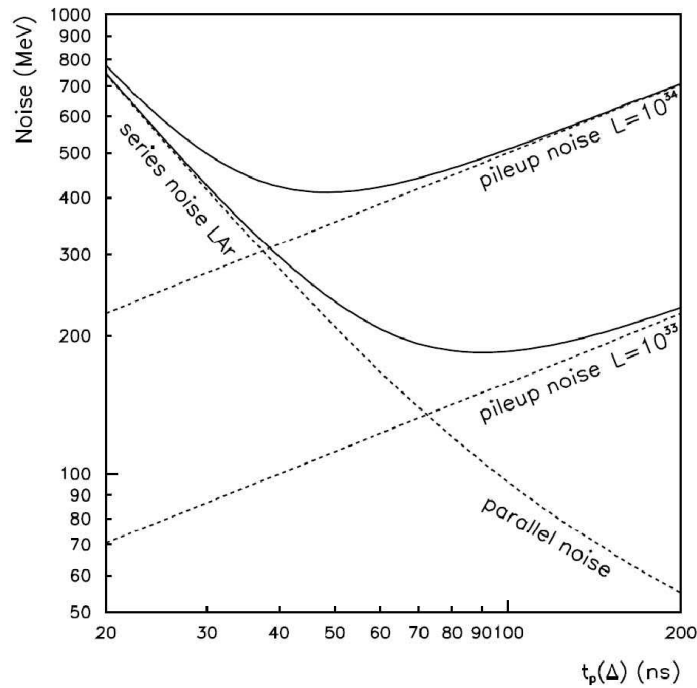


Figure 2.7: The different contributions from pile-up noise and from electronics noise (series and parallel) are given for the luminosities of 10^{33} and $10^{34} \text{ cm}^{-2}\text{s}^{-1}$ in dependence of the peaking time [13].

To deal with DC-coupled shaper input signals in the range from -0.9 V to $+2.5 \text{ V}$ with a baseline voltage of 0 V the SCA is powered asymmetrically ($V_{SS} = -1.7 \text{ V}$, $V_{DD} = +3.3 \text{ V}$).

SCA Controller and FEB Control

To bookkeep the signal samples stored in the SCA a Switched-Capacitor Array Controller (SCAC) ASIC is used operating at 5 MHz . Its task is to manage the read and write addresses of the SCA in a way to await and to evaluate the L1 trigger decision and to discard not triggered events. Positively triggered events are transferred to the de-randomising buffer of signals awaiting digitisation.

One First In - First Out (FIFO) memory is used to temporarily store SCA addresses during L1 trigger latency, a second one keeps track of the samples in digitisation and another FIFO manages recycling of used addresses.

Digitisation

At the end of the SCA, dual op-Amp chips are used to couple them to commercial 12 bit ADCs operating continuously at the same frequency as the SCAs.

To map the SCA output voltage range onto the limited ADC input signal range of 1 V , a less than unity gain is applied. In addition, to allow sampling of both the positive

and the negative lobes of the shaped calorimeter signals, a voltage offset (pedestal) is added. It corresponds to roughly 1000 ADC counts. Figure 2.8 shows the distribution of the pedestals over all FEBs.

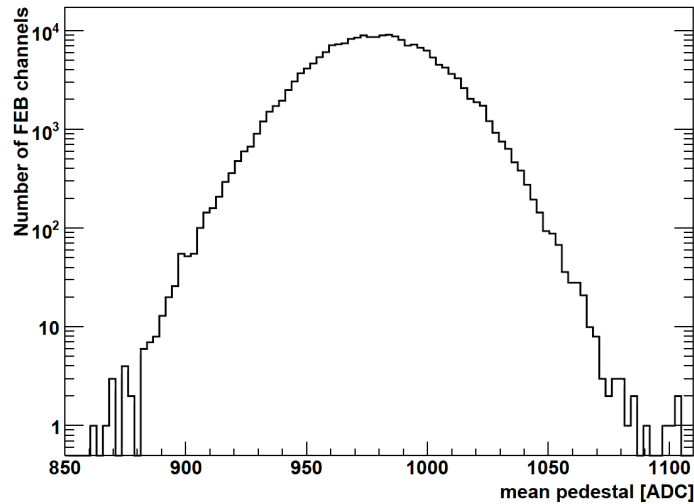


Figure 2.8: The measured pedestal distribution (for high gain) of all channels is depicted. The narrow and uniform distribution confirms proper assembly of the FEBs [13].

Gain Selection

After the digitisation, a selection of the most adequate ADC values out of the three possible gains (see table 2.3) has to be done individually for each calorimeter channel. The aim is to read out the ADC which has the highest gain but is not yet saturated. To do so, a Gain Selector ASIC (GSEL) is applied that can be configured to read out a single, two or all three ADCs. In normal LHC running the “Auto Gain” mode dynamically selects one (conjoint) gain scale for five consecutive samples.

The information of which gain is selected by the GSEL for the conjoint sample is mapped in two extra gain selection bits.

The gain selection method requires that six digitisations are performed per channel in order to find the conjoint gain selection and to get the final five samples. It avoids systematic effects which would be encountered in combining the five samples if they were not all digitized on the same gain scale. Further detailed studies of systematic effects on independently chosen gains for the 5 samples have to be carried out.

Level 1 Trigger Summing

To serve the L1 trigger, the FEB also provides two stages of analog sums of the signals. The first stage is incorporated in the shaper chips which sum their four input channels. A second stage are plug-in Layer Sum Boards (LSBs) mounted on the FEB which further sum the shaper outputs.

The LSBs finally transmit their data from the FEB to trigger boards, which perform further summing and processing to feed the L1 trigger.

FEB Data Output

The GSEL data is fragmented into 16 bit words, including a start 0 bit, a parity bit (P), two gain bits and the 12 bits from the ADC for one sample. A full frame contains an event header with Bunch Crossing Identification (BCID), the information of eight channels and additional status information as shown in table 2.1.

Table 2.1: Readout data format of the ATLAS LAr calorimeter FEB. The information of eight channels (gain and ADC value) is combined in a packet of 16 bit words with an overhead of four words including the Bunch Crossing Identification (BCID), event numbering and further details. A frame start and frame end tag enclose the event frame [13].

bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
frame start tag	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
event header	0	P	0	0	ADCID				EVENTN							
	0	P	0	0	BCID											
sample header	0	P	0	0	F	L	B	A	CELLN							
sample data	0	P	gain		ADC											
sample data	0	P	gain		ADC											
sample data	0	P	gain		ADC											
sample data	0	P	gain		ADC											
sample data	0	P	gain		ADC											
sample data	0	P	gain		ADC											
sample data	0	P	gain		ADC											
sample data	0	P	gain		ADC											
even trailer	0	P	0	0	1	S	E	SCAC status								1
end frame tag	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The FEB serial output uses a 1.6 Gb/s optical link to ensure the data transfer to the ROD. Nevertheless, the overall readout frequency is limited to the L1 trigger rate of 75 kHz.

2.3.2 Readout Driver Boards

The Readout Driver (ROD) boards [15] build the core of the off-detector data processing. Each of the 192 ROD modules receives the raw data from eight FEBs through eight optical fibres and processes thus up to 1024 detector cells.

Each ROD houses four processing units mounted on daughter boards, each equipped with two DSPs. The calculation done by the DSPs is an optimal filtering (see section 2.5). An average processing time of $12\ \mu\text{s}$ is required per event being consistent with the L1 trigger rate of 75 kHz.

The filtered energy, the signal time, and a pulse quality factor are then finally forwarded to the DAQ.

2.3.3 Data Acquisition System

The primary task of the Data Acquisition System (DAQ) [8] in the view of the readout chain is to receive digitised and filtered raw data from the RODs. It temporarily holds the data in local buffers to be analysed by a second Level 2 (L2) trigger, taking into account Regions of Interest (ROIs). Events that have passed the L2 trigger enter the event-building system and the Event Filter (EF), a third stage of trigger. Events that have passed the EF successfully are then subject to permanent storage at the CERN computer centre.

The DAQ also incorporates means for the configuration, controlling and monitoring of its soft- and hardware components.

2.4 Detector Upgrade Motivation

The current calorimeter readout is designed for LHC operation with a nominal luminosity of $10^{34} \text{ cm}^{-2}\text{s}^{-1}$ [14]. The Front End electronics is expected to resist the radiation damage of ten years of LHC operation, considering a Total Ionising Dose (TID) of 5 krad(Si), a Non-Ionizing Energy Loss (NIEL) equivalent to $1.6 \times 10^{12} \text{ n/cm}^2$ (1 MeV neutrons) and Single-Event Effects (SEE) from $7.7 \times 10^{11} \text{ hadrons/cm}^2$ (considering hadrons with $E > 20 \text{ MeV}$) [16].

That radiation dose refers to a total luminosity of about 700 to 1000 fb^{-1} . An upgrade of the LHC to the HL-LHC [17] foresees cumulative luminosities of the order of 3000 fb^{-1} due to an increase of the instantaneous luminosities up to $5 \times 10^{34} \text{ cm}^{-2}\text{s}^{-1}$ in the years beyond 2019/2020.

The Front End electronic is thus suspected to fail and it is therefore required to design new electronics for the ATLAS LAr calorimeter being able to cope with a five times higher radiation level and ionisation dose [18].

Besides this, the limited on-detector data buffer size, namely the SCA, does not allow any increase in L1 trigger latency nor bandwidth. That contradicts the plans of a possible modification of the ATLAS trigger system, where additional trigger signatures will lead to a longer trigger processing time.

Along with the higher luminosity, the number of proton-proton interactions and pile-up events is also expected to be about five times higher compared to the maximum the LHC will reach. Thus, an upgrade also on the detector readout side is demanded.

2.4.1 A Possible Readout Scheme for the HL-LHC

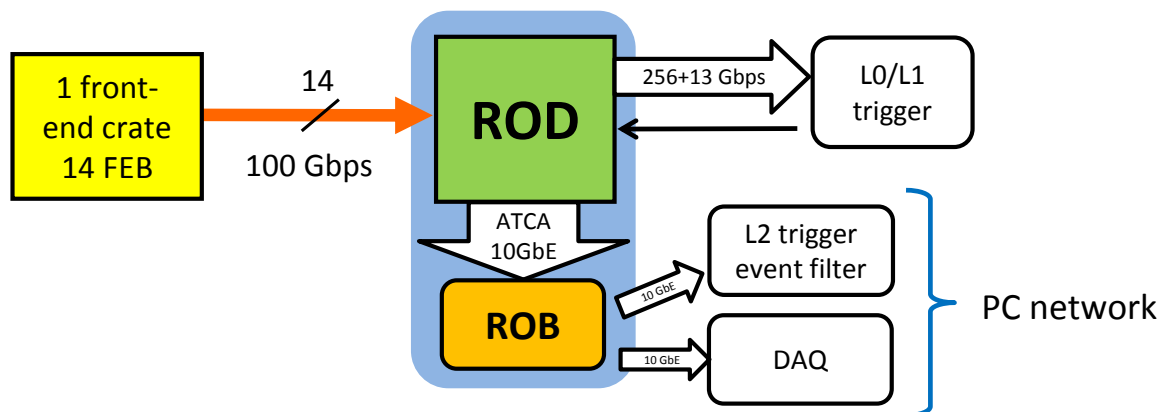


Figure 2.9: The envisaged readout scheme for the HL-LHC ATLAS LAr calorimeter [18].

In figure 2.9 a possible new readout scheme is shown. In this scheme the functionality of each component of the readout chain will change in comparison to the current design.

The FEBs have to preamplify, shape and digitise the data at a bunch crossing rate of up to 40 MHz. The analog signal buffering on the FEBs is subject to be omitted. Instead of buffering data on detector, all the data will be transmitted directly to the RODs with minimum latency.

The trigger design is also subject to change. It is foreseen to introduce a fast Level 0 (L0) trigger at an output rate of 40 MHz. A smaller portion of data is provided to the L1 trigger. Energy, bunch crossing time and quality factor are to be calculated by the RODs for each cell based on the received raw data. When the L1 trigger accept signal is arriving, this data is sent to the Readout Buffer (ROB) at about 200 kHz.

This implies the RODs to provide a digital data buffer which holds the data until it is sent out to the ROB.

The data rate per FEB is estimated to be about 100 Gb/s summing up to a total rate of 1.4 Tb/s for one ROD coping with 14 FEBs. In the barrel region of the LAr calorimeter, this corresponds to a slice of $2\pi/32$ in azimuthal angle.

2.4.2 Requirements of the Digital FIR Filter in the ROD FPGA

The task of a digital FIR filter for the ROD FPGA is to process the ADC data samples with three different gain factors sent by the FEB for each channel in real time at an input frequency of 40 MHz.

Due to the constant high data flow and the required continuous processing, despite selecting five samples conjointly with the same gain as it is done in the current readout, each sample has to be considered to have its own gain factor.

To minimise hardware utilisation to cover the full number of 182468 channels, the FIR filter should be able to deal a maximum of channels in parallel with a maximum speed requiring a minimum of resources.

Additionally it should be easy to update all filter parameters.

2.5 Energy Calculation

Here the aimed data processing and calculation is presented that plays the key role in this work. The basic idea is to implement a FIR filter. The incoming and outgoing data of that FIR filter are defined as well as the FIR filter itself.

2.5.1 FIR Filter

A Finite Impulse Response (FIR) filter is a special type of a discrete-time filter [19]. The impulse response, the filter's response to a Kronecker delta input, is finite because it settles to zero in a finite number of sample intervals.

The output $y(n)$ of a FIR filter can be described in terms of its input $x(i)$ by the equation

$$y(n) = a_0 \times x(n) + a_1 \times x(n-1) + \dots + a_N \times x(n-N), \quad (2.1)$$

where a_i are the filter coefficients, or weighting factors, and $N+1$ is the filter depth. This formula can be rewritten to a convolution in the following way:

$$y(n) = \sum_{i=0}^N a_i \times x(n-i). \quad (2.2)$$

As the aimed application will be the calculation of the energy by filtering channel (ch) and gain dependent data samples S_i with pedestal correction Ped_i , the final equation has the form

$$E(ch) = \sum_{i=1}^d a_i^{(\text{gain})}(ch) \cdot \left(S_i^{(\text{gain})}(ch) + Ped_i^{(\text{gain})}(ch) \right) \quad (2.3)$$

with the filter depth d .

The functionality of the filter can be represented easily with a sliding window of the width of the filter depth d which is given on the right side of figure 2.6.

Thus the general approach for the determination of the detected energy E from the measured samples S_i is that of optimal filtering [20] to maximise the signal to noise ratio.

2.5.2 Incoming Raw Data

The filter input signal are data samples S_i provided by a 12 bit ADC at different levels of amplification referenced by a 2 bit gain value "gain". That sums up to 14 bits of information conventionally transferred as a 16 bit (or 2 byte) value introducing two dummy bits. The input data bus is thus defined as shown in table 2.2.

Table 2.2: Definition of the input data bus: 2 bits encode the gain value and 12 bits are used for the ADC energy value. The position of the two dummy bits (x) is in principle arbitrary, but it is chosen to easily increase the data bit width of the ADCs.

Bit number	15	14	13	12	11 downto 0
Information	gain		x	x	energy

The energy samples range from 0 to $2^{12} - 1$ as the direct output data of a 12 bit ADC. The measured energy deposit is amplified by a certain gain factor before its digitisation. Table 2.3 indicates the gain definition.

Table 2.3: Definition of the 2 bit gain value

Value	Identification	Definition
00	invalid gain	invalid energy sample
01	low gain	amplification of the energy value by a factor of 0.8
10	medium gain	amplification of the energy value by a factor of 8.4
11	high gain	amplification of the energy value by a factor of 82.0

2.5.3 Offset: Pedestal

As shown in figure 2.6, the physical pulse after the reshaping also enters the negative domain. Due to the properties of an ADC, only values in the positive domain can be sampled. Thus an offset, namely a pedestal Ped_i (see equation 2.3), is introduced to shift the pulse into the positive domain and allowing the ADC to sample the full pulse range correctly. The bit width of this pedestal has to be at least the bit width of the ADC of 12 bits. Paying attention to the sign of the pedestal being negative, an additional bit is required to encrypt this information. To be conform with standards of information technologies it is chosen to spend three further bits finally introducing the pedestal as a 16 bit value, which corresponds to an `int` value in C programming language.

Additionally, one has to take into account that the pedestal value is channel dependent (see figure 2.8). Thus this filter parameter has to be obtained from calibration of each channel.

In the following sections the pedestal will occur without index i . That is due to the fact that the corresponding energy sample S_i enters the filtering only once and the pedestal correction can be done directly. So the pedestal corrected energy sample can be further processed and the pedestal itself does not play a role anymore.

2.5.4 Filter Coefficients

The proper choice of the filter coefficients is crucial for a physical correct result of the optimal filtering [20]. The filter coefficients have to be calculated offline based on a Monte Carlo simulation of the expected events and the pile-up, as well as the measured electronic noise in order to achieve an optimal signal to noise ratio. Details of the method applied are described in [21].

The filter coefficients convert a dimensionless 12 bit ADC value being proportional to the energy deposit of the detected particle to a real measured energy by formula 2.3. Thus they carry the unit of MeV.

As the aim is to achieve a filter result with optimal accuracy, the required bit width of the filter coefficients a_i has to be studied.

The following bit-wise example multiplications will demonstrate the behaviour and influence of the bit width used for the filter coefficients. The binary multiplication of 7×7 in comparison to the binary multiplication of 7×7.5 shall illustrate this influence on the uncertainty of the least significant bit.

$$\begin{array}{r}
 a_{3 \text{ bit}} \times b_{3 \text{ bit}} \\
 111 \times 111 \\
 \hline
 111 \\
 111 \\
 111 \\
 \hline
 = 110001
 \end{array}
 \quad (2.4)$$

$$\begin{array}{r}
 a_{3 \text{ bit}} \times b_{4 \text{ bit}} \\
 111 \times 111 \text{ 1} \\
 \hline
 111 \\
 111 \\
 111 \\
 11 \text{ 1} \\
 \hline
 = 110100 \text{ 1}
 \end{array}
 \quad (2.5)$$

From this example one can see that the 4th highest bit is influenced. That clearly indicates two things: First, for a 3 bit known value $a_{3 \text{ bit}}$ in a multiplication only the first three highest bits are reliable (as mathematically no higher precision can be achieved). Secondly it is no use to increase the precision of the second factor by more than the precision of the first factor and a lower number of significant bits would decrease the precision of the final result.

It obviously turns out that it is optimal to set the bit width of the two factors identically. That leads to the temporary conclusion to introduce the filter coefficients as 13 bit values, including one bit for the sign.

Now the channel dependence, and more crucially, the gain dependence have to be taken into account, bringing in the amplification factors encoded by the gain with direct impact on the bit width of the filter coefficients a_i . Two different approaches are considered:

1. Including the gain factor range in the filter coefficients bit width. That would mean a global increase of bit width of all filter coefficients for all three gains of at least the corresponding number of bits required to represent the ratio of the highest to the lowest gain factor of $\log_2(\frac{82}{0.8}) < \log_2 128 = 7$ bits,

2. Defining different pre-scaling factors $b_{n(\text{gain})} = 2^{n(\text{gain})}$ for each gain. That would make use of low bit width filter coefficients $\tilde{a}_i$ redefining the actual filter coefficients to the gain dependent product of $a_i \equiv b_{n(\text{gain})} \times \tilde{a}_i$.

Additionally, the actual range of the filter coefficients has to be respected. Computer simulations [22] come up with ranges of the filter coefficients for the three gains as shown in table 2.4, with initial resolution of 12 bits arising from the considerations above.

Table 2.4: The different ranges of the filter coefficients a_i of equation 2.3 as simulated [22] and the derived required minimal bit width for the 3 different gains. The range of the filter coefficients takes into account the previously defined gains of the energy samples (table 2.3).

gain identification	(decimal) range of $a_i^{(\text{gain})}$	12 bit resolution
low gain	-2000 to 2000 (-2^{11} to 2^{11})	$0.5 (2^{-1})$
medium gain	-200 to 200 (-2^8 to 2^8)	$0.05 (2^{-4})$
high gain	-20 to 20 (-2^5 to 2^5)	$0.005 (2^{-7})$

For the two approaches it can be concluded:

1. Including the gain factor range in the filter coefficients globally has to cover the range of 2^5 to 2^{11} with a granularity of 2^{-7} . Here, one ends up with a total bit width of 18. Taking into account the sign, another bit is consumed, so that 19 bits are required in total.
2. The bit width of the filter coefficients could remain at 12 bits (excluding the sign bit) by introducing binary pre-scaling factors in the range of 2^{-1} , 2^{-4} and 2^{-7} . Effectively, a pre-scaling range of 6 bits would be required.

2.5.5 Output Data Format

The output data format is predefined in the current readout system to a 16 bit number [23]. The calculated energy following formula 2.3 has a foreseen full range of -8 GeV to 4 TeV with a stipulated granularity of 1 MeV. That translates to integer numbers ranging from 1 to 4000000 , or 2^0 to about 2^{22} . Despite the possible concern of losing data precision by such a packing of a 22 bit value to a 16 bit bus, it shall be reminded, that the initial precision of the measured energy is 12 bits only, and, as shown in the example calculation 2.4 and 2.5, it is not possible to gain any further precision. The physical precision remains at 12 bits, thus the output bus has to carry at least those 12 significant bits plus 1 bit for the sign, leaving 3 bits for some encryption.

For the current readout system the packing procedure for valid values is defined in [23] which is applied in this work. Additionally the encryption of out-of-range and invalid values has been taken into account and is chosen as given in table 2.5.

Table 2.5: The data output specifications to pack a 22 bit energy value to a 16 bit bus to represent the filter result $E(ch)$ including range and sign. The encoding also allows an identification of invalid and out-of-range values.

range (2 bit)	sign (1 bit)	represented values (MeV) (13 bit)	granularity
00	1	$-8192 \dots -1$	1 MeV
00	0	$0 \dots 8191$	1 MeV
01	0	$8192 \dots 65565$	8 MeV
10	0	$65536 \dots 524287$	64 MeV
11	0	$524288 \dots 4194303$	512 MeV
01	1	$E < -8192$	out of range
10	1	$E > 4194303$	out of range
11	1	E invalid due to invalid S_i	invalid

The full energy range is divided into four different physically relevant domains, namely with granularity 2^0 , 2^3 , 2^6 and 2^9 MeV. That requires two bits, called “range”, to encode the corresponding domain. Taking into account one bit for the sign leaves 13 bits for the energy value. Doing so, the full stipulated range of 22 bits is covered.

It must be remarked, that the granularity presented in table 2.5 is not equal to the precision. The granularity is only the corresponding unit of energy that applies to the encoded value in the output bus. Its precision remains at 12 bits and equals thus the double of the granularity. A calculated energy of 4193280 MeV would have a granularity of 512 MeV, but it would still only have a precision of 1024 MeV.

3 Technical Introduction

This chapter elucidates hardware concepts, standards and tools required in this work.

3.1 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is a highly integrated circuit (IC) designed to be configured by the user or designer after manufacturing. It is a Programmable Logic Device (PLD) with the advantage to be reconfigurable.

