Activity Report

July 2007 - August 2008

European Organisation for Nuclear Research (CERN) 13 Month Placement

Author:
Jo Inge Buskenes
mail @ joinge.net

January 8, 2009

French Conseil Européen pour la Recherche Nucléaire

Supervisor:
Hans Muller
Hans.Muller @ cern.ch

Gjøvik University College

Supervisor:
Are Strandlie
are.strandlie @ hig.no

Created with LaTeX

Abstract

"Imagination is more important than knowledge"
Albert Einstein

I attended the technical student program at CERN (The European Organization for Nuclear Research) from July 2007 to August 2008. More specifically, I worked with the PHOS (ALICE Photon Spectrometer) detector, which is one of the sub-detectors in ALICE (A Large Ion Collider Experiment). With this report I will try to describe in essence the various activities I was involved in and present them in a step-by-step approach. The following topics will be covered:

- The building blocks of our world. My personal introduction to particle physics.
- General description of CERN, LHC (Large Hadron Collider), ALICE and PHOS.
- Digital design on the TRU (Trigger Region Unit). Card description and design concept analysis.
- Digital design on the TOR (Trigger OR). Design concept analysis.
- Writing C++ utilities for the RCUs. Description and analysis of offline control utility and TRU register scanner.
- Appendixes which mainly contains information about my experiences working on digital design in Linux, how to interface the electronics through embedded devices (called DCS cards, Detector Control System), and how to communicate with the TRU design through dedicated registers. The goal here is to provide hints and tips that might come in handy for people aiming to resume my work.

I assume that the reader possesses basic knowledge regarding electronics and digital circuits. However, most of the introductory material should be easy to understand for anyone interested. I will try to refrain myself from including any code in this document, and rather explain principles with extensive use of graphics. If the reader wants the actual lines of code, look up how to get it in 1.8 (Additional Resources). I strive to keep my code well commented and self-explanatory, and should complement the report when more details are desired.

All figures and pictures in this report are to be considered the result of my own work unless otherwise noted. This applies also for written material, where the sources of my knowledge will be mentioned and can be looked up in the bibliography.

Jo Inge Buskenes

I hereby assure that every effort has been made in order to provide an accurate and credible report.

Hans Muller

I have read this report and can verify that the contents of this report seems credible.

Contents

. . . the recipe for everything . . .
. . . hidden dimensions . . .
. . . the God particle . . .
. . . antimatter . . .
. . . dark matter . . .
. . . dark energy . . .

Interested, anyone?

1 Introduction

Have you ever looked up to the skies wondering what role you play in the big scheme of things? What you were actually looking at, and what a miracle it seems to be that the world as we know it exists at all? And has it ever occurred to you that through us, the universe is in fact aware of itself? How come this marvel ever came to be?

Fig. 1.1 - What crazy scheme are we involved in? [imgEAS]

By studying how the Universe behaves, it quickly becomes apparent that the further we go back in time the simpler the Universe was. This is how we will proceed; by travelling back in time as far as our knowledge can take us, I will give you a insight about what we already know about the building blocks of our world. Then I will move on to something even more interesting; what we *do not* know. And as you might suspect, there is a place where several of these questions might be answered very soon. Keep tuned for more information!

1.1 Particle Physics 101

. . . "Where "nature laws" does not apply, but are created". . .

The most commonly accepted theory we have as to "how it all started" is called the big bang, and refers a massive explosion that occurred 13.7 billion years ago sourcing an immensely energy-rich substance which was later bound to form the universe as we know it. Our knowledge goes back as far as to a few millionth of a second after the blast. Then the substance had cooled enough for the particles known as quarks, bosons and leptons to form.

Quarks are particles of matter, and make up protons, neutrons and a large number of lesser-known particles. Bosons are carriers of force, representing - respectively - the photon, carrier of electromagnetism; Z and W^+/W^- bosons, carrier of the weak force; gluons, carrier of the strong force; and the yet to be observed Higgs boson, a particle suspected responsible for imbuing other particles with mass[1]. Leptons are particles immune to strong forces; the most famous being the electron.

The strong force acts on gluons and quarks and ultimately makes up matter, the weak force changes particles and atoms from one type to another (nuclear reactions), and electromagnetism acts on electrically charged particles. The bosons have a fleeting existence as they only exist whilst carrying information from one matter particle to another.

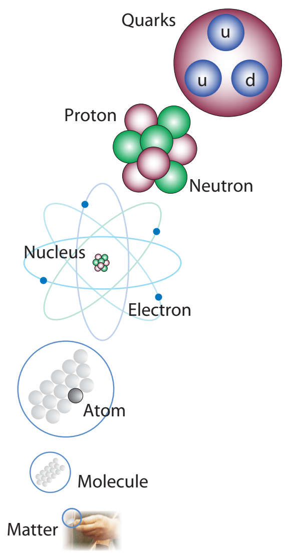


Fig. 1.2 - The Building Blocks of matter [docCERN]

Now, let us go from crash course in particle physics to the unknown!

[1]Why is gravity not mentioned? See 1.2 (paragraph 4).

1.2 *The Missing Pieces*

Did you know that the Sun contains enough matter to fuel this solarsystem with energy for billions of years to come? What does this mean? Matter must be some confined form of energy? What is the mechanisms behind this phenomena? How come energy have so many faces (this is one of the questions the ALICE experiment seek to find the answer to - see [docALICE])?

When scientists started to research the subatomic realm they discovered that all particles had an antiparticle, a counterpart with opposite electric charge. For every particle they created, an antiparticle was created. The conditions under which they were created are expected to be more or less the same as straight after Big Bang, so it is reasonable to think that there was an equal amount of matter and antimatter back then. Finally, if a particle and its antiparticle ever comes in contact with each other they will both be completely annihilated. Good thing that did not happen you might think. Yes, but *why* is this? And where is all the antimatter today?

There are billions of stars in our galaxy, and there are billions of galaxies in the Universe. These contain "everything that we know", such as stars and planets. However, astronomers and physicists have found that all of this only accounts for roughly 4% of the total Universe! The remaining part consists of dark matter (26%) and dark energy (70%). Dark matter are "invisible", but are proven to be there because of the huge gravitational fields "it" sets up. Dark energy is a mysterious substance that seems to be associated with the vacuum in space. It is perfectly distributed both in space and time, and causes the Universe to expand in an ever increasing pace. However, what this "dark stuff" actually is remains a mystery.

You may have noticed when we talked about bosons that gravity was not included. This is because the most widely recognised model for particle physics today - the *Standard Model* - does not explain how gravity fits into the picture. However, there is a model that also includes gravity - namely the *String Theory*. The only problem is, it also implies that in addition to the 4 "known" dimensions (space+time) there must also be 6 additional spatial dimensions! They are said to be "curled up" and so small that we can not see them. The question is *how* small they are, and can their existence ever be proven? And, if they exist, would it be possible to "move" between these dimensions?

And finally, we mentioned gravity. What *is* that? Any idea why matter has mass? Did you know that the the weight of a proton is 100 times that of the sum of the weight of the particles it is composed of? Where does all the remaining mass come from? Another particle? Indeed, this is what the scientists believe. They call it the Higgs particle, the God particle, but it remains to be observed.

So, what is next? *[2]*

[2] Based on [intCHP, Science ▶ All left side pages], [intHBU].

1.3 CERN

...“The Coolest Place in the Universe“...

CERN (The European Organisation for Nuclear Research) is located on the Franco-Swiss boarder not far from Geneva. It was founded in 1954 by a handful of pioneers [3] aiming to share the cost related to nuclear physics facilities and unite European scientists. Since then it has grown to become one of the world's largest and most respected centres for scientific research. It is run by 20 European member states[4], but 35 non-member countries are also involved in the programs in various ways. There are around 2500 people currently employed here, and in addition there are around 580 universities and institutes representing 85 different nationalities that use CERNs facilities. Roughly half of the worlds particle physicists come to CERN for their research.

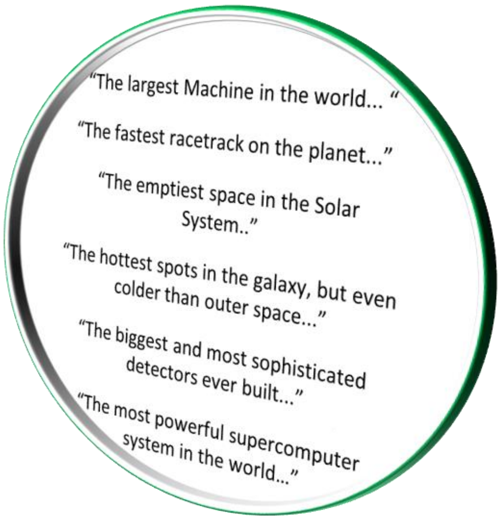
Fig. 1.3 - The CERN Globe [imgCGL]

The work done here has brought forth 5 Nobel Prizes, groundbreaking new knowledge about the inner workings of our world and the birth of the World Wide Web! Furthermore the technology developed here can be applied in other fields more relevant to our daily lives, for example in medical equipment better capable of keeping us healthy. All of this fits very well with the CERN philosophy:

- Leading the research into the fundamentals of our Universe.
- Advancing the frontiers of technology.
- Bringing nations worldwide together through science.
- Educating the scientists of tomorrow.

So what is really going on here? Take a look at some of the citations in fig. 1.4 and make a wild guess. They are from the CERN webpage. What is in the air, or maybe, in the ground?

[[5]]



"The largest Machine in the world... "

"The fastest racetrack on the planet..."

"The emptiest space in the Solar System.."

"The hottest spots in the galaxy, but even colder than outer space..."

"The biggest and most sophisticated detectors ever built..."

"The most powerful supercomputer system in the world..."

Fig. 1.4 - Some Citations from the CERN Web Page [intCHP] [sofPPT]

[3]Raoul Dautry (France), Pierre Auger (France), Lew Kowarski (France), Eduardo Amaldi (Italy), Niels Bohr (Denmark) et al.

[4]Norway is one of the member states.

[5]Based on [intCHP, About us ► All left side pages].

1.4 LHC

. . . "One Ring to Rule them All". . .

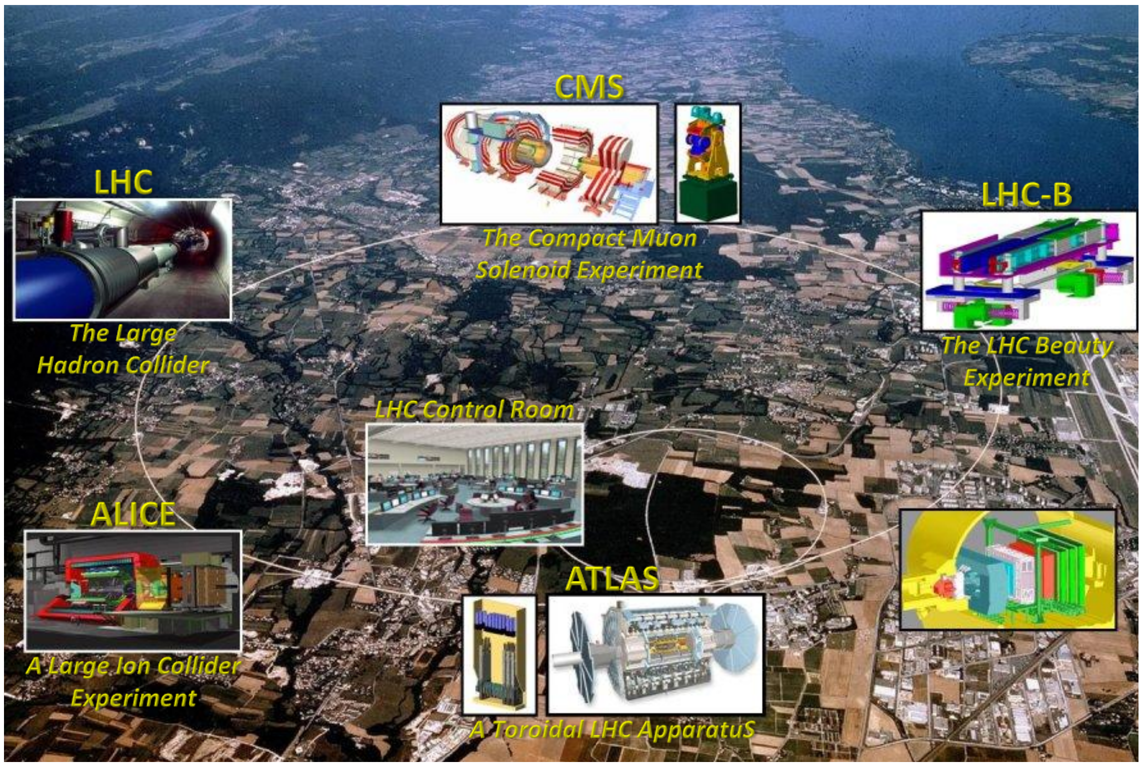

CMS
The Compact Muon Solenoid Experiment
LHC
The Large Hadron Collider
LHC-B
The LHC Beauty Experiment
LHC Control Room
ALICE
A Large Ion Collider Experiment
ATLAS
A Toroidal LHC ApparatuS

Fig. 1.5 - The Large Hadron Collider Complex [intPRESS, Custom Tags]

Take a look at fig. 1.5 and you will see a plane photo of CERN and its surroundings. In the upper left corner you can see lake Geneva, and in the right corner you can catch a glimpse of the Geneva airport. The CERN main site is located just under the illustration of the ATLAS (A Toroidal LHCApparatus) detector, with the southern part of the site sticking out from the upper right corner of the mentioned illustration. The white circles indicates the location of various particle accelerator tunnels, the biggest one is called the The Large Hadron Collider (LHC).

The LHC is - as mentioned - a huge circular particle accelerator situated 50-150m underground. It measures 27 km in circumference, and along with its detectors represents the largest scientific instrument ever made by human kind. The aim is to accelerate particles up to very high speeds and produce head-on collisions such that they decompose into more fundamental particles. While quite a few have been found (some were mentioned in 1.1), there are probably more to be detected. The LHC is a lean and mean particle hunt machine - the biggest one to date. Particles, beware.

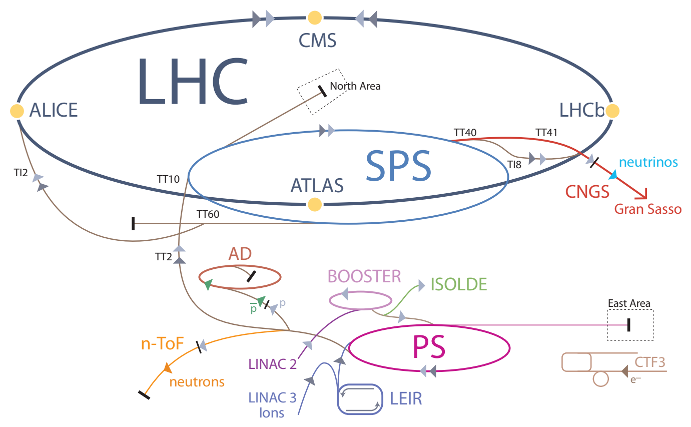


Fig. 1.6 - The Accelerator Complex [docLHCUG]

So what is the crazy looking fig. 1.6 representing? Well, when particles are to be accelerated to energies like at CERN several accelerators must be utilised. The figure shows the network of accelerators currently present at CERN (I advise the reader not to try to remember all the abbreviations). The particles start their journey in some small accelerators at 99.9998% the speed of light at an energy of 450GeV (eV = electron-volt). However, while no particle can travel faster than the speed of light, there is no limit to the amount of energy the particle can attain. When the particles ultimately are being injected into the LHC they are kicked up to 99.999999% the speed of light and an energy of 7TeV (the beam now contains enough energy to melt 500kg copper[6]). You might see that when measuring energy in particle physics, it is more appropriate to talk about electron-volts than the speed of the particle.

So what happens to particles when they are boosted to higher and higher energies? If we imagine ourselves travelling from the eV-range to the TeV-range, we pass through many distinct "landscapes". In the eV world matter follows the rules of chemistry and solid state electronics. Increasing the energy into the MeV-range will provoke nuclear reactions (atoms decompose into protons and neutrons (electrons are leptons), and we are now talking about roughly the energy that matter has attained in the center of the Sun. Moving into the GeV-range we will find that the matter decomposes even further into bosons, quarks and leptons - as mentioned earlier in 1.1. But what happens when the energy is increased into the TeV-range? Will it help us "fill in the blanks" in the knowledge of our world as mentioned in 1.2? Will we witness completely unknown phenomenas?

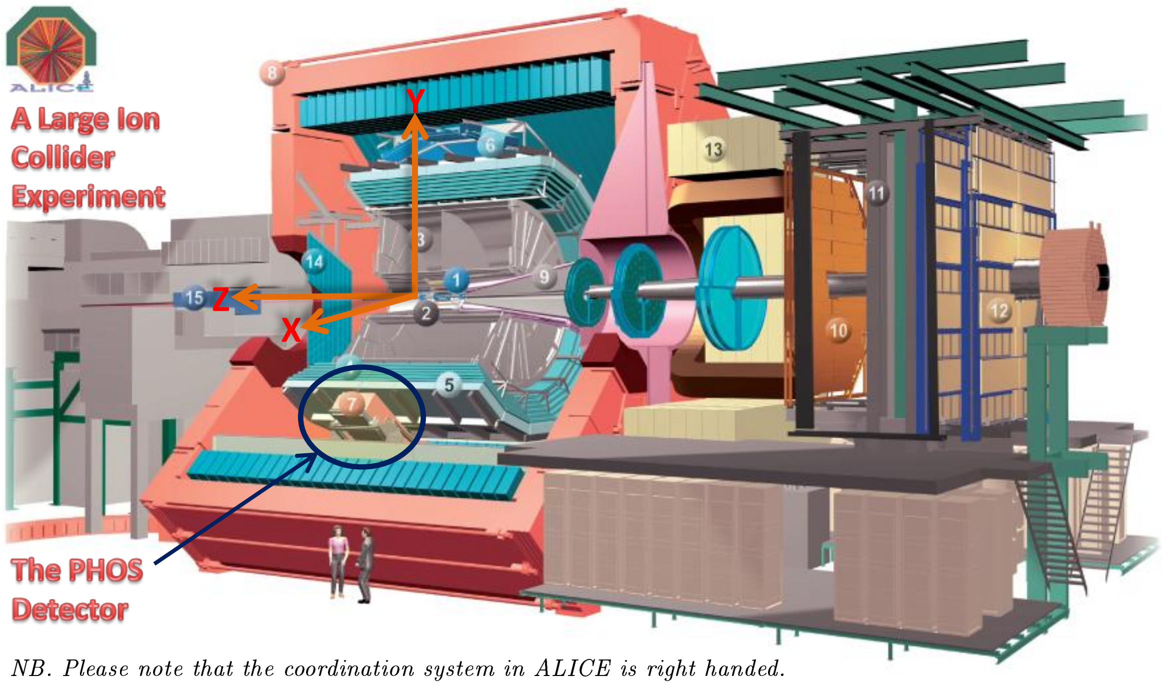
A vital part of the puzzle is missing; accelerating particles up to the high energies is to no use unless we also have the necessary equipment to detect what happens as the particles collide. From fig. 1.5 and fig. 1.6 you can see the 4 main detectors on the LHC ring: ATLAS, CMS (The Compact Muon Solenoid Experiment), LHCb (The Large Hadron Collider beauty Experiment) and ALICE. I worked with the latter, so let us have a look at it!

[[7]]

[6]For more fascinating facts, take a look at appendix E.
[7]Based on [docLHCUG], [magSCAM].

1.5 ALICE

. . . "in Wonderland". . .



NB. Please note that the coordination system in ALICE is right handed.

Fig. 1.7 - The ALICE Detector [imgALIDET, Custom Tags]

The ALICE detector is one of the four large detectors situated in the LHC accelerator ring and is mainly dedicated to the study of heavy-ion collisions. It weighs around 10 000 tons and measures 16m high and 26m long. It is composed of 18 subdetectors which together provide the necessary sensory equipment to track and identify the tens of thousands of particle tracks produced in each heavy-ion collision. In order to get a decent rate of interesting events the ALICE detector must be capable of processing 8000 collisions per second (see 1.5.1).

When the particles collide inside the ALICE detector the collision point will be 100 000 times hotter than the centre of the Sun (around 2000 billion °C). What happens to matter at these temperatures? The current theory of the strong interaction[8] predicts that at these immense energies quarks and gluons should no longer be confined inside composite particles, but rather exist in a "free" state of matter known as quark-gluon plasma. It is believed that this state existed just a few millionth of a second after the Big Bang, so we are sort of "looking back in time" with the ALICE detector. By studying how the quark-gluon plasma reverts back to a confined state of matter we can expect to find the answers to how the mechanism behind the confinement work, and to why the weight of a proton is 100 times that of the sum of the weight of the particles it is composed of (see 1.2).

[8]Called quantum chromodynamics.

As previously mentioned bunches of particles collide inside the ALICE detector at a rate of thousands of times per second. For only one of these collisions a vast amount of data is produced by all the various sensors. We need some sort of filter; some sort of mechanism that selects only the most interesting collisions (or events, as it is popularly referred to) to send to the computers for further processing. For this purpose, we filter the data with the use of a triggersystem consisting of specialised electronics, which - due its huge parallel potential - takes a look at all the data and creates ”triggers“ when the data has certain interesting properties.

1.5.1 The Trigger System

To filter out interesting data from what is not, ALICE utilises a triggersystem designed in 3 levels:

- **Level 0**. This trigger decision must arrive at the Central Trigger Processor (CTP) within a total latency of 1.2μs. With all setup times and cable delays this leaves barely 500ns for electronic processing, which means that the decision can only be based on a very simple criteria; like when the signal of a particle is bigger than the average.

- **Level 1**. If the Level0 decision is validated, the data of the pre-selected particle is searched by more refined algorithms that need a little bit more time. These are mostly based on physics criterias, like detection of two neighboring particles from a decay of a single one. Due to the increase of complexity in the trigger calculation, the total timeframe is 6.5μs.

- **Level 2**. Given that the data were accepted by the CTP by both level-0 and level-1 from several detectors for the same event, the Level2 trigger is asserted as a request for the electronics to start sending data. Later on in the process there might also come Level2 messages which rejects or accepts the event based on decisions by the CTP. The timing requirements are even more relaxed here, typically the Level2 trigger comes within 90μs and the “accept”/“reject” message within 500μs.

Now why am I mentioning this? Well, I was working on the Level0 trigger in one of the ALICE subdetectors - the Photon Spectrometer (PHOS, see 1.6). You can see the PHOS detector in fig. 1.7, it is situated on the bottom of the ALICE detector.

[[9]]

[9]Based on [docPHOSUM, page 85], [intIETNS] and [MyMind].

1.6 *PHOS*

. . . "Let there be light". . .

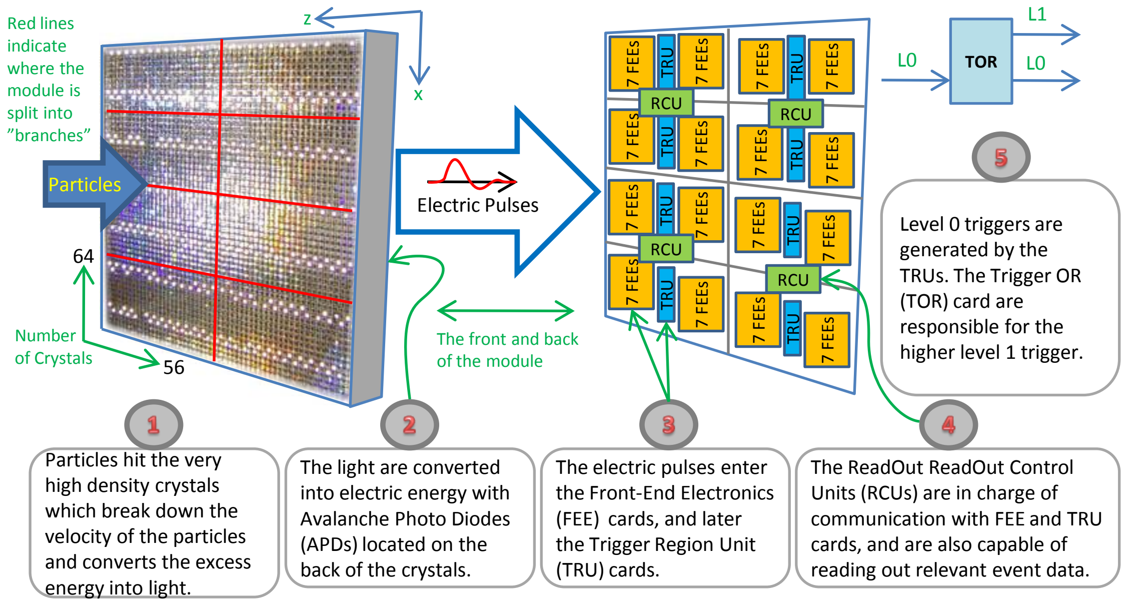

Red lines indicate where the module is split into "branches"
Particles
z
x
64
Number of Crystals
56
Electric Pulses
The front and back of the module
7 FEEs
TRU
RCU
L0
TOR
L1
L0
1
Particles hit the very high density crystals which break down the velocity of the particles and converts the excess energy into light.
2
The light are converted into electric energy with Avalanche Photo Diodes (APDs) located on the back of the crystals.
3
The electric pulses enter the Front-End Electronics (FEE) cards, and later the Trigger Region Unit (TRU) cards.
4
The ReadOut ReadOut Control Units (RCUs) are in charge of communication with FEE and TRU cards, and are also capable of reading out relevant event data.
5
Level 0 triggers are generated by the TRUs. The Trigger OR (TOR) card are responsible for the higher level 1 trigger.

Fig. 1.8 - The Basic Buildup of a PHOS Module [sofPPT] [sofPSP]

PHOS is - as the name implies - a detector designed to measure the energy of light particles (photons). As I have tried to illustrate in fig. 1.8, a PHOS module is composed of two "layers" - a set of crystals and the electronics necessary to process the data coming from the crystals. Only one PHOS module is depicted, but in the end there will be a total of 5 similar modules[10]. For each module there are a total of 3584 (56x64) high density lead-tungsten crystals ($PbWO_4$), each weighing around 1kg. But what are the electronics there for?

The clock in the LHC oscillates at 40MHz, which we have to use when the data are to to be digitised. If we had sampled the data from the crystals directly we would have had to deal with a steady stream of `40MHz * 12 [bits/sample] * 3584 [crystals]` ≈ `215GB/s` per module, assuming a sample resolution of 12 bits. It should be obvious that there is no way this amount of data can be processed in realtime in a computer.

The electronics layer is composed of 112 cards called FEEs (Front-End Electronics), 8 TRUs (Trigger Region Unit) and 4 RCUs (Readout Control Unit). Each RCU controls 2 "branches" of TRUs and FEEs, where each branch may be considered an independent group of cards. Basically, the function of the electronics layer is fully defined with one branch, but scaled up to 8 in total per PHOS module. Because of this, I will from now on speak about the operation of a single branch and not the entire module.

[10] As of today (3.July 2008), there are 1 PHOS module installed in ALICE and two more are being commissioned.

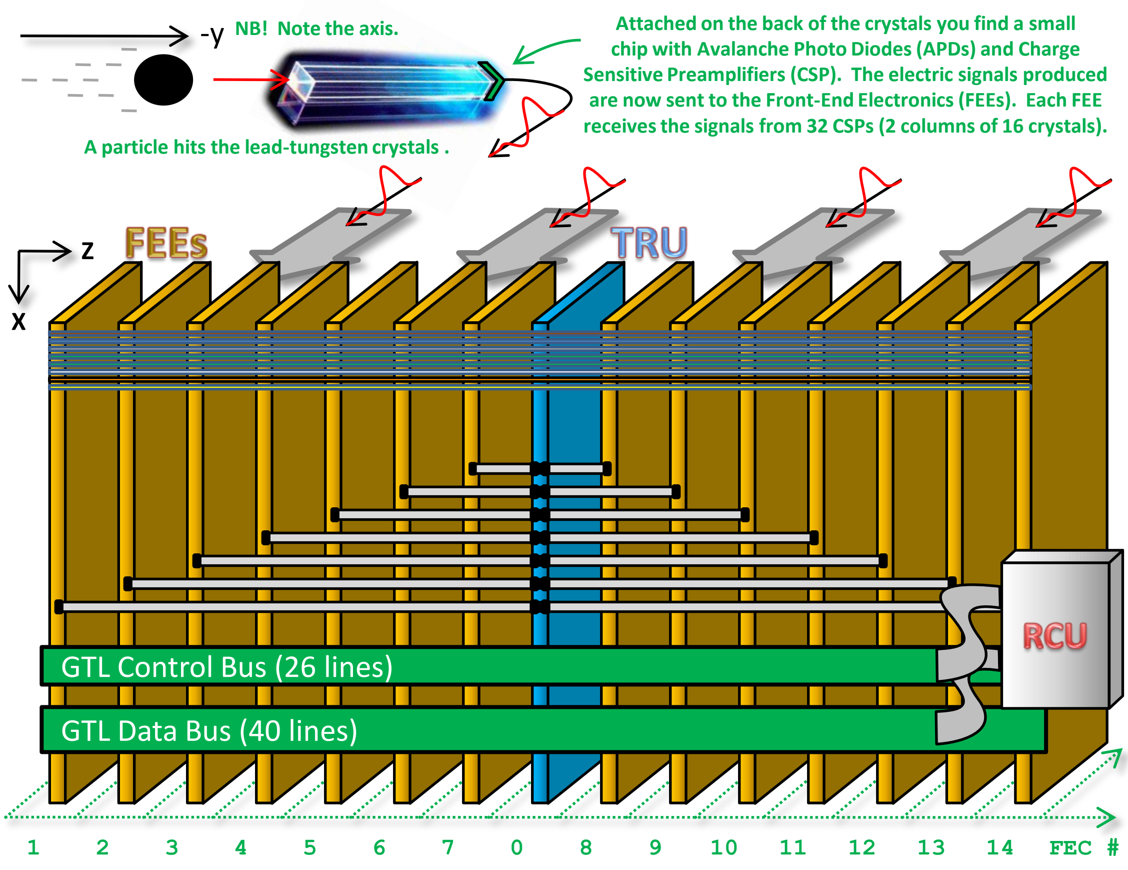

-y
NB! Note the axis.
Attached on the back of the crystals you find a small chip with Avalanche Photo Diodes (APDs) and Charge Sensitive Preamplifiers (CSP). The electric signals produced are now sent to the Front-End Electronics (FEEs). Each FEE receives the signals from 32 CSPs (2 columns of 16 crystals).
A particle hits the lead-tungsten crystals .
z
x
FEEs
TRU
RCU
GTL Control Bus (26 lines)
GTL Data Bus (40 lines)
1 2 3 4 5 6 7 0 8 9 10 11 12 13 14 FEC #

Fig. 1.9 - 1 of the 8 "Branches" in the Electronics Layer [sofPPT]

Fig. 1.9 shows one of the mentioned branches of cards, consisting of 14 FEE cards and 1 TRU. Some of the kinetic energy from the particles produced in a collision in ALICE is converted into photons in the PHOS crystals. The light is transformed into electric energy with Avalanche Photo Diodes (APDs)[11] which is in turn amplified with Charge Sensitive Preamplifiers (CSPs)[12]. Each FEE is connected to 32 CSPs (2 columns of 16 crystals, for geometrical positions see tab. C.1, right side). The FEE has two main tasks: (^{1}) Sampling and recording high-gain and low-gain versions of these signals and store them in ALTRO chips[13] where they can be read out by the RCU via the GTL (Gunning Transceiver Logic) bus, and (^{2}) perform an analog sum of 2x2 patches of crystals (for a total of 8 sums per FEE), shape them to a 100ns semi-Gaussian pulse and send them to the TRU (we call these signals FastOR) via. LVDS (Low Voltage Differential Signaling) cables (the red ones in fig. 1.9). With this data the TRU can decide whether a Level0 trigger is to be generated.

[11]The APDs have an efficiency of 80% - which means that they effectively converts 80% of the energy in the light into electric energy!

[12]The combination of APDs and CSPs in PHOS can sense bunches of photons down as low as 10 photons, which is an extremely low light strength.

[13]See [docALTRORC] and [docPHOSUM] for more information.

1.7 *Report Structure*

So what can you expect to find in the upcoming pages? In order of appearance, these are the chapters:

- **Introduction**. The one you are currently reading. I started off with a short introduction to particle physics and questions which still remains unanswered. Then I said a few words about CERN, the new particle accelerator LHC, the heavy-ion detector ALICE and one of its subdetectors - PHOS. In the last section of this introduction I will explain where additional resources can be found if desired.

- **Trigger Region Unit (TRU)**. Presenting the card with a general description and some history, before I proceed to discuss digital design where ADC (Analog to Digital Converter) interfacing and resource utilisation will play a vital role.

- **Trigger OR (TOR)**. After a short description I will proceed to describe some mapping problems and present a design suggestion for the communication protocol between the TOR and TRUs.

- **Readout Control Unit (RCU)**. Two programs will be described in this chapter, namely the TRU Register Scanner and the Offline Control Utility.

- **Conclusion**. Some final thoughts made in the aftermath of my stay.

As you can see, I will order my work into chapters representing the card I was working on (when the general introduction (this one) and conclusion is excluded). These chapters will start with a small introduction before the various topics are presented. These topics may be concluded separately, or they might be temporarily concluded while pending the final conclusion of the chapter. In between the introduction and conclusion the structure will vary depending on what topics are discussed, but the topics discussed in these chapters will generally be related to *firmware and software design*.

During my work I frequently encountered situations where necessary documentation seemed absent. However, from studying code and asking around I have grown to understand some of these poorly documented aspects. Throughout this report I have tried to blend this knowledge in between related topics, but some of it just does not well with these chapters. This and other relevant information may be looked up in the appendixes:

- **Other Activities**. As the title indicates this appendix will act as a container for other "activites" I was involved in and documented, but was not of such a nature that they would blend in well with the main body of this report. Keywords here are Linux experiences, BASH scripting for working with backups, DCS interfacing, miscellaneous work with mapping and phase adjustment tutorial for ADC clocks.

- **Chronograms**. Some chronograms from ModelSim showing a simulation in progress.

- **TRU Communication**. Some scripts you may use when trying to write and read the TRU registers via. the RCU will be presented, along with a complete list of the current TRU registers.

- **Fascinating Facts**. For those interested.

- **TRU Logic Usage**. For those interested.

I hope that I succeeded in supplying a document which is digestable, even though it contains quite technical material. The paradox is that whilst you are trying to keep it simple and the prerequisited knowledge at a minimum, you can not leave out too many details as this will render the documentation useless. The question should not be whether it is perfectly fulfilling the criteria of some template, but rather if the message was adequately presented.

When writing a report like this it is impossible not to use some abbreviated words, not to mention impractical as often they represent the names of electronic components in our project - and I find that short names are easier to remember than long names. However, unless they are thoroughly discussed somewhere locally in the report and not mentioned elsewhere - you will find them in the glossary.

This report is best read as a `pdf` file. This way you get to enjoy bookmark navigation and hyperlinks functionality. However, the colorcodes should make it pretty straightforward to read the paper version aswell. Green coloured words are links to the glossary, red coloured words are references to the bibliography. In the bibliography you will find an extensive list of sources and where to get additional information.

