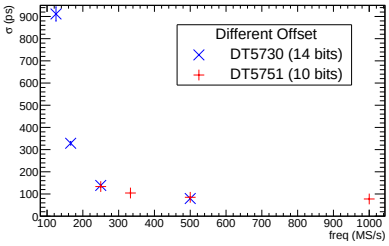
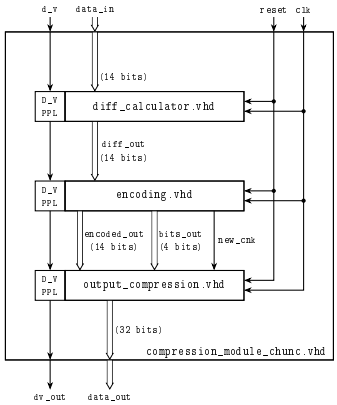
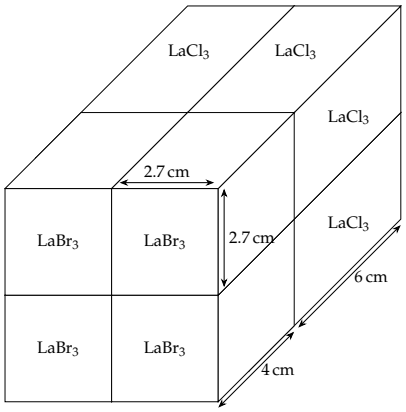
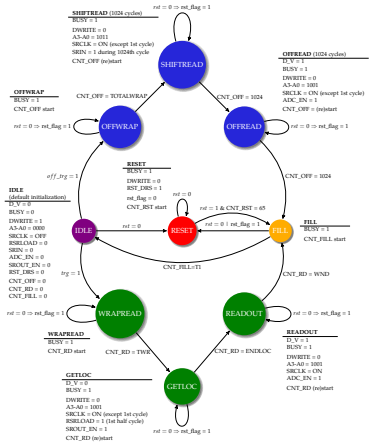
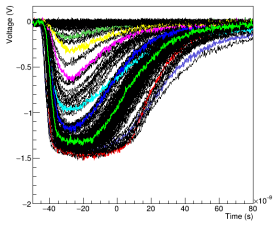
CHALMERS

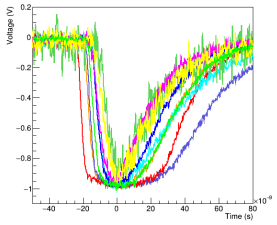


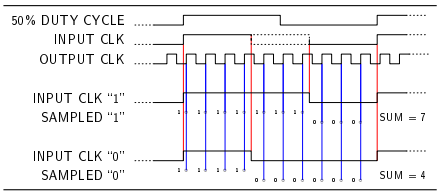












How to Lighten Experimentalists' Life with Electronics

Development of Front-End Readout for the CALIFA Detector

Master's thesis in Subatomic Physics

GIOVANNI BRUNI

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2017

MASTER'S THESIS IN SUBATOMIC PHYSICS

How to Lighten Experimentalists' Life with Electronics

Development of Front-End Readout for the CALIFA Detector

GIOVANNI BRUNI

Department of Physics
Division of Subatomic and Plasma Physics
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2017

How to Lighten Experimentalists' Life with Electronics
Development of Front-End Readout for the CALIFA Detector
GIOVANNI BRUNI

ISSN 1652-8557
Department of Physics
Division of Subatomic and Plasma Physics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
In the center a schematic of the CEPA4 prototype is depicted. On the top, results from studying the time resolution of phoswich crystals varying the sampling frequency are reported. On the right, the finite state machine developed for controlling the DRS4 chip is drawn. On the bottom right, signals and their sampling used in the new clock distribution scheme are depicted. On the bottom left, digitized traces from CEPA4 showing saturation of the photomultiplier tube are plotted. On the left, the block diagram illustrating the DPTC compression module is given.

Chalmers Reproservice
Göteborg, Sweden 2017

How to Lighten Experimentalists' Life with Electronics
Development of Front-End Readout for the CALIFA Detector
Master's thesis in Subatomic Physics
GIOVANNI BRUNI
Department of Physics
Division of Subatomic and Plasma Physics
Chalmers University of Technology

Abstract

Research in subatomic physics covers areas ranging from the investigation of the structure of and the interactions in atomic nuclei, to developing new materials and new medical treatments. Experiments in this field often rely on high sampling rates, good time resolution and fast processing of big quantities of data. This thesis describes work done in developing parts of the CALIFA particle detector, to be located at the GSI/FAIR-facility, Darmstadt. Several topics are covered: the design of readout procedures for a switched capacitor array (DRS4), as well as data trace compression algorithms and their implementations in FPGAs. A new clock delivery system for detectors was investigated and LaBr3-LaCl3 crystals were characterizated. The work was carried out in collaboration with the Physics Department at the Technical University of Munich and the Subatomic and Plasma Physics Division at Chalmers.

Keywords: Subatomic physics, nuclear physics, radioactive beams, R^3B collaboration, GSI/FAIR, charged particle detection, phoswich, FPGA, data compression

Acknowledgements

Although most of the information about the work is contained in the following pages, this part of the thesis can surely be considered the most important. Being honest and clear should be one of the first requirements for a scientist and therefore thanking people and recognising their role in a work is a first step in this direction. So, let's start.

First of all I want to thank *my parents* for their constant support, both moral and material. Thanks to *my relatives* for all the parties every time I'm back to Italy.

Second, all the people involved in the making of the thesis need to be thanked:

- The Subatomic Physics Group at Chalmers: *Paloma Diaz Fernandez* for providing most of the code and collaboration for the work on characterizing the crystals, *Håkan T. Johansson* for providing technical support and lots of ideas for the whole thesis project (especially the new clock distribution scheme and the compression module), *Andreas Heinz* for supervising, *Thomas Nilsson* for answering questions on the history of the CALIFA detector and (together with Andreas) for allowing me to participate in the Euroschool on RIBs, *Björn Jonson* for suggestions on the presentation, *Simon Lindberg* for all the information on GIT and ROOT, and *Markus Polleryd* for sharing the office and for small talks;
- The research group at the Department of Physics at the *Technische Universität München* for providing (and developing) the FPGA firmware for all the sections of CALIFA and for the work on the DRS4 add-on board: Max Winkel, Patrick Remmels, Phillip Klenze, Felix Stark, Roman Gernhäuser;
- Scientists of the R^3B Collaboration for their work on the CALIFA detector and the resulting Technical Design Reports.

Then a big thanks to all the friends (those from Italy, Sweden and rest of the world) and especially:

- *Cristina* and *Murad* for their support, especially in the beginning of the adventure in the Swedish land;
- *Quelli del Lunedì* for all the parties and the nice time we have passed together (and also for having come to Sweden for my defence and having endured the tough Swedish food);
- *Spiazzi Time* for pictures, videos, big laughters and for having been patient here in Sweden during the defence;
- *Robert Klaar*, *David Dagson*, *August Ekman*, *Simon Ovesen* and *Ingrid Strandberg* with their friends / girlfriends / families for having given me the opportunity to obtain some knowledge on Sweden and its culture.

Other thanks go to: *theorists* of the Subatomic Physics Group in Chalmers for having clarified that I'm not suited for that life, *Sir Daniel Gazda* for the not-physics-related time we passed together and for allowing me to live one year less than planned, *all the lunches* and *funny stories*, *Prof. Andrea Candelori* for his cover letter that helped to start the Chalmers adventure, *Giovanni Maestri & family* (and *Fiorenza and marito*) for dinners and the other enjoyable events, *Beatriz Moreira* and all the rest of people around the world.

For all the people who are not mentioned but would have liked to be in this part of the thesis, please fill the dots here: .....

Contents

Chapter 1

Introduction

1.1 Physics Background

Subatomic physics has a vast range of applications: it can be used for improving medical treatments of tumours [1], for developing new materials and, of course, for understanding how nature at its smallest scales works. In this thesis we focus our attention on this last argument and the fundamental aspects of the research in this field.

Physicists can exploit several ways to study the structure of atomic nuclei and the interactions between their constituents. A group of these methods relies on the usage of *radioactive ion beams* (RIBs), which offer the possibility to study a broader range of physics cases and isotopes. In this context, a fundamental technique used for producing radioactive beams is the *projectile fragmentation technique* [2]. The key point is that, from high-energy nuclei, we obtain fragments, which preserve almost all the momentum of the incoming ions. This offers a number of advantages, especially with respect to identification and acceptance. Different types of nuclear reactions can be exploited to investigate the radioactive nuclei [3]:

- **Knockout Reactions**: Examples of such reactions are (p,2p) and (p,pn), where the first term in each reaction denotes the incoming particle, while the second one refers to the stripped particles. Usually heavy nuclei are accelerated against light target nuclei. The direct interaction takes place only if the energy of the accelerated ion is such that the associated wavelength is comparable to that of a single nucleon, which means energies above $100\,A \cdot \mathrm{MeV}$. These reactions are used to study single-particle state occupation or the excited states of a remaining fragment. Unbound nuclei can also be investigated. For the latter, it is important to detect, other than protons and neutrons, also γ-rays originating from the de-excitation of the fragments. If we go even further in energy (200–$2000\,A \cdot \mathrm{MeV}$) and use a very light target, such as hydrogen, the chances for a nucleon-nucleon interaction drops and therefore the single proton can penetrate the larger nucleus and interact with deeply-bound nucleons. Those reactions are called *quasi-free* scattering reactions and are suitable for investigations of neutron-rich unbound nuclei.

- **Fragmentation Reactions**: here impact parameter, beam energies and target consisting of several nucleons cause a removal of a significant portion of the projectile heavy nucleus. Information comes therefore from decays and de-excitation (γ-rays) of the fragments.
- **Coulomb Excitation**: in this case the interaction happens between the electromagnetic field of a target and a projectile nucleus, which will then become excited. γ-spectroscopy is used to provide (n,γ) and (p,γ) cross sections, which are needed in astrophysics studies of processes such as proton and neutron capture.

With the equipment currently being built by the Reactions with Relativistic Radioactive Beams (R^3B) collaboration, it will be possible to investigate all the mentioned scenarios, providing insights on exotic clustering at the drip-line, evolution of nuclear shells and single-particle structure far from the valley of stability.

1.1.1 R^3B Experimental Setup

The R^3B collaboration is developing an experimental setup at the accelerator facility *GSI/FAIR* in Darmstadt, Germany [4]. This setup will use radioactive relativistic beams with energies up to $1\,A \cdot \text{GeV}$ impinging on stable target material. The different detectors employed will perform full kinematics measurements on all reaction products, providing the information needed to study the reactions described in the previous section.

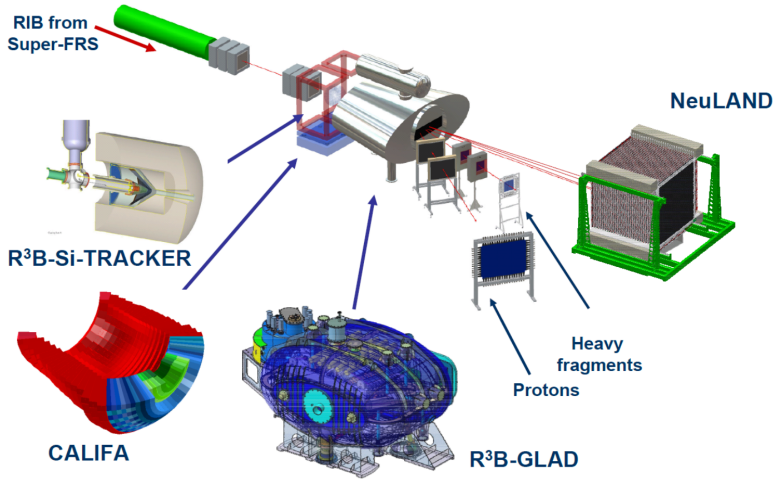


Figure 1.1: R^3B experimental setup at GSI/FAIR, Darmstadt. The incoming ion beam impinges against a target which is surrounded by the *R^3B-Si-TRACKER* and the CALIFA detector. Reaction products proceeding in the most forward direction are separated by R^3B-GLAD, a superconducting magnet, and addressed towards detectors for protons, heavy fragments and neutrons (NeuLAND). For more information, see the text. The figure is taken from Ref. [4].

In Figure 1.1 the placement of detectors is shown. The radioactive ion

beam, produced in the FRagment Separator (FRS) for GSI or in the super-FRS for FAIR, will impinge on a target placed inside the *R^3B-Si-Tracker*. This silicon-based detector will provide mainly information on trajectories of large-angle scattered particles generated by the interaction of the beam with the target. The R^3B-Si-Tracker is surrounded by the *CALorimeter for the In Flight detection of γ rays and light charged pArticles* (*CALIFA*), a barrel-shaped detector responsible for energy measurements of γ radiation and protons arising from reactions in the target. The shape of this detector, together with the high segmentation, allows to deal with the Lorentz boost characterising reaction products of the outgoing fragments. Particles which are travelling in the most forward directions will not interact with CALIFA thanks to a hole in it and pass through a large-acceptance superconducting magnet, *GLAD*, which will separate particles according to their magnetic rigidity. Charged particles such as protons and heavy fragments will be deflected towards specific detector systems, while neutrons will continue to travel straight, impinging finally on *NeuLAND* — a detector built for measuring these particles.

We now analyse more deeply the CALIFA detector, since it is the main subject of the studies presented in this thesis.

1.1.2 CALIFA

CALIFA, surrounding the target, has a central role in detecting protons and γ-rays arising from reactions in the target. The barrel shape allows CALIFA (Figure 1.2) to cover an angular range from 7° to 140° with respect to the beam axis. Because of the employment of relativistic beams, CALIFA will need to be highly segmented in order to deal with particles characterised by a high *Lorentz boost* and *Doppler shifting* of γ-rays. Around 2560 crystals will therefore be used.

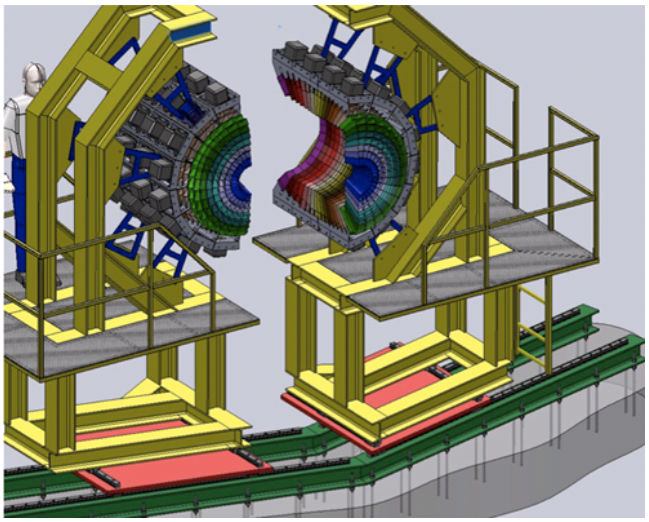


Figure 1.2: Drawing of the CALIFA detector. The barrel shape of the detector can be noticed, with the EndCap on the front. The picture is taken from Ref. [4].

The high granularity will allow to use CALIFA in different ways, such as:

- A *High resolution γ-ray spectrometer* in a low-energy range of 0.1–2 MeV (in the center of mass frame) for low multiplicity events. An energy resolution $\Delta E/E < 6\%$ is required to discriminate γ-ray cascades caused by de-excitation.

- A *High efficiency γ-ray calorimeter* for energies up to 10 MeV (in the center of mass frame) and high-multiplicity events. Measurements of total γ absorption, γ sum-energy and γ multiplicity are needed.

- A *Hybrid detector*, which can provide information on the energy of protons up to 700 MeV and on γ de-excitation of residual fragments.

Table 1.1 summarises the requirements for the final detection system.

Feature	**Constraint**
Energy range for protons	up to 700 MeV (in Lab system)
Proton/γ-ray separation	for 1–30 MeV
Energy Resolution	
γ-rays	$< 6\%\, \Delta E/E$ for 1 MeV
Stopped protons	$< 1\%\, \Delta E/E\, \frac{\sqrt{100\,\mathrm{MeV}}}{\sqrt{E}}$
Punching through protons	$< 7\%\, \Delta E/E$ at 500 MeV

Table 1.1: Requirement set for the CALIFA detection system. Data taken from Ref. [4].

1.1.2.1 CALIFA Structure

CALIFA is divided into 2 main parts, named *Barrel* and *Forward EndCap*. The Barrel will cover the angular range from 43° to 140°, while the EndCap matches the range from 7° to 43°. As detailed in Figure 1.3, the EndCap is then divided into 2 further subparts, *iPhos* (*Intrinsic PHOSwich detector*) and *CEPA* (*Califa Endcap Phoswich Array*).

Barrel The Barrel is composed of 1952 CsI(Ti) crystals. These crystals will be shaped in order to be able to stop protons with energies up to 320 MeV, enabling an energy resolution of less than 2%. Regarding γ-rays, these crystals allow for an energy resolution of 5–6% at 1 MeV.

Forward EndCap This part consists of 608 crystals. As written above, this section is divided into iPhos and CEPA. These two subparts are characterized by the use of 2 different crystal materials: CsI(Ti) is employed for the iPhos part, while so-called phoswich crystals of $LaBr_3/LaCl_3$ are used for the CEPA section.

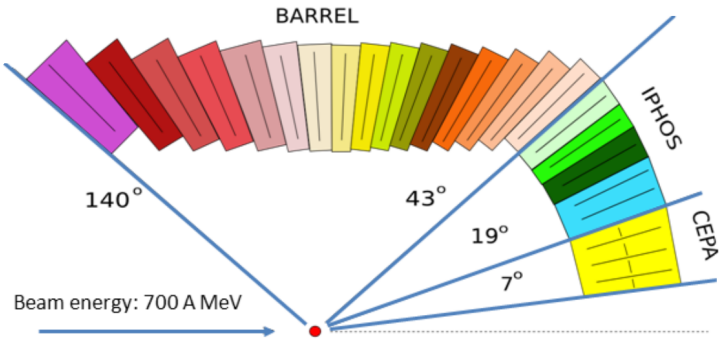


Figure 1.3: Partition of CALIFA. The *Barrel* and *iPhos* parts are composed by CsI(Ti) crystals, while for the *CEPA* section $LaBr_3/LaCl_3$*phoswich* crystals are used. *iPhos* and *CEPA* together form the *Forward EndCap* part. The figure is taken from Ref. [4].

1.1.2.2 iPhos

Consisting of 512 CsI(Ti) crystals, the iPhos part covers the polar angular range from 19° to 43°. A crystal length of 22 cm permits an energy resolution of about 7% for protons at 500 MeV (see Figure 1.4).

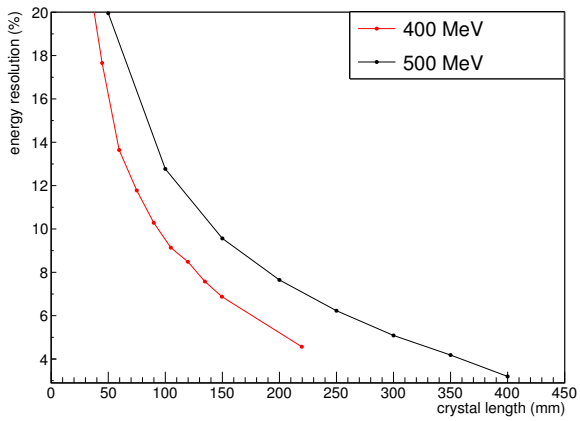


Figure 1.4: Proton energy resolution as a function of the crystal length in the case of CsI(Ti) detectors. Proton energies of 400 MeV and 500 MeV are considered. The picture is taken from Ref. [4].

The reason for the use of CsI(Ti) is that this material is characterised by two different scintillation processes, each of them having a different decay time. Using the *Reconstructive Particle IDentification* (RPID) algorithm (see Ref. [5]) it is possible to extract the amplitudes of the fast and the slow components. From this information we can distinguish between different particles types: Figure 1.5 shows how protons and γ-rays can be separated down to 1 MeV by plotting the amplitudes of the slow versus the fast components.

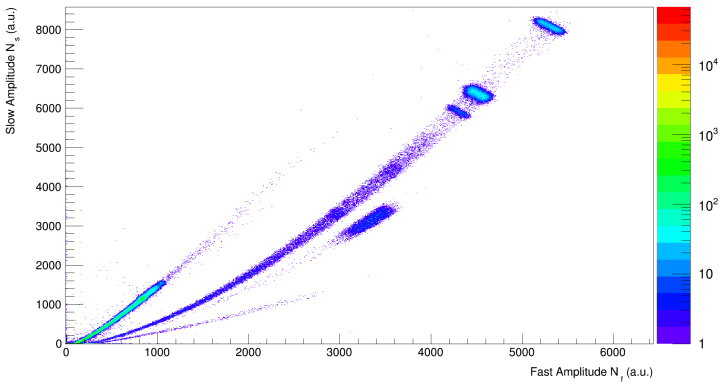


Figure 1.5: Plot of amplitudes of the slow component N_s versus fast component N_f. The amplitudes were extracted using an RPID algorithm on signals from CsI(Ti) crystals of the iPhos part. The picture is taken from Ref. [4].

1.1.2.3 CEPA

CEPA is formed of 96 crystals optically isolated from each other and organised in 8 sectors covering the angular range 7–19°. As depicted in Figure 1.6, each sector is made of 12 crystals wrapped in a can of aluminium, 0.2 mm thick. As Figure 1.6 also suggests, each crystal is a stack of two scintillator materials optically coupled: $LaBr_3$ on the inside (red) and $LaCl_3$ on the outside (green). A single photo-multiplier-tube is used for both crystals, so only one signal is read out. This entire approach is known as *phoswich*. In Figure 1.7 the pulses from the two materials are plotted together with the common read out signal.

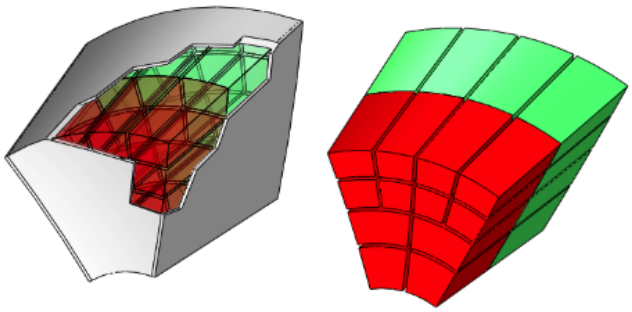


Figure 1.6: Sector for the CEPA section. Each sector is made up of 12 phoswich crystals. Red and green stand for $LaBr_3$ and $LaCl_3$ respectively [4].

The main reasons for using these materials are

- Fast decay times: $LaBr_3$ and $LaCl_3$ pulses have decay constants of 16 ns and 28 ns respectively. While these small values implicate the ability for high counting rates, the mismatch is also important because it allows to disentangle the two contributions through the *pulse shape analysis*.

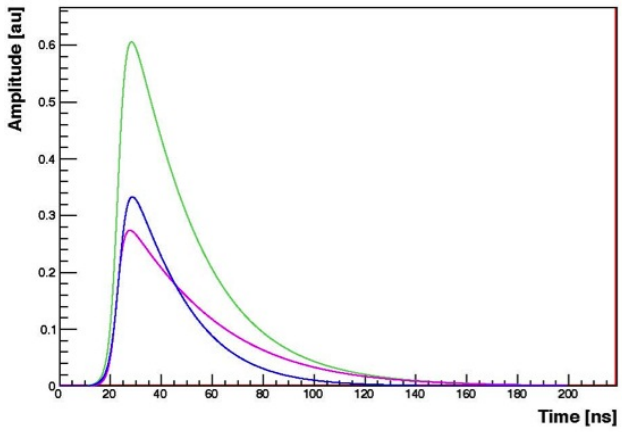


Figure 1.7: Pulses generated by phoswich crystals. The green line represents the total pulse, which is read out by the detector. The pulse from the $LaBr_3$ is depicted in blue, while that from the $LaCl_3$ is plotted in purple [6].

- Fast rising times: both crystals pulses show rising constants of 5 ns. Together with the fast decay times, these values are necessary for having a high counting rate.
- High time resolution: although the actual value depends on the size of the crystal, time resolutions of a few hundreds of ps have been observed (see Ref. [7] and chapter 4);
- High stopping power for light particles: this implies that it is possible to stop protons at high energy inside the crystal through electromagnetic interaction. This allows a better measurement of the particle energy in a broader energy range.
- Good energy resolution.

There are various motives for exploiting the phowshich approach in the detector:

- If we use CsI(Ti) for high energy protons ($E > 200$ MeV), we would need longer crystals to achieve the required energy resolution (see Figure 1.4). However longer crystals mean higher probability for protons to undergo nuclear reactions in the detector with the production of neutral particles (neutrons, pions), whose energy would not be completely measured, making it impossible to measure the (total) particle energy. In the case of phoswich crystals, exploiting the information about the energy deposited in the two different materials, the total energy of a particle can be extracted without the need of stopping it in the detector. Therefore the crystals can be shorter than those used in the iPhos branch, thanks also to the high stopping power of the $LaBr_3$ and the $LaCl_3$. This approach still ensures the required energy resolution, given also the good resolution properties of the employed materials.

- Thanks to the high stopping power, half of the γ-rays at 20 MeV interact by pair production within the first 5 cm of $LaBr_3$. This means that most of γ-rays will deposit all their energy in a small portion of space, therefore avoiding further Compton scattering.

- After the extraction of the two decay components from the single read-out signal, it is possible to measure the energy of particles and to (hopefully) distinguish them using *Pulse Shape Analysis* (PSA), a similar approach to the RPID described in Section 1.1.2.2.

This thesis work will focus on CEPA and the electronics used for sampling and processing signals generated by phoswich crystals. Before showing the content of this work, the electronic equipment planned to exploit the characteristics of the $LaBr_3/LaCl_3$ crystals will be analysed.

1.2 Electronics for CALIFA

1.2.1 Readout System

The current system planned for sampling, reading and storing signals from the detector will be based on the Data AcQuisition system (DAQ) developed at GSI, the so-called Multi-Branch System (MBS) [4]. This system however will be replaced in the future by the NUSTAR Data AcQuisition system (NDAQ) [8], which is currently under development and it will be the main system used in the new FAIR facility. Figure 1.8 provides a functional description of the DAQ system, while Figure 1.9 shows the setup that has been built at the laboratory of the Subatomic Physics Group in Chalmers.

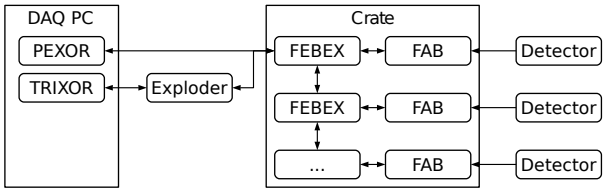


Figure 1.8: Diagram of the Data AcQuisition system (DAQ) [4]. Signals from the detectors arrive at the Febex Add-on Board (FAB), which provides connections, buffers and coupling. The FAB is connected to the Febex, where signals are sampled and processed. All the Febex cards sit in a Febex crate, thanks to which triggers and data can be sent to the Exploder+TRIXOR and to the PEXOR cards, respectively.

First of all, signals generated by detectors reach the *Febex* board through an add-on board[1], which can contain just connectors or preamplifier stages and other electronics. On the Febex, the signals are sampled and digitized by

[1] In Figure 1.8 it is labelled as FAB (*Febex Add-on Board*)

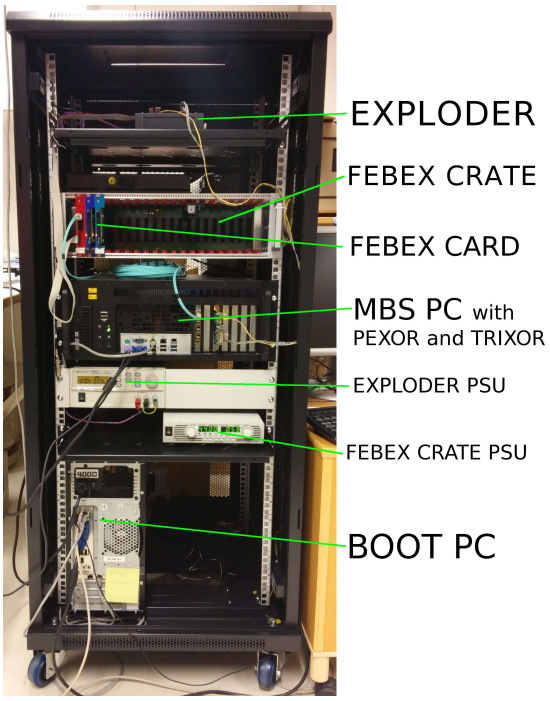


Figure 1.9: Setup built in the laboratory at Chalmers.

ADCs[2] and then sent to an FPGA (see section 5.3 for a description), where additional signal processing can be performed.

All Febex cards sit in a crate which provides power and a common bus for basic communication between the cards. The crate is connected to an *Exploder* module, which is responsible for the management and generation of the trigger. Based on the configuration used, the Exploder can receive (and send) trigger signals from (and to) the Febex cards and from (to) other systems in the experimental setup.

A third component is the DAQ computer (or MBS computer). This diskless computer, which needs assistance from an external computer for booting and interfacing with external networks, contains the PEXOR and TRIXOR cards. The former is responsible for the readout of the data from the Febex cards via an optical cable. The TRIXOR card instead is connected to the Exploder and receives the trigger signal. Another card called PEXARIA, which is not included in the system at the Chalmers laboratory, manages the timestamping of the events. Communications between all the components described relies on a protocol called *GOSIP*. For more information about the devices described above, see section 7.3.

[2]In this thesis, with *Analog-to-Digital Converter* (ADC) we mean an electronic circuit which, according to a given sampling frequency, transforms an analog input signal into an output digital signal. The input signal is *sampled* and *quantized*. While the sampling depends on the *sampling frequency*, the quantization is related to the number of bits which characterises the ADC: the maximum accepted input voltage range is divided into $2^{\text{number of bits}}$ steps and the value of each input sample is then rounded to the closest (upper or lower depending on the ADC design) step.

1.2.1.1 Febex

In Figure 1.10 we can see a picture of the Febex card. The main features of this electronic board developed at GSI are the 2 ADCs and an FPGA. Digitization of signals from the detectors is performed by the ADCs (AD9252), each of them with 8 input channels, at a maximum sampling frequency of 50 MS/s[3] and 14 bits of resolution. The handling of the resulting data is accomplished by the FPGA (Lattice ECP3 150). For each section of CALIFA (Barrel, iPhos, CEPA) a different firmware will be designed, in order to match detector requirements and signal characteristics. More detailed information about the Febex board can be found in section 7.3.1.

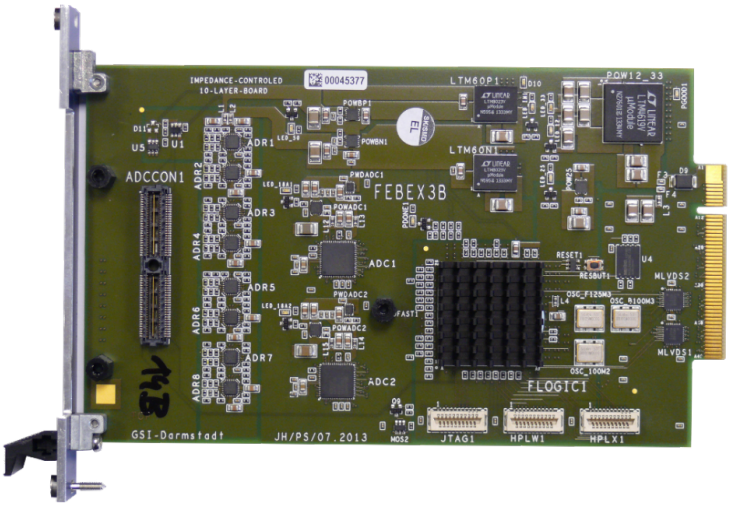


Figure 1.10: Febex3B card used in the DAQ system [4].

1.2.2 Electronics for CEPA

1.2.2.1 Switched Capacitor Array - DRS4

Since signals originating from phoswich crystals are pretty fast, a 50 MHz sampling frequency is not sufficient in order to capture the pulse shapes. For the CEPA part therefore a faster sampler is needed. This problem is solved by using a *Switched Capacitor Array* (commercially known as DRS4 [9]) mounted on an add-on board. In this piece of integrated electronics, an input signal is fed into a sequence of 1024 capacitors. Along this sequence a control signal (domino wave) enables one capacitor at a time: therefore the input signal is fed to the only activated capacitor at a time. A simplified schematic of a single DRS4 cell can be seen in Figure 6.1. The faster the domino wave moves along the sequence, the higher is the sampling frequency. The DRS4 features 9 input channels with a maximum sampling frequency of 5 GS/s. In our case the

[3]The sampling frequency of an electronic device is often reported as *number of samples per second* S/s. It can be also found written as Sa/s. The prefixes *mega* M or *giga* G are usually used.

sampling frequency is set to 1 GS/s. Having 1024 cells per channel means that around 1 µs of the detector trace can be stored in the capacitor array. The sampled data can be read out by the FPGA at a maximum frequency of 33 MHz. This mismatch in the frequencies, and the fact that the DRS cannot sample while the readout is being performed, implies that a continuous processing of the high-resolution phoswich signals is not possible.

1.2.2.2 CEPA Firmware

In Figure 1.11 the firmware for the CEPA part of the Forward EndCap is shown. Each single crystal will be connected to both ADC1 and the DRS4. Two main parts are easily visible:

- The *Real Time Branch* (purple): the signal from the detector is continuously sampled by ADC1 (at 50 MHz) and the resulting data are used for the trigger generation (blue) and for some basic energy computation (green);
- The *Fast Sampling Branch* (red): once a trigger is generated the *Readout* block starts to read out the DRS4 content. After the digitization, the data will be processed in order to extract the $LaBr_3$ and $LaCl_3$ components. The results are then stored and sent to the MBS computer.

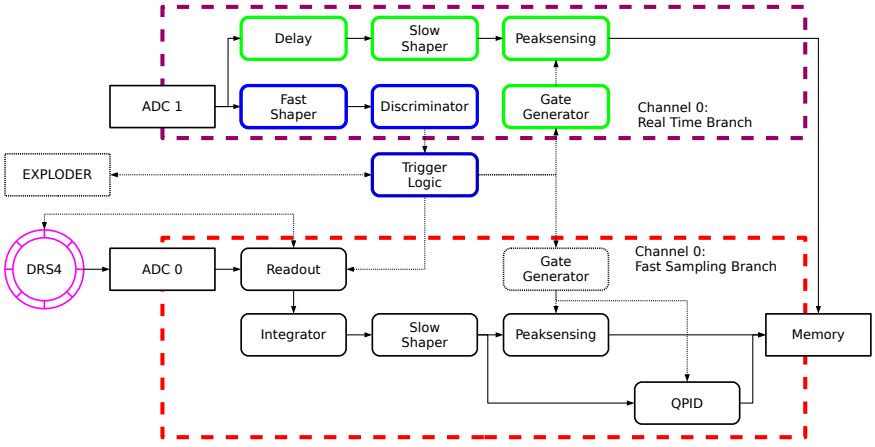


Figure 1.11: Schematic representation of the firmware for the CEPA detectors [4]. Blocks responsible for generating the trigger are highlighted in blue, while those recording basic data in case of dead-time are marked in green. Inputs of ADC1 and DRS4 come from detectors. The DRS4 is controlled by an internal module, which will be described in chapter 6. In the red-coloured block, the *pulse shape analysis* of the detector signals is performed.