FPGAs are often the alternative of using Application Specific Integrated Circuits (ASICs). Their capability to implement arbitrary functionality opens numerous fields of application. In comparison to ASICs, FPGAs offer the following advantages [24]:

- reconfigurability
- fast implementation
- low costs for small quantities
- low design risk

Several disadvantages on the other hand have to be mentioned:

- lower clock frequencies
- lower density of logic blocks
- higher power consumption at comparable functionality.

As any other IC, an FPGA contains input and output ports (I/O blocks), logic blocks and interconnects. The reconfigurability is provided by switch boxes of the interconnects illustrated in figure 3.1.

To determine the specific configuration of an FPGA, a Hardware Description Language (HDL) is used as described in section 3.3.

The configuration of an FPGA does not represent a programme that is executed as it is the case for processors or micro controllers. The HDL code design rather describes the behaviour of circuits on a higher level of abstraction. Thus it is possible to describe processes that run completely independently and in parallel and to duplicate complete calculation chains. A high speed of execution can be reached and the overall integrity is only limited by the logic resources of the FPGA used. The strength of FPGAs is

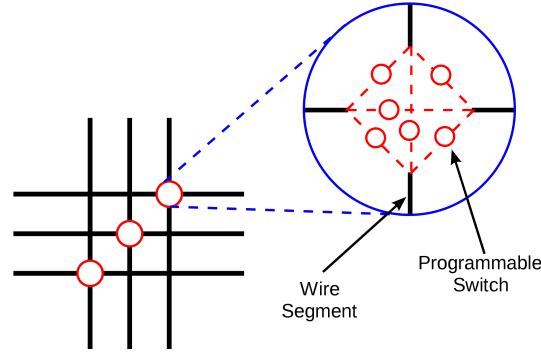


Figure 3.1: The reconfiguration of FPGAs is based on a hierarchy of programmable switch boxes where wire segments can be interconnected user specifically [25].

their massive parallelism offered by their architecture. Furthermore, in comparison to DSPs, they have the advantage of fast and multiple I/Os.

Some typical fields of application are digital signal processing, ASIC prototyping, medical imaging, speech recognition, cryptography, computer vision and a growing variety of other fields.

3.1.1 FPGA Components and Termini

Here the most important FPGA components and termini are briefly introduced. More detailed information can be found e.g. in [26].

Look-up Tables

Look-up Tables (LUTs) are small memory structures with n inputs (a_i), one output (O) and 2^n entries. The memory block can be programmed to implement any logical function of the n inputs.

An example for $n = 3$ shall demonstrate the idea of a LUT:

$$O = ((\bar{a}_0 \wedge a_1) \vee (\bar{a}_0 \wedge a_2)) \quad (3.1)$$

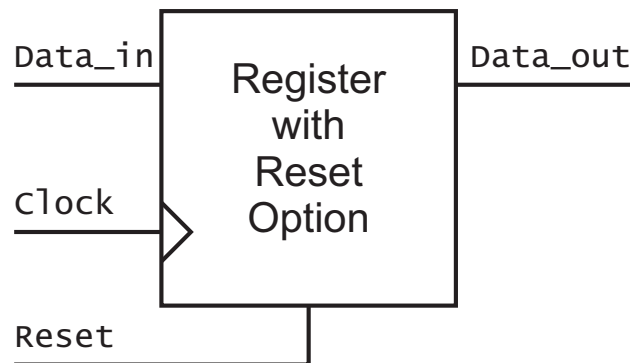
The LUT can be specified by its logic function (e.g. equation 3.1) or by its truth table (e.g. table 3.1). A more sophisticated and economical way to identify the LUT is to read the output column of its truth table downwards based on the static order of inputs, here “0101 0100” binarily, or “54” in hexadecimal, respectively.

Table 3.1: Truth table for the logic function $O = ((\bar{a}_0 \wedge a_1) \vee (\bar{a}_0 \wedge a_2))$

a_1	a_2	a_0	O
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

Registers

Registers (figure 3.2) are storage elements and allow data flow control. Unlike LUTs, they require a clock signal to propagate the input information to the output. Further different control signals are possible to apply, e.g. clocked (synchronous) or non clocked (asynchronous) reset, enabling or initialisation.

**Figure 3.2:** A register

Pipelining

Pipelining (pipelined processing) describes the capability to accept and process input data although the processing of the previous input data and creating the corresponding output is not yet terminated. E.g. in a chain of four clocked registers the input data will have to propagate through each register (and probably some combinational paths) before reaching the output after four clock cycles. Unlike for a detector with non cumulative dead-time, the input remains sensitive for new data. Only a delay of four clock cycles of the output data timing with respect to the input data timing has to be taken into account when designing, but the data flow itself is not interrupted.

Multiplexing

A very useful logic structure, particularly for FPGAs, is the multiplexer that selects one of its inputs depending on the value of its select input (s). It simply corresponds to the `if then else` statement for binary choices or to the `case` statement for higher levels of choice. It can be used for data selection as well as to decrease the amount of required data transmission lanes by increasing the data transmission frequency on the lanes with respect to the input signals.

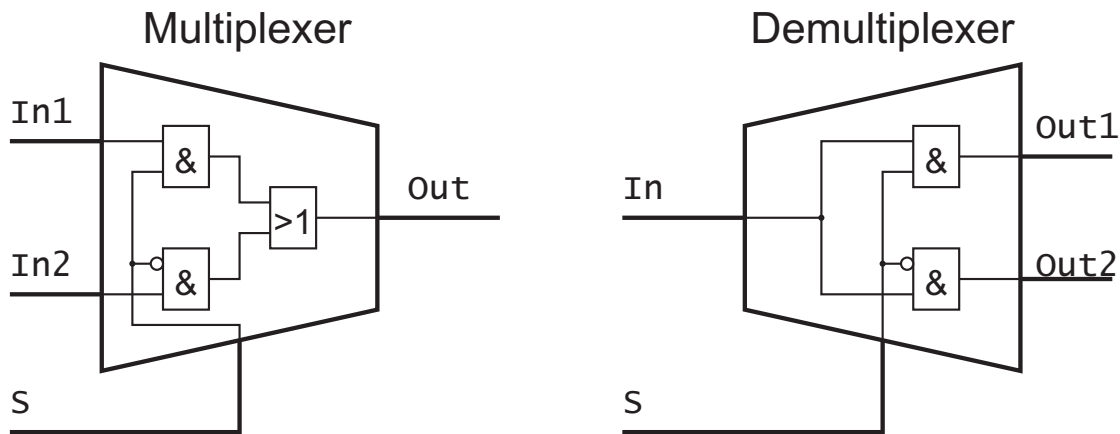


Figure 3.3: A multiplex - demultiplex unit

Interleaved Processing

Pipelining requires that the input data rate (clock frequency) is equal to the output data rate, as well as all internal processing clock frequencies. Considering a processing unit that can operate at much higher frequencies than the input leads to the idea of multiplexing several inputs to the processing unit and to demultiplex its output again. The processing frequency has to be a multiple of the data input frequency. This procedure is called interleaved processing.

3.1.2 Clocking

Stable clocking is crucial for any high frequency hardware application. The basic idea is to take an oscillator with a determined frequency known and to use this as a reference clock to generate all frequencies from it required for the design. In general there are 2 different ways to do so: The Digital Clock Manager (DCM) [27] and a Phase-Locked Loop (PLL) [28].

The Digital Clock Manager

The Digital Clock Manager (DCM) contains a Delay-Locked Loop (DLL) to completely eliminate clock distribution delays, by deskewing the DCM's output clocks

with respect to the input clock. The DLL contains delay elements (individual small buffers) and control logic. The incoming clock drives a chain of delay elements, thus the output of every delay element represents a version of the incoming clock delayed at a different point.

The control logic contains a phase detector and a delay-line selector. The phase detector compares the incoming clock signal against a feedback input and steers the delay line selector, essentially adding delay to the output of the DCM until the incoming clock signal and the feedback coincide.

The DCM is a standard module available in the Xilinx[®] CORE Generator providing several output frequencies: phase shifted (0, 90, 180 and 270 degrees), doubled (0 and 180 degrees), halved and user specified with arbitrary ratios. A `locked` output signal indicates when the DCM has properly engaged.

The Phase-Locked Loop

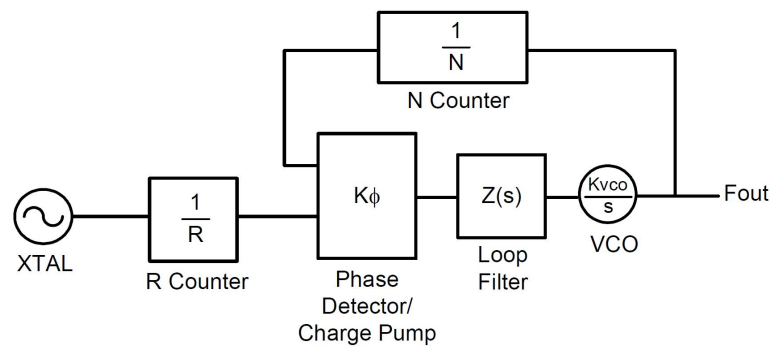


Figure 3.4: Basic layout of a Phase-locked loop (PLL) [28]

The Phase-Locked Loop (PLL) starts with a stable crystal reference frequency XTAL (see figure 3.4). The R counter divides this frequency to a lower one, which is called the comparison frequency f_{comp} as shown in figure 3.4. This is one of the inputs to the phase detector. The phase-frequency detector outputs a current that has an average value that is proportional to the phase error between the comparison frequency and the output frequency, after it is divided by the N divider. The constant of proportionality is called K_ϕ . This constant turns out to be the magnitude of the current that the charge pump can source or sink. If one takes this average DC current value from the phase detector and multiplies it by the impedance of the loop filter, $Z(s)$, then the input voltage to the Voltage Controlled Oscillator (VCO) can be found. The VCO is a voltage-to-frequency converter and has a proportionality constant of K_{VCO} . This tuning voltage adjusts the output phase of the VCO, such that its phase, when divided by N, is equal to the phase of the comparison frequency. Since phase is the integral of frequency, this implies that the frequencies will also be matched, and the output frequency will be given by:

$$f_{out} = \frac{N}{R} \times XTAL. \quad (3.2)$$

This applies for the locked state only. It requires a certain time (in the order of μs) for the PLL to acquire a new frequency. For FPGA applications, R and N are fixed by integer numbers. That implies that the PLL can only generate frequencies that are a multiple of f_{comp} . To achieve an optimal performance, it is preferable to have the comparison frequency as high as possible. In the case of the Virtex-5 FPGA, $f_{comp,max} = 1000 \text{ MHz}$ is the maximum.

The implementation of a PLL is done via a module provided by Xilinx[®] where R is chosen commonly for all output frequencies and N can be set individually for up to six output frequencies of the same module.

It turned out to be very practicable to instantiate two different PLL modules: one module with fixed output frequencies (50 MHz for the LCD panel, 125 MHz for the 1 GbE Ethernet interface) and another with variable N and R, the design parameters `clk_mult` and `clk_div` respectively (see appendix A.3). As for the DCM each PLL provides a `locked` signal to indicate the proper engagement.

3.2 ML507 and Virtex-5

The FPGA used in this work is a Virtex-5 XC5VFX70T from Xilinx® mounted on the industrial Xilinx® test board ML507 as shown in figure 3.5. This board supplies various input and output ports (I/Os) such as tri-mode Ethernet (up to 1 Gb on copper), USB, VGA, DVI, RS-232, SATA, PCIe and different kinds of I/Os for user specified purposes such as a LCD module, five switches, an eight bit dip- and a rotary switch. On the bottom side of the board a compact flash memory slot as well as a DDR2 block Random-Access Memory (RAM) slot can be found, both equipped with dedicated mass storage devices.

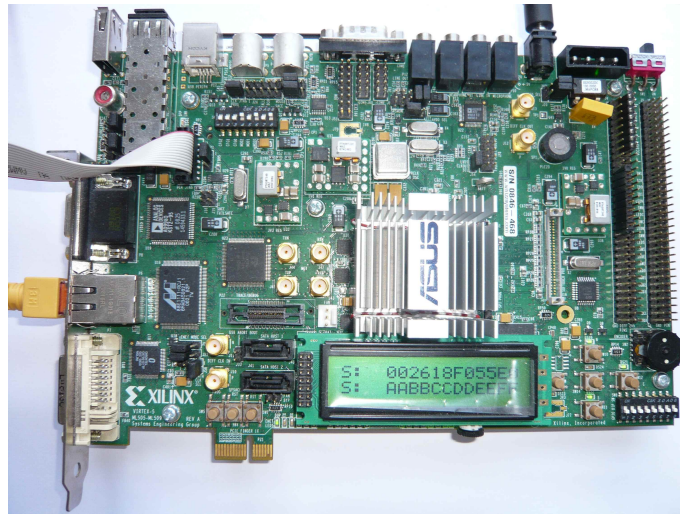


Figure 3.5: The Xilinx® test board ML507 with a Virtex-5 on it uses a RJ45 connector for the 1 Gigabit Ethernet link. The ribbon cable connects the board to the USB programming device.

The following list highlights the most important features of the Virtex XC5VFX70T [29].

- Real 6-input LUT technology
- Powerful clock management tile clocking (DCMs and PLLs)
- True dual-port RAM blocks
- Advanced DSP48E slices (25×18 , pipelining)
- System monitoring capability on all devices
- Tri-mode 10/100/1000 Mb/s Ethernet MACs
- 65 nm copper CMOS process technology

Table 3.2 summarises the main resources of the XC5VFX70T. Each Virtex-5 FPGA “configurable slice” contains four LUTs and four flip-flops (registers).

To monitor ongoing activities, eight user specific status LEDs can be used on the test board. For a more advanced monitoring there is an LCD panel with 2 times 16

Table 3.2: Resources of the Virtex XC5VFX70T [29]

resource	amount
Configurable Slices (registers, LUTs)	11200
Slice Registers	44800
Slice LUTs	44800
DSP48E slices	128
Block RAM	5328 kB
User I/O	640

characters. Its interface consists of three controlling bits and four data bits which is half the width naturally generated by a character. Before actually writing text to the LCD panel via its interface, there is a load sequence to follow where different parameters have to be defined such as cursor behavior (on, off; underline, full; blinking or not) and positioning (auto scroll to the left or right, or no scroll).

The LCD panel is controlled using a state machine, requiring a functioning clock network. State machines will be briefly explained in section 3.3.4.

3.3 VHDL: A Description Language for FPGAs

As FPGAs are reconfigurable hardware devices, it is required to know at least one HDL being capable to transcript man thought algorithms into a device specific description. The two most frequently used kinds of such HDL in the domain of FPGAs are Very high speed integrated circuit Hardware Description Language (VHDL) and Verilog.

HDLs differ from software programming languages because they include ways of describing the propagation of time and signal dependencies (sensitivity). A compiler transforms these descriptions and maps them to the logic resources of the PLD, e.g. an FPGA. As all describing code is then running in parallel and not executed sequentially, as know from software programming languages, the approach of HDLs is completely different. The code does not consist of a sequence of controls and it is thus required to describe e.g. sequential algorithms in separable logic components, so called modules, components or entities.

Each component contains a restrained functionality and is sensitive to input and output data, being defined as the port of the component. To increase the flexibility, generics can be used to define ports of variable size or any other application specific configuration.

In this view, HDL components could roughly be compared to functions in software programming languages, where the input ports can be compared to the parameters of the function and the output ports equal the return values.

3.3.1 Verilog

The major difference between Verilog and VHDL is syntax. The following 64-to-32 bit multiplexer (code example 3.1) shall serve as a brief example:

Code Example 3.1: 64-to-32 bit multiplexer in Verilog

```

1 // start by the definition of the ports of the module
2 module muxer (
3     // Inputs
4     clk, rst,
5     data_in, mux,
6     // Outputs
7     data_out
8 );
9
10 // a second definition of the ports has to be done in Verilog here
11 input clk; // clock
12 input rst; // reset
13 input mux; // multiplexing bit: 0 or 1
14 input [63:0] data_in; // 63:0 describes a bus of 64 bits
15
16 output [31:0] data_out; // the order could also be inverted (0:31)
17
18 // a process is started on every rising edge of the clock signal:
19 always @(posedge clk) begin
20     if (reset == 1'b1) begin // if reset = 1 then
21         data_out <= 64'b0; // set full bus to 0
22     end
23     else begin
24         if (mux == 1'b1) begin // if multiplexing bit = 1 then

```

```

26     data_out <= data_in[63:32]; // assign upper 32 bits to output bus
    end
    else begin
28         data_out <= data_in[31:0]; // assign lower 32 bits to output bus
    end
30 end
end

```

3.3.2 VHDL

The example multiplexer module is described in Very high speed integrated circuit Hardware Description Language (VHDL) as follows (code example 3.2):

Code Example 3.2: 64-to-32 bit multiplexer in VHDL

```

1  -- start by the definition of the ports of the module, again
2  entity muxer is
3      port (
4          -- controls
5          clk      : in  std_logic;           -- here the definition of the ports
6          rst      : in  std_logic;           -- is given once and complete
7          en       : in  std_logic;           -- in the entity port definition
8          -- Inputs
9          data_in  : in  std_logic_vector(63 downto 0); -- the 64 bit bus again
10         mux      : in  std_logic;
11         -- Outputs
12         data_out : out std_logic_vector(31 downto 0) -- inverting the order
13         );                                           -- of the bus by 0 to 31
14 end muxer;

16 -- a process is defined by a sensitivity list (here it is clk only)
17 process (clk)
18 begin
19     if rising_edge(clk) then -- start at each rising edge of clk
20         if (reset = '1') then -- if reset = 1 then
21             data_out <= (others => '0'); -- set full bus to 0
22         elsif (mux = '1') then -- else: if mux bit = 1 then
23             data_out <= data_in(63 downto 32); -- assign upper 32 bits to output
24             bus;
25         else -- else
26             data_out <= data_in(31 downto 0); -- assign lower 32 bits to output
27             bus;
28         end if;
29     end if;
30 end process;

```

It has to be remarked, that in this work Verilog has not been applied. All developed code was done in VHDL.

3.3.3 User Constraints

Before starting to think about any algorithms to implement, the outline of the FPGA has to be set. That means to define the I/O ports of the top level module and to assert them to the hardware pins of the FPGA by a one-to-one definition, namely the “User Constraints File” (ucf), being the first file in the VHDL project to be determined as given in code example 3.3.

Code Example 3.3: General outline of a User Constraint File

```

1 TIMESPEC "TS_clk_100MHz" = PERIOD "clk_100MHz" 10ns HIGH 50 %;
2 Net "leds8[0]"      LOC = H18 | IOSTANDARD = LVCMOS25;
3 ...
4 Net "leds8[7]"      LOC = AE24 | IOSTANDARD = LVCMOS18;
5 Net "LCD_DATA[7]"  LOC = T11 | IOSTANDARD=LVCMOS33;
6 Net "LCD_RW"       LOC = AC10 | IOSTANDARD=LVCMOS33;
7 Net "LCD_RS"       LOC = J17 | IOSTANDARD=LVCMOS25;
8 Net "LCD_E"        LOC = AC9 | IOSTANDARD=LVCMOS33;

```

Net "xxx" indicates the name of the signal xxx used in the VHDL code and the LOC statement assigns it to the hardware pin which can be found in the data sheet of the target FPGA. That may, of course, vary from device to device. Further constraints can be defined, such as timing constraints or the IOSTANDARD which indicates the voltage used for the logic levels.

3.3.4 State machines

The idea of a state machine is, as its name suggests, to define a sequence of different states that, based on certain stimuli, change from one state into another to control, e.g. an interface device. A state machine can be separated into three different parts:

1. A clocked process converting the current (**state**) to the new state (**next_state**),
2. a combinatory process with a sensitivity list of the state and all other input stimuli to assign the new state accordingly, and
3. an optional list of concurrent statements or other clocked processes depending on the (current) state that determine all controlling outputs of the machine.

The first part translates in VHDL to code example 3.4.

Code Example 3.4: First part of a state machine

```

1 architecture behavior of lcd is      -- start body of VHDL module
2 -- user defined states: names are conventionally chosen to be most self-
   explanatory
3     type display_state is (init, ..., set_addr, char, done);
4 -- the current state and the next_state for the next clock cycle are defined
5     signal state, next_state : display_state := init;
6 begin
7     ...
8     change_state: process(clk)      -- a process that is sensitive to the clock
9         signal
10    begin
11        if risingedge(clk) then      -- reacts on every rising edge of the clock
12            if rst = '1' then         -- always take into account a reset!
13                state <= init;        -- one should go back to the init state
14            else                      -- (whatever is defined to be or happen in init
15                )
16                state <= next_state;  -- else one selects the next_state that was
17                chosen
18            end if;                  -- in the 2nd part (whatever was chosen)
19        end if;
20    end process;
21    ...
22 end behaviour;

```

Code example 3.5 demonstrates the second part of a state machine.

Code Example 3.5: Second part of a state machine

```

1  display: process(state, i)      — a process with sensitivity to the
2  begin                          — state and a counter i
3      case state is              — one has to cover all possible states
4          when init =>           — if state = init
5              ...                — maybe initialise the counter i
6          when char =>           — if state = char
7              if(i = latency) then — has counter reached end value?
8                  next_state <= done; — (equals: has enough time passed?)
9                  i <= 0;          — if so, it is done
10             else
11                 next_state <= char; — else remain in char state
12                 i <= i + 1;        — to keep counting
13             end if;
14         when done =>           — if done state
15             if pos_flag = '1' then — react on new stimulus, pos_flag
16                 next_state <= set_addr;
17             elsif write_flag = '1' then — or write_flag
18                 next_state <= char; — to enter new state chain
19             else
20                 next_state <= done; — or just keep in done state
21             end if;
22         end case;
23     end process;

```

In the following code example 3.6 an optional list of concurrent assignments depending on the state is given.

Code Example 3.6: Optional list of concurrent assignments in a state machine

```

1  with state select              — depending on the state
2      tx_byte <=                 — set data to the according value
3      ...                       — (here it is not clocked, so concurrent)
4      upper_pos_bits & lower_pos_bits when set_addr,
5      upper_4bit & lower_4bit when char,
6      "00000000" when others; — cover all states, e.g. done
7
8  with symbol select            — some further external stimuli
9      upper_4bit <= ...          — that could finally modify
10
11 with posx select               — the data (tx_byte)
12     lower_pos_bits <= ...      — in any way

```

3.3.5 Block RAM

Another mandatory device for any data processing is Random-Access Memory (RAM). The Xilinx[®] CORE Generator provides modules that can be adjusted easily to user specific needs, as shown in figure 3.6. Most important settings are the RAM size required (data width and address depth), RAM organisation (usage of sources) and RAM access (single port or dual port).

To handle read and write requests independently, it is favourable to choose dual port RAM providing one port for writing and another for reading, both with their own clock, increasing flexibility for the design.

Aiming at a design dealing with Ethernet packets (see the following section 3.4), it turned out to be a convenient choice using 64 bit RAM with 256 addresses multiplying

up to 2 kB in a double way: On the one hand, most interfaces are based on 8 bit or multiples of 8 bit, so the RAM data width should be a multiple of 8 bit as well (to stay conform with conventions in information technology those multiples should be again integer powers of 2). On the other hand, an Ethernet packet is (normally) limited to 1522 bytes and an entire packet should be stored in the RAM.