This is the extract of my year at CERN, happy reading.

1.8 *Additional Resources*

In my report I will describe and discuss several programs and documents. These will be made available for the reader, either by logging into the server mentioned a few paragraphs down or by simply clicking the references in this document (if you are reading the electronic version). If you are reading the paper version you can still find the right files by simply looking up the reference-tag in the document bibliography.

I set the document-server up with `ftp` (File Transfer Protocol)[14]. To log onto it use the following information:

```
Site:      ftp.joinge.net

Username:  iyearn
Password:  forcern
```

You can expect to find the following information on this server:

- Sourcecode for this document.
- Sourcecode for the TRU (most recent).
- Sourcecode for the TRU (initial).
- Sourcecode for DCS Offline Control Utility.
- Sourcecode for TRU Register Scanner.
- Sourcecode for TOR (most recent).
- Sourcecode for TOR (previous versions).
- Datasheets.
- PHOS related documents.
- ChipScope Project Files (for diagnosis).

Each printed copy of this report will also be accompanied with a CD containing the same information as were present on this server at the date of printing.

[14] If you are using Windows you can can log onto `ftps` with Windows Explorer (the filesystem browser), or if you are using Linux you can use e.g. Konqueroror Nautilus - they all offer drag-and-drop functionality

2 Trigger Region Unit

The Trigger Region Unit (TRU) is the card in charge of calculating the Level0 trigger in PHOS (see 1.5.1 for more details about the triggersystem in ALICE). This is first trigger signal to be produced in the ALICE detector after a particle collision, and indicates whether the data recorded of the event might be worth looking more thoroughly into. In the case of PHOS, we need to produce a Level0 trigger whenever the energy absorbed by the crystals exceeds a certain threshold.

It is important that the Level0 signal arrives at the CTP in the ALICE detector no later than 800ns prior to the time of a particle collision. Due to propagation delay in the electronics the FastOR (The FastOR pulse is the analog sum of a 2x2 matrix of crystals) signals produced by the FEE cards arrives at the TRU ≈150ns after impact time. This means that the TRU must generate the Level0 trigger signal in less than 450ns (leaving around 200ns for propagation time from the TRU to the CTP). This might sound hard, but since the TRU has resources highly capable of performing parallel computations it is in fact quite feasible.

The TRU is the card I have spent most of my time with. My work related to the TRU can be split into two parts:

- Digital design for the Virtex 2 Pro FPGA (Field Programmable Gate Array) present on the TRU.
- Hardware testing, modifications and assembly.

This report will not deal with the latter, as I consider the former far more important. In order to provide an "gentle start" on the topic of digital design I will start this chapter with a general description of the TRU and present some of the key features of the Virtex 2 Pro FPGA. In the section dealing with digital design I will describe the parts of the design I was involved in and thoroughly discuss the various considerations that was made along the way. Finally, the chapter will be wrapped up with a conclusion where I will suggest what future work should be carried out.

2.1 Introduction

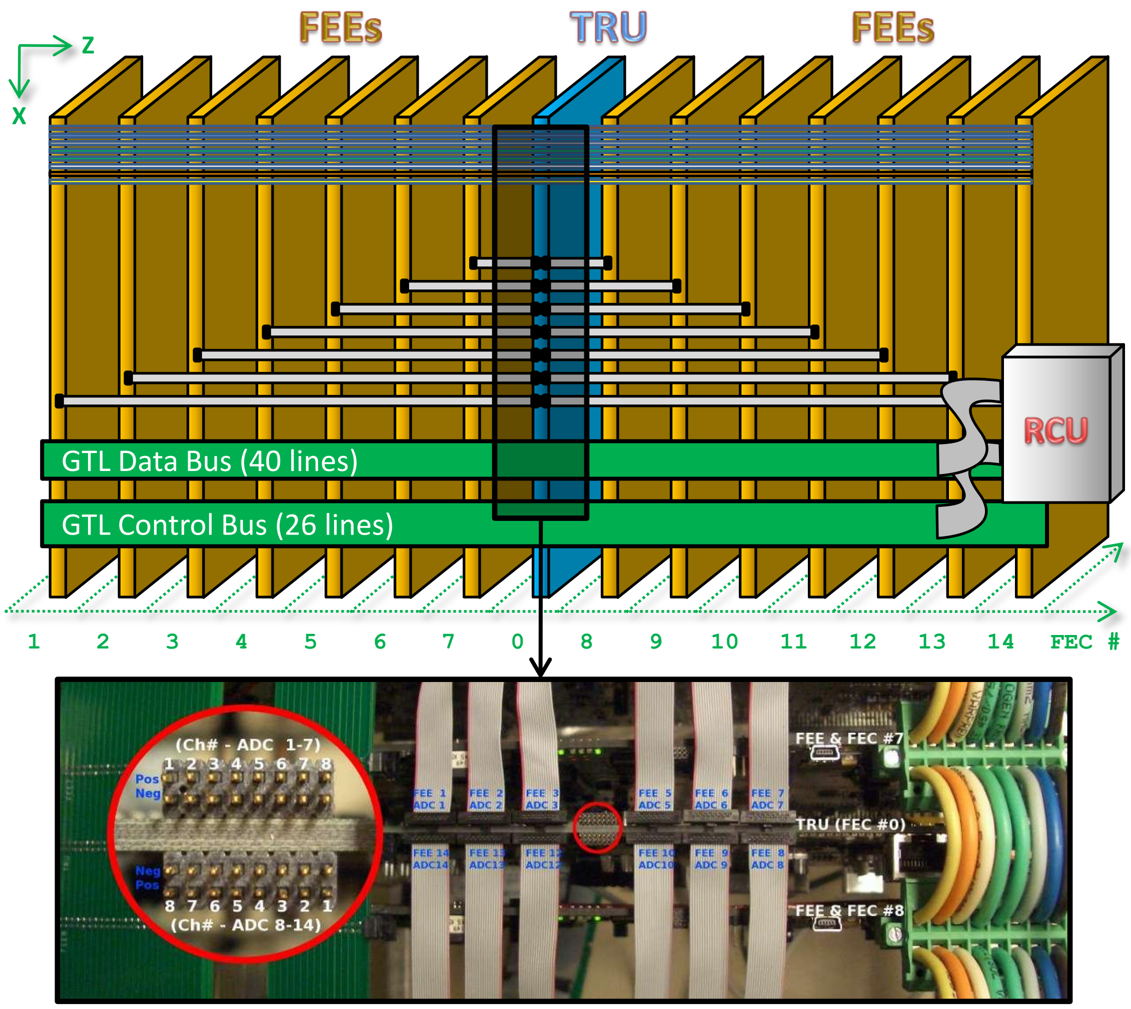


Fig. 2.1 - TRU Position in a Branch of Front-End Cards [sofPPT] [sofGMP]

In fig. 2.1 I tried to illustrate the environment in which the TRU operates. As you can see, it is situated in the middle of 14 FEE cards and receives 8 FastOR signals from each of them (the white cables in the illustration). The TRU is Front-End Card (FEC) number 0, and the FEE cards receives addresses along the ALICE x-axis. The mapping is such that equally indexed FEEs and TRU ADCs are connected together, as shown in the picture at the bottom of fig. 2.1.

The RCU communicates with the TRU over the GTL bus, either via a low speed serial communication protocol (called SlowControl) which utilises the GTL *Control* bus[1], or via a high speed parallel communication protocol (called ALTRO - ALICE ReadOut) which utilises the GTL *Data* bus.

[1]At the time of writing - 20.June 2008 - the SlowControl protocol is not operational for the TRU.

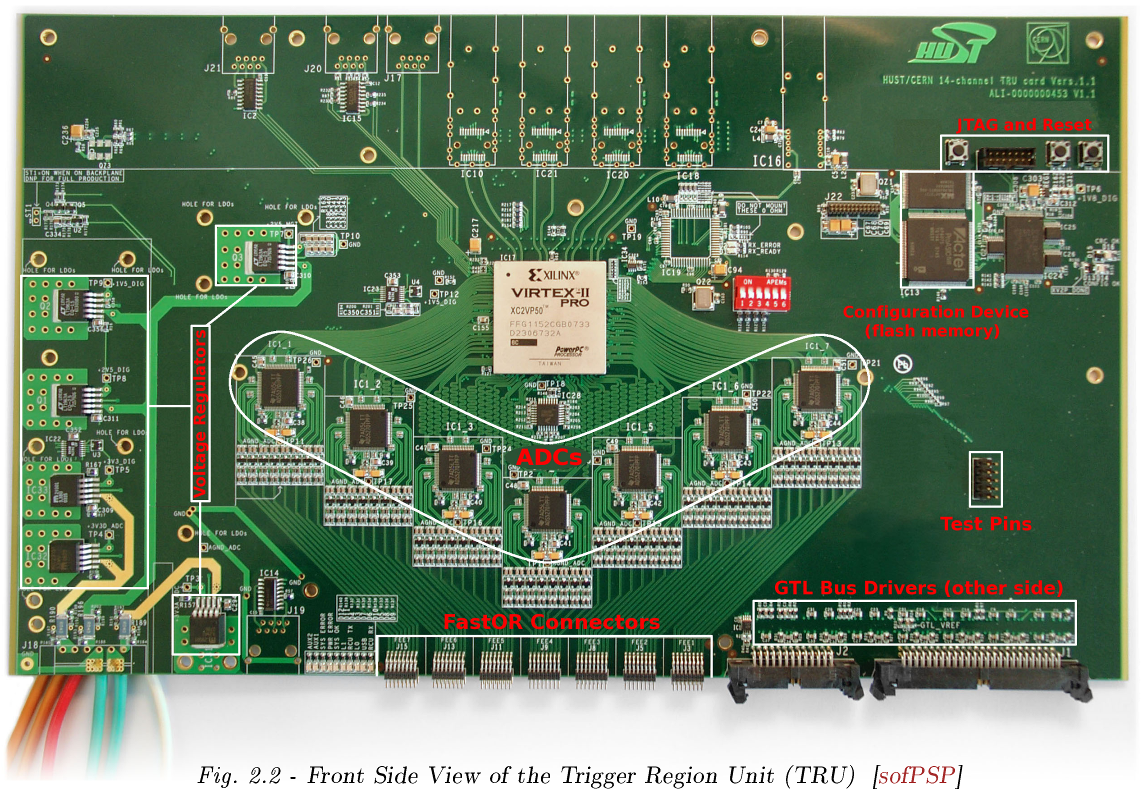

JTAG and Reset
Configuration Device (flash memory)
Voltage Regulators
ADCs
Test Pins
GTL Bus Drivers (other side)
FastOR Connectors

Fig. 2.2 - Front Side View of the Trigger Region Unit (TRU) [sofPSP]

In fig. 2.2 you can see the front-side of one of our new TRUs, which is a 12 layer printed circuit board (PCB) card. In the lower left corner you can see the power cables. Going from left to right you first have the two orange lines carrying 4.2V digital voltage, then a red cable carrying 3.3V digital voltage followed by two green cables with electrical ground, and finally a the white cable with 4.0V analog voltage.

In fig. 2.2 I have tagged what I believe is the most important components. A bunch of voltage regulators on the left side of the card will ensure a stable and accurate power supply to the electric components. The ADCs are there to digitise the analog FastOR signals coming from the FEE cards. The GTL drivers are located in the bottom right corner as close to the GTL bus as possible to prevent the capacitive load caused by the card to become higher than necessary. In the upper right corner you will find the logic needed for Virtex 2 Pro FPGA configuration storage and JTAG (Joint Test Action Group) connectivity. Finally, in the center of the board we have the FPGA - where all the magic happens.

2.1.1 Technical Data

I believe that a list of components present on the TRU and the links to where you can find the datasheets might come in handy, so I supply this information for TRU design v1.1 [docTRU1.1] (tab. 2.5)[2]:

Component	Type	Design Unit	DataSheet
Virtex 2 Pro FPGA	XC2VP50	IC17	[docV2PDS] [docV2PUG]
ADCs	ADS5270IPFP	IC1_[1,7],IC2_[1,7]	[docADC]
Configurable Flash Memory	XCF32P	IC24	[docXCF32P]
ProASIC Flash Memory	APA075	IC13	[docAPA075]
Flash memory	MX29LV640DBTI-90G	IC12	[docMX29LV]
GTL Drivers	GTL16612DGG	IC3-IC9	[docGTL]
Voltage Regulator	LT1963A-1.5	Q2	[docVR15V]
Voltage Regulator	LT1963A-2.5	Q1,Q3	[docVR25V]
Voltage Regulator	* LTC1844ES5-1.8	REG1	[docVR18V]
Voltage Regulator	MIC29301-33BU	IC33	[docVR33BU]
Voltage Regulator	MIC29501-33WU	IC31	[docVR33WU1]
Voltage Regulator	MIC20151-33WU	IC32	[docVR33WU2]
Power Supply Transistor	SO2222A	Q4,Q5	[docSO2222]
Temperature Sensor	* MAX6627	IC34	[docMAX6627]
Temperature Sensor	AD7417	IC22,IC23	[docAD7417]
Clock Distribution Chip	MPC9109	IC28	[docMPC9109]
LVDS Driver ALT_SCLK	* SN65LVDS101	IC1	[docLVDS101]
LVDS Driver J21,J19	* SN65LVDS31	IC2,IC14	[docLVDS31]
LVDS Receiver J20	* SN65LVDS32	IC15	[docLVDS32]
MUX/DEMUX	ADG774	IC25,IC26,IC27	[docADG774]
Transceiver Rocket IO	** V23818-K305-V17	IC16	
Receiver Rocket IO	** HDMP-1034	IC19	
Transceiver Rocket IO	** HFBR-5720AL	IC10,IC18,IC20,IC21	

** Component names from Design Sheet v1.1 [docTRU1.1]. I was not able to verify them.*
*** Rocket IO components were never installed.*

Tab. 2.1 - TRU Components

For my work there is in fact only the two topmost components that we really need to know a bit about. The FPGA is a pretty complex device that enables us to program the logic that tie all the other components together in an unified way. In the field of programming such devices - digital design - quite a bit of experience is required in order to be able to produce anything with a decent level of quality. A good starting point is to know the specifications of the FPGA we are working with, so we will take a closer look at the Virtex 2 Pro XC2VP50 FPGA in the next subsection.

[2]Some of the component names may differ from what is specified in the design sheet as these are the actual components installed on the board, visually verified.

2.1.2 *Virtex 2 Pro Resources*

From the Virtex 2 Pro datasheet we can dig up the following information (tab. 2.2):

RocketIO Transceiver Blocks	16
PowerPC Processor Blocks	2
18x18 Bit Multiplier Blocks	232
Configurable Logical Blocks (CLBs) providing:	
Slices	23 616
Max DistributedRAM (kb)	738
DCMs	8
Block Select RAM+ featuring:	
18 kbBlocks	232
Max BlockRAM (kb)	4 176
Maximum User IO Pads	852

Tab. 2.2 - Virtex 2 Pro (XC2VP50) Specifications [docV2PDS, page 2]

Now, while it may be a bit overwhelming to get all these numbers thrown at you, it is important to mention them as they represent the boundaries we have to keep clear of when designing the code. Especially important are they in the case of our project and the TRU, as we are struggling to fit all the functions we wish the TRU to have into this FPGA. This will be discussed in appropriate places throughout the report, but first I will walk you through the parameters from tab. 2.2. The subsections will be labelled accordingly.

Rocket IO Blocks *2.1.2.1*

RocketIO is a technology aimed to provide serial communication at very high speeds. In recent FPGAs from Xilinx the RocketIO functionality resides in the silicon fabric and offers excellent performance. However, the technology was introduced with the Virtex 2 Pro and in this generation the latencies were too high for it to be used in the TRU design. The components for RocketIO has therefore never been mounted on the board. [3]

PowerPC Blocks *2.1.2.2*

The *PowerPC Processors* are neat if you want to run a Linux OS in your FPGA. However, there is no need for this on the TRU so they will be left alone.

Multiplier Blocks *2.1.2.3*

The *Multiplier Blocks* are designed to be used with the Block SelectRAM (Read Access Memory) 18kb blocks. It features on-the-fly multiplications of the values stored in a Block SelectRAM block at a low power consumption, which is really nice if you for instance wish to implement digital filters. However, I do not see how we can utilise these in the TRU design as time is a very critical factor. Maybe future designers will find an area of use for them, but for now they will be left out of the design. [4]

[3]Based on conversation with John Evans, Xilinx seminar 28. May 2008.
[4]Based on [docV2PDS, page 58].

Configurable Logic Blocks (CLB) *2.1.2.4*

The *Configurable Logic Blocks* (CLBs) are the main logic resource for implementing sequential as well as combinatorial circuitry. Since it is an extremely important resource to understand when designing the logic for the FPGA, I made a drawing of how these logic blocks are built up in the Virtex 2 Pro (fig. 2.3):

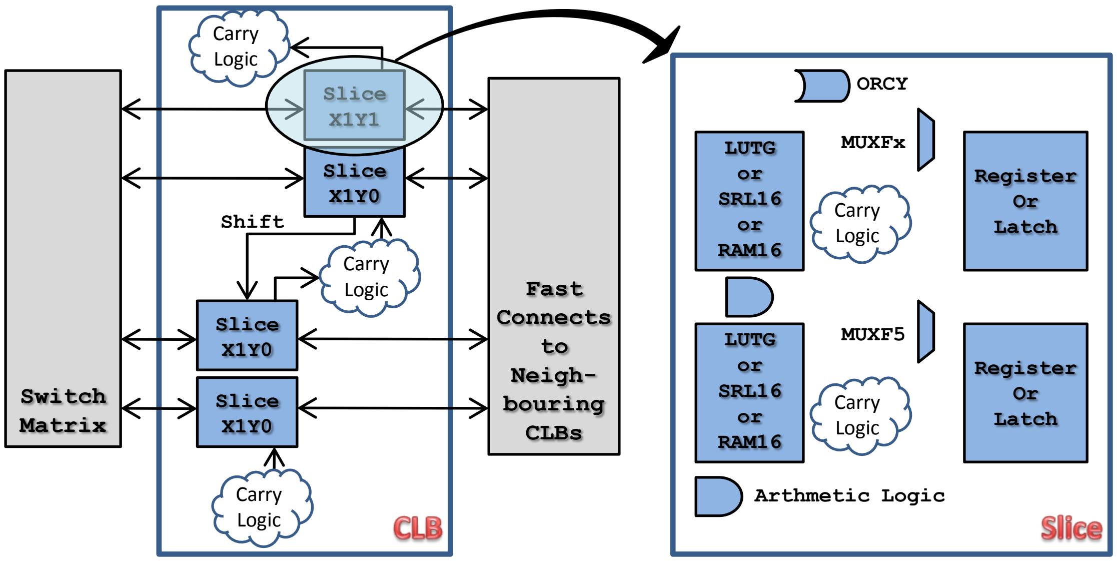


Fig. 2.3 - A Simplified Illustration of the Virtex 2 Configurable Logic Block (CLB)
[docV2PDS, page 45] [sofPPT]

First, have a look the right side of fig. 2.3 and notice what is called a *slice*. A slice is the smallest group of logic in the FPGA, and consists of 2 lookup-tables (LUTs), 2 storage elements, a few multiplexers and a some gates, chains and tri-state buffers. A slice is fully configurable, which means that it can be programmed to realise almost any digital circuitry.

Each lookup table (LUT) has 4 inputs and can be configured either as a function generator, 16 bits of read-access memory (*Distributed SelectRAM*) or a 16 bits shift register (*SRL*). When configured as a function generator it can cover all boolean functions of 4 inputs (by simply looking them up), but with the use of the multiplexers functions for up to 9 inputs can be realised within a single slice. A 2 input multiplexer can be implemented with 1 function generator and some associated multiplexers in the slice, where each additional function generator added will provide 2 extra inputs to the multiplexer.

The Distributed SelectRAM blocks have one address port for synchronous write and - if it is configured as dual-port - one address port for asynchronous read. The extra port will always require the use of an additional LUT, for instance will a 16 bits RAM block with dual port require 2 LUTs as opposed to 1 LUT for the single port version.

When the LUT is configured as a shift register the write will be synchronous, while the values may be read asynchronously with the 4-bit address bus. If a synchronous read is required the storage element may be utilised. This will also improve the system performance since the storage element provides far superior clock-to-out speed.

The storage elements can be configured either to realise a level triggered latch or an edge triggered register. The input data can either be supplied from logic inside or outside of the slice.

The carry logic and shift chains, plus the various multiplexers are used for implementing arithmetical and logical functions, but can alternately be utilised for connecting slices together within a CLB to extend the logic capacity. Each CLB can then provide the following logic:

LookUp Tables, can also be configured as:	8
Distributed SelectRAM	128 bits
Shift Registers	128 bits
FlipFlops	8
Multiplier AND Gates	8
Arithmetic and Carry Chains	2
SumOfProducts (SOP) Chains	2
Tristate buffers	2

Tab. 2.3 - Logic Provided by 1 Configurable Logic Block (CLB) [docV2PDS, page 52]

For most of my work with digital design the upper half of this table seems interesting whilst the lower half does not. I have never encountered a situation were there was a shortage of gates and arithmetical or logical routing, the bottleneck seems always to be related to data storage.

Please note that I just presented a rather simplified image of the CLB. While you may perform "rough calculations" on resource utilisation by referring to the numbers above, the reality might prove to differ since the software used to break up the code and create the FPGA configuration might alter or implement the design in a way you did not expect it to. It might decide that some resources ought to be shared or that the design should be redistributed in order to achieve a better compromise between logic utilisation and performance.

Each CLB is connected to the global routing network through a Switch Matrix and to adjacent CLBs with Fast Interconnect (see fig. 2.3). When necessary, CLBs can be connected together to create functions which can not be created with a single CLB. However, as the complexity of the logic increases so will the performance hit. I have come to believe that good coding style involves breaking down any problem into simple functions which can be implemented with as few slices as possible. *[[5]]*

[5]Based on [docV2PDS, page 45-54].

Digital Clock Managers *2.1.2.5*

The Virtex 2 Pro FPGA has a wide variety of interconnect for routing data and clocksignals[6]. For global clock signals, or for very high speed clocks the global clock routing network should be used. This is because it is the only net which is directly connected to everything in the FPGA, thus providing shared timing models). This network can host a maximum of 16 clocks, but with restrictions. To clarify, I made an illustration (fig. 2.4):

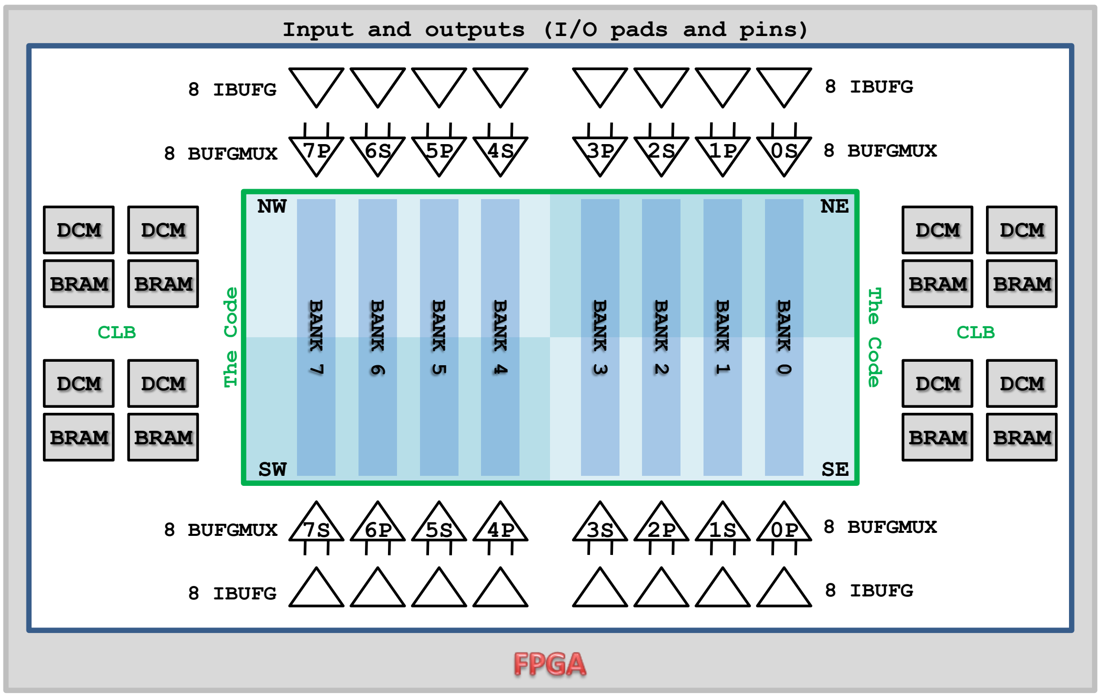


Fig. 2.4 - Virtex 2 Pro Global Clock Distribution [docV2PUG, page 69] [sofPPT]

The global clock network is made up of 8 “banks” or “clockmultiplexer pairs”, each of which contains a primary and secondary clockmultiplexer (or buffer, e.g. BUFGMUX 0S and 0P)[7]. Each multiplexer pair shares routing resources in the same quadrant of the FPGA (NW, SW, NE, SE, according to geometrical direction), so if the primary buffer access the NW quadrant, the secondary buffer may only access the remaining 3 quadrants. The input to the global clock buffers can come from either the global input clock pins (IBUFG)[8], from internal logic or from *Digital Clock Managers* (DCMs, described on the next page).

[6]For a nice hierarchical overview of the Virtex 2 Pro interconnect, refer to [docV2PDS, page 65].
[7]Global clock MUXs can shift glitch-lessly from one clock to another.
[8]Global clock pins can drive DCMs on the same clock edge.

While the global clock network is the best suited for distributing clocks in the design, secondary clocks or high fan-out signals can be routed using a pattern of alternative routing resources ("long lines[9]", "hex lines", "double lines", ..)[10] that results in low skew, but the best results are usually achieved by letting the implementation software handle it single-handedly. When you need to specify some timing criterias use timing constraints.

A *Digital Clock Manager* (DCM) is a self-calibrating and fully digital solution for:

- *Clock distribution.* Source a clock signal to it and let it supply internal logic or external components with high quality clock signals. The fan-out capability is very good and the phase-locked loop (PLL) inside the DCM will assure perfect duty-cycle.
- *Delay compensation.* Use the post-buffer clock signal as feedback to make the DCM "transparent" (clock has same phase before and after DCM).
- *Clock frequency multiplication and division.* Create derived clocks with a wide range of possible frequencies.
- *Coarse-grained clock phase shifting.* Supplies output clocks with 0°, 90°, 180° and 270° phaseshifts, respectively.
- *Fine-grained clock phase shifting.* Provides the ability to on-the-fly adjust the clock phase in increments of `T/256`[11]. *[[12]]*

Block Select RAM *2.1.2.6*

The `BRAM` blocks you see in fig. 2.4 are the 18kb *Block SelectRAM* mentioned in tab. 2.2. It features two synchronous ports where the width is programmable from 1 bit (depth 16 384 bits) to 36 bit wide (depth 512 bits). Each port has an address bus which is independently clocked. It is not as fast as the shallow CLB memory, but will in return provide massive storage space. *[[13]]*

Input and Output Buffers *2.1.2.7*

The *Input and Output Buffers* (IOB) is the interface between the "external" (outside FPGA) and internal logic. For both inputs and outputs a wide range of signalling standards are supported - for both differential and single ended schemes. Optional input delay elements may be used to synchronise input data streams, and input impedance can be selected either digitally (with Digitally Controlled Impedance, DCI) or by toggling an input termination parameter. Each IOB buffer has several registers, for driving the output pins or for clocking input data (even Dual Data Rate - DDR (Dual Data Rate)- is supported). However, the data may also just be routed through.

[9] The long lines are often referred to as the "backbone", you can force your design to use them with the USELOWSKEWNETS constraint.

[10] All interconnect is designed to minimise crosstalk, so for internal logic this does not need to be evaluated.

[11] Overridden by `DCM_TAP_MIN` and `DCM_TAP_MAX`, the minimum increment seems to be ≈40ps.

[12] Based on [docV2PDS, page 61-65] [docV2PUG, page 68-84, 89-93].

[13] Based on [docV2PDS, page 54-57].

2.1.3 History

From the previous subsection you have learned what resources the TRU FPGA has to offer. I will shortly take this one step further as I present my work the digital design (see 2.2). However, as many people has been working with the TRU prior to my arrival, I find it appropriate to provide an abstract of the work done so far (tab. 2.4):

Date:	**Event:**
2004	First TRU (v1.0) card proposed by Hans Muller. Two versions: PHOS: v1.0a, EMCAL (ElectroMagnetic CALorimeter): v1.0e.
2004/2005	Board design (schematics, layout, ..) completed by Rui Pimenta.
2005	Alexandra Oltean starts with code design. Major work put into a deserialiser using oversampling.
2006	A PCB layout with Cadence Allegro is made by Xi Cao.
2006	The PCB layout is produced with two prototypes in Wuhan, China.
2006	The prototypes are tested. Linfeng He (China) is improving the digital design (reset schemes, verifications, BoardController).
2006/2007	Rui Cai (China) started to work on hardware verifications (signal termination at the ADCs, ADC bias schemes, FEE channel mapping).
2007	Nicolas Degrenne (France) continues Rui Cais work.
2007	Alan Crouau (France) picks up the lead after Nicolas, and performs major work related to improvement of the deserialiser and creates the first prototype code for TRU-TOR data transmission.
2007	I arrive and spend a few months with Alan where he unloads some of his acquired knowledge to me, and where we discuss ideas for future TRU development.
2007	A revision of the TRU hardware design is done by Rui Pimenta and Xi Cao.
2007	New TRUs (v1.1) are produced and tested in Wuhan, China.
2008	The first PHOS module is installed in ALICE and with it 8 TRUs (v1.0a).
2008	Dong Wang (China) will resume TRU development where I left it.

Tab. 2.4 - TRU History [docNICOLAS, top part from page 88]

Let us move on to the digital design, shall we?

2.2 Digital Design

The task of programming the digital logic of the TRU FPGA has proved to be quite a challenge. The Virtex 2 Pro FPGA were simply never designed to take on a design such as the TRU, but nevertheless used, so we just have to try to find a way to make it work against the odds. You may wonder what on Earth I am talking about, so lets just start with basics - a break-down of the TRU design to date[14] (fig. 2.5):

2.2.1 Code Buildup

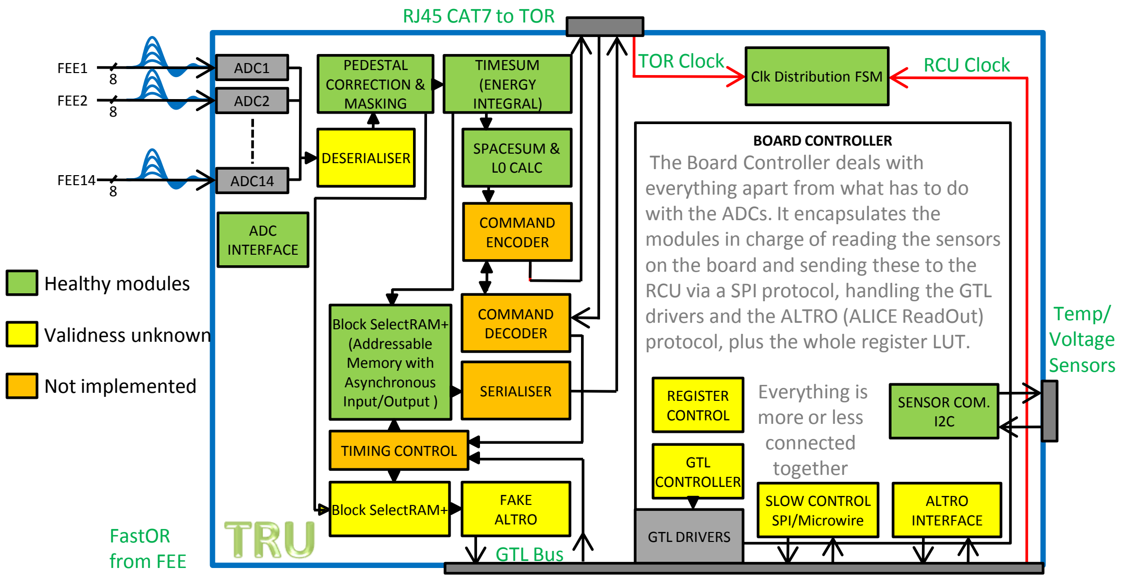

RJ45 CAT7 to TOR
FEE1
FEE2
FEE14
8
ADC1
ADC2
ADC14
ADC INTERFACE
PEDESTAL CORRECTION & MASKING
DESERIALISER
TIMESUM (ENERGY INTEGRAL)
SPACESUM & L0 CALC
COMMAND ENCODER
COMMAND DECODER
SERIALISER
Block SelectRAM+ (Addressable Memory with Asynchronous Input/Output)
TIMING CONTROL
Block SelectRAM+
FAKE ALTRO
GTL Bus
TRU
TOR Clock
Clk Distribution FSM
RCU Clock
BOARD CONTROLLER
The Board Controller deals with everything apart from what has to do with the ADCs. It encapsulates the modules in charge of reading the sensors on the board and sending these to the RCU via a SPI protocol, handling the GTL drivers and the ALTRO (ALICE ReadOut) protocol, plus the whole register LUT.
REGISTER CONTROL
Everything is more or less connected together
SENSOR COM. I2C
GTL CONTROLLER
SLOW CONTROL SPI/Microwire
ALTRO INTERFACE
GTL DRIVERS
Temp/ Voltage Sensors
FastOR from FEE
Healthy modules
Validness unknown
Not implemented

Fig. 2.5 - TRU Code Buildup [sofPPT]

In fig. 2.5 I have tried to illustrate how the TRU design can be "thought of" in terms of functional behaviour. The right half of the figure contains what we call the BoardController, which may be thought of as a "control unit" which monitors temperatures, voltages and currents, controls the GTL busses and the relevant communication protocols, and finally contains a rather huge list of registers - which is where the RCU can read data from or write data to. A wire variety of functions has its roots in the BoardController registry, as can be seen from the TRU registry list supplied in appendix C.

The left half of the figure represents the framework of modules in charge of the actual Level0 trigger calculation. A brief description will follow on the next page, and more thorough explanations can be found throughout this subsection.

[14] As of 26. July 2008.

In the upper left corner you can spot the ADCs, coloured grey since they represent hardware and not programmable logic. There are 14 ADCs in total, each connected to a respective FEE card from which it receives 8 FastOR signals. The dynamic range of the ADC inputs is ± 1V (linear range, input bias within a few mV), which is digitised with 12 bits accuracy (resulting in a resolution of $2V/2^{12} \approx 0.5mV$) at 40MHz. These bits are then sent serially to the FPGA at both edges of a 240MHz clock, effectively creating a bitrate of 480Mbps.

When these bits arrive at the FPGA they first need to be *synchronised* with the internal clock of the FPGA (see 2.2.5), and then *deserialised* (see 2.2.6). Several factors related to the operation of the ADCs may be controlled with a dedicated ADC interface (see 2.2.4).