Part I

CEPA4 Characterization

Chapter 2

Introduction

2.1 CEPA4 Prototype

CEPA4 is a prototype, developed by Saint-Gobain, which was used by the CALIFA working group to investigate the properties of the $LaBr_3/LaCl_3$ phoswich crystals and to design the readout system [10]. The device is visible in Figure 3.4. As shown in Figure 2.1 this detector is made up of 4 phoswich-crystals arranged in a 2×2 configuration. Each segment is composed of a 4 cm $LaBr_3$ crystal optically coupled to a 6 cm long $LaCl_3$ crystal. Their front cross-section is about 2.7 cm $\times$ 2.7 cm. A photomultiplier tube (PMT) is attached to the rear side of the $LaCl_3$ crystal.

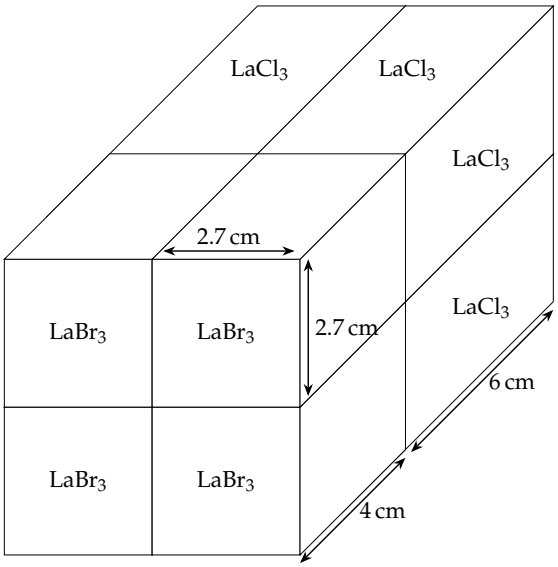


Figure 2.1: Schematic of the CEPA4 prototype. The four $LaBr_3/LaCl_3$-phoswich crystals are optically isolated between each other. Photomultiplier tubes are placed on the back of the $LaCl_3$ crystals.

2.2 Readout Hardware

2.2.1 Photomultiplier

Although in Ref. [10] each crystal is described as coupled to a Hamamatsu R5380 PMT, in the current situation the Hamamatsu R7600U-200 [11] PMT is employed, as also required by the CALIFA Technical Design Report [4]. The Hamamatsu R5380, compared to the R7600U-200, has a 1000 times lower gain. As explained in section 3.1, the characterisation of the CEPA4 crystals was carried out also to test if the detection system using these new PMTs is appropriate to satisfy the requirements set in the CALIFA TDR [4].

2.2.2 Digitizer

The electronics designed for the actual CALIFA detector is based on the combination of a Febex3B board, for the signal digitization and processing, with an add-on board featuring a DRS4 chip. The latter is a switched capacitor array and is responsible for the fast sampling of the signal from the phoswich detector (for more information see section 1.2.2.1). While the input voltage scale range of $1\,V_{pp}$, and the sampling frequency of 1 GHz are due to the DRS4 [9], the resolution of 14 bits is given by the ADC AD9252 [12] mounted on the Febex3B.

The add-on board with the DRS4 is not available yet, therefore tests were carried out using two CAEN digitizers and a LeCroy oscilloscope. One of the CAEN modules is the DT5751: it features an input voltage range of $1\,V_{pp}$, a sampling frequency of 1 GHz and a data resolution of 10 bits [13]. The second module is a CAEN DT5730 which, compared to the DT5751, has a larger input voltage range of $2\,V_{pp}$, a better resolution of 14 bits, but a smaller sampling frequency of 500 MHz [14]. The LeCroy oscilloscope is a 7100A model, which can sample an input voltage range of $5\,V_{rms}$ (at $50\,\Omega$) at a frequency up to 20 GHz with a resolution of 8 bits. Table 2.1 summarizes the information on the three systems used in the tests, together with the Febex3B/DRS4 combination.

	Input Voltage Range (V)	Sampling Frequency (GHz)	Resolution (bits)
DT5751	1	1	10
DT5730	0.5 or 2	0.5	14
LeCroy 7100A	$5\,V_{rms}$ at $50\,\Omega$	up to 20	8
Febex3B/DRS4	1	1	14

Table 2.1: Main characteristics of the CAEN digitizers, DT5751 and DT5730, the LeCroy 7100A and the Febex3B/DRS4 system.

2.2.2.1 CAEN Software

In order to configure the two CAEN digitizers, the manufacturing company itself provides software through which we are able to configure some of the

device settings, such as the acquisition mode, internal thresholds, recording length, etc. We mainly used the so-called *digiTES* software, which comes with an editable text configuration file. Another program, the DPP-PSD control software presents, on one hand optimised routines for energy spectrum calculation, threshold settings, etc., on the other it does not offer the same flexibility given by *digiTes*. Therefore, because of its stability, it was employed mainly for checking the correct functioning of the devices. Both applications were used in a Linux environment.

2.2.3 Additional Devices

In the following sections, several experimental setups will be presented. Although the main elements have already been described above, other electronic devices and components have been involved:

- **ORTEC CF4000**: constant fraction discriminator with 4 independent channels. This device releases a square pulse every time the input signal amplitude exceeds a given threshold. The threshold and both the time width and amplitude of the output signal are configurable through screwdriver adjustments.
- **ORTEC CO4020**: it allows to implement logic functions with 4 inputs. 4 independent sets of input channels are available. We used this module to distinguish the coincidence among four plastic crystals.
- **DELAY**: simple delay cables are used;
- **SPLITTER**: for splitting the signal, a cascade of resistive dividers is used. The number of stages depends on the desired attenuation. A single divider, as shown in Figure 2.2, is based on a *star* configuration where three equal resistors are connected together through one common pole. The remaining poles are connected to one cable each. The input signal arrives on one cable (V_0) and is then split between the other two cables (V_1 and V_2). The choice of the correct value for the resistors ($Z_0/3 = 16.6\,\Omega$) guarantees an input impedance Z_0 of $50\,\Omega$ for each port, with an equal splitting coefficient and impedance matching [15], introducing no reflections together with $50\,\Omega$ cables.

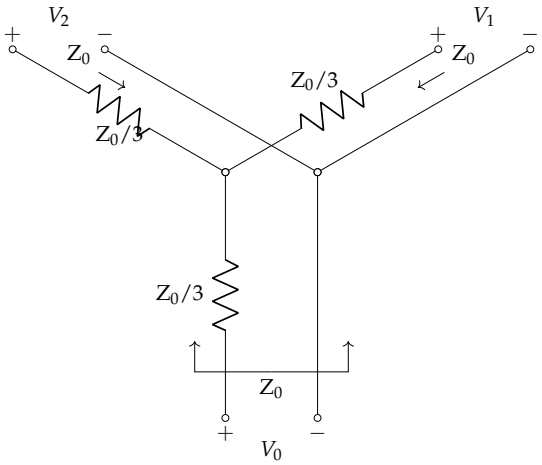


Figure 2.2: Resistive divider: in this configuration the input signal is split equally between the other two ports.

Chapter 3

Signal Readout

3.1 Motivation

The goal of the study we are going to illustrate is to check the feasibility of the detection of both γ-rays and protons using the same electronics chain (PMT+HV+digitizer). According to Figure 3.1 [4], crystals of the CEPA part are expected to measure at most energy depositions, for protons, of around 200 MeV in the $LaBr_3$ part of the phoswich and of around 280 MeV in the total phoswich crystal $LaBr_3$+$LaCl_3$.

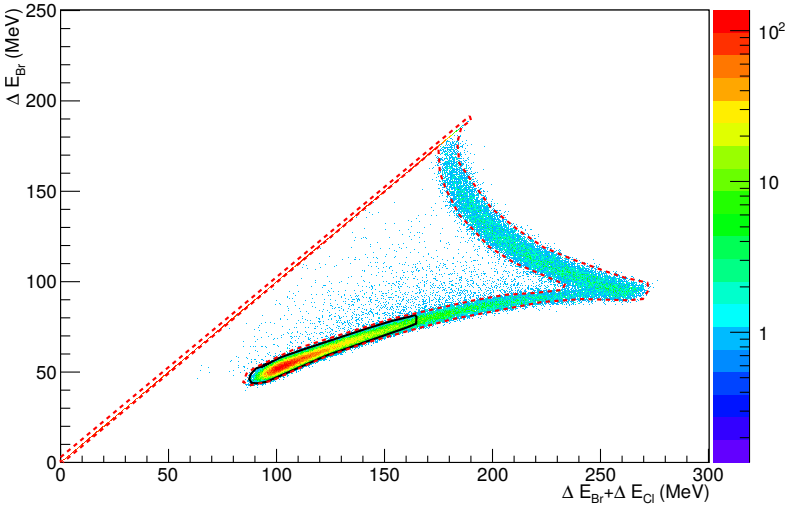


Figure 3.1: Plot of the energy deposition, by simulated protons at 1000 MeV, in the $LaBr_3$ versus the one in the whole phoswich crystal. Notice that the maximum energy loss in the $LaBr_3$ is around 200 MeV, while the maximum energy deposition before punchthrough is around 280 MeV. Only electromagnetic interactions are taken into account. The figure is taken from Ref. [4].

Based on Table 1.1 [4], the expected γ-ray energy resolution for 1 MeV γ-rays is smaller than 6%. In order to keep this resolution for proton energies up to

280 MeV we would need a dynamic range in the digitizer equal to

$$\frac{20\,\text{keV}}{280\,\text{MeV}} = \frac{1}{14000} \approx 2^{-13.8}. \tag{3.1}$$

This result means that a digitizer with a resolution of *at least* 14 bits is required in order to satisfy the 6% requirement on the resolution. Note that the calculation does not take into account any noise, which would require a higher number of bits.

From Table 3.1 we can see that the light yield of $LaBr_3$ is larger than that of $LaCl_3$. This means that, by employing the same photomultiplier voltage, for

Material	Light Yield (photons/keV)	Decay Time (ns)
$LaBr_3$	63	16
$LaCl_3$	49	28

Table 3.1: $LaBr_3$ and $LaCl_3$ light yield and decay time.

the same energy deposited in the crystals, the pulse from the $LaBr_3$ is larger than that from the $LaCl_3$. Since the main problem we encountered concerns the (large) amplitude of the output signal, our analysis focuses on the $LaBr_3$ crystal. Because of the low energy of the particles used in the studies, we could not investigate the punch-through in the $LaBr_3$ and the subsequent energy deposition in the $LaCl_3$. This represents an important limitation to our tests. Indeed *if* the total energy deposited is close to the entire phoswich punch-through energy, the produced pulse would be larger than the largest pulse achievable in the $LaBr_3$ alone (see results of calculations reported in Table 3.2).

Energy Deposited (MeV)	Deposited $LaBr_3$ (MeV)	Deposited $LaCl_3$ (MeV)	# photons
200	≈ 200	≈ 0	$\approx 12.6 \times 10^6$
220	≈ 120	≈ 100	$\approx 12.46 \times 10^6$
270	≈ 100	≈ 170	$\approx 14.63 \times 10^6$

Table 3.2: Energy deposition and photons generated for protons with 200 MeV (close to punch through of $LaBr_3$), 220 MeV and 270 MeV (both representing a punch through of $LaBr_3$ and stop in the $LaCl_3$). The energy values have been taken from Figure 3.16 of the TDR [4].

In the end, it will be clear that detecting γ-rays and proton using the same electronic setup is not feasible and that a change of the photomultiplier tubes has to be considered.

3.2 Experimental Setups

Two different schemes were devised to trigger on signals from the CEPA4 crystals, while using the same configuration for PMT, High Voltage (HV) and digitizer. This approach is needed because of the nature of the two different probes exploited in this characterisation: γ-rays (with a maximum energy of 1.3 MeV) and cosmic muons (up to a few tens of MeV).

While γ-rays are almost completely stopped in a few cm of $LaBr_3$, muons are able to pass through the whole crystal.Therefore, when dealing with muons, an external trigger system was used: two plastic scintillators were placed above and two below the crystal and a coincidence of signals from all four was used as trigger (for more information see section 3.2.1). This was done to

- Only consider muons which crossed just one of the crystals, the $LaBr_3$ under investigation;
- Use only muons which follow a similar path in the crystal: because of the distance between the plastic scintillators and the small common area covered by them, all muons responsible for the coincidence trigger intersect the crystals with an angle close to 0° with respect the vertical axis.

For γ-rays this external trigger is unnecessary (and useless as well). While charged particles, such as muons, lose energy continuously in a medium until they are stopped, γ-rays undergo (macroscopically) discrete interactions. Therefore they can not trigger all the scintillators in the same event. To cope with this behaviour, the *self-triggering* feature given by the digitizers was used (see section 3.2.2 for more details).

3.2.1 Setup for Muon Detection

As previously discussed, in order to detect muons we arranged four plastic scintillators in such a way that the part of CEPA4 under investigation was surrounded from above and below. The layout of the setup is shown in Figure 3.2, while a photograph of it can be seen in Figure 3.4. Muons that hit the two upper and two lower plastic scintillators generate signals which are first discriminated against a threshold (ORTEC CF4000) set in order to avoid recording electronic noise. The output square pulses given by the discriminator are sent to a module (ORTEC CO4020) that checks for coincidences among the signals from all the plastic detectors. Once a coincidence of all four detectors occurs, a trigger reaches the digitizer, allowing it to record the pulses caused by the muon in the upper and lower $LaBr_3$ crystals. The samples are then stored as text files in the computer connected to the digitizer. Signals from the phoswich crystals can be attenuated, before reaching the digitizer, using resistive dividers.

3.2.2 Setup for γ-rays Detection

The setup for measuring γ-rays is based on a self-triggering approach. To accomplish our purpose we exploited a feature common to both CAEN digitiz-

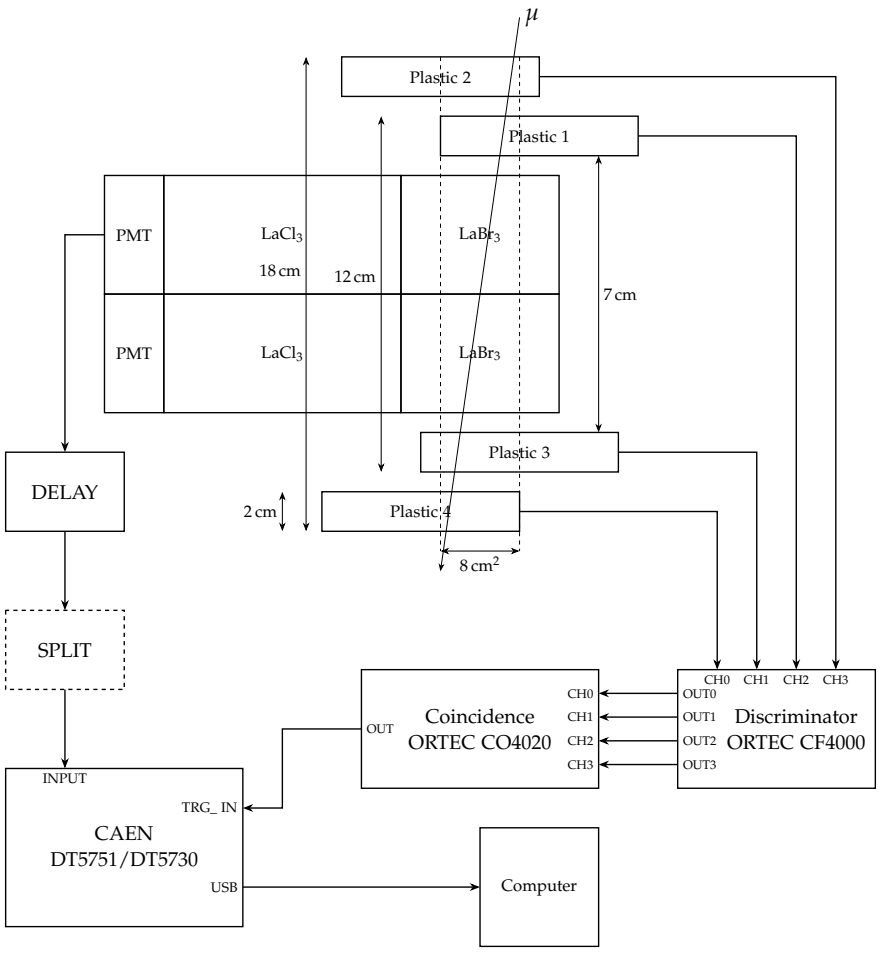


Figure 3.2: Setup schematic used for measuring signals generated by muons. The electronic chain treating signals generated by the fours plastic scintillators is responsible for the trigger and is visible in the bottom-right side. Signals from the CEPA4 crystals (left side) reach the digitizer after being delayed and split, if necessary. For more details see text.

ers: analysing the signal we feed to them, they are able to generate a trigger based on the amplitude of the input signal, with no need for an external trigger. Among all the parameters the user can set in the configuration file of the driving program (see section 2.2.2.1), the most important are the *trigger threshold* (given in ADC channels), the *pre-trigger* value (in samples) and the recording length (in samples).

Figure 3.3 shows in detail the electronics chain used for measuring γ-rays. The γ-source is placed in front of the tested $LaBr_3$ crystal and held there by tape: we mainly used a ^{60}Co-source. Due to the energies of the two dominating γ-rays (1.17 and 1.33 MeV) most of the them are absorbed by the crystal right in front of the source.

Following the schematic, when a γ-ray hits the upper crystal, the electric

pulse generated by the PMT first can be split, according to the chosen attenuation, by a cascade of resistive dividers. Then it reaches the input channel of the digitizer. As explained above, the trigger is generated internally by the digitizer, which then sends the sampled data to the computer, where they are stored as text files.

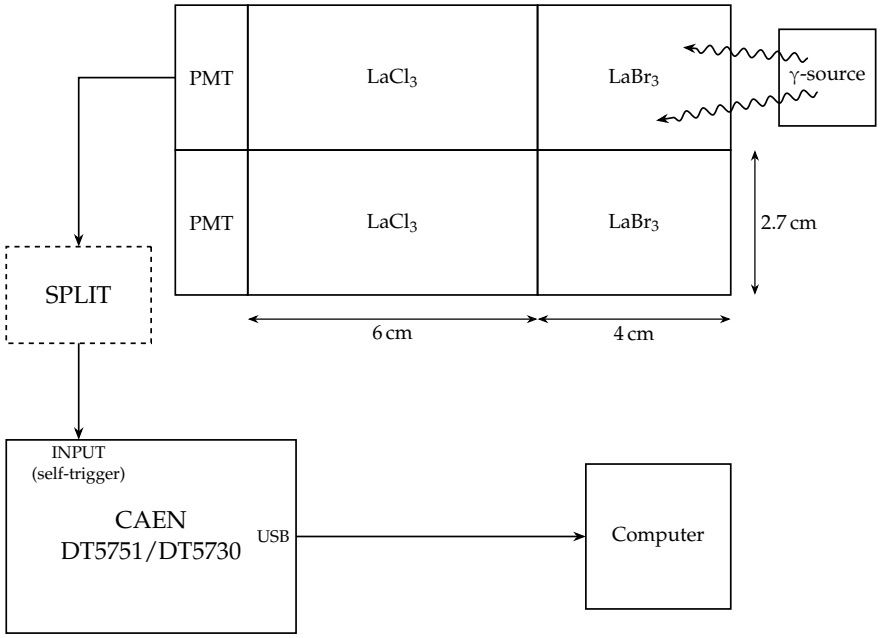
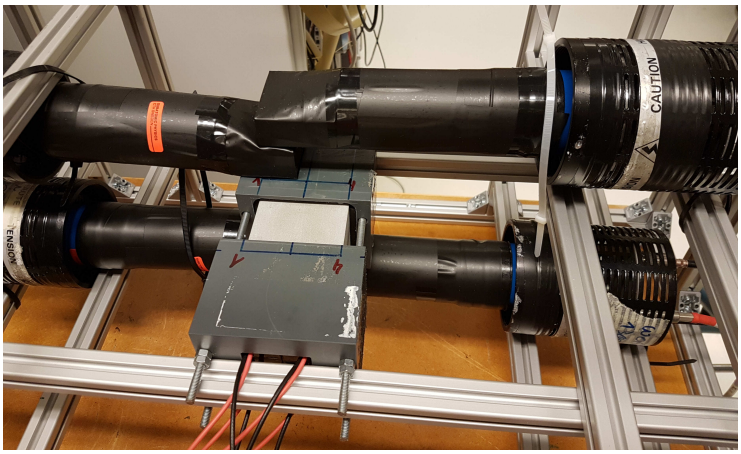
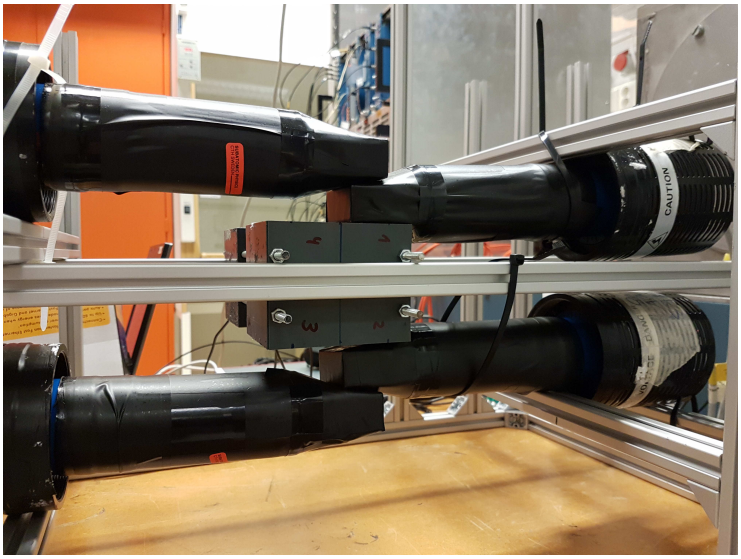


Figure 3.3: Setup schematic used for measuring signals generated by γ-rays. Different from the muon setup, there is no electronic chain for the trigger generation needed. Signals from the phoswich crystals are sent to the digitizer and internal *self-triggering* is operated. For more details see text.



(a) Top view.



(b) Front view.

Figure 3.4: Setup for muon detection. The four black cylindrical devices are the plastic scintillators (together with their PMTs) used for triggering. Phoswich crystals and the attached PMTs are enclosed in the grey box in the center.

3.3 Data Analysis

As explained in section 3.1, in this study we want to characterise the energy spectrum of incoming particles that can be detected with the CEPA4 and the electronics previously described. In order to measure the energy deposited by each particle, we have defined three different methods:

- **Area1**: the energy is given by the integral of the entire recorded wave;
- **Max**: the maximum amplitude of the wave is considered as energy deposited;
- **Area2**: the energy is calculated as integral of the wave inside a fixed-size time window centred around the peak, usually from 50 samples before the peak to 100 samples after.

An important aspect when calculating an integral of a signal is the level of zero amplitude, also called *baseline*. Because of the presence of a small bias and noise in signals from PMTs, the value of the integral can be heavily influenced even by small shifts, especially if the integral is carried out over a large amount of samples. To cope with this issue, the baseline is calculated as the average of the first 20 amplitude samples, which contain just noise. The lack of pulses in these initial samples is ensured by the fact that the digitizer records a certain amount of samples before the triggered signal: this interval is called *pre-trigger* and its value can be set in a configuration file (see section 2.2.2.1).

All the studies performed were done using signals from the crystal labelled as 3, in the CEPA4 prototype. This choice was made accordingly to the gain trend while varying the voltage applied to the PMTs. In Figure 3.5 the peak mean value of the muon energy spectrum, calculated using *Area1*, is plotted versus the PMT voltage: it can be noticed that crystal 2 and 3 have a behaviour which averages among all. From the plot, it can be notice the exponential trend of the average pulse energy with the high voltage applied. This behaviour can be confirmed by the exponential trend of the PMTs gain versus applied high voltage, shown in Ref. [11].

3.3.1 Noise study

Before showing results obtained from the measurements, we try to characterise the noise given by the detectors, PMTs and electronics in order to understand the different contributions and the influence it can have in the final setup.

Figure 3.6 shows the mean value of the noise distribution in relation to the high voltage (HV) applied to the PMTs. The uncertainty reported is just the standard deviation of the noise distribution. This was done for voltages between 450–600 V in steps of 25 V.The noise level for each wave is calculated using the first 20 samples of the trace: since the trigger comes much later (at least 100 ns later), the choice of this time range ensures that the signal is flat and it does not describe detection of any particle. The noise amplitude is therefore calculated as the difference between the minimum and the maximum value of the 20 samples.

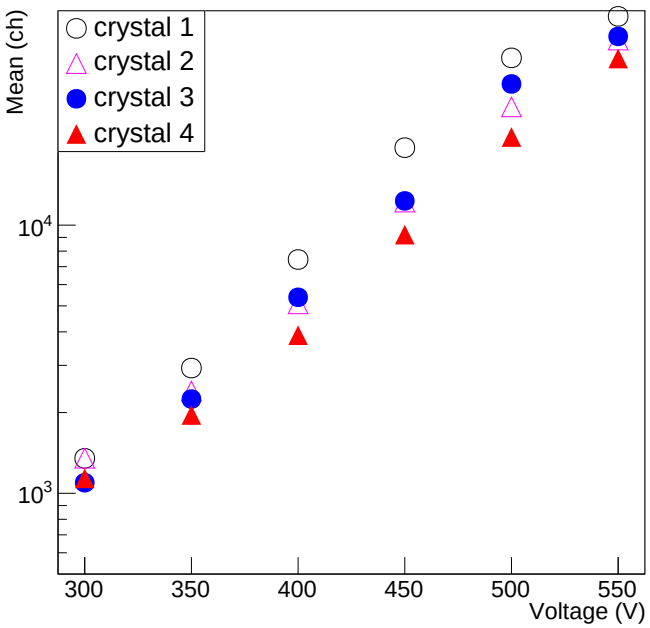


Figure 3.5: Mean value of the muon peak calculated using *Area1* versus the voltage applied to the PMT.

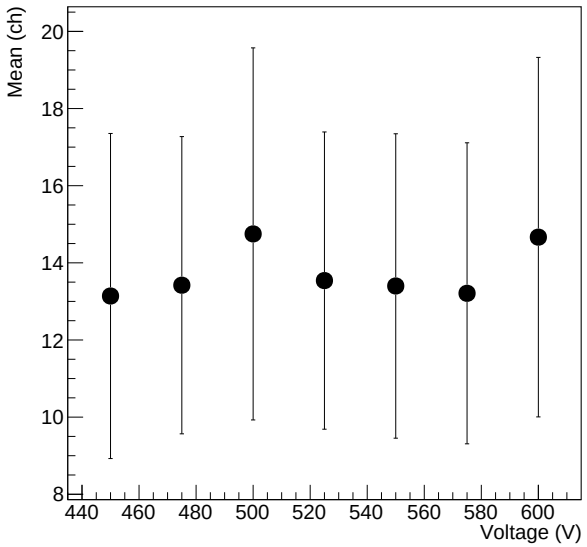


Figure 3.6: Mean value of the noise distribution versus voltage. The uncertainties are the standard deviations of the noise distribution.

We studied the noise distribution in other operational conditions, such as without high voltage applied, network or other unnecessary cables connected: to be completely sure of not being influenced by any background radiation and electromagnetic interference, we carried out the noise measurements in three different locations as well. The results are shown in Table 3.3: the data tagged as *lab* and *office* were obtained from measurements done in different locations but in the same building, while those tagged as *origo* come from measurements carried out in a different building. As can be easily noticed, numbers show the same noise behaviour within error bars.

Voltage (V)	Input Type	Location	Mean (LSB)	Std. Dev. (LSB)
600	detector	lab	14.66	4.66
0	detector	lab	14.97	4.87
0	50 Ω	lab	9.23	1.83
0	50 Ω	office	9.27	1.94
0	open	office	9.28	1.92
0	50 Ω	origo	9.14	1.84
0	open	origo	9.25	1.84

Table 3.3: Mean and standard deviation of the noise distribution with and without HV. The measurements with no voltage were done without network or other unnecessary cables connected to the devices in three different locations.

3.4 Results

In the following sections, the detector signal is passed through several attenuation stages in order to reduce its amplitude. A single stage consists of one resistive divider (see section 2.2.3) and therefore the signal amplitude is split in half at each stage. In the following plots the attenuation factor is reported as *number of splits*: this means that a splitting number x corresponds to an attenuation factor of $1/2^x$.

To fit the peaks in the γ-rays energy spectra and recover information about mean value and variance, we used a Gaussian function, to represent the peak, summed to a linear function, which takes the background into account.

3.4.1 Maximum Energy Range

In this section we check the maximum energy range covered by using the $2\,V_{pp}$ input available in the DT5730 for different attenuation factors. The PMT bias voltage was set to $-600\,V$ for all the cases analysed.

Figure 3.7 shows the resulting range depending on the number of splits. For calculating the maximum energy value we used the calibration information obtained by using ^{60}Co as γ-source. The energy of each pulse was initially calculated using the three methods explained in section 3.3. However a comparison of the results from these three methods shows that the ranges given

by *Max* are the smallest ones: assuming a conservative approach we therefore plotted these values in Figure 3.7.

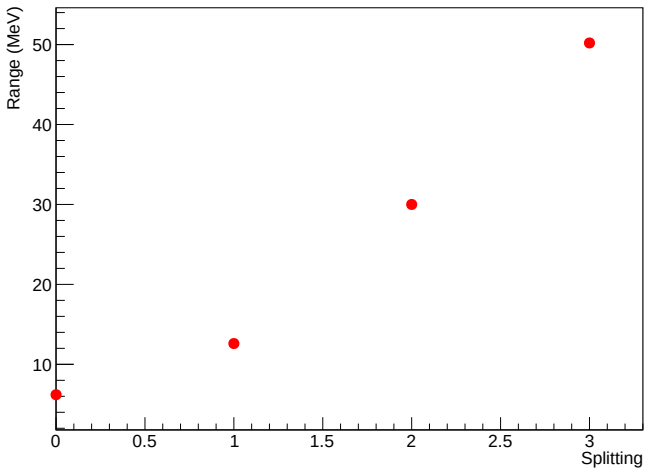


Figure 3.7: Maximum energy range achievable in a $2\,V_{pp}$ input voltage range versus number of times the signal was split. All the measurements were carried out using a PMT bias voltage of $-600\,V$ and a ^{60}Co γ-source.

Muon measurements with the same HV setting but with an attenuation factor of 8 give the following maximum energies that can be digitised:

- *Area1*: 72 MeV;
- *Max*: 50 MeV;
- *Area2*: 72 MeV.

These values are provided with an error smaller than 10 MeV.

In Figure 3.8 the three calibrated energy spectra for muons are plotted. During the acquisitions we paid attention to have the peak amplitude of the largest pulses inside the limit of the digitizer input range: in this way the range obtained from the pulse area gives the maximum energy limit. This value results to be 1.44 times larger than the range obtained using the amplitude approach. Further investigations revealed that the traces from muons at higher energies than the 20 MeV peak are distorted. Reasons for this distortion are discussed in section 3.4.3. A comparison between Figure 3.9 and 3.10 show the effects of this distortion on the results previously described. In Figure 3.10 there is no distortion, since the high voltage applied is quite low ($-450\,V$), and the relation between the integral and the maximum value of the pulse is linear. This linearity instead is broken if the high voltage is raised to $-600\,V$: starting from values of *Max* around 900 and up, it is clear that the area of a pulse is increasing faster than the peak value does. Therefore this results in larger values given by the area methods than the expected ones.

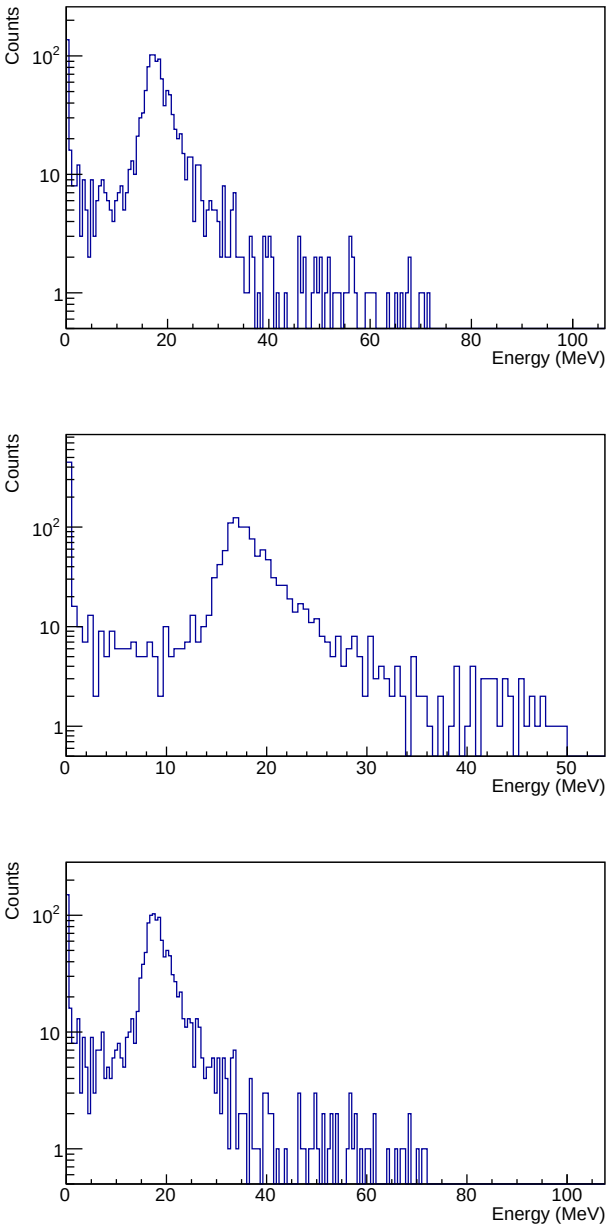


Figure 3.8: Calibrated energy spectra for muons using a PMT HV of $-600\,\mathrm{V}$ and an attenuation factor of 8. The three plots represent energy ranges obtained using *Area1* (top), *Max* (middle) and *Area2* (bottom).

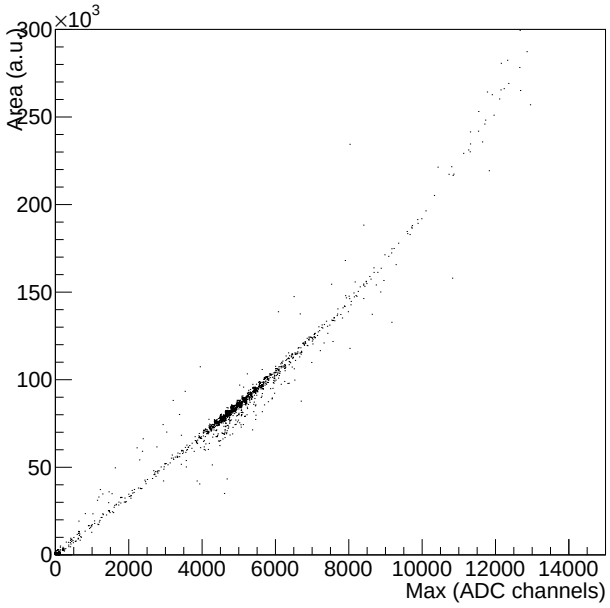


Figure 3.9: Values of the area, obtained using the Area1 method, are plotted against the corresponding value of the pulse peak (Max). Data come from measurements using a HV of $-600\,\mathrm{V}$ and an attenuation factor of 8 applied to the output signal.