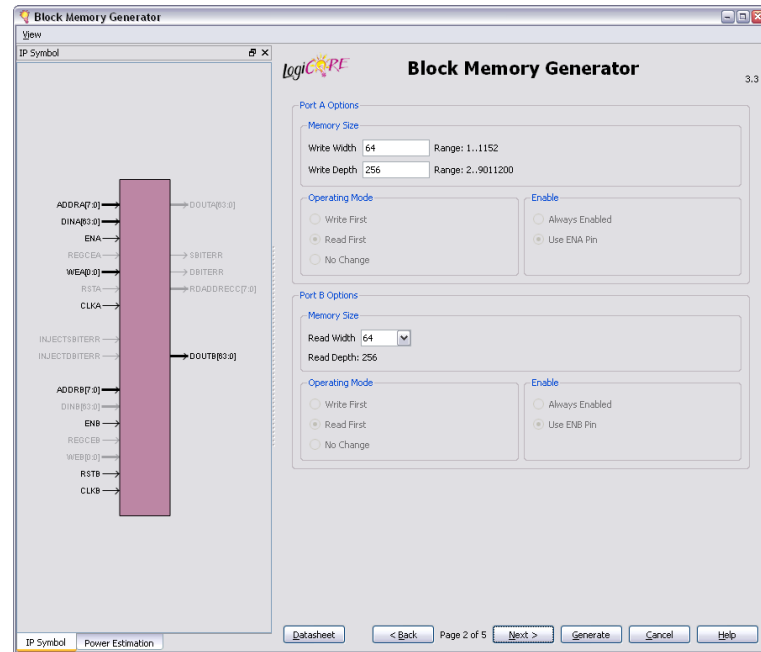


Figure 3.6: Block RAM configuration settings using Xilinx® CORE Generator

The chosen user specified block RAM module has the interface given in code example 3.7.

Code Example 3.7: Interface of the chosen dual port RAM

```

1 ENTITY blk_mem_gen_v3_3 IS
2   port (
3     clka : IN std_logic;           -- write clock
4     ena  : IN std_logic;           -- write enable
5     wea  : IN std_logic_VECTOR(0 downto 0); -- write enable (2)
6     addra : IN std_logic_VECTOR(7 downto 0); -- write address
7     dina  : IN std_logic_VECTOR(63 downto 0); -- write data
8     clkb : IN std_logic;           -- read clock
9     enb  : IN std_logic;           -- read enable
10    addrb : IN std_logic_VECTOR(7 downto 0); -- read address
11    doutb : OUT std_logic_VECTOR(63 downto 0); -- read data
12 END blk_mem_gen_v3_3;
```

The datasheet [30] indicates a proper functionality up to 400 MHz, depending on the actual size and settings adjusted to the module.

Very crucial to know is the timing behaviour of such a block RAM memory. Writing requests are carried out immediately: when setting the enable flag (**ena**) and write enable flag (**wea**) to '1', the data (**dina**) will be stored at the address (**addra**) within the same clock cycle of the write clock (**clka**). Contrarily, reading requests have a delayed response: when setting the read enable flag (**enb**) to '1', the data **doutb**

stored in address `addrb` will be ready to read out one clock cycle of the read clock (`clkb`) later.

3.3.6 DSP48E Units

To implement the functionality of the FIR filter, well suited FPGA components have to be used to carry out the basic operations of multiplication and summation and to achieve optimal resource utilisation as well as optimal timing behaviour. Those operations are incorporated in DSP elements.

Depending on the particular type of the FPGA used, most likely they will provide such DSP elements. An exemplary list of the DSP resources of the Xilinx® Virtex-5 family is given in table 3.3 [31]. In particular the type XC5VFX70T offers 128 DSP48E modules (48 bits output width, E for Extreme), supplying various configuration possibilities for numerous different applications. Relevant properties are in this case the maximum frequency and the width of the I/Os.

The block diagramme [31] of the DSP48E of the Virtex-5 is depicted in figure 3.7.

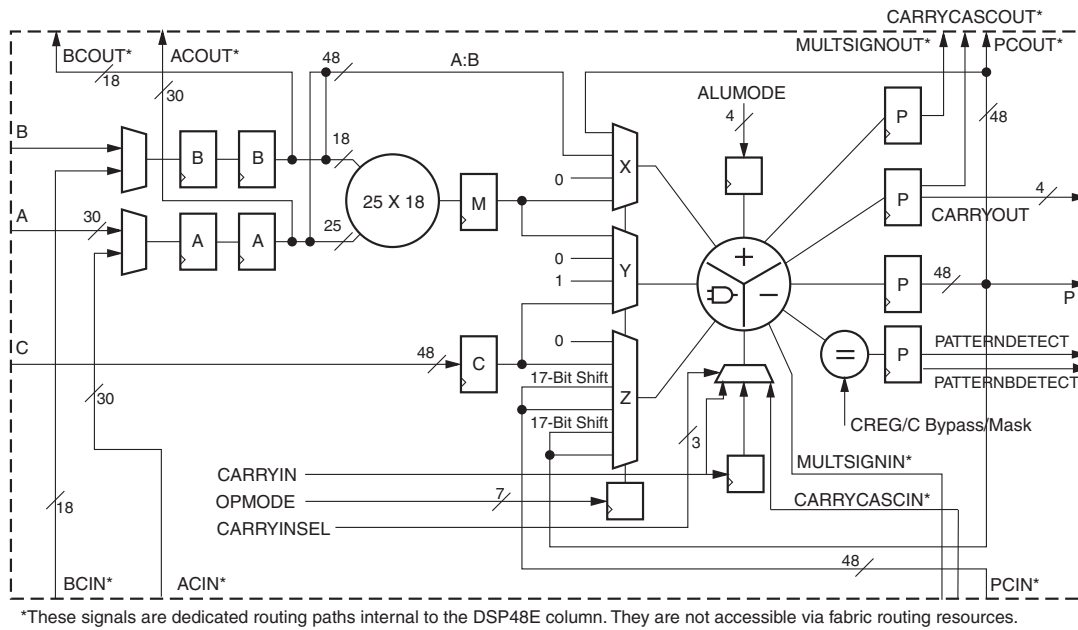


Figure 3.7: Block diagramme of a Virtex-5 DSP48E slice: Most relevant for this work are the 25×18 multiplier fed by the inputs **A** and **B** and the consecutive adder function (assigned by the **ALUMODE** selector) producing a 48 bit wide result **P** [31].

Regarding the data width of both factors *A* and *B* of the DSP48E unit in comparison to the expected data width of the ADC values of the FEBs, there are no constraints: with 18 and 25 bit width both ports largely exceed the required 12 bit and will thus be fully convenient.

Table 3.3: Number of DSP48E slices per Virtex-5 family member

Virtex-5 device	Number of DSP48E per device
XC5VLX20T	24
XC5VLX30	32
XC5VFX70T	128
XC5VFX100T	256
XC5VSX95T	640
XC5VSX240T	1056

The corresponding data sheet [31] also indicates a maximum operation frequency of 550 MHz for the operation when using optimal settings.

3.4 Ethernet

In order to be competitive with commercial hardware components on the receiving side, a well established networking standard for data transfer is used between the ROD and ROB modules. The ROD functionality is based on FPGA technology and the ROB is planned to be a PC or CPU based module. This section will thus briefly introduce the Ethernet standard [32].

Over time, Ethernet has evolved to the leading network technology for local area networks (LANs). Due to its constant development from the first patent protection in 1979 by Robert Metcalfe as “Multipoint Data Communication System with Collision Detection” (US patent number 4063220), Ethernet could successfully compete with similar networking technologies in the beginning of the 21st century. Today more than 90% of current LANs are using Ethernet and it is supposed to sustain its significant role in future. Gigabit Ethernet (GbE) with a transfer data rate of 1 Gb/s is very well established nowadays and commercial solutions for even 10 Gb/s are available on the market. From the beginning, the american institute for standardisation IEEE (Institute of Electrical and Electronical Engineers) supported the development and carried out the specifications and standardisations given in table 3.4.

Table 3.4: The different specifications of Ethernet by time.

standard	specification	year	data signaling rate	transfer medium
802.3	10BASE5	1983	10 MBit/s	thick coaxial cable
802.3a	10BASE2	1985	10 MBit/s	thin coaxial cable
802.3i	10BASE-T	1990	10 MBit/s	UTP cable
802.3u	100BASE-TX	1995	100 MBit/s	TP cable
	100BASE-FX	1995	100 MBit/s	optical fibre cable
802.3z	1000BASE-LX	1998	1000 MBit/s	optical fibre cable
802.3ab	1000BASE-T	1999	1000 MBit/s	TP cable
802.3ae	10GBASE-FX	2002	10 GBit/s	optical fibre cable
802.3an	10GBASE-T	2006	10 GBit/s	TP cable

Ethernet, as all networking standards, is based on the Open Systems Interconnection (OSI) model that describes the technical functionality of a network system. As can be seen in table 3.5, there are seven layers in the OSI model.

Table 3.5: The OSI model reference model.

#	layer	content
7	Application Layer	HTTP, FTP, ...
6	Presentation Layer	SSL, ...
5	Session Layer	
4	Transport Layer	User Datagram Protocol (UDP), TCP, ...
3	Network Layer	Internet Protocol (IP), ...
2	Data Link Layer	Ethernet, ARP, ...
1	Physical Layer	Ethernet, RS-232, ...

The Physical Layer defines the electrical, optical and mechanical connection to the transfer medium used. In the Data Link Layer the data transfer, regrouping of single bits to transferable units and access to the network is defined. Therefore it is also called Media Access Control (MAC) Layer.

It is an important fact, that, despite the different media and data transfer rates, the manner of networking is, generally speaking, equal. That means, that the data format, the structure of an Ethernet packet and the network access is unchanged. That provides a sustainable solution also for long term projects. Table 3.6 indicates the composition of the Data Link Layer, including a variable Cyclic Redundancy Check of 32 bits (CRC-32).

Table 3.6: Definition of an 802.3 MAC Frame (Data Link Layer).

OSI model	Definition	Data amount
Data Link Layer pre-header	Preamble Start-of-Frame Delimiter	15 nibbles of 0101 1 nibble of 1101
Data Link Layer header	MAC destination MAC source (optional) VLAN-tag Length	6 bytes 6 bytes (4 bytes) 2 bytes
Data Link Layer data	Payload (Data)	46 to 1500 bytes
Data Link Layer footer	CRC-32	4 bytes
Data Link Layer post-footer	Interframe gap	12 bytes

3.4.1 UDP/IP

As defined in the OSI model, Ethernet serves as a carrier for embedded protocols of the next higher layer. Out of the variety of different higher level protocols, for this work UDP/IP was chosen based on the following considerations:

- Internet Protocol, version 4 (IPv4):
 - 20 bytes header only
 - currently very well spread standard in PC networking
 - embedding layer for UDP
- UDP:
 - 8 bytes header only
 - communication is mainly mono-directional without any handshake, error correction or supplementary control flow
 - included checksum (over full packet) is optional

Another widely spread protocol is TCP/IP (Transmission Control Protocol). As the name suggests, TCP (situated in the Transport Layer) is a reliable stream delivery service that guarantees delivery of a data stream sent from one host to another without duplication or losing data and is thus optimised for accurate delivery rather than timely delivery. Consequently, TCP incorporates error detection, flow control capabilities and congestion control and is therefore rather complicated to implement and not suited for mainly mono-directional point-to-point communications.

Using the IP implies to choose between the currently well known IPv4 standard or its newer concept, Internet Protocol, version 6 (IPv6). The most important reason for the development of IPv6 has been the restricted address range of IPv4 ($2^{32} \approx 4 \times 10^9$ addresses) which is expected to be exhausted within the next few years. Although IPv6 contains a less complex header, its size, given in table 3.7, sums up to at least 40 bytes, which exceeds the one of IPv4.

Table 3.7: Structure of the IPv6 header in units of 4 bits.

1	Version	Traffic Class		Flow Label			
2	Payload Length			Next Header		Hop Limit	
3	Source IP Address						
4							
5							
6							
7	Destination IP Address						
8							
9							
10							

Table 3.8 explains the different components of the IPv6 header.

Table 3.8: Definition of the IPv6 header.

Version	4 bit version field: for IPv6 it is 6 (0110)
Traffic Class	8 bit priority indication
Flow Label	20 bit value for real time services
Payload Length	16 bit number defining the total number of bytes of the IP packet (without header); minimum is 20, maximum is $2^{16} - 1$
Next Header	8 bit field defining the protocol used in the data portion of the IP datagram
Hop Limit	8 bit Time To Live value: limiting the datagram life time to avoid circling it in the network
Source IP Address	128 bit IPv6 address of the source
Destination IP Address	128 bit IPv6 address of the destination

In contrast, the IPv4 header is made up of 20 bytes only, illustrated in table 3.9. The explanation of the corresponding component is given in table 3.10.

Table 3.9: Structure of the IPv4 header in units of 4 bits: the Header Checksum is mandatory.

1	Version	Header Length	TOS	Total Length
2	Identification			Flags
3	TTL		Protocol	Fragment Offset
4	optional Options			Header Checksum
5	Padding			
6	Source IP Address			
7	Destination IP Address			

Table 3.10: Definitions of the IPv4 header

Version	4 bit version field: for IPv4 it is 4 (0100)
Header Length	4 bit indicating the number of 32 bit words the header is made of; minimum is 5 (0101) = 20 bytes, maximum is 15 (1111) = 60 bytes
TOS	8 bit Type of Service: indicates routing services such as priority and path choice
Total Length	16 bit number defining the total number of bytes of the IP packet; minimum is 20, maximum is $2^{16} - 1$
Identification	16 bit number identifying fragments of an original IP datagram when fragmented
Flags	3 bit value handling fragmentation indication
Fragment Offset	13 bit value measuring the offset of a particular fragment relative to the beginning of the original non fragmented IP datagram
TTL	8 bit Time To Live: limiting the datagram life time to avoid circling it in the network
Protocol	8 bit field defining the protocol used in the data portion of the IP datagram
Header Checksum	16 bit checksum of the header
Source IP Address	32 bit IPv4 address of the source, e.g. 192.168.123.123
Destination IP Address	32 bit IPv4 address of the destination, e.g. 192.168.123.124 (7B7C in hex)

Due to the obvious more economical header size of IPv4, this protocol is chosen.

For a dedicated purpose, most of the IPv4 header components can be set statically, whereas the “Total Length” will differ with the length of the IP packet as well as the header checksum will depend on any changes of the IPv4 header.

As defined within RFC 1071 [33] the method to compute the checksum is as follows: *The checksum field is the 16 bit one’s complement of the one’s complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.*

An example calculation with specific header data (hex-values) used in the sample IPv4 packets should clarify that statement:

$$\begin{aligned}
 C511 + 006C + 1212 + C0A8 + 7B7B + C0A8 + 7B7C &= 34FD6 && \text{I: sum up} \\
 3 + 4FD6 &= 4FD9 && \text{II: complement} \\
 FFFF - 4FD9 &= B026 && \text{III: invert}
 \end{aligned}
 \tag{3.3}$$

As defined by the Protocol tag in the IPv4 header, UDP is finally embedded. Its minimalist header of 8 bytes is shown in table 3.11.

Table 3.11: Structure of the UDP header in units of 4 bits: the Header Checksum is optional.

1	Source Port				Destination Port		
2	Length				Header Checksum		

Table 3.12 explains the single components of the UDP header.

Table 3.12: Definition of the UDP header.

Source port	16 bit address of the source
Destination port	16 bit address of the destination
Length	16 bit number defining the total number of bytes of the UDP packet, including the header; minimum is 8, maximum is $2^{16} - 1$
Header Checksum	16 bit checksum of data and header (optional)

The big advantages of UDP is, as pointed out before, its tiny header and the fact that the checksum is optional and can be left zero. No further calculation is required except the setting of the length accordingly, following the simple formula

$$\text{UDP-length} = \text{IP-length} - 20. \tag{3.4}$$

3.5 Auxiliary Tools

Here, a short overview is given on the supplementary tools used for the development process.

3.5.1 ISE: A Xilinx Design Suite

In order to develop code for the FPGA, the Xilinx[®] design suite ISE is used. This design suite contains the “ISE Project Navigator” where the VHDL code is written and managed in project files, synthesised and finally compiled to a bit file. Additionally to code generation, simulation and verification tools are provided by this design suite, namely “ISE CORE Generator”, “ISE ISIM” and “ISE ChipScope Pro Analyzer” and more.

The development process is done with the ISE Project Navigator: The idea of an algorithm has to be coded in VHDL and passed through different stages of evolution. First, there is a syntax and compatibility check (so called “Synthesize - XST” module) resolving the VHDL code into (programmable) logic device chains creating three kinds of output: Errors, most times originating from invalid VHDL statements that have to be solved; warnings, indicating valid but ambiguous assignments or assignments of signals never used or vice versa; and information messages, pointing out non critical supplementary information. Information messages and warnings will not abort the compilation process whereas the compilation will stop on errors. The following process “Implement Design” consists of three subprocesses, called “Translate”, “Map” and “Place & Route”, with the task to generate the target scheme for the FPGA, each with the same three types of output. Finally, the bit file to programme the FPGA is generated by the “Generate Programming File” module. The time needed for the entire compilation process is, as usual for any compilation, dependent on the project itself. The typical time spent on compilation ranged from one minute up to half an hour.

Once the bit file is generated, it can actually be transmitted to the FPGA. For this purpose there is the stand alone application called “ISE Impact”. Here, one specifies the bit file to assign as well as the target device, namely Virtex-5, XC5VFX70T. After a short transfer period, the implementation then instantaneously starts its operation.

3.5.2 Wireshark

Wireshark plays a major key role in this work as it is used to analyse the Ethernet output of the FPGA test board and thus to verify all the functionality of the implementations.

Wireshark itself claims to be the world’s most popular network protocol analyser. It is Open Source Software released under the GNU General Public License.

Using Wireshark provides very complex network analysing capabilities. It is possible to capture the complete Ethernet traffic of a selected network card, supporting all known protocols, as shown in figure 3.8.

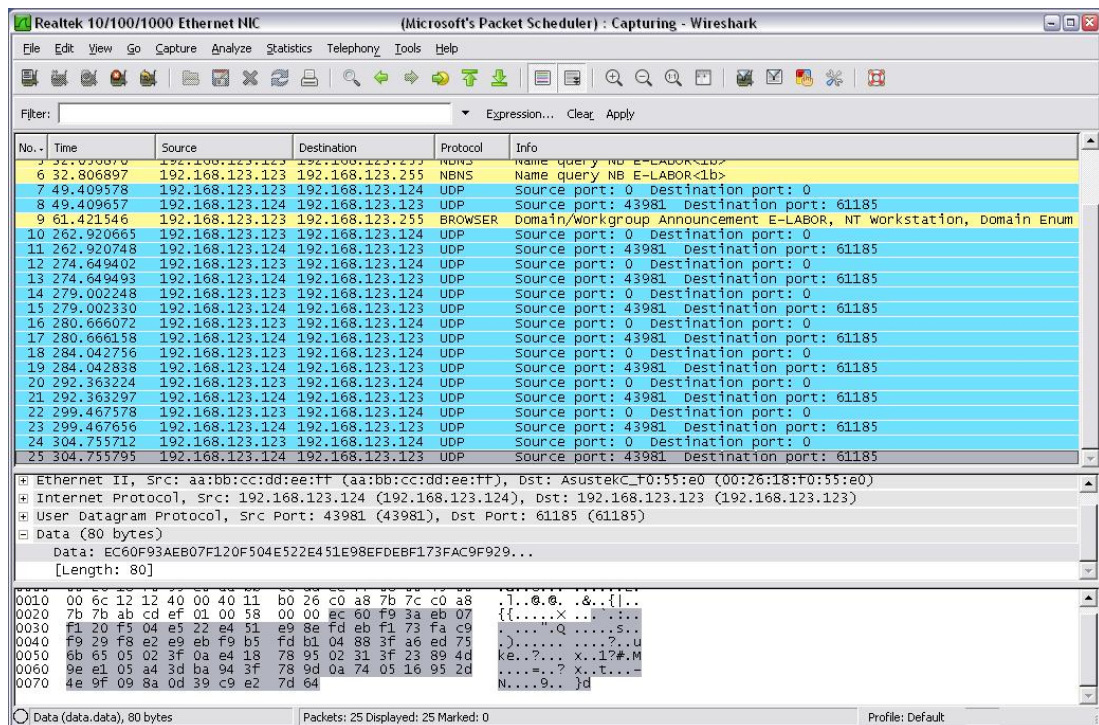


Figure 3.8: Wireshark in capturing mode: the full Ethernet traffic of a chosen Ethernet interface is logged.

3.5.3 Packet Builder

The Colasoft Packet Builder is the second tool required to verify the functionality of the design. It is also a free networking software.

It allows the user to build up Ethernet packets byte by byte giving him the full control. The Packet Builder allows to configure well formatted UDP packets as well as packets with incorrect IP checksum or incorrect length indications, for example, or any other user intended manipulation.

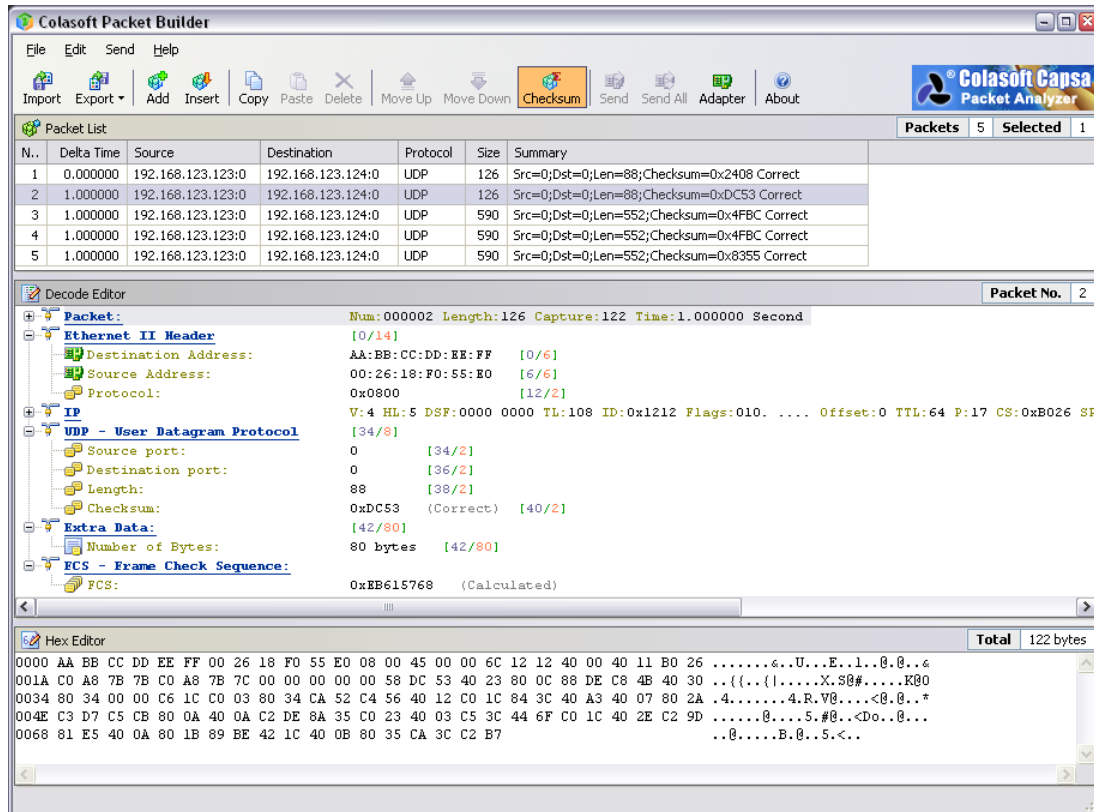


Figure 3.9: The Colasoft Packet Builder is used to create sample UDP packets to be sent to the FPGA. Packets for the calibration (of the size of 590 bytes) as well as energy sample packets (of the size of 126 bytes) are prepared.