Each FastOR pulse is around 100ns long, which can be represented with 4 samples in our 40MHz digital world. As only the positive part of the FastOR pulses contains any valuable information, the average is subtracted on a per-channel basis (referred to as *pedestal subtraction*, see 2.2.7). If the channels are noisy they may be masked out with the use of a mask register. We call the data at this point *raw data*. Since this is the type of data the FakeALTRO protocol are meant to ship to the RCU upon request, it will be stored in a Block SelectRAM element which can retain a history of 256 samples (equals 6.4μs).

The Level0 trigger is based on an evaluation of the energy in a "shower" of photons. In time the energy is spread out over 100ns (the length of the FastOR pulse). To integrate the energy in time, we simply sum up the 4 samples which describes the respective FastOR pulse. At this point we call the data *time summed* (see 2.2.7), which is the type of data the TOR needs in order to calculate the Level1 trigger. Similar to the "raw data" storage for the FakeALTRO protocol, the "time summed" data will be sent in a Block SelectRAM element retaining a history of 256 samples.

When particles hit the PHOS crystals the light might disperse into several adjacent crystals[15]. In order to get a scalar representing the total energy of the particle, we will need to sum up matrices of 4x4 crystals (referred to as *space summing*, see 2.2.7). As each of the FastOR signals represent the sum of 2x2 crystals, the 4x4 sum can be calculated by summing 2x2 FastOR signals. All possible combinations of 2x2 FastOR signals will be summed (the idea being that at least one of these "squares" be be centered over the particle entry point), but as the edges of the branch can not be summed (the TRU can not "see across" the branch boundaries) only 91 space sums may be calculated out of the 112 FastOR signals received[16].

If the energy of any of the 91 space+time summed values exceeds a certain set threshold, a *Level0 trigger* will be generated. It is currently sent to the TOR on a dedicated line, but this might change as the TOR communication interface is being developed (see 2.2.8).

[15]The dispersion factor is both a function of the particle angle and size.

[16]From the 112 FastOR signals (14 FEEs (z-axis) times 8 channels (x-axis)) only `13x7=91` squares of 2x2 FastOR signals may be summed. Only the Level0 calculation will suffer from this "flaw"; only time summed data will be sent to the TOR which allows it to re-do the space sum without the branch boundary limitation (the TOR receives data from all TRUs, thus the only boundary will be those of the PHOS module).

2.2.2 General Design Changes

I have made quite a few general edits to the code since the version of the TRU code I received upon arrival (see [srcTRUDESIGN]). They are summarised in the following list:

- *Conversion to Verilog 2001.* The design code I received used almost extensively the Verilog 95 syntax, which means no multidimensional arrays, no generate statements, and port lists where the ports had to be redefined 3 times in order to declare the name, direction and type, respectively. For those of you who are not into the terminology: The design code very quickly becomes very messy and very huge. I suggest that you load a few source files from the new code and the old to see the difference.

- *Rewriting functions to improve logic distribution.* The old design used synchronous design elements for everything[17]. Even for concurrent arithmetical operations where only the end results were interesting, the data was clocked into respective registers along the way. The main advantage of such an approach is that design elements can run at very high speeds, but as the TRU base clock runs at 40MHz (which is considered a low/moderate speed) this involves bad performance (as for the greatest part of a clock cycle nothing will get done) and a non-optimal logic distribution (as in my experience the design works best with a balanced logic usage). As registers were always the bottleneck in the old design, and the alternative resources were hardly used, the logic should be redistributed.

- *Design altered to better integrate with future design elements.* High up on the wish-list for my supervisor at CERN and other people in my project were the communication interface to the TOR and the FakeALTRO interface for the RCU. The former is necessary if we want the Level1 trigger in PHOS as the TOR can not calculate anything without any raw data from the Level0 trigger event. The FakeALTRO, is, as the name implies, a "fake" ALICE ReadOut protocol. It enables the RCU to read out the raw data from the TRU in the same way as it reads out the raw data from the FEE cards (from the ALTRO (ALICE ReadOut) chips [docALTRORC]). Both design objects requires access to the relevant data, but as storage is limited I redesigned the code a bit in order for it to be plug-and-play with these functions - with a minimum of extra logic required.

The rest of this subsection will deal with design elements which I altered or added. In the next subsection I will talk about which clock domains the TRU must relate to (see 2.2.3).

[17]If you "think" like in a high-level programming language when designing hardware, you will end up with the register elements (in a CLB, see 2.1.2.4) as your bottleneck. The reason is simple: What is stored in the registers are always accessible at any given time (not like in a RAM where only some addressed bits can be accessed at a given time), which provides data in exactly the same "state" as in a high level variable.

2.2.3 Managing Clocks

I rewrote the global clock and reset distribution scheme for the TRU (the "box" you can see in the top-right corner of fig. 2.5). What does this mean? First, have a quick look at the table below (tab. 2.5) for a complete list of clocks entering and exiting the TRU FPGA:

Clock name:	Description	Type:	Frequency:	Global?
BRD_CLK40M	On-board crystal clock	LVCMOS	40MHz	Yes (4S)
ALT_SCLK	ALTRO Sample clock (LHC domain)	LVDS	40MHz	No
TorReadOutClk	TOR Readout clock (LHC domain)	LVDS	40MHz	No
ALT_RDOCLK	ALTRO Readout clock	LVCMOS	40MHz	Yes (6P)
RCU_SCL	RCU Serial clock	LVCMOS	10MHz	No
SNS_SCL	Serial clock to various sensors	LVCMOS	10MHz	No
ADC_SCLK_P	Serial clock to ADC	LVDS	13MHz	Yes (0P)
ADC_SCLK_N	Serial clock to ADC	LVDS	13MHz	Yes (1S)
ADC_CLK40M_P	Base clock to ADC	LVDS	240MHz	No
ADC_CLK40M_N	Base clock to ADC	LVDS	240MHz	No
ADC_ADCCLK_P[1:14]	Frame clocks from ADC	LVDS	40MHz	Yes (2P)
ADC_ADCCLK_N[1:14]	Frame clocks from ADC	LVDS	40MHz	Yes (3S)
ADC_LCLK_P[1:14]	Bit clocks from ADC	LVDS	240MHz	No
ADC_LCLK_N[1:14]	Bit clocks from ADC	LVDS	240MHz	No

Tab. 2.5 - TRU External Clocks

The Readout Clock (`ALT_RDOCLK`) is the clock we must use if/when data has to be sent to the RCU ("Fake ALTRO" implementation in the TRU). The RCU serial clock is used for the "Slow Control" communication protocol[18], and the `SNS_SCL` signal is supplied to the various sensors on the TRU. The `ADC_SCLK` signal is supplied to the ADC for communication purposes (see 2.2.4), and the `ADC_CLK40M` signal is supplied to the ADC to drive the sampling process. Back from the ADCs we receive a serial data stream along with a synchronised bit- and frame-clock (`ADC_ADCCLK` and `ADC_LCLK`, respectively).

You may have noticed that I skipped the three topmost clocks listed in tab. 2.5. These represent alternatives for the global 40MHz clock. So far the TRU design has been "driven" by the internal crystal (`BRD_CLK40M`) which is mounted on the board. However, when the TRU is installed in ALICE it must utilise the LHC clock domain (such that all the electronics in the LHC scheme is synchronised). There are two paths through which the TRU can receive the LHC clock: From the TOR (`TorReadOutClk`), or from the RCU as a signal labelled `ALT_SCLK` on the GTL bus.

[18]The "slow control" protocol does not work on the TRU as of 25.June 2008.

What we want to do here is to have both the LHC clock and the crystal clock connected to their respective DCM, and connect the output clocks to a global clock multiplexer[19]. The multiplexer should be controlled with the `LOCKED` signal of the LHC clock DCM (which indicates whether the LHC clock is present) such that the LHC clock is used whenever possible.

I mentioned that I ended up redesigning the entire clock and reset distribution scheme. This was caused by the following factors:

- The need to create the global clock multiplexer (as previously mentioned) interfered with the old global reset scheme.
- The old ADC control modules were nested into the clock and reset scheme. This was confusing, and since I needed a control mechanism with more features anyway I redesigned both this and the clock distribution module.
- The old clock distribution code was messy. It was modified many times by several people and proved to be very hard to read. Some design choices were also bad, for instance was the clock supplied to the ADCs gated through flip-flops. This is not an optimal solution since these logical blocks were never designed to provide high quality clock signals. A high quality clock has perfect phase and duty cycle, rapid rise/fall times and are guaranteed to be glitch free. With bad clocks we may quickly find ourselves struggling with metastability issues [intMETA].
- There were quite a few "remains" from previous deprecated design elements. Cleaning up the code is important in order to get a clear picture of what you are doing.

The clock distribution design elements are encapsulated in the `InitAndClkDist` module in the TRU design code [srcTRUDESIGN], have a look if you are interested. In the next subsection we will take a look at the ADC interface, which - among other features - contains the "control modules" that were removed from the global clock scheme.

[19]Note that the clock multiplexing scheme is not present in my last version of the code as I never got around to re-add it after I reverted my code to an older version in my work with the ADC synchronisation, see 2.2.5.

2.2.4 ADC Communication

The TRU-ADC interaction is split into two parts: (1) Communication between the two required in order to alter the default behaviour of the ADCs, and (2) dealing with the sampling process. Take a look at fig. 2.6 to see how it all fits together. This subsection will deal with the communication protocol, while the next (see 2.2.5) will describe the various challenges related to ADC data synchronisation.

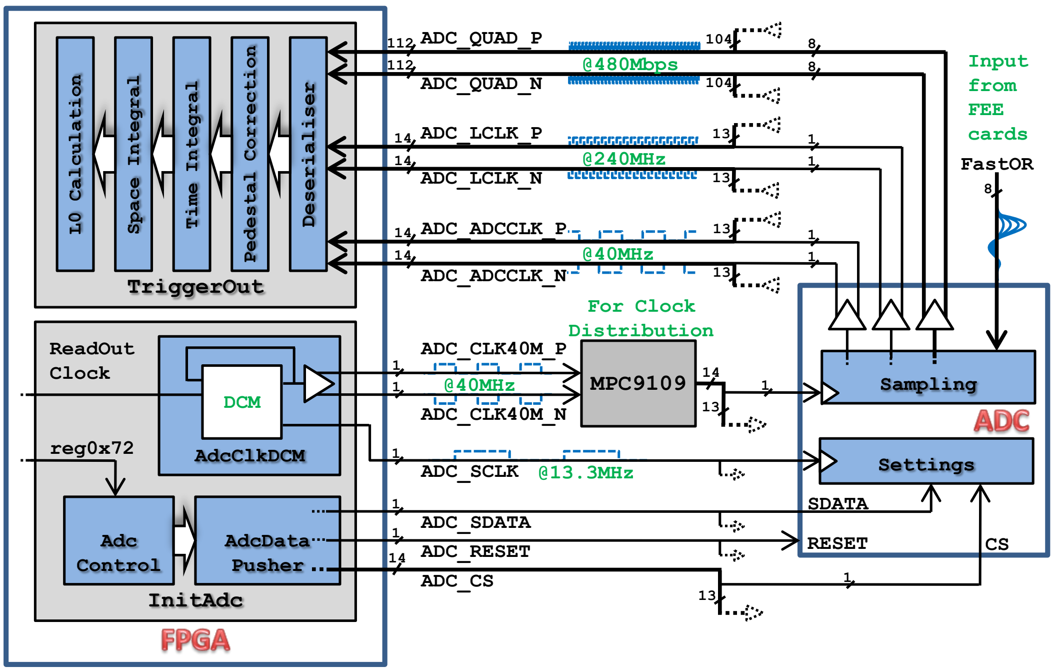


Fig. 2.6 - ADC Interaction [sofPPT]

Motivation *2.2.4.1*

The TRU had an ADC interface module which were located in the BoardController. When I was redesigning the deserialiser (see 2.2.6) the need to know exactly what input to expect from the ADCs arose. Since the ADCs are capable of sending out custom patterns, I researched how to set them up in such a way. Basically, you will need to send a few 8 bit words to the ADCs via a SPI/Microwire serial protocol. However, even though there was already a TRU module aimed to provide this functionality, I was not able to get the the ADC interface working (it was not even connected initially) and could not find any documentation on its inner workings.

So I went ahead and made my own protocol, and here is the documentation!

Communication Protocol *2.2.4.2*

In the lower half of fig. 2.6 you will find the relevant signals and “blocks” that make up the communication system. Basically, the communication is a one way deal where the TRU tells the ADCs how to behave. It does so by targetting the relevant ADCs through the use of the “ChipSelect” lines (*ADC_ CS[13:0]*), one for each ADC. An ADC accepts the communication words when the “ChipSelect” line is low. The command word is sent serially over the data line (*ADC_ SDATA*) with MSB (Most Significant Bit) first, which is and clocked into the ADC on positive edge of the dedicated clock line (*ADC_ SCLK*). This clock has the same source as the 40MHz clock driving the sampling process, but is 3 times slower.

The module *AdcControl* listens to `reg0x7c` (bits [7:0]) and will initiate a process of updating all the ADCs with the relevant data should the change prove to be stable for at least 5 clock cycles. The data in `reg0x7c[7:0]` corresponds to the bits D7:D0 of the command word you wish to send to the ADCs (for all possible commandwords see C.1). The ADCs are first targeted by the *AdcControl* module, then the control is handed to the *AdcDataPusher* module which will send the word according to protocol.

As a precaution measure I set the *AdcControl* module up to update only one ADC at a time. A chronogram of the whole process undergoing simulation can be found in fig. B.1.

Conclusion *2.2.4.3*

The communication with the ADCs seems to be working flawlessly. I have used it extensively to adjust the output current of data and clock signals, set up various data patterns (for synchronisation or deskew purposes) and to power up/down channels while debugging the design. For a complete list of possible commands, have a look at register `0x7c` in the TRU register list (see C.3).

While the communication interface works nicely now, it would also be nice to specify settings for only a certain set of ADCs. As the protocol is currently designed, the 8 least significant bits of the value held by register `0x7c` is written to every ADC. Maybe the remaining 8 most significant bits of this register could be used to specify which ADCs should be updated with the value set? I never got around to do this myself.

The module containing these relevant design elements is labelled `InitAdc`, look it up in the TRU design (see [srcTRUDESIGN]). Now, let us take a look at the data synchronisation part of the ADC interface.

2.2.5 ADC Data Synchronisation

As you can see from fig. 2.6, the ADCs receives - in addition to the communication clock (ADC_SCLK) - a clock operating at 40MHz (ADC_CLK40M) which is fed into the ADC PLL (Phase Locked Loop) to drive the sampling process. The PLL - used to assure perfect duty cycle and low skew internal clock distribution - can not operate at frequencies lower than 20MHz or higher than 40MHz. These are the boundaries within which we have to keep our sampling speed in order to get predictable results.

The FastOR signals are digitised into 12 bit samples at a 40MHz frequency. Each ADC input has a dynamic range of ± 1 V, which translates to an effective sample resolution of $\approx 500\mu$V. The bits are sent serially to the FPGA [20] over the dedicated data lines[21] (ADC_QUAD), with an accompanying bit-clock (ADC_LCLK) and word-clock (ADC_ADCCLK) that can be used to ease the deserialisation process. The serial approach means less interconnect (hence less EMI (ElectroMagnetic Interference) and possibly higher speed), less clock-to-data skew, and allows the designer to choose an FPGA with fewer pins (which will reduce the cost a lot in big designs). The drawback is that a deserialiser must be implemented in the other end, which must be done manually in the FPGA used in the PHOS TRU [22].

In this subsection I will deal with the ADC data synchronisation issue, but this topic is also very closely related to the ADC data deserialiser, a variant of which will be described in the next subsection (see 2.2.6).

Motivation 2.2.5.1

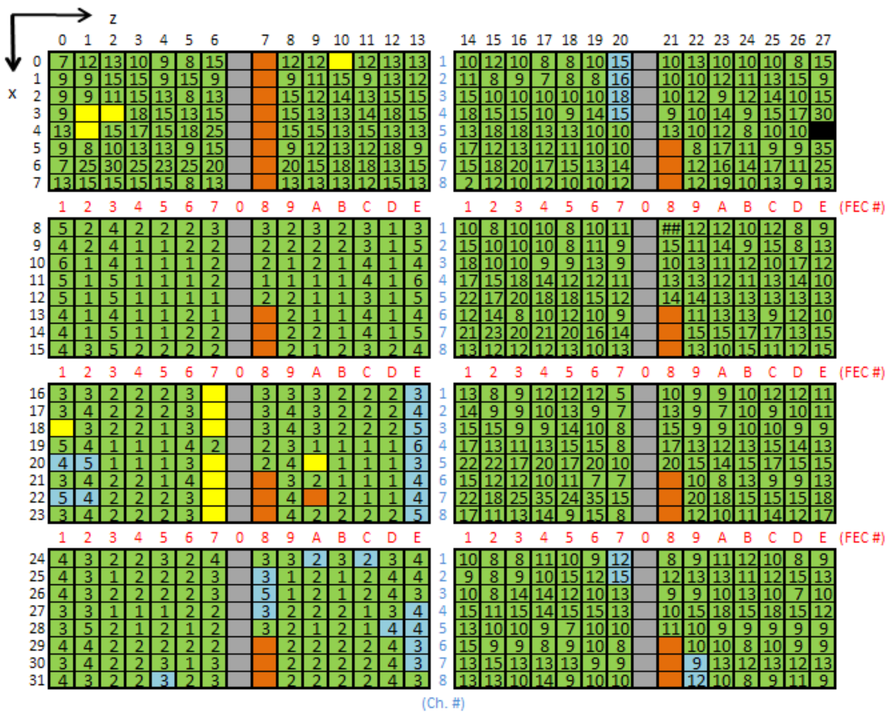
To synchronise the ADC data with the internal FPGA clocks has proved to be a big design challenge for the TRU. Quite a few people has been working on this issue, with varying degree of success. When I received the TRU code from Alan Crouau he was very close to a fully working solution. On the next page I supply an illustration (fig. 2.7) of a readout I performed of the entire PHOS module 2 while it was installed in ALICE [23], which used Alans design. I manually read the TRU ADCs out with ChipScope (see A.2.3), then roughly calculated the size of the average noise in ADC counts. These values I then put in the coordination system (of the ADC channels) you can see in fig. 2.7. Green and blue channels are more or less healthy, but the red and yellow ones are not. This is a synchronisation problem.

[20] The user can define whether MSB or LSB (Least Significant Bit) are to be sent first with the use of the ADC communication interface (see 2.2.4.2)

[21] The datalines are designed such that they all have the same length. This is in order to match up the phase of the data as good as possible at the FPGA entry point.

[22] The Xilinx Virtex-II Pro FPGA which does not support hardware deserialisation.

[23] Readout performed 11. July 2008.



🟧 Random output from ADC. Cause: Bad sync? Broken ADC?

🟨 Spikes. Major spikes (up to 4000 ADC counts) occuring at average every 25us.

🟦 Oscillations (~10 ADC counts @ 1MHz).

⬛ Hot Spot. OK but huge amount of noise.

⬜ TRU.

🟩 Everything seems OK. Perfectly sampled white noise (<20counts with HV on APDs).

Fig. 2.7 - Noise Map of PHOS Module 2 [sofXLS]

A readout like the one illustrated in fig. 2.7 is not very accurate as only a data history of ≈0.2μs will be read out for every ADC channel[24]. This means that channels that are "nearly unstable" might be coloured green if they happened to be in a "good state" when they were read. This especially applies to the "spike" effect as I tagged yellow in fig. 2.7. I believe the spikes indicates channels which are "living on the edge", and even small environmental changes (temperatures, external noise, electromagnetic interference, etc.) may provoke a faulty state. Even if this applies for only an instant, it might be enough to set off a Level0 trigger (depending on the threshold levels). When the TRUs were configured with Alans design we saw that way too many Level0 triggers were generated, even when the completely unstable channels were masked out (the red ones in fig. 2.7). How could this be fixed?

The solution did not come about easy. In the next subsection I will present an analysis of the problem, which is based on my experiences with the design process.

[24]The amount of data read out by ChipScope is defined by the size of the ChipScope core. The mentioned ≈0.2μs history is achieved by using a ChipScope core which stores 8188 samples of data per channel.

Problem Breakdown ***2.2.5.2***

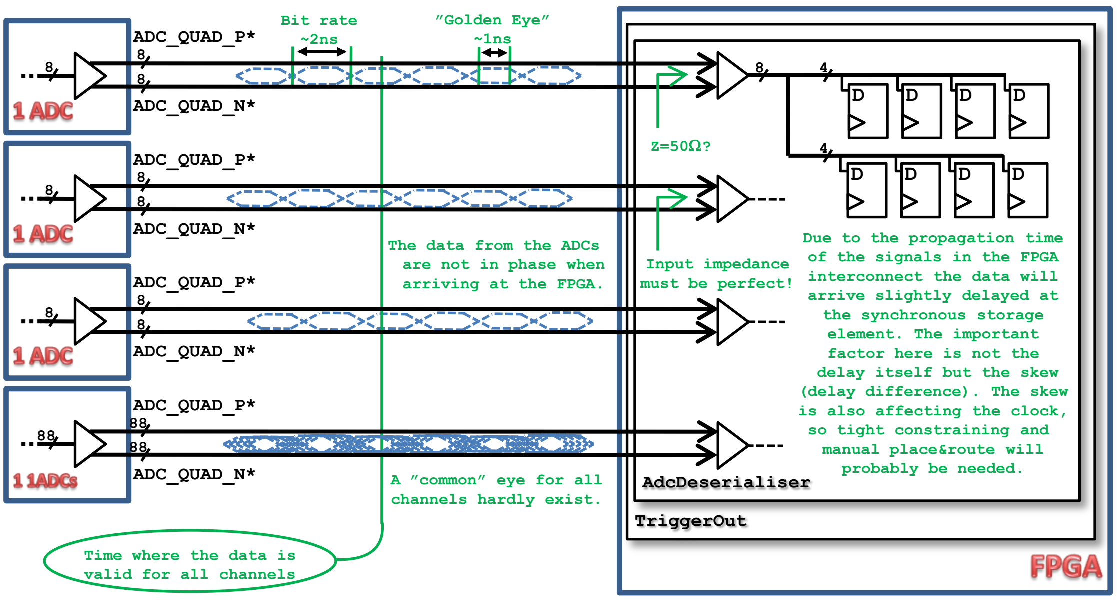

Bit rate
~2ns
"Golden Eye"
~1ns
ADC_QUAD_P*
ADC_QUAD_N*
1 ADC
The data from the ADCs are not in phase when arriving at the FPGA.
A "common" eye for all channels hardly exist.
Time where the data is valid for all channels
Z=50Ω?
Input impedance must be perfect!
Due to the propagation time of the signals in the FPGA interconnect the data will arrive slightly delayed at the synchronous storage element. The important factor here is not the delay itself but the skew (delay difference). The skew is also affecting the clock, so tight constraining and manual place&route will probably be needed.
AdcDeserialiser
TriggerOut
FPGA

Fig. 2.8 - Synchronising the ADC Databits [sofPPT]

Fig. 2.8 shows the key aspects of concern when the ADC data is to be synchronised. The key factor here is the high 480Mbps data bit rate, effectively causing problems with:

- *Signal integrity.* As the speed of clock and data signals increases the signal quality generally tend to decrease. Even small impedance mismatches may cause big signal reflections, and parasitic components such as capacitive load will kill the signal edges. The duty cycle of clock signals may deteriorate as a function of jitter (unintended phase variations) and the "golden eye" of the data will shrink[25].
- *The datasignals arrive at the FPGA with different phases.* One ADC has 8 channels, each of which is providing datasignals with relatively low skew (≈100ps). However, the the skew is much higher if channels from several ADCs are considered. This means that the "common golden eye" is very small, or hardly exists at all.
- *Delays in the FPGA routing network.* When signals are to be distributed internally in the FPGA, delays of a few ns from source to destination are not uncommon (depending on the type of interconnect utilised, see 2.1.2.5). Here arises the problem of how to correctly synchronise all the bits when the bitrate is higher than the skew of our internal clocks.
- *Design verification.* The ChipScope module becomes very hard to work with at speeds greater than 80MHz, which means that you must validate this part of the design in simulation and hope for the best when configuring the FPGA with it (which works well for low speed designs, but not at all for high speed designs).

[25]The "golden window" is the timeinterval of which the data is stable.

Discovering Impedance Mismatch *2.2.5.3*

At this point you may ask yourself why just not use the bit- and frameclock supplied by the ADCs to synchronise the data in the FPGA? This is exactly the same question I asked myself when I received the TRU design code. Previous designers all chose to go with DCMs for clocking the bits and frames into the FPGA, but why?

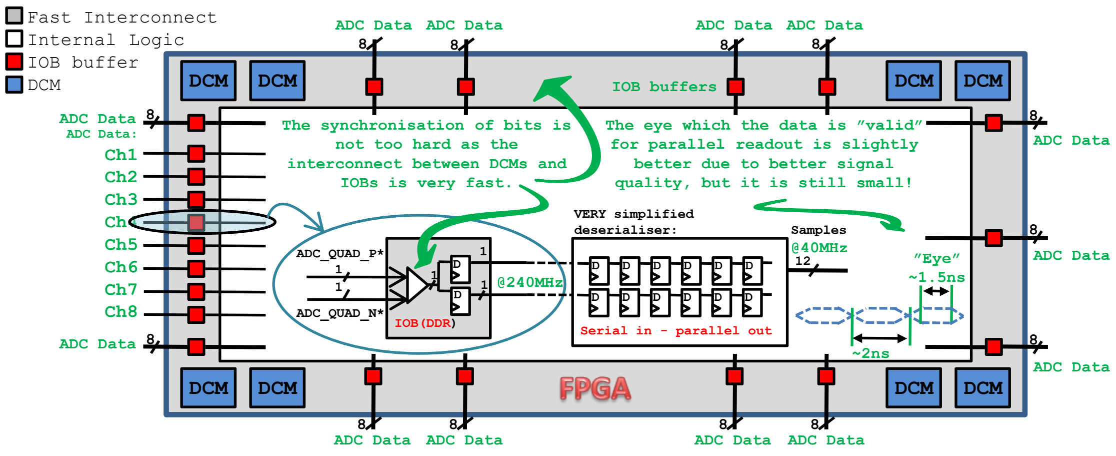

Fast Interconnect
Internal Logic
IOB buffer
DCM
ADC Data
ADC Data:
Ch1
Ch2
Ch3
Ch4
Ch5
Ch6
Ch7
Ch8
8
DCM
IOB buffers
The synchronisation of bits is not too hard as the interconnect between DCMs and IOBs is very fast.
The eye which the data is "valid" for parallel readout is slightly better due to better signal quality, but it is still small!
ADC_QUAD_P*
ADC_QUAD_N*
IOB (DDR)
@240MHz
VERY simplified deserialiser:
Serial in - parallel out
Samples @40MHz
12
"Eye" ~1.5ns
~2ns
FPGA

Fig. 2.9 - Synchronising Bits and Frames [sofPPT]

In fig. 2.9 I have illustrated the "input path" of a FastOR pulse (1 ADC channel). First it enters the input buffer (IOB block, see 2.1.2.7), where the signalling standard (LVDS_25 in most cases) may be set, along with various other settings (termination, pullup/pulldown, input impedance, delay, ..). The input buffers can be either synchronous[26] (store the bits) or route-through, and if the former is desired dual-data rate register (DDR) can be chosen[27], which is the case in fig. 2.9. These IOB registers must be supplied a clock directly from the DCMs, which will propagate through very fast interconnect (the "grey field" in fig. 2.9) effectively assuring a minimum of clock skew.

There are only a total of 8 DCMs available (see 2.1.2), and since 3-4 DCMs must be reserved for the global clock distribution scheme and to drive/receive external signals, 3-4 ADCs must share bit- and frameclock from the same DCM. However, say you use 1 DCM from each quadrant of the FPGA to clock the ADCs in that respective quadrant. Then the clock skew can almost be neglected, and relative skew between IOB channels can be finetuned with the input buffer delay constraint (`IOBDELAY`)[28]. Since the IOB buffers and DCMs never change position this approach can be considered permanent. Once done, it will never have to be redone.

[26]Synchronous IOB buffers is preferred as this provides very good fan-out properties.

[27]IOB DDR registers can be considered "free" as they are localised in the IOB block itself. They were not previously utilised, but when internal clocks are used to do the bit-synchronisation they are ingenious as they make the serial datastream "system synchronous" as opposed to "source synchronous". As a bonus you get two bitstreams with half the initial bitrate, which are easier to handle. Specify the usage with the `INST "<hierarchical path to instance>" IOB=TRUE` constraint.

[28]Thanks to Dr. Raúl Esteve Bosch, Universidad Politécnica de Valencia, for tipsing me about this one.

In fig. 2.9 I made an simplified illustration of the simplest deserialiser one can make; clock the bits into a shift-register consisting of flipflops, and clock the data out when all the bits are in the right position. Naturally, the data is only in the "right position" for only about 2ns (less when jitter is evaluated). This means that such an approach requires a frameclock with minimal skew (remember, we have to use the same few DCMs to supply frameclocks). But, even if the global clock banks (see 2.1.2.5) are utilised, the skew in the internal logic of the FPGA will be higher than that of the fast interconnect surrounding the IOB buffers. How can be synchronise the frameclock in this environment?

My first take on the situation was to construct a deserialiser which uses Distributed SelectRAM (see next section, 2.2.6) to allow the bit- and frameclock to be completely phase independent. Basically this means that the frame synchronisation would be directly resolved in the deserialiser. However, due to problems with verification of the design and shortage of time I chose to make a deserialiser so simple (like the one shown in fig. 2.9) that "nothing could possibly go wrong". I knew it would be hard to get it working on all ADC channels, but for most of them it would (as this was also Alans approach). Then I set one DCM up to provide the bitclock, and another to provide the frameclock. After creating a few statemachines I was able to adjust the frameclock with register `0x79`, and the bitclock with `0x7a` (see C.3 for a description of the registers, and A.6 for directions on how to use them).

Using the ADC setup register `0x7c`, the phase adjust registers `0x79` and `0x7a` and ChipScope to read samples on-the-fly I was now able to "map" the real cause of the deserialiser failure. From fiddling with this setup for a while I gained some unexpected but valuable knowledge:

- *The phaseinterval where the bit-clock correctly clocked the bits for a couple of ADCs were at least 400ps (10 values difference in register 0x7c).* This seems to verify my previous statements; the DCM-IOB interconnect is fast enough for the bits to be clocked with a single clock.
- *A bitsequence like 0000 0011 1111 was often interpreted as 0000 0010 1111*, and
- *data was not stable as a function of ADC output current (drive strength).* These two observations had me puzzled. Were the ADCs generating patterns which were not in accordance with the settings I specified? Was the deserialiser still unstable, or were the ADCs suffering from a bad power supply (the given bitpattern causes all 112 ADC channels to go from 0 to 1 at the same time, which may cause the energy deposits in the powersupplies to deplete)? Or could there be an impedance mismatch somewhere?

All the TRU input buffers were previously configured for the 2.5V LVDS signalling standard, with a differential termination of 100Ω. Maybe this was not the optimal solution? I tried a few alternatives, and finally found that using buffers with Digitally Controlled Impedance (DCI) completely resolved the problem[29]! With a digitally defined input impedance of 50Ω the bits were suddenly clocked in perfectly, and the "golden window" of the datastream widened quite a bit!

[29]The name of the DCI buffer I used: `LVDSEXT_25_DCI`, where "EXT" means "extended" and refers to the voltage range (maximum and minimum ratings).

Conclusion *2.2.5.4*

While this subsection could have been merged into the next one dealing with the deserialiser (see 2.2.6), I chose to separate the two in order to specify which elements our synchronisation problem really consist of. While I have tried to present this information in a way that is "natural to read", the problem was only slowly defined as I started to investigate the reason(s) for why the deserialiser was unable to deliver the correct samples.

To sum it all up, the data may appear badly synchronised if:

- The input buffers are slightly misconfigured.
- The databits are clocked into the input registers when the data is in a metastable state (bad phase of the bitclock).
- The parallel read of bits from a deserialising array of flipflops are not synchronised correctly (bad phase of the frameclock).

The first point were discovered by creating a deserialiser with variable bit- and frameclocks, such that those two factors could be tuned out of the picture. When a better input buffer type was selected, the the results acquired when testing designs with various bit- and frameclock schemes suddenly seemed much more reliable. A disturbing factor had been removed from the design.

I also learned that the problem was not as much to synchronise the bitstream on the way into the logic, as it was to correctly synchronise the bits with the 40MHz internal clock. Even across ADCs a single bit-clock seemed to to the trick, but in order not to push our luck we should generate clocks and deserialise channels on a quadrant-to-quadrant basis[30]. This should assure that the bits always receive the best possible welcome upon FPGA arrival.

You may have noticed that I never speak about using the frame- and bitclock supplied by the ADCs to clock the data. The reason for this is that I initially tried them out, but with very bad results[31]. However, with the new DCI buffer this seems to work flawlessly [32]. The DCMs are a valuable resource which can be used for alternative purposes, and besides, setting them up to generate a 240MHz clock from a 40MHz clocked input means pushing them beyond the absolute ratings[33] (though the functional behaviour still seems to be correct). On the other hand, using DCMs provides better timing models (as the design becomes "system synchronous") and eases the process of setting up timing constraints.

[30]To me it does not seem to make any major difference whether the IOB register is clocked by the DCM or by externally/internally generated clocks, but the datasheet recommends the use of the former.

[31]For instance, I struggled half a day trying to understand why a DCM with a 240MHz clock from one of the ADCs connected to the input refused to let me adjust the output phase. I eventually noticed that the input clock was so bad that the DCM was not even able to lock on it.

[32]Dong Wang, the Chinese (Wuhan) student who continued my work has done so with success (see 2.2.6.7).

[33]When the input of a DCM is clocked at 40MHz the `DFS_FREQUENCY_MODE` parameter of the DCM is set to "`LOW`", and this mode does not support upscaling of the frequency to 240MHz. If you "cascade" DCMs (increase the frequency in two steps) you *must* connect the `CLK2X` port of the first DCM to the input of the next. Using the `CLKFX` port messes up the timing models, causes distorted clocks (according to datasheet [docV2PDS] the jitter conditions of an `CLKFX` signal is not optimal) and is not recommended by Xilinx designers (you will not find this approach in any application note).

2.2.6 ***Deserialiser***

You should not read this subsection unless you first read the previous one (see 2.2.5). Issues described here and in the previous subsection are very closely related.

Motivation ***2.2.6.1***

As previously mentioned the Virtex 2 Pro FPGA does not provide a hardware solution for deserialisation of external serial streams (as is featured by more recent models in the Virtex family), so this must be implemented manually in the FPGA internal logic. There are several ways of doing this, and almost all have been tried in the TRU design with varying degree of success. The first solution, designed by Alexandra Oltean, featured an approach where the input data stream were "oversampled". She used 4 input registers per ADC data channel, and used 1 DCM to generate 4 clocks with 0°, 90°, 180°and 270° phase offset, respectively. The approach was very similar to the one discussed in this application note [docXAPP225].