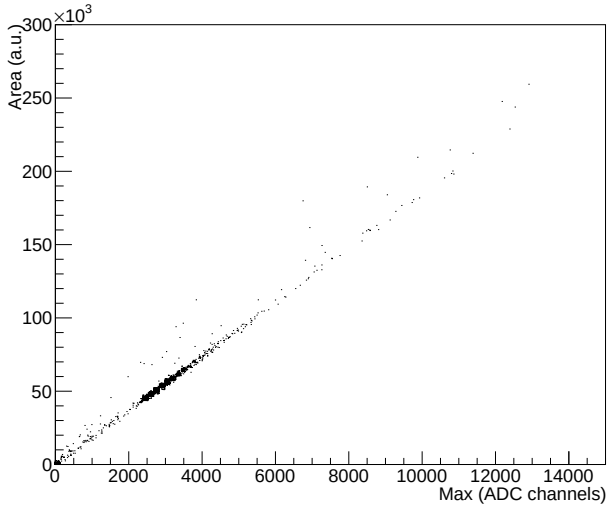


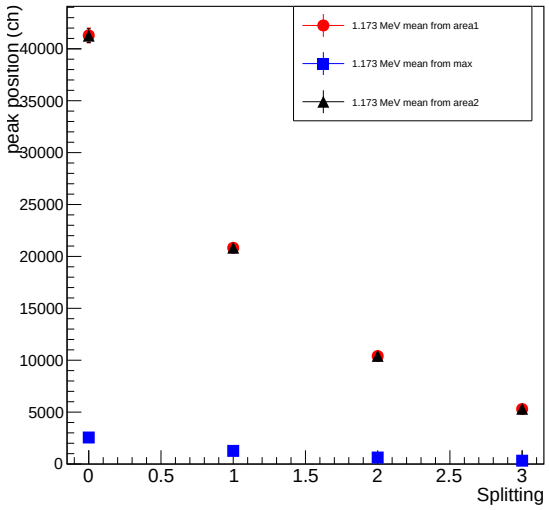
Figure 3.10: Values of the area, obtained using the Area1 method, are plotted against the corresponding value of the pulse peak (Max). Data come from measurements using a HV of $-450\,\mathrm{V}$. No attenuation was applied.

3.4.2 Resolution of γ-peaks

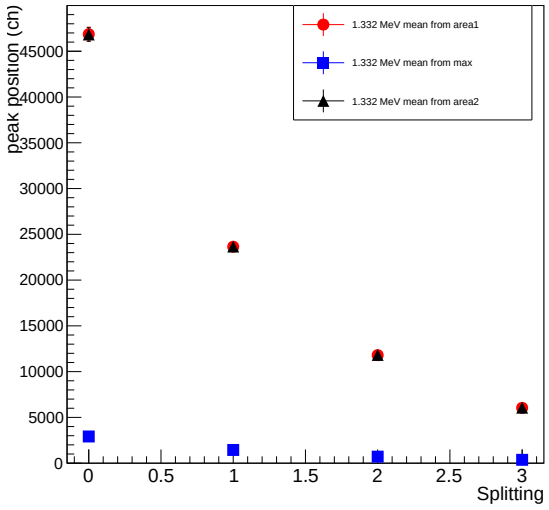
In this section, first we study how the energy spectrum given by a ^{60}Co source changes by varying the attenuation factor: we focus on the two energy peaks at 1.173 MeV and 1.332 MeV. After that we analyse how the energy resolution is affected by the attenuation factor: for this case we take the γ-peaks from ^{60}Co (at 1.173 MeV and 1.332 MeV) into account, as well as the one from ^{137}Cs (0.662 MeV). The HV was set to -600 V for all cases.

Plots in Figure 3.11 show how the mean values for the two γ-peaks of ^{60}Co change compared to the applied attenuation factor. The values are calculated using the three methods explained in section 3.3, as specified in the figure legends. The results are plotted as function of the number of ADC channels. The sigma of each peak was used as the uncertainty of the corresponding mean value.

In order to characterise the energy resolution of the CEPA4 crystals we use data from both a ^{60}Co and a ^{137}Cs source: this is done to analyse a larger energy range, thanks to the lower peak energy of γ-rays generated by the ^{137}Cs source. The energy resolution of the 0.662 MeV, 1.173 MeV and 1.332 MeV peaks was calculated as $R(\%) = 2.35 \cdot \sigma / \bar{x}$, where $\bar{x}$ is the mean value and σ the standard deviation obtained from the peak fit. The values are shown in Figure 3.12: the resolution is more or less constant up to an attenuation factor of 2 (splitting equal to 1), while it starts to get worse for higher values. In all cases, *Area1* and *Area2* methods provide a very similar resolution, while the one calculated from the *Max* method is worse: the best value is obtained using the method *Area2*.



(a) Mean value for the 1.173 MeV peak from ^{60}Co.



(b) Mean value for the 1.332 MeV peak from ^{60}Co.

Figure 3.11: Mean value (as peak position in ADC channels) for the two peaks from ^{60}Co versus splitting times. The uncertainties used are the sigmas of the energy peaks. The methods used to reconstruct the energy spectrum are indicated in the legends. The HV applied to the PMT is -600 V.

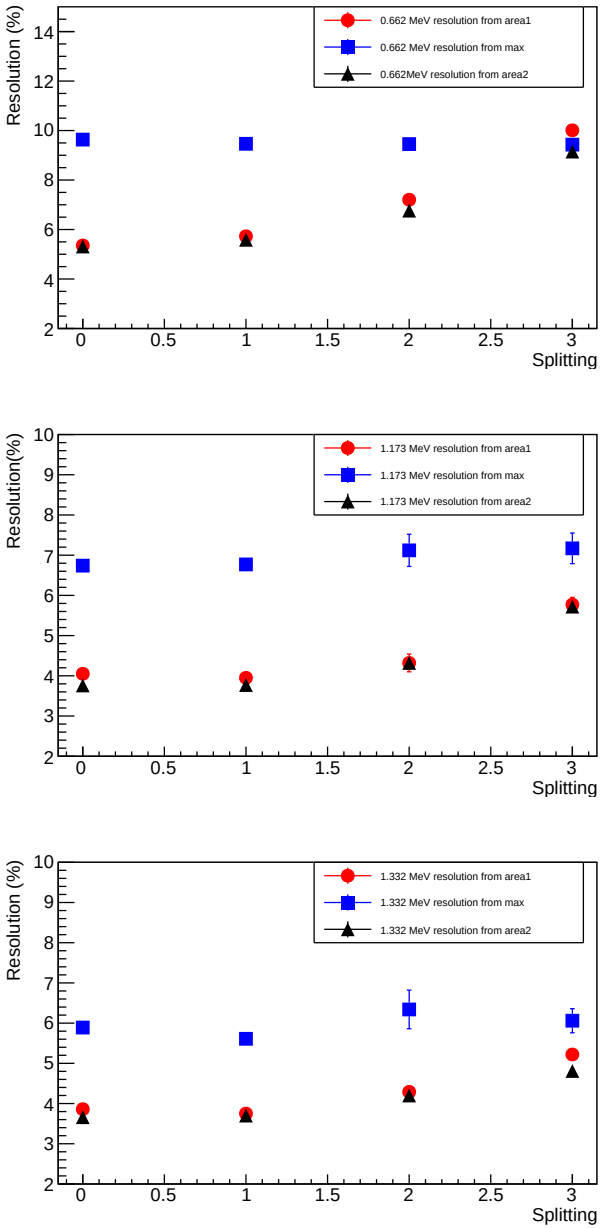


Figure 3.12: Resolution versus splitting for the 0.662 MeV (top), 1.173 MeV (middle) and 1.332 MeV (bottom) peaks. Methods used to reconstruct the energy spectrum are specified in the legend. In all cases the HV applied to the PMT is -600 V.

3.4.3 PMT Saturation and Signals Distortion

From the previous sections we conclude that a HV of $-600\,\mathrm{V}$ would be enough to satisfy the requirements on the γ-energy resolution. However to detect muons using this voltage, we need to apply an attenuation factor of 8, which deteriorates the γ-energy resolution. An idea to solve this problem is to increase the HV even further, meanwhile applying a higher attenuation factor. Nevertheless this approach reveals to have bad side-effects on the output signal from the PMTs.

To investigate the behaviour of the photomultiplier at higher HV values, the muon setup was used at voltages of $-700\,\mathrm{V}$ and $-800\,\mathrm{V}$. In Figure 3.13, the left column shows the waves acquired in the $LaBr_3$ crystal setting the HV to $-700\,\mathrm{V}$ (top) and $-800\,\mathrm{V}$ (bottom). An attenuation factor of 16 was used in both cases.

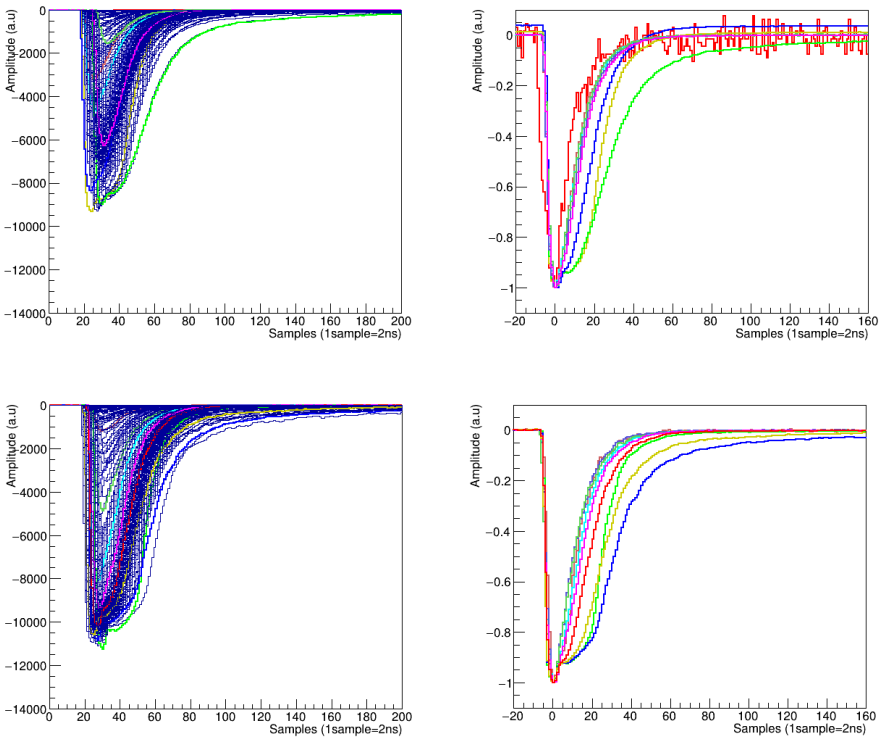
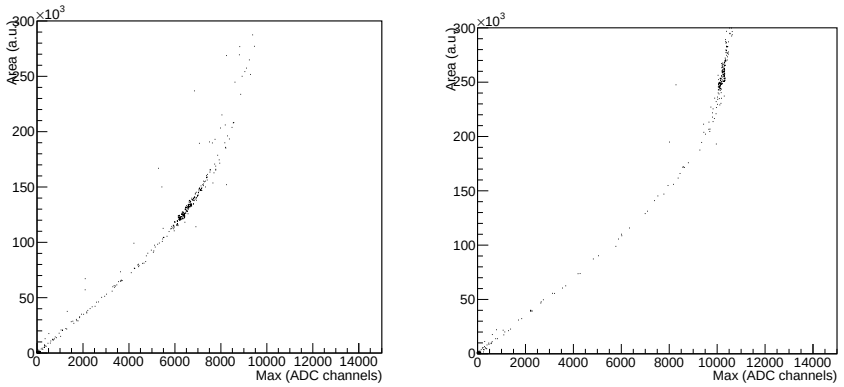


Figure 3.13: *Left Column*: measured waves from muons interaction in the $LaBr_3$ using $-700\,\mathrm{V}$ (top) and $-800\,\mathrm{V}$ (bottom), both with an attenuation factor of 16. *Right Column*: only the coloured waves from the two respective pictures in left column are normalized to their maximum amplitudes.

Simulations performed using *ggland* predict the muon peak at $\approx$22 MeV. The denser blue bands in Figure 3.13, i.e. at amplitude ≈ -7000 (top plot) and $\approx -10\,000$ (bottom plot), correspond to the mean energy of the muons. It is important to notice here the distortion of the signals due to the high voltage

level combined with the energy deposited by the muons. As expected, the distortion takes place at lower energies in the bottom plot compared to the top one. This is due to the larger HV applied to obtain the bottom plots. This effect is also visible also when the pulse energy is plotted as a function of the pulse peak: Figure 3.14a and 3.14b show that the area of distorted pulses increases while their maximum value does not.



(a) (b)

Figure 3.14: Values of the area, obtained using the Area1 method, are plotted against the corresponding value of the pulse peak (Max). Data come from measurements using a HV of $-700\,\mathrm{V}$ (a) and $-800\,\mathrm{V}$ (b). In both cases an attenuation factor of 16 is applied to the output signal.

To be sure that the CAEN digitizer is not somehow causing the distortion, the same type of measurements was carried out using a digital *LeCroy* oscilloscope as acquisition device, at a sampling frequency of $10\,\mathrm{GS/s}$. The results are shown in Figure 3.15: in the left column, the top plot shows the waves acquired using a HV of $-600\,\mathrm{V}$ and an attenuation factor of 4, while for the bottom one we used a HV of $-800\,\mathrm{V}$ and an attenuation factor of 16 (same as on the bottom plot from Figure 3.13). When comparing the waves obtained for $-800\,\mathrm{V}$ with the ones from Figure 3.13, it is observed that they do not have the same shape, but rather a distortion for larger amplitudes. At $-600\,\mathrm{V}$ instead there are very small distortions.

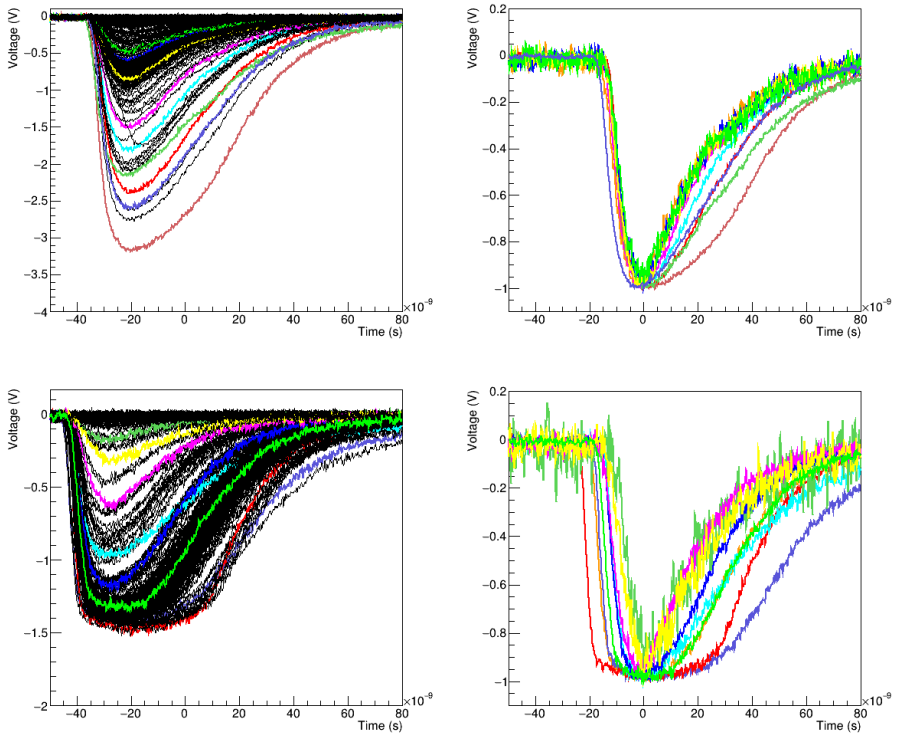


Figure 3.15: *Left Column*: measured waves from muons interaction in the $LaBr_3$ usign $-600\,V$ and an attenuation factor of 4 (top) and $-800\,V$ and an attenuation factor of 16 (bottom). *Right Column*: only the coloured waves from the two respective pictures in left column are normalized to their maximum amplitudes.

Chapter 4

Time Resolution

4.1 Motivation

With the studies here shown we want to investigate the time resolution achievable with the phoswich crystals and the electronics described in section 2.2. Several aspects are examined and results are promising:

- Tuning the electronics properly and using fitting operations, the time resolution of CEPA4 $LaBr_3$ crystals reaches values of around 50 ps;
- It is possible to obtain time resolution values smaller than 100 ps with sampling frequencies down to 250 MS/s;
- Lowering the bit resolution from 14 to 10 bits results only in a $\approx 6\%$ worsening of the time resolution.

We remind the reader that the electronics proposed for the CEPA part of CALIFA [4] plans to achieve a sampling frequency of 1 GS/s and a bit resolution of 14 bits. In this chapter however we do not discuss implications on the *pulse shape analysis* if the bit resolution and the sampling frequency are reduced.

It needs also to be considered that the σ_t values shown in the following sections refer to the time difference between two crystals. Therefore, to get the resolution of the individual crystals one needs to divide those values by $\sqrt{2}$.

4.2 Time Resolution using Muons

To characterise the time resolution of the CEPA4 crystals, we first exploit the muons' capability of traversing two adjacent crystals. In this way we can characterise the difference in the crossing time between the two crystals and give a first taste of the timing abilities of phoswich crystals.

4.2.1 Experimental Setup

The experimental setup is similar to the one used in the previous studies for detecting muons (see section 3.2.1). A HV of -450 V was used. However, to

monitor two stacked crystals at the same time, we need to feed the sampling device simultaneously with signals from two crystals on two separate channels.

The layout of the setup is shown in Figure 4.1 and a picture of it is shown in Figure 3.4. A muon that hits the two upper and two lower plastic scintillators creates a trigger that allows to record the pulses generated by the same muon in the upper and lower $LaBr_3$ crystals. The study is carried out mainly using the $LaBr_3$ part of the phoswich, because its higher time resolution compared to the $LaCl_3$ [7], allows us to improve the readout electronics so that we can exploit this feature.

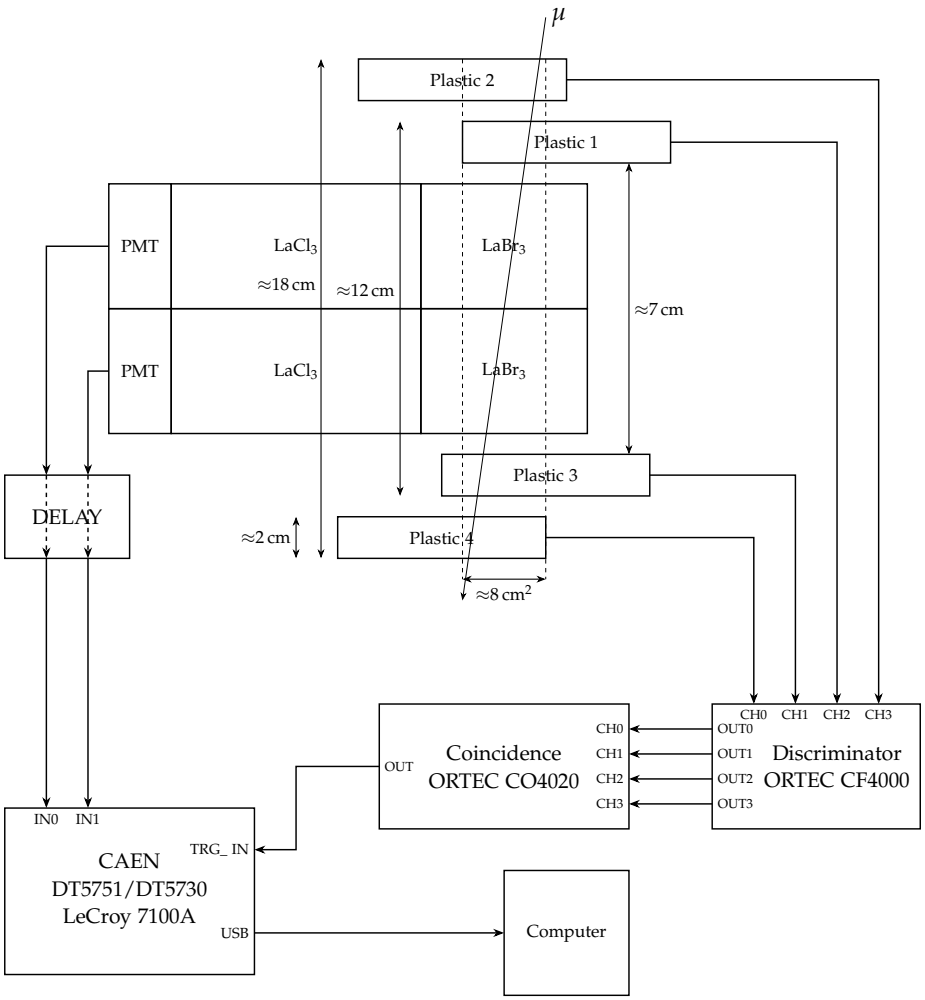


Figure 4.1: Schematic of the setup used for measuring muons (for details see text).

4.2.2 Procedure

The acquired raw data are stored in text files, where amplitude and time are sorted into arrays whose size depends on the acquisition system used. The steps followed to get the time difference between the pulses generated by the same muon in the upper and lower crystals are:

- **Baseline**: calculated using 20 samples in case of the CAEN digitizers or 1000 samples in case of the LeCroy oscilloscope. As explained in section 3.3, the baseline is the average of the first 20 amplitude samples. The obtained value is then subtracted from the wave samples.
- **Maximum Amplitude (MA)**: determined for each pulse and then used to set a threshold, which is defined as a *fraction* (f) of the MA. While for the CAEN digitizers the MA is simply the largest amplitude, for the LeCroy oscilloscope a Gaussian fit to the top part of the wave is performed and the maximum amplitude is taken with respect where the mean value of the fit is. This different procedure is needed to get a more accurate value, as the LeCroy device features a very large sampling rate with much noise.
- **Time Determination**: to determine the time reference of each pulse three different approaches were used:
  1. **Linear Interpolation**: a linear interpolation is performed between two points of the waveform, the one just before the threshold and the one just after the threshold;
  2. **Linear Fit**: a linear fit is performed in the region around the threshold (using three samples for the digitizers and a larger number for the oscilloscope). The time reference is given by the fit at the threshold.
  3. **Polynomial Fit**: a second order polynomial fit is performed in the region around the threshold (using three samples for the digitizers and a larger number for the oscilloscope). The time is given by the fit at the threshold.
- **Time Difference Calculation**: the difference between the time reference of two pulses is calculated for each of the three methods mentioned in the previous point and the data are histogrammed. Then a Gaussian fit is performed to obtain the width, which is the sought time resolution. Only pulses with a minimum required amplitude, which are completely inside the dynamic range of the device, were used to calculate the resolution.

As the two CAEN modules differ in sampling frequency and bit resolution, we started the study by using them without altering the output data. This was done to see if (and, in case, how) the different features influence the time resolution. We used data from the LeCroy oscilloscope as well, since its characteristics are dissimilar from the other two.

After these initial basic investigations, we moved forward trying to simulate a broader range of sampling frequencies and bit resolutions using the data acquired with the two CAEN modules. In section 4.4 these studies are addressed.

4.2.3 Results and Comparison

In this section we present data and results obtained using the three digitization modules described. A comparison between the final data will be carried out in the end.

4.2.3.1 Data Elaboration

Following the procedure illustrated in the previous section, we calculate the time difference using the three listed methods for each digitizer: an example using data from the DT5730 is shown in Figure 4.2. This is done using fractions f ranging from 5 to 50% of the MA of the pulse.

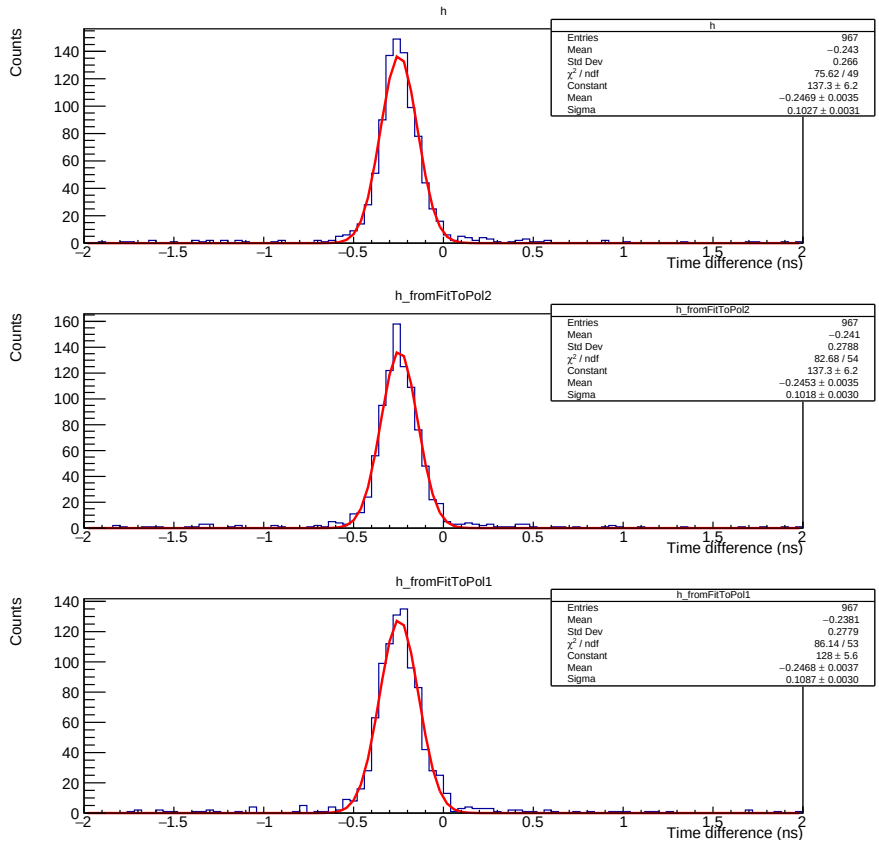


Figure 4.2: Histograms of the time difference using data from the DT5730. Top: time difference obtained from the linear interpolation method. Center: time difference obtained from the second order polynomial fit. Bottom: time difference obtained when the first order polynomial fit is used for the time calculation. All three plots used f=50%.

Procedure for LeCroy 7100A Data As described in section 4.2.2, data from the LeCroy oscilloscope need to be treated differently than those from the CAEN modules. To show this procedure we take as example for the MA determination an event where both the pulses are inside the dynamic range of the oscilloscope. The waves are shown in Figure 4.3, with in the left plot the pulse from the upper crystal and in the right plot that coming from the lower crystal.

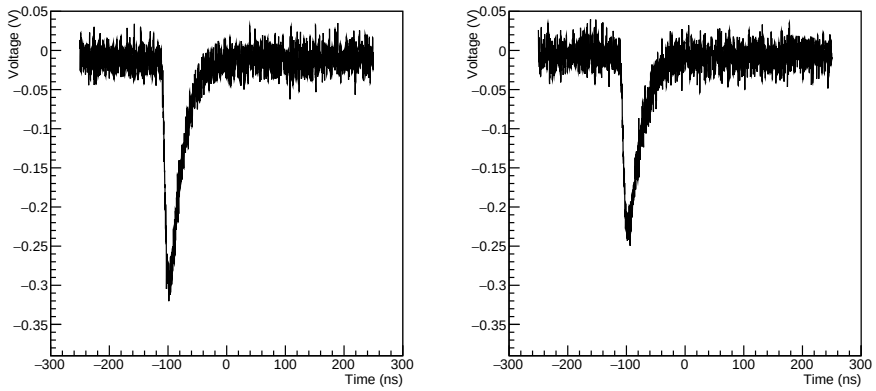


Figure 4.3: Waves acquired with the LeCroy oscilloscope for one event. Left: upper crystal. Right: lower crystal.

After subtracting the baseline, the maximum amplitude is determined: first the point with the maximum amplitude is found for each pulse and used to define the range where the Gaussian fit will be performed. Figure 4.4 shows the Gaussian fits performed in a region around the maximum amplitude point. Once the MA is obtained, the procedure is identical to that used for the other two digitizers.

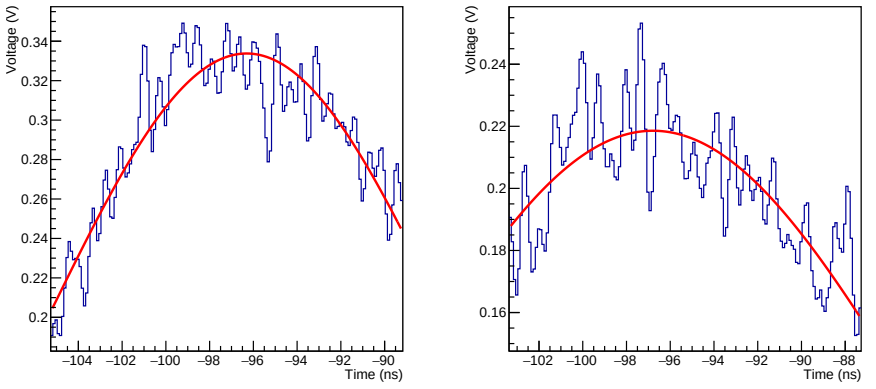


Figure 4.4: Gaussian fits performed to the waves acquired with the LeCroy oscilloscope to determine the maximum. Left: upper crystal. Right: lower crystal.

4.2.3.2 Comparison

The sigma values from the time differences taken into account for a comparison are only those obtained using the two CAEN modules. Values produced from the LeCroy data show a much worse resolution, probably due to the lower bit resolution: using the three methods, the values are (533 ± 53) ps, (242 ± 9) ps and (229 ± 9) ps, respectively. We therefore didn't proceed with a deeper analysis of these last numbers.

Figure 4.5 and Tables 4.1a and 4.1b show the dependence of the σ_t obtained from the Gaussian fits to the time difference plots obtained by the three methods for the two digitizers. What can be seen is that, for low fraction values, the best approach for both CAEN modules is to use the second order polynomial fit. This is expected because low fraction values means trying to fit in the curved rising part of the pulse. For medium fraction values, the fit is performed in the linear rising part of the pulse and therefore a simple interpolation gives almost the same result of a polynomial fit.

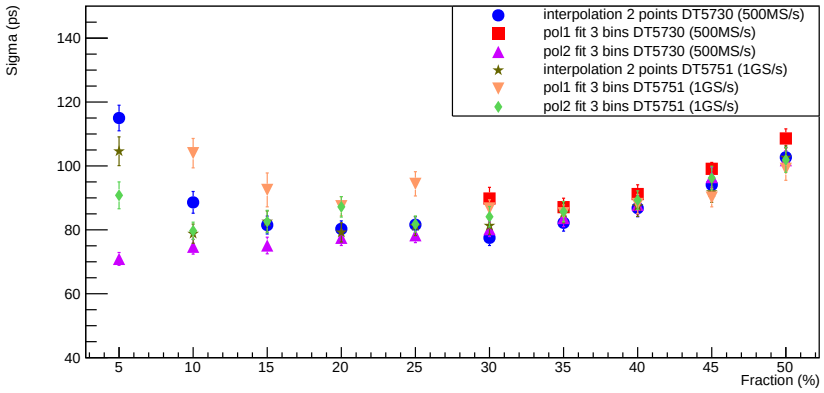


Figure 4.5: Extracted σ_t versus the fraction used to set the threshold calculated by the three methods. Data for both the CAEN digitizers are provided.

f (%)	σ_i (ps)	σ_f (ps)	σ_s (ps)
5	115.0±4.0	—	**70.9±2.0**
10	88.6±3.4	—	74.7±2.3
15	81.5±2.8	—	75.1±2.6
20	80.3±2.5	—	77.5±2.4
25	81.6±2.5	—	78.3±2.3
30	**77.5±2.4**	89.8±3.5	80.2±2.3
35	86.8±2.7	**87.1±2.8**	83.7±2.7
40	86.8±2.7	91.2±2.9	87.8±3.0
45	94.1±3.0	99.1±2.0	96.5±3.0
50	102.7±3.1	108.6±3.0	101.8±3.0

(a)

f (%)	σ_i (ps)	σ_f (ps)	σ_s (ps)
5	104.6±4.5	—	90.8±4.2
10	**78.8±3.0**	104.0±4.6	**79.7±2.7**
15	82.6±3.3	92.5±5.3	81.9±3.5
20	79.2±3.0	87.4±3.0	87.2±3.2
25	81.2±3.0	94.4±3.8	81.8±2.5
30	81.3±2.7	86.7±2.7	84.1±3.2
35	85.3±3.1	**85.2±2.6**	85.7±3.2
40	87.4±3.2	88.1±3.0	89.3±2.9
45	91.8±3.2	90.1±2.9	96.2±3.7
50	102.6±3.7	99.1±3.6	101.9±3.9

(b)

Table 4.1: Calculated σ_t from the Gaussian fits to the time difference plots obtained from the three methods using data from (a) the CAEN DT5730 (500 MS/s) and from (b) the CAEN DT5751 (1 GS/s). The subscripts i, f and s identify the method used for determining the time reference of the single pulse and therefore stand for the *interpolation*, fit to a *first* and *second* order polynomials, respectively.

4.3 Time Resolution using γ-rays

The ideal setup to measure time resolution using γ-rays would have a ^{22}Na source located between two crystals, which are placed at sufficient distance to each other but symmetrically with respect to the source. This arrangement would allow to exploit the back-to-back γ-γ coincidence. Such a setup could not be used because the CEPA4 crystals can not be separated. Therefore other ways of measuring the time resolution for γ-rays were investigated.

4.3.1 Experimental setup and Procedure

To calculate the time resolution using γ-rays, two different experimental setups were developed:

- **Setup A**: a ^{60}Co source is attached on the front and in the center of crystals 3 and 4 of CEPA4. In this way the two γ-rays from the source may hit both crystals. In this case we measure the time difference between signals from the two $LaBr_3$ crystals.
- **Setup B**: a ^{22}Na source is positioned in between crystal 3 of CEPA4 and a plastic scintillator. In doing so the two 511 keV γ-rays hit both detectors in the same event.

Only the CAEN DT5751 was used for this measurement and in both the setups only two channels were involved. Two different software triggers were taken into account to calculate the time difference:

- **AND of both channels**: the digitizer records data only if two pulses, one for each channel, are detected within a configurable time window. For our purposes this window was set to 10 ns.
- **ch0_TO_ALL**: every time a pulse in channel 0 is detected, the digitizer stores traces from both channels.

In both cases the trigger for the single pulse is self-generated by the digitizer, according to a user-set threshold. All the configurations and the acquisitions of data were done using the CAEN software *digiTes* (see section 2.2.2.1).

A number of problems were encountered with the setups and the trigger generation. Setup B was not suitable for any trigger scheme: although we do not have a conclusive explanation, we notice that the photopeak efficiency for plastic scintillators is rather low [16], which cause the detector to trigger also on spurious and background events. What we noticed was the poor time resolution of a few nanoseconds. Using setup A together with the AND trigger, we were not able to obtain a good energy spectrum.