The Packet Builder is used to define the UDP packet of energy samples for the FPGA as well as the configuration data packet containing the filter coefficients a_i as well as the gain and channel dependent pedestal Ped (see formula 2.3). In figure 3.9 a UDP packet containing energy samples is highlighted.

4 Implementation

This chapter will guide through the various developments that have been done in this work. The first part will introduce the implementation of the 1 GbE link as a prototype for the communication between the ROD and ROB modules at the end of the upgraded, hardware based readout chain of the LAr calorimeter. The second part will cover the development of a FIR filter prototype that is to be applied at the very beginning of the upgraded readout chain in the ROD FPGAs.

4.1 Implementation of UDP/IP

A hardware implementation of Ethernet (Physical Layer) for a Virtex-5 FPGA is available directly from Xilinx® for up to the 1000BASE-T specification. The Xilinx® Tri-Mode Ethernet CORE combines the three operating modes of 10BASE-T, 100BASE-TX and 1000BASE-T (1 GbE) (compare table 3.4) - a module that can be implemented by the CORE Generator.

Out of the full packet size (see table 3.6), only the header and data part of the Data Link Layer is of interest here. That is the data provided by the TriMode Ethernet CORE to the user via the LocalLink interface. When sending an Ethernet packet via this module, the user only has to provide the header and data bytes of the Data Link Layer to the module. The evaluation and calculation of the CRC-32 (footer of the Data Link Layer) is autonomously dealt, there are `good_frame` and `bad_frame` signals provided to inform about the CRC-32 accordingly.

An example design using this TriMode Ethernet CORE is provided by Xilinx®. It receives an Ethernet packet byte by byte and loops it back to the recipient by simply exchanging the source and destination MAC addresses in the Address Swap Module. All the other data is simply looped through as shown in figure 4.1 (only one interface, EMAC0, is used).

This example design was modified to implement two new separate modules: a receiving module that takes data whenever incoming and stores them to the RAM, and a sending module with inverse task on user demand.

A general remark has to be given at this place: The suffix “_n” of a signal name is used to indicate its logic inversion between high (1) and low (0). For example in figure 4.2 the signal `rx_ll_sof_in_n` to indicate the start of a frame drops at the start of the frame whereas the actual signal `rx_ll_sof_in` rises at that moment. It is sometimes simply easier to deal with inverted logic during a process and to negate the signal at the end and is thus convention.

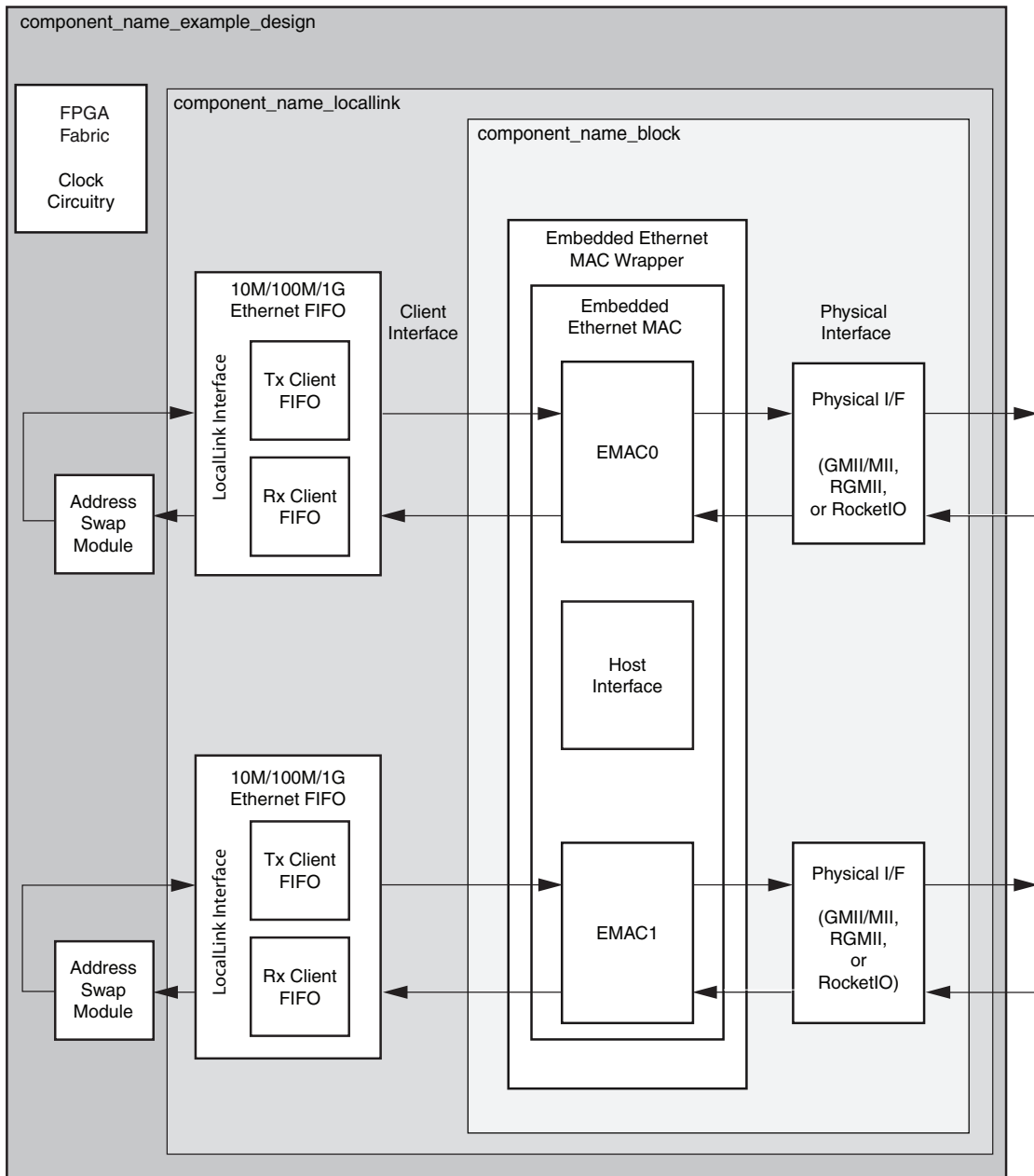


Figure 4.1: The TriMode Ethernet CORE example design [34] includes a Host Interface and an Embedded Ethernet MAC with two separate EMACs. The EMACs make the transition between the LocalLink input and output client FIFOs of the Data Link Layer and the Physical Interface of the Physical Layer. The Physical Interface can be configured to used different interfaces. The GMII is chosen to be conform with 1 GbE. The Address Swap Module on the Data Link Layer side was replaced by the UDP transmitting and receiving modules. The Host Interface was not used.

The user interface for the LocalLink (ll) (including control signals for transmitting (**tx**) and receiving (**rx**) or combined (**rtx**)) is specified in code example 4.1.

Code Example 4.1: Port definition of the LocalLink for the Ethernet interface

```

1  port (
2    ...
3    rtx_ll_clock      : in  std_logic; -- Input CLK from TRIMAC Receiver
4    rtx_ll_reset      : in  std_logic; -- Synchronous reset signal
5    rx_ll_data_in     : in  std_logic_vector(7 downto 0); -- Input data
6    rx_ll_sof_in_n    : in  std_logic; -- Input start of frame
7    rx_ll_eof_in_n    : in  std_logic; -- Input end of frame
8    rx_ll_src_rdy_in_n : in  std_logic; -- Input source ready
9    tx_ll_data_out     : out std_logic_vector(7 downto 0); -- Output data
10   tx_ll_sof_out_n    : out std_logic; -- Output start of frame
11   tx_ll_eof_out_n    : out std_logic; -- Output end of frame
12   tx_ll_src_rdy_out_n : out std_logic; -- Output source ready
13   tx_ll_dst_rdy_in_n : in  std_logic; -- Input destination ready
14   ...
15 );

```

The timing specification is shown in figure 4.2.

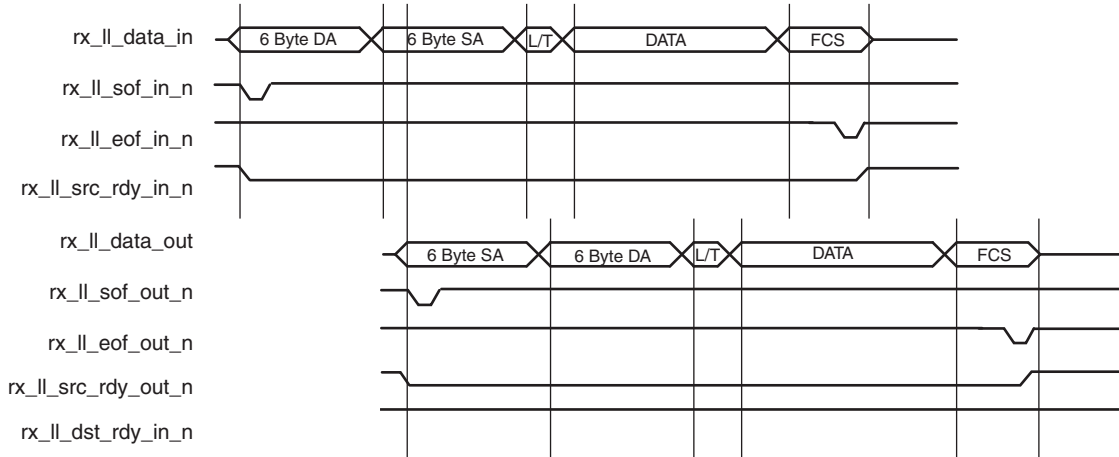


Figure 4.2: Timing of the control signals of the MAC wrapper [34].

Together with the eight data bits three or four additional control bits have to be set or evaluated according to the validity of the data and the readiness of the interface.

To meet the 1 GbE standard, a 125 MHz clock is required (**rtx_ll_clock**) which is provided via a DCM/PLL module from a 100 MHz on-board base clock.

For the direct communication with the LocalLink Interface (figure 4.1) a UDP management module is introduced incorporating the receiving (**UDP_rx_module**) and transmitting (**UDP_tx_module**) submodule. Their task is to set the eight data bits and the control bits according to the interface definition as depicted in figure 4.2.

4.1.1 UDP Transmitting Module

The task of the transmitting submodule is to read data from the RAM initiated by the stimulus `send_frame_flag` and to create the Ethernet data and control stream from it. Code example 4.2 shows the interface of the UDP transmitting module.

Code Example 4.2: Interface definition of the UDP transmitting module

```

1 entity UDP_tx_module is
2   port (
3     -- Ethernet clocking
4     clk          : in  std_logic; -- Input CLK from TRIMAC Reciever
5     rst          : in  std_logic; -- Synchronous reset signal
6
7     -- Ethernet data + controls
8     tx_data_out   : out std_logic_vector(7 downto 0); -- Modified output
9       data
10    tx_sof_out_n   : out std_logic; -- Output start of frame
11    tx_eof_out_n   : out std_logic; -- Output end of frame
12    tx_src_rdy_out_n : out std_logic; -- Output source ready
13    tx_dst_rdy_in_n  : in  std_logic; -- Input destination ready -- check
14
15    -- Sending counter for outer module
16    tx_counter      : out std_logic_vector(10 downto 0);
17
18    -- external stimuli
19    send_frame_flag : in  std_logic; -- start frame signal
20
21    -- RAM read requests from UDP-module:
22    r_flag_enet     : out std_logic;
23    r_addr_enet     : out std_logic_vector(7 downto 0);
24    r_fine_enet     : in  std_logic;
25    r_data          : in  std_logic_vector(63 downto 0) -- out read data
26  );
27 end UDP_tx_module;
```

The length of the data packet itself is stored in the RAM, as well as (most of) the rest of the UDP/IP-header. When initiated, a counter (`tx_counter`) is enumerated indicating the number of the byte being transmitted. All control signals are set properly while the packet is sent. The signal `tx_dst_rdy_in_n` indicating the readiness of the LocalLink is respected to prevent data overflow. The IP checksum value is calculated as shown above, in parallel to the running transmission and inserted to the stream when required.

To gain time for the IP checksum calculation, an eight-staged shift register is used to store eight bytes read from the RAM temporarily. That not only reduces the RAM read requests to 12.5%, but also allows a parallely running second 4-staged 16 bit shift register (`CRC_sr_content`) looping through the data considered for the IP checksum.

The checksum implementation (equation 3.3) in VHDL was realised using a 16 bit fulladder module with carry-in and carry-out bit and a clocked process. The inversion is finally done simply by negation of the inverse checksum `IP_CRC_n` when put into the header stream. In code example 4.3 the idea of this implementation is given.

Code Example 4.3: Implementation of the IP CRC in the UDP transmitting module

```

1  crc_calculator: FullAdder_16bit -- 16 bit fulladder:
2  port map (
3      a_in => adder_in_a,          -- a + b (both 16 bits)
4      b_in => adder_in_b,
5      c_in => adder_in_c,          -- + c (1 bit)
6      s_out => adder_out_s,        -- = s (16 bits)
7      c_out => adder_out_c        -- + overflow bit c
8  );
9
10 with tx_count select
11     adder_in_a <=
12         x"00_00" when 0,          -- reset (value 0) in the beginning
13         IP_CRC_init_value when 1, -- set to predefined init value
14         IP_CRC_n when others;    -- take last calculated sum
15
16 with tx_count select
17     adder_in_c <=
18         '0' when 1,              -- reset in the beginning
19         carry when others;       -- take last overflow bit
20
21 manage_IP_CRC_from_tx_count : process(clk) is -- on every clk cycle
22 begin
23     if rising_edge(clk) then
24         if (rst = '1') then      -- always take into
25             IP_CRC_n <= (others => '0'); -- account the reset
26             adder_in_b <= (others => '0'); -- of all used signals
27             carry <= '0';
28         elsif tx_enable = '1' then -- if data goes out
29             carry <= adder_out_c;    -- save last overflow bit
30             IP_CRC_n <= adder_out_s; -- save sum
31
32             adder_in_b <= crc_sr_content(3); -- save actual data
33         else
34             carry <= '0';          -- else reset
35             adder_in_b <= (others => '0'); -- input signals
36         end if;
37     end if;
38 end process;

```

4.1.2 UDP Receiving Module

The UDP receiving module serves to store incoming data packets in the RAM. It is separated between two kinds of packets: calibration packets, tagged by `calib_signature`, and calculation packets, tagged by `mcfilt_signature`. Both signatures are defined in the global parameter file `my_definitions.vhd` given in the appendix A.3. Incoming packets that do not carry one of these signatures are ignored by the module. The full interface is shown in code example 4.4.

Code Example 4.4: Interface definition of the UDP receiving module

```

1 entity UDP_rx_module is
2   port (
3     -- Ethernet clocking
4     clk          : in  std_logic; -- Input CLK from TRIMAC Reciever
5     rst          : in  std_logic; -- Synchronous reset signal
6
7     -- Ethernet data + controls
8     rx_data_in   : in  std_logic_vector(7 downto 0); -- Input data
9     rx_sof_in_n  : in  std_logic; -- Input start of frame
10    rx_eof_in_n   : in  std_logic; -- Input end of frame
11    rx_src_rdy_in_n : in  std_logic; -- Input source ready
12
13    -- Receiving counter for outer module
14    rx_counter    : out std_logic_vector(10 downto 0);
15
16    -- communication UDP module <=> LCD module
17    capture_src_flag : out std_logic; -- SRC MAC @ was found
18    capture_dst_flag : out std_logic; -- DST MAC @ was found
19    mac_readout_flag : in  std_logic; -- MACs have been read out
20
21    -- communication UDP module <=> Calibration module
22    calib_start      : out std_logic;
23    calib_enable     : in  std_logic;
24
25    -- communication UDP module <=> McFilt module
26    calc_start       : out std_logic;
27    calc_enable      : in  std_logic;
28
29    -- RAM write requests from UDP-module:
30    w_flag_enet      : out std_logic;
31    w_addr_enet      : out std_logic_vector(7 downto 0);
32    w_data_enet      : out std_logic_vector(63 downto 0);
33    w_fine_enet      : in  std_logic;
34  );
end UDP_rx_module;

```

The module analyses the start of frame signal `rx_sof_in_n` and starts a counter `rx_counter` on an incoming packet. Based on this counter, the packet header is analysed on UDP/IP, first. The packet size is retrieved from the dedicated IP header information as well as the packet type is determined from the IP identification flag (calibration data or data samples). That information could also be defined to be the first byte of the UDP data part, for example, but it is aimed not to generate any artificial overhead.

As done for the UDP transmitting module, an eight-staged shift register is applied. This shift register offers the advantage that firstly, there is time left for analysing the incoming packet header storing it directly or in a modified way to the RAM and secondly, that writing takes place only once in eight clock cycles. The real data part (UDP data) is treated equally but without further analysis.

An auxiliary feature of the UDP receiving module is the indication of the reception of a new packet by the signals `capture_src_flag` and `capture_dst_flag` right at the reception of the header which allow to illustrate network activity e.g. by the LCD panel.

The calibration itself is a slow process (see section 4.4) in comparison to the reception speed of the calibration data. When receiving a valid calibration packet the UDP receiving module automatically initiates the calibration by setting the signal `calib_start` right after the reception of the first eight calibration bytes.

In contrast, the FIR filter calculation is a very fast real time process (see section 4.3), faster than the reception of the calculation data in the test design itself. Here, the calculation is initiated by the UDP receiving module only after the complete reception of a valid data packet, indicated by setting the signal `calc_start`.

In the presented design, once a calibration or a calculation is running (indicated by the signals `calib_enable` and `calc_enable`), no further incoming packets are accepted as the currently occupied RAM would be overwritten.

At this point it has to be remarked explicitly that this UDP receiving module in this version is not suitable for the final target design of the ROD upgrade. The data samples will not be transmitted by such an interface. The calibration data might be transmitted in such a way. The solution implemented here was chosen to allow data transfer into the target FPGA for testing the FIR filter and to develop a first prototype of a high speed Ethernet interface. The final ROD management interface might be based on such a 1 GbE interface but will have to include much more functionality.

4.1.3 Resource Utilisation

Table 4.1 indicates the resource utilisation for the UDP transmitting and receiving modules. It is not claimed to have achieved an optimal solution.

Table 4.1: Resource utilisation of the UDP transmitting and receiving modules.
The percentages refer to the total available resources (see table 3.2).

resource	UDP receiving module	UDP transmitting module
Slice registers	214 (0.48%)	253 (0.56%)
Slice LUTs (total)	248 (0.55%)	418 (0.93%)
Total occupied slices	125 (1.12%)	175 (1.56%)

4.2 Design Overview

The realisation of the complete task to implement a FIR filter in a Gigabit Ethernet test environment has been split into separable function blocks. Figure 4.3 illustrates those different pieces. It shall be reminded, that this is the implemented design for the embedded FIR filter prototype of this work which does not represent the final design for the upgraded ROD.

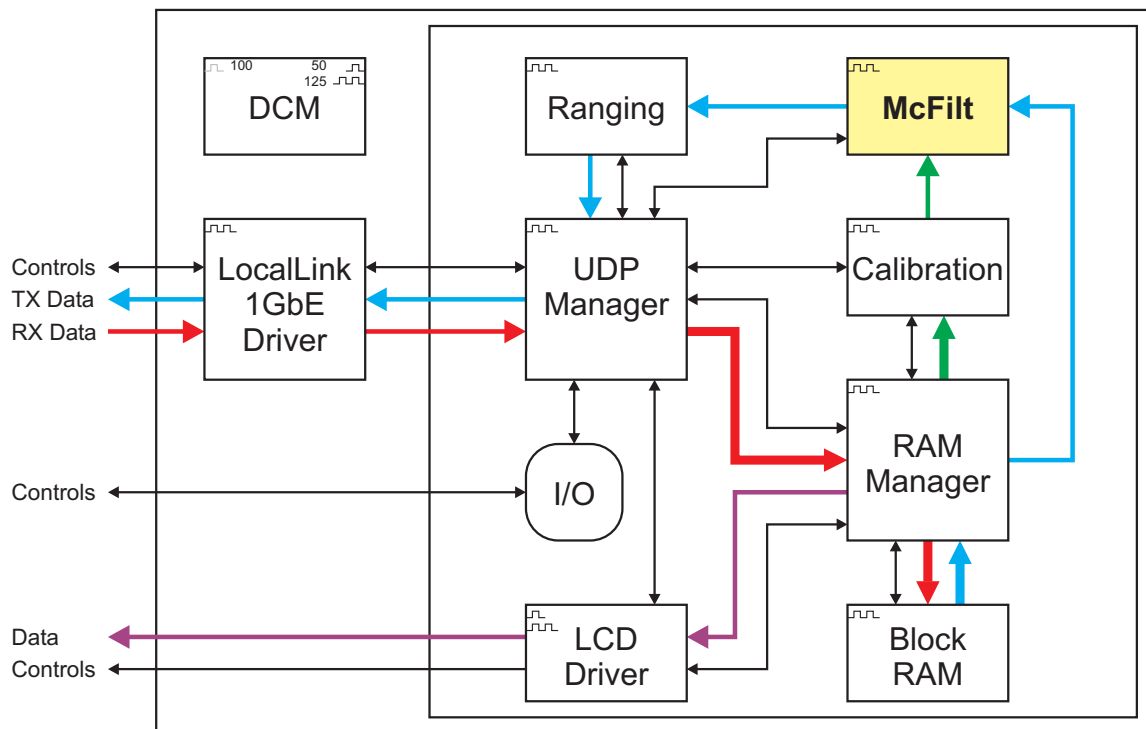


Figure 4.3: The first layout of the FPGA filter implementation using the LocalLink is divided into functional parts. Incoming data (red data path) is first stored in the block RAM. In the case of being configuration data, the calibration is started (green path). Being sample data (blue path), the filtering is launched looping the data through the McFilt unit and the output data encoding module (Ranging) to send them directly to the external interface. Activity is monitored by an LCD panel (lilac path). Black paths represent control signals. The different widths of the paths represent the different bus widths used between the modules.

As presented in the previous sections, the application specific filter layout is embedded in the Xilinx[®]-provided example design using the LocalLink 1 GbE Driver (outer frame) to handle the data transfer with external devices such as a PC. All supplementary and specific modules to perform the filtering, the actual UDP/IP driver, etc. have been implemented in an inner module (inner frame) to clearly distinguish given sources from own contributions.