I find this approach elegant, but it has several potential pitfalls. First, from a 240MHz clock you will get 4 clock signals which are 1ns (90°) out of phase. Add jitter (10-20%) and skew (more than for other approaches as more CLBs are utilised in this approach) to this equation, and you are looking at very strict timing and placement requirements. Besides, the CLB represents the most valuable resource we have in the TRU FPGA (it always seems to represent the "bottleneck" in the design) so using 4 times the number of input registers does not seem like a good idea to me.

My predecessor, Alan Crouau, made a deserialiser similar to the one I mentioned in the previous subsection (see 2.2.5.3). He did not use IOB registers, but routed the data through the IOB buffers and clocked them into a flipflop chain with a dynamically adjusted bitclock which were common for all ADCs (the design was an upscaled and slightly modified version of the one discussed in this application note [docXAPP774]). He used the internal 40MHz clock for frame synchronisation, but had to use an extension stage (use twice the number of flipflops in order to postpone the moment where the bits have to be synchronised with the frameclock) to get it working. He was very close to a fully working solution, in his last design ≈90% of the ADC channels were correctly deserialised (see fig. 2.7).

Just before Alan left, we had some interesting discussions about how to make an even better deserialiser. The approach in this application note [docXAPP194] seemed particularly interesting, and a little on the side I probed into it so see if it could be used in our case. Finally, I gave in for the temptation and felt that I had to give it a try because:

- It would require only half of the CLBs utilised in in Alans first deserialiser.
- The outputs and inputs would be fully phase independent, effectively resolving the frame synchronisation problem completely.

Let us have a look at it!

Design Introduction *2.2.6.2*

The deserialiser which will be discussed has a very untraditional approach to the deserialisation issue. The idea is to use multiplexers to sort the bits into RAM blocks, where each of them store bits with a given level of significance. The RAM blocks can then be read out independently of the write mechanism, and with a well thought out addressing method the samples can be put together and synchronised with a few registers.

I will implement the deserialiser through the use of Distributed SelectRAM (see 2.1.2.4). This renders possible a very clean approach to coping with the various clock domains introduced by the ADCs, and at a minimum of logic usage.

Input Stage - Sorting the Bits *2.2.6.3*

When using RAM to implement deserialisers there are a few issues we need to consider. First, while RAM generally has good storage capacity, only the addressed data can be accessed during a single clock cycle. This problem can be confronted by increasing the frequency at which the RAM output operates, but this will in turn require stricter timing constraints and make the design more prone to errors. Also, with this approach you will most likely be designing the module representing the biggest hit to performance in the overall design.

As illustrated by Xilinx digital designers (see [docXAPP194]) one can avoid an increase in frequency if shift registers and multiplexes are utilised. The trick is to shift the words one bit relative to each other, such that no bit with the same level of significance appears at the same clock cycle. I have chosen this approach, but with a tweak in order for it to be applicable for our design. The design discussed by the Xilinx designers had the same number of channels as sample resolution (hence good symmetry), and the raw data was single data rate (SDR) serial streams. In our situation we have to cope with DDR data streams and asymmetry between the channel number and sample resolution; the TRU ADCs deliver samples with a resolution of 12 bits, while there are 8 ADCs per TRU.

The first part of my deserialisation design is illustrated in fig. 2.10 (page 40). It shows two shift registers clocking in data on positive and negative edge of the sampleclock, respectively. The ADCs has 8 channels, so I build the deserialiser in segments of 4 and 4 channels. If 2 “dummy” channels are added to each segment, the symmetry will be restored (6 bits word on each edge of the clock, 6 channels).

The input counter will cycle through the numbers 0 to 5, which specifies which channels the multiplexers will select when the counter is used as an index in the look-up table (`MUXchLUT` in fig. 2.10). Each multiplexer has an unique index offset, which means that no multiplexer will access the same bit at any given time. The dummy-bits will not be written to RAM as the “write enable” port will be disabled when the dummy bits are indexed, this “write mask” will be specified with the LUT (`WEmask`), as is illustrated in fig. 2.11 (page 41). Each RAM block are indexed in accordance with the index of their respective multiplexer. The RAM blocks and the multiplexers come in pairs, each multiplexer is dedicated to the task of selecting the bits to feed a single RAM block.

Output Stage - Phase Correction *2.2.6.4*

The input stage and the output stage may live their separate lives, as long as we make sure that the exact same address of the RAM blocks is not written to and read from at exactly the same time. In the design this can easily be solved by having a two-bit counter keep track of the number of cycles the input counter has performed. Each Distributed SelectRAM block has a depth of 16 bits, which means that when 12 of them are utilised (1 for each bit with a given level of significance) they may hold a total of 4 samples. Here we encounter the the best part of the design, by making sure that readout is delayed - say 25ns (which equals 1 clock cycle) - we will not have to worry about the phase of the readout clock at all!

To clarify, if samples are being written to RAM address 0:3, we read from address 8:11 and may allow outself to relax because we have a skew margin of almost 45ns. For the next sample, when the addresses 4:7 are written to, we read from addresses 12:15, and so on.

The only "problem" in the output stage is that the sample from all 4 channels must naturally be read out before a new set of samples are being written in. This means that the outputs must be read at a rate of 160MHz, which will then have to synchronised down to 40MHz. The best idea I could think of was to connect the output line to 4 registers, and use the readout address as a ChipEnable signal to direct the bits into their respective registers.

I can imagine your thoughts right now goes something like: "wait a minute, the period where the 160MHz registers all contain the correct bits - and must be clocked over to the 40MHzregisters - are only 6.25ns! How come this does not create the same sort of frame-synchronisation problem as this design was trying to solve"? A very good question, but one that can easily be answered. Remember that the outputs of the RAM block is phase independent, which means that if the 160MHz clock is created with the same DCM that is distributing the 40MHz base clock, and a global routing buffer is selected for both clocks, the phase of these two clocks will be perfectly matched no matter what part of the design they are distributed to.

The next two pages contains the concept drawings I made of the deserialiser. After this I will list the logic usage and mention some of the problems I encountered during the design process.

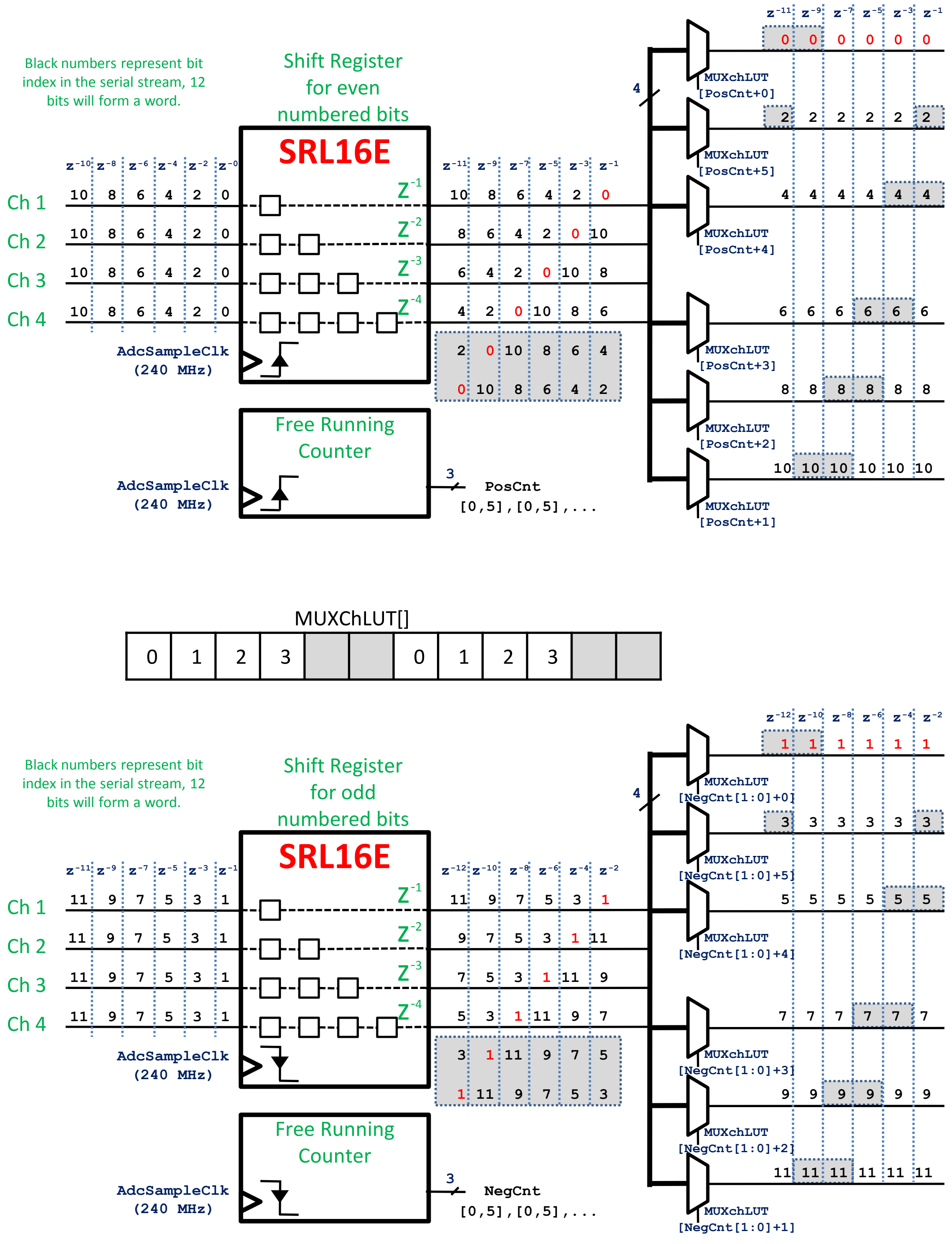


Fig. 2.10 - Deserialisation Stage 1 - Serial Stream Decomposition [sofPPT]

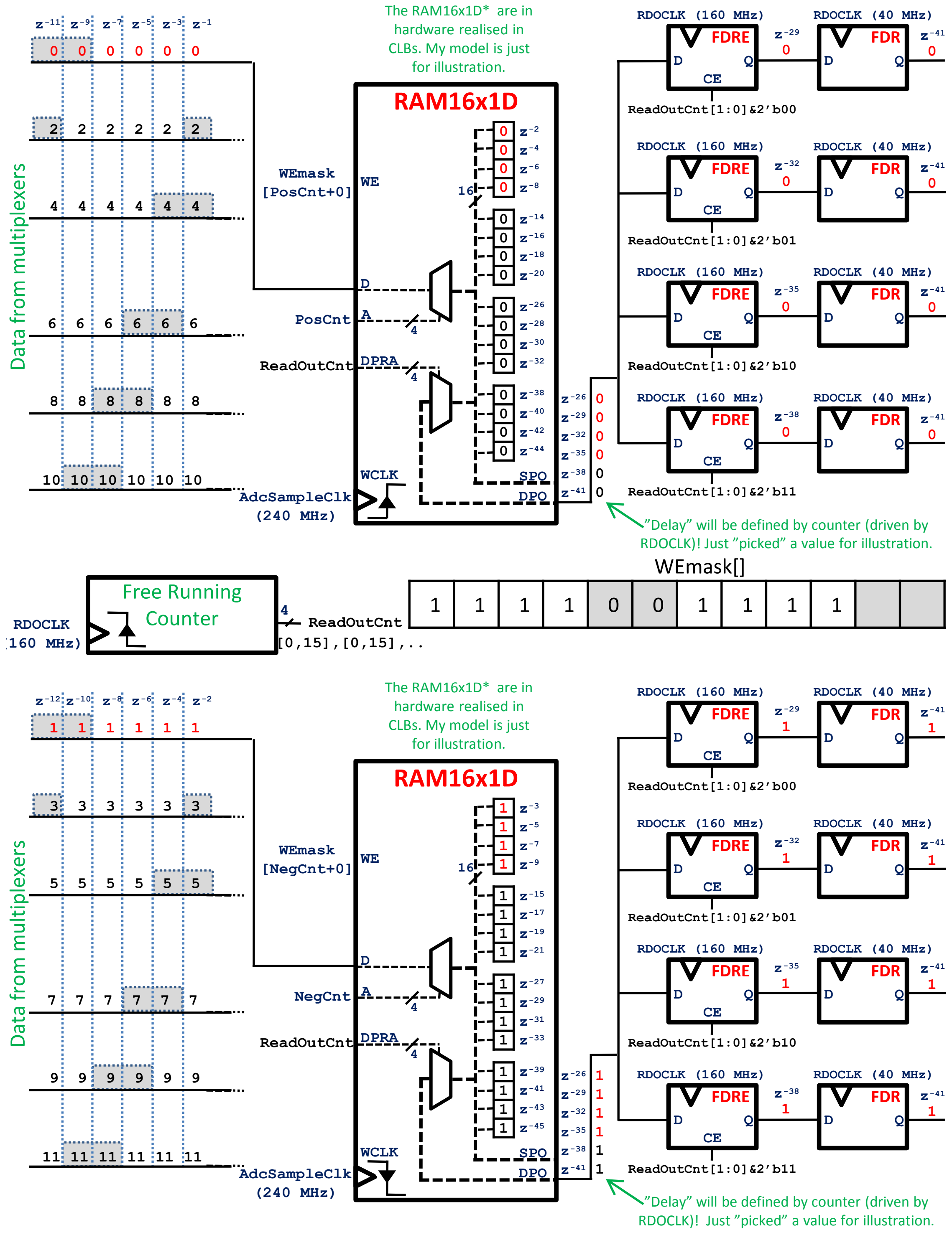


Fig. 2.11 - Deserialisation Stage 2 - Deserialisation and Phase Correction [sofPPT]

Resource Utilisation 2.2.6.5

I have previously claimed that this design requires only about half the logic of the deserialiser made by Alan. I feel that these statements might need some weight, so here are some numbers for you (tab. 2.6):

Part of Design:	**Type of Logic:**		**Logic Usage:**	
	LUTs	**Registers**	**Slices**	**CLBs**
Alans design:				
Shift Chain FlipFlops		1344	672	168
Extension Stage		1344	672	168
Output Synchronisation		1344	672	168
TOTAL		**4032**	**2016**	**504 of 5 904 (8.537%)**
My design:				
Shift Registers	288		144	36
4-input MUXs	672		336	84
Distributed SelectRAM	336		168	42
Output Synchronisation		448	224	56
Counters		90	45	11.25
LUTs	15		7.5	1.875
TOTAL	**1311**	**538**	**924.5**	**231 of 5 904 (3.9%)**

Tab. 2.6 - Deserialiser Logic Usage - Compared with Alans Deserialiser

If you have no idea of what I am talking about here, I suggest you take a look at the subsection where the Virtex 2 Pro logic was discussed (see 2.1.2).

Running Into Problems 2.2.6.6

When I designed this deserialiser I made the bad mistake of assuming that ChipScope would be there for design end-verification and fine-tuning. Even though the ChipScope core itself stays stable for clock speeds up until 400MHz, the signals routed to the core will naturally be bothered with internal propagation delays which basically renders verification at speeds greater than ≈80MHz impossible (as mentioned in last subsection, see 2.2.5.2).

Nevertheless, I was pretty sure that I would be able to get it working. I had ModelSim (see A.2.4) to simulate my design, and I could control what bitstreams were generated by the ADCs and read the samples produced with ChipScope. However, while I was trying to get this design working I had not yet discovered the issue with the input impedance mismatch (see 2.2.5.3). For the most part I was debugging the design with the clocks supplied from the ADCs and a bad buffer configuration, which were doomed to fail.

Since this design is entirely made up of CLB logic (see 2.1.2.4) the design should be able to easily handle the speeds required. The limiting element when it comes to performance seems to be the shift registers (as can be seen in the Virtex 2 Pro datasheet [docV2PDS, page 103]). If I ever had time to retry this design with proper input data and still found that it failed, the first thing I would try is to make the outputs of the shiftregisters synchronous by connecting the output to a register (which provides superior clock-to-out performance).

Conclusion *2.2.6.7*

In the aftermath of working with the deserialising scheme for the TRU I would certainly say that a few valuable lessions has been learned, the most important being:

- If you are having problems with interfacing external signals *make sure you know what input data you are dealing with.*
- If you are planning your design *make sure you will be able to verify and debug the design when it is done*, or
- If you are constructing a high speed design assume that post-implementation verification tools (like ChipScope for Xilinx FPGAs) will be absent. Only start on such a design if you know the timing parameters by heart, and feel confident that can get the design working without the mentioned tools.

It might seem odd, but the deserialiser was one of the first designs I made. The development of the ADC communication protocol and bit-/frameclock adjustment protocol, and the research into clock routing networks and timing parameters were all subsequent actions. I had no real idea of the scope of the deserialisation/synchronisation problem when started working on this design, which is why this design was sort of started "in the wrong end". The importance of a subject called called *timing closure* was initially ignored.

Timing closure is a term defined by digital designers which describes the challenge of getting a design to meet the timing constraints. A nice summary of the techniques available for tuning your design to meet the timing specifications can be found here [docTIMCLOS]. From my own experiences, consultations with other digital designers and some reading on the Xilinx forums I have come to conclude as follows:

- *Always strive to keep the level of constraints specified at a minimum.* As soon as you start specifying constraints you are interfering with the natural flow of the implementation software, hence the general quality of the implementation decreases and the time it takes to implement the design will increase[34]. Look up the term *over-constraining* to see what I mean.
- *If your design does not achieve timing closure without constraints, always reevaluate the design first to see if you can perform changes in coding style that can resolve the issue.* There seems to be a general consent among digital designers in this respect; if the automatic implementation tools can not find a way to place the design according to the timing and placement constraints specified by the designer, changes are that there is a flaw on the design entry level.

These were the main arguments for developing the deserialiser described in this subsection. But what if it refuses to work no matter how well it gets polished?

[34]For the TRU design I have had implementation times ranging from half an hour to a couple of hours depending on the constraints I set, and they were never many.

I believe that for the TRU design, only two deserialiser designs will do the trick: The Distributed SelectRAM and serial-in/parallel-out variant. See comparative summary below:

Distributed SelectRAM deserialiser

Advantages:

- Output runs at 40MHz and is phase independent from input, hence no problems should occur with frameclock (and constraints will not be needed).
- Will require 45% to 85% less logic than the serial-in/parallel-out deserialiser*. The distribution of logic will also be better as there are more LUTs available than registers.

Disadvantages:

- Design is more complicated, and harder to verify since ChipScope does not support the operating speed of the input stage.
- The counters on the input stage need some finetuning the the indexing to be correct (this can be done by using the ADC communication interface to pulse known patterns**.

Serial-in/Parallel-out ("simple") deserialiser

Advantages:

- Simple design. The deserialiser itself takes only a few minutes to design. Constraints are pretty easy to set as register elements is the only logic type utilised.
- "Less things can go wrong". This is an aspect which one learns to appreciate, especially at speeds when verification tools can not be used.

Disadvantages:

- The output is hard to synchronise as the frameclock must hit the "valid data" window of ≈2ns.
- Location and timing constraints must be used in order to get it working.
- Will use 16% to 122% more logic than the Distributed SelectRAM deserialiser*, and the logic type utilised is more unfortunate (there are fewer registers available than LUTs, and this deserialiser utilises only registers).

* *The logic utilisation difference may vary depending on whether synchronous multiplexers and shift registers are required in order to make the Distributed SelectRAM deserialiser work, and whether the extension stage of the simple deserialiser becomes redundant after post-implementation tweaking.*

** *The finetuning of the Distributed SelectRAM can be done by using the phase-adjustable clocks (controlled with register **0x79** and **0x7a**) to clock the counters.*

Just before I left CERN I was discussing further actions on this part of the design with my successor, Dong Wang, which is a student from China. Previously he had been working on the design of the FEE BoardController, and had knowledge of post-implementation logic tuning from this and previous projects. Basically, there was never any doubt of which path he wanted to go; tuning the serial-in/parallel-out deserialiser. The Distributed SelectRAM deserialiser would have required some time "getting into", and time is a critical factor.

A few weeks later he had successfully synchronised all ADC channels by using a program called *Floorplanner* for setting up location constraints[35] (specify where you want your design elements physically placed in the FPGA) and *FPGA Editor* to change some routing paths[36]. When input buffers with Digitally Controlled Impedance were used the clocks from the ADCs seemed to work nicely, so these were used. In addition, he used the *Synplify* synthesiser (see A.2.5) instead of Xilinx XST (as I was stuck with in order to be cross-compatible with the designers which were working with other versions of the TRU).

[35] After Floorplanning you get a lot of new constraints which look like this:
`INST "<hierarchical path to instance>" LOC = "SLICE_<x-y-coordinate>".`

[36] Dong told me that the implementation software insisted on routing all clocks through the global multiplexer buffers, which had to "fixed" as this is naturally not what we want to do.

While Dong Wang found a way to achieve timing closure on the deserialiser design, the method of constraining the design down to specific logical elements in the FPGA were not optimal. He later told me that as soon as he started making edits to other parts of the TRU design, the timing closure tended to break leading to synchronisation failure on several ADC channels. Also, when more than 90% of the available CLBs were used the constaints placed on the deserialiser would cause other parts of the design to fail. While this substantiates my previous arguments about being careful with the usage of constraints, it may not be a way around it. We push the Virtex 2 Pro FPGA to its limit, so making it work perfectly may never be possible. With this in mind, I suggest that the Distributed SelectRAM deserialiser is revisited if:

- there is a critical shortage of register logic, and there is no other “obvious” way of improving this situation.
- the implementation process is getting too slow to work with.
- the design starts to suffer from overconstraining (using constraints to “fix” one part of the design only causes new problems to appear elsewhere).
- no other parts of the design has higher priority.

No matter the future development on this area, I will look back at a lot of fun designing the deserialiser and probing into the topic of timing closure.

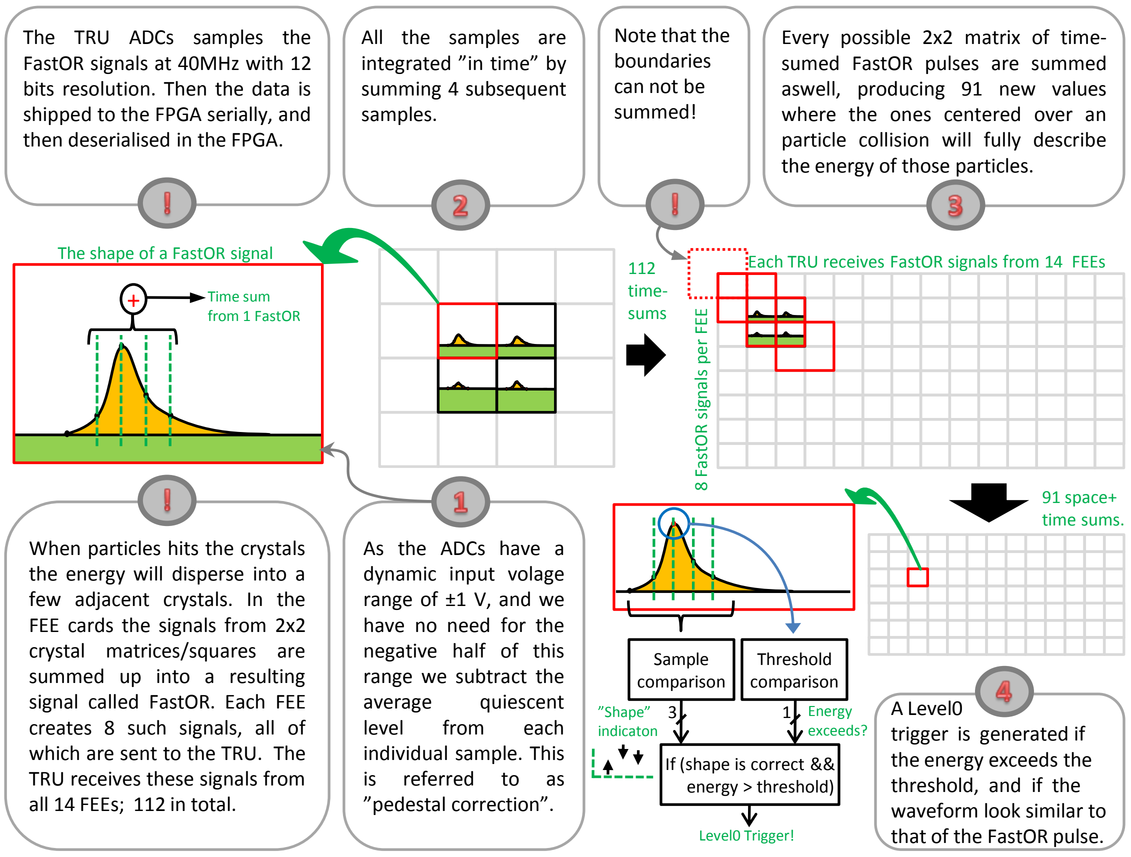
You may have noticed that I avoided discussing specific timing parameters and their impact on the design. Presenting the numbers and following them up with evaluation would have made this report a lot bigger and more technical, so I have tried to “conclude with words” instead. In fact, Xilinx spokesmen claim[37] that few digital designers ever get completely familiar with all of the timing parameters (I assure you, setting up the constraints file by hand is easier said than done). I assume that this is why you see that Xilinx releases several software packages these days which provides a graphical approach to the timing closure topic (Floorplanner and FPGA Editor is only two out of many).

As a final tip I would like to suggest that a “spike filter” is added behind the deserialiser, one for each ADC channel. The cause of the spikes are unknown, although I guess it is inherent to point out unstable synchronisation or “bit-flips” in the ADC digitising process as prime suspects. Just a single spike may cause a fake Level0 trigger (depending on the size of the spike and the threshold levels specified), but these can easily be filtered. It may simply be enough to subtract the previous sample from the current and assume that it is a spike if the difference is greater than a certain specified value. A real FastOR pulse will always be “smeared” out in time.

[37] I spoke to some Xilinx spokesmen on a Xilinx seminar at CERN, 28. May 2008.

2.2.7 Level0 Trigger Calculation

As I have come to mention several times during this report - but never really thoroughly discussed - the TRU is in charge of the Level0 trigger generation. The changes I made to this part of the design and tips for the future will be provided throughout this subsection, but let us start with the principle (fig. 2.12):



The TRU ADCs samples the FastOR signals at 40MHz with 12 bits resolution. Then the data is shipped to the FPGA serially, and then deserialised in the FPGA.

All the samples are integrated "in time" by summing 4 subsequent samples.

Note that the boundaries can not be summed!

Every possible 2x2 matrix of time-sumed FastOR pulses are summed aswell, producing 91 new values where the ones centered over an particle collision will fully describe the energy of those particles.

When particles hits the crystals the energy will disperse into a few adjacent crystals. In the FEE cards the signals from 2x2 crystal matrices/squares are summed up into a resulting signal called FastOR. Each FEE creates 8 such signals, all of which are sent to the TRU. The TRU receives these signals from all 14 FEEs; 112 in total.

As the ADCs have a dynamic input volage range of ±1 V, and we have no need for the negative half of this range we subtract the average quiescent level from each individual sample. This is referred to as "pedestal correction".

A Level0 trigger is generated if the energy exceeds the threshold, and if the waveform look similar to that of the FastOR pulse.

Fig. 2.12 - Level0 Trigger Generation [sofPPT]

1. Subtract the pedestal value from each individual sample.
2. Sum 4 subsequent samples of each FastOR pulse in order to "integrate" the energy of this pulse in time.
3. Sum all possible 2x2 matrices of FastOR pulses in order to "integrate" the energy of total photon shower (the light will disperse into several adjacent crystals).
4. Generate a Level0 trigger if:
   - *energy exceeds that of the trigger condition.* This is accomplished by comparing the 91 scalars with a set of given thresholds.
   - *the shape of the signal seems correct.* A simple comparison of the samples is done here; I defined a "valid" pulse as one "up" and two "downs". This will make sure the trigger is always generate at the same "spot" on the pulse, and filter out some unfortunate conditions which might affect the Level0 energy evaluation (like spikes).

Motivation *2.2.7.1*

When I received the TRU design code and started development I quickly encountered a variety of problems related to a shortage of free logic resources in the FPGA (have a look at appendix D for a summary from the *map* process (part of the implementaion processes), where logic utilisation is listed for the old and new design). This naturally lead to the research into how the design could be optimised, and my conclusions can be found below - along with my choice of action.

Design Changes *2.2.7.2*

- *The design used an excessive amount of CLB registers.* When the base clock speed is 40MHz there is no need to synchronously buffer your signals for every logical/arithmetical operation that is carried out. This only makes the design slower as for the greater part of a clock cycle nothing happens. The time margins will be very good margins (for setup, hold and clock-to-our time), but you will use more logic to "remember" data than you need to.
  - ▶ Lead to the redesign of "SumInSpace" and "SumInTime". Within a 40MHz clock cycle there is plenty of time to perform both of these arithmetical operations in a combinatorial way, the only bits we need to store with registers is the final sum.
- *Design looked messy due the previous designers using Verilog95 syntax.* Verilog95 was a very limited HDL language feature-wise. It lacked multidimensional array and adaptive instantiation capabilities, thus the design code became incredibly long and hard to follow (as replications were hardcoded). Take a look at the old design to see what I mean (see [srcTRUDESIGN]).
  - ▶ I rewrote all modules discussed in this subsection to Verilog2001 syntax, which drastically reduced the amount of code and made it less error-prone (as replication of logic can be done syntax-wise and not with a copy-paste/hard-coding approach.
- *The old design used an algorithm to find the "peak" of the pulse which produced a Level0 trigger.* Is this necessary?
  - ▶ I redesigned this to what was already mentioned on the last page, the "shape indication" mechanism. If the trigger criteria itself contains not only an energy evaluation but also a "weak" shape evaluation, the triggerpulse will always be in the same position when the Level0 trigger is generated (the "peak" will always be the second sample).

However, there is still work to be done. Tips for digital designers in the future can be found in the conclusion on the next page.

Conclusion *2.2.7.3*

I have not encountered any problems related to my design changes (not much can go wrong here really, this part of the design is pretty simple). However, I believe that the design should still be improved in certain areas, and some insight I have gained needs to be communicated. We start with the pedestal correction and work ourselves back to the Level0 trigger generation.

Ideally, when the ADCs digitise the FastOR input signals, an reference input voltage of 0V should be represented with a sample value of 2048 (the center value of a 12 bit sample). However, the digital values have a typical offset error of ± 8 and a maximum error of ± 30 counts[38]. To correct for this undesirable effect and remove the pedestal from the FastOR signals, the average value the last 32 samples are subtracted from each new sample begin produced. There are two important factors I feel the designer should be aware of here:

- *The averaging function should be improved.* There is no way you can store 32 subsequent samples in order to calculate the average. What you must do is remember the average value of the last 35 subsequent samples and have the value of the most recent sample affect the average with a factor of 1/36 (this solution was used in the old design). However, this means that every time a FastOR pulse comes along a positive offset will be added to the average. This is undesired, but can be easily fixed by *locking* the averaging function when a new sample deviates from the previous average too much (and add some logic to make sure the function is not locked when the TRU is booting up).

- *Do not throw away the MSB just because the pedestal is removed.* What happens if you subtract the pedestal of a channel where the samples has an average value of, say 2035, and then throw away the MSB? The dynamic range would be `2`^{12}`-2035 > 2`^{11}, which means that *11 bits is not enough to hold all the values in the dynamic range.* You could fix this by AND-ing the 11 LSBs with the MSB, and then use the 11 LSBs of the results , but to me it does not sound resource efficient to to create 11 2-input AND gates in order to avoid storing an extra bit.

[38]The ADS5270 datasheet [docADC] specifies a typical DC offset error of $\pm 0.2\%$, and a maximum $\pm 0.75\%$ of full scale (equals ± 8 to ± 30 values).

The following factors related to the time- and space-summation should be looked into:

- *Perform the time-sum before the space-sum.* This might sound confusing to you, as in all my illustrations and discussions I have claimed that the time-sum was performed before the space-sum. However, this was not the case in the old design and I never got around to change it (a great part of the design process is planning for the future). Since we are only talking about summations here, which is a *commutative* arithmetical operation, the sum will be equally valid no matter the order of the summations. Due to the trigger timing constraints in ALICE (discussed in 2.2.8) only 1 sample per ADC channel can be sent to the TOR, inherently suggesting that this sample must be a timesum. It should also be a timesum of the raw data (and not spacesums), as this enables the TOR to re-do the space-sum across branch boundaries (see page 25, and 2.2.8.3).

- *Use Block SelectRAM for FakeALTRO and TOR data transmission storage.* There are two types of RAM in the Virtex 2 Pro FPGA, small and fast Distributed SelectRAM blocks implemented in the CLBs and big (but slower) Block SelectRAM (see 2.1.2 for logic description). Distributed SelectRAM will be used by the ChipScope core (but only in the development phase) and for my own version of the deserialiser (if it ever enters the design again).

  The Block SelectRAM was not used at all, but they are perfect for storing data for the FakeALTRO protocol and for data transmission to the TOR. The former needs access to a history of pedestal corrected data, and the latter needs access to time-summed data. If a single Block SelectRAM block is configured as dual-port with a port-width of 36 bits it will be 512 addresses deep (`36*512=18432 bits`) [docV2PDS, page 55]. This means that no matter the type of data, it should not pose any problem to store 2 samples/timesums per Block SelectRAM block per cycle. There will be a total of 112 samples and 112 timesums to store per cycle, so a total of 112 Block SelectRAM blocks will be required (there are 232 blocks in total) with this approach. The depth of 512 addresses means that a data history of `512*40MHz=12.8`μ`s` can be retained.

2.2.8 TOR Communication

The TOR will - when all 5 PHOS modules are installed - communicate with 40 different TRUs. Since the TRU is short on resources it is vital that we properly define this communication protocol such that it can be taken into account in the design planning process, ultimately leading to a nicely integrated solution.

Motivation 2.2.8.1

In June/July 2007 I worked with Alan Crouau to come up with a design for the TRU/TOR interaction (the communication is parallel for all TRUs so I will only discuss the relationship between one TRU and the TOR). The communication medium is a category 7 (CAT7 (Category 7)) TP (Twisted Pair) cable[39] which provides 4 LVDS lines (so 4 signal lines per TRU). Our initial approach was to use one of the 4 available transmission lines for the Level0 signal, one as a busy line (for the TOR to tell the TRUs when to start pushing data), and the two remaining lines for the data itself.

However, in the above mentioned scheme we were a little optimistic as to what amount of data we would be able to send. It was also a pure data transmission scheme, and other aspects of the communication between the TRU and the TOR were improperly defined. The design needed to be revised. The key reasons for my involvement here were as follows:

- The communication protocol must be implemented on both the TRU and the TOR side. Since I worked extensively with the TRU and was familiar with its various design aspects my input would most likely be of some value.
- In order for the TOR to be able to calculate the Level1 trigger it needs some data - from which the Level0 trigger was based - sent from each TRU. Which data type and size that are chosen will have a major impact on certain parts of the TRU design. It has to be stored in a memory until the TOR requests it, and when the request comes the right set of data needs to be retrieved from the history retained. As the TRU is short on resources we *do* want to find the best (or cheapest, logic-wise) way to do things.

The current design suggestion is illustrated in fig. 2.13. It is expected to represent a fairly good compromise between timing, data value and logic usage. The model also introduces the concept of command encoders/decoders, which will (if implemented) render possible more powerful and feature-rich communication methods.

[39]The CAT7 cable is rated for transmission speeds up to 600 MHz.