Therefore the only way to measure reliable data was *setup A* with the software trigger `ch0_TO_ALL`. The events were further filtered using the energy requirements, for a single event, of $\approx$1.2 MeV in crystal 3 and $\approx$1.3 MeV in crystal 4 or vice versa: these two numbers are the energies of the γ-rays from the decay of ^{60}Co. To obtain the time resolution we used the same procedures as employed for muons.

4.3.2 Results

Table 4.2 shows the σ_t results when the fraction f ranges from 5-50%. The best value was obtained using the second order polynomial fit and a fraction f of 5%. Considering results obtained with the muons interaction (Table 4.1b), it is easy to see that the best value of time resolution with γ-rays is worse than any result obtained with muons.

f (%)	σ_i (ps)	σ_f (ps)	σ_s (ps)
5	**193.1±4.5**	237.4±6.7	**189.4±4.8**
10	198.2±4.9	**201.8±4.7**	199.4±5.0
15	209.6±5.1	208.6±5.0	205.6±4.6
20	220.3±4.9	219.4±5.0	226.8±5.4
25	230.7±5.2	227.5±4.8	239.5±5.5
30	245.2±5.4	241.8±5.2	249.1±5.4
35	258.6±5.8	258.3±5.6	257.5±5.5
40	273.4±5.8	273.4±5.7	285.2±6.0
45	293.5±6.1	286.7±6.3	292.4±6.0
50	318.3±6.8	312.7±6.5	322.1±7.0

Table 4.2: Obtained σ_t from the Gaussian fits to the time difference plots obtained from the three methods using data from the CAEN DT5751 (1 GS/s). γ-rays from a ^{60}Co source were used to generate pulses in $LaBr_3$ crystals. The subscripts i, f and s identify the method used for determining the time reference of the single pulse and therefore stand for the *interpolation*, fit to a *first* and *second* order polynomials, respectively.

4.4 Frequency and Bit Resolution Studies

In the previous sections we analysed how the time resolution is influenced by the particles, and therefore the energy deposition, considered. The procedure, including fits, thresholds, etc., to obtain the final data was also scrutinised.

Now we want to investigate how the sampling frequency and the bit resolution can influence the time resolution. Although we have access to two models of digitizers (see section 2.2.2), characterised by different features, we want to extend the range of the studied sampling frequencies and bit resolutions. To do so, we manipulate via software the collected data in order to simulate different digitisation cases.

Following the studies conducted in the previous sections, we show how the time resolution for several fractions f of the MA varies by changing the sampling frequency and the bit resolution.

4.4.1 Sampling Frequency Analysis

Two separate analyses are conducted for the two CAEN digitisers. The procedure for obtaining various sampling frequencies is the same in both cases, although the ranges were different. In the end a comparison between the two sets of results will be carried out.

4.4.1.1 Data Manipulation Procedure

If we acquire traces within a time window T_w and we store a number of samples N for each trace. The sampling frequency is $f_s = N/T_w$: from this relation it is possible to simulate other (lower) f_s values just by skipping samples, therefore reducing N.

Suppose now we want to delete every second sample. We can proceed removing the 2^{nd}, the 4^{th}, the 6^{th} etc. or the 1^{st}, the 3^{rd}, the 5^{th}. In this work we call this shifting in choice *offset*. While in this example the *offset* can be set as either 0 or 1, when it comes to higher removal factors r, the offset can take values between 0 and $r-1$. Since we are always considering data from two separate detectors, the easiest approach is to use the same offset value for two pulses in the same event: in the results this will be labelled as *same offset*. However actual pulses from different detectors are not in perfect coincidence at the sampling stage, because of different cable lengths or particles hitting detectors at different moments. To be able to simulate this scenario, two random-generated offsets are applied to the removal procedure in the two channels. Data from this approach will be labelled as *different offset*. In the following sections, we show mainly data obtained using this approach.

4.4.1.2 Data from the CAEN DT5751

The DT5751 module has a sampling frequency of 1 GS/s (period of 1 ns). By removing one out of two samples, we therefore simulate a digitizer with a sampling frequency of 500 MS/s (2 ns), removing two out of three samples the

sampling interval becomes ≈333 MS/s (3 ns), etc. These studies were done for sampling frequencies ranging from 250 MS/s to 1 GS/s.

Figure 4.6 and Table 4.3a show the σ_t results obtained for the time difference using data from the DT5751. Different sampling intervals selected with different offsets between traces are used.

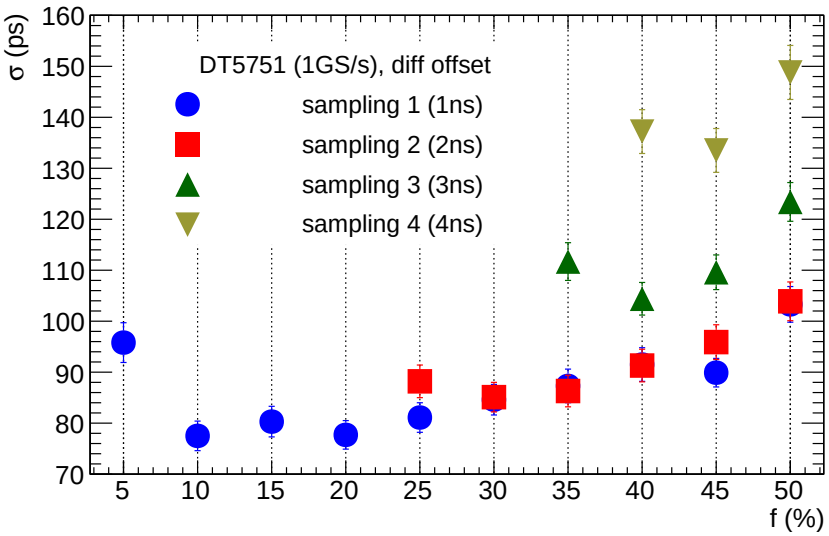


Figure 4.6: Extracted σ_t from the Gaussian fits to the time difference plot obtained for the DT5751 module versus fraction for different samplings, using differently random-generated offsets for the two traces.

Table 4.3b shows data obtained with the *same offset* approach. A fast comparison with data of Table 4.3a reveals the better values obtained with the *same offset* mode compared to the (more realistic) *different offset* approach.

f (%)	σ_1 (1 GS/s) (ps)	σ_2 (500 MS/s) (ps)	σ_3 (333 MS/s) (ps)	σ_4 (250 MS/s) (ps)
5	95.8±3.9	–	–	–
10	**77.5±2.9**	–	–	–
15	80.3±3.0	–	–	–
20	77.7±2.8	–	–	–
25	81.1±2.9	88.2±3.2	–	–
30	84.6±3.0	**85.1±2.9**	–	–
35	87.3±3.3	86.3±3.1	–	–
40	91.5±3.3	91.3±3.2	**104.4±3.5**	137.2±4.3
45	89.9±2.8	95.9±3.4	109.6±4.1	**133.5±4.3**
50	103.3±3.5	103.9±3.8	123.4±4.7	148.8±5.3

(a)

f (%)	σ_2 (500 MS/s) (ps)	σ_3 (333 MS/s) (ps)	σ_4 (250 MS/s) (ps)
5	–	–	–
10	–	–	–
15	–	–	–
20	–	–	–
25	**80.5±2.8**	–	–
30	82.6±2.7	–	–
35	84.0±2.8	–	–
40	88.5±3.1	**93.2±3.7**	**96.5±3.6**
45	91.9±3.2	96.2±3.6	98.0±3.5
50	99.9±3.5	98.3±3.2	101.2±4.3

(b)

Table 4.3: Elaborated σ_t obtained from the Gaussian fits to the time difference plots using data from the DT5751 digitizer. In (a) two random-generated offsets, one for each channel, were used when samples were removed. In (b) instead the same offset for both channels was used. All the data analysed have a 10 bits resolution but different sampling frequencies, as indicated within parentheses. The subscripts *1*, *2*, *3* and *4* highlight this different sampling obtained via post software operations.

4.4.1.3 Data from the CAEN DT5730

The DT5730 has a sampling frequency of 500 MS/s (period of 2 ns). By removing one sample every two we then simulate a digitizer with a sampling frequency of 250 MS/s (4 ns), removing two every three samples the sampling frequency turns to be ≈166 MS/s (6 ns), etc. These studies were done for sampling frequencies ranging from 125 MS/s to 500 MS/s.

Figure 4.7 and Table 4.4 show the σ_t results obtained for the time difference when the DT5730 is used at various sampling frequencies selected with a random offset between pulses.

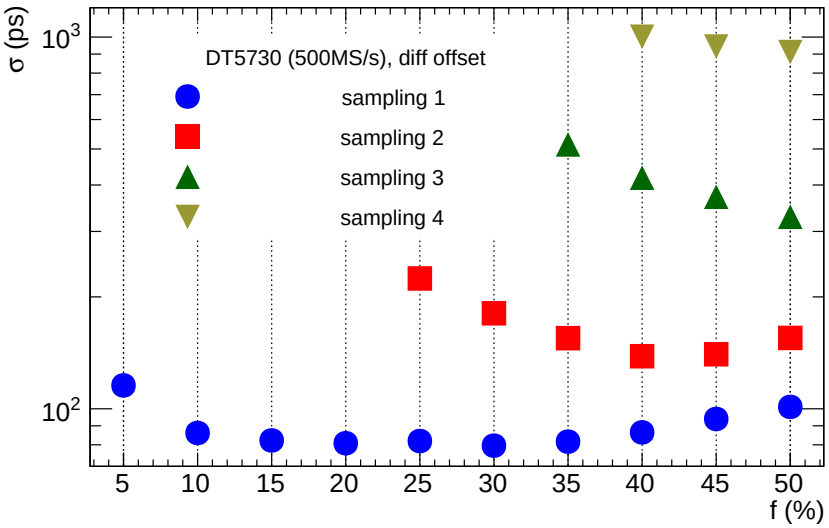


Figure 4.7: Measured σ_t from the Gaussian fits to the time difference plot obtained for the DT5730 versus fraction for different samplings, using differently random-generated offsets for the two traces.

f (%)	σ_1 (500 MS/s) (ps)	σ_2 (250 MS/s) (ps)	σ_3 (166 MS/s) (ps)	σ_4 (125 MS/s) (ps)
5	115.6±4.4	–	–	–
10	86.1±3.3	–	–	–
15	82.2±3.2	–	–	–
20	80.8±2.5	–	–	–
25	81.9±2.4	224.2±6.6	–	–
30	**79.6±2.5**	180.7±4.5	–	–
35	81.6±2.6	154.9±3.8	514.3±16.3	–
40	86.4±2.8	**138.5±3.9**	418.5±12.0	1002.0±38.6
45	93.9±3.0	140.2±4.4	371.9±10.0	943.2±36.2
50	101.2±3.1	154.1±4.5	**328.6±8.1**	**911.3±31.8**

Table 4.4: Obtained σ_t from the Gaussian fits to the time difference plots using data from the DT5730 digitizer and the linear interpolation method. Two random-generated offsets, one for each channel, were used when samples were removed. All the data analysed have a 14 bits resolution but different sampling frequency, as indicated within parentheses. The subscripts *1*, *2*, *3* and *4* want to highlight this different sampling obtained via post software operations.

4.4.1.4 Results Comparison

Figure 4.8 shows the best σ_t values for each sampling frequency, using random offsets. Data come from Table 4.4 for the DT5730 (blue X-es) and from Table 4.3b for the DT5751 (red crosses): only the best values for each column (in bold) are used in the plot. The first thing to notice is how the time resolution deteriorates for sampling frequencies below 250 MS/s. The second characteristic, which will be analysed more deeply in the next section, is that the data from the two digitizers show a good agreement between each other, although two different bit resolutions are used.

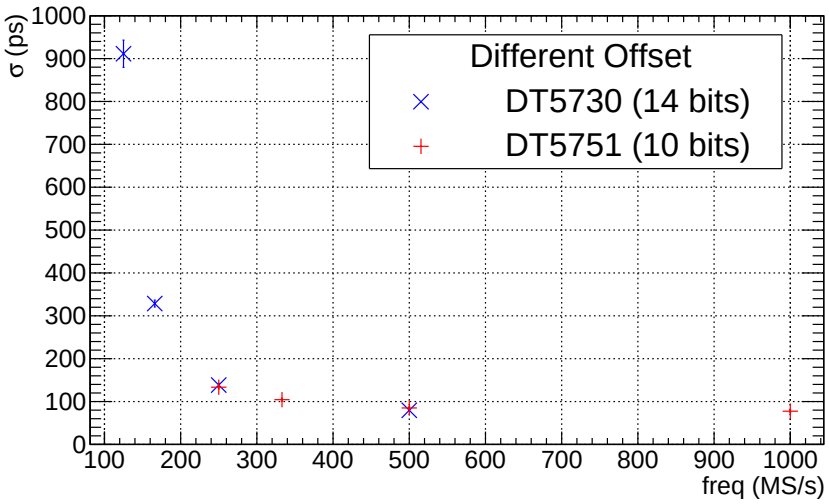


Figure 4.8: Measured σ_t from the Gaussian fits to the time difference plots obtained for the DT5730 (blue circles) and the DT5751 (red squares) versus different sampling frequencies using a different random offset. For each sampling frequency, among all fractions f, we considered only the best σ_t value obtained for each digitizer.

4.4.2 Bit Resolution Analysis — DT5730 only

This study was conducted using only data from CAEN DT5730. The reason is that this module has a bit resolution of 14 bits, while the DT5751 is limited to 10 bits.

4.4.2.1 Data Manipulation Procedure

Usually the output data from a digitizer come characterized by a fixed number of bits. Increasing the number of bits with post-acquisition operations would be difficult and inaccurate because it requires oversampling and has the effect of removing noise.

Therefore we only try to reduce the bit resolution: to do so, we divide the amplitude of each sample by $2^{init\_b - sim\_b}$, where *init_b* is the initial bit resolution and *sim_b* is the resolution we want to simulate. In the particular case of CAEN DT5730, *init_b* is 14 and we try to simulate data with 12, 10 and 8 bits (*sim_b*). No reduction in sampling frequency is carried out, so data acquired at 500 MS/s are used.

To calculate the time reference of each pulse, the linear interpolation method around a fraction f of MA was used.

4.4.2.2 Results

Figure 4.9 shows the resulting σ_t values of time differences varying the fraction f for several bit resolutions. Numbers are reported in Table 4.5. It is easy to note that there is almost no difference between the 14, 12 and 10 bits cases, however lowering the resolution to 8 bits increases the σ_t significantly. In all the cases, results follow the same trend.

f (%)	**14 bits** (ps)	**12 bits** (ps)	**10 bits** (ps)	**8 bits** (ps)
5	115.6±4.4	124.0±4.8	129.2±5.8	263.3±9.2
10	86.1±3.3	88.9±3.2	90.8±2.8	157.0±5.5
15	82.2±3.2	84.9±3.0	90.0±2.8	135.0±4.4
20	80.8±2.5	83.5±2.6	86.3±2.8	122.0±3.7
25	81.9±2.4	**81.5±2.4**	**84.4±2.6**	**114.6±3.8**
30	**79.6±2.5**	82.4±2.6	85.1±2.7	118.8±3.8
35	81.5±2.6	85.3±2.3	86.8±2.6	122.7±3.7
40	86.4±2.8	88.2±2.8	90.0±2.9	120.7±3.8
45	93.9±3.0	95.5±3.2	93.0±3.1	133.9±4.2
50	101.2±3.1	102.8±3.1	105.6±3.7	139.9±4.9

Table 4.5: Obtained $\sigma_t$as the width of the Gaussian fits to the time difference distributions, acquired using the DT5730 digitizer. The calculations are carried out for different bit resolutions and different fractions.

In Figure 4.10 the σ_t is studied as a function of the bit resolution: here, among all the fractions f, only the best σ_t values are considered for each bit

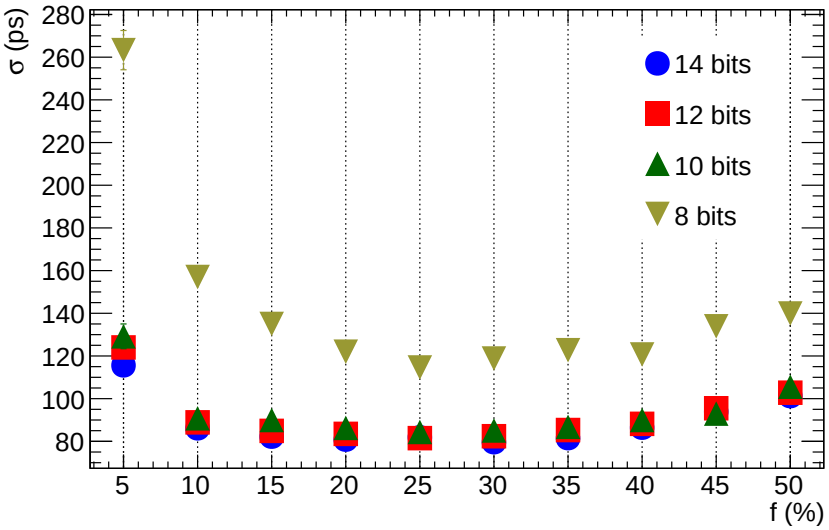


Figure 4.9: Extracted σ_t from the Gaussian fits to the time difference plots using data from the DT5730 versus fraction for different bit resolutions.

resolution. An exponential decrease of the σ_t while increasing the bit resolution is seen. Comparing this plot with Figure 4.9, we can confirm the need to avoid to go below 10 bits of resolution. However there is not a big improvement between the 10 and 14 bits cases.

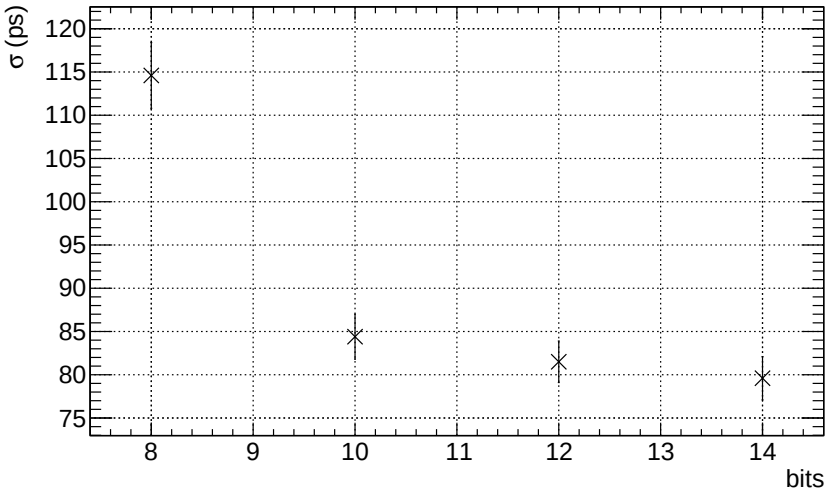


Figure 4.10: Extracted σ_t from the Gaussian fits to the time difference using data from the DT5730 versus different bit resolutions. For each bit resolution, among all fractions f, we considered only the best σ_t value obtained.

4.5 Time Resolution for $LaCl_3$ Crystals

$LaCl_3$ is known to have worse time resolution than the $LaBr_3$ [7]. Here a short study on the time resolution of a CEPA4 $LaCl_3$ crystal is reported. The study was carried out with both the DT5730 and DT5751 modules using the approach for muons and the same procedure shown for the $LaBr_3$ crystals (see section 4.2.2). Table 4.6a and 4.6b summarizes the results obtained with the DT5730 and the DT5751 respectively. A fast comparison with data in Table 4.1a and 4.1b shows that the time resolution given by $LaCl_3$ crystals is larger (at least $\approx$50 ps) than the one offered by $LaBr_3$ crystals.

f (%)	σ_i (ps)	σ_s (ps)	σ_f (ps)
5	155.9±2.7	170.5±4.0	–
10	136.9±2.0	154.2±2.0	–
15	134.4±1.9	**140.5±1.8**	–
20	**132.6±1.8**	156.5±2.0	–
25	133.9±1.7	177.6±2.2	–
30	136.9±1.7	187.6±2.3	222.3±4.6
35	144.2±1.8	201.5±2.5	192.2±2.8
40	152.1±2.0	202.0±2.6	**171.0±2.3**
45	162.4±2.1	208.5±2.5	172.2±2.3
50	173.8±2.2	212.7±2.5	183.6±2.5

(a)

f (%)	σ_i (ps)	σ_s (ps)	σ_f (ps)
5	195.7±3.3	209.0±4.3	—
10	155.8±2.8	161.8±3.1	195.2±3.3
15	145.9±2.7	147.2±2.7	146.6±2.7
20	**141.5±2.4**	**144.5±2.8**	**135.1±2.5**
25	141.8±2.6	145.8±2.9	137.1±2.7
30	144.6±2.6	147.3±2.7	**135.1±2.4**
35	149.5±2.7	152.8±2.9	139.2±2.5
40	156.0±2.8	158.7±3.1	145.5±2.6
45	165.2±3.0	165.2±3.0	154.9±2.7
50	170.3±2.9	193.7±4.3	179.7±3.8

(b)

Table 4.6: Measured σ_t obtained from the Gaussian fits to the time difference plots obtained from the three methods using data from (a) the CAEN DT5730 (500 MS/s) and from (b) the CAEN DT5751 (1 GS/s). Pulses were generated by CEPA4 $LaCl_3$ crystals. The subscripts *i*, *s* and *f* identify the method used for determining the time reference of the single pulse and therefore stand for the *interpolation*, fit to a *second* and *first* order polynomials, respectively.

Part II

Electronics and Readout

Chapter 5

Introduction to Some Electronics Concepts

In this introductory chapter, we describe some basic concepts related to electronics, which will be helpful for understanding the following sections.

5.1 Digital Electronics

Digital electronics is the branch of electronics which deals with *discrete*, or *digital*, signals: in most of the electronics circuits the voltage of a node is considered as such a signal. Considering only two-valued signals[1], the *boolean logic* can be easily employed in order to describe them. The information that these signals carry is referred to as having either a 1 or a 0 value (or *true/false*) and it is called *bit*. The bit information can be also stored in some electronic components for further elaborations.

Digital circuits and components can be divided into two main categories: *combinational* and *sequential* logic circuits.

5.1.1 Combinational Logic

This category gathers all electronic circuits which implement *boolean logic functions*, where the output values depend only on the simultaneous input values[2]. Examples of simple logic gates are AND, OR, NOT, XOR and they have the same properties as the corresponding boolean operators.

A boolean function is described using a *truth table*, where any combination of input values is associated with an output value. Such functions can be

[1] As said, we are going to describe digital electronics as logic operation of two values. This is still true when talking about logic operations in circuits. However, recent developments in the memory business have led to the ability to store, inside one unique cell, a larger number of values: this has been made possible thanks to manufacturing advancements. These multilevel signals however cannot be employed directly as inputs in logic operations.

[2] This is obviously true only on paper or in simulations: in the real world the circuit between the input and the output pins introduces some delay. Except for this delay, the output signals depends completely on the input ones.

characterised by any number of inputs, as well of outputs. An example of such a table can be found in Table 5.1.

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

Table 5.1: Truth table of the logic function $F = A$ NAND B, where A and B are the inputs, while F is the output.

All logic functions can be built using a set of basic gates, specifically NAND, NOR[3] and NOT gates [18]. There are two forms to represents logic functions: *sum-of-products* and *products-of-sum*, where a *sum* is an OR/NOR operation and a *product* stands for an AND/NAND. For every expression, there are many ways to represent the same logic function. To implement efficient logic systems, we need to simplify and minimise these expressions. One simple way of doing this is through a *Karnaugh map* [18].

5.1.1.1 Karnaugh Maps

Karnaugh maps (K-maps) are useful tools for simplifying and minimising logic functions through human inspection: for this last reason, K-maps with more than six variables are difficult to manage, though still functioning.

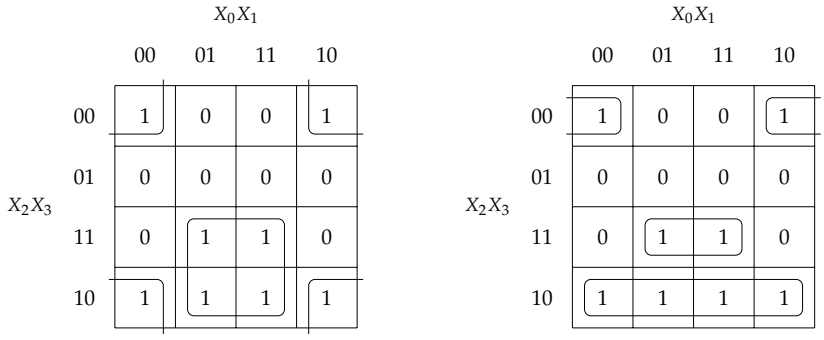
To use a K-map we first need to specify the outcome of the function under investigation for all possible input combinations. Once this is done, we start building the map, which is a table where the inputs are split between rows and columns. In the case of a 2-input function, one input characterises the columns, while the other the rows. For a 3-input function, two input combinations can be listed along the columns, while the remaining variable runs along the rows. In a 4-input case, we can split equally the inputs along the rows and the columns. Each outcome is then placed inside the cell corresponding to the input combination. In the end what is obtained is a two-dimensional truth table.

There is however an important rule to follow for building a usable map: the input combinations need to be ordered in a *Gray code* fashion. The basic characteristic is that two adjacent combinations differ only by one bit value. In the case of two variables, the sequence is: 00, 01, 11, 10. It is important to remember this feature, since usually we order binary numbers according to their value, such as 00, 01, 10, 11.

Now we need to gather adjacent "1" values in the table using the smallest possible amount of groups, or, in other words, using groups as large as possible. However each group needs to be shaped as a line (part of a column or a row) or a rectangle and it has to contain a number of 1s which is equal to a power

[3] In the manufacturing world, the usage of the negated version of AND and OR is more common because the realisation of the first ones requires a smaller number of transistors compared to the non-negated [17].

of 2 (so 1, 2, 4, 8, etc. elements). Moreover, in a K-map the cells of one edge are adjacent to the cells of the opposite edge. It is then important to remember that groups can overlap in order to obtain the simplest configuration, where each group covers as much area as possible and where the number of groups is the smallest achievable. Figure 5.1 shows a K-map with many of the possible groups highlighted[4]. It is easy to understand that Figure 5.1a uses the smallest amount of groups to gather all the 1s, while Figure 5.1b uses one group more and smaller groups as well, giving a more inefficient result.



(a) Efficient grouping.

(b) Inefficient grouping.

Figure 5.1: Karnaugh Map for the function F = $X_1 \cdot X_2 + \overline{X_1} \cdot \overline{X_3}$. Many groups of 1s are highlighted, but not all of them need to be considered in the simplification process. As shown in the left figure, with only two groups, made of four elements each, it is possible to gather all the 1s. The right picture instead uses three groups and it would bring to a more inefficient result.

We therefore take into account only Figure 5.1a. At this point, we can write the logic expressions representing each block. The rule here is to include in the expression only those variables which do not change value inside a block. If their value is 0, they need to be considered as *negated*, otherwise they are just taken as they are. Then those variables are *multiplied* (`and` operation) among each other and finally all the different expressions are summed (`or` operation) together.

- **Square block**: X_1 and X_2 are the two variables remaining the same and their value is 1. So the expression for this block is $X_1 \cdot X_2$.
- **Corners block**: X_1 and X_3 keep their value constant. However, in this case both are 0, so they need to be negated, giving the expression $\overline{X_1} \cdot \overline{X_3}$.

Finally we just sum together the two expressions we got and the result is

$$\mathrm{F} = X_1 \cdot X_2 + \overline{X_1} \cdot \overline{X_3}$$

or in logic terms

$$\mathrm{F} = (X_1 \texttt{ and } X_2) \texttt{ or } (\texttt{not} X_1 \texttt{ and not} X_3)\,.$$

[4]Showing all the possible groups would have resulted in a quite messy picture.

5.1.1.2 Multiplexer — MUX

A block which will be widely used in this part of the work, is the *multiplexer*. Although we could described it with its logic function, it becomes more obvious and easier using the *black-box* approach. A multiplexer is a module with 2^n inputs, one output and n controlling inputs. The values of the controlling inputs direct only one input towards the output.

5.1.2 Sequential Logic

In circuits implementing *sequential logic* the output values depend also on previous input values, other than current ones. To use old values, these kind of circuits need blocks able to store information, such as *registers*, *flip-flops* and *latches*. These components store the information either using a positive feedback, creating a *bistable* circuit which holds the data whilst the supply voltage is on (*static*), or charging and discharging a capacitor (*dynamic*). In both cases the circuit needs a *gating signal*, that is a signal which temporarily breaks the positive feedback or allows to charge/discharge the capacitor. If the gating signal is a clock signal, the circuit is called *synchronous*, otherwise it is called *asynchronous*.

There is a lot of confusion in the literature about the classification of the different blocks [17]. Usually a *latch* is referred to be *level-sensitive*, meaning that its output (the stored information) follows the input value while the level of the gating signal is high (positive latch) or low (negative latch). A flip-flop instead is a *edge-sensitive* device, which means that the stored information changes value only around the moment when the gating signal flips between low to high (positive edge-triggered) or high to low (negative edge-triggered). In our work, this distinction is not so important since the design of the circuit at this level is accomplished by software.

5.1.3 Finite State Machines

To control and define the behaviour of a digital circuit that performs some (multicycle) task, or a part of it, designers make use of so-called *Finite State Machines* (FSMs). Derived from automata theory [19], an FSM is defined through a finite set of states, the accepted inputs and possible outputs and a set of transition and output functions. These functions describe how, given some inputs and a present state, the machine moves to the next set of states and which outputs are released [20]. In this work we deal with *deterministic* FSMs, where transition and output functions lead to only one possible next state and output configuration.

The implementation of FSMs in digital electronics is a good example where combinational and sequential logic work together. Thanks to the deterministic character, the present state of the machine needs to be remembered until the next transition and therefore a sequential circuit is needed. On the other hand the transition and output functions use the inputs and the stored information for computing the next state and the output configuration: hence a combination circuit is required.

FSMs are usually built following one of two models, the *Mealy* and the *Moore* machines[5]. In the first approach, the output configuration depends on the current state and current inputs. In the Moore case, the outputs depend only on the present state of the machine. Therefore, to get the new outputs, a Moore machine needs to "wait" until the new state is set.

Thanks to this "faster" behaviour, the Mealy machine seems to be superior to the Moore one. However in a *synchronous* circuit, such as our projects, if inputs changes inside a clock cycle, the outputs will change as well, provoking unwanted consequences. Usually this is solved by placing registers which record the output configuration at the beginning of every clock cycle, keeping their value constant along the whole period.

5.2 VHDL

With the growing complexity of digital circuits, it is more and more difficult to design circuits transistor by transistor, or even by logic blocks. To be more efficient, engineers started using *Hardware Description Languages* (HDLs) to design circuits. HDLs are used to describe the behaviour of the circuit (a more abstract level), without going to the low level of physically placing and routing of components. The VHSIC[6] Hardware Description Language (VHDL) is, together with Verilog, one of the most used languages in digital design. Although it was initially developed for simulation purposes, with time it has been equipped with characteristics useful for circuit synthesis as well. These languages have a syntax similar to that of more ordinary programming language, such as C. One of the fundamental differences regards instructions. In VHDL there are indeed two types of instructions:

- **Sequential instructions** behave in the same way as a classic programming language, since they are executed in the order they are written (sequentially[7]). In VHDL these instructions can be found only inside a structure called `process` and they are used to describe both combinational and sequential circuits.
- **Concurrent instructions** describe only combinational logic and therefore gates. They are activated every time a signal in their right part (which represents the inputs) changes value.

Some of the most used types of signals in the VHDL language are `std_logic`, `std_logic_vector`, `signed` and `unsigned`. The `std_logic` type models a one-bit signal, so the '0', '1' and 'Z' (*high impedance*, typically used to model tristate drivers). The other three types instead are array of `std_logic`, whose length is defined by the user. While the `std_logic_vector` type is used to model signal

[5] Although in computational theory there is an equivalence principle between Mealy and Moore machines, in the hardware implementation this is not always the case, mainly because of different latency times [21].

[6] Acronym that stands for *Very High Speed Integrated Circuit*.

[7] Do not confuse sequential instructions with sequential circuits! Although sequential circuits can be described only using the `process` structure, sequential instructions can describe combinational logic as well.

paths as wide as the length of the array, the `signed`and the `unsigned`types represent integers with or without sign, respectively. For more information, see Ref. [22].

Using HDLs, a designer avoids to spend time on gate-related details and can concentrate more on the functions the circuit needs to deliver. All the placements and routing of the components are taken care of by electronic design automation (EDA) software.

Once the code describing the circuit has been written, a circuit needs to be created. Several steps follow (simplified):

- **Analysis and Simulation**: the syntax is checked and a simulation can be run. This process is based only on the code, so no delays or other physical characteristics are taken into account. Only a verification of the algorithm implemented can be carried out. The simulation is usually performed including the design in a so called *test bench*. This is a VHDL file which does not represent a synthesisable circuit, but it is rather a tool for testing the designed circuit recreating different signal configurations and scenarios. It is also possible to read data from text files and write the output on disk.

- **Synthesis**: the code is translated into electronic blocks, such as registers, gates, flip-flops, related to a specific hardware implementation. However the level of devices specification is still pretty rough (so called register-transfer level, RTL).

- **Place and Route**: a netlist including all the devices used and their routing is produced from the RTL design. After this step, timing simulations can be run, since the physical characteristics of the components used are included in their description. In this last part constraints on area and performance are taken into account.

5.3 FPGA

A *Field-Programmable Gate Array* is an integrated circuit which can be configured by the user from a hardware point of view by, e.g., activating/deactivating logic gates and interconnections, setting the clock frequency. The internal structure of the FPGA is a repetition of a basic block which provides logic gates, clocks routing, flip-flops and memory elements. A dense network connects all these blocks together. Usually, as it happens for Altera and Xilinx, two FPGAs manufacturers, these basic blocks contain memory elements, such as registers, simple adders and *logic units*. The latter feature usually 4 or 6 inputs and its internal scheme is able, using multiplexers and storing the truth table, to reproduce boolean functions. There are then other blocks in an FPGA whose design however is an *Intellectual Property* (IP) of the manufacturer. The internal structure is usually kept secret and they need to be instantiated in the code according to the manufacturer's instructions. The use of this kind of blocks

inside a VHDL code makes the code *not* portable among different FPGA[8]. In our work, we avoided the use of IP components.

As for general digital circuits, the design is described using *HDL code*, such as *VHDL* or *Verilog*, which will be then handled by a software, called *synthesizer*, producing a *configuration bitstream*. This stream will be loaded onto the chip, configuring the internal blocks of the FPGA. Most of the FPGA products uses either *SRAM* or *FLASH* memories to store this configuration in the circuit. While in both cases the FPGA can be programmed many times, with SRAM we need an external nonvolatile memory to hold the configuration in case of powering off. Other types of FPGA, such as *fuse*, *antifuse*, *PROM* FPGAs, are instead one-time programmable.

Intel and Altera

Due to Intel's acquisition of Altera in 2015 [23], Altera products have been renamed Intel: however most of the users still knows these products under the brand Altera. Therefore in this document we use only the name Altera.