The central module in the view of managing data flow is the UDP Manager, with its receiving and transmitting submodules (see previous sections 4.1.2 and 4.1.1). The RAM Manager (section 4.5) controls the numerous read and write requests of the LCD Driver, the UDP Manager, the Calibration and of the McFilt module.

The next sections will present the development of the McFilt (section 4.3) and the Calibration (section 4.4).

4.3 The Filter Algorithm

The implementation and optimisation of the FIR filter is the main task in the work. Due to the continuous development of that core unit, the main stages will be discussed. All stages of development make use of the DSP units provided by the FPGA.

4.3.1 The First Filter Design

The first attempt to implement the five-staged FIR filter was done using the Xilinx® CORE Generator. It provided an easy solution for a multiplier using one DSP48E unit per multiplication. The module was configured to multiply a 12 bit unsigned data sample with a 14 bit signed coefficient. Five of such generated modules were implemented in a higher level calculation module. All of them were connected with the data sample port to the data sample input bus S_i of the calculation module. The port for the coefficient was connected to a static LUT of the channel and gain dependent filter coefficients a_1 to a_5 . Each multiplier unit thus provides a 26 bit wide signed result P_i :

$$P_i \equiv a_i \times S_i. \quad (4.1)$$

To generate the sliding window, the outputs of the second to fifth multiplier unit were delayed for two to five clock cycles using one- to four-staged shift registers. The summation was finally performed using a six input ports wide 16 bit fulladder. It has to be remarked, that the sixth adder input was driven by a global pedestal Ped , which was also selected gain and channel dependently, but only with respect to the first data sample, which is not exactly the realisation of the formula 2.3.

Nevertheless, the general idea of subtracting the pedestal only once remains true for designs where the uniquely entering energy is pedestal corrected directly in the beginning and provided for further processing (see next sections). That is the main difference between formula 2.3 and its realisation, so the pedestal values Ped_i actually simplify to only one pedestal value Ped which will be used in the following.

Despite of simply forwarding the result P_i to the fulladder submodule, the idea described in point 2 to introduce gain dependent pre-scaling factors was realised. The for testing purpose arbitrarily selected first powers of 2, namely the factors 2, 4 and 8, were chosen for the three increasing gains. The fourth possible gain value of 00 leads to a vanishing result of

$$\tilde{P}_i = 0 \times P_i = 0. \quad (4.2)$$

Table 4.2 summarises the effect of the gain dependent pre-scaling on the calculation of the first filter design.

The data output encoding as described in section 2.5.5 has not been implemented at that stage of development.

The filter design layout is shown in figure 4.4.

Table 4.2: The gain influence in the first filter design: a gain-dependent pre-scaling is applied.

Gain value	Influence
00	$\tilde{P}_i = 0$
01	$\tilde{P}_i = 2 \times P_i$
10	$\tilde{P}_i = 4 \times P_i$
11	$\tilde{P}_i = 8 \times P_i$

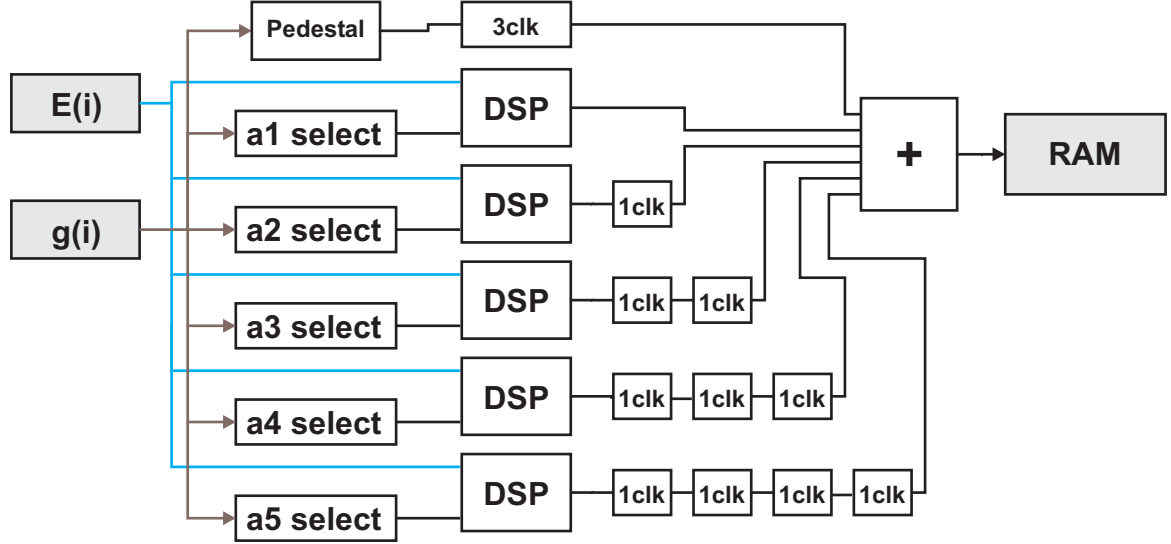


Figure 4.4: Data flow of the first calculation design: The incoming 8 bit data bus is split into two gain bits g_i and a 6 bit data sample S_i . Driven by the gain, the five filter coefficients a_i as well as a global pedestal Ped are selected using a static LUT. A depth-dependent delay is introduced to implement the sliding window.

The surrounding Ethernet interface provides an 8 bit wide data bus at 125 MHz. So the full calculation was using the lowest eight bits of the input energies, restrictively, and only the lowest eight bits of the result could be validated. This reduced the range of the data samples to a 6 bit value as the gain selection using two bits was properly implemented. As shown in figure 4.4, the calculation was put in between a preliminary UDP receiving module and the storage to the block RAM. By manually pressing a push button on the test board that set the signal `send_frame_flag`, the RAM content was sent to a PC by a preliminary UDP transmitting module.

Due to the design in a processing chain, the calculation was constrained to one channel. The same clock frequency was used throughout all stages. That would consequently lead to a restriction of the filter to the operational frequency of the LHC of 40 MHz, while the multiplier DSP units themselves are designed for much higher operation clock speed in the range of one order of magnitude.

Tests with a standard sample of 18 energies proved the functionality of the design as desired. The full design worked properly at the 1 GbE base frequency of 125 MHz. The samples were chosen to cover all possible circumstances of the filter: all three

valid gains were tested, and the sample values and filter parameters (a_i and Ped) were chosen to have E enter the positive and negative domain.

As can be seen in figure 4.4, in total ten stages of shift registers are required in this design for one channel. The resource utilisation of this filter design is given in table 4.3.

Table 4.3: Resource utilisation of the first filter design. The percentages refer to the total available resources (see table 3.2).

Resource	Amount
Slice registers	68 (0.15%)
Slice LUTs	240 (0.54%)
Total occupied slices	132 (1.18%)
DSP48E	5 (3.91%)

4.3.2 The Second Filter Design

To make profit of the high performance of the DSP units and to minimise resource utilisation, the shift registers to implement the sequential summation are to be rearranged in a more optimised way. A method to achieve higher internal DSP frequencies by constant outer data input frequency has to be found. Figure 4.5 illustrates two stages of improvements in this direction: First of all, the registers in front of the DSP units are replaced, economising 60% of them. Additionally, the bit width of those registers is reduced from 26 bits (corresponding to the output width of the DSPs) to only 15 bits (13 bits pedestal corrected sample data plus 2 bits gain). Secondly, multiplexing is introduced to increase the internal DSP operation clock frequencies to multiples of the LHC clock.

The design was coping with four channels. It was also modified to deal with the full bit width of 12 bits per data sample instead of only 6 bits in comparison to the first design. The output bus was widened to 16 instead of 8 bits. To achieve the doubled bit width for the correct calculation without modifying the general prototype design layout, it was necessary to slow down the filter by a factor of two: incoming data is received byte-wise at 125 MHz and two bytes are required to build up one data sample, thus only every second clock cycle the calculation is carried out by setting the enable signal of the filter correspondingly.

To perform the filtering, a separate calculation module was introduced.

Code Example 4.5: Interface definition of the first calculation module

```

1 entity sum_calculation is
2   port (
3     clk           : in  std_logic;
4     calc_enable   : in  std_logic;
5     S             : in  E_by_channels;
6     Esum          : out E_by_channels;
7     channel       : in  integer range 0 to hig_channel
8   );
9 end sum_calculation;
```

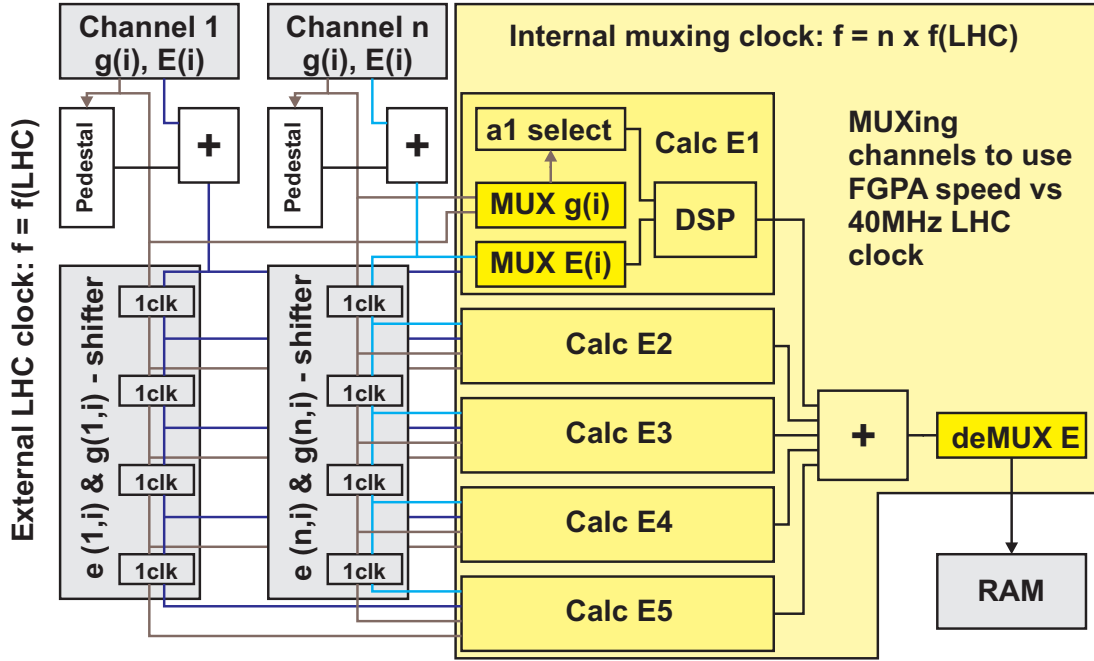


Figure 4.5: Data flow of the second calculation design: The inner multiple DSP core unit operates at n -times the outer clock frequency by multiplexing n channels. The incoming data bus is split into two gain bits g_i and a 12 bit data sample S_i . The gain and channel dependent pedestal Ped is removed directly before entering further stages. To implement the sliding window, a shift register is introduced in front of the inner multiple DSP core unit. The filter coefficients a_i are selected from a constant LUT and provided for each DSP unit.

In the port definition of this module given in code example 4.5, the signal `clk` is the 125 MHz 1 GbE base frequency and the control bit to enable the calculation, `calc_enable`, is set by the outer module to be '1' every second clock cycle. The total number of four channels is represented by `hig_channel + 1`, a global design parameter. The counter `channel` circularly enumerates those channels to address the slot of `S`, the vector of incoming data samples, to be calculated by the module. The filter result is stored into the corresponding slot of `Esum`, the vector of outgoing energies.

The vector of incoming data samples `S` was filled up with new data samples at a certain frequency, e.g. 40 MHz, to perform the calculation at a frequency of `hig_channel + 1` times that frequency and to sequentially provide the filter result `Esum` to the outer module.

In deviation of the presented filter layout of figure 4.5, the selection of filter coefficients a_i was not done individually for every filter stage directly in front of the DSP units, but conjointly at the very beginning when selecting the pedestal Ped . The filter coefficients thus also had to be stored in shift registers until they were processed by the corresponding DSP component.

Furthermore, it was required to loop through the two gain bits as well as the pre-

scaling and validity check was implemented (see table 4.2).

Like in the first filter design, the data output encoding had not been implemented. Tests including all circumstances had shown a proper functionality of the implementation. A verification C programme had been written to compare the output of the FIR filter implementation to the expected result.

The analysis of this design revealed a latency for the calculation unit of six clock cycles from the gathering of the first data sample S_i to the result $\mathbf{Esum}$ (actually being twelve clock cycles at a frequency of 125 MHz as the calculation is enabled every second clock cycle, respectively).

As this filter solution included the multiplexing of four channels, and thus provided the required storage devices (registers, etc.), the resource utilisation as indicated in table 4.4 was enlarged approximately by the factor of four in comparison to the previous design.

Table 4.4: Resource utilisation of the second filter design in comparison to the first one: an increase of an approximative factor of four can be noticed by the implementation of a filter coping with four channels. The percentages refer to the total available resources (see table 3.2).

Resource	1st	2nd
Slice registers	68 (0.15%)	726 (1.62%)
Slice LUTs	240 (0.54%)	476 (1.06%)
Total occupied slices	132 (1.18%)	376 (3.36%)
DSP48E	5 (3.91%)	5 (3.91%)

4.3.3 The Third Filter Design

To further decrease the latency of the calculation module, several changes became necessary. The most dramatic modification was to implement the DSP processing units not using the CORE Generator provided module, but to implement them by hand. This offered the supplementary positive effect that the in-built adder could be used to replace the external one. The DSP slice (figure 3.7) was adjusted to work as a combination of a multiplier and an adder, as shown in figure 4.6.

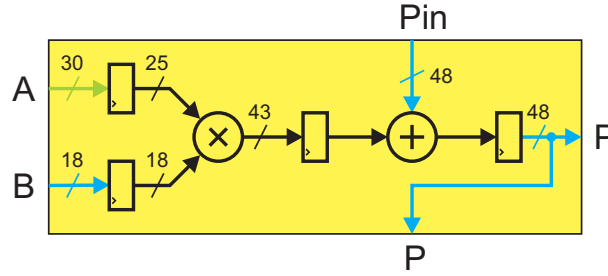


Figure 4.6: A DSP48E component in the configuration of a multiply-adder, a combination of a multiplier (A, B) and an adder ($A \times B, P_{in}$), which promises fast calculations up to 550 MHz [31]. The multiplier supports a width of 25 bits for input A and 18 bits for input B to provide a 43 bit wide product. The unit is equipped with a side input P_{in} for the internal 48 bit adder. The adder sum is provided at two places: On the right side for external use, and on the lower side to interconnect different DSP units directly. The scheme clearly illustrates the latency of three clock cycles caused by the three registers. Pipelined processing is supported.

Five DSP components are interconnected directly to build up the five-staged FIR filter. Interleaved design allows the calculation of several channels at 125 MHz. As a special feature and update of the previous designs, the Multi channel Filter (McFilt) provides the possibility to reconfigure the filter parameters without any negative influence on the latency. Figure 4.7 shows a general layout of the McFilt.

A latency of only three clock cycles is achieved, meaning that no overhead is produced by the surrounding environment of the DSP48E slices and leading to a total latency of

$$\Delta t_{E-S, 125 \text{ MHz}} = 3 \times (125 \text{ MHz})^{-1} = 24 \text{ ns.} \quad (4.3)$$

The indicated latency includes the encoding of the output data E of the McFilt described in section 2.5.5. In the regime of a 125 MHz clocked design it is still possible to maintain an quasi-instantaneous response of the encoding module. Its resource utilisation is given in table 4.10, the resource utilisation of the full filter including the encoding module is included in table 4.9.

The functionality is tested using the Xilinx® simulator ISIM (see section 3.5.1). An example is given in figure A.1. The hardware implementation of this design was also subject to tight tests, again including all possible circumstances.

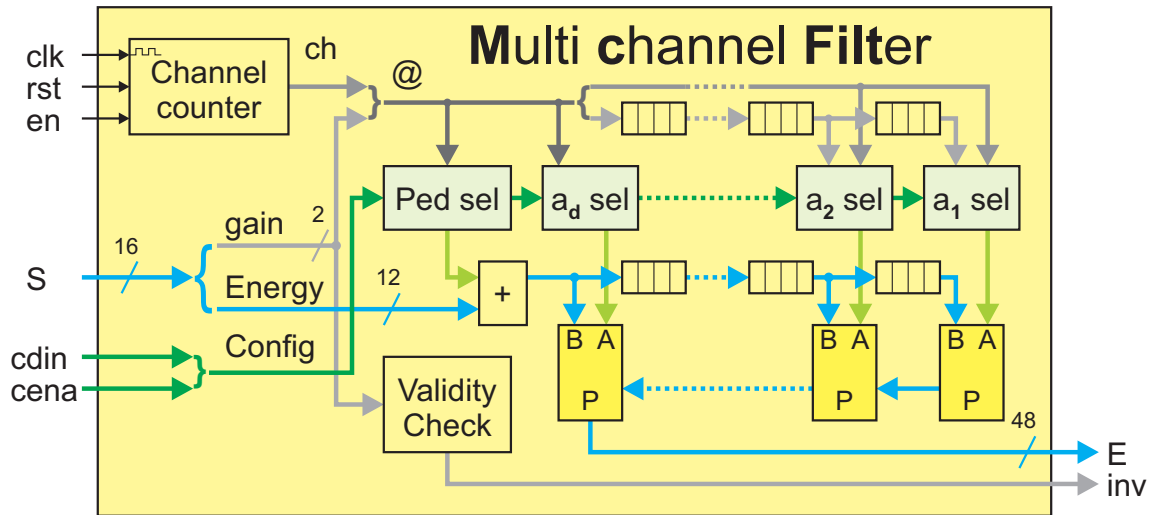


Figure 4.7: The scheme of the Multi channel Filter (McFilt) unit of filter depth d operational at 125 MHz is shown. It incorporates d DSP48E core processing units to perform the optimal filtering following the blue data path of incoming data samples S_i . The gain is separated from the samples and combined with a channel counter (grey data path) to make the gain and channel dependent selection of the filter parameters a_1 to a_d and Ped . The McFilt unit is driven by a clock signal clk , controlled by the enable signal en and a reset rst . Interleaved design using channel-1 shift registers from stage to stage allows the filtering of several channels quasi parallelly. The green data path illustrates the embedded filter parameter reconfiguration requiring a data ($cdin$) and an enable ($cena$) port.

4.4 Filter Calibration

Running the detector at variable settings, such as luminosity or bunch spacing, in a tight time frame requires a well suited calibration of the filter parameters a_i and Ped . As the filter parameters are stored in the FPGA, two solutions are considerable:

1. Offline recalibration with a complete FPGA firmware update. Here, the filter parameters are stored in permanent LUTs using a fixed VHDL module containing them as constant values.
2. Online recalibration without changing the firmware of the FPGA. That would require to implement fast rewritable memory slots for the filter parameters that can be updated using an interface during the normal FPGA operation, introducing a calibration mode.

The easiest choice is, of course, to introduce static memory by defining the filter parameters in a VHDL module. This procedure was used for the first implementations, as shown in the example code 4.6 (see the appendix for the full coefficient set used).

Code Example 4.6: Implementation of filter parameters using static memory

```

1  type tfullcoeffset is array (1 to 3) of std_logic_vector(PED_SIZE + DEPTH*A_I_SIZE - 1
2      downto 0);
3  type tfullcoefftable is array (0 to CHANNELS-1) of tfullcoeffset;
4
5  --predefined coefficients: 4 channels à 3 gains (gain 4 = everything 0)
6  constant coeff_init_values : tfullcoefftable :=
7  (
8      (
9          pedestal & a1 & a2 & a3 & a4 & a5
10         x"FFFF" & x"000004" & x"ffffff" & x"000012" & x"000055" & x"000083" --channel 0,gain 1
11         x"FFFE" & x"000003" & x"fffffd" & x"0000d2" & x"000035" & x"0000c3" --channel 0,gain 2
12         x"FFFD" & x"000002" & x"fffffe" & x"000072" & x"000015" & x"000013" --channel 0,gain 3
13     ),
14     ...
15 );

```

Here, 16 bits are used for the pedestal Ped and 24 bits are used for the filter coefficients a_i , as it is the case for the final design.

A more sophisticated implementation is to use fast rewritable memory. Ordinary block RAM does not fulfill that requirement, as the readout of RAM introduces additional latency. An optimal choice for such an application are reconfigurable LUTs.

The Virtex-5 comes along with 32 bit Shift Register Look-Up Tables with Clock Enable (SRLC32E) [35]. The SRLC32E component is a variable length, 1 to 32 clock cycle shift register implemented within a single Virtex-5 LUT. The shift register can be of a fixed length, static length, or it can be dynamically adjusted by changing the address lines to the component. The SRLC32E also features an active high clock enable and a cascading feature in which multiple SRLC32Es can be cascaded in order to create greater shift lengths.

To implement a SRLC32E unit, the ports have to be asserted as given in table 4.5.

The general idea is to take one (or more) of those SRLC32E components to store one bit of a parameter. The address depth of 32 is used to choose (multiplex) the different gains and channels for the specific bit of the parameter. As the gain is available as a 2 bit value, those two bits will make up the lowest two bits of the address bus A. The

Table 4.5: Port definition of an SRLC32E component.

Port name	Direction	Size	Function
Q	output	1 bit	shift register data output
Q31	output	1 bit	shift register cascaded output (to be connected to the D input of a subsequent SRLC32E)
D	input	1 bit	shift register data input
clk	input	1 bit	clock
ce	input	1 bit	active high clock enable
A	input	5 bit bus	address bus for dynamic depth selection of the SRLC32E

remaining three bits are used to address the channel selection for up to 2^3 channels. If more than eight channels are to be implemented, another SRLC32E is introduced in line to cover another eight channels, and so on (but due to the limited frequency of the DSP units not more than 16 channels are realistic for one McFilt unit). Figure 4.8 illustrates the implementation of this concept.

To store the 16 bit pedestals Ped and the 24 bit filter coefficients a_i for up to eight channels, $16 + 5 \times 24 = 136$ SRLC32E components are required. This number doubles for a system of 9 to 16 channels.

The difference in reading and writing the parameters is remarkable: Due to the fact that the SRLC32E components are arranged subsequently over the full set of parameters, they build up a chain with only one global input bit and one output bit on the configuration path whereas the readout of the coefficients a_i and the pedestal Ped in their full width is possible independently due to the parallel readout of a group of SRLC32E components.

Thus, when reconfiguring the filter parameters, it is required to stream the complete set of filter parameters into the memory module, non regarding if there are unused slots in the SRLC32E components or not. For example, implementing a filter of four channels wastes half the memory provided by the SRLC32Es as empty places have to be filled up to complete the chain. But this waste of resources is willingly accepted in order to reach a vanishing latency by this solution in comparison to the usage of block RAM that consumes at least one clock cycle.

Additionally, due to the chained arrangement, the bit order of the stream of calibration data is modified. A C programme is written to perform that task which is dependent on the number of channels.

For recalibration, the UDP/IP 1 GbE interface is used. The size of a calibration packet depends on the number of channels and is bound to the SRLC32E components. The packet size sums from 42 bytes UDP/IP header data and the real data, as shown in table 4.6. The calibration UDP packet used for testing is given in appendix A.4.