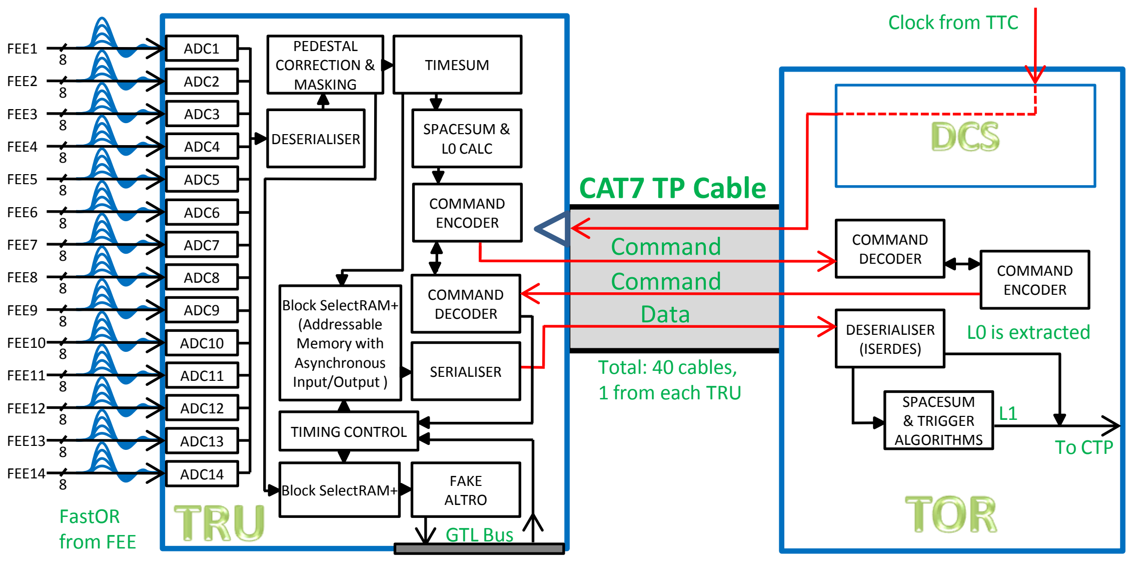


Fig. 2.13 - Design Suggestion for TRU-TOR Communication [sofPPT]

Timing *2.2.8.2*

The TOR has two clock sources - the internal crystal and the clock received through optical fibres from the TTC (Timing Trigger Controller). When present, the latter should be used in order for the electronics to be synchronised with the LHC clock[40]. From the TOR the clock is sourced to the TRU which makes it easy to synchronise communication in this direction. The TRU will use the same clock for clocking data onto the bus, and the necessary phase adjustment will be committed in the TOR through the use of a DCM. Since we must use the same phase correction for all incoming lines to the TOR (there is not enough DCMs available) it is very important that the CAT7 TP cables between TOR and TRU have the exact same length.

The main limitation when it comes to transmission speed seems to be the TOR routing [docLSF] [41]. Some channels are downright bad, but we were able to push the clock frequency to around 200MHz on “good channels”, or effectively 400Mbps when a we use both edges of the clock to clock the data (DDR). The transmission capacity can be expressed as (eq. 2.1):

$$\begin{aligned} Transmission\ Capacity &= t_{datatransfer} * f_{effective} * n_{lines} \\ &= t_{datatransfer}\ [\mu s] * 400\ [bits/\mu s] * n_{lines} \end{aligned} \tag{2.1}$$

[40]If the internal clock are used there will be an additional clock domain the TRU must cope with, there is currently no support for this.

[41]Since the routing in the TOR is proven to be poor, it will probably be replaced by the STU (Summary Trigger Unit) in EMCAL in the future

Where $t_{datatransfer}$ represents the time it takes to send all necessary data from the TRUs to the TOR. But how much time do we have?

If the Central Trigger Processor (CTP) in ALICE validates an initial Level0 trigger condition (see 1.5.1), a Level0 "accept" signal will be sent back to the subdetectors to initiate the process of determining whether the event also qualifies for a Level1 trigger. In PHOS the Level0 "accept" signal will arrive at the TOR ≈1.2μs after the particle collision. When this happens, the TOR must initiate a readout of all the TRUs in order to get the raw data o the event. This data will be reevaluated with a different set of criterias (physics related) to see if a Level1 condition is justified. If it is, Level1 trigger must be at the CTP in less than 6.5μs after the particle collision. This leaves us with a timeframe of ≈4.8μs (see eq. 2.2) for the data to be shipped from the TRUs to the TOR and to do the necessary Level1 calculations.

$$\begin{aligned} t_{window\ for\ TOR} &= t_{Level1\ @CTP} - t_{propagation\ delay\ TOR->CTP} - t_{Level0\ \text{``}accept''\ @TOR} \\ &= 6.5\ [\mu s] - 500\ [ps] - 1.2\ [\mu s] \\ &= 4.8\ [\mu s] \end{aligned} \tag{2.2}$$

To leave some time for the Level1 calculations, we aim to get the data sent from the TRUs to the TOR in less than ≈3μs. As you can see from fig. 2.13 we intend to use only 1 line for data, which means the total amount of data sent per line should not exceed ≈1200 bits (given by eq. 2.1).

Data Transmission *2.2.8.3*

A TRU has 112 ADC channels (14 ADCs and 8 channels per ADC). We can now see that - if we only send 1200 bits from each TRU - we can see that we can only send `1200 [bits] / 112 [channels] ≈ 10 [bits per ADC channel]`. What data can we fit within these constraints?

When the analog FastOR signals arrive at the TRU they are first sampled, deserialised and pedestal corrected. The data has now a resolution of 12 bits, and since the FastOR pulses are ≈100ns long 4 samples will cover it ($T_{sampleclock} = 25$ns). It should be obvious that we must send timesummed data (energy integral), as we can not send 4 samples with a fair resolution from each ADC. The ADCs has a dynamic range of ±N V, so only 11 bits are needed to hold the pedestal corrected values (but do not throw away the MSB of the sum to achieve this, see 2.2.7.3 for more information). When 4 subsequent samples are summed two more bits are needed, so the net resolution must be 13 bits.

Since it does not matter whether the timesum or spacesum is performed first (the calculation is commutative, see 2.2.7), I chose to do the timesum first. This enables us to send the data untouched by the spacesum calculation, thus circumventing the branch "edge-effect" [42]. The spacesum calculation can be implemented with combinatorial logic (and low logic usage) in the TOR, ultimately providing the same triggerdata but without the undesirable edge-effects (except for the module edges of course).

We will need to store the timesums before they are sent to the TOR. This for two reasons. The first is that the TRUs has two clock domains - the clock received from the TOR and the readout clock from the RCU. The latter controls the ADCs, so we must use some sort of temporarily storage just to synchronise the data stream between these two clock domains. The second reason why we need to store the data is because the actual readout request (which comes from the TOR) will arrive at the TRUs several clock cycles after the trigger data which caused it.

The memory resource I suggest are used for this storage purpose is the Block SelectRAM (refer to 2.1.2.6 for more information), which is integrated in our Xilinx FPGAs. These RAM modules deliver to the table 18kB of storage and dual asynchronous ports with a adjustable width of up to 32 bits. I suggest that the max width is used, and that 2 timesum scalars are written to each of the RAM blocks per clock cycle. The timesum from all 112 ADC channels can then be clocked into 56 RAM blocks with space to spare. Add the 56 RAM block I suggest are used for raw data storage (for FakeALTRO protocol) and we end up at a new total of 112 *Block SelectRAM* modules - of the 232 available in the Virtex 2 Pro FPGA in the TRUs. With this setup the Block SelectRAM has an addressing depth of 512 bits, which means that a data history of 12.8μsis retained.

You may ask why I recommend to use the *Block SelectRAM* blocks instead of the *Distributed SelectRAM* memory modules. This is because the former has a much greater storage capacity, and there is frankly not much else to use them for. The latter has a much greater range of application (can for example be used for deserialising the ADC data, see 2.2.6), are used by ChipScope and the various synthesisers are better at automatically utilising the features they provide. When doing digital design for the TRU we can not afford the luxury of choosing the "easy" solutions, as they usually involves spending more logic than necessary.

Before the data are sent onto the bus they need to be serialised. This should be a pretty straightforward process so I will not discuss it here, but rather move on to the deserialisation process (at the TOR). I suggest that the *ISERDES* (Input SERialiser/DESerialiser) primitives are used for this purpose. They do not offer out-of-the-box 13-14 bit word deserialisation, but they offer 7 bit word deserialisation. Send 14 bits from the TRU and let it chew a couple of times before you merge the two 7 bit parallel words into the correct 14 bit word. It will be completely up to the TOR designer to decide what he/she wants to do in this matter.

[42]The TRU can only "see" one branch of FastOR signals, so the spacesum will be "broken" around the edge of its region.

Command Exchange 2.2.8.4

When the TOR receives a Level0 signal from one of the TRUs, it will issue a busy signal to all the TRUs indicating they should enter a state where no subsequent Level0 signals will be produced and data readout can commence. To make communication more capable and versatile we decided to encode the commands into 10 bit words (and clock it with the same clock as used for the data). Of course, other package sizes might also be worth considering.

The packages should contain 1 bit for parity check (to detect single bit errors), then the rest it will be up to the designer to decide. Use a bit for Level0 trigger maybe? And one for acknowledge? Maybe set it up such that the first package indicate what packages (or package types) are to come? This way you can read/write all the TRU registers without even touching the GTL bus, which might prove to be a much more enjoyable experience.

Also, if this is implemented there should be no problem sending some extra information over the "communication lines" for example about compression properties (as not all data will contain any valuable information). With this setup you could select the channels worth reading out in the TRU, then sending the "readout pattern" over the communication lines while the actual data is sent over the data lines. Or, since the capacity are there send all data over the dataline, and some information about *why* the channels triggered. Again, the possibilities are many for future designers.

Conclusion 2.2.8.5

The TOR-TRU communication protocol discussed in this subsection should only be considered a draft. It is hard to reach a good conclusion here as I have never implemented this design and experienced its "validness". I suspect that there might still be flaws in the design, but I am not able to point them out at this point.

That being said, it is not even sure it ever will be implemented. However, even defining alternative designs might prove useful. How can you as a designer promote a specific design solution if you never considered the alternatives? During my work at CERN I have really learned the importance *properly defining* design solutions, as you can avoid a lot of potential pitfalls with careful planning.

It is like my supervisor tended to say, "the easy solutions generally works best". Naturally, if it does not sound easy then there are aspects of the solution which you are not fully aware of. Both the initial design process and future maintenance are likely to cause you headache in these situations.

2.2.9 *Testbench*

I spent a lot of time with the testbench[43] to verify various parts of my design with ModelSim simulations. Despite this fact, I will not spend too much time explaining the "ins" and "outs" of my testbenches. I suspect future designers will find their own methods for performing pre-synthesis verification (like simulation), and as such never bother to resume the work on a somewhat messy alternative testbench.

A few inputs in the TRU design are always stimulated in the same way. The 40MHz base clock needs to be simulated, and all signals tied to the GTL bus has to be tri-stated (as this is how this bus operates). Also, to make sure that the design boots up properly I usually always produce a reset signal a few μs into the simulation time. Most of the testbench design will of course be specific for what you want to test. For me, these three parts of the design created the most noteworthy testbenches:

1. *ALTRO Communication Protocol.* This was the most thorough version of the testbench. It read the file `rcu-commands.txt`, interpreted the commands listed in this file and "acted" as the RCU when it controlled the various signals on the GTL bus. This way I found a few bugs in the GTL driver protocol in the BoardController, and generally learned a lot about testbench design and the ALTRO protocol[44].
2. *ADC Data Deserialisation and Synchronisation.* This testbench was basically generating 112 serial data streams to simulate the 112 ADC channels. What words were sent, and phase delays from channel to channel could be adjusted.
3. *Level0 Calculation Algorithms.* To make sure that the Level0 trigger were produced under similar conditions as before I redesigned this part of the design, this testbench generated some artificial samples for each ADC channel, and every now and then values indicating a FastOR pulse would come along. The thresholds was naturally also set with the testbench.

The most valuable lession learned from my work with testbenches is that "object orientation is your friend". *Hardware is per definition "objects"*, so any other approach in the testbench design will lead to messy, hard-to-read and less flexible testbenches. However, Verilog2001 is not an object-oriented language. The new System Verilog HDL (Hardware Description Language) language will be, but it is not fully supported by the various synthesisers yet.

Look up the sourcecode for the testbench and other parts of the TRU design in the bibliography (see [srcTRUDESIGN]).

[43] A *testbench* is a module which encapsulates the design. A testbench normally generates stimuli for the design inputs, and monitors the design outputs.

[44] The ALTRO protocol needs to be understood in-depth when the FakeALTRO protocol are developed. For a while I was working on the FakeALTRO protocol, and had the entire GTL bus hooked up to a logical state analyser in order to debug it. However, it simply takes too long do post-implementation verification, so I planned to use this testbench for FakeALTRO protocol debugging. However, time was short and priorities were changed before I was able to complete this work.

2.3 Conclusion

I have gained very valuable experience from my work with the TRU design. I performed a lot of research into the Virtex 2 Pro FPGA technology, and wrote design solutions for it with the Verilog2001 Hardware Description Language (HDL). I designed clock distribution schemes, communication protocols, investigated data integrity issues, came up with a very specialised deserialiser, made countless testbenches and simulations (hence learned ModelSim) and rewrote/redesigned almost the entire part of the design dealing with the ADCs and Level0 trigger generation. I have spent many evenings studying application notes from Xilinx designers and probing forums wanting to learn more about FPGA design techniques. This was in order to be able to design logic which not only works well, but also "looks elegant" and are properly documented and commented.

But maybe the most valuable lession of them all was learned when I realised that when working on big projects and with other people, you must use version numbers and log the development[45]. Not only will this keep your own head straight when you are sitting there late at night tapping your keyboard realising that your sense of the general picture is long gone, but it will help others to keep track of your work[46] and it serves an important role in the aftermath where you have to sit down and think about what you where actually doing.

I also performed all my development in a Operating System (OS) called Linux, which were not done by any of my predecessors. This naturally involved a little hassle in the beginning, but I gained a lot from it in the end both with respect to the general quality of my work and in knowledge. I wrote down some of my experiences with Linux in appendix A.2.

As I mentioned in the introduction of this section about the TRU, I hardly made any edits to the BoardController. My only contribution here was a clean-up of the port-lists (remember, in Verilog95 they must be redefined 3 times, as opposed to once in Verilog2001) and the `register_block` module, and a small correction in the GTL driver protocol (as it was not working properly). Since I hardly spent much time with this part of the design, I will be careful about stating anything about what needs to be looked into here.

Finally, when looking through my log files and I see that there are more topics than I can possible start to discuss in this report. I initially planned to discuss in-depth the various technical aspects of the design, and walk through the logs created by the implementation tools to add some weight to my statements, but realised that this was not feasible. In order not to lose my intended readers I have tried to the discussions on a "concepts only" level. The topic I chose to document here were based on a selection where "time spent on the topic", "topic importance" and "how well the topic blended into the general layout of the report" was the main criterias.

[45]The log from the last part of my design process can be found in the BackUp folder in the TRU design (see [srcTRUDESIGN]).

[46]My version numbering started from `0.0.00`, but was not a good idea as the previous versionnumber for the TRU design was 0.2. This will most likely cause confusion one day, so I am sorry about that. It was not done with intention.

3 TriggerOR

The main purpose of the TOR (Trigger OR card) is to calculate the Level1 trigger (see 1.5.1 for more details about the triggersystem in ALICE). While the Level0 trigger is created when the energy of some FastOR signals exceeds a certain threshold, the Level1 trigger needs to be calculated based on a certain set of physics criterias.

A summary of my work related to the TOR can be found in the list below:

- Verifying the TOR pin-out mapping.
- Stabilising the LVDS inputs.
- Digital Design with the TOR FPGA.
- Definition and specification of the communication between the TOR and the TRUs.

Verifying the TOR pin-out mapping and creating the BusController/pulsegenerator was done very early in my placement, as a means to learn Verilog 2001 (the programming language used here) and digital design - programming of FPGAs. These topics will therefore not be very technical.

The communication between the TRU and TOR is subject to ongoing discussions, where the exact protocol to be used is not yet completely defined. However, during my stay I have spoken to several people regarding how this should be defined - and I have spent numerous hours in my office with my supervisor (Hans Muller) going over numbers and constraints to make sure that the design is possible to implement. The result from this work can be found in the last section in this chapter.

I received the TOR code from a French student named Alan Crouau, which had started to write the TOR design from scratch and had some basic functionality in place. I will notify the reader of when I am talking about his work and mine.

3.1 TOR Pin-out Mapping

3.1.1 Motivation

The TOR is connected to the 40 TRUs in PHOS, distributed over 5 modules which translates to 8 TRUs per module. The chosen media for signalling between TOR and TRUs is category 7 twisted pair TP cables, one from each TRU. As described in A.5 it is important to get the mapping correct with these cables, as bad mapping translates to bad conditions for high speed communication. When working with a FPGA, correct mapping means connecting the pin-names (or rather coordinates, since this is how they are indexed) to the correct design variables (in the code).

3.1.2 Verification

The mapping of pins to design variables is an operation which fall under the category *location constraints* in the design. As with all constraints which can be user-modified they should be written in the constraint file (filename `*.ucf`)[1]. When I received the TOR code from Alan Crouau he had set up the location constraints needed for the buscontroller (see 3.3) and for a few of the RJ45 connectors (which he used for data transfer from the TRU to TOR at 400MHz, see 2.2.8).

The verification were carried out in the following way:

- I wrote the constraints file from scratch whilst keeping the pin names according to design sheet [docTORDESIGN]. I tried to be consistent and keep the design names as signal names (usually capital letters), such that it will be easy to spot what is an IO variable and what is not in the design. I also grouped pins into two- and threedimensional arrays since this makes the design cleaner. I have no idea why this was not done before.
- When I finished writing the constraints file, I made a small `C++` application (find the sourcecode in the bibliography, [srcMAPCHECK]) which compared my new constraints file with the previous one, and outputted the deviations (it might sound silly, but it is the best way to eliminate the "repetition-makes-you-sloppy" effect).
- For the few deviations, I doublechecked the values and corrected them accordingly.

While I was at it, I performed the same operation on the TRU code (but it will only be discussed here).

3.1.3 Conclusion

I believe that the mapping of the TOR should now be verified. During my more recent work on this card I have not encountered any effects where it is natural to suspect the mapping. I added all the signals listen in the design sheet, so for all future development on this card there should be no need to change these.

[1] The use of the `ucf` file has - in addition to general "cleanness" of your code - the added benefit of not affecting the synthesis, hence lead to a reduction in recompile time.

3.2 *Stabilising LVDS Inputs*

3.2.1 *Motivation*

When the TOR is connected to the TRUs - which it receives the Level0 triggers from - we discovered that the triggerrate seemed surprisingly high. In fact, at times it more or less skyrocketed. While this might be caused by the TRU actually producing such a high number of triggers (unintentionally, if this is the case), I noticed that the TOR sometimes interpret floating inputs (no cable connected) as logical high and not low. This should not be the case as the LVDS inputs are internally (in the FPGA) terminated with a 50Ω impedance between the differential wire-pairs.

3.2.2 *Investigation*

The natural response here would be to try to get rid of the "floating" effect while maintaining the differential termination. The TOR FPGA supports several internal features here, and I ended up trying out all possible combinations of pull-up, pull-down or keeper (make the inputs a little more resilient to noise). The only way to get a "floating" level to get interpreted as "low" is by pulling the negative input buffer high and/or the positive complementary low *and* remove the differential termination (see fig. 3.1). The weak pull-ups/pull-downs seemed not to disrupt normal dataflow when the input were connected to LVDS cables.

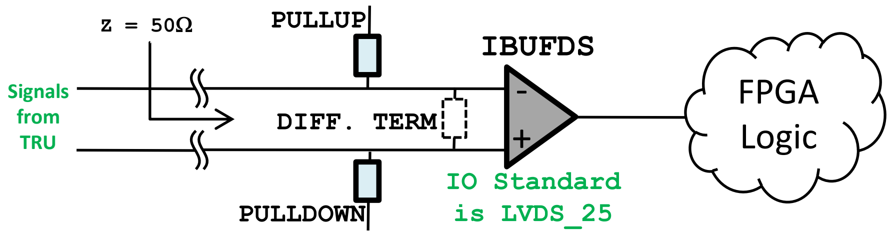


Fig. 3.1 - LVDS Input Buffer [sofPPT]

3.2.3 *Conclusion*

To be sure that you maintain signal integrity any input should always be terminated such that the energy of the signal is absorbed at the input and not reflected back towards the source. Even if the data integrity seemed good when the differential termination was removed, I would be reluctant to actually implement this as a means to solve the problem. There are still some options which should be investigated, for instance the use of input buffers with digitally controlled impedance. A workaround would be to leave the differential termination and use the mask register on the TOR to "mask off" the TRUs that are not connected or offline.d

3.3 Digital Design

3.3.1 Motivation

When I first arrived at CERN I was working with Alan Crouau on the documentation of the data transmission protocol from the TRU to the TOR. It is safe to say that back then I did not know much about digital design on FPGAs, but that had to change. Since Alan had just recently designed a BusController for the TOR RCU bus (the TOR card has a DCS card attached to it, just like the RCUs [2]) he suggested that I should look into it and pull some wires just for the sake of learning.

Half a year later I worked with a German student named David Königseder, which were designing a trigger multiplexer for switching between locally and externally generated triggers in PHOS. To test his design he needed a pulsegenerator, so I made a module in the TOR to provide this to him.

In the previous section I mentioned that we at times noted very high trigger rates from the TRU, and a possible reason for this is that the TRU possibly sends out heaps of triggers when a bunch of particles hit the crystals. We have a "busy-box" in production which is supposed to "hold off" subsequential triggers for a certain amount of time after the first one is produced, but it might be nice to have this busy functionality in the TOR aswell for situations where the busy-box is not present.

3.3.2 Operation

I would like to emphasise that the BusController is not originally my work, but the reader should be aware of the modification if he ever decides to dig up the code on from my source database (see 1.8 for more information) as it is the module currently in use in the most recent version. The BusController enables you to read and write to variables in your code with the familiar syntax `rcu-sh <w/r> <Reg No.> (<Value>)` from the DCS card. `Value` is only used when writing, and both `Reg. No.` and `Value` must be on the form `0x....`.

[2]The TOR has an "RCU bus", not because it is connected to the RCU bus in any way, but because the DCS firmware for the TOR is just a modified version of the DCS firmware for the RCU.

Look up the pulsegenerator as a module in the `TestPulses.v` file in the TOR design. The time between pulses must be defined when the module is instantiated, but the pulsewidth can be adjusted on-the-fly. For the pulsegenerator I set the BusController up with the following registers (tab. 3.1):

Register Name:	**Address:**
LVDSOut	0x3
FastPulseEN	0x4
SlowPulseEN	0x5
PulseWidthSlow	0x6
PulseWidthFast	0x7

Tab. 3.1 - Pulse Generator Registers

As you might suspect from tab. 3.1 I set the TOR up with up two pulsating lines; one sending pulses at 0.5s intervals and another at 2s intervals (hence the "FastPulse" and "SlowPulse" tags). Register `0x3` enables the LVDS outputs, `0x4` and `0x5` enables/disables the fast and slow pulses (non-zero value = enable), and the width of the pulses are defined in `0x6` and `0x7` where the value indicates the pulse duration in number of clock cycles (shortest pulse would then be 25ns, which is the period of the onboard clock).

I implemented the busy functionality, but never tested it. I suspect there is still some work to be done here for it to be fully functional.

3.3.3 Conclusion

The few mentioned designs above represent my work on digital design for the TOR. For the designer that will end up making the Level1 trigger (that is, actually doing the real design for the TOR) I recommend starting from scratch. Take the constraints file (which I believe should be 100% accurate) and the BusController (either mine or Alan's), and make the design your own.

4 Readout Control Unit

The RCU (Readout Control Unit) is in charge of communication with the FEEs and TRUs in PHOS. There are 4 RCUs per PHOS module, and each of these communicates with 2 "branches" of cards. Each branch has 14 FEE cards and 1 TRU. See fig. 1.9 for an illustration of such a branch and the position of the RCU (only one branch is shown but the RCU is - as previously mentioned - also connected to an additional branch).

My work on the RCU can be split into two parts:

- Writing the Offline Control Utility for the PHOS module.
- Writing the TRU Register Scanner.

4.1 *Motivation*

The DCS card that resides on the RCU has an Ethernet connection, which is our "way in" when we wish to communicate with these cards[1]. All the DCS cards has a static IP address and DHCP (Dynamic Host Configuration Protocol) functionality. If the card fails to acquire an IP address dynamically (which will happen when it is not connected to the CERN network) the static IP address will be used (see A.4.1 for more information). Would it be possible to control the module if it is not even connected to the CERN network? Yes, my Offline Control Utility should provide the necessary features for this.

The need to test the TRUs arise quite often either because some new boards arrives or because we wish to be reassured that the old ones are still healthy (which are certainly not always the case). My TRU Register Scanner is aimed to thoroughly test the ALTRO communication (communication over the GTL bus) by writing and reading TRU registers and write relevant test results to a log file. This was the first step in a program I planned to develop to fully test all functions of a TRU, thus providing a fully automated way to do this.

Lets start with the basics shall we?

[1] With a certain RCU firmware (from a project called TPC) it is also possible to communicate with the RCU via the optical-fibre cable which is connecting the RCU with the computers that will receive and process the data from it. However, at the point of writing it is not working with PHOS.

4.2 *Compiling the Source Code*

Both of the already mentioned programs are written in `C++` and compiled with a special crosscompiler called `arm-uclibc-gcc`[2] [intDCSCOMP][3]. Follow the instructions on that page to install the compiler, and compile the source code of my programs with the following command:

```
arm-uclibc-g++ -W <file containing source code> -o <specified output file>
```

If you plan to make any changes to my code be aware that this compiler has some limitations as to what features it support[4]. Do not trust the code to work with this compiler just because it worked with some other `C++` compiler (such as `g++` as I used for reference).

Keep it simple! Stick with the `C++` standard `std` library, as the DCS cards are already short on space and the amount of libraries stored on them should be kept at a minimum.

4.3 *Preparing the DCS Cards*

I have been told that the DCS cards have around 8MB of local memory. However, with NFS (Network FileSystem) you can mount folders on your computer into the filestructure of the DCS card (folders are shared between the two, but the actual contents are only located on the computer, see A.4.3 for instructions on how to set up a NFS connection). You can do this as soon as you are connected to the DCS cards, but you might also wish to have the programs reside on the card such that there will be less hassle getting them up and running[5]. No matter whether you decide to use a NFS mount or to store the programs locally, you should aim to keep the memory usage of the programs as low as possible. When the output binary size ends up of a few 100kB means that you are already pushing the limits.

In order to run my programs on the DCS cards they must be set up with the proper libraries. On all the cards I have ported it to so far, the library files `libstdc++.so.5.0.5` and `libgcc_s.so.0.9.9` were missing - both needed in order for the `C++` program to start. Add them to `/lib` and you should be fine. You might also need to set a symlink[6] to the standard `C++` library (`libstdc++.so.5 -> lystdc++.so.5.0.5`).

[2]Version 3.3.1.

[3]The other compiler mentioned on [intDCSCOMP] is not needed to compile my programs and make them work on the DCS cards.

[4]Be extra aware of this when using getline() and concurrent executions.

[5]I am not sure if it is possible to have NFS mounts when you connect to the DCS cards through a serial line (see A.4.2).

[6]Syntax for setting symlinks in Linux: `ln -s -T <target> <symlink name>`

4.4 PHOS Offline Control Utility

Please look up [srcPHOSCTRL] for example files and sourcecode.

4.4.1 Motivation

When PHOS is powered on and the RCUs are connected to the CERN intranet, there is higher level software available for controlling the module. But what about those situations where you might not be able to connect to the CERN intranet? How can you control your cards "manually", without all those fancy buttons?

Well first of all, you will need to connect to the DCS cards either via Ethernet or via the serial port. For information on how to do this, refer to appendix A.4. When you are logged onto the DCS cards - and it sits on a RCU - you can execute commands with the syntax `rcu-sh <command>`[7]. However, controlling the module like this quickly becomes very timeconsuming as you need to write several commands for doing simple things[8], and it is almost impossible to remember the syntax for all the different commands.

For this reason, we normally put commands into BASH scripts. These scripts are part of most Linux systems and acts as textfiles which can be executed[9]. The BASH scripts are read line by line, so it is perfect for listing commands which are to be run consequently. However, while you can do almost everything with these scripts I can not say I am an expert at writing them, nor was any of the people I was working with.

So I sent some e-mails around to find out whether someone had written a `C` or `C++` compiler for the DCS cards. This seemed indeed to be the case (as you can see in 4.3), so I quickly decided to write this control application in `C++`. You can find it the the `/dcs/controlProgram` folder on each of the DCS cards which is currently[10] in use for the first PHOS module that went down into the ALICE cavern.

[7]Refer to [docPHOSUM], [docRCUFM] or [docALTROUM] for more information.

[8]Take a look at C - first two sections - to see the amount of commands you normally execute just to read and write from the TRU.

[9]Similar to the `.bat` files from Microsoft.

[10]As of 5.July 2008

4.4.2 ***Features And Usage***

Summary *4.4.2.1*

I will start with a summary of the features the program provides, which will be followed with a more in-depth explanation.

- Support both the ALTRO protocol and the “Slow Control” protocol.
- All user customisable settings are read from file.
- Automatic detection of online and/or offline TRUs/FEEs.
- Power on/off FEEs and/or TRUs with a “soft” step-by-step approach.
- Sets the FEE cards up with initial trigger and ALTRO configuration.
- Applies high voltage bias’ to the Avalanche Photo Diodes.
- Some other small features which basically simplifies readout of some important RCU registers.

PreCompilation Options *4.4.2.2*

When the PHOS Offline Control Utility is compiled, you may choose whether it is to be run in *ALTRO* (ALICE (TPC) Readout) mode or *Slow Control* mode. The ALTRO protocol utilises the GTL data bus and is a custom made communication protocol for the PHOS and TPC subdetectors. It has a wide variety of control signals and flags, and is used when data has to be read out via the RCU. The “Slow Control” protocol is - like the name implies - a slow serial protocol, but as it is meant to be used only for control signals the communication speed is less important. The mode is default set to use the “Slow Control” protocol, as we had best results with this one. If you wish to use the ALTRO protocol, you can change the `COM_MODE` variable in the source code and recompile. See [docALTROUM] for more information about both protocols.

In the sourcecode you also have the option of enabling the use of a log file (`log.txt`), which logs all the commands the program executes. However, since the size of this file quickly increases into several hundred kB, I left it default off.

Finally, you can also set a flag in the sourcecode which disables the use of “automatic detection” of active cards, and rather manually specify it in the file `currentACL.txt`.

Settings File *4.4.2.3*

The program receives all of its parameters from a textfile called `HARD_TRIG_SETTINGS.txt`, which must be located in the same folder as the program itself. It will be read into the program when you execute it, which is why you need set it up prior to executing the program. The format of the file is as follows (tab. 4.1):

Format:

```
0000 0000 0000 0001 0000 0000 0000 0000                       (1)
0000 0000 0000 0001 0000 0000 0000 0000                       (2)
2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1   (3)
2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1 2b1   (4)
000                                                           (5)
044                                                           (6)
f                                                             (7)
```

Comments:

```
(1)  Active Cards List (From Left to Right: Branch B FEC14 -> Branch A FEC0)
(2)  Altro Enable List (From Left to Right: Branch B FEC14 -> Branch A FEC0)
(3)  HV Branch A (First is "global" - then FEC14 down to FEC1)
(4)  HV Branch B (First is "global" - then FEC14 down to FEC1)
(5)  ACQ Start (10 bit)            - BoardController register 0xA[19:10]
(6)  ACQ End   (10 bit)            - BoardController register 0xA[9:0]
(7)  Number of pretrigger samples  - BoardController register 0x0C[3:0]
```

Tab. 4.1 - PHOS Control Utility - Settings File

When Executed *4.4.2.4*

When the program is launched it will try to detect which TRU/FEE cards are powered on/off. It makes the RCU try to read all cards on both branches, and determines the current "status" based on an interpretation of the outcome of the read operation. This is important for the program to "know", since only one card card should be turned on/off at the time ("soft" power on/off). Without this information the program would have no way of knowing where to start powering on or off in order to achieve the desired map of active/inactive cards (as specified in the settings file).

The program then heads on to read the settings file, and finally outputs whatever it has read in from this file plus the current active cards list - which the program determined in the first step. This way the user can visually check whether the program initialised correctly. Does the "active cards list" actually show the correct map of active cards? Was the settingsfile read in correctly? Both might be worth having a look at, as the determination of the active cards list (ACL) is based on a rather simple method and the read of the settings file is deemed to fail if the file is not 100% correctly formatted.

The Menu 4.4.2.5

```
What do you want to do?
<0>  -> Quit                        <60>  -> Read 0x6000 (LastReadValue)
<1>  -> Run "INIT"                  <78>  -> Read 0x7800 (ErrorStatus)
<2>  -> Load status and settings    <80>  -> Read 0x8000 (ActiveCardsList)
<3>  -> Run "HARD TRIG"             <81>  -> Read 0x8001 (SoftErrorList)
<4>  -> Set high voltages
<11> -> Write to a register         <22>  -> Read from a register

Choice: _
```

This menu lists the operations available to the user when the PHOS Offline Control Utility has booted up. The interesting ones are the single digit commands, and they should be executed in the order they are listed. They perform the following operations:

- `<1>` - *Initialise RCU.* This command will reset the RCU a few times, them make the DCS card "master" of the RCU (which is the default), before it also launches a reset of the FEEs/TRUs.
- `<2>` - *Restart.* If you commit a change to the settings file or the power status on some of the cards in the branch changes, you can hit this command instead of restarting the program. The effect will be the same.
- `<3>` - *Initialise FEE/TRU* . This command first turn on or off the FEEs and TRUs according to the active cards list (ACL) listed in the settings file. Then it resets the cards that were powered on, and sets up the ALTRO Enable List (line 2 in the settings file) and some initial trigger configuration - such as acquisition start and stop, and number of "pretrigger samples" (line 5-7 in the settings file - see [docPHOSUM] for more information about the ALTRO chips). Finally it enables the "mezzanine" on the RCU, through which it can receive external triggers.
- `<4>` - *Set APD bias voltages.* In order for the FEE cards to be completely initialised, the Avalanche Photo Diodes have to be set up with the correct bias setting. This is to "finetune" the voltages across these diodes such that the gain equals out and becomes the same. This the values are written to a set of registers on the FEE cards, which will in turn adjust the voltages accordingly (the voltages are controlled by the FEE cards).
- `<11>` and `<22>` - *Read and Write to BoardController Registers.* When you need to read or write to one of the many registers in the BoardController of either the FEE cards or the TRUs [11], you may use this command. Note that these commands are not very robust. When asked for `FEC #` enter a *single* hex value, when asked for branch enter *either* `'A'` or `'B'`, when asked for register number enter *two* hex digits, and - if you are writing to a register - when asked for data enter *four* hex digits.