[8]Sometimes an IP component is available only in a given family of FPGAs: in this case the portability is therefore difficult to achieve even for products made by the same company. Examples of families are Spartan and Virtex by Xilinx and Arria and Stratix by Altera.

Chapter 6

DRS4 Readout

6.1 DRS4 Management

In order to manage the functioning of the DRS4, a finite state machine (FSM) has been implemented, where for each state a configuration of signals drives the DRS4 and the receiving part of the FPGA. The FSM is implemented as a *Mealy machine with resynchronization registers* [21]: once the machine is activated by an input or an internal signal change, the future state is computed. The present state and the signal configuration is updated at the next clock cycle. Everything works (or at least should work) *synchronously*, therefore also the input signals (especially those generated internally by other blocks in the firmware) should work synchronously with the FSM. This approach ensures there will be no hazards and no spurious transitions at outputs due to transitions in the input signals during a clock cycle. The drawbacks are the use of more registers and the latency of (at most) 1 clock cycle in releasing the new signal configuration. In Figure 6.4 a diagram of the FSM is shown.

6.2 DRS4 Problems

A couple of problems about the DRS4 surfaced during the design of the procedures for its management:

- After intense scrutinising of the datasheet and an email to Stefan Ritt (developer of DRS4), it became clear that it is not possible to read out some of the channels while simultaneously sampling the others. As reported by Stefan Ritt in a private communication, although on the datasheet a procedure for reading out a single channel while sampling in another one is described, "a bug in the silicon" prevents us from using this approach. A careful observer could point out that, according to Figure 6.1, in the sampling channels the input and the output are connected while we are sampling in a given cell (`WRITE` and `READ` are both "1" at the same time), damaging the stored value. However, in another private communication with Stefan Ritt, it was made clear that at

the output of each channel there is a buffer which prevents this unwanted behaviour.

- As can be seen in Figure 6.1, each cell contains a buffer, which is used for sending the stored value to the output. Each buffer is characterized by an offset, which is different from cell to cell. Therefore a correction of the acquired data is necessary. The idea is to measure this offset for each cell and build a table (*offset table*) which will be loaded into the FPGA and used for correction during data read out. The generation of such a table can be handled more easily by an external computer. The FPGA will be triggered by this computer, it will readout the data from the DRS4 (see section 6.8.5 for more information) and send them to the computer, where an average of multiple measurements and the table building can be done. Then the final data can be sent to the FPGA as part of the configuration registers.

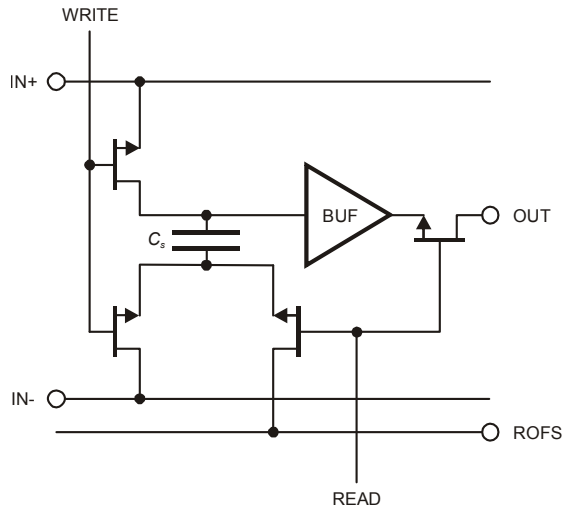


Figure 6.1: Simplified circuit of a single DRS4 cell. After enabling `WRITE`, the capacitor C_s is charged storing the sampled value. Activating `READ` this data will be then sent through the buffer `BUF` to the output. At the channel output a buffer prevents from distorting the internal sampling due to simultaneous writing and reading. The mentioned buffer is not shown here, neither it is mentioned in the datasheet.

6.3 Optimization of the DRS4 Readout Procedure

The DRS4 provides two different options for readout. In the *Full Readout Mode* all 1024 cells are read out consecutively. Assuming a parallel readout of all channels at a frequency of 33 MHz, this mode would require about 33 µs for the complete data readout. This would mean a too large dead time for the experiments we are aiming for. The second way, offered by the DRS4, is the so-called *Region Of Interest Mode* (ROI): in this case a trigger signal provided by an external circuit stops the sampling operation when needed and only the

portion of cells which contains the interesting data is read out (see Figure 6.2). Assuming we want to read out 100 cells (which corresponds to a time span of 100 ns) at 33 MHz, we would need only around 4 µs. As can be seen, the dead time introduced by this mode is a factor of 10 shorter than in the previous one. A deeper analysis of the ROI will be discussed in section 6.8.6. What needs to be taken into account in any of the modes is the fact that the trigger signal `trg` has to be generated within 1 µs in order to avoid overwriting the portion of data we want to read out.

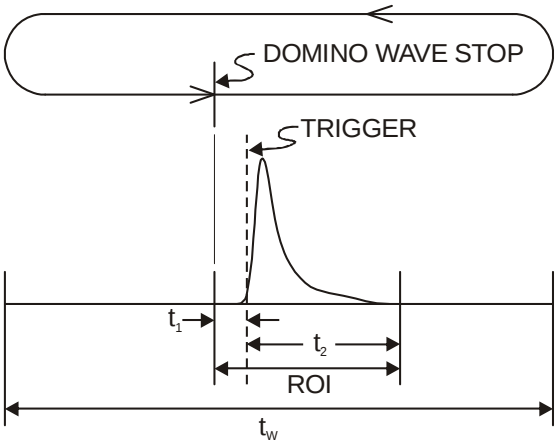


Figure 6.2: Region Of Interest (ROI) mode: t_w is the time needed by the internal domino wave to complete the whole chain (1024 ns if we are sampling at 1 GHz), t_1 is the portion of the signal we want record, which occurs before the trigger generation point, t_2 is the part covered by the signal we are interested in. For more information see sections 6.3-6.8.6.

6.4 Internal Clock Frequency

Choosing a clock frequency can be challenging in an FPGA: having many different clock domains can complicate the synthesis of a circuit for many reasons. In order to choose the proper clock frequency for the FSM circuit we need to understand what blocks the FSM is exchanging data with and what the already implemented clock frequencies are. On the one-hand side the FSM needs to drive the DRS4 and the ADC and receive data from them: according to the two datasheets, the maximum usable frequencies are 33 MHz for the DRS4 and 50 MHz for the ADC. Since those two need to be synchronised, a common clock generator would be ideal. Other already existing clocks in the FPGA have frequencies of 50-100-200 MHz: this would mean that the easiest way is to use a 25 MHz clock for driving DRS4. On the other hand the FSM receives data from the trigger generation branch, which will be driven (according to the barrel code) at 100 MHz. Since signals between the trigger branch and the FSM need to be synchronous, this would imply a clock frequency for the FSM of 100 MHz. Although this value complicates the calculations of limits in the FSM internal counting procedures and state transitions, it allows the

FSM to be more compatible with surrounding blocks and to compute the next state within one 100 MHz clock cycle (10 ns) instead of one 25 MHz clock cycle (40 ns). Summarising:

- The FSM internal clock `CLK` will run at 100 MHz;
- The clock for the DRS4 `SRCLK` generated by the FSM will run at 25 MHz.

Later in this thesis, when referring to clock cycles we will write "`CLK` cycles" or "`SRCLK` cycles", respectively, meaning 100 MHz-clock cycles given by the `CLK` signal and 25 MHz-clock cycles given by the `SRCLK` signal.

6.5 Data Sampling

6.5.1 Timing

The sampling of the data from the DRS4 is done using an AD9252 ADC by *Analog Devices*. An important fact to remember is *when* the ADC can sample the output signal from the DRS4. According to the DRS4 datasheet [9] the optimal instant for sampling is right before the next value starts to appear at the output pin. As shown in Figure 6.3 [9], after the address bits `A3-A0` have been set for exposing the data of a cell, there is a delay of $t_o = 10\,\text{ns}$ before the output voltage starts to be affected. Then at $t_{clk} = 40\,\text{ns}$[1] the output voltage reaches the new value: at this point a new `SRCLK` pulse will shift the Read Shift Register[2], exposing the new data. However, thanks to the internal delay, the old value will stay constant for a time $t_o = 10\,\text{ns}$.

As the datasheet suggests, in order to avoid degrading the DRS4 linearity, the sampling of the (old) data should be done right before ($\approx 2\,\text{ns}$) the end of the extra time t_o: in our implementation this corresponds to 48 ns after the `SRCLK` pulse which refers to that sample or 8 ns after the next `SRCLK` pulse. A block providing a controllable delay needs therefore to be implemented in order to obtain the wanted delay between the `SRCLK` signal and the signal driving the ADC sampling.

6.5.2 Enabling Signals

How do we actually start the sampling of data from the DRS4 and of the `SROUT` bits sent to the FPGA? Well, if we stay inside this FSM (and we do!), it remains a mystery. The FSM has however been made responsible for *enabling* the sampling procedures:

- The procedure which controls the ADC sampling is already implemented in the *barrel firmware* and it should be compatible for the readout of DRS4 data. The FSM is going to drive this procedure through the signal `ADC_EN`: it rises when the present state becomes `OFFREAD` or `READOUT`, while it goes to 0 one `CLK` cycle after the present state becomes `FILL`. While the rising part should fit all our needs, the falling part could present some problems:

[1] In this case t_{clk} is the clock period of the `SRCLK`, which is running at 25 MHz.
[2] More information about these operations will be given in section 6.8.5 and section 6.8.6.

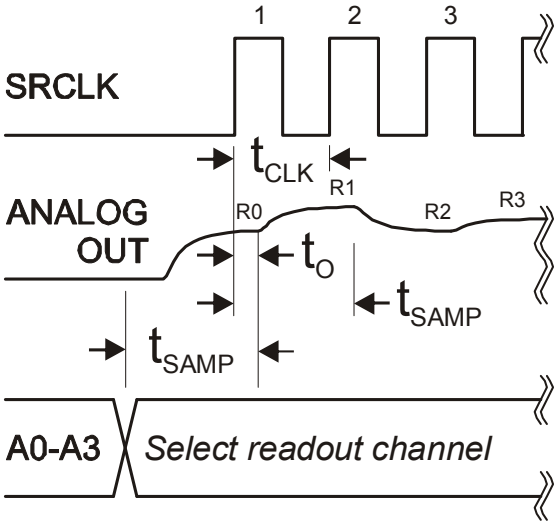


Figure 6.3: Sampling timing reference for DRS4 output data. Due to internal delay, data from a cell of the DRS4 needs some time before it starts to appear at the output pin. The ADC sampling needs therefore to be synchronised, remembering this additional delay, as explained in the text.

indeed the output data is available one SRCLK cycle after the machine goes to FILL, *not* after one CLK cycle. Extending the duration of this signal should not be however a problem, since we just need to increase the upper threshold for going back to FILL. Another way could be to increase the threshold by one SRCLK cycle, while skipping the sampling of the last data, so setting ADC_EN to zero at the same time causes the present state to change[3].

- The sampling of SROUT is managed in the same way presented above: the signal SROUT_EN is used to enable the sampling of the SROUT pin. It rises with the rise of RSRLOAD and it falls when the present state switches to READOUT.

The enabling processes are *not* extremely reliable and fully ready to be used. This is not due to a lack of will, but rather because the sampling procedures are in part unknown, therefore the exact behaviour[4] cannot be taken into account in shaping the enabling processes. The time evolution of ADC_EN and SROUT_EN are reported in the figures regarding the different signal configurations.

6.6 Operations - Briefly

This is a very brief introduction to the FSM states and transitions: a deeper analysis is carried out in section 6.8.

[3] If this last idea will be implemented, ADC_EN becomes quite similar to D_V. Therefore maybe we could decide to use directly D_V, without managing ADC_EN.

[4] For example: if there is an internal finite state machine which delays the effects of the enabling signal by one clock cycle, or if everything happens in one clock cycle, etc.

Following the diagram depicted in Figure 6.4, after the electronics has been turned on, a reset signal should be sent to the FSM in order to reset the DRS4 and the FSM itself. Once the reset procedure has been completed, the machine passes some time in FILL (see section 6.8.4) in order to obtain some data samples in the DRS4. After that the FSM moves to IDLE (see section 6.8.3), where it stays until it receives something, such as a trigger from the trigger branch in the firmware or a dummy trigger from the external computer for the generation of the offset table. One CLK cycle after it has received something, the FSM moves to the next state carrying out the

- **Offset Readout Procedure** after receiving off_trg = 1: states OFFWRAP, SHIFTREAD and OFFREAD are accessed in this order (see section 6.8.5);
- **Readout Procedure** after receiving trg = 1: states WRAPREAD, GETLOC and READOUT are accessed in this order (see section 6.8.6);
- **Reset Procedure** after receiving rst = 1, which leads the FSM to move to RESET (section 6.8.2);

If the machine is in one of the "busy" states (OFFWRAP, SHIFTREAD, OFFREAD, WRAPREAD, GETLOC, READOUT) when a reset signal rst is received, the whole procedure is completed first and then, once the FSM returned to FILL, the reset procedure is activated (rst_flag turned to 1 when rst activated).

Behaviour of BUSY and D_V

- BUSY is set to 1 for the entire duration of FILL, OFFWRAP, SHIFTREAD, OFFREAD, WRAPREAD, GETLOC, READOUT and RESET;
- D_V goes to 1 one CLK cycle before the start of OFFREAD and READOUT and returns to 0 when the FSM returns to FILL: however because of the DRS4 internal delays, the last data will be ready the next SRCLK cycle.

This topic will be revisited in connection with the description of all interested states.

Back to IDLE After the FSM is back from READOUT or OFFREAD or RESET, there are no available data in the DRS4: the stored data refer to the old trace recorded before starting those procedures, since in the meantime DWRITE has been turned off. The FSM has therefore to wait for a time t_1 (as described in section 6.3 and section 6.8.6) before to accept any new trg signal. Considering the off_trg signal instead, we need to wait for 1024 ns, in order to fill all the cells. This *waiting-for-filling* has been implemented in different parts of the FSM. The basic idea is to continue the operations done in IDLE but keeping BUSY = 1. Check section 6.8 for more information.

6.7 Address Bits A3-A0

There are some features to remember when working with address bits A3-A0:

1. When a change in A3-A0 happens, the change of other signals[5] needs to be delayed for a minimum period of time, which typically is 5 ns. In the current implementation we have introduced one CLK cycle of delay between all signals which are affected by this. However the physical routing inside the FPGA and outside on the board can cancel the effect introduced by this extra CLK cycle: therefore a study on the characteristic delay of the various tracks is needed.

2. When the FSM is in IDLE, RESET, FILL, WRAPREAD or OFFWRAPstate, no A3-A0 configuration is explicitly specified in the datasheet. During these periods the DRS4 is recording data (except in RESET) and therefore no output needs to be connected and no internal registers need to be addressed. Therefore two possibilities occur:

   - A3-A0 = ''1111'': all the outputs are disabled. However this configuration is also called *standby* and it is used in the *standby mode* explained in the datasheet, where nothing is (apparently) recorded. This seems suspicious and an email from Stefan Ritt confirms that many internal circuits are *disabled* in this mode.
   - A3-A0 = ''1010'': this is called *transparent mode*, where the input signals are recorded and at the same time are applied to the output. We do not need this last feature, however this configuration seems to be the only one, which allows to sample input signals. We did not receive get any feedback about its use by Stefan Ritt in his email.
   - A3-A0 = ''0000'': in this mode channel 0 is sent to MUXOUT. The use of this mode has been suggested by Stefan Ritt in the email cited previously: connecting a channel to MUXOUT does not seem to alter the sampled value of the input signal, since an extra buffer[6] sits between the channel output cells and MUXOUT, avoiding what is described in section 6.2. This mode has been implemented for IDLE, RESET, FILL, OFFWRAP and WRAPREAD states.

[5] Mostly SRCLK and RSRLOAD, but other signals can be affected as well: check each section and the figures about signal configuration for more information. The DRS4 datasheet has obviously all the information needed.

[6] No extra buffers are reported in Ref. [9].

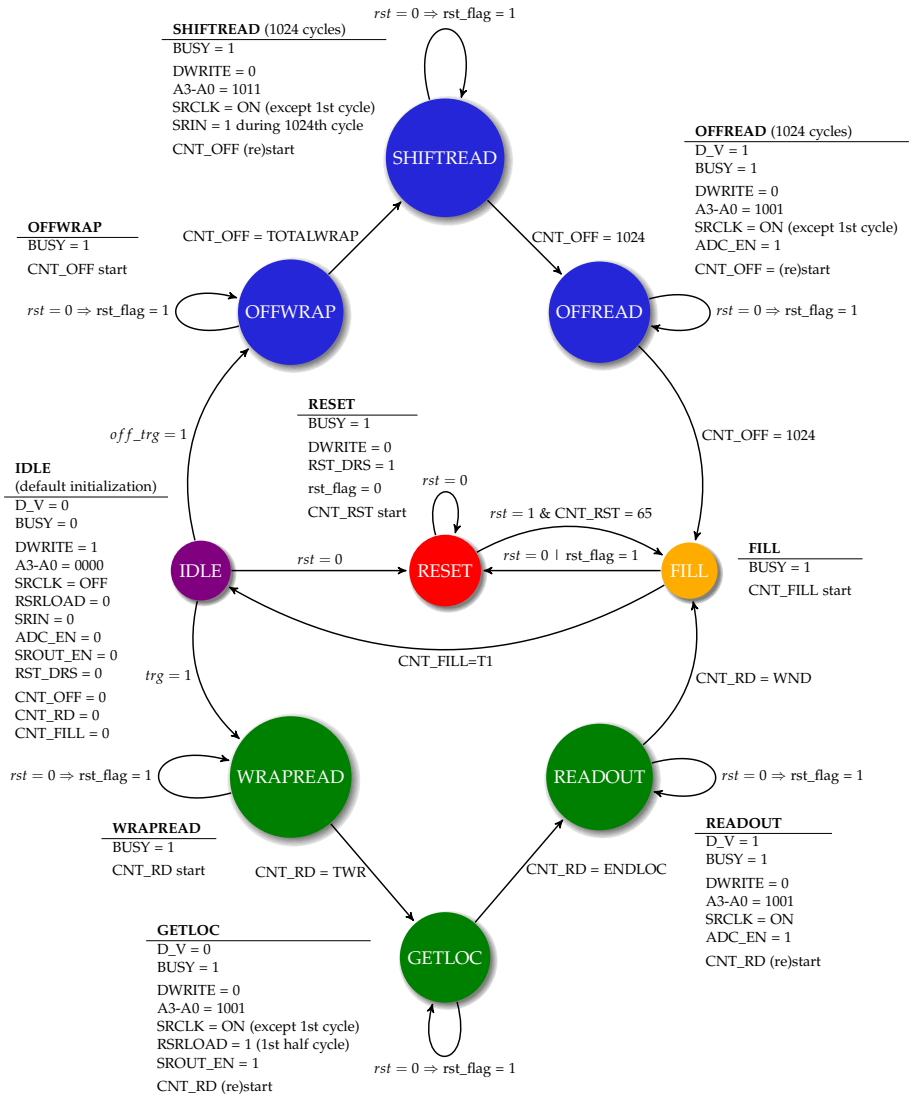


Figure 6.4: Finite State Machine for the management of the DRS readout. The (assumed) initial state is `RESET` (in red). The two main procedures are highlighted with different colors: the *offset readout* is depicted in blue, while the *data readout* procedure is green. The term "cycle" refers (usually) to a `SRCLK` cycle. The values for transitions reported in this picture are an approximation of the real ones: the correct values can be found in the corresponding VHDL code. Signals reported in *italics* are inputs to the circuit. The signal configuration of each state is reported in a table close to the associated node. However, to reduce the length of the tables, only the signals with a different value with respect to the one in the `IDLE` (*default*) configuration are reported in each table.

6.8 DRS4 FSM Implementation

6.8.1 VHDL Libraries Used

This is the list of libraries used in `readout_fsm.vhd`:

- `library ieee;`
- `ieee.std_logic_1164.all;`
- `ieee.numeric_std.all;`

6.8.2 RESET

In the current implementation the FSM has an *active low* reset, so when `rst` has been turned to 0, the machine goes to state `RESET` and, thanks to the counter `CNT_RST`, it remains there for 66 `CLK` cycles: this time is needed for resetting, bit by bit, first the CONFIG register (`A3-A0 = 1100`) and then the WRITE SHIFT register (`A3-A0 = 1101`) of the DRS4. At the same time the FSM states and output signals are reset, avoiding the FSM to stay or end up in so-called *illegal states*. Once `rst` has been released, after 66 `CLK` cycles the machine therefore will go to `FILL`. At the activation of the circuit, thanks to an external signal `rst` the machine will access the `RESET` state in order to reset the DRS4 and itself. In Figure 6.5 a time progression of signals driving the DRS4 is shown.

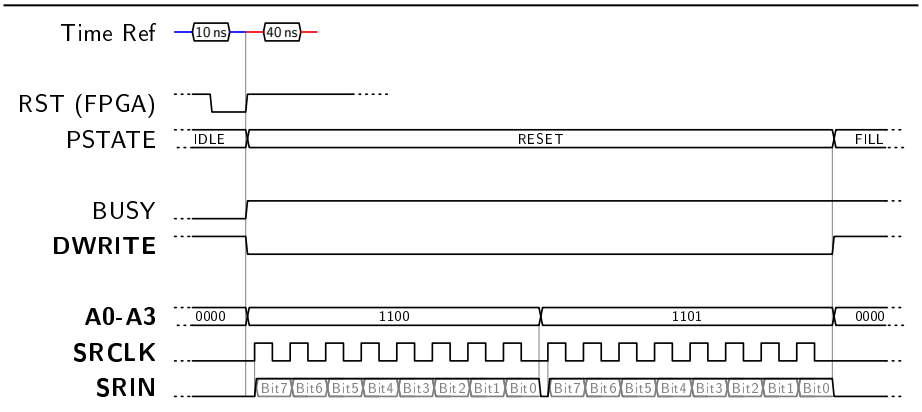


Figure 6.5: Signal configuration for the `RESET` state. Signals marked in bold are those sent to the DRS4.

In the data sheet a reset procedure is described: here a *negative* pulse, which has to last at least 10 ns, is sent to the DRS4 in order to reset internal registers and to start the domino wave. According to information recovered through the DRS4 forum [24], this procedure should be avoided, because of possible side-effects.

6.8.3 IDLE

In this state the machine is waiting for a trigger signal that initiates one of the following:

- `off_trg` = 1 will start the procedure for the readout of the (assumed) empty DRS4 for the creation or update of the offset table (go to `OFFWRAP`);
- `trg` = 1 will begin preparing for the data readout of DRS4 (go to `WRAPREAD`);
- `rst` = 0 resetting process (go to `RESET`);

In an initial design of the FSM, `rst_flag` = 1 was checked after being back to `IDLE`. In the most recent implementation instead, this check is done after the FSM is back to `FILL`, since there's no need to take samples if the next step would be resetting the DRS4.

6.8.4 FILL

As mentioned in section 6.6 "After the FSM is back from `READOUT`, `OFFREAD` or `RESET` there are no available data in the DRS4". Therefore the DRS4 needs some time to record new data in order to be prepared for a new readout.

In this condition of an empty buffer, the FSM could face two uncomfortable situations: receiving a `off_trg` signal or a `trg` signal. In the first case, the problem is handled by the `OFFWRAP` state (explained in section 6.8.5). When instead a `trg` is issued, the DRS4 does not need to have samples in all the cells at that moment. The minimum requirement is to have enough data to cover the part of trace requested before the `trg` signal[7] (check section 6.8.6 for more information). Therefore the FSM passes a very short time in `FILL` and then it will switch back to `IDLE`.

In case of `rst` = 0 while in `FILL`, the machine will go directly to `RESET`. The resetting process can be activated also by the presence of `rst_flag` = 1: this flag is set to 1 when a `rst` = 0 happens while the FSM is in one of the following states: `OFFWRAP`, `SHIFTREAD`, `OFFREAD`, `WRAPREAD`, `GETLOC`, `READOUT`. If such an event takes place while the FSM is in one of these states, the entire current procedure is first completed and then, after the FSM goes back to `FILL`, `rst_flag` is checked and the transition to `RESET` will happen.

The expression "entire current procedure" refers to the whole readout procedure. In this implementation there are 2 readout procedures:

- **Offset Readout** consisting of `OFFWRAP`, `SHIFTREAD` and `OFFREAD` states;
- **Data Readout** consisting of `WRAPREAD`, `GETLOC` and `READOUT` states.

As an example: if `trg` = 0 occurs while the machine is in the `WRAPREAD` state, `rst_flag` is immediately set to 1 and the procedure continues, first spending the required time in `WRAPREAD`, then moving to `GETLOC` and, when that is done, to `READOUT`. After the `READOUT` state tasks have been fulfilled, the FSM moves to `FILL`: here `rst_flag` is checked and since it is set to 1, the machine moves to `RESET`.

[7] This is what is called t_1 in Figure 6.2 and also in section 6.8.6.

6.8.5 Offset Readout

The procedure for the readout of the cells offset is activated by setting `off_trg` to "1" and it is carried out following the Full Readout Mode already mentioned previously in section 6.3. In this mode it is needed to load a "1" value in the first bit of the Read Shift Register: then this "1" is moved along the register, revealing at the output the value stored in each cell. Before starting the readout procedure however, we need to be sure that the capacitor arrays contain *no data*.

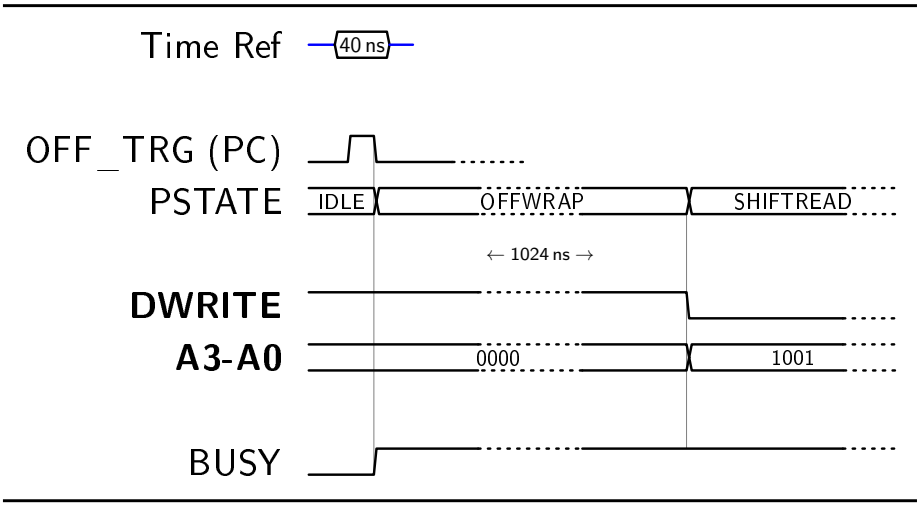


Figure 6.6: Signal configuration for the first part of the offset readout. Signals marked in bold are those sent to the DRS4. Following the PSTATE evolution: after receiving an `off_trg`, `BUSY` turns 1 while `DWRITE` remains active in order to continue the sampling. `A3-A0` does not change value. This configuration persists for about 1024 ns: after that it turns to `SHIFTREAD`. For more information see the text.

As discussed in section 6.6, after a data readout[8] or a reset procedure has been carried out, the DRS4 cells contain old samples of traces, which could disrupt the values needed for the *offset table*. Therefore the DRS4 needs to take another set of data, simply running a-sort-of `IDLE` state while no events (hopefully) happen in the detector. This is obtained entering the state `OFFWRAP` and remaining there until all the capacitors content has been overwritten. The signal configuration of `OFFWRAP` is quite similar to the one of `IDLE` and it is shown in Figure 6.6. The only difference concerns `BUSY`, which is set to 1 to signal the impossibility of accepting other triggers, and the activation of `CNT_OFF`, which changes the FSM state to `SHIFTREAD` after reaching the `TOTALWRAP` value. For the DRS4 readout, the FSM performs a full readout of the capacitor arrays, as mentioned in the beginning, and therefore we have first the `SHIFTREAD` state and then, after 1024 `SRCLK` cycles, the state named `OFFREAD`.

In Figure 6.7 the signal development is shown. The transitions between the three states are regulated according to the internal counter `CNT_OFF`, which is

[8] In this case with *readout* we mean both the *offset* and the *data* readout procedures.

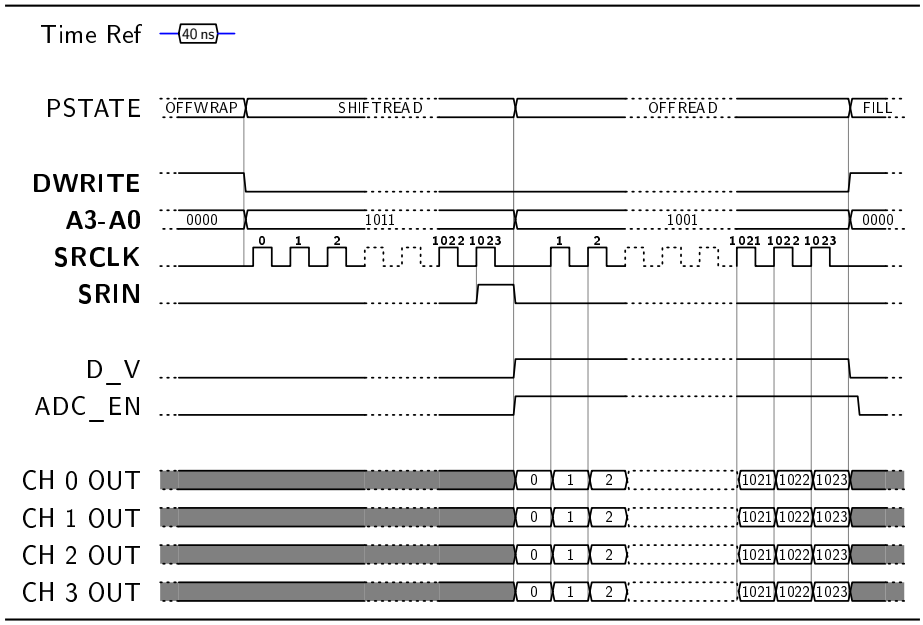


Figure 6.7: Signal configuration for the offset readout. Signals marked in bold are those sent to the DRS4. The left part of the picture depicts the `SHIFTREAD` state: `A3-A0` changes values 1 `CLK` cycle before `SRCLK`, therefore a delay of 10 ns takes place. This is in agreement with the requirement of at least 5 ns between these two changes.

increased by one each `CLK` cycle, therefore

- For the transition to `SHIFTREAD`, `TOTALWRAP` is about

$$\frac{1024\,\text{ns}}{10\,\text{ns}} = 102.4\,\texttt{CLK}\ \text{cycleslong}$$

  The actual value used in the VHDL code is 103 since we want to be sure to overwrite all the 1024 cells.

- For going to `OFFREAD` and for internal signal changes instead, the limits are about 4 times the amount of `SRCLK` cycles needed.

One `CLK` cycle after the FSM enters `SHIFTREAD`, `SRCLK` is turned on. In the last `SRCLK` cycle before going to `OFFREAD`, `SRIN` is brought to 1: this step sets a 1 in the first position of the Read Shift Register. After having moved to `OFFREAD`, `SRCLK` is turned off for the first `SRCLK` cycle.

As already described in section 6.5.2, `ADC_EN` rises when the FSM goes to `OFFREAD` and goes to 0 one `CLK` cycle after the machine is back in `FILL`. `BUSY` is set to 1 for all the duration of the procedure, while `D_V` goes to 1 one `CLK` cycle before the start of `OFFREAD` and returns to 0 when the FSM returns to `IDLE`.

Differently from the datasheet, an extra `SRCLK` pulse is missing in the end of `OFFREAD`: this is due to the fact that this pulse is needed for starting another readout from the first cell. This is not needed in our case.

If a `rst` signal is sent to the FSM while it is in `OFFWRAP`, `SHIFTREAD` or `OFFREAD`, the `rst_flag` will be set to "1" and the operation will continue un-

changed. Once the readout is over and the FSM is back in FILL, in the following CLK cycle it will move to the RESET state (as explained in section 6.8.4).

6.8.6 Data Readout

The readout of the data is split into three parts. The already explained *Region of Interest Mode* is used. After getting a "1" in trg, the machine enters the WRAPREAD state: here the DRS4 is left running for a time

$$t_{WR} = t_w - t_1 - t_T - t_{SROUT} \tag{6.1}$$

where (see Figure 6.2 for a representation of some of these times),

- t_w is the time needed by the internal domino wave to complete the whole chain (1024 ns if we are sampling at 1 GHz);
- t_1 is the portion of signal which comes before the trigger generation point and that we are interested to record as well;
- t_T is the time needed by the trigger branch to generate the trg signal with respect the actual time it is referring to;
- t_{SROUT} is constant and equal to 10 ns. This extra-time is needed in order to read out the value of the position in the capacitor array from where we start the data readout.

The result will be set in TWR, appropriately scaled according to the internal clock frequency of the FSM. At the moment the idea is to use constant values for t_1 and t_T (set by the central system in the configuration register) or to send them as input signals (send by the trigger logic). In the first case both of those values need to be estimated through simulations, calculations or tests. In the current implementation, the signals are initialised to some value just for simulations purposes.

Right before leaving WRAPREAD the sampling in the DRS4 needs to be stopped: this is done by setting DWRITE to 0. In Figure 6.8 the change in the signal configuration is reported.

Using the internal counter CNT_RD, the transition to GETLOC is managed according to the value of TWR. Following Figure 6.9, a pulse in RSRLOAD dumps the domino wave chain into the Read Shift Register, assigning a "1" to the register position where the sampling has been stopped. In this state *no* data from the DRS4 are saved: the only purpose of GETLOC is to obtain the stopping position of the domino wave in the DRS4. This reference is needed for establishing a time reference for the data and for subtracting the correct offset value of the sampled cells. The position value is obtained recording what the SROUT pin of the DRS4 sends out during the first 10 SRCLK cycles: the position value is encoded in 10 bits as a *binary unsigned* value. The first bit sent out is the *Most Significant Bit*, followed then by the other bits, down to until the *Least Significant Bit*.