Those UDP packet sizes are well within the standard limit of 1522 bytes of a normal Ethernet frame. The calibration packet used for testing four channels is also given in

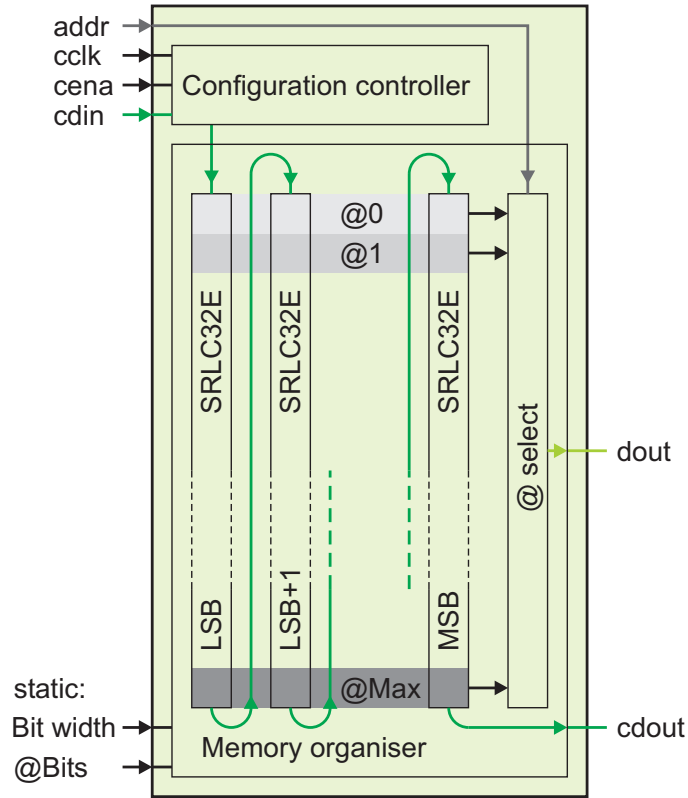


Figure 4.8: The sequential arrangement of SRLC32E components in the memory module allows an implementation of a fast and rewritable memory block for the filter parameters. While reading a specific n bit wide parameter is done instantaneously by selection of its address (gray horizontal masks), the reconfiguration is done globally by streaming the full set of filter parameters through all SRLC32E components bitwise (green data path).

the appendix A.4.

Table 4.6: UDP calibration packet size for different numbers of channels (sizes in bytes).

Number of channels	Data size	Packet size
1 to 8	544	586
9 to 16	1088	1130

4.5 RAM Management

Here, briefly the solution of the RAM management shall be presented. The problem is that there is only one data storage device, the block RAM module, with one dedicated read port and one separated write port, but there are several instances which want to access this RAM module. Like in any operating system, a RAM controller had to be written to deal all read and write requests and, in case of simultaneous requests, to decide about their sequence.

In the first stage of development of the RAM manager, the approach was to define dedicated ports for each module trying to access the RAM as shown in the example code 4.7.

Code Example 4.7: Port definition of the first RAM manager

```

1 entity RAM_management is
2   port (
3     -- controls
4     clk          : in  std_logic;    -- clock signal
5     rst          : in  std_logic;    -- reset
6
7     -- RAM requests of calibration: read only
8     r_flag_calib : in  std_logic;    -- module wants to read when set
9     r_addr_calib : in  std_logic_vector(7 downto 0); -- address to be read
10    r_fine_calib  : out std_logic;    -- confirmation of successful read
11
12    -- RAM read requests of UDP-module:
13    r_flag_enet   : in  std_logic;
14    r_addr_enet   : in  std_logic_vector(7 downto 0);
15    r_fine_enet   : out std_logic;
16
17    -- RAM write requests of UDP-module:
18    w_flag_enet   : in  std_logic;    -- module wants to write when set
19    w_addr_enet   : in  std_logic_vector(7 downto 0); -- to this address
20    w_data_enet   : in  std_logic_vector(63 downto 0); -- this data
21    w_fine_enet   : out std_logic;    -- confirmation of successful write
22
23    ...
24
25    -- RAM data output
26    r_data        : out std_logic_vector(63 downto 0) -- read data output
27  );
28 end RAM_management;

```

This was a very practical approach since every module came along with those ports and a one-to-one connection could be made. The prioritisation of the different modules was then done using a simple priority list, such as presented in example code 4.8.

Code Example 4.8: Priority selection of the first RAM manager

```

1 -- prioritised choice list of write requests
2 out_selector <=
3   McFilt when w_flag_mcfilt = '1' else
4   Enet   when w_flag_enet   = '1' else
5   none;
6
7 -- set ram write address accordingly
8 with out_selector select w_addr <=
9   w_addr_mcfilt when mcfilt ,
10  w_addr_enet   when enet ,
11  (others => '0') when none;

```

This solution worked fine but had the disadvantage that for any new module trying to access the RAM, a redefinition of the RAM manager had to be made and the prioritised choice list had to be reconfigured by hand by adding the new module accordingly.

Thus another approach was implemented to make the RAM manager independent of the number of external modules by introducing a different port definition as given in example code 4.9.

Code Example 4.9: Port definition of the second RAM manager

```

1 entity RAM_management_bus is
2   generic (
3     — number of concurrent writing modules to manage
4     write_cnt      : positive;
5     — number of concurrent reading modules to manage
6     read_cnt       : positive
7   );
8   port (
9     — controls
10    clk           : in  std_logic;
11    rst           : in  std_logic;
12
13    — write ports
14    w_flag_bus    : in  std_logic_vector(write_cnt-1 downto 0);
15    w_addr_bus    : in  t_w_addr(write_cnt-1 downto 0);
16    w_data_bus    : in  t_w_data(write_cnt-1 downto 0);
17    w_fine_bus    : out std_logic_vector(write_cnt-1 downto 0);
18
19    — read ports
20    r_flag_bus    : in  std_logic_vector(read_cnt-1 downto 0);
21    r_addr_bus    : in  t_r_addr(read_cnt-1 downto 0);
22    r_fine_bus    : out std_logic_vector(read_cnt-1 downto 0);
23
24    — RAM data output
25    r_data        : out std_logic_vector(r_data_width-1 downto 0)
26  );
end RAM_management_bus;

```

The second RAM manager made use of the derived port definition types shown in example code 4.10. Here, **w_data_width** is the data width and the width of the RAM write (read) address bus is given by **w_addr_width** (**r_addr_width**).

Code Example 4.10: User defined port types for the second RAM manager

```

1 type t_w_addr is array(natural range <>) of std_logic_vector(w_addr_width-1 downto 0);
2 type t_w_data is array(natural range <>) of std_logic_vector(w_data_width-1 downto 0);
3
4 type t_r_addr is array(natural range <>) of std_logic_vector(r_addr_width-1 downto 0);

```

The priority indication here is given by the order of modules assigned to the slots of the bus so no more choice list has to be set up by hand.

The proper assignment of the data write address reduces to one line (example code 4.11) making use of the function **lssb_idx** returning the position of the least-significant set bit of the writing flag bus **w_flag_bus**.

Code Example 4.11: Automatised priority selection by the second RAM manager

```

1 — select addr and data to be taken from bus
2 w_addr <= w_addr_bus(to_integer(unsigned(lssb_idx(w_flag_bus))));
  w_data <= w_data_bus(to_integer(unsigned(lssb_idx(w_flag_bus))));

```

The total resource utilisation of both, the new and the previous RAM manager, is presented in table 4.7.

Table 4.7: The resource utilisation of the RAM managers for two writing and four reading modules vs. for five modules each: A slight improvement can be noticed. The percentages refer to the total available resources (see table 3.2).

resource	2 write, 4 read modules		5 read, 5 write modules	
	old	new	old	new
Slice registers	3 (0.01%)	0 (0%)	3 (0.01%)	0 (0%)
Slice LUTs	98 (0.22%)	94 (0.21%)	178 (0.39%)	173 (0.39%)
Total occupied slices	58 (0.52%)	57 (0.51%)	140 (1.25%)	85 (0.76%)

Not only the flexibility of the RAM manager is thus increased for a use in any design, but also the resource utilisation could slightly be optimised.

4.6 High Clocking of the Multi channel Filter

The functionality of the test design and the implemented McFilt unit was successfully tested at the clock frequency of 125 MHz. In order to achieve higher clock frequencies, an important change in the data flow is needed. As shown in figure 4.3, the McFilt is constrained in clock frequency by the output interface of the 1 GbE UDP/IP link.

To aim at higher clock frequencies for the prototype filter, it is thus required to strictly separate the calculation from the sending interface. To do so, the McFilt unit is embedded in a dual clock domain FIFO environment (McFilt Manager) handling the data transfer from and to the RAM. That allows to use completely independent clock frequencies on the inner and on the outer side of the filter but goes in hand with a disrupted data flow as depicted in figure 4.9.

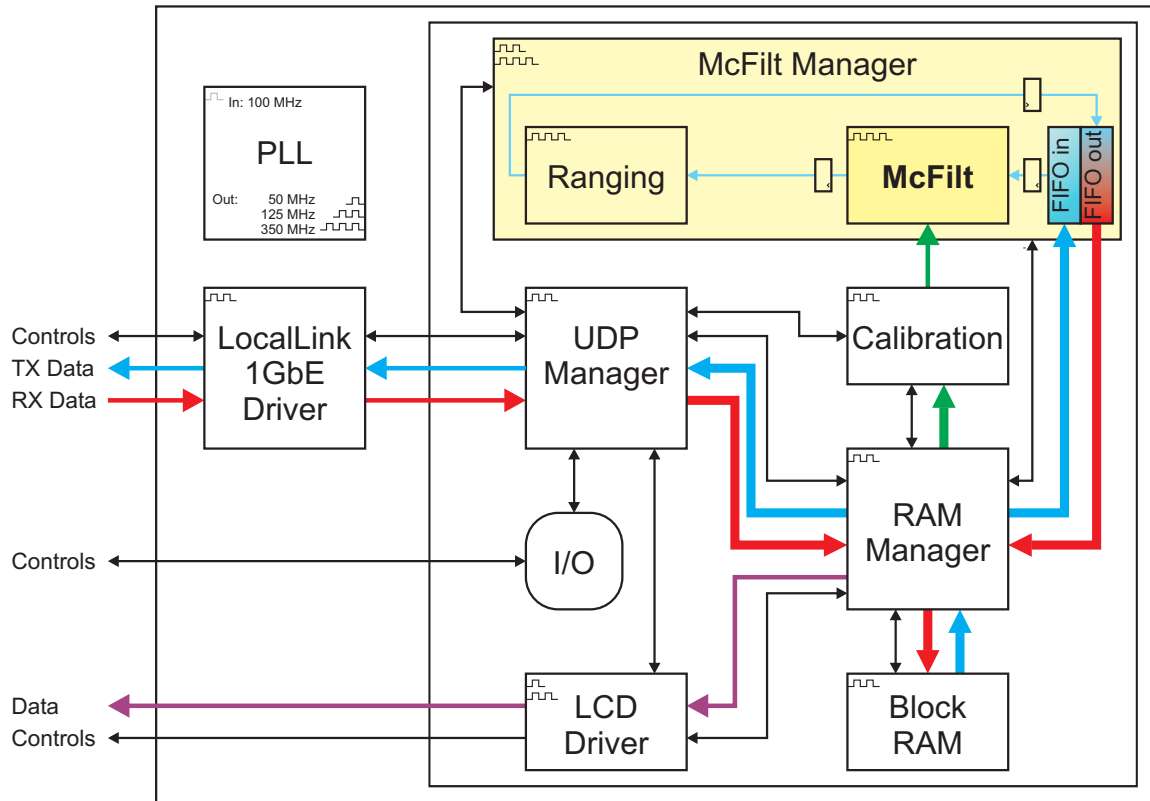


Figure 4.9: The second layout of the FPGA filter implementation makes use of two separate clock domains in the McFilt unit environment (McFilt Manager). As in the first layout (figure 4.3), incoming data samples are stored in the RAM first. The main difference is the introduced McFilt Manager containing an input and an output FIFO. The input FIFO is filled up from the RAM, four samples at a time assuring McFilt not to run out of input data, functioning even at $4 \times 125 \text{ MHz} = 500 \text{ MHz}$. The filter result is buffered in the output FIFO to collect four results to be stored in the RAM again conjointly. After the termination of the calculation, the result is then transmitted as an UDP/IP packet.

The decoupling of the McFilt unit from the 1 GbE interface and the RAM by FIFOs is a major step towards the final architecture of the ROD modules in prototype developing. Up to now, neither the internal nor the external interfaces of the RODs are defined exactly. Thus, dual clock domain FIFOs represent the most flexible and independent solution. To avoid artificial latency originating from these FIFOs, it is recommended to leave them out for the final architecture or to replace them by adequate elements.

To profit from a stable and more precise clock, the DCM has been exchanged by two PLL modules. The first module provides the design static clocks for the LocalLink (125 MHz) and the LCD Driver (50 MHz) with a fixed comparison frequency of 1000 MHz based on a reference frequency of 100 MHz. The second, McFilt-dedicated PLL uses the same reference frequency of 100 MHz, but is implemented with variable multiplication factor `clk_mult` and divider `clk_div`. By assigning different values, it was possible to drive the design up to 350 MHz sequentially. Table 4.8 indicates the successfully tested design frequencies.

Table 4.8: Steps taken to increase the clock frequency of the Multi channel Filter

Multiplier	Divider	Achieved frequency in MHz
10	4	250
9	3	300
10	3	333
7	2	350

Increasing the clock frequency was accompanied by introducing supplementary registers at dedicated locations to resolve timing constraints caused by data path lengths in the FPGA. Not only the embedding McFilt Manager is affected, as can be seen in figure 4.9, but also the core processing unit McFilt was slightly adjusted. Two further register stages were required to allow a high speed calculation of up to 350 MHz. The modifications of the McFilt unit can be seen in figure 4.10.

In total, the design suffers from two supplementary stages of registers, summing up the latency of the McFilt to five clock cycles instead of three. But regarding the total latency in real time, that turns out to be an improvement of about 40% in comparison to the previous result operating at 125 MHz:

$$\Delta t_{E-S, 350 \text{ MHz}} = 5 \times (350 \text{ MHz})^{-1} \approx 14.3 \text{ ns} \quad (4.4)$$

Table 4.9 summarises the resource utilisation of the presented four filter designs. An evaluation of those numbers will be given in the next chapter.

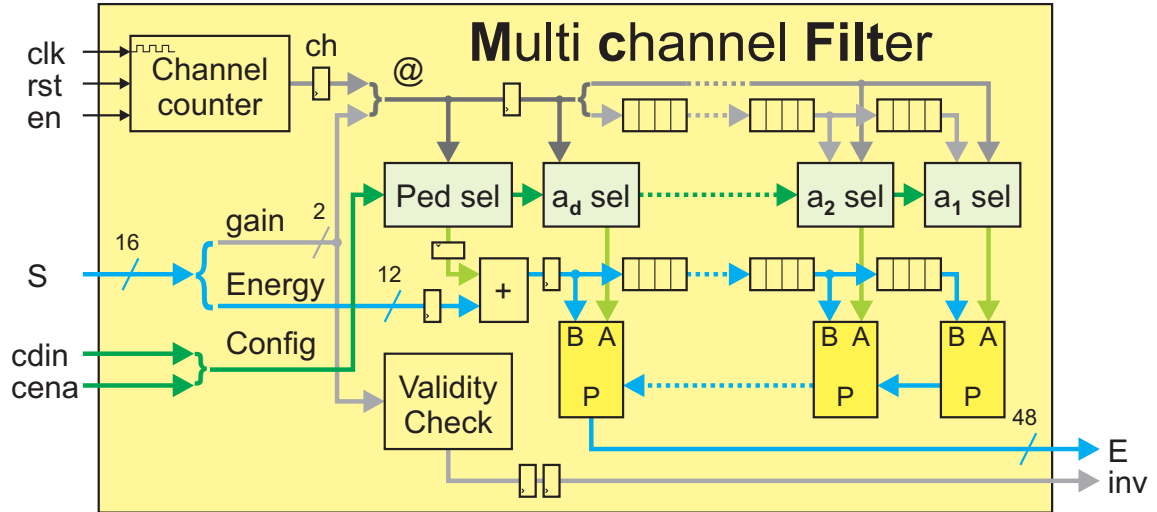


Figure 4.10: This scheme of the McFilt unit illustrates the additional stage registers (compare to figure 4.7) that were inserted to shorten the constraining data path lengths. It basically concerns the selection of the filter parameters and the adder to correct the data sample by its pedestal.

Table 4.9: Resource utilisation of the different filter designs. The percentages refer to the total available resources (see table 3.2).

Resource	1st	2nd	3rd	4th
Slice registers	68 (0.15%)	726 (1.62%)	82 (0.18%)	132 (0.29%)
Slice LUTs	240 (0.54%)	476 (1.06%)	213 (0.48%)	223 (0.50%)
Total occupied slices	132 (1.18%)	376 (3.36%)	73 (0.65%)	80 (0.71%)
DSP48E	5 (3.91%)	5 (3.91%)	5 (3.91%)	5 (3.91%)

4.7 Output Data Encoding

In addition to the FIR filter, the final result has to be formatted to a predefined data bus. The requirements are detailed in section 2.5.5. The implementation to achieve this goal is done on the logical bit level as given in example code 4.12.

Code Example 4.12: Implementation of the encoding of the output data

```

1  — Input is the 48 bit result from the McFilt module
2  — The sign and the 13 most significant bits of that value have to be found
3  — based on 4 range scales. The range is distinguished by different bases of
4  —  $2^n$  ( $n * SHIFT$ ),  $SHIFT$  is a free parameter from 1 to 3, with 3 being the
5  — target design value.
6  — 2 bits are used to decrypt the range, representing  $n = 0$  to  $n = 3$ 
7  — negative results are only ranged up to  $n = 0$ 
8
9  sign <= Input(Input'left);      — the most left bit is always the sign
10
11 — high level decision if not in range: too negative when higher bits than
12 — the 13 lowest are set; too positive when bit higher than  $3 * SHIFT$  are set
13 notinrange <= '0'
14 when 0 = unsigned(not(Input(Input'left downto LOWESTBIT+13)))
15   or 0 = unsigned(Input(Input'left downto LOWESTBIT+13+3*SHIFT))
16   else '1';
17
18 — the for  $n = 1$  to  $n = 3$  ranging considered bits are defined as prefix
19 — intherange selects the range with respect to that prefix
20 prefix <= Input(LOWESTBIT+12+3*SHIFT downto LOWESTBIT+13+0*SHIFT);
21
22 intherange <=
23   "00" when sign = '1' else
24   "11" when 0 /= unsigned(prefix(3*SHIFT downto 1+2*SHIFT)) else
25   "10" when 0 = unsigned(prefix(3*SHIFT downto 1+2*SHIFT)) and
26   0 /= unsigned(prefix(2*SHIFT downto 1+1*SHIFT)) else
27   "01" when 0 = unsigned(prefix(3*SHIFT downto 1+1*SHIFT)) and
28   0 /= unsigned(prefix(1*SHIFT downto 1+0*SHIFT)) else
29   "00";
30
31 — outtherange takes care of invalid inputs
32 outtherange <= (not sign & sign) or (inval & inval);
33
34 — prevail are the 13 significant bits selected via intherange
35 with intherange select prevail <=
36   Input(LOWESTBIT+12+3*SHIFT downto LOWESTBIT+3*SHIFT) when "11",
37   Input(LOWESTBIT+12+2*SHIFT downto LOWESTBIT+2*SHIFT) when "10",
38   Input(LOWESTBIT+12+1*SHIFT downto LOWESTBIT+1*SHIFT) when "01",
39   Input(LOWESTBIT+12+0*SHIFT downto LOWESTBIT+0*SHIFT) when others;
40
41 — the Output ranging and sign bits are set with outtherange override
42 with notinrange nor inval select Output(15 downto 13) <=
43   intherange & sign when '1',
44   outtherange & '1' when others;
45
46 Output(12 downto 0) <= prevail;

```

As can be seen from the resource utilisation (table 4.10), a large amount of LUTs is required for this relatively direct translation to VHDL. Analysing the created schematic (figure A.2) shows a sequential arrangement of three stages of different types of LUTs. Clocking this module by introducing a registered output turns out to be feasible up to a frequency of $1/(2.4 \text{ ns}) \approx 416 \text{ MHz}$. That frequency drops to $1/(2.55 \text{ ns}) \approx 392 \text{ MHz}$ when introducing registers with enable and reset function.

Table 4.10: Resource utilisation of the output data encoding module. The percentages refer to the total available resources (see table 3.2).

Resource	Utilisation
Slice registers	0 (0%)
Slice LUTs	33 (0.07%)
Total occupied slices	25 (0.22%)

5 Results

This section summarises the results achieved and evaluates the feasibility of an upgrade of the LAr calorimeter readout system.

5.1 FPGA Resource Estimation

The full design presented in the previous sections has a total resource utilisation of the FPGA as given in table 5.1. The design in this configuration is not suitable for an upgrade but will serve to extrapolate the required resources in the following, based on the resources used for one McFilt unit and the auxiliary environment providing the connectivity of the McFilt unit to the environment.

Table 5.1: Resource utilisation of the full design. The percentages refer to the total available resources (see table 3.2).

Resource	1 McFilt unit	all auxiliary environment	total design
Slice registers	132 (0.29%)	1448 (3.23%)	1580 (3.53%)
Slice LUTs	223 (0.50%)	1741 (3.89%)	1964 (4.38%)
Total occupied slices	80 (0.71%)	722 (6.45%)	802 (7.16%)
DSP48E	5 (3.91%)	0 (0.00%)	5 (3.91%)

The following resource estimation will be based on an operation frequency of $f_{\text{McFilt}} = 400$ MHz for the McFilt unit, corresponding to the treatment of ten channels at the LHC clock speed of $f_{\text{LHC}} = 40$ MHz. Let k represent this multiplicity:

$$k = \frac{f_{\text{McFilt}}}{f_{\text{LHC}}} = 10. \quad (5.1)$$

A FEB, also in the upgraded LAr calorimeter readout, is supposed to cover 128 channels. Demanding an upgraded ROD to group 14 FEBs, it will have to deal with 1792 channels, corresponding to about 180 McFilt units.

As was shown in the tables indicating the resource utilisation of the separate components, the number of slices provided (registers and LUTs) does not lead to any limitations. Requiring 80 slices for each McFilt unit, one ends up with 14400 slices in total. This amount is well exceeding the available number of slices in the Virtex XC5VFX70T by a rough factor of 1.5. But there are more integrated versions of the Virtex-5, such as XC5VSX95T, which provide a sufficient amount of resources,

neglecting the overhead produced by the surrounding interface and management components. That effectively means, that one entire ROD could integrate the necessary functionality using one large FPGA, as regards registers and LUTs only.