[11]The commands for reading and writing registers always uses the "Slow Control" protocol (ALTRO mode is not implemented), which means it will not work well with the TRU before this protocol is properly verified in the TRU BoardController.

4.4.3 Conclusion

This was my first C++ program written for the Linux environment which is running on the DCS cards. Due to a shortage of time when it was made, it is rather poorly commented and most of the features it provides are “hardcoded” into the sourcecode, which means the only way to change them would be to edit the sourcecode and recompile. Optimally, all settings should be user configurable.

When started to write the program I used the ALTRO protocol for everything, but for some reason this solution was far from reliable[12]. So I switched to use the “Slow Control” protocol instead, even though the TRU did not support it at the time. That being said, there is no real need to “initialise” the TRU as we do with the FEE cards (at least not yet), so the only real demand at the moment is that it must be able to control the FEE cards. Which it also does, but with a limited amount of customisability.

Although the TRUs does not need to be initialised, it would surely be nice if you could read and write to the various registers it has. This is not possible with this program as the read and write functions utilises the “Slow Control” protocol. However, even though my program can not provide this functionality (at least, not right now) you can easily read and write to the TRU using the BASH scripts I suggest in C.1 and C.2.

When using the “Slow Control” protocol, the determination of the active cards will always fail to detect the correct status of the TRU (or rather, the program will always assume that the TRU is off). This is no disaster, however, as you can manually make sure they are off by specifying this in RCU register `0x8000` (Active Card List). This problem will automatically be resolved as soon as the “Slow Control” protocol on the TRU is fixed. However, you could also make the program more robust by making it read the Active Cards List register on the RCU and correlate it with the “try and error” detection method as is used now.

Before a program is handed to a third-party user, the designer should make sure that it is as robust as possible. This utility has a long way to go here. The requirement was to get it working, and when it was working it was operated by myself. This way robustness never really became an issue, and further development of the code was stopped as other issues received higher priority. I would like to conclude with a plot acquired from the computer controlling the powersupplies in PHOS, when my program powered up the cards (fig. 4.1):

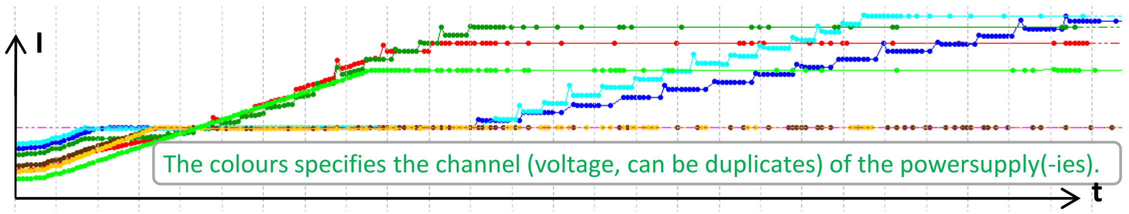


Fig. 4.1 - Current Supplied by the PHOS PowerSupply during Card-by-Card PowerUp [sofGMP] [sofPPT]

[12]More specifically, the only problem with the ALTRO protocol which seemed hard to work around was that the RCU error-status register `0x7800` returned unforeseen values.

4.5 TRU Register Scanner

Please look up [srcTRUREGSCAN] for example files and sourcecode.

4.5.1 Motivation

As mentioned in the introduction of this chapter, the need to test the TRUs arise every now and then. No criterias were set in advance as for how these tests should be carried out, but some sort of test routine should be in defined in order to ensure proper validation. For instance could such a routine require you to:

- Inspect the TRU visually in order to spot any obvious damages.
- Verify that the voltage regulators operate properly by measuring the voltages at each testpoint.
- Flash the TRU FPGA with a configuration which enables us to read out the RMS (root mean square) noise measurement of each ADC channel. The RMS values may be read from from a set of dedicated registers, or via the FakeALTRO protocol if some flag were added which toggled between readout of "raw-data" and the RMS data. The RMS values should prove to be a pretty good indicator of the ADC healthiness.
- Run a testprogram on the RCU which performs the following tests:
  (1) Read and write various TRU registers for a few minutes. If this works then you know that the GTL drivers and the FPGA are operational.
  (2) Read the RMS noise from the ADC channels while the FEE cards are inactive, and verify that the noise has a reasonable value. Now you know that the ADCs work.
  (3) Read out temperature, voltage and current values from the respective registers. Verify that they seem OK.

At the moment[13] neither the proper configuration file for the FPGA nor the comprehensive testprogram exists. However, I considered it wise to have these potentials in mind when I made my TRU Register Scanner. My arguments for making it were as follows:

- The functions could be used to one day make a complete testprogram.
- I was requested to provide some sort of statistics of the performance of read and write operations of the TRU registers.
- It could be used as a means to verify which registers in the TRU BoardController were healthy and which were not. Which were read-only, inactive or not connected? I wanted to know this as I was also asked to provide a complete and final list of TRU registers with some explanations on how to use them (see C.3 for this list).
- It would not take long to make it as several elements of the code would be reused from the PHOS Offline Control Utility (see 4.4), which I had previously developed.

[13]As of 5.July 2008

4.5.2 *Features And Usage*

When I wrote the TRU Register Scanner I made some improvements in the routines from the PHOS Offline Control Utility. Since one of the keyelements in this program is to read and write TRU registers, I tried to come up with a design which not only performed the read and write, but also had verification routines and kept some statistics about the operation. For instance, straight after a write operation a read operation should carried out in order to verify that the data was written correctly. This information should then be crosschecked against the error status flags, to make sure these indicators never deviates. Upon write failure (the rate of which seems exponential to the level of activity on the GTL bus), the program retries for a set number of times whilst recording statistical data. If a write operation fails for a total of 5 times straight, the register is considered dead and flagged as such in the results file (`results.txt`).

PreCompilation Options 4.5.2.1

While you may edit any part of the code you like, I would like to draw the attention to the `WRITE_LOG` constant, which defines whether a complete list of `rcu-sh ...` executions performed by this program are to be written to a log file (`log.txt`). While it is nice to have this feature enabled for debugging purposes, it should be disabled if the program are to be stored locally on the DCS cards as the filesize quickly exceeds 500kB.

Settings File 4.5.2.2

For this program the settings file contains a long list of registers and instructions for the program of how they should be treated. The format is as follows:

Format:
```
<branch> <FEC#> <register#> <value to write> <'r' if read-only, else 'w'>
<a line which describes the register, will be outputted to the results file>
```

Example:
```
a 0 001 00321 w
0x01      [T1_TH]     R/W  Maximum Temperature
a 0 002 00321 w
0x02      [A4V0_TH]   R/W  Min. 4V Ana. Voltage
a 0 003 00321 w
0x03      [A4V0C_TH]  R/W  Max. 4V Ana. Voltage
```

Tab. 4.2 - TRU Register Scanner - Settings File

Make sure you get the number of digits right for each value, as the program will fail otherwise.

When Executed *4.5.2.3*

When we are only dealing with the TRU there is no need to initialise the FEE cards, thus not many initialisation commands are required. In fact, we will only need to setup the RCU. This usually involves invoking a soft-reset a few times, specifying that the DCS card are to be master of the RCU and that that the backplane type is PHOS (and not TPC)[14]. The program will run these commands when executed.

Results *4.5.2.4*

As previously mentioned the statistical data from this program are stored in the `results.txt` file. After a successful execution the top part of this file may look as follows (tab. 4.3):

```
RCU and FECs are reset.  Test commencing:

Error Status:  0

Test 1:  Registers:
0x01     [T1_TH]     R/W    Maximum Temperature       [  OK  ] (Attempts: 1) (Def.: 0xa0)
a 0 002 00321 w
0x02     [A4V0_TH]   R/W    Min. 4V Ana. Voltage      [  OK  ] (Attempts: 1) (Def.: 0x1d8)
a 0 003 00321 w
0x03     [A4V0C_TH]  R/W    Max. 4V Ana. Voltage      [  OK  ] (Attempts: 1) (Def.: 0xc)
```

Tab. 4.3 - TRU Register Scanner - Results File

The output will vary depending on the settings specified in the `settings.txt` file. The first line simply informs the user that the RCU and the front-end cards are reset and should be ready to go. At this stage, should the error status on the second line be anything but zero there is something seriously wrong with the setup and the rest of the resultsfile can be disregarded as rubbish. You might have to powercycle the RCU and DCS card to get the setup up and running again[15].

On the contrary, should no error occur we may have a closer look at the read and write statistics. The left half of this section of the results file are a duplicate of the settings file, and the right half contains the test summary. The word in the square brackets may be either `OK` or `ERROR`. If the register tested was read-only, the error status will be triggered if the RCU error status register at any given time during the read operation were unequal to zero. For registers which can be written to, the error status will be triggered if the specified values are not written and read back correctly within 5 attempts. For these registers the number of attempts are also written to the results file. Finally, the rightmost entry in the results file shows the default value of the register, which are read into the program when the given register is targeted for a write operation. When the test is finished, the default values are written back to the respective registers.

[14]The TPC project utilises the same RCUs, but the front-end card addressing scheme is different.
[15]This are still some bugs in the RCU/DCS software, so power-cycles are quite common.

4.5.3 Conclusion

As mentioned in the introduction this program may play a role in a bigger test scheme for the TRU. It performs read and write operations, and will keep track of some statistics for you. It should be fairly easy to expand this program to include more specific test algorithms, such as a more in-depth evaluation of the values held by the various registers (for instance, statistics about temperatures, currents and voltages).

There are several ways to get the RMS noise measurement from each ADC channel. Instead of implementing the calculation in hardware, you could use the Fake ALTRO protocol (when it is implemented) to send a few samples of raw data from the TRU to the RCU, which could then be picked up by the test program running on the DCS card. The RMS will now be pretty easy to calculate, and the values could be printed to the result file as “noise plots” (like fig. 2.7). This way, not only could you verify that the TRUs actually works, but you can use the data for more sophisticated diagnostics (for instance could you compare noise plots from the FEE cards and the TRU to see where the noise is introduced on the TRU).

I guess it all comes down to how thorough you want to be when asked to test the TRUs. A routine like this can be configured to perform all sort of tests, and once its made it can be easily deployed and executed on several cards with a minimum of extra effort required. The program can also run on the PHOS module which is now [16] installed in the ALICE detector, and be used as a “gatherer” of useful information which could be fetched from the result file.

As for the build quality of the program, there might still be bugs present and the code is rather poorly commented. Time is always a critical factor, and as soon as something “works” the priority is shifted onto another project before the finishing touch can be put on the previous project. That being said, this project was first and foremost created by myself to verify certain aspects of the old TRU design. Even though it will probably never be continued it gave me the answers I was searching for, thus it served its initial purpose.

[16] As of 5.July 2008.

5 Conclusion

What is the future of particle physics? Scientists all around the world are scratching their heads trying to come up with good predictions about the future, but predictions will never make it past that stage unless some new hard facts are put on the table. The Large Hadron Collider (LHC, see 1.4) is designed to boost particles up to energies never before probed by human kind, and the various detectors are designed to provide the ultra sensitive sensory equipment necessary to see what is actually going on. Together they form a giant microscope which is able to create and detect particles down to a scale of a nanometer of a nanometer (10^{-18}m)!

When I came to CERN I had the feeling of entering a completely foreign “environment”. I use the word environment as many people consider CERN *the* place to be when the inner mysteries of our world is put on the agenda. “Only on CERN...” is a phrase I have come to hear often, and it applies to quite a few areas. I would go as far as to say that CERN does something to your state of mind, a state which you can never revert from. The “impossible” is merely a challenge for the future.

In ALICE we are trying to find out *why* matter has the properies it has. The idea is that we heat it up to extreme temperatures such that it decomposes into more fundamental particles, and study what happens when the temperature decreases again and the matter reverts to its old state. There ALICE detector is made up of several subdetectors, each of them specially designed to detect a certain range of particles and/or the properies these may have. PHOS- a PHOton Spectrometer - is a subdetector aimed to detect and measure the exact energy of photons produced when certain particles hit the lead-tungsten crystals that make up the inner “wall” of this detector.

ALICE has a triggerscheme which is supposed to act as a filter that only let through data with a certain set of interesting properties. The triggers come in 3 levels, where Level0 is the simplest and fastest one. In the case of PHOS the Level0 trigger is generated if the energy of the photons produced in the mentioned crystals exceeds a certain user-defined threshold. The pulses of photons are approximately 100ns wide, and may disperse into several adjacent crystals. To sum up all the energy of this light pulse the energy must be integrated both in time (equals the discrete sum of 4 samples with a sampling speed of 40MHz), and in "space" (equals the discrete sum of 4x4 crystals centered over the particle entry point). Both of these calculations are made by the Trigger Region Unit (TRU).

The single greatest task I had at hand while working at CERN was to do digital design for the TRU, the main emphasis was put on improving the logic in charge of generating the PHOS Level0 trigger. The term digital design refers to the process of using Hardware Description Languages (HDL) to *define* the logical layout of a configurable logic device. In the case of the TRU this device was a Field Programmable Gate Array (FPGA). To design hardware in this way requires knowledge about the HDL you are using (in my case Verilog), digital logic in general and more specifially about the resources available in the FPGA you are using. For high speed designs more in-depth knowledge about the logic placement processes and timing constraints has to be thoroughly understood, but even though I have spoken about this in general terms throughout this report I have never gone into great detail.

The digital design field is huge and there is no way I could discuss all the various parameters I have probed into in a report like this. Instead of going into great detail of the various parts of the design, I have discussed the concepts and spent the remaining part of the report to speak about other aspects of my work. The intention with this report was never to be dead-on specific, but rather to give a general impression of the various activites I was involved in, hence the name *activity report*.

But what activities are we really talking about? The chapters of this report deals with the mentioned digital design concepts with regards to the TRU FPGA mainly, but also some design aspects with the TOR card are described. I also spent one chapter describing some of my experiences with C++ programming for the RCU DCS cards. The keyword for all these chapters was *design*, a decision made in order to have a somewhat clear line of topics in the main body of the report.

In the appendixes a variety of topics are described. I experienced great gratification as I started working with Linux, so I start the "other activities" appendix off by argumenting for how this operating system made me more efficient og productive during development. Since it was not obviuous how to get the digital design software up and running in Linux, I provide some quick hints about this in addition to general experiences with the programs in the "software" section. I became a fan of using BASH scripts for automating certain repetetive tasks in my work, for instance for making backups and jumping between them. This will be mentioned next in the appendix.

In my work at CERN I experienced embedded designs[1] for the first time in the form of the DCS cards. They run a small Linux core, and most of the storage space are mounted into the filesystem from external devices through ethernet network protocols. There are compilers available for compiling programs for these Linux cores (I used one of these to compile my C++ programs), so these embedded design naturally becomes the "interface layer" between the electronics and the higher level software (where software engineers are in charge). What I learned about logging onto the DCS cards are discussed in the "DCS Interfacing" section in the "other activities" appendix.

The remaining part of the "other activities" appendix provides some information about the RJ45 standard (a must to know about this one when you are making ethernet cables, as I found myself doing every now and then), a short tutorial of how to adjust the phase of the ADC clocks with the respective TRU registers, and finally a figure illustrating the CSP to FastOR mapping through a FEE card will be shown. They should be considered reference material if the reader happens to be interested in this information.

The next appendix lists some "chronograms", or waveforms, which shows the response of some of my FPGA designs when they are simulated in a dedicated testbench. I spent many hours studying such waveforms, so if you are curious have a look (though you might not get much out of them unless you have already studied the design code). The "TRU Communication" appendix presents some self-made BASH scripts for reading and writing TRU registers (a full list of registers comes shortly after).

Every now and then a period came along where I was mostly busy with modifying hardware and making testsetups for testing my designs or debugging various electronics, thus improving my handyness with the soldering iron and various test equipment (especially useful was the experience gained through the use of a logical state analyser on the GTL bus). Some of this work - like assembly and modifications of new TRU and FEE cards that we received - required components which were initially missing, so I learned how to order them through the CERN *EDH* system[2].

Through working in a big collaboration like CERN I have become much better organised. I tried to blog day to day activities in my journal at `http://joinge.livejournal.com`, and when working with designs I have learned the importance of using version numbers and writing changelogs. The latter is especially important to keep your head straight during late hours. Unfortunately I did not get properly started with the logging before February 2008, but rather late than never right?

[1]The design is referred to as *embedded* if the FPGA processor core and memory interfaces are utilised, for instance for running a Linux core.

[2]If you want to learn more about the procedures involved in component ordering through EDH refer to the excellent explaination by David Koenigseder in his Activity Report, [docDAVID, page 85-87]

To experience a quantum leap in your english skills is inevitable when you work in an international environment like at CERN. At work or off work, the language was always english. You get to know people from all around the world, and learn to adapt to the various cultures present in your working environment. I found it very interesting to see how the various cultures put their distinctive touch to the environment, and what conflicts often arose across cultures and professions.

I have also learned a lot about document typing. This report was my first "real" document in english, and it has undergone a lot of revisions since its early beginning. I wrote it in a typesetting language called LaTeX, which is widely used amoung international students for the creation of professional documents. As you can see in the source files for this document, I have customised the packages quite a bit to get things exactly how I wanted them. I spent a great deal of time doing this as I wanted to create an enviroment for myself in the future for writing professional documents, with ease, and without having to redo the typesetting for every document.

The bottom line is: My stay here at CERN has been a very enjoyable and selfevolving affair. The whole experience stand out as very positive event which enlightened me in many ways. My work may not have have been groundbreaking, but when I look back at where I was when I arrived at CERN I can certainly say that I have come a long way since.

I would like to thank my supervisor, Hans Muller, for trusting me with challenging tasks even though I was not always able to solve them quite as well as I wished for.

I would also like to thank the reader for keeping with me this far. Hopefully the report was digestable, even though it might have been somewhat lengthy (but hey, how can you summarise one year abroad otherwise?), but if anything was unclear and you are curious about what I meant please feel free to contact me!

A
Other Activites

This appendix contains - as the title indicates - some information about various other activies I carried out and documented, but was of such a nature that I chose not to put them in the main body of this report. The following topics will be covered:

- *Linux as Operating System (OS).* Why I chose to do almost all my work in this operating system.
- *Software for Digital Design of Xilinx FPGAs .* Most of this software I was able to get working in Linux, a small how-to get them up and running and some usage tips will be provided. The topic of installation instructions gain relevance as there are almost no documentation available on the CERN software server (`dfs`).
- *Creating backups.* Now why would I talk about this? Basically because I used BASH scripts to do this instead of the built-in "snapshot" option in ISE, and as I believe my scripts represents a better option I will supply them here.
- *Interfacing the DCS Cards.* These cards hosts a small Linux system which I frequently had to log into in my work. I will explain how to connect to these cards and try to show how they fit into the overall PHOS picture.
- *The RJ45 Standard.* I got to know this one well when I was making Ethernet cables. Might be worth the read for the sake of reference.
- *Adjusting the TRU ADC Clocks.* You have no idea how to go ahead in order to get the ADC bit- and frameclock adjusted? This section provides a quick tutorial.
- *FEE CSP to FastOR Mapping.* Want to know which CSP pin is connected to each pair FastOR output pins on the FEE? Then this might be worth looking into.

A.1 Why Linux?

Why Linux was my OS of choice when working in PHOS:

- One can interface the PHOS electronics via the DCS cards. These cards have a few MBs of flash memory and an FPGA with a processor core. They host a small Linux system which can be accessed via the onboard Ethernet port or serial port. There is a DCS card attached to the TOR and each of the RCUs. You log into them via SSH (Secure Shell) [1], which is bundled with almost all of todays Linux systems.

- It is very easy to send files from one Linux filesystem to another, but not so easy to transfer files between Windows based systems and Linux based systems. In Windows I know of two alternatives: (1) You can set up a NFS server, share a folder and mount it as a part of the DCS filesystem (see 4.3), or (2) you can use the Secure Copy protocol (`scp`[2] which is part of the SSH (secure shell) package in Linux. In Windows you can achieve the same thing by installing e.g. the graphical software *WinSCP*, but it will not blend in as well as with Linux. Furthermore, you must make sure that the newline characters in the textfiles you send are converted[3]. By using Linux you may avoid these small obstacles, and the overall efficiency can be greatly improved if you utilise the powerful features of the BASH shell and scripts (for instance for managing concurrent file transfers.

- In Linux you can use SAMBA (A Service for Print and File Sharing) to mount Windows folders into your Linux filesystem. This provides an easy way of transferring files between your Linux machine and other Windows machines. However, the DCS cards have a very limited Linux OS and does not support SAMBA (so the previous point still applies).

- I find that I work much more efficient in Linux than any other OS I have tried. It better utilises the available hardware resources, almost never crashes and generally just leaves me with the impression that it wishes to help out rather than making life miserable.

[1]Syntax for logging into a system using SSH: `ssh <username>@<card ID>` where the *username* is usually *root* and *card ID* can be *IP-number* or *DNS name* with or without domain postfix.

[2]Syntax for copying files with SSH: `scp <path to file> <user>@<card ID>:<path to put the file>`. Swap the two parameters if you want to copy the file from a card to your own filesystem instead.

[3]Linux and Windows utilise different newline characters.

- Backup and restore becomes a breeze with the use of BASH scripts (see A.3). The Xilinx Integrated Software Environment (ISE) offers “snapshots”, but I prefer using my own scripts because:
  (1) *it enables me to specify exactly what files I want to backup.* When I make a selected assortion of files from the ISE project folder I usually end up with a backup measuring a size of 4MB, the equivalent size of the backup the “snapshot” tool creates is often several hundred MB[4].
  (2) *it becomes easy to distribute the code to external computers, hence these can be used to implement your code and less strain will be put on the system you are working on.* Again, this will only work nicely if you send small amounts of data, in accordance with (1). A handy feature of Linux here is how files can be opened in several programs simultaneously, and where these programs “catch” the changes committed to the file externally. This very nice for instance if I use BASH scripts to revert to and old copy of my code by overwriting the project directory with the backup files. In this situation the text editors immediately refreshes its opened files according to the changes, and without further actions you can start editing this version of your work.

  (3) *making backups and “reverting” to old versions becomes easy.* These two operations are simply two sides of the same story.

A.2 Software

Since I decided to do the Linux route all the software necessary for doing digital design on Xilinx FPGA had to be adapted aswell. Unfortunately not many people at CERN was working with Linux[5] which was probably the reason why I was not able to find any documentation on how to install the Linux editions of the various software on the CERN intranet/websites. Since I know how to do this now, this section will serve as a short “how-to” for people with some Linux experience. Three software packages will be discussed:

- The *Xilinx Integrated Software Environment (ISE) and ChipScope.* ISE has all necessary tools for breaking up the HDL code (like Verilog), producing the binary file used to configure the FPGA and for actually programming the FPGA.

- The simulation tool *Modeltech ModelSim.* Used for all sorts of simulations, and is far superior the light simulator in ISE.

- The synthesiser *Synplicity Synplify* (which I believe is superior to the Xilinx XST synthesiser - bundled in ISE - both with respect to features[6] and synthesis quality.

[4]If you choose to “cleanup project files” in ISE prior to making the snapshot the backup size decreases to a total of around 50MB. However, since I tend to make hundreds of backups this is simply not good enough.
[5]As I discovered on the Xilinx seminar at CERN the 28.May 2008 when the presenter asked how many of the designers present were running Linux.
[6]In my humble opinion Synplify has better support for Verilog HDL syntax, more options for finetweaking the synthesis, better represents the results (both with graphics and text) and is easier to use.

All the programs I will dedicate some words to can be found at

```
ISE & ChipScope  "\\<server>\dfs\Services\caeprogs\xilinx"      [locXilinx]
ModelSim         "\\<server>\dfs\Services\caeprogs\modeltech"   [locModeltech]
Synplify         "\\<server>\dfs\Services\caeprogs\synplicity"  [locSynplicity]
```

The servers you can choose from are `cern.ch`, `cernhome01` or `cerndfs01`. You should be connected to the CERN intranet via cable for full access (and much greater speed) when getting the programs from the specified locations.

A.2.1 Xilinx ISE

The Integrated Software Environment from Xilinx pretty much provides all necessary tools for FPGA development to the table. It features a texteditor, the XST synthesiser, implementation tools (like translate, map and place&route) and a binary configuration file generator. After installation, the following variables needs to be set:

```
PLATFORM="lin"
XILINX="<installation path>"
LMC_HOME="${XILINX}/smartmodel/lin/installed_lin"
PATH="${XILINX}/bin/lin"
LD_LIBRARY_PATH="${XILINX}/bin/lin:/usr/X11R6/lib"
NPX_PLUGIN_PATH="${XILINX}/java/lin/jre/plugin/i386/ns4"
qtDir="<your home directory>/.qt"
myxilinxrc="${qtDir}/xilinxrc"
```

You can also run the *settings.sh* script in your ISE installation directory, but I did not succeed in getting satisfying results this way. The licence for both the Windows and Linux version of ISE can be found in `<locXilinx>/CERN_readme.txt`.

On the next page I will say a few words about one of the programs in the ISE package called *iMPACT*, which is a standalone program we always used to program the configuration memory of our electronics cards.

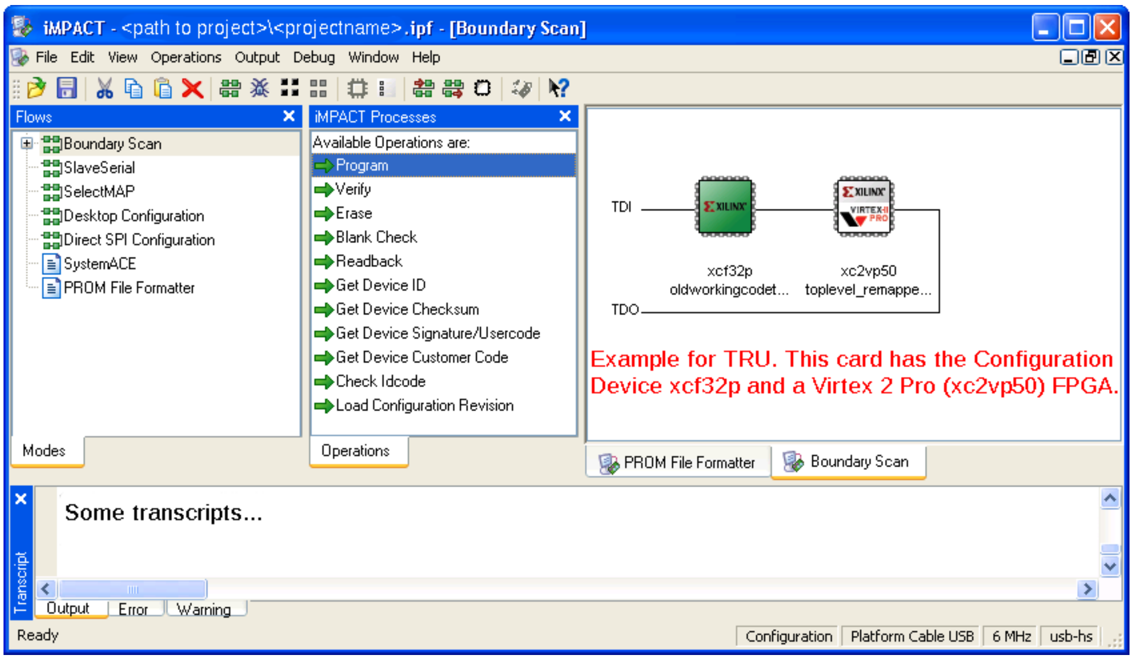
A.2.2 ***iMPACT***

Impact is the software you have to use in order to program the Xilinx FPGAs[7]. ISE should also be able to do this, but as Impact is made for the single purpose of programming FPGAs and configuration devices it became my preferred choice. Since programming the configuration devices is a task I encountered quite often - and no good tutorial was ever made on how to do it - I will try to give a simple and direct explanation on how to do it here [8].

- **iMPACT Project** is the title of the pop-up window you should get when launching iMPACT. Select `create a new project (.ipf)` and hit `OK`.
- **iMPACT - Welcome to iMPACT** is the title of the next window you will get. Select `Prepare a PROM File` and click `Next`.
- **iMPACT - Prepare PROM Files**. Check that *I want to target a...* is set to `Xilinx PROM`, *PROM File Format...* is set to `MCS`, and *Checksum Fill Value (2 Hex Digits)...* is set to `FF`. Finally you need to fill the *PROM File Name...* field, and set the location to where you want the `MCS` file stored. When done, hit `Next`.
- **iMPACT - Specify Xilinx PROM Device**. Make sure the *Select a PROM (bits)...* leftmost rolldown box says `xcf`, and the rightmost rolldown box contains the name of the configuration device you want to program[9]. When these boxes are set hit `Add`. Make sure all the checkboxes on this page are unchecked, and click `Next`.
- ***iMPACT - File Generation Summary***. Verify the settings and click `Finish`.

A small pop-up box informs you that you should now adding files to the project. Click `OK`, then select the `.bit` file you want to program the configuration device with. Answer `NO` to the next box asking you whether you want to add more devices or not. Next doubleclick the “flow” named `Boundary Scan` in the topleft “Flows” window. The “main” window should now say *Right click to Add Device or Initialise JTAG chain.* Do the latter and select the proper `.bit` file and `.mcs` file when confronted with this. In the same window you should now be able to see two icons, respectively representing the configuration device and the FPGA (see fig. A.1).

[7]Impact comes bundled with ISE
[8]Approach may vary with software version. I used iMPACT version 9.2.04i.
[9]The configuration device on the TRU are named `xcf32p`



Example for TRU. This card has the Configuration Device xcf32p and a Virtex 2 Pro (xc2vp50) FPGA.

Some transcripts...

Fig. A.1 - Screenshot of iMPACT

To start the programming procedure hit arrow tagged `Program` in the "iMPACT Processes" window. In the window that appears check the box reading `Erase Before Programming` (under *General CPLD And PROM Properties*); likewise with the box labelled `Load FPGA` (under *PROM Specific Properties*). All the other checkboxes I tend to leave unchecked. When done hit the `OK` button and the programming should commence.

When done you might want to save your project in order to just pick up the trace where you left it the next time you are in need of flashing some configuration devices. This basically wraps up my knowledge of this program. I simply found an approach "that works", and kept doing it that way. Feel free to explore.

A.2.3 ChipScope

The ChipScope Pro is a JAVA based application which allows you to analyse FPGA logic, various buses and even provides a function for making virtual inputs/outputs. While the ChipScope Pro core are optimised for size and performance and are designed to operate at speeds up to 200 MHz in Virtex-II Pro FPGAs, caution should be taken when speeds exceed 150 MHz or logic usage exceed 80%. The ChipScope core uses some logic and a fair bit of *Distributed SelectRAM* (see 2.1.2.4). However, in our TRU/TOR designs the ChipScope core will never driven at these speeds. This means we should be fine as long as the ChipScope cores are not made too big.

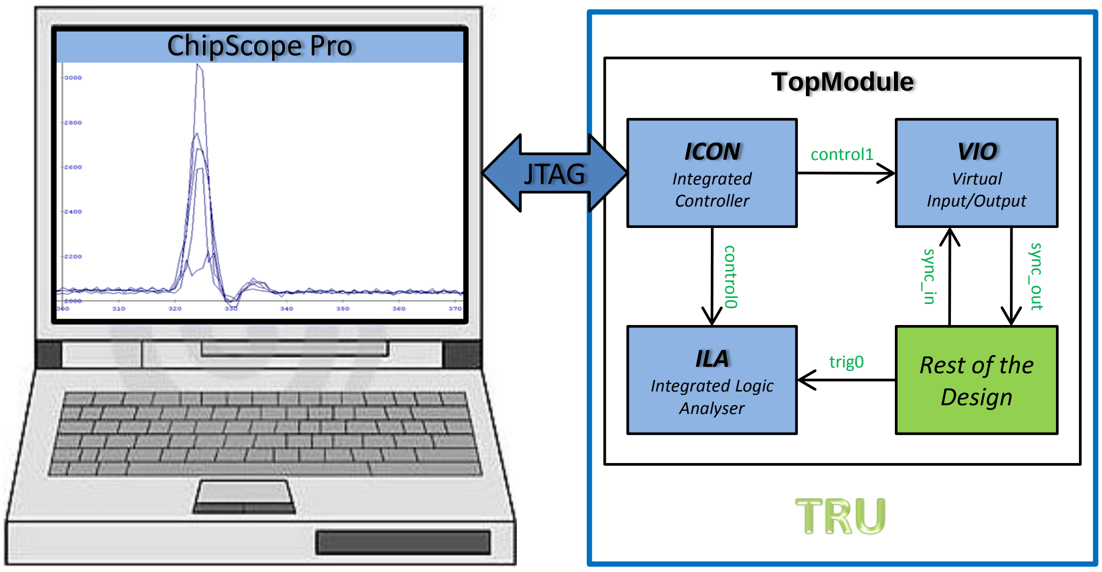


Fig. A.2 - ChipScope Principle [sofPPT]

For a complete tutorial on ChipScope I suggest looking into this one [docXCSTT].

ChipScope can be used to program FPGAs, but can not program the configuration devices. This means that the FPGA will keep its configuration until it is powercycled, when it will be reprogrammed with the data in the configuration memory.

I was never able to get the drivers for the Xilinx USB programmer dongle to work under Linux. While I know that this is doable I did not care to bother with it as PHOS has a laptop dedicated to the task of programming electronics (it runs Windows XP). I used the CERN Desktop for transferring files between my private laptop and the PHOS laptop[10].
[11]

[10]The path for a user specific CERN Desktop is:
`\\cern.ch\dfs\Users\<first letter of username>\<username>\Desktop`

[11]Based on Based on [docXCJ50, page 54],[docXCSTT].

A.2.4 ModelSim

First, if you are installing ModelSim on a private computer you will need to set up a few variables.

```
PATH="<installation path>/modeltech/linux"
MGLS_LICENSE_FILE="1717@licman1.cern.ch:1717@licman2.cern.ch:1717@licman3.cern.ch"
```

It is important to put the location of all the binaries in your PATH to get full interoperability for ISE, ModelSim and Synplify. Especially the compilers start to misbehave if the binaries are not in you PATH variable. The licence is "floating" here at CERN, which means there are CERN servers handing them out. The above licences work for both Linux and Windows versions of ModelSim.

When you have ModelSim up and running, you have to perform some initial procedures:

- Clean Start. If you have simulated your project with ModelSim before, you yourself a favour and head over to your project location and delete the old `work` library (folder), the `modelsim.ini` file, and the `<projectname>.mpf` before doing anything else. I am not sure why, but my ModelSim has a tendency to mess up badly if I skip this step.

- Set Project Directory. `File` ▶ `Change Directory...`.

- Create Working Library. `File` ▶ `New` ▶ `Library...`. Use default library name "work", check that *a new library and a logical mapping to it* is selected and hit `OK`.

- Create the Project. `File` ▶ `New` ▶ `Project...`. Select a proper name for it, check that the project location is set right and make sure "work" is set as the default library. Do *not* copy settings from a previous `modelsim.ini` file, as we want to start blank in order to know things are under control. Make sure that field is blank, and that the option for copying library mappings is checked. When done, hit `OK`.

- Add items to the Project. This is the label of the window appearing after you created the project. You may want to start making some folders to set up a hierarchy. When adding files I suggest you choose to reference to your files instead of copying them (at least I want it to simulate my source files and not some "backup" I made).