Although we have already equipped our FSM with the necessary code for managing GETLOC, here we report the three basic solutions which were taken into consideration:

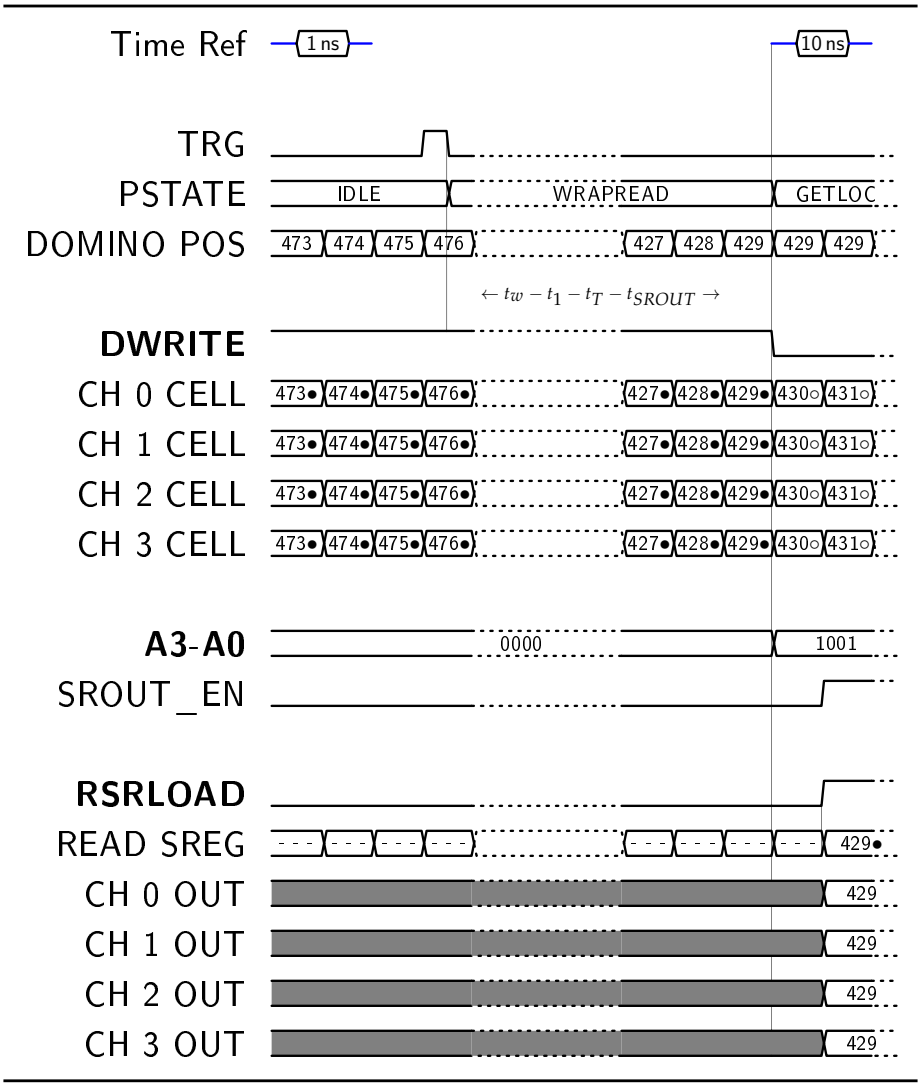


Figure 6.8: Signal configuration for the wrapping procedure. Signals marked in bold are those sent to the DRS4. Filled dots indicate that a value has been written inside that register or cell. Empty dots instead refers to the fact that nothing has been written. The line named "domino position" describes the progress of the domino wave along the capacitor chain

1. Storing the first 10 data sets and after we've gotten all the `SROUT` stream, we can start with the computation procedures. This approach requires some storage space on the FPGA.

2. Rejecting the first 10 data sets. This can be done because we can start reading out the DRS4 from 10 cells before the required one: the only needed change concerns the value of `TWR`, whose calculation is reported in Equation 6.1 with $t_{SROUT} = 10\,\text{ns}$ representing these extra 10 cells.

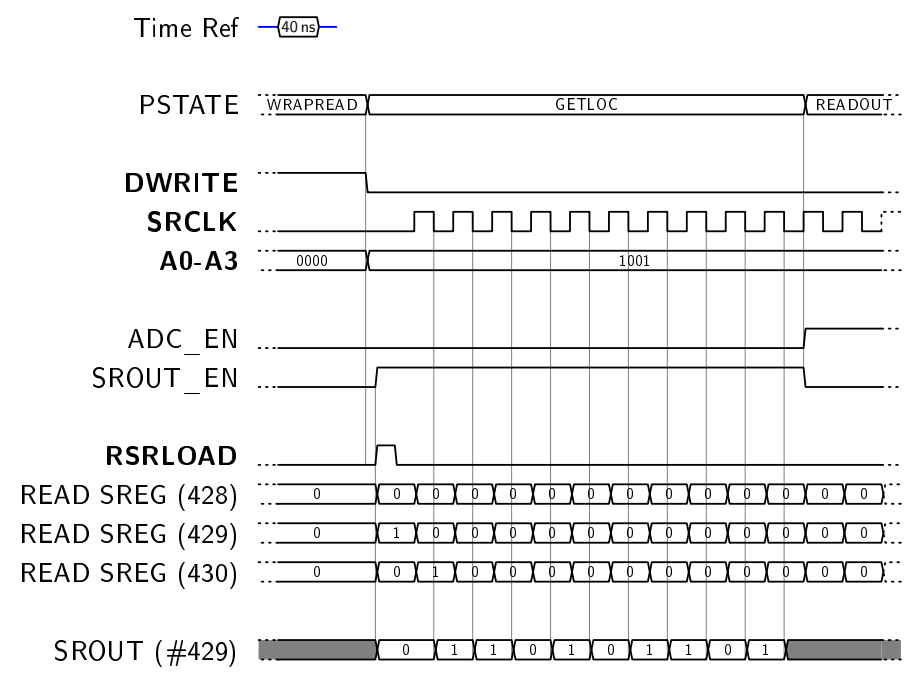


Figure 6.9: Signal configuration for receiving the bits of the first position in the Read Shift Register from SROUT. Signals marked in bold are those sent to the DRS4. The lines named READ SREG (#) show how the "1" dumped by RSRLOAD is pushed forward by SRCLK pulses. In the SROUT line instead we can see that the first output bit is the Most Significant Bit of the stream representing the position value.

3. Inserting a new state between WRAPREAD and READOUT which is used just for collecting SROUT. A similar calculation for TWR as in the previous point needs to be carried out.

The current implementation is based on a combination of the last two points listed above.

Figure 6.9 illustrates what happens during GETLOC: the bits from SROUT are released according to the *falling edges* of SRCLK. In the current implementation there is *no* an *actual* readout and storing procedure of SROUT: in the meantime a signal called SROUT_EN is set to 1 while the readout is needed. More precisely, following what is written in section 6.5.2, SROUT_EN rises together with RSRLOAD one CLK cycle after the machine moves to GETLOC. Then it is 0 when the FSM turns to READOUT.

After the initial position has been recovered the FSM is set to READOUT: the signal configuration for this state is shown in Figure 6.10. Pulses in SRCLK shift the "1", initially dumped by RSRLOAD during GETLOC, through the cell range of interest. The value of this range is stored in WND and the current idea is to set this value in the configuration register as well as the values for t_1 and t_T.

ADC_EN rises when the machine is set to READOUT and is set to 0 one CLK cycle after the FSM is back in FILL, as also described in section 6.5.2. BUSY is

set to 1 for all the duration of the procedure, while D_V goes to 1 one CLK cycle before the start of READOUT and returns to 0 when the FSM returns to FILL.

TWR and WND Values of TWR and WND used by the circuit cannot be simply set in time units, but they need to be scaled and adapted to the FSM CLK frequency ($f = 100\,\text{MHz}$ or $T = 10\,\text{ns}$ in the current implementation):

- TWR is calculated according to Equation 6.1 and scaled by a factor 10. As example, consider $t_w = 1024\,\text{ns}$, $t_1 = 40\,\text{ns}$, $t_T = 80\,\text{ns}$ and $t_{SROUT} = 10\,\text{ns}$[9]. TWR will be calculated as

  $$\texttt{TWR} = (t_w - t_1 - t_T - t_{SROUT})/10 - 1 - 1 = 87.4\ \texttt{CLK cycles}$$

  where the first -1 is due to the fact that we start one CLK cycle after trg has been raised, so 10 ns pass while trg is 1. The second -1 is due to the fact that CNT_RD starts counting from 0. In the code the actual value used is 87, since we want to *not overwrite* more cells than the necessary ones.

- WND given in ns represents the amount of cells we want to read out. Therefore a simple conversion between the 100 MHz clock domain and the 25 MHz one is enough to give the right limit: therefore for each cell we need 4 CLK cycles. As example, if we want to get 200 ns of data we need

  $$\texttt{WND} = 200 \cdot 4 - 1 = 799\ \texttt{CLK cycles}$$

  where the -1 is due to the fact that CNT_RD starts counting from 0.

[9] As already explained, t_{SROUT} is needed in order to stop the writing operations 10 cells before the original point. This 10 ns quantity is due to the fact that the domino wave advances at a *speed* of 1 cell/ns, since it is sampling at 1 GHz.

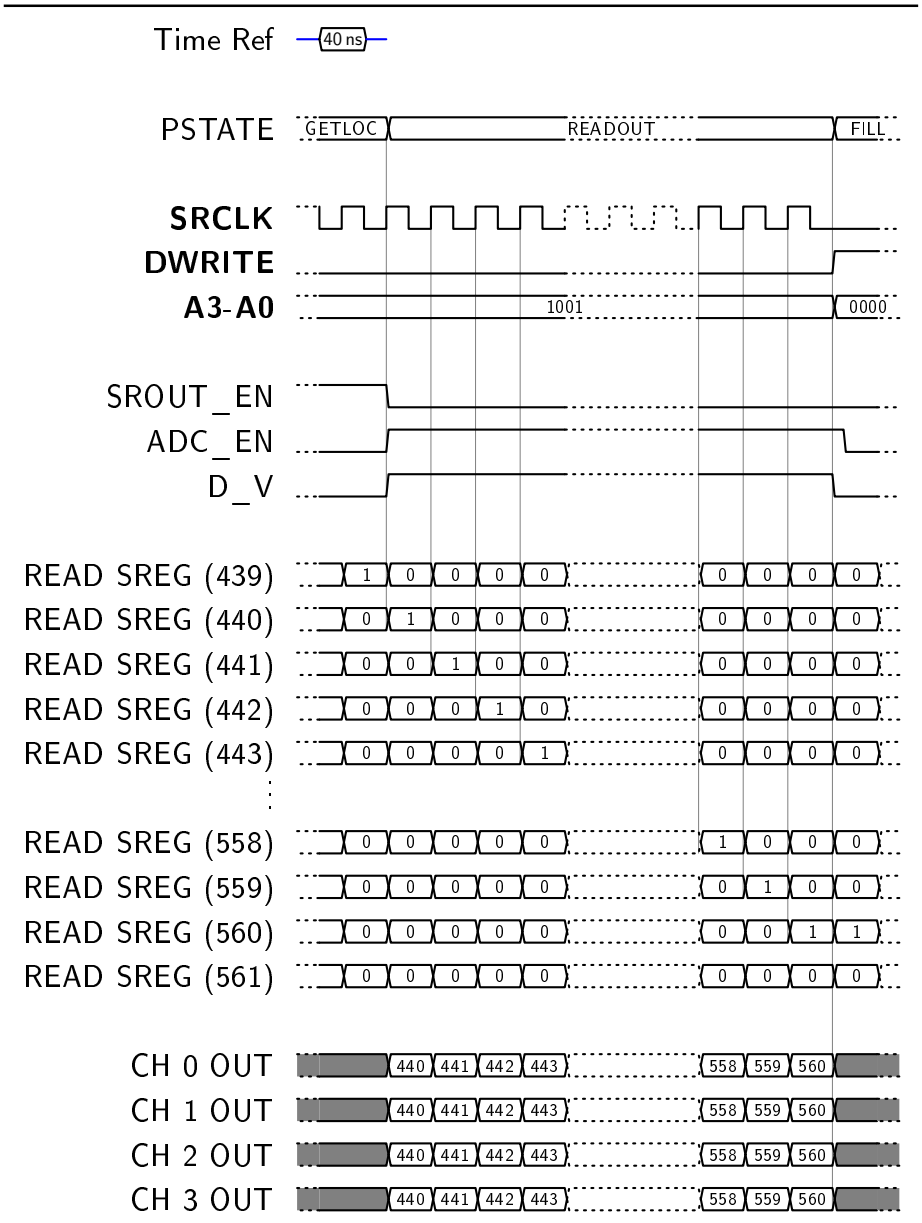


Figure 6.10: Signal configuration for the readout of the data from the DRS4. Signals marked in bold are those sent to the DRS4. The lines named READ SREG (#) show how the "1" dumped by `RSRLOAD` is pushed forward by `SRCLK` pulses. The last four lines instead show the position in the capacitor chain of the data sent out.

Chapter 7

CEPA Clock Distribution

7.1 Motivation for a New Clock Delivery Scheme

The request for a simple but still precise clock delivery system comes from the need to provide a clock reference for time stamping in the CEPA part of the CALIFA experiment. The main idea behind this proposed clock delivery protocol deals with different aspects:

- Delivering the clock signal/reference and the labels of the fronts[1] on only one line/cable: although using electric signals is possible, an optical connection would be the best solution. Using one line/cable would ensure simplicity of the delivering and to avoid ambiguity, since the labelling and the clock signal/reference are always aligned. The current clock delivering method (provided by White Rabbit receivers) involves two different cables, one carrying the clock signal and the other the label information: temperature differences or errors in the cables length can induce signals jitters and misalignment between label and clock front.

- Delivering the signal directly to the system, involving at most modules which just route the signal: in our case this means delivering directly to the Febex cards or to the Exploder. This will allow the system to time-stamp events according to the reference clock and not something that could be "contaminated" by other modules. From the signal delivered to the FPGA we can generate the main clock used in the cards, allowing to timestamp with respect the absolute reference of the WR system.

In the next sections we will first examine the electronic modules available for arranging this new distribution. Then we will show how this protocol will work and finally we will propose some ideas for implementing the delivery scheme.

[1] These numerical labels are needed to identify the corresponding clock front with respect an initial reference point. In this way we can deliver the timestamp information.

7.2 Phoswich Crystals Properties

Phoswich crystals for CEPA are made of two different materials optically coupled together: 7 cm of $LaBr_3$ and 8 cm of $LaCl_3$. The photomultiplier is placed at the end of the $LaCl_3$ crystal. These materials show interesting characteristics summarised in Table 7.1 [7]. Together with fast *decay times* and large *light yields*, these materials have a very good time resolution: as shown in Figure 5a on page 2 of [25], values of the coincidence resolving time (CRT) are in the range of **a few hundred ps**. The actual value depends on the dimension of the crystal involved. The experiments for measuring the CRT involved two crystals coupled to PMTs and a ^{22}Na source [25]. The crystal under characterization was used as STOP signal, while signals from another smaller $LaBr_3$ crystal were used as START. The CRT is related to the time resolution according to the following

$$\delta^2_{CRT} = \delta^2_{start} + \delta^2_{stop} \, .$$

Material	Light Yield (photons/keV)	Decay Time (ns)	CRT (ps)
$LaBr_3$	63	16	≈100–300
$LaCl_3$	49	28	≈200–500

Table 7.1: Main characteristics of $LaBr_3$ and $LaCl_3$ materials. CRT refers to Coincidence Resolving Time and the range reported here is more detailed in Figure 5a on page 2 of [25].

Because of these time resolution characteristics, the distribution of a precise clock reference is needed.

7.3 Analysis of the Electronics - Hardware

7.3.1 Febex3B - Blue Card

The Febex3B board is equipped with [26][2]:

- FPGA: Lattice ECP3-150. In Table 7.2 we have reported some properties of this FPGA [27]. What needs to be highlighted is the requirement for the duty cycle of the PLL input clock signal: with any clock frequency, the minimum requirement is 0.5 ns in either high and low level. The FPGA is responsible for the data processing and for driving the DRS4 chip.
- 8 Low noise amplifiers ADA4932-2 whose inputs are connected to the front-end connector where an add-on board with mechanical connectors

[2]The referenced data sheet is not the one describing the Febex3B board, but rather the Febex3A board: the only difference noticed concerns the ADC used. Documentation about the Febex3B is not available.

(e.g. lemo connectors) can be attached. These amplifiers have two differential input-output channels each [28].

- 2 ADCs: Analog Devices AD9252. With 8 channels each, they feature 50 MHz sampling frequency, 14 bit resolution and 2 V_{pp} as input voltage range [12]. The inputs are all routed from the low noise amplifiers. The outputs are connected to FPGA input pins.

- 3 oscillators Si531 family by Silabs [29]. Two of them are characterised by a frequency of 100 MHz with a supply voltage of 3.3 V, while one has a frequency of 125 MHz and a supply voltage of 2.5 V. All three are connected to some FPGA inputs.

- Front-End connector featuring:

  - 16 differential analog inputs;
  - 16 differential LVDS I/Os;
  - 4 code inputs.

- PCIe slot featuring:

  - Four serial multi-gigabit connections;
  - 8 fold trigger bus inputs/outputs (differential MLVDS);
  - Power supply.

FEATURE	VALUE
Logic Units	$\approx 149 \times 10^3$
Multipliers (18×18)	320
Memory	
Embedded Memory	6850 kbit
Distributed RAM	303 kbit
SERDES	
Amount	4
Channel Frequency	150 Mbps to 3.2 Gbps
PLL	
Amount	10
Input Clock high	min. 0.5 ns
Input Clock low	min. 0.5 ns
Input Frequency range	2–500 MHz
Output Frequency range	4–500 MHz

Table 7.2: Main characteristics of the FPGA Lattice ECP3-150 [27].

7.3.2 SFP Module - Red Card

The SFP module provides connections between the external world and the back-plane of the Febex crate. While it is connected to the crate through a PCIe connector (see section 7.3.3), on the front it features:

- 2 Optical connectors, one of which is needed for sending data to the MBS pc (PEXOR card);
- 1 ribbon cable connector with 26 lines which is used for connecting the SFP module to the EXPLODER.

7.3.3 Crate

The crate is responsible for providing power and a simple connection between cards, especially between the Febex cards and the SFP module (for trigger and data delivery). The cards are allocated in PCIe slots, although no PCIe protocols or communications are involved. It features:

- A daisy chain between all the Febex cards using a bi-directional 1.6 Gbps connection for readout and control via the GOSIP protocol [4];
- 8 MLVDS lines on the back for trigger signal exchange: however further information about how the connections are arranged is not available.

7.3.4 DRS4 Add-on Board

This add-on board is designed for the CEPA part of the CALIFA detector. It is needed in order to provide a higher sampling frequency for the fast signals coming from phoswich crystals. It features:

- 8 front-end LEMO connectors and connection to a Febex board through the Febex front-end connector;
- DRS4 chip: it is a switched capacitor array circuit with 8 channels, each of them having an array of 1024 capacitors [9]. It allows to sample 8 channels up to 5 GHz (for CEPA 1 GHz has been chosen) and its input voltage range is $1\,V_{pp}$. It is driven by the FPGA sitting on the Febex board, receiving start, stop and configuration signals. However the most important of these signals for our analysis is the clock responsible for the generation of the internal sampling wave.
- Signal splitting:
  - From main input the signal is split in order to send one signal to the DRS4 chip and another to an ADC on the Febex board for triggering purposes. The latter is fed into an integrator before being sent to the Febex card (already implemented).
  - The signal to the DRS4 will then be split another time but in different weights into a gamma and a proton branch (*not* implemented yet).

7.3.5 Exploder

The Exploder is central in managing trigger signals. It communicates with the Febex crate through a ribbon cable connection to the SFP module. It is also connected to the TRIXOR card in the MBS pc: this board manages the triggers sent by the Exploder. It features [30]:

- FPGA Xilinx Spartan 6 (XC6SLX150T-3CSG484): 150k Logic Blocks and 4.824 Mbit Block RAM;
- Four optical fiber transceivers (2 Gbps SFP);
- Daughter Board (this is *not* the White Rabbit extension board) [30]:
  - 8 LVDS inputs, 8 LVDS outputs, 8 ECL outputs, 8 TTL and NIM inputs, 8 TTL or NIM outputs;
  - 2 MLVDS connectors with 8 differential I/O pairs, LCDisplay.

7.3.6 Cables

7.3.6.1 Ribbon Cable

It has been noticed that transmission on ribbon cables is highly affected by the material which surrounds the cable, especially if no shielding is provided. If the material changes (for example moving an object closer to the cable), the signal is shifted in time, even shifts in the ns range are possible. Therefore employing this type of cable should be avoided if we want to reach a sub-ns precision for our clock signal, without excessive calibration work.

7.3.6.2 Optical Fiber

As also suggested in the Nustar DAQ TDR [8], signals outside modules/systems should be optical. This is due to the fact that they are not affected by electromagnetic interference, have no problems with impedance mismatch, and avoid to influence on the grounding of separate systems. Employing an optical cable therefore would be the best choice in order to achieve sub-ns precision.

7.3.7 EXTRA Hardware: SPEC and FMC cards

The combination of a SPEC card and a FMC card by Seven Solutions is what in the WR terminology is called a *node*. This system is able to

- Interface with a WR switch through an optical connection (SPEC card);
- Keep itself synchronised with the WR clock [31];
- Deliver a clock signal (FMC card) which is synchronised with the WR clock [32].

The SPEC card can interface itself with another card through a PCIe connector, from which it receives also its power. Among all the components, this card features a Xilinx Spartan 6 (XC6SLX-45T/100T) FPGA and a SFP cage (the transceiver is not provided) in which the WR signal is delivered. It is able to achieve sub-ns time accuracy. The FCM card instead sits directly on the SPEC card and provides components in order to interface with other modules. All the signal processing is done in the FPGA on the SPEC card.

7.4 Proposal for New Clock Delivery

In order to deliver on one line both the label information and a signal, which can serve as clock reference, we need to send two kinds of information at the same time. Talking about the signal responsible for generating the clock, we take a look at how FPGAs generate a clock signal starting from a given input clock. Although it is not an absolute rule, producers such as Altera, Xilinx and Lattice claim that the internal PLLs generate the output clock by locking on the rising edges of the input clock signal. However usually manufacturers then impose some restrictions on the allowed duty cycle[3] of the input clock[4], with Altera being the most restrictive (range 40–60%) while Lattice just requires the presence of a falling edge. Xilinx is seen to be in the middle, with an allowed duty cycle range of 25–75%.

The label information instead is a 64 bit word and it can be sent as 1 bit/cycle. Considering the previous discussion of the duty cycle, the "1" or "0" information of the single bit sent per cycle can be encoded with a different duty cycle of the input signal. The generation of this kind of signal can be easily done inside an FPGA using a counter and a clock with a higher frequency than the target signal. In Figure 7.1 this new concept is shown: the idea is to feed the FPGA with the "INPUT CLK", which then is split and sent to a

- **PLL**: this component can generate a clock signal which drives the FPGA logic based on an input clock. Moreover the PLL guarantees that the output signal is *phase locked* with respect to the input: this means that there is a *fixed* relation (constant offset) between the input and output phases, *not* that the phase is equal to 0.

- **Sampling branch**: the input signal is sampled using a clock gated latch scheme, where the clock is derived (or even the same) from the PLL output. Since each single bit of a 64 bit word is encoded in one separate input clock cycle, the clock for sampling needs to have a higher frequency than the input one. In this way for each input clock cycle, and therefore each bit, we will obtain a bunch of samples, some equal to "0" and others equal to "1". All the sampled values inside one input clock cycle are then summed together. The result is then compared with a threshold

[3]The *duty cycle* of a periodic digital signal is intended here as the ratio between the time the signal is high ("1") and the total period time.

[4]For Altera the performances of the Max 10 family are reported, which are the same of the Arria II FPGA contained in the VETAR2 module, while for Xilinx we checked the Virtex 4 family. In the case of Lattice instead, the data about the ECP3 family are shown.

which will determine if in that clock cycle a "1" or a "0" bit has been transmitted. The threshold value needs to be chosen according to the two values of the duty cycle of INPUT CLK and to the frequency at which INPUT CLK is sampled. In Figure 7.2 a sequence of signals involved in the single bit recovery is shown.

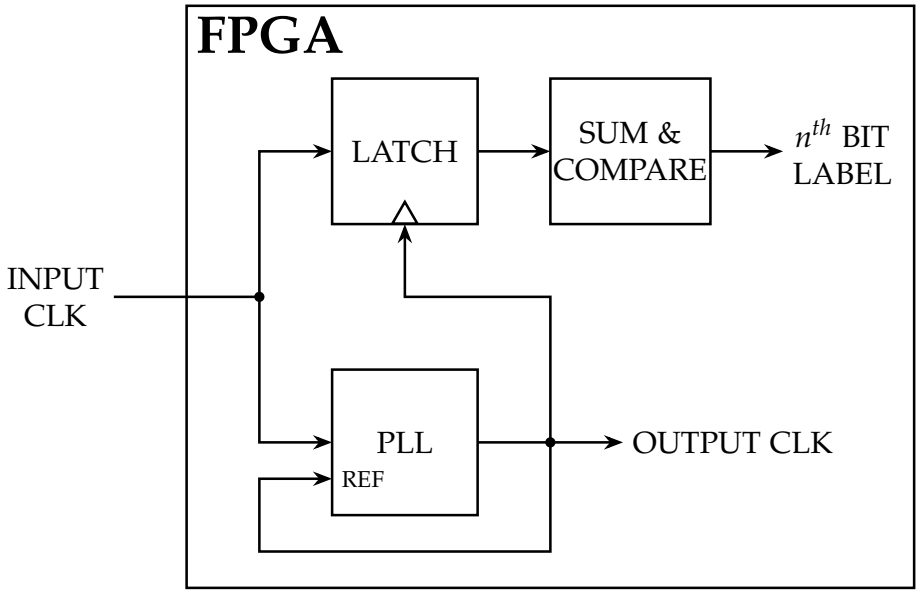


Figure 7.1: The signal from a WR node (or any other module) is injected into the FPGA, where it is addressed (i) to a PLL, which will be responsible for generating a clock signal for the FPGA logic, and (ii) to a logic circuit responsible for "sampling" the input signal INPUT CLK and therefore recover the sent bit of the label. The sampling is done using a clock gated latch. The samples are then collected and summed in order to determine the value of the encoded bit. The clock used can come directly from the PLL OUTPUT CLK or it can be derived from that one. The OUTPUT CLK is also used by the PLL as a reference clock during its internal elaborations.

In the above explanation the labels sent to the FPGA are described as 64 bit words: in the real implementation this will be probably changed in longer words in order to include error correction capabilities and DC-balance characteristics.

7.4.1 Threshold, Quiet Margins and Duty Cycle

The capability of the sampling branch to determine which value has been sent depends on the

- **Number of samples available**, which is proportional to the ratio between the OUTPUT CLK frequency and the INPUT CLK frequency;
- **Duty cycle values** for the "1" and the "0" clock cycle of the INPUT CLK.

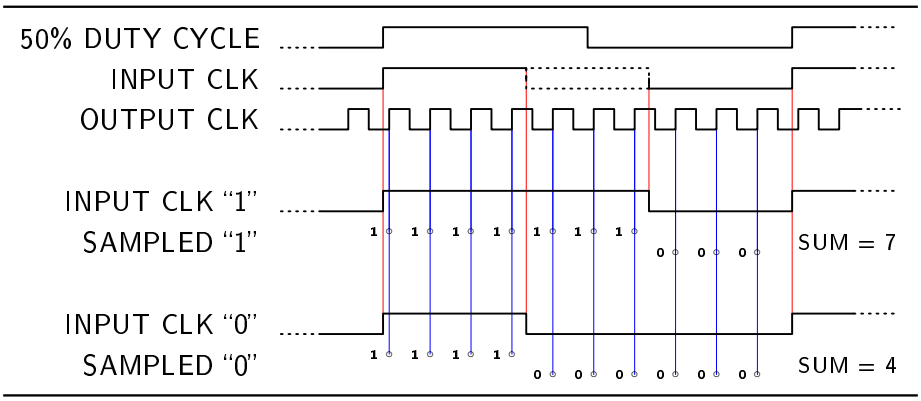


Figure 7.2: Examples of signals involved in the proposed clock delivery scheme. The input signal (INPUT CLK) to the PLL is a signal with a fixed frequency when considering the rising edges, while the falling edges are shifted according to the need of transmitting a "1" or a "0". In this case the PLL OUTPUT CLK has a frequency 10 times higher than the input signal. OUTPUT CLK and INPUT CLK are phase locked thanks to the PLL, which means that the shift in time is kept constant. The existence of this shift is due to the internal signal path of the PLL and reaching the clock input of the latch, compared to the INPUT CLK reaching the signal input of the latch. The two SUM terms are sums of the sampled values of the INPUT CLK in the two different signal shapes.

Knowing these two parameters we can decide on the value for the *threshold* that is needed for determining which value has been sent. The two duty cycles correspond to two different *sums* of the samples: the threshold can be therefore easily computed as the average point between the two obtained values. Now comes the challenging part: the two sums (or duty cycles) have some *margins* (which we call *quiet* margins) between them and the calculated threshold. Having wide margins is a requirement in order to recognise the two encoded values more easily. To achieve this property, a high difference between the two duty cycles, together with a high sampling frequency, is needed.

Although the calculation of the threshold can be "hard-coded" as circuit constant parameter, a *dynamic determination at runtime* can be implemented: a *min-of-max* and *max-of-min filter* respectively for the long and short pulses can be used to adapt the threshold to the circuit performances.

Edges Jittering Concerning the correct recognition of the encoded value, we cannot avoid to face the possible problem due to the jittering of the INPUT CLK edges with respect the sampling clock generated from the OUTPUT CLK. What can happen in the case of jittering edges (especially considering the falling edge that encode the label bit) is that the sum of the samples in one input clock cycle results to be different than the designed value. This problem can be mitigated (if not altogether avoided) by ensuring to have *sufficiently* large quiet margins: these margins need to be adapted to the time width of the jitter and therefore to the amount of involved samples. Although a study of the jitter in the FPGA, which will be employed, is necessary. We could guess that a jitter larger than 1 sample is difficult to experience. In this situation a

difference of at least 3 between the short-pulse SUM and the long-pulse SUM will ensure to calculate the right transmitted bit.

7.5 Proposed Electronic Implementations for the New Clock Delivery Scheme

The content of this section is based heavily on emails that the Chalmers group exchanged with the Munich group. The ideas were elaborated mostly by the Munich group with some exceptions (such as the gray rabbit picture). The images here reported (which were drawn mostly by colleagues in Munich) have not been changed with respect those attached to the emails.

After presenting the current state of clock delivery in section 7.5.1, a design for future implementation of a clock delivery system is explained.

7.5.1 Present State

The configuration described here is currently used for the *barrel* part of CALIFA and has been developed by the research group at the *Technische Universität München* (TUM) in Munich, Germany. A 20 MHz clock for timestamping is generated inside the EXPLODER and then delivered to all the FEBEX cards through the ribbon cable connector of the *SFP card - Red Card*: each Febex then derives its own timestamp from this clock (see Figure 7.3) by counting from a global reset. The clock from a WR switch is received by the PEXARIA card. This card then talks with the Exploder (which send triggers). These triggers are then used for synchronising the local timestamps with the WR timestamp in an event builder. Synchronisation of several Exploder modules has been developed.

From the 20 MHz clock received by the Febex, a ≈500 kHz clock is generated for the DRS4 chip. The PLL in the DRS4 chip helps in reducing problems with possible clock jitter. Attention needs to be paid to checking if the signal received by the DRS4 is well delivered through connectors and metal traces.

Using this scheme a problem of counting mismatches can arise if the FPGA in a Febex misses, for one cycle, the 20 MHz reference clock, leading the local timestamps to differ from each other.

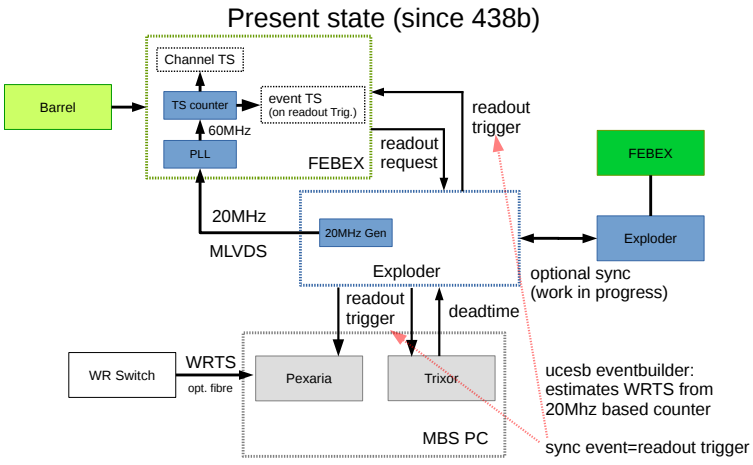


Figure 7.3: In the current configuration used for the barrel, the signal from a WR switch is received by the PEXARIA card in the MBS PC. The Exploder communicates with the Febex cards, receiving and elaborating trigger signals, and with the PEXARIA card, transmitting trigger events. The Exploder generates a 20 MHz clock signal which is delivered to all the Febex cards: this signal is then used in an internal timestamp counter to assign an initial timestamp reference. The official timestamp will be computed by software in the event builder. For CEPA this clock delivery scheme needs to take care of the DRS4 add-on board: in this case a ≈500 kHz clock will be generated inside the Febex FPGA from the original 20 MHz clock from the Exploder. [Information from private communication]

7.5.2 Proposed Scheme

An electronic implementation which exploits the new clock delivery scheme is shown in Figure 7.4[5]. A WR receiver such as PEXARIA card, VETAR module or Exploder (together with its WR add-on board), can be used for generating the 25 MHz[6] clock with the encoded label in it. This signal is then delivered on one cable to the Exploder, used in the experimental setup and from there to the Febex cards, always on one cable. The Exploder uses this signal for time referencing the triggers. Meanwhile the Lattice FPGA on each Febex board decodes the label and generates the internal clock as described in section 7.4. Also the clock for the DRS4 is generated from the same input clock.

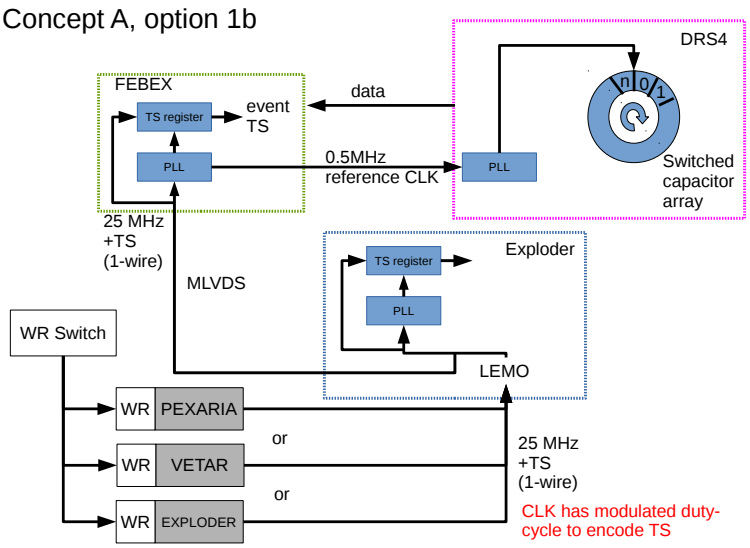


Figure 7.4: This electronic implementation exploits the proposed clock delivery scheme which use just one cable to deliver both clock and label information. The signal from a WR switch is first received and elaborated from a WR receiver such as a PEXARIA card or a VETAR module or an Exploder (together with its WR add-on board). This receiver generates the clock signal with the encoded label and sends it to the Exploder used in the experimental setup. From here the signal will be sent to the Febex cards, where the Lattice FPGA will generate their internal clock and the clock for the DRS, while recovering the label. Inside the Exploder instead the clock will be used for time referencing the triggers. [Based on private communications]

[5]The figure has been obtained by modifying a schematic drawn by colleagues in Munich.
[6]25 MHz is a convenient ratio of both 125 MHz and 50/100/200 MHz.