Another crucial source are the DSP48E slices which turn out to be the bottleneck of the implementation. Five of them are required for each McFilt unit. The total number n of DSPs per ROD treating 14 FEBs can be estimated by

$$n \frac{\text{DSP}}{\text{ROD}} = \frac{5 \text{ DSP}}{\text{channel}} \times \frac{1}{k} \times 128 \frac{\text{channel}}{\text{FEB}} \times 14 \frac{\text{FEB}}{\text{ROD}} = 8960 \frac{\text{DSP}}{k \times \text{ROD}}. \quad (5.2)$$

That number largely exceeds the available number of DSPs in the Virtex-5 used, even with multiplicity $k = 10$. But as given in table 3.3, the most integrated versions (XC5VSX240T) offer enough DSPs to satisfy the needs so that under optimal settings one ROD board could be equipped with only one FPGA coping with the optimal filtering.

Another estimation that can be made, is the overall number of FPGAs, F , required to process all 182468 channels of the LAr calorimeters. When neglecting the granularity imposed by FEBs and RODs that number can be estimated by

$$F = \frac{5 \text{ DSP}}{\text{channel}} \times \frac{1}{k} \times 182468 \text{ channels} \times \left(\frac{\text{DSP}}{\text{FPGA}} \right)^{-1}. \quad (5.3)$$

Table 5.2 gives the total number of FPGAs and table 5.3 indicates the FPGAs per ROD required for different multiplicities k and versions of Virtex-5 to cover all channels.

Table 5.2: Total number of FPGAs, F , required for the full LAr calorimeter read-out according to equation 5.3 for different numbers of DSPs per FPGA and multiplicities k .

Virtex-5 version	DSPs	$k = 8$	$k = 9$	$k = 10$	$k = 11$
XC5VFX70T	128	892	793	713	649
XC5VFX100T	256	446	397	357	325
XC5VSX240T	1056	109	97	87	79

Table 5.3: FPGAs per ROD required for the full LAr calorimeter readout according to formula 5.2 considering the different numbers of DSPs per FPGA and multiplicities k .

Virtex-5 version	DSPs	$k = 8$	$k = 9$	$k = 10$	$k = 11$
XC5VFX70T	128	9	8	7	7
XC5VFX100T	256	5	4	4	4
XC5VSX240T	1056	2	1	1	1

Based on these estimations it seems to be feasible to produce ROD boards carrying only one FPGA for the optimal filtering in restricted view of resource estimation.

5.2 Latency Estimation

The latency of the McFilt has been calculated in equation 4.4 to be 14.3 ns which is far less than the time spacing between two bunch crossings at the LHC of 25 ns. That is a very positive result as the overall latency for the upgraded readout system is very tight.

Figure 5.1 summarises these latency requirements as well as the expected data amounts that will occur in the readout system. Table 5.4 breaks down the latency estimation of the different stages of the foreseen upgrade [36].

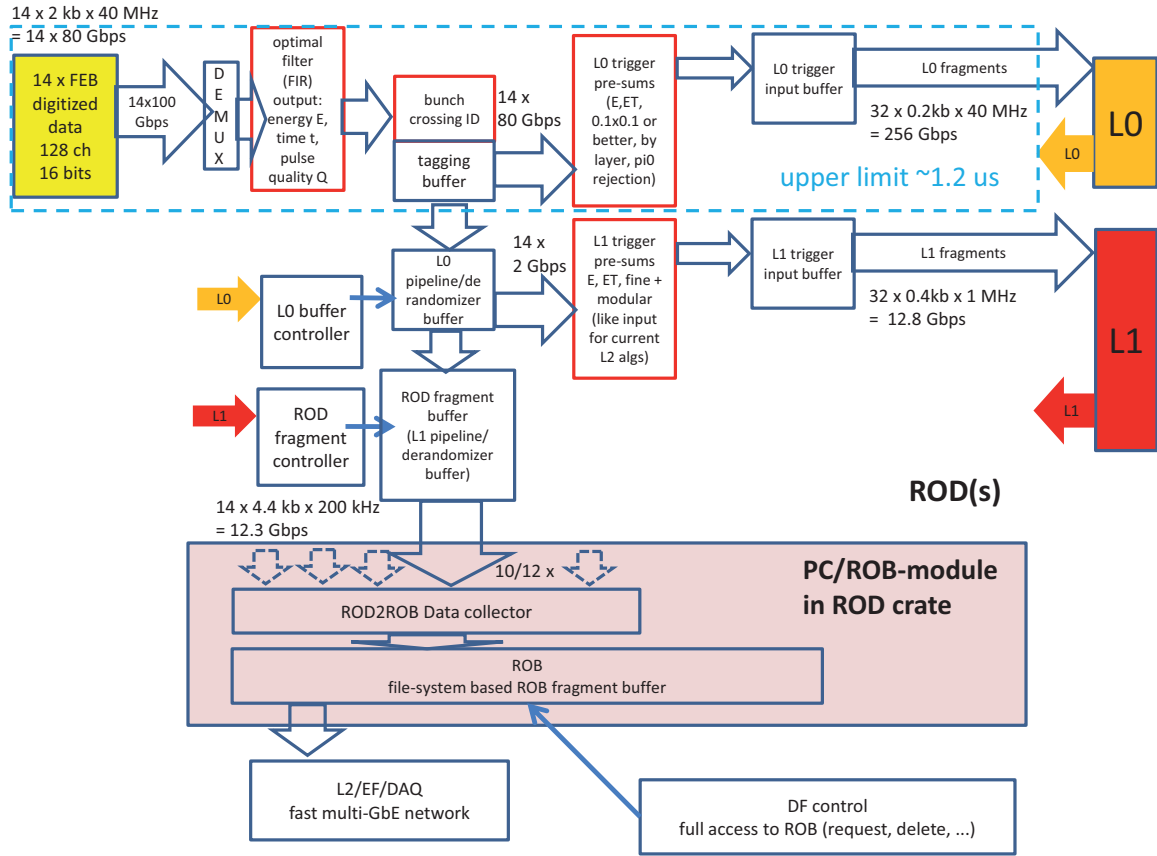


Figure 5.1: Scheme of the upgraded readout system illustrating the high requirements of latency and data transfer between the different modules [37]. The optimal filtering is part of the direct data path to the L0 trigger. An upper limit of that latency is $1.2 \mu s$, so that the on-detector electronics receives a trigger signal in time. It is thus required to minimise the latencies of each component as much as possible.

The latency of the overall design including the Ethernet interface has been tested. Using the sample data packet of 40 samples (ten samples for four channels) an average total latency of about $80 \mu s$ from sending the packet with the packet builder to its reception can be measured (compare figure 3.8). This time explicitly splits into 14.3 ns McFilt latency, $40 \times (350 \text{ MHz})^{-1} \approx 114 \text{ ns}$ calculation time for the 40 samples and about $2 \mu s$ required to receive and transmit the data by the UDP Manager (2 times

Table 5.4: The table indicates the latency estimation for the different stages of the upgraded LAr calorimeter readout system [36] in units of bunch crossings (1 BC = 25 ns). The signal delay due to the cabling is remarkable. The developed FIR filter calculation is well within the foreseen time budget of 5 BCs reserved for energy and time calculation.

Stage	BCs	Subtotal (BCs)
Time-of-flight from interaction point to detector	0.6	28.2
Cable to FEB	1.2	
Processing on FEB	12.4	
Optical cable (70m) from FEB to ROD	14	
Deserialising on ROD	2	
Channel demultiplexing on ROD	1	
Pedestal subtraction	1	
E, t calculation	5	14
Digital summation	2	
Multiplexing on ROD	1	
Serialiser on ROD	2	
Optical cable (15m) from ROD to L0 trigger	3	3
Total		45.2 (1.13 μ s)

header (46 bytes) and data (80 bytes) at 125 MHz). The remaining 78 μ s arise from the Ethernet MAC wrapper, the internal data transfer in the FPGA from the MAC wrapper FIFOs to the RAM and reverse, and the off-FPGA data transfer on the ML507 test board to the dedicated Ethernet chip, as well as in-PC processing time of the test setup.

It has to be precised that the different latency contributions from the overall design will enter the upgraded readout of the LAr calorimeter at different places with different timing constraints. The most critical one to meet is the L0 trigger constraint of 1.2 μ s (compare table 5.4). Here, the McFilt latency of 14.3 ns enters directly as well as some further transmission time for the data samples from the FEBs. The foreseen time budget of five bunch crossings is very well met by the FIR filter implementation and enough budget is left to implement further calculations.

A less crucial latency constraint of about 6 μ s is imposed by the L1 trigger. During this time the data has to be buffered on the ROD boards. For this purpose, the internal as well as the external RAM resources of different FPGAs will have to be studied.

5.3 Data Rates

As partly depicted in figure 5.1, the data rates coming from one FEB with 128 channels, 16 bit width and the LHC frequency of 40 MHz are estimated to 80 Gb/s. One ROD combining 14 FEBs will thus have to deal with 1.12 Tb/s of data (excluding overhead). The feasibility of data transfer using commercial parallel optical transceivers running at 75 Gb/s has successfully been tested with Virtex-5 FPGAs [38].

The total amount of data to be forwarded from the RODs to the ROB is reduced by the ratio of the readout frequency of the FEBs (40 MHz) to the L1 trigger accept rate (about 200 kHz), which is a factor of 20.

The communication between the RODs and ROB to transfer the remaining 12.3 Gb/s of L1 triggered data will be based on the Ethernet networking standard. The prototype of the 1 GbE interface of the design presented has to be further developed towards an implementation of 10 Gigabit Ethernet to handle this data rate in a most efficient way.

6 Conclusion and Outlook

In this work a Field Programmable Gate Array (FPGA) implementation of a fast five-staged Finite Impulse Response (FIR) filter prototype to process digitised data samples of the Liquid Argon (LAr) calorimeter Front End Boards (FEBs) was developed. The FIR filter was successfully tested up to an operation frequency of 350 MHz consuming a minimum of resources and achieving a total latency of only 14.3 ns. The FIR filter implementation makes use of interleaved design to cope with multiple input channels and the gain specification of every sample is respected.

6.1 Data Transfer

The implementation of 10 Gigabit Ethernet is still an outstanding task. It has to go in hand with detailed studies of the latencies of the different parts of the readout system, based on already existing estimations [38]. This work contributes to this task by providing a properly functioning UDP implementation for 1 Gigabit Ethernet. To make the transition to 10 Gigabit Ethernet, a hardware implementation for common interfaces (different from those applied for 1 Gigabit Ethernet) has to be developed and a slight modification of the provided UDP implementation will be required to meet the higher timing constraints.

6.2 Filter Development

In future, the possibility of further enhancements of the operation frequency of the McFilt unit has to be studied as this frequency directly influences the number of channels that can be processed within one bunch crossing. This may lead to a further reduction of the overall resource utilisation.

Furthermore, a demonstrator FPGA containing a maximum of McFilt units is to be developed to verify the feasibility of the estimated values given in the previous section. There might occur clocking problems or problems of integrity when trying to implement the full number of required McFilt units (about 180, depending on the multiplicity k) in one Virtex-5 XC5VSX240T ROD FPGA that will have to be solved. Additionally, newer FPGA families, like the Xilinx[®] Virtex-6 or FPGAs of alternative vendors like Altera, are to be considered and tested.

Another general point is the calculation of further physical relevant data from the data samples as indicated in figure 5.1. Additional to the calculation of the energy out of

five samples, the calculation of the corresponding event time t and a quality factor Q must be implemented. That requires still some calculation overhead, but the basic procedure is similar. It implies a supplementary amount of DSP48E slices which, in first approximation, leads to a doubling of resources. Especially for calculating the event time t an effective solution has to be found, as a further division of $E \times t$ by the energy, E , calculated in parallel is required.

6.3 HL-LHC and Physics Measurements

The development presented here is a step required towards an upgrade of the LHC to the HL-LHC. The pile-up of proton-proton collisions which is expected to be much higher, can be suppressed significantly by evaluating the full LAr calorimeter information, in other words to read out 182468 channels individually. Only with an effective pile-up suppression the very interesting measurements to test the well established Standard Model of particle physics and models beyond that can be taken.

For example, the analysis of the longitudinal vector boson scattering $W_L W_L \rightarrow W_L W_L \rightarrow e^+ \nu e^- \nu / e^\pm \nu q q$ must show a deviation from the Standard Model to prevent unitary violation [39]. The existence of extra dimensions could be explored by the observation of heavy gauge boson decays $Z' \rightarrow e^\pm e^\mp$ [40]. Another very interesting analysis is the measurements of the Higgs boson self-coupling in $HH \rightarrow 4W \rightarrow e^\pm e^\mp + 4$ jets events to put constraints on the Higgs potential [41].

All these reactions contain electrons in the final state and are potentially rare processes. Thus, these analysis rely on a functioning, high granularity readout system of the electromagnetic calorimeter with high performance. The FIR filter with very low latency developed here is a first effort in proving the feasibility of the sophisticated upgrade plans of the LHC.

List of Figures

2.1	Outline of the ATLAS detector at the LHC	5
2.2	The ATLAS liquid argon calorimeter	6
2.3	Sketch of a LAr calorimeter barrel module	7
2.4	Current LAr calorimeter readout scheme	8
2.5	LAr calorimeter FEB signal processing chain	9
2.6	Signal pulse of the front end electronics	10
2.7	Noise vs. peaking time of the FEB	11
2.8	Pedestal distribution of the Front End Boards	12
2.9	Envisaged LAr calorimeter readout scheme for the HL-LHC	15
3.1	Switch box topology in FPGAs	24
3.2	A register	25
3.3	A multiplex - demultiplex unit	26
3.4	Basic layout of a Phase-locked loop (PLL)	27
3.5	Xilinx [®] ML507 Virtex-5 test board	29
3.6	Block RAM configuration settings using Xilinx [®] CORE Generator	35
3.7	Block diagramme of a Virtex-5 DSP48E slice	36
3.8	Wireshark as a Ethernet sniffer	45
3.9	Colasoft Packet Builder to form UDP packets	46
4.1	TriMode Ethernet CORE example design	48
4.2	Timing of the control signals of the MAC wrapper	49
4.3	First layout of the FPGA filter implementation using UDP/IP via 1 GbE	54
4.4	Data flow of the first calculation design	57
4.5	Data flow of the second calculation design	59
4.6	A DSP48E component in the configuration of a multiply-adder	61
4.7	The Multi channel Filter unit in its first design at 125 MHz	62
4.8	Memory module: arrangement of SRLC32E components	65
4.9	Second layout of the FPGA filter using two separate clock domains	69
4.10	The Multi channel Filter unit design for up to 350 MHz	71
5.1	Scheme of the upgraded readout system including latency and data flow	77
A.1	Simulation of the Multi channel Filter unit at 125 MHz	89
A.2	ISE schematic of the output data encoding module	90

List of Tables

2.1	Readout data format of the ATLAS LAr calorimeter FEB	13
2.2	Definition of the input data bus of the filter	18
2.3	Definition of the 2 bit gain value	18
2.4	Ranges of the filter coefficients a_i for different gain domains	20
2.5	Data output specifications including ranging	21
3.1	Truth table for the logic function $O = ((\bar{a}_0 \wedge a_1) \vee (\bar{a}_0 \wedge a_2))$	25
3.2	Resources of the Virtex XC5VFX70T	30
3.3	Number of DSP48E slices per Virtex-5 family member	37
3.4	The different specifications of Ethernet by time	39
3.5	The OSI reference model	39
3.6	Definition of an 802.3 MAC Frame	40
3.7	Structure of the IPv6 header	41
3.8	Definition of the IPv6 header	41
3.9	Structure of the IPv4 header	42
3.10	Definition of the IPv4 header	42
3.11	Structure of the UDP header	43
3.12	Definition of the UDP header	43
4.1	Resource utilisation of the UDP rx and tx modules	53
4.2	Gain influence in the first filter design	57
4.3	Resource utilisation of the first filter design	58
4.4	Resource utilisation of the second filter designs	60
4.5	Port definition of an SRLC32E component	64
4.6	UDP Calibration packet size for different numbers of channels	65
4.7	Resource utilisation of the RAM manager	68
4.8	Steps taken to increase the clock frequency of the Multi channel Filter .	70
4.9	Resource utilisation of the different filter designs	71
4.10	Resource utilisation of the output data encoding module	73
5.1	Resource utilisation of the full design	75
5.2	Total number of FPGAs, F , required for the full LAr calorimeter readout	76
5.3	FPGAs per ROD required for the full LAr calorimeter readout	76
5.4	Latency estimation of the upgraded LAr calorimeter readout	78
A.1	UDP/IP sample data packet for 4 channels	94
A.2	UDP/IP Calibration data packet for 4 channels	95

List of Code Examples

3.1	64-to-32 bit multiplexer in Verilog	31
3.2	64-to-32 bit multiplexer in VHDL	32
3.3	General outline of a User Constraint File	33
3.4	First part of a state machine	33
3.5	Second part of a state machine	34
3.6	Optional list of concurrent assignments in a state machine	34
3.7	Interface of the chosen dual port RAM	35
4.1	Port definition of the LocalLink for the Ethernet interface	49
4.2	Interface definition of the UDP transmitting module	50
4.3	Implementation of the IP CRC in the UDP transmitting module	51
4.4	Interface definition of the UDP receiving module	52
4.5	Interface definition of the first calculation module	58
4.6	Implementation of filter parameters using static memory	63
4.7	Port definition of the first RAM manager	66
4.8	Priority selection of the first RAM manager	66
4.9	Port definition of the second RAM manager	67
4.10	User defined port types for the second RAM manager	67
4.11	Automatised priority selection by the second RAM manager	67
4.12	Implementation of the encoding of the output data	72

A Appendix

A.1 Design Simulation

During the development phase the ISIM Simulator was used to evaluate the correct filter behaviour of the McFilt unit.

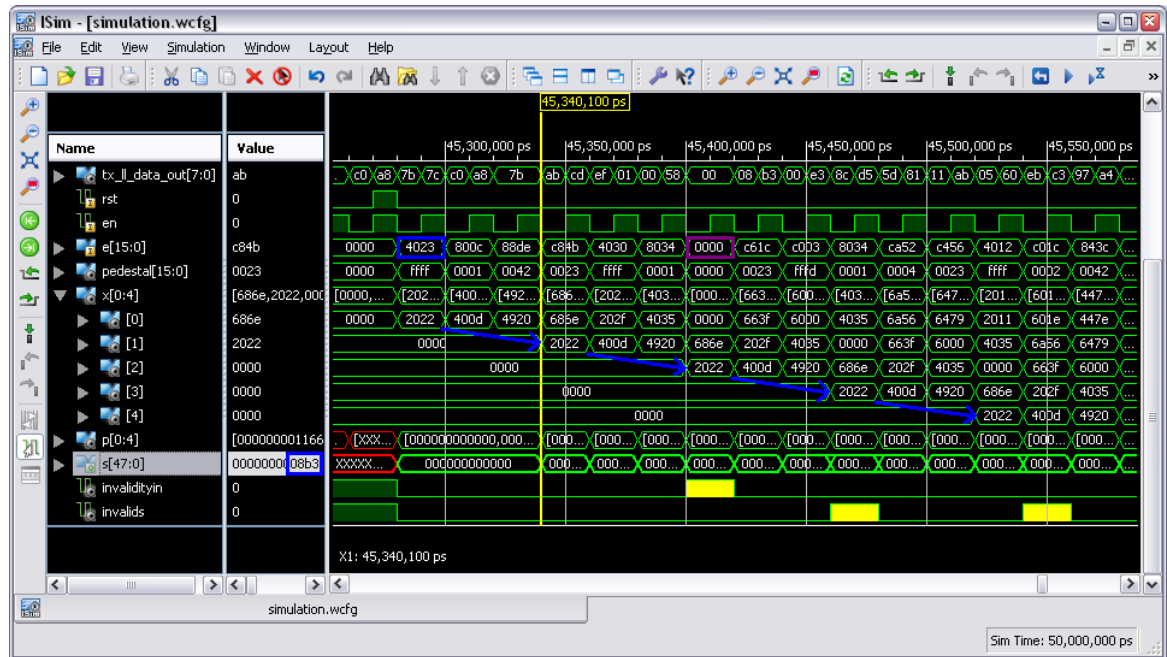
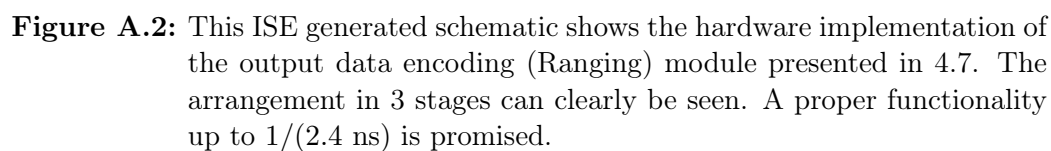


Figure A.1: The simulation of the Multi channel Filter unit illustrates the summation of a test value 4023 representing the value of 23 with low gain (01) that sequentially enters the 5 DSP units (indicated by the blue flashes) after the pedestal correction of -1 (ffff). A second test value is highlighted purple representing an invalid data sample. It is first marked by `invalidityin` and as entering the calculation 5 times, influencing the five concerning results marked by `invalids`.

A.2 Schematic of the Output Data Encoding Module

The schematic generated by ISE illustrates the three stages of LUTs required to perform the aimed output data encoding.