When this is all done, you will need to compile the files the "ModelSim way" and add them to a simulation. This is best accomplished by using a .do file. It is more reliable, much quicker and features a wide variety of options for customising the simulation to better suit your preferences.

```
quit -sim                  #Quit any old simulation you may have got running.
vlog +acc -sv TopModule.v  #Compilation of the files you might have.
                           #(-acc = detailed transcript  -sv = SystemVerilog)
vsim work.TopModule        #Start the simulation
radix hex                  #I prefer the default radix to be hex
formatTime +commas +bestunits #I also prefer the use of "scientific units"
add wave -label <name> <ModuleName>=<SumModuleName>=<SignalName>
run 15 us                  #Simulation length
view wave                  #Show the wave window.
```

The compiler will probably complain about it not being able to find the Xilinx primitives you have used throughout your code. However, Xilinx has been so kind as to provide modules which will behave the same way as the primitives when they are run in simulation. Get the ones you need and copy them into your project (say, into a folder labelled primitives).

```
<ISE installation path>/verilog/src/unisims/         #High level modules.
<ISE installation path>/verilog/src/XilinxCoreLib/    #Low level components.
```

Some other documents you should read

```
Get Started Tutorial       <installation path>/modeltech/docs/pdfdocs/modelsim_se_tut.pdf
UserManual                 <installation path>/modeltech/docs/pdfdocs/modelsim_se_user.pdf
Support for SystemVerilog  <installation path>/modeltech/docs/technotes/sysvlog.note
```

A.2.5 ***Synplify***

The Xilinx synthesiser called XST does a fairly good job, but Synplify is - in my opinion - far superior. What you do is open the source code files into Synplify (`.v` , `.ngo` , `.edn` , `.ucf`), run the synthesis (this is the first step of the implementation process) and notice that `.ngc/ngo` files are created as output. There is a specific project type in ISE which has these filetypes as "basis", so you basically create one of these and let ISE do the rest of the implementation after Synplify. After installation of Synplify, the following variables will need to be specified:

```
PATH="<installation path>/fpga_902/bin:<installation path>/identify_25/bin"
SYNPLICITY_LICENSE_FILE="1709@licman3:1709@licman2:1709@licman1"
```

Also these licences seem to work flawlessly on both Linux and Windows operating systems.

When you wish to start using Synplify on a new project there is a small trick I use to get started quicker. First load up ISE with the project you want you synthesise, then set your synthesiser to be Synplify (Pro). Run the synthesis once to get ISE to hand along some parameters to the Synplify project that will be made [12]. Once done (it does not matter whether the synthesis succeeded or failed) close ISE and start up Synplify (Pro). Have it open the Synplify project which is now located in your project directory, and hit Implementation Options -> Verilog. Check the box tagged "SystemVerilog" and accept the changes. Voila, you are good to go!

[12] If you use Synplify version 9.0.1 it will always fail due to a bug that causes incompatibility between ISE and Synplicity

A.3 *Creating Backups*

When you are creating backups you need to know what files are important in your project directory. If you simply copy it all you will end up with backups of several hundred MB, most of it being binaries from the implementation software which is not necessary to backup (as you can simply rerun the implementation to get them back). From starting a project from scratch with a bare minimum of files, I have found that the only "important" files we need to backup are:

```
*.v             Verilog Source Code Files (The designer should know which are important)
toplevel.ucf    The constraints file for the design
TopModule.ise   The ISE Project File
*.ngo \& *.ngc  Presynthesised code, or "black boxes". TopModule.ngc not needed.
```

The rest are just software generated files. If you ever run into problems where the software you use does not seem to "catch changes", then delete all files not mentioned above. This will strip the design down to a absolute scratch, and force the programs to start all over with the source files.

```
#! /bin/bash

cd <project path>/Backups                        #Assumes a "Backups" folder exist
mkdir ${1}                                       #Create the directory to store the data
cp ../*.v ${1}                                   #Copy all the Verilog source files,
cp ../*.ucf ${1}                                 #the .ucf constraints file,
cp ../TRURevised.ise ${1}                        #the ISE project file,
cp ../*.bit ${1}                                 #the .bit file (used to program the FPGAs),
cp ../*.ngo ${1}                                 #all .ngo and
cp ../*.ngc ${1}                                 #.ngc files (presynthesised modules)
rm ${1}/TopModule.ngc                            #except this one which is created by ISE,
cp -r ../TriggerOut  ${1}                        #and finally all the files for TriggerOut,
cp -r ../BoardController ${1}                    #BoardController and
cp -r ../Clk ${1}                                #Clk.
```

A.4 DCS Interfacing

A.4.1 Via Ethernet

The easiest way to connect to a DCS card is if it is connected to the CERN intranet. Then it will be assigned an IP-address from a CERN router (via DHCP), and you can access it by using this IP or by using the name of the card (which will be translated to the IP-address with the CERN DNS (Dynamic Name Server) service). Look at fig. A.3 to see which DCS cards were used in ALICE. In addition to these cards, I also interfaced a DCS card situated on a TOR development board (called `pcdcstor2`), and a DCS card situated on RCU development board (called `pcdcs0002`). If the DCS cards are not connected to the CERN network they will use an IP-address specified in the `/etc/network` file. Often this IP-address can be found with the following method:

```
10.1. (DCS number.>256)?1:0 . (DCS number.>256)?DCS number-256:DCS number
```

For instance, DCS number 279 (called `alphdcs0279` on the network) would have IP-address:

```
10.1.1.(279-256) = 10.1.1.23
```

While other methods for addressing are used in ALICE, the method mentioned here was used for the DCS cards prior their installation in the ALICE detector.

If you know the local IP-address of a DCS card you can connect to directly with your computer, or to via a router. You just have to make sure that you manually set the IP-address of your computer such that it is on the same subnet as the DCS card you want to access. For instance, if you want to access a DCS card with the IP-address 10.1.1.23, you should set your computer up with for instance the IP-address 10.1.1.10. Now you can connect without problems.

When you connect to the DCS card with the IP-cable (SEMTEC connector), make sure that the red Light Emitting Diode (LED) starts to blink. This indicates that the communication is up and running. If nothing happens, try to powercycle the +4.2V power supply. This is a common problem and is caused by a bug in the DCS firmware (the bug is known, but is not yet corrected).

A.4.2 Via Serial Line

You can also connect to the DCS cards over the serial line (RS232 protocol). Fire up any program that can deal with this for you (for instance the *HyperTerminal* application in MS Windows) and connect with the following settings:

```
COM Port ........ 1
Baud Rate ....... 57600
Data Bit ........ 8
Stop Bit ........ none
Parity .......... none
```

In my experience it is best if you try to connect straight after you rebooted the DCS card (either via. typing the command `reboot` or by powercycling the +4.2V power supply).

A.4.3 Set Up NFS Shares

NFS shares are folders which are physically present only on a host, but can be "mounted" into the filesystem of other devices on the network (such as the DCS cards). Using this method with the DCS cards enables them to access lots of storage space even if they only have a few MB of local memory.

To get started the computer need to have a NFS server up and running (see A.1), which should the the out-of-the-box case for most Linux distributions (distros). If not, they are easy to set up. When the server is up, share (or export) the folder you wish to share (should also be pretty self-explanatory). The computer is now ready.

The DCS cards are already ready for mounting NFS folders. The command used to do this is listed below:

```
/usr/local/sbin/nfsmount <computer name>:<path to folder> <path to local folder> rw
```

The *computer name* can be either an IP-address or a DNS address, and the *rw* switch simply indicates that the folder should be both readable and writeable. If you want the DCS card to attempt to mount your folder every time it starts up place the mount command in the file:

```
/usr/local/sbin/nfsmount_all
```

It is surprisingly easy to do this, and it works surprisingly well.

A.4.4 Connecting to DCS in ALICE

Even when the PHOS module is in the ALICE cavern and none of the DCS cards are publicly available on the CERN network, there is a way to remotely communicate with them. You have to be added in the PHS (short for PHOS) security group, and when you are you will be able to login to a controlnode called `alidcscom001.cern.ch` via *MS Windows Remote Desktop Connection*[13]. The mentioned node runs a Windows operating system, so to get from there to the DCS cards (which runs Linux) you need to use a program called *Putty*. Putty provides a SSH interface for use with Windows.

You might also wish to transfer files from and to the DCS cards, which you can use a program called *WinSCP* for. For filetransfer between the `alidcscom001` node and your own computer, use the DFS Desktop:

```
cern.ch ▶ dfs ▶ Users ▶ <first letter of username> ▶ <username> ▶ Desktop
```

When the PHOS module was installed in the ALICE detector the DCS cards were allocated different names than they had on the CERN network. The current[14] names of the various DCS cards can be looked up in fig. A.3, along with a summary of what I have just mentioned.

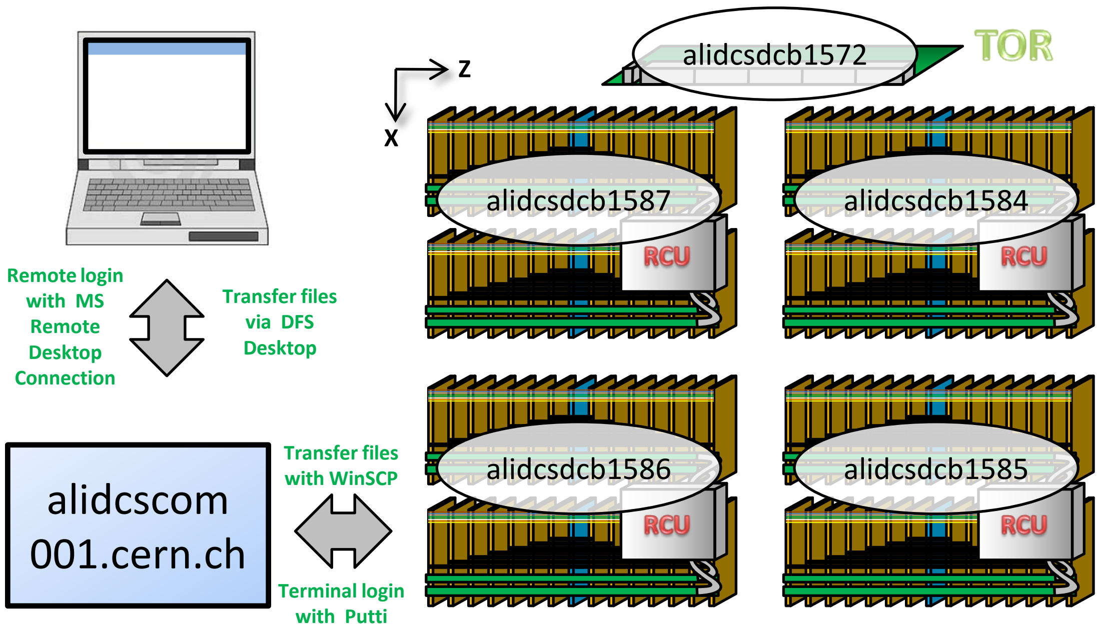
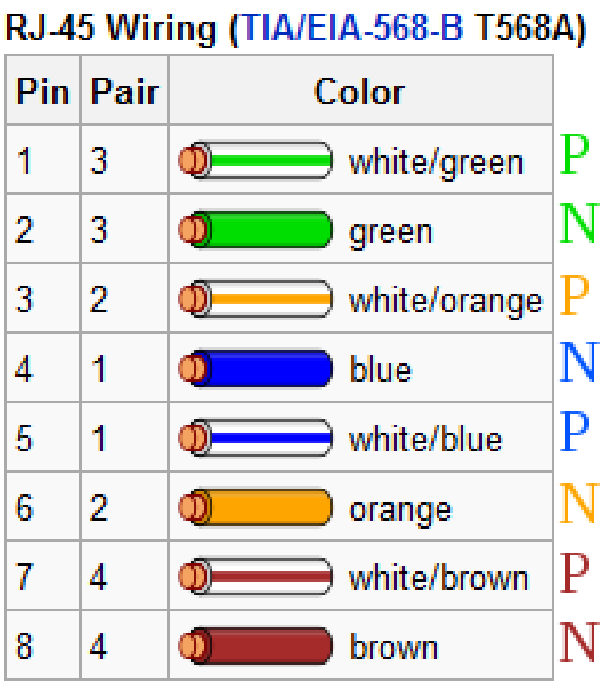

alidcsdcb1572
TOR
Z
X
alidcsdcb1587
alidcsdcb1584
RCU
Remote login with MS Remote Desktop Connection
Transfer files via DFS Desktop
alidcsdcb1586
alidcsdcb1585
alidcscom 001.cern.ch
Transfer files with WinSCP
Terminal login with Putti

Fig. A.3 - Communication Flow when Remote Controlling DCS in ALICE [sofPPT]

[13]MS Windows Remote Desktop Connection should be a part of the bundled software that comes with most Windows installations.
[14]As of 5.July 2008

A.5 *RJ45 Standard*



RJ-45 Wiring (TIA/EIA-568-B T568A)

Pin	Pair	Color	
1	3	white/green	P
2	3	green	N
3	2	white/orange	P
4	1	blue	N
5	1	white/blue	P
6	2	orange	N
7	4	white/brown	P
8	4	brown	N

Fig. A.4 - The RJ45 Standard

The only category 7 CAT7 twisted pair TP cable connected to the TRU are intended for use with TOR communication (described in detail in 2.2.8). This means there are 4 pairs of wires available for signalling between these two units. Fig. A.4 shows the correspondence between pin numbering, LVDS pairs and wire colours for the T568B variant. The most important factor to "get right" is the pair matching. The positive and negative complementary LVDS pulse should be sent through the same twisted pair. This insures better data integrity (less crosstalk from other wires and better impedance matching) than if a random pair of wires were selected. Also, for consistency, the designer should try to make sure that pulses with given polarity propagate through defined type of wire (e.g. positive pulses always propagate through wires with homogeneous colour).

A.6 Adjusting the TRU ADC Clocks

Test procedure for adjusting the bit- and frameclock for synchronising ADC data:

1. `0x0040` ▶ `0x7c` - Sample[3:0] = 4'h0
   `0x0050` ▶ `0x7c` - Sample[7:4] = 4'h0
   `0x0064` ▶ `0x7c` - Sample[11:8]= 4'h4
   `0x000c` ▶ `0x7c` - Launch pattern, 3.5mA

   This sets the ADCs up to output 1 on Sample[10]. Note what channel is showing this bit.

2. `0x0040` ▶ `0x7c` - Sample[3:0] = 4'h0
   `0x0050` ▶ `0x7c` - Sample[7:4] = 4'h0
   `0x0068` ▶ `0x7c` - Sample[11:8]= 4'h8
   `0x000c` ▶ `0x7c` - Launch pattern, 3.5mA

   This sets the ADCs up to output 1 on Sample[11]. Did this 1 appear on the correct channel (the more significant than the previous one)? Yes (4). No (3).

3. `0x0005` ▶ `0x7b` - Add 45° to ADC_LCLK

   See if the bits swap places. Keep adding 45° till they do.

4. `0x0010` ▶ `0x79` - Add 22.5°to ADC_ADCCLK

   See how the MSB move. Keep adding 22.5° till its on Ch11.

You have now successfully tuned the bit- and frameclock to roughly the correct spot. By performing small phaseadjustments while watching the samples generated you may find the "golden eye" to clock each ADC channel.

If the samples are stable over time but the bits are correctly placed the frameclock is badly adjusted. When samples vary over time either clock (or both) may be incorrectly adjusted, but chances are the bitclock is the sinner.

A.7 *FEE CSP Mapping*

The FEE performs analog sums of the signals coming from the CSPs of 2x2 crystals, and sends them to the TRU. It also stores high-gain and low-gain versions in on-board ALTRO chips (for more information: [docALTRORC]), which can be read out by the RCU over the GTL bus. There is extensive documentation on the FEE cards currently available, and if more information is desired I suggest you start with the PHOS User Manual [docPHOSUM].

I did not work a whole lot on the FEE, but performed a CSP-to-FastOR mapping. It was requested of me to provide documentation about this, so here it is.

Basically, there are 32 CSPs (each of which is connected to a single crystal) connected to each FEE [15]. The map was performed by simply pulsing one and one CSP input on the FEE and note which FastOR output reacted to the input stimulus (a 1ms step function). Every other pin on the FEE input connector is `GND`, which the stimulus were referenced to over a 50Ω resistor. The FEE shapers are quite capable of dealing with step functions, producing 100ns FastOR signals as if the inputs were connected to CSPs. The preamp was tested in a similar fashion, the result of it all is summarised in fig. A.5:

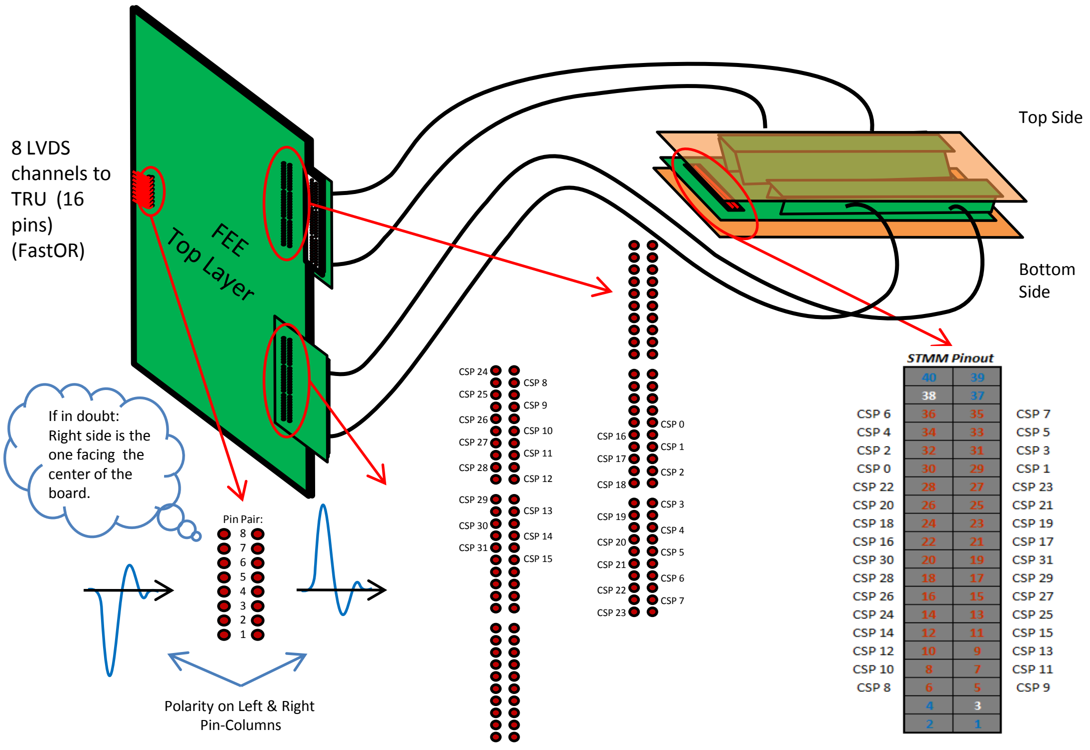


Fig. A.5 - STMM, FEE Input and FEE Output (FastOR) Pinout Mapping

[15]The geometrical position of each CSP can be seen in tab. C.1 (the right side of the table).

B
Chronograms

The chronograms shows the responses of some of my designs when they underwent simulation in Modeltech ModelSim. The testbenches used to create stimuli to the design inputs are mentioned in 2.2.9. The signal names are the same as in the design code (see [srcTRUDESIGN]), so unless you have a look this aswell the chronograms will probably seem cryptic to you. If you just wanted to have a quick look then consider them a technological preview of the “waveform” function in ModelSim.

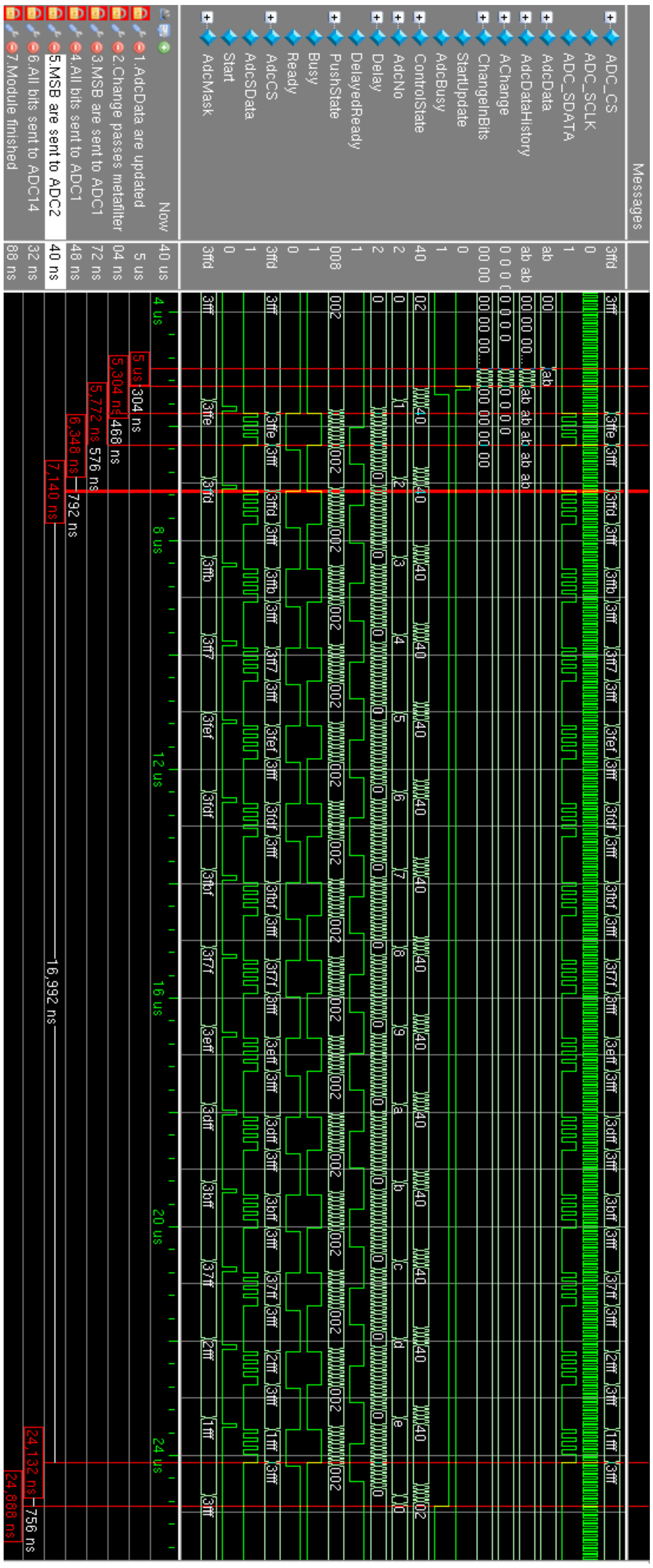


Fig. B.1 - The Process of Updating the ADCs with new Data

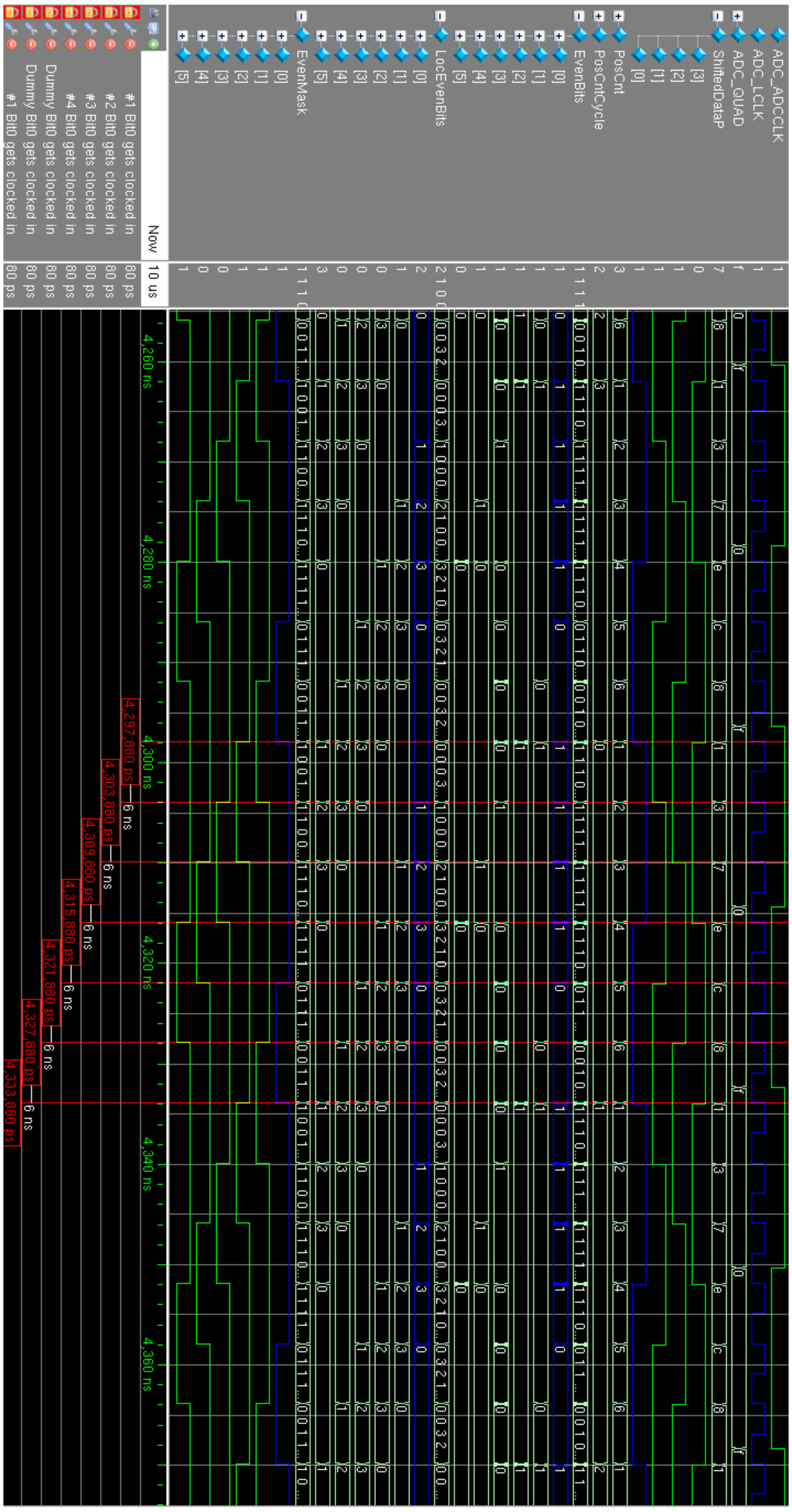
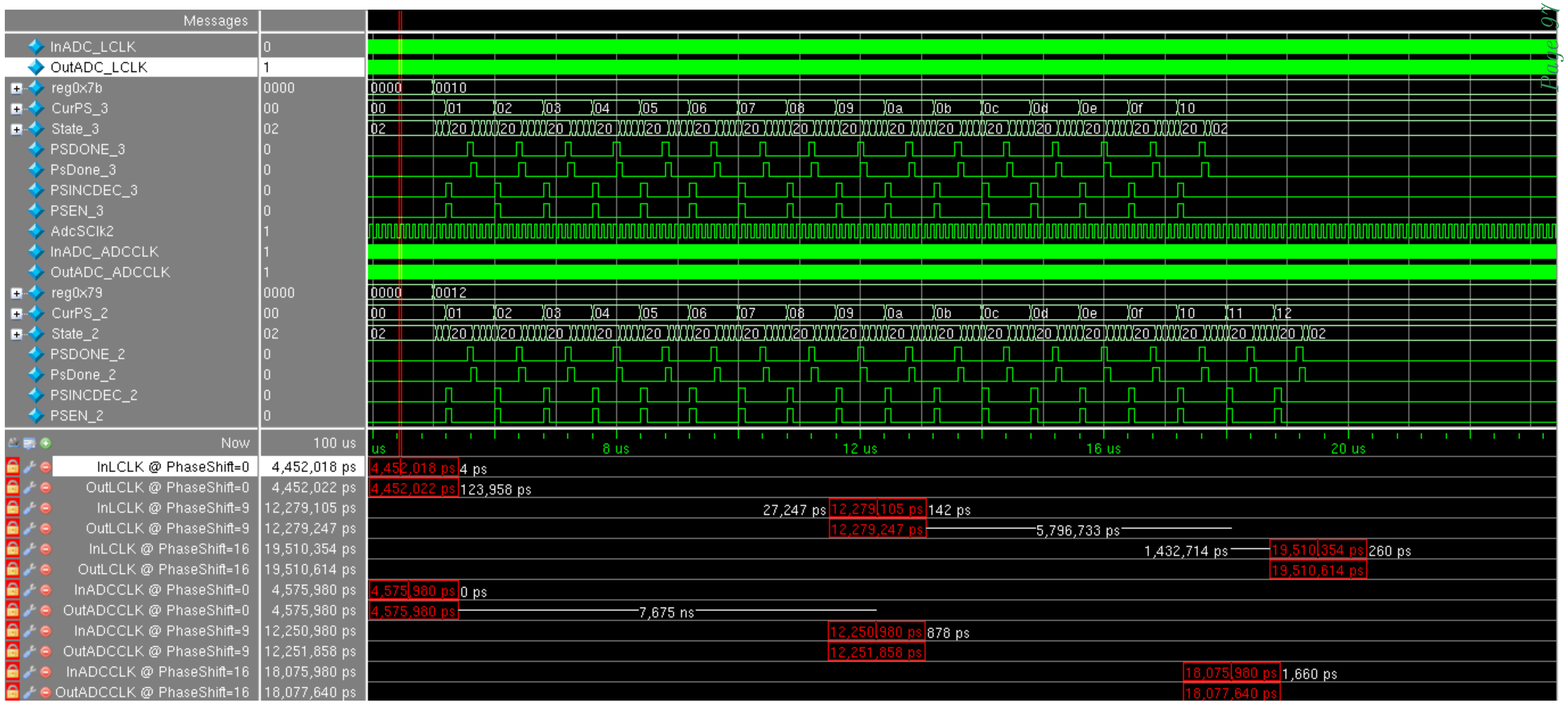


Fig. B.2 - First Stage of Deserialisation - Sorting the Bits [sofSIM]



Messages	
InADC_LCLK	0
OutADC_LCLK	1
reg0x7b	0000
CurPS_3	00
State_3	02
PSDONE_3	0
PsDone_3	0
PSINCDEC_3	0
PSEN_3	0
AdcSClk2	1
InADC_ADCCLK	1
OutADC_ADCCLK	1
reg0x79	0000
CurPS_2	00
State_2	02
PSDONE_2	0
PsDone_2	0
PSINCDEC_2	0
PSEN_2	0
Now	100 us
InLCLK @ PhaseShift=0	4,452,018 ps
OutLCLK @ PhaseShift=0	4,452,022 ps
InLCLK @ PhaseShift=9	12,279,105 ps
OutLCLK @ PhaseShift=9	12,279,247 ps
InLCLK @ PhaseShift=16	19,510,354 ps
OutLCLK @ PhaseShift=16	19,510,614 ps
InADCCLK @ PhaseShift=0	4,575,980 ps
OutADCCLK @ PhaseShift=0	4,575,980 ps
InADCCLK @ PhaseShift=9	12,250,980 ps
OutADCCLK @ PhaseShift=9	12,251,858 ps
InADCCLK @ PhaseShift=16	18,075,980 ps
OutADCCLK @ PhaseShift=16	18,077,640 ps

Notice the phasedifference of input/output clocks:

ADC_LCLK with 0 taps of phaseshift = 0ps
ADC_LCLK with 16 taps of phaseshift = 260ps
ADC_ADCCLK with 0 taps of phaseshift = 0ps
ADC_ADCCLK with 16 taps of phaseshift = 1660ps

The ratio 260ps/1660ps $\approx T_{ADC\_LCLK}/T_{ADC\_ADCCLK}$, and a full period will be shifted at 255 taps.

Fig. B.3 - Synchronising ADC Inputs with DCM [sofSIM]

C
TRU Communication

In this attachment I will describe the various TRU registers currently[1] available for the TRU. However, as a starting point I will show a few scripts I made in BASH which makes the task of communicating with the TRU pretty easy.

I will use the ALICE ReadOut protocol (ALTRO)[2] for this, although communication can also be done via the low-speed serial lines (slow control)[3]. While these are BASH scripts, my two `C++` programs (the DCS Offline Control Utility and TRU Register Scanner) have both slow control and ALTRO functionality implemented, see 4 for a description of these programs. I also implemented the ALTRO protocol in my TRU Testbench, as a means to validate the GTL controller (see 2.2.9).

The registers listed here comes straight from the code, but has been trimmed down quite a bit since earlier versions of the code. The reason for this is that several registers were addressing functionality that is no longer present, or - if present - has been rendered redundant due to a change in design.

[1]Register list as of 30. June 2008

[2]For more information about the ALTRO protocol, take a look at the ALTRO User Manual [docALTROUM] and the RCU Firmware Manual [docRCUFM].

[3]See RCU Firmware Manual for more information about the slow control protocol [docRCUFM]. This communication protocol does not work very well with the TRU yet.

C.1 Reading from the TRU

Make a scriptfile on the DCS card (or in a folder mounted on the DCS card) - say `<path to file>/ReadFromTRUReg.sh` and put the following text in it:

```
#! /bin/bash

if [ $# -ne 2 ]; then
    echo 1>&2 "Usage: $0 <register> <branch>"
    echo 1>&2 "Example: $0 72 a"
    echo 1>&2 "Read the register 72 on branch a"
    exit 127
fi

branch=""

if [ ${2}x = "ax" ]; then
branch=2
fi

if [ ${2}x = "bx" ]; then
branch=3
fi

if [ ${branch}x = "x" ]; then
echo 1>&2 "Not a valid branch!"
exit 127
fi

rcu-sh w 0x7000 0x5${branch}00${1}    #Setup address to read from.
rcu-sh w 0x0 0x300000                 #Set stop flag.
rcu-sh wait 1 us
rcu-sh r 0x7800                       #Read ErrorStatus Register
rcu-sh wait 1 us
rcu-sh r 0x6000 1                     #Output the value read back*
```

Now call the command - e.g. do `<path to file>/ReadFromTRUReg.sh 72 a` - and see what value register `0x072` (which holds the value for the Global Level0 Threshold) on branch `a` contained.

If you really want to know what all these commands do, feel free to look it up in the ALTRO User Manual [docALTROUM] or in the PHOS User Manual [docPHOSUM].