Chapter 8

DPTC - Difference Predicted Trace Compression

Conventions — Most/Least Significant Bits

- If not explicitly stated otherwise, in this chapter the *Most Significant Bit (MSB)* is always on the left side of a string, picture etc. The *Least Significant Bit (LSB)* is always on the right side.

- `std_logic_vector`, `signed` and `unsigned` (see section 5.2) are built with *descending array range* (example: 15 `downto` 0). Therefore the MSB is the bit with the highest range value, while the LSB is the one with the lowest range value.

8.1 Motivation

A particle detector does not only require good timing and/or energy resolution, excellent readout electronics and superb clock distribution. As crucial is the transfer and the storage of the large amount of data generated by the digitization and processing of the signals. On the one-hand side, increasing the transfer rate usually requires investments in purchasing systems capable to deliver faster data transmissions. Considering the needed storage, buying or accessing larger servers might solve the problem.

However there is a cheaper solution which can ease the pressure: *data compression*. If implemented as software running on a PC, it would compress data, which have already been sent from the signal processing unit, therefore there would not be less demand on the transfer rate. To be effective on both sides, an implementation on the FPGA, where usually the signal processing takes place, is needed.

In this chapter we will first show some ideas for compressing data. Then the implementation on the FPGA of this approach, which exhibits the best compression, will be carried out using VHDL code.

8.2 Common Ground

First of all, knowing the carefulness with which data is usually collected in physics, the compression schemes we are going to present are all *lossless*, meaning that they allow a full recovery of the original data information.

Well-known compression algorithms such as the *Huffman encoding* algorithm base their operations on symbol occurrence frequency [33]. The encoded information is recovered from data that are already all available before the start of the compression.

In our case we require instead the compression to work *online*, using data released directly by modules responsible of the digitization of detector signals. This implies that we do not have full knowledge of the occurrence frequency of symbols.

Our compression schemes are going to work with data from *traces* and this fact has two consequences: the *frequency of symbols* is biased and we can *predict* data trends, especially along a pulse rising and falling parts.

It is important to understand that the two compression algorithms, explained in the following sections, do not work on data coming directly from the digitization device, for example an ADC. To be able to compress data effectively, the input data values of these algorithms are exploiting the following two basic characteristics of the signals.

8.2.1 Frequency of Symbols

The first thing to notice is that in a trace each value is usually not very distant from the surrounding ones: therefore encoding the *difference* between one sample and the previous one strongly skews the spectrum of values to encode. This characteristic is particularly visible in each single trace, where the values are found lying along the so-called *baseline* value and also, to a lesser degree, in slow pulses. Thus, developing compression schemes which encode *smaller* values with *fewer* (*Huffman coding* style) bits should reduce a lot the occupation due to flat traces significantly.

8.2.2 The Predictor

Another characteristic that can be exploited is the *trend* of the samples. This approach could be seen as a sort of *prediction* algorithm, another set of methods used in data compression [33].

Assume Considering a pulse, we get a first set of data where the differences between one data and the previous one are all positive (rising part), and a second part where instead all the differences are negative (falling part). These trends can be used for storing the *difference of two consecutive differences*: in this way we are going to store smaller values than otherwise. We call *predictor* the part of circuit or code which implements this calculation.

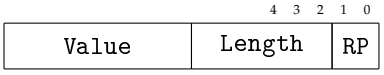
The predictor needs to be activated once a rising or a falling sequence has been detected. This is done by monitoring the last n differences of each value: when all these are positive or all negative, the predictor is activated. Once a

trend is interrupted, the procedure goes back to the usual one, namely the calculation of the difference between one sample and the previous one. The choice of the value of n needs to be *not* too *large*, otherwise we fail to take advantage of the beginning of rising or falling parts, *nor* too *small*, otherwise the predictor is activated also during flat noisy periods, which actually will inflate the values to be stored.

Our implementation Our suggestion, based on empiric studies, is to activate the predictor after 3 differences have shown the same sign.

8.3 Single Word Compression

In this approach to compression, a single data value at a time is encoded and the resulting output data is organized as follows[1]:



(8.1)

where

- **RP** stands for *Recognition Pattern* (2 bits): it signals the beginning of a new encoded word. As shown in Table 8.1 it can take four possible values, three of them (`00`, `01` and `11`) are directly linked to a value (0, +1 and -1 respectively), while the fourth (`10`) represents the beginning of a new sequence which encodes a larger value.
- **Length** represents the number of bits which are used to encode the input data in `Value`. The value stored in `Length` however is not necessarily the `unsigned` version of the real length. Together with the length information, the *sign* information of the encoded data is stored as well in this part. The width of `Length` varies and depends on the number of bits needed to encode the value, which will be processed.
- **Value** contains the encoded value of the data. The resulting bits value depends on the particular algorithm applied.

Value	Meaning
`00`	Encoding 0 value
`01`	Encoding +1 value
`11`	Encoding -1 value
`10`	Encoding larger values

Table 8.1: Values of the *Recognition Pattern* `RP`. They signal the beginning of a new encoded word.

It needs to be pointed out that the representation of the output data displayed in 8.1, although it fully follows what was stated in the beginning of the chapter, it will be always turned around in the following sections. This choice allows the reader to follow more easily those picture, because of the more common left-to-right way of reading text.

In the following sections encoding algorithms, which follow the *Single Word Compression* guidelines, and possible improvements will be presented.

8.3.1 OddBits Compression

What is missing in the previous section is a procedure that, starting from an input data, writes the `Length` and `Value` parts of the block shown in 8.1. This

[1]We suppose that the reading proceeds right-to-left: this is not uncommon when dealing with CPU and other type of digital electronics. Therefore the *beginning* of the data is localized on the right side of the block and vice versa for the *end*.

is accomplished by an encoding algorithm. The encoding approach shown in this section has the peculiarity to output data symbols with an *odd* width.

Given a data value `input` characterised by a width of n bits (`n-1 downto 0`)

1. If `input` is positive (or negative), start looking for the first occurrence of 1 (or 0) from the MSB;

2. Once it has been found at position `k`, the `Length` part is filled with `k` 0s (or 1s, in case `input` is negative), while `Value` takes the part of `input` between the first occurrence of 1 (or 0). Visually the output word has the following structure

10	k 0s	input(k downto 0)

(8.2)

if `input` is positive, or

10	k 1s	input(k downto 0)

(8.3)

if `input` is negative.

From the procedure described above, it follows that the minimum width of the encoded words is 5 (`k`=1)[2]. Considering only encoded words m-bits long, with $m = 2 + 2k + 1$ (according to 8.2 and 8.3), the amount of encoded patterns available is:

$$2^{\frac{m-1}{2}}\ , \quad m \in \{5, 7, 9, 11, \dots\} \tag{8.4}$$

Considering now a maximum output width of l bits, the total number of encodable words is

$$3 + \sum_{m=5}^{l} 2^{\frac{m-1}{2}}\ , \quad m \in \{5, 7, 9, 11, \dots\} \tag{8.5}$$

where the 3 comes from the values encoded using the *Recognition Pattern* (see Table 8.1). Considering the summation term, we can substitute $m = 2 + 2k + 1$. Setting $k_{MAX} = \frac{l-3}{2}$, we can solve the equation

$$\begin{aligned}\sum_{k=1}^{\frac{l-3}{2}} 2^{\frac{2+2k+1-1}{2}} = \sum_{k=1}^{k_{MAX}} 2^{1+k} = \sum_{k=2}^{k_{MAX}+1} 2^k = \left(\sum_{k=0}^{k_{MAX}+1} 2^k\right) - 2^0 - 2^1 = \\ = \frac{1 - 2^{k_{MAX}+1+1}}{1-2} - 3 = 2^{k_{MAX}+2} - 1 - 3 = 2^{\frac{l+1}{2}} - 4\ .\end{aligned}$$

Adding the 3 of Equation 8.5 we finally obtain

$$2^{\frac{l+1}{2}} - 1\ , \quad l \in \{5, 7, 9, 11, \dots\} \tag{8.6}$$

which gives the number of encoded words using at most l bits as output width.

[2]Someone could protest and say "Why don't you consider the case `k`=0?". Well, if `k`=0 we can not have any information about the length of the value part, therefore no data can be stored in this way. In the following paragraphs this problem will appear again.

Known Problems A problem surfaces when using the algorithm as explained above together with the other encoded values described in Table 8.1. When we try to apply the algorithm above to -2 = `0b1...10` we obtain:

10	`k=0` 1s	`input(k=0 downto 0)`

10	-	0

However this conversion of -2 cannot be used since the decoder does not read information from the `Length` part, therefore it does not obtain any information about the length of the `Value` part.

A solution to this problem involves a change of the 2-bit encoding table (Table 8.1). Expressing -2 using 3 bits without using `10` as recognition pattern for it, we'll obtain a new set of encoding values shown in Table 8.2. This approach has the inconvenience of increasing the number of bits needed for encoding -1 by one.

Value	Meaning
`00`	Encoding 0 value
`01`	Encoding +1 value
`111`	Encoding -1 value
`110`	Encoding -2 value
`10`	Encoding larger values

Table 8.2: Values of the *Recognition Pattern* `RP` with a fix for the -2 value.

Advantages The main advantage of this algorithm is its simplicity in computing the encoded word: once the first 0 [1] has been discovered, we just need to dump the rest of the data bits. No mapping or other operations are required.

8.3.1.1 Decoder for OddBits Compression

In order to encode data with all the features described above, the decoder needs to

1. Read the *recognition pattern*: if it already represents a value, then the job is done. In case instead `RP = 10`, then the encoder continues to read.
2. Read the following bits until there is a change from 0 to 1 or vice versa. This is the `Length` part described in section 8.3. If this part starts with a 0, it means that a positive value has been encoded. On the contrary a 1 would denote a negative value. This feature is however not exploited explicitly by the decoder, as it will become clear in the next step. The number of bits before the change tells the decoder the length of the `Value` part.
3. Take all the bits starting right after the *recognition pattern* until the end point set using the information found in the `Length` part. The decoder then just needs to interpret these bits as a `signed` binary value.

8.3.1.2 EvenBits Compression

A compression where all words have an even length has been taken into account. However the only way we found to built such a compression required the use of a *map* in order to encode easily the input value. The fundamental approach is the same as the one explained in section 8.3.

8.3.2 Drawbacks

While developing the *Single Word Compression* approach, a number of weaknesses surfaced.

First, every time one more bit is needed for encoding the input data, the resulting sequence grows by 2 bits. This is due to the fact that both the `Length` and the `Value` patterns need to be increased by one bit each: the first one is needed for signalling that we have one more bit, the second one is the value bit itself. The encoded sequence width therefore increases always in steps of 2, no matter which encoding scheme is used. This leads to an important inflation in the encoded words width when large numbers need to be compressed. Considering the procedure shown in section 8.3.1, the maximum value of `k` which allows us to reduce the number of bits is

$$2 + k + k + 1 \leq n \Rightarrow k \leq \frac{n-3}{2} .$$

Another problem concerns the number of values which are encoded with symbols whose bit width is larger than that of the input data. To show this problem we use an example: consider a 14 bits signed input data. In order to keep the output width shorter than the input, we can use a maximum of `k` equals to 5. This indeed implies a maximum length of 13 bits for the encoded words and a coverage of an input values range of $[-64, +63]$, for a total of 128 values. While the total amount of input data is $2^{n-1} = 8192$, the total number of encodable words, having a shorter width than the input, is 127 (according to Equation 8.6). Therefore a large amount of input data words would be encoded with a larger number of bits compared to the input bit width. However, since we are interested in compressing traces data using differences, we do not expect to have large variation between two consecutive samples. Therefore smaller values are more probable than larger ones and a reduction in data occupation should still be visible.

Lastly, from simulations where we could tune the amount of noise in the data, it appeared clear that the compression ability drops if the noise raises.

Mainly for this last reason, the compression described in section 8.3 and 8.3.1 is not implemented in any VHDL code. The only compression scheme which is considered for implementation purposes is presented in section 8.4.

8.4 ChunC - Chunk Compression

The *Chunk Compression* (ChunC) scheme is a completely different approach with respect the *Single Word Compression*. In this case the compression involves a group of r data values[3] (*chunk*) at a time, where r is always the same. The procedure is quite simple: considering all the data in a chunk, first we look for the highest number of bits needed for encoding (as `signed` binary number) each data value in that chunk. The encoded data are arranged as follow:

1. First x bits are used to mark the width[4] of each encoded data value;
2. Then the encoded versions of the input values are stacked together one after the other.

The final package of bits will appear to be as follow[5]

r-th data	···	2nd data	1st data	Length

(8.7)

while representing the occupation of each part

m bits	···	m bits	m bits	x bits

(8.8)

Values range of x The value of x is determined in order to accommodate the width the largest input data, since this is the *worst-case scenario*. An observation is needed: given x, we can cover an `unsigned` value range starting from 0 up to $2^x - 1$. However in the case x encodes a 0 value, this would mean that no other bits are going to follow. Although this can be a good approach for encoding really efficiently r consecutive 0s, another way of interpreting the 0 value has been implemented[6]. Having to deal with ADCs, which feature 8 or 16 bit resolution, we can exploit the 0 value to shift the values range between 1 and 2^x: in this way an 8 bits width can be encoded using 3 bits, rather than 4, and a 16 bits width uses only 4 bits instead of 5. Therefore the encoder and the decoder just need to know that the encoded `Length` is equal to the actual length minus 1.

Advantages The ChunC method solves (almost) the problem about the width inflation of encoded words, which affects the *Single Word Compression* (see section 8.3.2). The worst case scenario for the CHUNC approach indeed happens when we need to deal with the entire input data, so that the length of each data remains the same. The resulting encoded word will have an increase in occupation with respect the initial one of just x bits: this quantity is however shared among the r data values which make up the chunk. As an example, if $r = 8$, $x = 4$ and the input width is 14 bits, the worst case encoded word will

[3] It is important to remember that this compression scheme handles the differences between two adjacent samples or the difference of two differences, as explained in section 8.2.

[4] The x-bits content is given as `unsigned` binary number.

[5] Read the block from right to left.

[6] A long sequence of 0s is however not very common, as it implies that the recorded noise is much smaller than 1 bit.

have $4 + 8 \times 14 = 116$ bits, compared to 112 bits of uncompressed data. The 4 bits more correspond to an average of $4/8 = 0.5$ bit per input data addition in the worst case.

Compression Factor Simulations on data from traces have shown compression factors up to 4: this means that for a 16-bits long sample, the average compressed value occupies 4 bits. The compression rate depends heavily on the type of traces involved. The flatter the trace is, the higher the factor becomes, because it implicates that all the differences values are close to 0.

Another factor that influences the compression rate is the level of noise in the signal. With a high noise level, the differences are not close to 0, requiring the compressor to use a larger amount of bits to encode. Figure 8.1 shows how the number of bits needed for each sample increases, following the logarithm of the noise level. In the simulation, the sampled signal was pure noise. In order to keep the compression factor high, the suggestion is to delete the part of the sample which is accounting just for the noise.

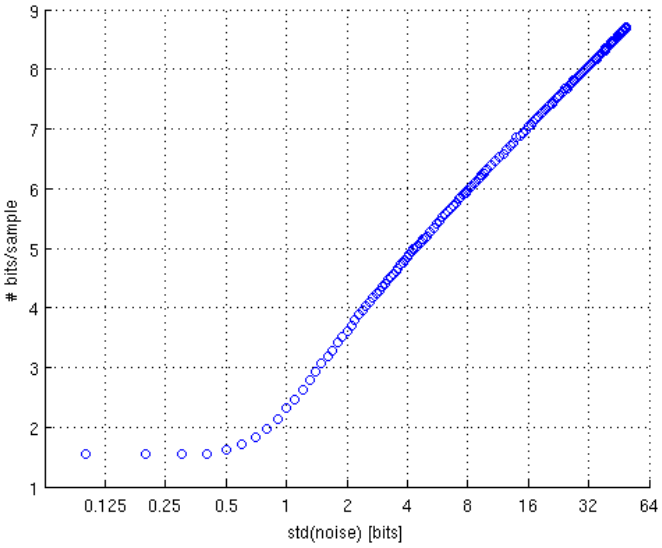


Figure 8.1: Dependence of the number of bits required for compressing a sample on the noise level. The noise is reported as standard deviation of the noise distribution.

This compression scheme has shown good properties in dealing with noisy data, and therefore we have created a VHDL implementation of it to be used (mainly) in FPGA projects. Difference Predicted Trace Compression *or* DPTC *is the name given to this implementation of ChunC.*

8.5 DPTC - Implementation on FPGA of ChunC

Implementing the ChunC compression on FPGA we tried to keep the circuit as configurable as possible. Therefore a package file (`compr_pkg.vhd`) containing constants is provided: the circuit characteristics which can be configured are the width of the input and the output words and the length of the chunk.

The design is made up of 3 modules:

1. **Difference Calculation**: the input and output data are transmitted serially and the module calculates the differences between two consecutive inputs, and applies the predictor;
2. **Encoding**: in this module the differences are encoded, while an output signal marks the beginning of a new chunk. The number used for each data is released by this module as well.
3. **Output Compression**: this module packs the output data from the *encoding* module in order to provide an output with a fixed length. A data valid signal tells when a new output is ready.

All these modules are bound together in a top level design, which therefore represents the interface to outside world. This splitting has been carried out in order to give the possibility to remove a single module or to easily replace it with another one implementing a different algorithm.

In the following sections, the design of DPTC using VHDL code is described. Although some circuit parameters can be changed by the user in order to fit his/her requirements, the present documentation will use the particular values chosen as default ones: they can be found in Table 8.3. In section 8.5.3 all configurable parameters are described in detail.

8.5.1 Language and Libraries

The circuit has been developed using VHDL code and we have tried to create a code compatible with both the VHDL-1993 and the VHDL-2008 version of the standard. Moreover we did not employ any *Intellectual Property* (IP) component from any EDA or FPGA vendors. All this means that our source code should be compatible with most software on the market, and that it can be employed for a long time (hopefully)[7].

Libraries used are all recognized as official standards:

- `library ieee;`
- `ieee.std_logic_1164.all;`
- `ieee.numeric_std.all;`
- `std.textio.all;`
- `ieee.math_real.all`.

[7]However FPGA vendors and EDA software companies (still) provide products which are not implementing completely all the new features introduced in VHDL-2008: an example is given by the *reducing boolean operators* which are not implemented in Altera Quartus Prime 16.1 [34].

In this way the code provided should be read without any problem by all the synthesis programs available on the market.

8.5.2 DPTC Module - `compression_module_chunc.vhd`

This is the top level design where the three computational modules are bound together. Figure 8.2 illustrates how these three sub-modules are connected to each other. The top layer works more as an interface to allow an easier usage of the compression circuit in another design.

The circuit has no central unit (usually implemented as a *finite state machine*) for controlling the computational behaviour, such as reset, data valid generation and initialisation of a new chunk. This is not needed as the states are very simple. Each module is fed with the input reset and clock signals, while the generation of a data valid signal at the output is managed with a pipeline which carries the input data valid in parallel to the input data. The output data is a fixed 32 bits `std_logic_vector` and the availability of a valid output data word is generated by an `and` operation between the data valid signal and a signal that tracks the filling of the output buffer.

For the management of the beginning of a new chunk, an internal counter, enabled by the presence of a positive data valid value at the right stage, keeps track of how many input values have been received.

8.5.3 Compression Package - `compr_pkg.vhd`

In this package a number of constants and functions used in various parts of the circuit are defined. Constants are used mostly for describing the width of data words and the pipeline depth. A description of each constant can be found in Table 8.3, together with default values used in this documentation and in our implementation. Two constants are marked with a † symbol: these

Constant Name	Meaning	Default Value
`input_width`	Length of input data	14
`output_width`	Length of output data	32
`ppl_depth`	Number of data in a chunk	4
`bits_width`†	Width for encoding the `Length` part	4
`accbits_width`†	Width of counter for the accumulation	5

Table 8.3: Description of the constants defined in `compr_pkg.vhd` and used for configuring the whole circuit. Constants marked with † are defined through a mathematical operation carried out automatically using previous constants. In the left column, default values, used as well in the implementation here presented, are reported. See text for more information.

are constants whose values are based on other previously defined ones.

- `bits_width` is defined as the least integer able to hold the value of the

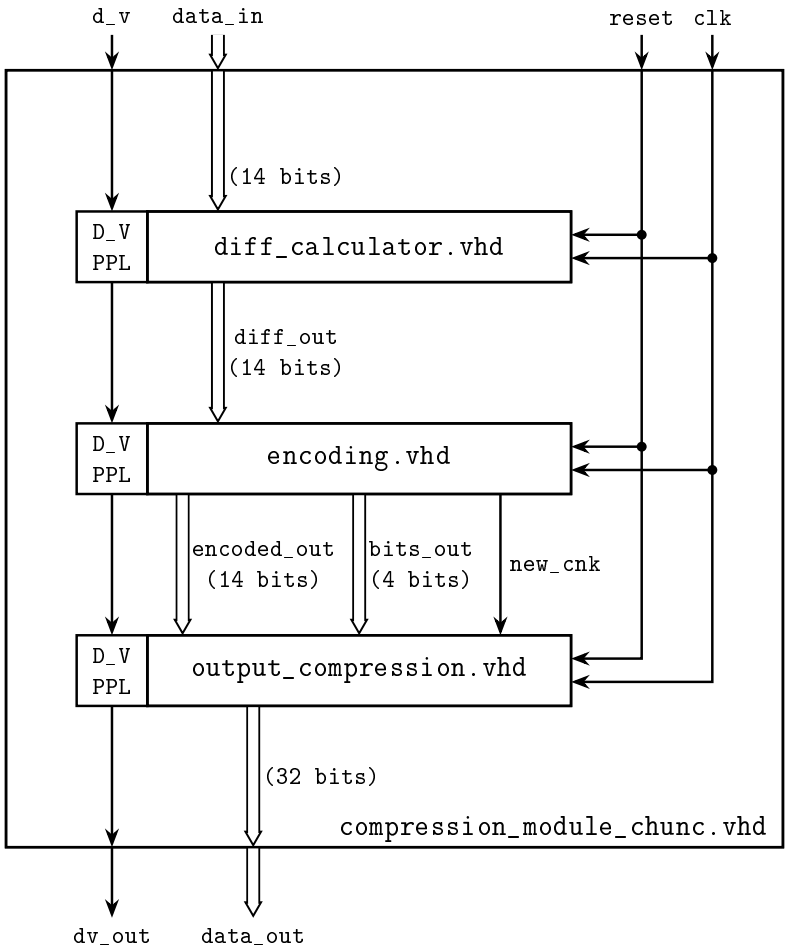


Figure 8.2: Internal structure of the compression module. This partition is not done just for documentation purposes — it exists in the VHDL code. For more details see the text.

number of bits in `input_width`

$$\begin{aligned}\texttt{bits\_width} &= \lceil \log_2(\texttt{input\_width}) \rceil \\ &= \texttt{natural(ceil(log2(real(input\_width))))}\big|_{\text{VHDL}}\end{aligned}$$

It describes the number of bits used for encoding the `Length` information at the beginning of a new chunk (see section 8.5.6).

- `accbits_width` is the least integer able to represent the number of bits in `output_width`

$$\begin{aligned}\texttt{accbits\_width} &= \lceil \log_2(\texttt{output\_width}) \rceil \\ &= \texttt{natural(ceil(log2(real(output\_width))))}\big|_{\text{VHDL}}\end{aligned}$$

This value is used for defining the width of signals used in the accumulation process in the output module (see section 8.5.7).

While the pipeline depth can be set by the user to any preferred value, the `input_width` and the `output_width` need to be carefully coordinated with the length of an internal buffer, which will be deeply described in section 8.5.7, in order to avoid to overfill the output buffer. The worst-case scenario appears when the output buffer is missing one bit for enabling the output reading and the new coming data, which accidentally needs to be described in its full width, corresponds to the beginning of a new chunk: in this case the data comes along with the bits used to identify the length of each chunk data. Therefore the width of the internal buffer needs to be equal to the sum of `input_width`, `bits_width` and `output_width-1`.

8.5.4 Effects of Data Valid and Reset Signals

The circuit behaviour can be controlled using the *data valid* `dv_in` and the *reset* `reset` (as shown in Figure 8.2). Those affect the way the circuit functions in different ways:

- `reset` is an *active low synchronous hard reset*. This means that every time its value is brought to 0 (*low active*), at the next rising edge of the clock (*synchronous*), all (*hard*) the circuit content is deleted and internal modules (such as counters and registers) are reset to some default values.
- `dv_in` is useful for telling the circuit when incoming data is available. Its value needs to be set to 1 at the same rising clock edge when the first data is presented to the input. Once there are no more data to feed, `dv_in` has to be set to 0, in order to avoid encoding useless and unwanted patterns. Once `dv_in` becomes 0, the whole compression module will be gradually emptied from data in the pipeline, so that no information gets lost. At the output stage, a `dv_out` signal notifies when output data are available to be read. This signal works in a same fashion as `dv_in`: it rises at the beginning of the clock cycle when data is provided, whereas it remains low during periods while no data is ready.

8.5.5 Difference Calculator - `diff_calculator.vhd`

This module is responsible for calculating the difference between two consecutive input values and it applies the predictor as explained in section 8.2.2. The predictor is activated after 3 differences show the same sign. The first implementation of this module does not rely on any pipelining of its internal processes, therefore when an input is provided at the beginning of a clock cycle, its related output will be ready at the end of the same clock cycle.

As shown in Figure 8.3, once a new value appears at the `data_in` input, a subtraction between it and the previous value is performed. Based on the current content of the two shift registers, the output data will be either the just-obtained difference, or the result of a subsequent difference between the initial difference and the one calculated in the previous clock cycle. As already explained in section 8.2.2, the decision is based on the sign of the previous three (initial) differences. If they are all positive or all negative, one of the two shift registers contains a '1' value in all its positions. Doing an `and` operation among

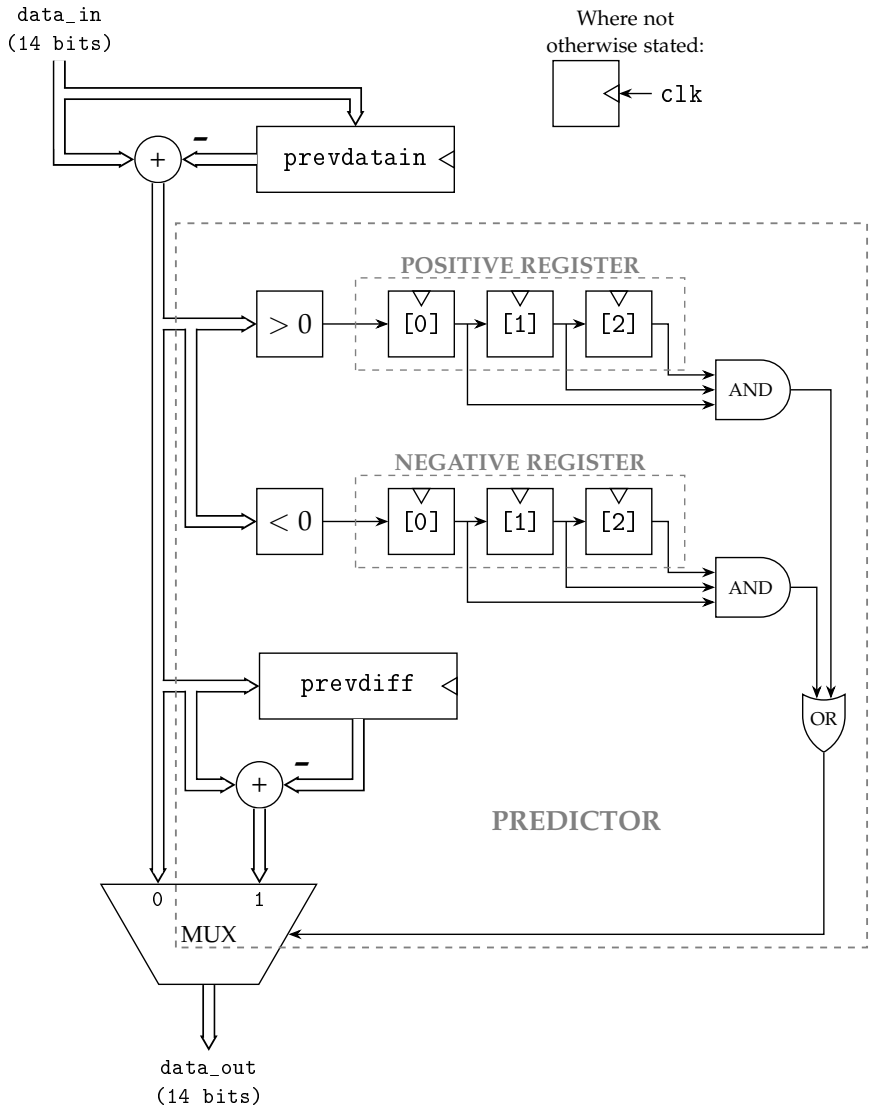


Figure 8.3: Internal structure of the difference calculator. The part of the circuit implementing the predicting feature is highlighted in grey. As can be noticed, there are no registers on the path between `data_in` and `data_out`: this means that the output is calculated within the same clock cycle as `data_in` is provided. For more details see the text.

their respective content thus reveals if one register satisfies the condition. The `or` gate activates the predictor (through the multiplexer).

The data valid signal sent to the module just passes throughout it without being sampled in any register.

8.5.6 Encoding - `encoding.vhd`

In this part of the circuit, the data released by the `diff_calculator` is analysed in order to extract information about the largest number of bits needed to express the carried value. A mask is produced from each data and an `or` of all the masks obtained in a chunk gives the pattern used for both extracting the number of needed bits and for masking the data which will be stored.

The circuit is shown in Figure 8.4. The value `data_in` is first scanned beginning from the MSB down to the LSB in what is called the `MASK GENERATOR`: the purpose is to obtain a pattern, which has a '1' bit in all the positions starting from the highest bit with a value different from the MSB (sign). In all the higher positions there is a '0' bit value. This operation can limit the maximum circuit frequency when the input data has a large length. In order to deal with input data, which have all the bits with the same value (when `data_in` encodes a 0 or -1), the final mask `active_mask` discards its own MSB and a '1' is added after its LSB. This way allows also to keep track of the sign in the encoded data: without this additional bit, the "1s" pattern in the mask would cover only the part of data which comes after the change of bit value.

Now assume we are dealing with consecutive data inside the same chunk. At every clock cycle new data comes in and a new mask is generated. The old mask therefore needs to be stored: in order to avoid to store 4 different masks and then do an `or` among all in order to find the one with more "1s", at every clock cycle we store the OR between the new mask and the result of the previous `or`, which is kept in `mask`. In this way we just need one register for the mask and one (wide) `or` gate.

The beginning of a new chunk is tracked using a counter (`CNT_CNK`): every time the counter reaches 0 the signal `new_cnk` is set to '1' for one clock cycle. This signal is both sent to the output module and used internally. Its rising is exploited for two purposes:

1. Once a new chunk starts, the accumulated value contained in `mask` must not be used for generating the next pattern. Therefore a multiplexer, fed with `new_cnk` as selector, is used for storing either the `or` result (`new_cnk` = 0) or a newly generated mask (`new_cnk` = 1).

2. Once a chunk ends, the content of the `mask` register needs to be copied in another register (`active_mask`), which will retain this value for the next 4 clock cycles.

The content of `active_mask` is used in two ways:

1. It is sent to another module which performs a summation of *set bits*[8]. The obtained information is encoded in a `std_logic_vector` signal which is `bits_width` bits long (4 bits in our implementation): other than for tracking the shift needed in preparing the output buffer (section 8.5.7.3), it is used in the `Length` part at the beginning of the chunk (section 8.5.7.2).

2. It is used to mask, through an `and` operation, the useless part of all the data being part of the same chunk. In this way, the data that arrive to

[8] A *set bit* is a bit which is set to '1'.

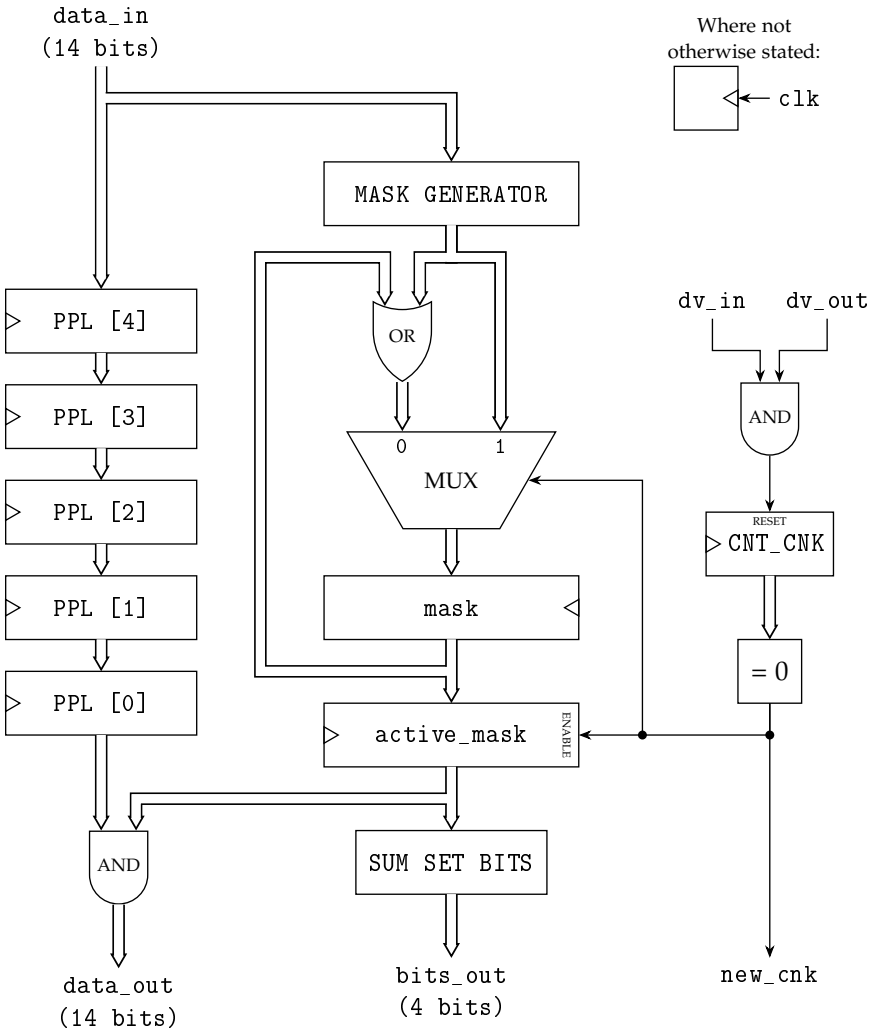


Figure 8.4: Structure of the module responsible for encoding the difference. Most of the computational part is concentrated in the central part, while on the left the data pipeline is visible. The part of circuit on the right keeps track of the start and end of each chunk. For more details see the text.

the output module are already prepared to be shifted and stored, while potential overlapping of the unused part among consecutive data will not affect the final result.

From the information reported in the above it is easy to see that we need to pipeline the input data in order to get the length information of all the data of a chunk. Indeed, while a data is scanned, the previous ones belonging to the same chunk need to be kept in the correct order before being released after having experienced 5 clock cycles inside the pipeline.

Parallel to this pipeline, the data valid signal follows a similar path in order to maintain the same delay of the input data throughout the circuit.