A.3 Global Design Parameters

In the following the source code of the configuration file `my_definition.vhd` containing all design parameters is given.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 package mytypes is
6
7   --BEGIN function declarations
8   -- external: calculate # of bits to encode number
9   function log2ceil(arg : positive) return natural;
10  -- external: select max of arg1 and arg2
11  function sel_abits(arg1, arg2 : positive) return positive;
12
13  -- internal: set # of bytes of calibration packet
14  function sel_coeff_bytes return signed;
15  --END function declarations
16
17  --BEGIN clocking declarations: filter frequency
18  constant clk_mult : positive := 7; -- clock multiplier for PLL with 100MHz clk input
19  constant clk_div : positive := 2; -- clock divider for PLL with 100MHz clk input
20  -- filter frequency = 100MHz * clk_mult / clk_div
21  --END clocking declarations
22
23  --BEGIN LCD declarations
24  constant milisec : integer := 50000; -- 50MHz: 50k == 1 ms
25  -- SIMULATION:
26  -- constant milisec : integer := 5;
27
28  constant DISPLAYDELAY : positive := 125*milisec;
29  --END LCD declaration
30
31  --BEGIN Filter declarations
32  -- fundamental filter settings
33  constant CHANNELS : positive := 4; -- number of channels for 1 DSP-set
34  constant DEPTH : positive := 5; -- number of DSPs in 1 DSP-set (= filter depth)
35  -- output formatting (ranging)
36  constant SHIFT : positive := 2; -- 3 in normal case
37  constant LOWESTBIT : positive := 1; -- LSB of Sum of Filter result: 0 < LSB < 47-15-3*SHIFT
38
39  constant PED_SIZE : positive := 16; -- multiple of 8! -- bit width of pedestal value in
40  LUT
41  constant A_I_SIZE : positive := 24; -- multiple of 8! -- bit width of ai coefficients
42  in LUT
43
44  constant DSPDELAY : positive := 3; -- delay caused by the DSP multiplication
45  constant GAIN_COUNT : positive := 4; -- number of possible valid gain values plus 1 for
46  invalid gain
47
48  constant GAIN_BIT : positive := 15; -- bit position of upper gain bit in sample (in
49  input value)
50  constant X_GAIN_BIT : positive := 14; -- bit position of upper gain bit in sample (in
51  calculation)
52  constant E_BIT : positive := 11; -- max 11
53
54  -- McFilt Manager
55  constant E_delay : positive := 1; -- number of registers introduced on fifo input side
56  constant S_delay : positive := 1; -- number of registers introduced on fifo output
57  side
58
59  -- derived constants
60  constant CHANNEL_WIDTH : positive := log2ceil(CHANNELS); -- # of bits to encrypt
61  CHANNELS bitwise
62  constant GAIN_WIDTH : positive := log2ceil(GAIN_COUNT); -- # of bits to address gain
63  values
64  constant GAIN_BIT : positive := GAIN_BIT - GAIN_WIDTH + 1; -- bit position of lower
65  gain bit (in input value)
66  constant X_GAIN_BIT : positive := X_GAIN_BIT - GAIN_WIDTH + 1; -- bit position of lower
67  gain bit (in calculation)
68  --END Filter declarations
69
70  --BEGIN UDP manager declarations
71  constant calib_signature : std_logic_vector(7 downto 0) := x"FF";
72  constant mcfilt_signature : std_logic_vector(7 downto 0) := x"12";
73  constant max_frame_length : integer := 1600;
74  --END UDP manager declarations
75
76  --BEGIN RAM management declarations
77  constant ram_data_start : unsigned(7 downto 0) := x"08"; --@ in RAM where data begins
78  constant ram_coeff_bytes : signed(10 downto 0) := sel_coeff_bytes; -- # of calibration
79  bytes
80  constant ram_E_samples : signed(10 downto 0) := to_signed(40,11); -- # of samples for
81  filter
82  --END RAM management declarations
83
84  --BEGIN SIMULATION declarations
85  type tEnergies is array(1 to 10) of std_logic_vector(15 downto 0);

```

```

74 type tDATA is array(0 to CHANNELS-1) of tEnergies;
--predefined samples: 4 channels à 10 samples
76 constant DATA_init_values : tDATA :=
(
78   (x"4023", x"4030", x"C003", x"4012", x"4007", x"800a", x"c023", x"C01c", x"400a", x"
      400b"),
      (x"800c", x"8034", x"8034", x"c01c", x"802a", x"400a", x"4003", x"402e", x"801b", x"
      8035"),
80   (x"88de", x"0000", x"ca52", x"843c", x"c3d7", x"c2de", x"c53c", x"c29d", x"89be", x"
      ca3c"),
      (x"c84b", x"c61c", x"C456", x"40a3", x"c5cb", x"8a35", x"446f", x"81e5", x"421c", x"
      c2b7")
82 );

84 type tIPheader is array(1 to 42) of std_logic_vector(7 downto 0);
constant IPheader_data : tIPheader :=
86 (
      x"AA", x"BB", x"CC", x"DD", x"EE", x"FF",-- DST MAC
88   x"00", x"26", x"18", x"F0", x"55", x"E0",-- SRC MAC
      x"08", x"00",-- Protocol: IP
90   x"45", x"00", x"00", x"6C",--Version, IHL, TOS, Total Length
      x"12", x"12", x"40", x"00",--Ident, Flags, Fragment
92   x"40", x"11", x"B0", x"26",--TTL, Protocol: UDP, Checksum
      x"C0", x"A8", x"7B", x"7B",--IP SRC
94   x"C0", x"A8", x"7B", x"7C",--IP DST
      x"00", x"00", x"00", x"00", x"00", x"58", x"DC", x"53"--UDP header: SRC, DST, Length,
      Checksum
96 );

98 constant IPheader_calib : tIPheader :=
(
100   x"AA", x"BB", x"CC", x"DD", x"EE", x"FF",-- DST MAC
      x"00", x"26", x"18", x"F0", x"55", x"E0",-- SRC MAC
102   x"08", x"00",-- Protocol: IP
      x"45", x"00", x"02", x"3C",--Version, IHL, TOS, Total Length
104   x"12", x"ff", x"40", x"00",--Ident, Flags, Fragment
      x"40", x"11", x"ad", x"69",--TTL, Protocol: UDP, Checksum
106   x"C0", x"A8", x"7B", x"7B",--IP SRC
      x"C0", x"A8", x"7B", x"7C",--IP DST
108   x"00", x"00", x"00", x"00", x"02", x"28", x"4f", x"bc"--UDP header: SRC, DST, Length,
      Checksum
110 );

-----
112 --
113 -- bitwise definition of coefftable:
114 --
115 -- predefined coefficients: 4 channels à 3 gains (gain 4 = everything 0)
116 --
-----

118 type tfullcoeffset is array (1 to 3) of std_logic_vector(PED_SIZE + DEPTH*A_I_SIZE - 1
downto 0);
120 type tfullcoefftable is array (0 to CHANNELS-1) of tfullcoeffset;
122 constant coeff_init_values : tfullcoefftable :=
(
124   (
--
126     pedestal & a1 & a2 & a3 & a4 & a5
      x"FFFF" & x"000004" & x"fffff" & x"000012" & x"000055" & x"000083",--channel 0, gain
      1
      x"FFFE" & x"000003" & x"fffffd" & x"0000d2" & x"000035" & x"0000c3",--channel 0, gain
      2
128     x"FFFD" & x"000002" & x"fffffe" & x"000072" & x"000015" & x"000013" --channel 0, gain
      3
    ),
130   (
      x"0000" & x"000006" & x"fffcf" & x"000042" & x"000025" & x"000021",--channel 1, gain
      1
132     x"0001" & x"000005" & x"fffaf" & x"000012" & x"000045" & x"000023",--channel 1, gain
      2
      x"0002" & x"000001" & x"fffbf" & x"000008" & x"000035" & x"000013" --channel 1, gain
      3
134   ),
136   (
      x"0003" & x"000045" & x"fff1f" & x"000012" & x"0000c5" & x"00002e",--channel 2, gain
      1
      x"0042" & x"000034" & x"fff2f" & x"000023" & x"0000a5" & x"00002d",--channel 2, gain
      2
138     x"0004" & x"000023" & x"fff3f" & x"000021" & x"0000b5" & x"00002b" --channel 2, gain
      3
    ),
140   (
      x"0046" & x"000045" & x"fff4f" & x"000066" & x"000012" & x"00001a",--channel 3, gain
      1
142     x"0076" & x"000024" & x"fffc3" & x"000052" & x"000035" & x"00001b",--channel 3, gain
      2
      x"0023" & x"000011" & x"fff26" & x"000041" & x"000045" & x"00001c" --channel 3, gain
      3
144   )
146 );
--END SIMULATION declarations
148 end mytypes;

```

```

150 package body mytypes is
151
152 -- calculate the number of bits required to encrypt arg binarily
153 function log2ceil(arg : positive) return natural is
154   variable tmp : positive;
155   variable log : natural;
156 begin
157   tmp := 1;
158   log := 0;
159   -- satisfy synthesis tools: prevent warnings about loop iterations
160   if arg > 1 then
161     while arg > tmp loop
162       tmp := tmp * 2;
163       log := log + 1;
164     end loop;
165   end if;
166   return log;
167 end;
168
169 -- select the maximum of 2 arguments
170 function sel_abits(arg1, arg2: positive) return positive is
171 begin
172   if arg1 < arg2 then
173     return arg2;
174   else
175     return arg1;
176   end if;
177 end;
178
179 -- set the number of bytes used for the storage of the coefficients
180 function sel_coeff_bytes return signed is
181 begin
182   assert CHANNELS * GAIN_COUNT < 64
183   report "Too many CHANNELS or GAINS. Only CHANNELS * GAIN_COUNT < 64 allowed!"
184   severity failure;
185
186   if CHANNELS * GAIN_COUNT < 32 then
187     return to_signed((PED_SIZE + A_I_SIZE*DEPTH)/8 * 32,11);
188   elsif CHANNELS * GAIN_COUNT < 64 then
189     return to_signed((PED_SIZE + A_I_SIZE*DEPTH)/8 * 64,11);
190   else
191     return to_signed(0,11);
192   end if;
193 end;
194
195 end mytypes;

```

A.4 Data and Calibration sample packets for 4 channels

In the following the test data sample for four channels is given (table A.1) as well as the calibration packet used throughout the design testing (table A.2).

Table A.1: The UDP/IP sample data packet consists of 10 energy samples for 4 interleaved channels. The order of channels is defined by 0, 1, 2, 3, 0, 1, 2, 3 and so on.

UDP/IP header	AA BB CC DD EE FF 00 26 18 F0 55 E0 08 00 45 00 00 6C 12 12 40 00 40 11 B0 26 C0 A8 7B 7B C0 A8 7B 7C 00 00 00 00 00 58 DC 53
sample part	40 23 80 0C 88 DE C8 4B 40 30 80 34 00 00 C6 1C C0 03 80 34 CA 52 C4 56 40 12 C0 1C 84 3C 40 A3 40 07 80 2A C3 D7 C5 CB 80 0A 40 0A C2 DE 8A 35 C0 23 40 03 C5 3C 44 6F C0 1C 40 2E C2 9D 81 E5 40 0A 80 1B 89 BE 42 1C 40 0B 80 35 CA 3C C2 B7

Table A.2: The UDP/IP Calibration data packet for 4 channels contains all filter parameters but in a special bit order. Spare places have to be left to be conform with the arrangement in SRLC32E slices that require a multiplicity of 8 channels.

UDP/IP header	AA BB CC DD EE FF 00 26 18 F0 55 E0 08 00 45 00 02 3C 12 FF 40 00 40 11 AD 69 C0 A8 7B 7B C0 A8 7B 7C 00 00 00 00 02 28 4F BC
<i>Ped</i> data part	00 00 00 0E 00 00 00 0E 00 00 00 0E 00 00 00 0E 00 00 00 0E 00 00 00 0E 00 00 00 0E 00 00 00 0E 00 00 00 0E 00 00 04 6E 00 00 00 CE 00 00 00 4E 00 00 00 0E 00 00 08 6E 00 00 86 E6 00 00 42 8A
a_1 data part	00 02 20 00 00 0C 40 00 00 04 80 00 00 00 00 00 00 66 62 00 00 28 0C 00 00 CA A4
a_2 data part	00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE EE 00 00 EE 04 00 00 E2 06 00 00 EC C8 00 00 E8 A0 00 00 EE E2 00 00 EE EA 00 00 AE EE 00 00 6E E6
a_3 data part	00 04 00 00 00 EC 20 00 00 28 0C 00 00 4E 42 00 00 00 80 00 00 20 00 00 00 6E 66 00 00 80 0C
a_4 data part	00 E0 00 00 00 28 24 00 00 C4 4A 00 00 86 E8 00 00 00 00 00 00 EC EE 00 00 02 00 00 00 EC EE
a_5 data part	00 06 00 00 00 04 00 00 6E 00 00 00 80 E8 00 00 0E E0 00 00 06 80 00 00 CA 6E 00 00 EC 4E

Glossary

ADC Analog-to-Digital Converter. 1, 11–13, 16–19, 36

ASIC Application Specific Integrated Circuit. 10–12, 23, 24

BCID Bunch Crossing Identification. 13

CERN Conseil Européen pour la Recherche Nucléaire: European Organisation for Nuclear Research. 3, 14

CRC-32 Cyclic Redundancy Check of 32 bits. 40, 47

DAQ Data Acquisition System. 7, 14

DCM Digital Clock Manager. 26–29, 49, 70

DLL Delay-Locked Loop. 26, 27

DSP Digital Signal Processor. 7, 14, 24, 36, 56–59, 61, 64, 76, 89

EF Event Filter. 14

FEB Front End Board. 7, 9, 12–14, 16, 36, 75, 76, 78, 79, 81

FIFO First In - First Out, a data storage device. 11, 48, 69, 70, 78

FIR filter Finite Impulse Response filter. 1, 2, 16, 17, 36, 47, 53, 54, 56, 60, 61, 72, 78, 81, 82

FPGA Field Programmable Gate Array. 1, 2, 16, 23, 24, 26, 28, 29, 31–33, 36, 39, 44, 46, 47, 53, 54, 56, 63, 69, 70, 75, 76, 78, 79, 81

GbE Gigabit Ethernet. 39, 47–49, 53, 54, 57, 59, 64, 69, 70, 79

GMII Gigabit Media Independent Interface. 48

HDL Hardware Description Language. 23, 31

HL-LHC High Luminosity Large Hadron Collider. 1, 2, 15, 82

IP Internet Protocol. 39–43, 46, 50, 52, 54, 64, 69

IPv4 Internet Protocol, version 4. 40–43

- IPv6** Internet Protocol, version 6. 41
- L0 trigger** Level 0 trigger. 16, 77, 78
- L1 trigger** Level 1 trigger. 7, 9–16, 78, 79
- L2 trigger** Level 2 trigger. 14
- LAr calorimeter** Liquid Argon calorimeter. 1, 5–9, 13, 15, 16, 47, 75, 76, 78, 81, 82
- LHC** Large Hadron Collider. 1–5, 10, 12, 15, 57, 58, 75, 77, 79, 82
- LSB** Layer Sum Board. 12, 13
- LUT** Look-up Table. 24, 25, 29, 53, 56–60, 63, 68, 71–73, 75, 76, 89
- MAC** Media Access Control. 29, 40, 47–49
- McFilt** Multi channel Filter. 54, 55, 61, 62, 64, 69–71, 75–78, 81, 89
- OSI model** Open Systems Interconnection model. 39, 40
- PLD** Programmable Logic Device. 23, 31
- PLL** Phase-Locked Loop. 26–29, 49, 70
- RAM** Random-Access Memory. 29, 30, 34, 35, 47, 50, 52–54, 57, 63, 64, 66, 67, 69, 70, 78
- ROB** Readout Buffer. 16, 39, 47, 79
- ROD** Readout Driver. 7, 13, 14, 16, 39, 47, 53, 54, 70, 75, 76, 78, 79, 81
- SCA** Switched-Capacitor Array. 10, 11, 15
- SCAC** Switched-Capacitor Array Controller. 11
- SRLC32E** 32 bit Shift Register Look-Up Table with Clock Enable. 63–65
- TP** Twisted Pair: type of wiring for the purpose of canceling out electromagnetic interference. 39
- UDP** User Datagram Protocol. 39, 40, 43, 46, 48, 50, 52, 54, 64, 65, 69, 81
- UTP** Unshielded Twisted Pair: type of wiring for the purpose of canceling out electromagnetic interference. 39
- VCO** Voltage Controlled Oscillator. 27
- VHDL** Very high speed integrated circuit Hardware Description Language. 31–33, 44, 51, 63, 72

Bibliography

- [1] L. Evans and P. Bryant, *LHC Machine*, Journal of Instrumentation **3** (2008) no. 08, S08001.
- [2] P. Lebrun, *Commissioning and First Operation of the Large Hadron Collider (LHC)*. *oai:cds.cern.ch:1284331*, Proceedings of the ICEC 23: 23rd International Cryogenic Engineering Conference. Aug, 2010. CERN-ATS-2010-178.
- [3] K. Fuchsberger et al., *Operational experience during initial beam commissioning of the LHC*, Proceedings of the IPAC 10: 1st International Particle Accelerator Conference. May, 2010. CERN-ATS-2010-181.
- [4] H. F. Schopper and R.-D. Heuer, *LEP: The lord of the collider rings at CERN 1980-2000. The making, operation and legacy of the world's largest scientific instrument*. Springer, Berlin, 2009.
- [5] The ALICE Collaboration, K. Aamodt et al., *The ALICE Experiment at the CERN LHC*, Journal of Instrumentation **3** (2008) no. 08, S08002.
- [6] The LHCb Collaboration, A. Augusto et al., *The LHCb Detector at the LHC*, Journal of Instrumentation **3** (2008) no. 08, S08005.
- [7] The CMS Collaboration, S. Chatrchyan et al., *The CMS Experiment at the CERN LHC*, Journal of Instrumentation **3** (2008) no. 08, S08004.
- [8] The ATLAS Collaboration, G. Aad et al., *The ATLAS Experiment at the CERN Large Hadron Collider*, Journal of Instrumentation **3** (2008) no. 08, S08003.
- [9] The LHCf Collaboration, O. Adriani et al., *The LHCf detector at the CERN Large Hadron Collider*, Journal of Instrumentation **3** (2008) no. 08, S08006.
- [10] The TOTEM Collaboration, G. Anelli et al., *The TOTEM Experiment at the CERN Large Hadron Collider*, Journal of Instrumentation **3** (2008) no. 08, S08007.
- [11] The ATLAS Collaboration, G. Aad et al., *Expected Performance of the ATLAS Experiment - Detector, Trigger and Physics*. CERN, 2009. CERN-OPEN-2008-020.
- [12] The ATLAS Collaboration, *ATLAS liquid-argon calorimeter: Technical Design Report*. Technical Design Report ATLAS. CERN, Geneva, 1996.

- [13] N. J. Buchanan et al., *Design and implementation of the Front End Board for the readout of the ATLAS liquid argon calorimeters*, Journal of Instrumentation **3** (2008) no. 03, P03004.
- [14] A. Straessner, *Development of new readout electronics for the ATLAS LAr Calorimeter at the sLHC*, Proceedings of the TWEPP-09: Topical Workshop on Electronics for Particle Physics. Oct, 2009. ATL-LARG-PROC-2009-016.
- [15] The Liquid Argon Back End Electronics Collaboration, J. Ban et al., *ATLAS liquid argon calorimeter back end electronics*, Journal of Instrumentation **2** (2007) no. 06, P06002.
- [16] N. J. Buchanan et al., *Radiation qualification of the front-end electronics for the readout of the ATLAS liquid argon calorimeters*, Journal of Instrumentation **3** (2008) no. 10, P10005.
- [17] A. Straessner, *LHC State of the Art and News*, Proceedings of the Vulcano Workshop 2010 Frontier Objects in Astrophysics and Particle Physics. Jul, 2010. ATL-GEN-PROC-2010-005.
- [18] A. Kielburg-Jeka and S. Stärz, *A Readout Driver for the ATLAS LAr Calorimeter at a High Luminosity LHC*, Proceedings of the TWEPP-10: Topical Workshop on Electronics for Particle Physics 2010. Sep, 2010. ATL-LARG-PROC-2010-012.
- [19] B. G. Lawrence R. Rabiner, *Theory and Application of Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey, first ed., 1975.
- [20] A. Papoulis, *Signal Analysis*. The McGraw-Hill Companies, first ed., 1977.
- [21] W. E. Cleland and E. G. Stern, *Signal Processing Considerations for Liquid Ionization Calorimeters in a High Rate Environment*, Nucl. Instrum. Methods Phys. Res., A **338** (1994) 467–497.
- [22] T. Reinhardt, *private communication*, 2010.
- [23] Laboratoire d’Annecy-Le-Vieux de Physique des Particules, Computing division, *The Physics ROD 64x PU algorithm*, 2002.
- [24] R. Schnell, *Untersuchungen zu First-Level-Datenauslesestrukturen für den Siliziumstreifendetektor im Mikro-Vertex-Detektor von PANDA*, Master’s thesis, TU Dresden, 2009.
- [25] Ignaciomella, *Switch_box.svg*, Wikipedia, the free encyclopedia, 29 April 2007. http://en.wikipedia.org/wiki/File:Switch_box.svg.
- [26] P. Zainalabedin Navabi, *Embedded Core Design with FPGA*. The McGraw-Hill Companies, first ed., 2007.
- [27] Xilinx®, *Virtex-5 FPGA User Guide*. http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

- [28] D. Banerjee, *PLL Performance, Simulation, and Design Handbook*. National Semiconductor, fourth ed., 2006.
<http://www.national.com/appinfo/wireless/files/deansbook4.pdf>.
- [29] Xilinx®, *Virtex-5 Family Overview*.
http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.
- [30] Xilinx®, *LogiCORE Block Memory Generator*. http://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen_ds512.pdf.
- [31] Xilinx®, *Virtex-5 FPGA XtremeDSP Design Considerations, User Guide UG193*.
http://www.xilinx.com/support/documentation/user_guides/ug193.pdf.
- [32] J. Rech, *Ethernet - Technologien und Protokolle für die Computervernetzung*. Heise Zeitschriften Verlag, GmbH & Co, KG, 2., aktualisierte und überarbeitete ed., 2007.
- [33] *Computing the Internet checksum*, Tech. Rep. RFC 1071, Network Working Group, BBN Laboratories, Sep, 1988.
- [34] Xilinx®, *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper, User Guide UG340*. http://www.xilinx.com/support/documentation/ip_documentation/v5_emac_gsg340.pdf.
- [35] Xilinx®, *Virtex-5 Libraries Guide for Schematic Designs*.
<http://www.xilinx.com/itp/xilinx92/books/docs/v5lsc/v5lsc.pdf>.
- [36] H. Chen, *Update of LAr/L1Calo Latency Estimate*, Sep, 2010.
- [37] G. Broijmans et al., *ATLAS Liquid Argon Calorimeter Upgrade Plans*, Aug, 2010. ATL-LARG-INT-2010-009.
- [38] H. Chen, *ATLAS LAr calorimeters readout electronics upgrade R&D for sLHC*, Proceedings of the CALOR 2010: 14th International Conference On Calorimetry In High Energy Physics. May, 2010. ATL-LARG-PROC-2010-010.
- [39] S. D. Rindani, *Strong gauge boson scattering at the LHC*, Oct, 2009. arXiv:0910.5068.
- [40] The ATLAS Collaboration, *ATLAS sensitivity prospects to W' and Z' in the decay channels $W' \rightarrow l\nu$ and $Z' \rightarrow l+l-$ at $\sqrt{s}=7$ TeV*, . ATL-PHYS-PUB-2010-007.
- [41] U. Baur, T. Plehn, and D. L. Rainwater, *Determining the Higgs Boson Self Coupling at Hadron Colliders*, Phys. Rev. D **67** (2002) 33003. 28 p. hep-ph/0211224. CERN-TH-2002-327. DESY-02-183. DESY-2002-183. FERMILAB-PUB-2002-199-T. UB-HET-2002-05.

Acknowledgement

The author wants to thank all persons who contributed to this work directly or indirectly. Without this support this work would not have been possible.

Special thanks go to the supervising director, Jun.-Prof. Dr. Arno Straessner. The provided subject was interesting, up to date, challenging and in the same time trend-setting. He was always willing to revise the script with the actor and to go through important details when required.

Special thanks go to the First Technical Assistant Unit, Andreas Meyer, Andreas Glatte, and Andy Kielburg-Jeka, for the technical support offered at any time.

Special thanks go to the Second Technical Assistant Unit, Thomas Preusser and Martin Zabel, for transfer of profound knowledge and for patient implementation of tiny, but very important details with the actor.

Special thanks go to the Catering Team, Konrad Müller, Sven Krönke, and others. The culinary rest periods and coffee breaks were refreshing and offered the chance to receive new input and new ideas.

Special thanks go to the Unit of Public Relations, especially to André Dankert, Peter Drechsel, Ulrik Günther, Christin Hille, and Philipp Anger. The promotion of the subject, as well as the treatment of cross-project issues, and the subject unrelated exchange during regular social events provided the needed compensation to gain new power for the main undertaking.

Special thanks go to the Event Hosing Team, Haike and Indy, for the location shots in Aachen.

Special thanks go to the Funding Agency, my parents. Without the financial support, a realisation of this project would never have been possible in this extensiveness.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Steffen Stärz
Dresden, Dezember 2010