C.2 *Writing to the TRU*

Make a scriptfile on the DCS card (or in a folder mounted on the DCS card) - say `<path to file>/WriteToTRUReg.sh` and put the following text in it:

```
#! /bin/bash

if [ $# -ne 3 ]; then
  echo 1>&2 "Usage: $0 <value> <register> <branch>"
  echo 1>&2 "Example: $0 00ff 72 a"
  echo 1>&2 "Write the value ff to TRU register 72 on branch a"
  exit 127
fi

branch=""

if [ ${3}x = "ax" ]; then
branch=2
fi

if [ ${3}x = "bx" ]; then
branch=3
fi

if [ ${branch}x = "x" ]; then
echo 1>&2 "Not a valid branch!"
exit 127
fi
rcu-sh w 0x6c01 0x0                 #Clear the Error Status Flag
rcu-sh w 0x7000 0x6${branch}00${2}    #Setup address to write to.
rcu-sh w 0x7001 0x70${1}            #Setup data to write.
rcu-sh w 0x7002 0x5${branch}00${2}    #Setup address to read from.
rcu-sh w 0x7003 0x390000            #Set stop flag.
rcu-sh wait 10 us
rcu-sh w 0x0 0x300000               #Execute
rcu-sh wait 10 us
rcu-sh r 0x7800                     #Read ErrorStatus Register
rcu-sh wait 10 us
rcu-sh r 0x6000 1                   #Output the value read back*

if [ $# -ne 3 ]; then
  echo 1>&2 "Usage: $0 <value> <register> <branch>"
  echo 1>&2 "Usage: $0 <value> <register> <branch>"
  echo 1>&2 "Example: $0 00ff 72 a"
  echo 1>&2 "Write the value ff to TRU reguster 72 on branch a"
  exit 127
fi
```

Now call the command - e.g. do `<path to file>/WriteToTRUReg.sh 00ff 72 a` - and find that the value `0x00ff` was written to TRU register `0x072` on branch `a` (which holds the value for the Global Level0 Threshold).

C.3 Register List

In case you wonder, these registers should be the same as the ones being scanned if you are using my TRU Register Scanner (see 4.5). The format is as follows:

```
<register address>  [design name]  <'R'ead-only or 'R'ead/'W'rite>   <A short description>
```

Temperature, Voltage and Current Sensors

```
0x01     [T1_TH]       R/W     Maximum Temperature
0x02     [A4V0_TH]     R/W     Min. 4V Ana. Voltage
0x03     [A4V0C_TH]    R/W     Max. 4V Ana. Current
0x04     [D4V2_TH]     R/W     Min. 4.2V Dig. Voltage
0x05     [D4V2C_TH]    R/W     Max. 4.2V Dig. Current
0x06     [TEMP1]        R      Temperature Value ( decimal value/4 = temperature
0x07     [A4V0]         R      4V Analogue Voltage Value
0x08     [A4V0C]        R      4V Ana. Current Value
0x09     [D4V2]         R      4.2V Dig. Voltage Value
0x0a     [D4V2C]        R      4.2V Dig. Current Value
```

Interrupt registers

```
0x11     [CSR0]         R      Interrupt - Mask Register
0x12     [CSR1]         R      Error Status Register
```

Fake ALTRO register

```
0x1A     [CHRDO]        W      Channel ReadOut (Fake ALTRO register)
```

Note: This is not a BC register!

The TRU retains a history of 256 samples (or 256*40MHz = 6.4μs) in Block Select RAM (which is pedestal corrected), of which the 10 last samples (250ns) may be read as trigger data type 1. This is what the Fake ALTRO readout mechanism are for.

Consult the PHOS User Manual [docPHOSUM, page 102] for the ALTRO protocol data format.

Important! When the ALTRO Readout command are executed the RCU are addressing the ALTRO chips (which are located on the FEE cards, see [docPHOSUM] for more information). This means that my "example script" can not be used to launch this command ("branch numbers" must be changed from 2 and 3 to 0 and 1, respectively). Also, an extension in the TRU BoardController will be needed here in order not to ignore this command as an ALTRO request (a command is either a BoardController or ALTRO request, never both).

Firmware revision

```
0x20     [BC_VER]       R      Version of BoardController
```

Temperature, Voltage and Current Sensors

0x21	[T2_TH]	R/W	Maximum Temperature
0x22	[D3V3_TH]	R/W	Min. 3.3V Dig. Voltage
0x23	[D3V3C_TH]	R/W	Max. 3.3V Dig. Current
0x24	[D2V5_TH]	R/W	Min. 2.5V Dig. Voltage
0x25	[D1V5_TH]	R/W	Min. 1.5V Dig. Voltage
0x28	[TEMP2]	R	Temperature Value (decimal value/4 = temperature)
0x29	[D3V3]	R	3.3V Dig. Voltage Value
0x2a	[D3V3C]	R	3.3V Dig. Current Value
0x2b	[D2V5]	R	2.5V Dig. Voltage Value
0x2c	[D1V5]	R	1.5V Dig. Voltage Value
0x2e	[ADC_ID]	R	Temp./Voltage Sensor ID
0x2f	[TEST_REG]	?	Test Register???

Auxiliary Interrupt registers for Monitoring (not clear if we need these)

0x65	[VTS]	bits to be re-allocated for TRU	Aux. Interrupt Flag register
0x66	[INT_MASK]	bits to be re-allocated for TRU	Interrupt mask

Hardware Versions

0x70	[CARD_NO]	R	TRU board version (Current is 2.0)
0x71	[SER_NO]	R	TRU serial Number (just upcounting)

Global Level0 Threshold

0x72	[GTHR_L0]	R/W	L0 Global Threshold

This threshold only applies for those 4x4 which has a “local 4x4 threshold” set to a value unequal to 0, see the registers `0xa5` to `0xff`.

TOR Readout Sync

0x78	[TORSYNC]	R/W	Relative bunch number register (8 bits)

The TOR will receive 14 bit time-summed data which are stored in Block Select RAM (since these are huge a history of 256 samples will be retained). The plan is to use this register to specify the position of the data in the RAM block to fineadjust what data the TOR receives. The position will be relative to “position of peak” which will be internally stored for the last Level0 produced.

The first bunch will arrive at the TRU ≈350ns after beam crossing (particle collision). After 580ns the Level0 must be produced and so the necessary history to store is 580ns350ns = 230ns- which translates to roughly 10 samples.

ADC Deserialiser Sync

```
0x79      [ADCoutSYNC] R/W     Value 0..511 - PhaseShift for DCM sourcing "frameclock"
0x7a      [ADCinSYNC]  R/W     Value 0..511 - PhaseShift for DCM sourcing "bitclock"
```

Refer to attachment A.6 for instructions on how to use these registers.

ADC Control Register

```
0x7c      [ADCCONTROL] R/W     ADC serial data pattern (ADS5270 data sheet)
```

ADDRESS				DATA				DESCRIPTION	REMARKS
D7	D6	D5	D4	D3	D2	D1	D0		All Data Outputs
0	0	0	0					**LVDS Buffers**	
				0	0			Normal ADC Output	(default after reset)
				0	1			Deskew Pattern	*
				1	0			Sync Pattern	*
				1	1			Custom Pattern	*
						0	0	I_{OUT} LVDS ≈ 3.5mA	(default after reset)
						0	1	I_{OUT} LVDS ≈ 2.5mA	
						1	0	I_{OUT} LVDS ≈ 4.5mA	
						1	1	I_{OUT} LVDS ≈ 6.0mA	
0	0	0	1					**CLOCK CURRENT**	
				0	X	X	0	LVDS Clock Output Current	I_{OUT} ≈ 3.5mA(default)
				0	X	X	1	2x LVDS Clock Output Current	I_{OUT} ≈ 7.0mA
0	0	0	1					**LSB/MSB MODE**	
				0	0	X	X	LSB First Mode	(default after reset)
				0	1	X	X	MSB First Mode	
0	0	1	0					**POWER DOWN ADC CH.**	
				X	X	X	X	Power-Down Ch. 1 to 4.	
								D3 = Ch.4, D0 = Ch.1	
0	0	1	1					**POWER DOWN ADC CH.**	
				X	X	X	X	Power-Down Ch. 5 to 8.	
								D3 = Ch.8, D0 = Ch.5	
								CUSTOM PATTERN	
0	1	0	0	X	X	X	X		*
0	1	0	1	X	X	X	X		*
0	1	1	0	X	X	X	X		*

* *Take a look at tab. C.2*

Tab. C.1 - Commandwords you can send to the ADC [docADC][Page 6]

Serial Output	**LSB**											
ADC Output	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11
Deskew Pattern	1	0	1	0	1	0	1	0	1	0	1	0
Sync Pattern	0	0	0	0	0	0	1	1	1	1	1	1
Custom Pattern	$D0_4$	$D1_4$	$D2_4$	$D3_4$	$D0_5$	$D1_5$	$D2_5$	$D3_5$	$D0_6$	$D1_6$	$D2_6$	$D3_6$

Tab. C.2 - Description ADC Patterns [docADC][Page 6]

Trigger mask registers for 2x2 inputs before pedestal correction

```
0x7d     [mask_reg1]  R/W     Mask Reg. ADC1&2
0x7e     [mask_reg2]  R/W     Mask Reg. ADC3&4
0x7f     [mask_reg3]  R/W     Mask Reg. ADC5&6
0x80     [mask_reg4]  R/W     Mask Reg. ADC7&8
0x81     [mask_reg5]  R/W     Mask Reg. ADC9&10
0x82     [mask_reg6]  R/W     Mask Reg. ADC11&12
0x83     [mask_reg7]  R/W     Mask Reg. ADC12&14
```

ChipScope MUX table (diagnostics via. JTAG)

```
0x92     [CSSelect]   R/W     ChipScope Input Select
```

ChipScope can - with the core currently[4] compiled into the TRU code - look at a variable 192 bits wide over a maximum of 8192 clock cycles. I have tried in several ways to sample the mentioned 192 bit variable at high speeds (200MHz+), and in most of my versions of the TRU code this has been the case. However, as from version 0.1.49 and onwards I settled with 80MHz clock speed. This means that when you read out data using ChipScope now you will buffer a total of *8192 samples * 80MHz* $= 102.4\mu s$

The `CSSelect` register can hold the values mentioned in tab. C.3. I also supply the name of the ChipScope project you should load together with the choice of the MUX value, simply because this will give you the name of the signals in ChipScope (a struggle to read the values if busses etc. are not defined properly).

CSSelect Value	**Data Sent to ChipScope**	**Which Project to Load in ChipScope**
0x0000	Pedestal Corrected 2x2 from ADC1&2	TRUNew.cpj
0x0001	Pedestal Corrected 2x2 from ADC3&4	TRUNew.cpj
0x0002	Pedestal Corrected 2x2 from ADC5&6	TRUNew.cpj
0x0003	Pedestal Corrected 2x2 from ADC7&8	TRUNew.cpj
0x0004	Pedestal Corrected 2x2 from ADC9&10	TRUNew.cpj
0x0005	Pedestal Corrected 2x2 from ADC11&12	TRUNew.cpj
0x0006	Pedestal Corrected 2x2 from ADC13&14	TRUNew.cpj

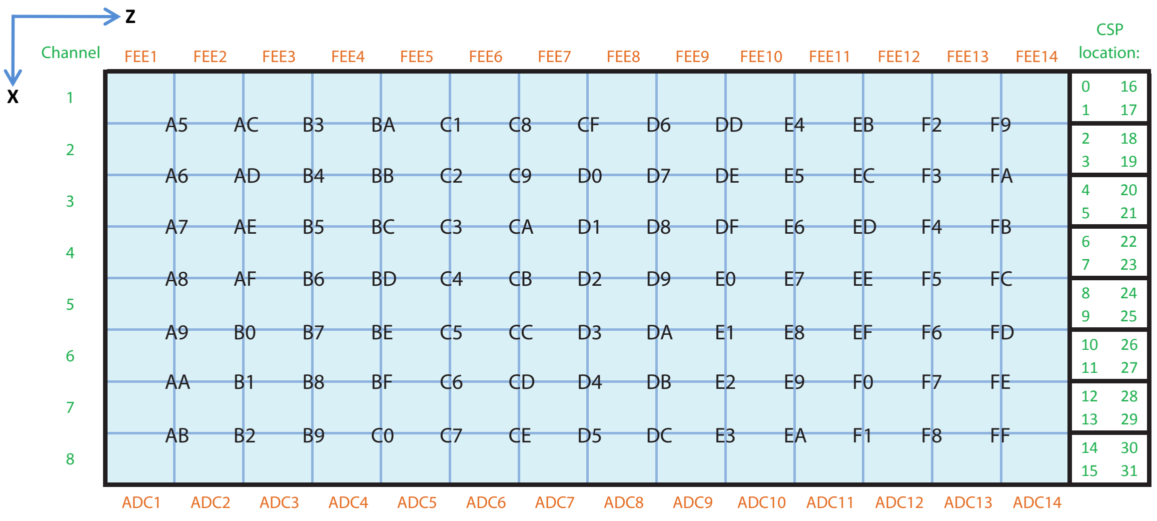
Tab. C.3 - ChipScope MUX Values

[4] As of 30. June 2008

Trigger threshold registers (91 instances)

```
0xa5
 to       [4x4Thr]     R/W     4x4 local thresholds to space-time-sums (16 bit )
0xff
```

To get some impression of which of the addressed `4x4Thr` registers applies to which geometrical 4x4 bunch of crystals, take a look at fig. C.1.



A5	AC	B3	BA	C1	C8	CF	D6	DD	E4	EB	F2	F9
A6	AD	B4	BB	C2	C9	D0	D7	DE	E5	EC	F3	FA
A7	AE	B5	BC	C3	CA	D1	D8	DF	E6	ED	F4	FB
A8	AF	B6	BD	C4	CB	D2	D9	E0	E7	EE	F5	FC
A9	B0	B7	BE	C5	CC	D3	DA	E1	E8	EF	F6	FD
AA	B1	B8	BF	C6	CD	D4	DB	E2	E9	F0	F7	FE
AB	B2	B9	C0	C7	CE	D5	DC	E3	EA	F1	F8	FF

Channel	CSP location:	
1	0	16
	1	17
2	2	18
	3	19
3	4	20
	5	21
4	6	22
	7	23
5	8	24
	9	25
6	10	26
	11	27
7	12	28
	13	29
8	14	30
	15	31

Fig. C.1 - 4x4 Threshold Map [sofXLS]

Fig. C.1 depicts a branch of 14 FEE cards (the TRU sits in the middle but I left it out), seen along the positive y-axis (that is, looking at the "back" of the PHOS module; the electronics layer). FEEs are mapped to equally indexed ADCs on the TRU, and the FastOR channels to equally indexed ADC channels. The numbering increases along the positive axis' of `x` and `z`, respectively. On the right side of the table I attached the CSP numbering (charge sensitive preamplifiers, one per crystal). This way you can now see which CSP are being mapped to which 4x4, and which 4x4 threshold(s) that are affecting it.

Example: `<path>/WriteToTRUReg.sh 00ff a5 a` would write to the TRU on branch `a`. You write to register `a5` the value `0x00ff`, which means that if the energy of CSP0,1,2,3,16,17,18,19 from FEE1 and FEE2 combined exceeds the value `0x00ff` then you have a level0. Remember that you sum up the energy not only in space, but also in time. Also, what the number `0x00ff` actually represent in respect to "energy" is not yet clear.

D
TRU Logic Utilisation

This appendix simply shows a small selection of the *MAP* process summary (a part of the implementation software in Xilinx ISE). They give an indication of the logical usage of the old versus the new TRU design, but should not be considered absolute as there are several factors that can affect them. First of all, the synthesiser, translate-, map- and place and route function can all introduce small variations to the final distribution of logic. Only a minor change in the code can cause several domino-effects which makes these tools decide to go for a complete different set of logical elements. Especially true is this for designs that consumes an amount of logic which is close to the limit of resources available in the FPGA. Then the tools will start to find ways to "pack" logic together and share common resources fabric, which makes it very hard to keep track of what sort of impact your change in design had for the total level of resource usage.

Also, I have seen that the numbers change quite a lot between different versions of Xilinx ISE. From version 9.1 to version 10.1 the register usage reported by XST (Xilinx ISE Synthesiser) increased by about 5%, while the place and route tools reported about the same decrease. Both of the summaries below is therefore from an implementation with ISE 10.1.

The single most important reason why you should not compare the numbers listed here directly is that the old and the new design *contains different functional elements*. Some new functions were added, some old and redundant were removed, and a lot of the code was revised. These factors aside, please feel free to have a look if you are interested. Note that the new design uses less logic in general (the CLB usage is below the critical 90% level), and takes only a fraction of the time to implement.

D.1 Old Design

```
Design Summary
--------------
Number of errors:      0
Number of warnings:   59
Logic Utilisation:
  Total Number Slice Registers:      32,505 out of  47,232   68%
    Number used as Flip Flops:       32,491
    Number used as Latches:              14
  Number of 4 input LUTs:            32,809 out of  47,232   69%
Logic Distribution:
  Number of occupied Slices:         23,614 out of  23,616   99%
    Number of Slices containing only related logic:  19,068 out of  23,614  80%
    Number of Slices containing unrelated logic:      4,546 out of  23,614  19%
      *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:      34,706 out of  47,232   73%
    Number used as logic:            29,978
    Number used as a route-thru:      1,897
    Number used for 32x1 RAMs:           20
      (Two LUTs used per 32x1 RAM)
    Number used as Shift registers:   2,811
  Number of bonded IOBs:                395 out of     692   57%
    IOB Flip Flops:                       5
    IOB Master Pads:                    118
    IOB Slave Pads:                     118
  Number of TBUFs:                        1 out of  11,808    1%
  Number of RAMB16s:                    130 out of     232   56%
  Number of BUFGMUXs:                    11 out of      16   68%
  Number of DCMs:                         6 out of       8   75%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           15

Peak Memory Usage:  734 MB
Total REAL time to MAP completion:  12 mins 41 secs
Total CPU time to MAP completion:   12 mins 27 secs
```

D.2 New Design

```
Design Summary:
Number of errors:      0
Number of warnings:  131
Logic Utilisation:
  Total Number Slice Registers:      17,192 out of  47,232   36%
    Number used as Flip Flops:       16,982
    Number used as Latches:             210
  Number of 4 input LUTs:            26,007 out of  47,232   55%
Logic Distribution:
  Number of occupied Slices:         20,882 out of  23,616   88%
    Number of Slices containing only related logic:  20,882 out of  20,882 100%
    Number of Slices containing unrelated logic:          0 out of  20,882   0%
      *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:      27,566 out of  47,232   58%
    Number used as logic:            24,798
    Number used as a route-thru:      1,559
    Number used for Dual Port RAMs:     672
      (Two LUTs used per Dual Port RAM)
    Number used for 32x1 RAMs:           20
      (Two LUTs used per 32x1 RAM)
    Number used as Shift registers:     517
  Number of bonded IOBs:                396 out of     692   57%
    IOB Master Pads:                    144
    IOB Slave Pads:                     144
  Number of TBUFs:                        1 out of  11,808    1%
  Number of RAMB16s:                     50 out of     232   21%
  Number of BUFGMUXs:                     9 out of      16   56%
  Number of DCMs:                         3 out of       8   37%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           15

Peak Memory Usage:  576 MB
Total REAL time to MAP completion:  2 mins 24 secs
Total CPU time to MAP completion:   2 mins 19 secs
```

E
Fascinating Facts

Please note that this is a copy-paste (with few edits) from [docLHCUG], but I put them in anyway since I found them to be a good read.

1. When the 27-km long circular LHC tunnel was excavated, between Lake Geneva and the Jura mountain range, the two ends met up to within 1 cm.
2. Each of the 6400 superconducting filaments of niobium-titanium in the cable produced for the LHC is about 0.007mm thick, about 10 times thinner than a normal human hair. If you added all the filaments together they would stretch to the Sun and back five times with enough left over for a few trips to the Moon.
3. All protons accelerated at CERN are obtained from standard hydrogen. Although proton beams at the LHC are very intense, only 2 nanograms of hydrogen are accelerated each day. Therefore, it would take the LHC about 1 million years to accelerate 1 gram of hydrogen.
4. The central part of the LHC will be the world's largest fridge. At a temperature colder than deep outer space, it will contain iron, steel and the all important superconducting coils.
5. The pressure in the beam pipes of the LHC will be about ten times lower than on the Moon. This is an ultrahigh vacuum.
6. Protons at full energy in the LHC will be travelling at 0.999999991 times the speed of light. Each proton will go round the 27 km ring more than 11 000 times a second.
7. At full energy, each of the two proton beams in the LHC will have a total energy equivalent to a 400 t train (like the French TGV) travelling at 150 km/h. This is enough energy to melt 500 kg of copper.
8. The Sun never sets on the ATLAS collaboration. Scientists working on the experiment come from every continent in the world, except Antarctica.
9. The CMS magnet system contains about 10 000 t of iron, which is more iron than in the Eiffel Tower.
10. The data recorded by each of the big experiments at the LHC will be enough to fill around 100 000 DVDs every year.

List of Tables

List of Figures

Bibliography

[MyMind] *Definately my greatest source of information.*

Books

[bokAsh08] **Peter J. Ashenden,** *Digital Design - An Embedded Systems Approach Using Verilog,* Morgan Kaufman (2008), ISBN:978-0-12-369527-7

Documents

[docAD7417] **Temperature Sensor**
http://www.joinge.net/cern/report/bib/AD7417.pdf

[docADC] **TRU ADC (ADS5270)**
http://www.joinge.net/cern/report/bib/ADC.pdf

[docADG774] **Multiplexer/demultiplexer**
http://www.joinge.net/cern/report/bib/ADG774.pdf

[docALICE] **ALICE Brochure No. 2 2003**
http://www.joinge.net/cern/report/bib/ALICE.pdf

[docALTRORC] **ALTRO ReadOut Chip - IEEE Transactions on Nuclear Science Vol. 50, No. 6, December 2003**
http://www.joinge.net/cern/report/bib/ALTRORC.pdf

[docALTROUM] **ALICE ReadOut Protocol (ALTRO) User Manual June 2002**
http://www.joinge.net/cern/report/bib/ALTROUM.pdf

[docAPA075] **ProASIC Flash Memory**
http://www.joinge.net/cern/report/bib/APA075.pdf

[docCERN] **CERN Brochure No. 2 2008**
http://www.joinge.net/cern/report/bib/CERN.pdf

[docDAVID]

[docDAVID] **David Koenigseders Activity Report** ϕ
http://www.joinge.net/cern/report/bib/DAVID.pdf

[docGRID] **GRID Brochure No. 2 2006**
http://www.joinge.net/cern/report/bib/GRID.pdf

[docGTL] **GTL Drivers**
http://www.joinge.net/cern/report/bib/GTL.pdf

[docLHC] **LHC Brochure No. 2 2006**
http://www.joinge.net/cern/report/bib/LHC.pdf

[docLHCUG] **LHC The Ultimate Guide No. 1 2008**
http://www.joinge.net/cern/report/bib/LHCUG.pdf

[docLSF] **Olivier Bourrion - (TOR-TRU) Link Speed Feasability Test Report (rev. 10/07/2007)**
http://www.joinge.net/cern/report/bib/LSF.pdf

[docLVDS101] **LVDS Driver**
http://www.joinge.net/cern/report/bib/LVDS101.pdf

[docLVDS31] **LVDS Driver**
http://www.joinge.net/cern/report/bib/LVDS31.pdf

[docLVDS32] **LVDS Receiver**
http://www.joinge.net/cern/report/bib/LVDS32.pdf

[docMAX6627] **Temperature Sensor**
http://www.joinge.net/cern/report/bib/MAX6627.pdf

[docMPC9109] **Clock Distribution Chip**
http://www.joinge.net/cern/report/bib/MPC9109.pdf

[docMX29LV] **Flash memory**
http://www.joinge.net/cern/report/bib/MX29LV.pdf

[docNICOLAS] **Nicholas Degrenne' Activity Report - 2007**
http://www.joinge.net/cern/report/bib/NICOLAS.pdf

[docPHOSUM] **PHOton Spectrometer (PHOS) User Manual - 4. January 2007 (Hans Muller)**
http://www.joinge.net/cern/report/bib/PHOSUM.pdf

[docRCUFM] **ReadOut Control Unit (RCU) Firmware Manual 18.December 2006**
http://www.joinge.net/cern/report/bib/RCUFM.pdf

[docSO2222] **Power Supply Transistor**
http://www.joinge.net/cern/report/bib/SO2222.pdf

[docTIMCLOS] **Timing Closure Techniques (supplied by a Xilinx designer on the Xilinx forums)**
http://www.joinge.net/cern/report/bib/TIMCLOS.pdf

[docTORDESIGN] **TOR Design Sheet**
http://www.joinge.net/cern/report/bib/TORDESIGN.doc

[docTRU1.1] **TRU Design Sheet v1.1**
http://www.joinge.net/cern/report/bib/TRU1.1.pdf

[docV2PDS] **Virtex 2 Pro Datasheet**
http://www.joinge.net/cern/report/bib/V2PDS.pdf

[docVR15V] **1.5V Voltage Regulator**
http://www.joinge.net/cern/report/bib/VR15V.pdf

[docVR18V] **1.8V Voltage Regulator**
http://www.joinge.net/cern/report/bib/VR18V.pdf

[docVR25V] **2.5V Voltage Regulator**
http://www.joinge.net/cern/report/bib/VR25V.pdf

[docVR33BU] **3.3V Voltage Regulator**
http://www.joinge.net/cern/report/bib/VR33BU.pdf

[docVR33WU1] **3.3V Voltage Regulator**
http://www.joinge.net/cern/report/bib/VR33WU1.pdf

[docVR33WU2] **3.3V Voltage Regulator**
http://www.joinge.net/cern/report/bib/VR33WU2.pdf

[docV2PUG] **Virtex 2 Pro User Guide**
http://www.joinge.net/cern/report/bib/V2PUG.pdf

[docXAPP194] **Xilinx Application Note - Serial to Parallel Converter**
http://www.joinge.net/cern/report/bib/XAPP194.pdf

[docXAPP225] **Xilinx Application Note - Data to Clock Phase Alignment**
http://www.joinge.net/cern/report/bib/XAPP225.pdf

[docXAPP774] **Xilinx Application Note - Connecting to TI ADS527x Series ADCs**
http://www.joinge.net/cern/report/bib/XAPP774.pdf

[docXCF32P] **Configurable Flash Memory**
http://www.joinge.net/cern/report/bib/XCF32P.pdf

[docXCJ50] **XCell Journal Issue 50**
http://www.joinge.net/cern/report/bib/XCJ50.pdf

[docXCSTT] **Xilinx ChipScope Tutorial**(*Ben Nham, Stanford University*)
http://www.joinge.net/cern/report/bib/XCJ50.pdf

Images

[imgALIDET] **Image of the ALICE Detector**
http://doc.cern.ch//archive/electronic/cern/others/multimedia/poster/poster-2004-004.pdf

[imgCAC] **Image of the CERN Accelerator Complex**
http://public.web.cern.ch/PUBLIC/en/Research/AccelComplex-en.html

[imgCGL] **Image of the CERN Globe**
http://www.theage.com.au/ffximage/2007/02/18/18e_cern_wideweb__470x308,0.jpg

[imgEAS] **Image of the Earth and Stars**
http://science.punchstock.com/images/galleries/space/bxp46188.jpg

Internet

[intAHP] **ALICE Homepage**
http://aliceinfo.cern.ch

[intASW] **ASIC World**
http://www.asic-world.com

[intBGB] **Big Bang**
http://en.wikipedia.org/wiki/Big_Bang

[intCHP] **CERN Homepage**
http://public.web.cern.ch/public/

[intDCSCOMP] **DCS Cross Compiler**
http://frodo.nt.fh-koeln.de/~tkrawuts/dcs.html

[intHBU] **How Big is the Universe?**
http://www.pbs.org/wgbh/nova/universe/howbig.html

[intIETNS] **IEEE Transactions on Nuclear Science (vol. 55, no. 1, Feb. 2008)** http://web.ift.uib.no/~kjeks/wiki

[intIFT] **IFT-Wiki** http://web.ift.uib.no/~kjeks/wiki

[intMETA] **Wikipedia - Metastability in Electronics** http://en.wikipedia.org/wiki/Metastability_in_electronics

[intNSS] **T Oetiker, H Partl, I Hyna, E Schlegl - The Not So Short Introduction to LaTeX 2_ε** http://www.ctan.org/tex-archive/info/short

[intNTM] **Carol A. Vidoli - Technical Report Writing (rev. 5/18/00)** http://grcpublishing.grc.nasa.gov/Editing/vidcover.CFM

[intPRESS] **The CERN Press Service** http://press.web.cern.ch/press/

Magasines

[magSCAM] **Scientific American February 2008,** Special Report - *The Future of Physics (Page 30)*

Software - Third Party

[sofGMP] *GNU Image Manipulation Program (GIMP). A neat little program I often used in conjunction with Excel/PowerPoint.*

[sofPPT] *Microsoft Powerpoint 2007. I made almost all the illustrations with this software.*

[sofPSP] *Photoshop. A really powerful image editor I often found myself using in conjunction with Excel/PowerPoint.*

[sofSIM] *Modeltech ModelSim 6.3. The software of choice when I needed to simulate of HDL code and to generate chronograms.*

[sofVST] *Microsoft Visual Studio 2008. Used when I wrote and debugged C++ code.*

[sofXLS] *Microsoft Excel 2007. Used when I needed more advanced table features than Latex had to offer.*

SourceCode

[srcPHOSCTRL] **PHOS Offline Control Utility** http://www.joinge.net/cern/report/src/rcu/control_program/

List of executed commands	Executed.txt
Executable	HARD_TRIG
Settings file	HARD_TRIG_SETTINGS
Optional. "Memory" of which cards are active	currentACL.txt
The sourcecode	HARD_TRIG.cpp
File used to get output from Linux commands into the program	var.tmp

[srcMAPCHECK] **TOR Mapping Deviation Checker**
http://www.joinge.net/cern/report/src/etc/MAPCHECK.cpp

[srcTRUDESIGN] **TRU Design**
http://www.joinge.net/cern/report/src/tru/

The initial code (TBCv0.2), received 27.8.2007	TRUCode - TBCv0.2
TBCv0.2 base, beta ALTRO, received 10.9.2007	TRUCode - betaFakeALTRO

[srcTRUREGSCAN] **TRU Register Scanner**
http://www.joinge.net/cern/report/src/rcu/tru_test/

Executable	TRUTestFinal
Log file	log.txt
Example of result output	results.txt
The sourcecode	TRUTestFinal.cpp
Settings file	testopt.txt
File used to get output from Linux commands into the program	var.tmp

Glossary

Notation	Description	Page List
ADC	Analog to Digital Converter	11, 12, 15, 16, 23, 25, 27–38, 42–45, 48, 49, 52, 53, 55, 56, 69, 72, 75, 77, 92, 105
ALICE	A Large Ion Collider Experiment [intAHP]	I, 3, 6–11, 14, 15, 23, 26, 27, 31, 49, 52, 57, 64, 65, 72–74, 88, 90, 98
ALTRO	ALICE ReadOut protocol. Used by the RCUs to communicate with the FEE and TRU cards.	26, 27, 62, 65, 67
ATLAS	A Toroidal LHC Apparatus	5, 6, 109
BASH	Bourne Again SHell	12, 64, 68, 74, 75, 77–79, 98
CAT7	Category 7 Cable	50, 51, 91
CERN	The European Organization for Nuclear Research, derived from *French Conseil Européen pour la Recherche Nucléaire* [intCHP, AboutUs - TheNameCERN]	I, 4–6, 11, 26, 44, 45, 54, 60, 62, 64, 73–77, 79, 80, 83, 84, 88, 90, 109
ChipScope	A "virtual oscilloscope" for debugging Xilinx FPGAs. See A.2.3.	31–33, 35, 42–44, 49, 53

Notation	Description	Page List
CLB	Configurable Logic Block, see 2.1.2.4 for more information.	19, 20, 22, 26, 37, 42, 45, 47, 49, 106
CMS	The Compact Muon Solenoid Experiment	6, 109
CSP	Charge Sensitive Preamplifier	75, 77, 93
CTP	Central Trigger Processor	8, 14, 52
DCI	Digital Controlled Impedance, see 2.1.2.7 for more information.	22, 35, 36
DCM	Digital Clock Manager, see 2.1.2.5 for more information.	21, 22, 28, 34–37, 39, 51
DCS	Detector Control System	I, 12, 13, 60, 62–64, 67, 68, 70–72, 74, 78, 88–90, 98–100
DDR	Dual Data Rate	22, 34, 38, 51
DHCP	Dynamic Host Configuration Protocol	62, 88
DNS	Dynamic Name Server	88, 89
EMCAL	ElectroMagnetic CALorimeter	23, 51
EMI	ElectroMagnetic Interference	31
FakeALTRO	An implementation of the ALTRO protocol on the TRU, which enables the RCUto read out the data which the TRUbased its Level0on.	25, 53, 69
FastOR	The FastOR pulse is the analog sum of a 2x2 matrix of crystals	14–16, 25, 31, 34, 45, 46, 48, 52, 53, 55

Notation	**Description**	**Page List**
FEE	Front-End Electronics	9, 10, 14–16, 23, 25, 26, 44, 62, 65–69, 71, 72, 75, 77, 93, 101, 105
FPGA	Field Programmable Gate Array	14, 16–25, 27, 31, 33–37, 43–45, 47, 49, 53, 56–60, 69, 74, 75, 77–81, 83, 106
GTL	Gunning Transceiver Logic (InALICE everyone refer to theALTRO bus as theGTL bus)	10, 15, 16, 24, 27, 54–56, 62, 65, 69, 70, 75, 93, 98
HDL	Hardware Description Language	55, 56, 74, 79
IOB	Input/Output Block, see 2.1.2.7 for more information.	22, 34–37
JTAG	Joint Test Action Group	16
Level0	Level 0 Trigger, see 1.5.1 for more information.	14, 24–26, 32, 45–48, 50, 52, 54–56, 74

Notation	Description	Page List
Level1	Level 1 Trigger, see 1.5.1 for more information.	25, 26, 50, 52
LHC	Large Hadron Collider	I, 5–7, 9, 11, 27, 28, 51, 73, 109
LHCb	The Large Hadron Collider beauty Experiment	6
LSB	Least Significant Bit	31, 48
LUT	LookUp Table, see 2.1.2.4 for more information.	19, 38, 44
LVDS	Low Voltage Differential Signaling	10, 35, 50, 57, 59, 61, 91
MSB	Most Significant Bit	30, 31, 48, 52
NFS	Network FileSystem	63, 78, 89
PHOS	ALICE Photon Spectrometer	I, 8–11, 13, 14, 23, 25, 26, 31, 50, 52, 58, 60, 62, 64, 65, 67–74, 77, 78, 83, 90, 93, 99, 101, 105

Notation	Description	Page List
PLL	Phase Locked Loop	31
RAM	Read Access Memory	18, 19, 22, 25, 26, 35, 38, 39, 44, 45, 49, 53
RCU	Readout Control Unit	I, 9–12, 15, 24–27, 53, 55, 60, 62, 64–69, 71, 72, 74, 78, 88, 93, 98, 101
RMS	Root Mean Square, an indication of the power of a signal	69
SAMBA	A Service for Print and File Sharing	78
SSH	Secure Shell	78, 90
STU	Summary Trigger Unit	51
TOR	Trigger OR	I, 11, 13, 25–27, 49–54, 57–61, 74, 78, 83, 88, 91, 102
TP	Twisted Pair	50, 51, 58, 91
TPC	TPC	65, 71
TRU	Trigger Region Unit	I, 9–18, 23–32, 34, 35, 37, 38, 43–48, 50–60, 62, 64–72, 74, 75, 77, 81, 83, 91, 93, 98, 100–102, 104–106

Notation	Description	Page List
TTC	Timing & Trigger Controller	51