8.5.7 Output Creation - `output_compression.vhd`

Because of its complexity, in this documentation the module and the explanation of how it operates are divided into four parts (sections 8.5.7.2 to 8.5.7.5): this separation is not however visible in the code, since all the segments are merged in one architecture.

8.5.7.1 General Introduction

From the encoding block, the output module receives an encoded value every clock cycle (`data_in`), while the `new_cnk` signal and the number of bits (`bits_in`) needed for representing the data changes every 4 clock cycles. The data valid signal is sent by the previous module as well, while the reset is directly connected to the main reset input.

The whole module is responsible for preparing an output with a fixed width and its internal structure is shown if Figure 8.5. In our case the output provided is a 32 bits `std_logic_vector`. Two different ways of preparing the output vector have been implemented.

- **Right Alignment (recommended)**: the data are accumulated starting from the LSB ("right" side) of the output buffer. No particular treatment is needed before the shifting part. This is the version which is used by the current code. It allows to work better and faster in the decoding part, since it requires shorter shifts in the internal registers of the CPU.
- **Left Alignment**: here the storing starts from the MSB of the output buffer ("left" side). First the bits order of each data is reversed using the `reverse_vector` function. In case we are managing the first data of a chunk, both the incoming data and the `Length` word are reversed individually. This processing allows to easily place the data in the correct position before being shifted. This type of alignment is *not* recommended: it was made available because it was the first implementation for aligning data, but it is not compatible with the provided software decoder.

Both the alignments have in common the way in which the data are accumulated and how the availability of an output is signalled. They indeed rely on an internal buffer which has a larger length than the output word. The idea is that, after each clock cycle, we keep (*accumulate*) only those bits which are not released as output.

Before explaining the operations carried out by the circuit, we need to illustrate the composition of `bits_used`. This is an `unsigned` with a length of `accbits_width` $+ 1$ bits (6 bits in our implementation), therefore describing values up to twice the length of output data. The information contained in this register is used for two different purposes:

1. `bits_used(4 downto 0)`: this value is used for shifting the incoming data to the right position. By not including the MSB of `bits_used`, the value represents the output width, and is thus suitable for shifting.
2. `bits_used(5)`: the MSB is used for checking the availability of a new complete output data word.

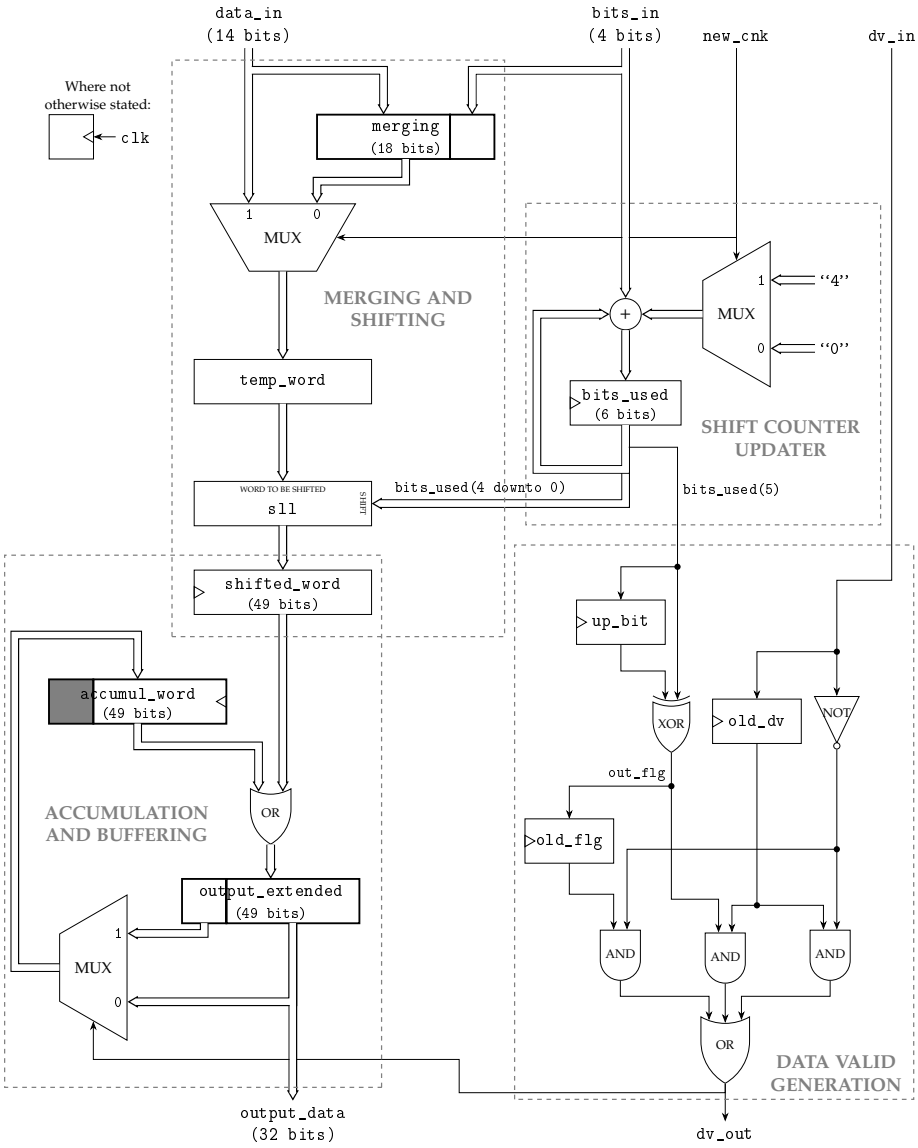


Figure 8.5: Internal structure of the module responsible for creating the output pattern. This module has been divided in sub-modules for documentation purposes. For more details see the text.

8.5.7.2 Merging and Shifting

The inputs to this module are the encoded data (`data_in`), the bits occupation information (`bits_in`) and the `new_cnk` signal, all from the encoding module (see section 8.5.6); the 5 bits of `bits_used` instead come from the sub-module responsible for updating the amount of accumulated bits (see section 8.5.7.3). The output is stored in the `shifted_word` register and then used in the sub-module where the accumulation process happens (see section 8.5.7.4).

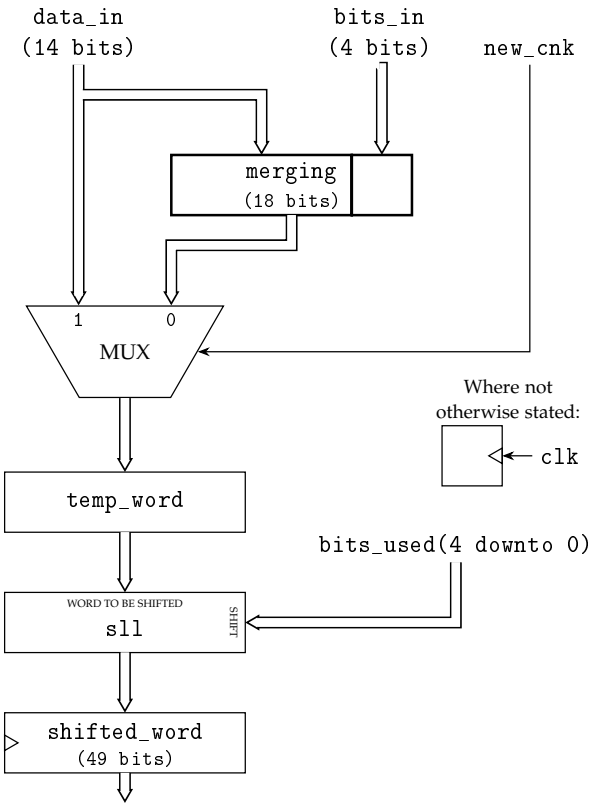


Figure 8.6: Part of the output module responsible for composing the to-be-stored data (`merging` or only `data_in`) and for the following shift (`sll`) of the resulting word in order to not overlap with buffered data. For more details see the text.

The explanation follows from Figure 8.6:

1. The data (`data_in`), together with the bits length information (`bits_in`), is made available at a given clock cycle. According to the `new_cnk` value we can have two different situations:

   - `new_cnk = '1'`: this case notifies the beginning of a new chunk of data. Therefore both the two input vectors need to be stored: in order to achieve this, they are paired together in the right order (see `merging`).
   - `new_cnk = '0'`: here only `data_in` will be considered for being stored;

2. The selected data (`temp_word`[9]) is shifted according to the value contained in `bits_used` (without its MSB), in order to not overlap data already stored in the accumulation buffer. The resulting word is stored in a register (`shifted_word`) at the next clock rising edge.

[9]This block is used just to give a name to the signal after the multiplexer.

8.5.7.3 Shift Counter Update

In this module the inputs `bits_in` and `new_cnk` come directly from the encoding block, as in the previous sub-module. The content of `bits_used` keeps track of the occupation of the output buffer, which has several uses.

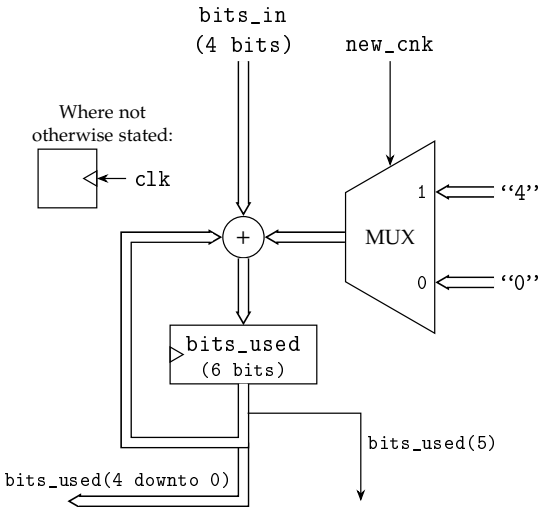


Figure 8.7: Part of the output module responsible for keeping track of the occupation of the output buffer. The content of `bits_used` is used both for shifting the data in the right position and in the generation of the signal notifying the availability of an output data. For more details see the text.

Figure 8.7 shows how `bits_used` is updated every clock cycle, depending on the value of `new_cnk` (therefore the presence of a multiplexer):

- `new_cnk = '1'`: in this case the circuit deals with the beginning of a new chunk of data. Since both the data and the `bits_in` vector are stored, the new value of `bits_used` will be given by the sum of its current value, the length of the input data (contained in `bits_in`) and the 4 bits needed for allocating the vector `bits_in`.

- `new_cnk = '0'`: here instead only `data_in` is involved in the storing process. Therefore to the new value we need to add just the content of `bits_in`.

8.5.7.4 Accumulation and Buffering

The `shifted_word` register is set according to the operations described in section 8.5.7.2. The information `dv_out` comes instead from the sub-module, which computes the data valid signal (see section 8.5.7.5).

Following Figure 8.8, the procedure for the case of an incoming data at the beginning of a new chunk is here described: At the beginning of a given clock cycle `shifted_word` and the accumulated information (`accumul_word`) are merged together through a bitwise `or` operation and a temporary vector

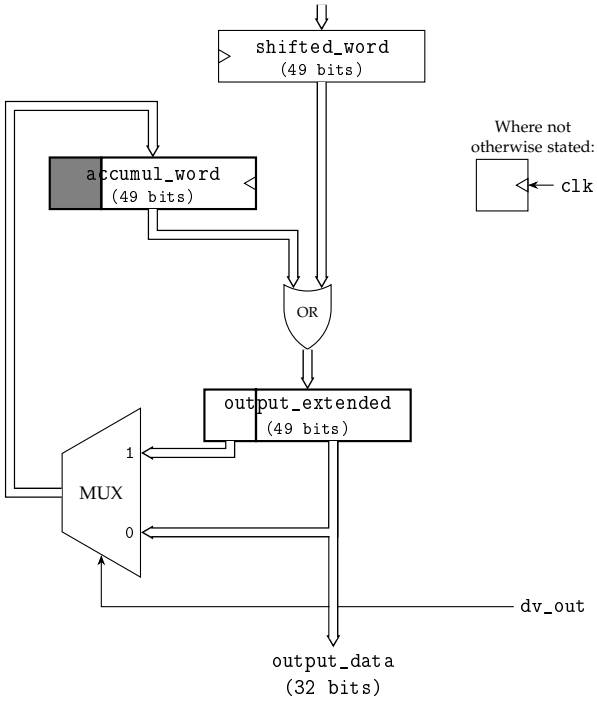


Figure 8.8: This sub-module updates the output buffer, merging the accumulated data (`accumul_word`) with the new shifted one (`shifted_word`). For more details see the text.

is formed (`output_extended`). Using the `dv_out` signal produced by the *data valid computation* block (see section 8.5.7.5), we can face two different scenarios:

- `dv_out = '0'`: this means that no output data is made available and therefore the useful part of `output_extended` fits inside `output_width-2 downto 0` (the first 31 bits in our implementation). Therefore only this part of `output_extended` is stored in the accumulation buffer `accumul_word` at the next rising clock;
- `dv_out = '1'`: here the resulting useful word equals or exceeds the `output_width` range, therefore the first 32 bits are released as output in the current clock cycle. `dv_out` is used to tell the external world about the availability of a valid output. At the next clock rising edge, only the small remaining part of `output_extended` is stored inside the accumulation buffer, while the part which was sent to the output is lost.

8.5.7.5 Data Valid Generation

In Figure 8.9 the part of the circuit responsible for generating the data valid signal is shown. In order to extend the following analysis (and the circuit itself) easily to any configuration of the whole compression circuit, we assume that the user of this circuit wants to use an output data width equal to a power of 2. In this way, we do not need to reset `bits_used` once we fill the output data width, since the natural overflow automatically resets its value.

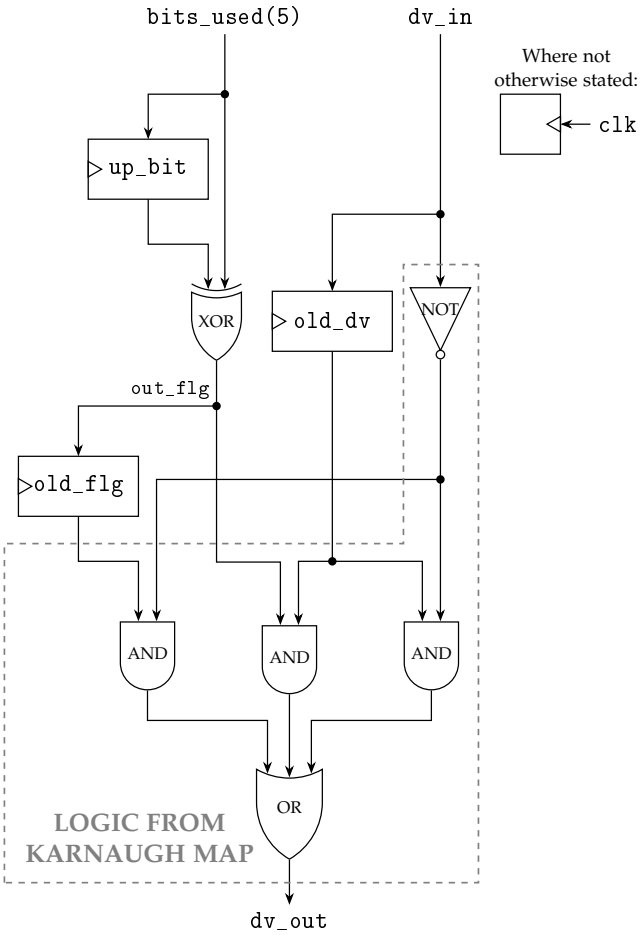


Figure 8.9: Circuit for generating the output data valid signal. The result is based on the data valid signal travelling along the whole compression module (`dv_in`), a signal tracking the filling of the output buffer (`out_flg`) and another signal, which helps notifying the end of input data (`old_dv`). The logic generated from the analysis done with the Karnaugh Map is highlighted. For more details see the text.

As already pointed out, the MSB of `bits_used` is not needed for filling the output buffer. However every time it changes value, it signals that the part of `bits_used` responsible for the shifting has just overflowed, meaning that

- the next shift will move the incoming data inside the first 32 bits range of `output_extended`;
- the first 32 bits of `output_extended` are full of useful data that need to be read out.

These are the reasons we monitor the *change* of the MSB value through a comparison (`xor`) between its current and previous values. The resulting value is labelled here as `out_flg` for shorter referencing.

To easily understand how the `dv_out` is generated, we show the procedure

used to built the circuit, rather than analysing it. We first need to realise when `dv_out` has to be set to '1' and when to '0':

1. If `dv_in` = '1' and `old_dv` = '0', it means that the data valid signal (`dv_in`) just arrived to the output stage, after the whole circuit has been activated. Therefore `dv_out` has to be '0'.
2. When `dv_in` = '1' and `old_dv` = '1', the data have already begun to accumulate in the output buffer. Therefore `dv_out` depends only on the value of `out_flg`, which signals the availability of a legitimate output data.
3. If `dv_in` = '0' and `old_dv` = '1', it indicates that the last set of data has reached the end of the circuit. Consequently the output data need to be readout anyway, even if the buffer is not full (meaning `out_flg` is not '1'). So `dv_out` has to be '1'.
4. A clock cycle after the scenario presented in the previous point, i.e. `dv_in` = '0' and `old_dv` = '0', there could be still some data left in the buffer. This happens when the last set of data arriving fills the buffer above its capacity. Therefore only the part sent to the output would be readout (see previous step), while the remaining bits are being accumulated. This implies that in the next clock cycle they need to be readout: this scenario is notified by the `old_flg` being raised to '1'.

These different situations are summarised in a truth table shown in Table 8.4.

dv_in	old_dv	out_flg	old_flg	dv_out	CASE
1	0	1	0	0	1
1	0	1	1	0	1
1	0	0	0	0	1
1	0	0	1	0	1
1	1	1	0	1	2
1	1	1	1	1	2
1	1	0	0	0	2
1	1	0	1	0	2
0	1	1	0	1	3
0	1	1	1	1	3
0	1	0	0	1	3
0	1	0	1	1	3
0	0	1	0	0	4
0	0	1	1	1	4
0	0	0	0	0	4
0	0	0	1	1	4

Table 8.4: Truth table for generating the `dv_out` signal. The *CASE* number refers to the previous list describing the different scenarios.

From the truth table we can derive the wanted logic function using the Karnaugh Map procedure explained in section 5.1.1.1. In Figure 8.10 the K-

map is reported with the groups already highlighted. Using the rules shown previously, it is possible to see that the searched logic function is $\texttt{dv\_out} = \texttt{old\_dv} \cdot \texttt{out\_flg} + \overline{\texttt{dv\_in}} \cdot \texttt{old\_dv} + \overline{\texttt{dv\_in}} \cdot \texttt{old\_flg}$, where the first term is given by the squared group on the bottom, the second term by the vertical rectangular group and the last one by the squared group on the left. Representing multiplications as `and` and additions as `or`, the function becomes `dv_out` = (`old_dv and out_flg`) `or` (`not dv_in and old_dv`) `or` (`not dv_in and old_flg`) and the resulting circuit is the one shown in Figure 8.9.

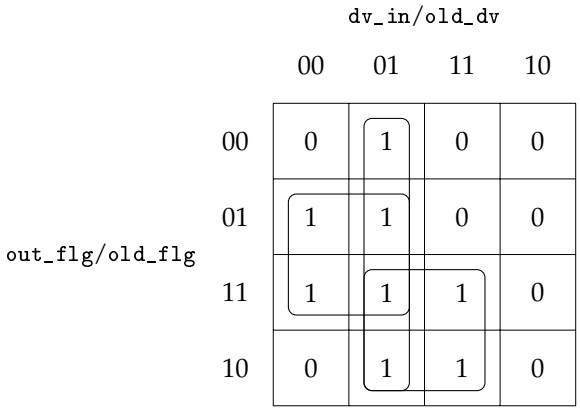


Figure 8.10: Karnaugh Map for the logic function used for generating the output data valid signal. The resulting function is $\texttt{dv\_out} = \texttt{old\_dv} \cdot \texttt{out\_flg} + \overline{\texttt{dv\_in}} \cdot \texttt{old\_dv} + \overline{\texttt{dv\_in}} \cdot \texttt{old\_flg}$ or, with logic operators, `dv_out` = (`old_dv and out_flg`) `or` (`notdv_in and old_dv`) `or` (`notdv_in and old_flg`).

8.5.8 Test Bench - `tb_compr.vhd`

The provided test bench (see section 5.2) contains an instance of the the `compression_module_chunc` entity, which is fed with a clock, a reset and a data valid signal. The input data values are read from a text file. The output is consumed every time the output data valid signal goes to 1 and it is then written into another text file. The resulting output values are then compared with data (so called *golden model*) generated by a C program designed for testing the correct behaviour of the circuit.

8.6 Circuit Synthesis and Analysis

8.6.1 Tools Employed

FPGA Device The target device for testing the design is an Altera MAX10 10M50DAF484C7G and its features are summarised in Table 8.5. The reason why we use this kind of FPGA is because we own a Terasic DE10-Lite board [35].

In the MAX 10 FPGA family, the *logic elements* (LEs) have, among other blocks, a 4-input lookup table (LUT) [36]. Compared to a Spartan 7 FPGA by Xilinx, which is equipped with 6-input LUTs [37], this means that the number of used LEs will probably be larger than in the Spartan case. The reason for this is that more complex logic functions can be implemented in a single 6-input LUT than in a 4-input one.

FEATURE	VALUE
Logic Elements	$\approx 50 \times 10^3$
Multipliers (18×18)	144
Memory	
Registers	$\approx 50 \times 10^3$
M9K Memory	1638 kbit
User Flash Memory	5888 kbit

Table 8.5: Main characteristics of the FPGA Altera MAX10-10M50DAF484C7G [38].

Synthesis, Place and Route Software To prepare and test the designed code for the target device, we used Altera Quartus Lite Edition 16.1, the free version of the Altera program, dedicated to VHDL synthesis and analysis for FPGAs. This basic version inevitably does not support the high-end devices and it is not equipped with enhanced features for synthesis and routing [39]. Although what we have designed is not a complex circuit, the user who work with better tools should keep this fact in mind.

8.6.2 Synthesis

The synthesis and place and route of the circuit have been executed fixing a constraint of 200 MHz on the clock frequency, in order to see what was the highest usable value. The results are summarised in Table 8.6.

FEATURE	**Synthesis**	**Place & Route**
Logic Elements	526	512 (1% of total)
Registers	243	243 ($<$ 1% of total)
Max Clock Frequency	—	147.3 MHz

Table 8.6: Main characteristics of the synthesised and the placed and routed circuit.

As can be seen, the occupation of logic elements and flip-flops is quite small compared to the resources of this FPGA. However we have to remind the fact that only the circuit we designed was taken into consideration. This circuit indeed will never be used alone, but rather as the last stage of a (probably much more) complex circuit. Therefore in the resulting routed circuit, the compressor could be characterised by a larger occupation than what stated in this document. Note that the compressor uses, relatively, more logic elements than registers. This is due to the main component, the large shifter, at the end of the output stage.

8.6.2.1 Timing Analysis

The timing analysis, performed by the TimeQuest software included in Altera Quartus, can be run different scenarios, where the working temperature has a major influence. The model we employ, uses an FPGA temperature of 85 °C. The results are therefore worse than in the case of the model using 0 °C as temperature. However this gives us a taste of the worst-case scenario.

An important concept usually used when analysing the circuit timing characteristics is the *slack*. When dealing with circuits which are a mix of combinational and (synchronous) sequential logic, the designer has to remember that data are stored in registers at the limits of each clock cycle. This means that the combinational logic chain between two consecutive stages of registers must not introduce a delay bigger than a single clock period. In case this requirement is not met, signals flowing in the combinational logic will not reach the next register in time to be stored and the output data will not be the expected. The time difference between the clock period and the delay introduced by the combinational logic between two consecutive register stages is called *slack*. The slack is calculated for each path and it takes a positive value when the path delay is smaller than the clock period, or a negative value when the delay exceeds the clock period.

As stated in Table 8.6, the maximum clock frequency at which the circuit can operate is ≈147 MHz. In Table 8.7, circuit paths which are limiting the clock frequency below the target of 200 MHz are shown: the related slack values are reported as well.

START POINT	END POINT	WORST SLACK (ns)
`active_mask`*	`bits_used`‡	−1.789
`active_mask`*	`shifted_word`‡	−1.704
`data_out`*	`shifted_word`‡	−0.769
`bits_used`‡	`shifted_word`‡	−0.745
`prevdatain`†	`positive register[0]`†	−0.307
`new_cnk`*	`shifted_word`‡	−0.032
`data_in`*	`mask`*	−0.020

Table 8.7: Paths which do not allow to reach an operating frequency of 200 MHz. To identify in which module each point is located, a symbol was used: † refers to the DIFF_CALCULATOR, * to the ENCODING, while ‡ to the OUTPUT_COMPRESSION.

Chapter 9

Conclusion

In this thesis work, many aspects of the CALIFA ENDCAP have been analysed, while other ideas and designs have been developed. The studies presented carry several consequences.

Good Performances of CEPA4 Crystals Analyses conducted on the CEPA4 crystals and described in chapter 3 and 4 demonstrate good energy and time resolution. The characterisation confirms what previous studies, most of them used to write the CALIFA ENDCAP TDR [4], reported.

Limitations and Opportunities in the Readout Electronics Investigations into the electronics planned in Ref. [4] have been carried out as well. Chapter 3 illustrates problems in reading out signals from the CEPA4 crystals: pulses from γ-ray and muon interactions can not be recorded properly at the same time using the same electronic settings. The main consequence is the limited dynamic range which can be inspected using the designed electronic chain. To solve this issue, our proposal is to change photomultiplier tubes in order to use two signals, one from the two last dynode stages, and one from the anode.

On the other hand, in chapter 4 we notice that some requirements of the proposed readout system, such as bit resolution and sampling frequency, can be relaxed without degrading its good time resolution capabilities too much. Solutions with these characteristics can be cheaper or can help to remove the dead-time problem that we have in the current setup. We have however to remind the reader that we have not taken possible effects of these ideas on performances of the pulse shape analysis into consideration.

Readout Procedures for the DRS4 To exploit the great characteristics of the phoswich crystals, a fast sampling frequency is needed and in the current readout system this task is left to the DRS4 chip. The design of procedures for reading out samples taken by this chip is described in chapter 6. Although the final result is a big block of VHDL code, which will be part of a more complex firmware developed at the Department of Physics of the *Technische Universität München*, the work done revealed issues and peculiarities of the DRS4, which have to be evaluated properly when dealing with this piece of electronics.

New Clock Delivery Scheme Experiments such as the R^3B setup present a scenario where many detectors are working together. Although good crystals and great electronics can deliver good time resolution, the acquired data are useless if they are not adequately time-correlated with those from other detectors or just with those from other crystals in the same detector. The currently designed system for delivering the time reference to the detectors and to each unit in a single detector is not able to deliver a clock signal with a sufficiently high time resolution. In chapter 7 we therefore describe a new scheme, which should ensure a clock signal with higher time resolution. It would help to exploit the high time resolution provided by $LaBr_3/LaCl_3$ phoswich crystals.

Reduce the Data Occupation Chapter 8 shows the design of a module for compression of data from pulse traces. Written in VHDL, it is suitable for digital circuits on FPGA. Compression of data inside an FPGA can reduce the amount of data transferred to the central unit as well as simply reduce disk usage. While building this block, we aimed at keeping the circuit occupation low and at having the highest working frequency achievable. In both cases the final goal is to deliver a module which is as flexible as possible. Future development aims to increase the compression ratio and to allow even more flexibility in managing input data.

Although all the work described in this report regards just small parts of the entire CALIFA, it easily conveys the complexity of building a particle detector. Engineers and physicists in research facilities around the world cope every day with challenges ranging from detection materials to signal processing and data analysis. The developments in these areas are not always an end unto themselves, but they can also improve the life of humanity.

Bibliography

[1] Gerhard Kraft. *History of the Heavy Ion Therapy at GSI*. GSI Helmholtzzentrum für Schwerionenforschung GmbH. URL: https://three.jsc.nasa.gov/articles/Krafts_GSI.pdf (visited on 07/08/2017).

[2] D. J. Morrissey and B. M. Sherrill. 'In-Flight Separation of Projectile Fragments'. In: *1st* 651 (2004), pp. 113–135.

[3] M. A. Ronja Thies. 'Bits and Pieces for the Nuclear Puzzle. Exploring light exotic nuclei with radioactive ion beams'. PhD thesis. Chalmers University of Technology, 2016. ISBN: 978-91-7597-426-2.

[4] *Technical Report for the Design, Construction and Commissioning of The CALIFA Endcap*. R^3B Collaboration. 3rd Aug. 2015. URL: http://www.fair-center.eu/fileadmin/fair/experiments/NUSTAR/Pdf/TDRs/TDR_R3B_CALIFA_ENDCAP_public.pdf.

[5] M. Bendel et al. 'RPID - A new digital particle identification algorithm for CsI(Tl) scintillators'. In: *The European Physical Journal A* 49 (6 2013), pp. 1–7.

[6] Olof Tengblad et al. '$LaBr_3$(Ce):$LaCl_3$(Ce) Phoswich with pulse shape analysis for high energy gamma-ray and proton identification'. In: *Nuclear Instruments and Methods in Physics Research Section A* 704 (2013), pp. 19–26.

[7] *BrilLanCe Scintillators - Performance Summary*. Saint-Gobain. 2009. URL: http://www.crystals.saint-gobain.com/sites/imdf.crystals.com/files/documents/brillance_scintillators_performance_summary_69795.pdf.

[8] *Technical Report for the Design of the NUSTAR Data AcQuisition System*. Nustar DAQ Working Group. 8th Mar. 2016.

[9] *DRS4 Data Sheet*. Version REV. 0.9. PSI - Paul Scherrer Institut. 2009. URL: https://www.psi.ch/drs/DocumentationEN/DRS4_rev09.pdf.

[10] E. Nácher et al. 'Proton response of CEPA4: A novel $LaBr_3$(Ce)-$LaCl_3$(Ce) phoswich array for high-energy gamma and proton spectroscopy'. In: *Nuclear Instruments and Methods in Physics Research Section A* 709 (2015), pp. 105–111. DOI: 10.1016/j.nima.2014.09.067.

[11] *Photomultiplier Tubes - R7600U Series*. Hamamatsu Photonics. 2016. URL: https://www.hamamatsu.com/resources/pdf/etd/R7600U_TPMH1317E.pdf.

[12] *AD9252 Data Sheet*. Version REV. E. Analog Devices, Inc. 2017. URL: http://www.analog.com/media/en/technical-documentation/data-sheets/AD9252.pdf.

[13] *Technical Information Manual - Mod. DT5751*. Version REV. 11. CAEN S.p.A. 2015. URL: http://www.caen.it/servlet/checkCaenManualFile?Id=11299.

[14] *User Manual UM3148 - Mod. DT5730/DT5725*. Version REV. 2. CAEN S.p.A. 2016. URL: http://www.caen.it/servlet/checkCaenManualFile?Id=12049.

[15] David M. Pozar. *Microwave Engineering*. John Wiley & Sons, Inc., 2012.

[16] *Efficiency Calculations for Selected Scintillators*. Saint-Gobain. 2016. URL: http://www.crystals.saint-gobain.com/sites/imdf.crystals.com/files/documents/efficiency_calculations_brochure_69670.pdf.

[17] Jan M. Rabaey, Anantha Chandrakasan and Borivoje Nikolic. *Digital Integrated Circuits. A Design Perspective*. Pearson Education, Inc., 2003.

[18] Randy H. Katz and Gaetano Borriello. *Contemporary Logic Design*. Pearson Education, Inc., 2005.

[19] John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson Education, Inc., 2007.

[20] Tiziano Villa et al. *Synthesis of Finite State Machines: Logic Optimization*. Springer-Science+Business Media, LLC, 1997.

[21] Hubert Kaeslin. *Digital Integrated Circuit Design*. Cambridge University Press, 2008.

[22] Andrew Rushton. *VHDL for Logic Synthesis*. John Wiley and Sons Ltd., 2011.

[23] *Intel to buy Altera for $16.7 billion in its biggest deal ever*. Reuters. 1st June 2015. URL: http://www.reuters.com/article/us-altera-m-a-intel-idUSKBN0OH2E020150601 (visited on 23/05/2017).

[24] *DRS4 Discussion Forum*. Paul Scherrer Institute PSI. URL: https://midas.psi.ch/elogs/DRS4+Forum/ (visited on 21/06/2017).

[25] *BrilLanCe 380 Scintillation Material*. Saint-Gobain. 2016. URL: http://www.crystals.saint-gobain.com/sites/imdf.crystals.com/files/documents/brillance380-material-data-sheet_69765.pdf.

[26] *Febex3/16, preliminary specification*. Version Preliminary. GSI/FAIR. 2011. URL: https://www.gsi.de/fileadmin/EE/Module/FEBEX/febex3.pdf.

[27] *LatticeECP3 Family Data Sheet*. Version 02.8EA. Lattice Semiconductor Corp. 2012. URL: http://www.latticesemi.com/~/media/LatticeSemi/Documents/DataSheets/Lattice/LatticeECP3EAFamilyDataSheet.pdf.

[28] *ADA4932-1/ADA4932-2 Data Sheet*. Version REV. E. Analog Devices, Inc. 2016. URL: http://www.analog.com/media/en/technical-documentation/data-sheets/ADA4932-1_4932-2.pdf.

[29] *Si530/531 data sheet*. Version D. Silicon Labs. 2013. URL: https://www.silabs.com/documents/public/data-sheets/si530.pdf.

[30] *EXPLODER3, preliminary specification*. Version Preliminary. GSI/FAIR. 2012. URL: https://www.gsi.de/fileadmin/EE/Module/EXPLODER/exploder3_v4.pdf.

[31] *WHITE RABBIT Simple FMC PCIe Carrier v4 SPEC*. Seven Solutions. URL: http://sevensols.com/index.php/download/brochure-spec/?wpdmdl=989.

[32] *FMC Digital I/O - 5 channels TTLA - FMC DIO 5CH*. Seven Solutions. URL: http://sevensols.com/index.php/download/brochure-fmc-dio/?wpdmdl=975.

[33] Khalid Sayood. *Introduction to Data Compression*. Elsevier, Inc., 2012.

[34] *Quartus® Prime Support for VHDL 2008*. Altera Intel. 2016. URL: http://quartushelp.altera.com/16.1/#hdl/vhdl/vhdl_list_2008_vhdl_support.htm (visited on 23/05/2017).

[35] *DE10-Lite Board - Overview*. Terasic, Inc. URL: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=234&No=1021&PartNo=1 (visited on 21/08/2017).

[36] *MAX 10 FPGA Device Architecture*. Version 2016.08.11. Intel Corp. 2016. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/max-10/m10_handbook.pdf (visited on 03/08/2017).

[37] *7 Series FPGAs Data Sheet: Overview*. Version 2.5. Xilinx Inc. 2017. URL: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (visited on 03/08/2017).

[38] *MAX 10 FPGA Device Overview*. Version 2017.02.21. Intel Corp. 2017. URL: https://www.altera.com/en_US/pdfs/literature/hb/max-10/m10_overview.pdf (visited on 02/06/2017).

[39] *Intel Quartus Prime Design Software - Compare Lite, Standard, and Pro Editions*. Intel Corp. 2017. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/po/ss-quartus-comparison.pdf (visited on 02/06/2017).