SAPIENZA
UNIVERSITÀ DI ROMA

Study of the Readout System of a Microcapillary Position Detector Filled with Scintillating Liquid

Department of Physics

Corso di Laurea Magistrale in Physics

Candidate

Guglielmo Gemignani

ID number 1401792

Thesis Advisor

Prof. Stefano Veneziano

Academic Year 2011/2012

Study of the Readout System of a Microcapillary Position Detector Filled with Scintillating Liquid

Master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: November 3, 2012

Author's email: guglielmogemignani@gmail.com

Ai miei genitori
e ai miei fratelli

Acknowledgments

I like to think of this work as the report of a group-project that I had the honour to take part into. Such group has been superbly guided by my advisor Prof. Stefano Veneziano who had the patience of receiving my daily incursions in his office, always helping me in the best way a student could ever hope for. During these seven months, I was often helped by Paolo Bagiacchi and Prof. Francesco Safai Tehrani to whom I would also like to express my deep gratitude. Further thanks go to my laboratory partner, Agostino di Francesco, as well as to Lorena Stellato, Dr. Alessia D'Orazio, all the members of the LABE, and the EPFL Lausanne and CERN groups for all the assistance they gave me.

During this master course I had the privilege of learning from many outstanding academic figures. Therefore my appreciation goes to all my professors, especially Dr. Fabrizio Ameli, Prof. Franco Meddi and Prof. Lucia Sorrentino Zanello for their guidance and support throughout this period full of many significant decisions.

The work undertaken would have been so much harder if I didn't meet several great companions during these two intense yet stimulating years. In particular I would like to express my gratefulness to Andrea, Caterina, Francesco, Guglielmo, Livia, Matteo, Nausicaa, and Roberto for tolerating my fluctuating mood and my recurrent stress moments. I would also like to acknowledge my Pisa fellows, specifically Alessandro, Bruno, Camilla, Guido, Ivan, Silvio, and Tommaso, for their constant presence in my life in spite of the distance that divides us. Equally, my lifelong friends have an incessant presence in my life despite my recurrent habit of being absent from home; so thank you Andrea, Daniela, Emanuele, Francesca, Francesco, Giulia, and Lara for your invaluable friendship.

I would not be here if it wasn't for my relatives that helped me during the most difficult moments of my life; to you, thanks for your constant teaching and valuable support. Through life's joys, difficulties and everyday contrasts, we managed to arrive here; my gratitude and affection to my dad and siblings that have been the most important life teaching figures in my entire life. Finally, to the most important person of my life, Mom, thank you for everything you have ever done to me. I miss you.

November 3, 2012

Contents

Abstract

Beam diagnostics is an essential need for any accelerator system. These systems can be thought as the 'organs of sight' to control the behaviour and the properties of a beam. Even though about 3% to 10% of the total cost of an accelerator facility must be dedicated to diagnostic instrumentation, usually the amount of man-power for the design, operation and further development of these devices far exceeds 10%, due to the complex physics and techniques involved. A large variety of parameters need to be measured for a good characterization of the beam.

One of these parameters is represented by the beam profile, meaning the spatial intensity on a particular plane, transverse to the particle direction. The beam profile is usually monitored with scintillator screens, wire scanners, diamond detectors or scintillating fibres that may have problems of resolution and radiation resistance. Thanks to the advanced researches made in Micro Electro-Mechanical Systems, nowadays scientists are designing a new kind of particle detector that may solve these problems.

This thesis will describe a new kind of detector for beam profile monitoring composed by microcapillary tubes filled with a scintillating fluid and the dedicated readout electronics. Such a new kind of detector has been described for the first time in the article 'Scintillation particle detection based on microfluidics' by A. Mapelli et al. [1] in 2010. A collaboration started in the first half of 2012 between INFN Rome, EPFL Lausanne and CERN in order to develop this detector.

This work reports what has been accomplished so far in order to build the readout electronics of the detector prototype. It has been structured as follows:

- The first chapter covers a brief history of the evolution of the available technology that led to the design of the microcapillary detector, the measurements needed for beam profile monitoring and the general structure of the devices used.

- The theory behind particle tracking and monitoring is presented in the second chapter. In particular the scintillators' behaviour, the theory of energy loss through matter and the detector geometry will be discussed.

- In chapter three the available light sensors are described.
- The readout components and the readout architecture will be described in chapter four.
- The laboratory setup and the debugging tests are covered in chapter five.
- In chapter six the first results are presented.
- The last chapter will finally be dedicated to the conclusions that can be drawn from this work.

Chapter 1

Introduction

Starting from the beginning of the eighties, scientists had the ambition of manufacturing processes that can create exceptionally small machines in order to explore, build and control the extremes of length and time scales. Wanting to achieve this goal, Micro Electro-Mechanical Systems (MEMS) have been invented thanks to the progress of the latest solid-state physics researches. The acronym MEMS refers to devices that have a typical length of less than $1\,mm$ but more than $1\,\mu m$, that combine electrical and mechanical components and that are assembled using integrated circuit batch-processing technologies. This manufacturing process is carried on using surface and bulk silicon micromachining, electrodeposition, plastic molding, lithography and electrodischarge machining (EDM) [2].

To these days MEMS are finding increased applications in a variety of industrial and medical fields, with a potential worldwide market in the billions of dollars. Thanks to the growth of this market a lot of researches have been funded, leading to a fast evolution of the modern technology and letting scientists hope in a short time Nano Electro-Mechanical Systems development. Having achieved this goal, medical and high energy physicists have the possibility to take advantage of these devices to develop a new generation of particle detector for beam diagnostics.

In this chapter the essential beam profile monitoring characteristics for both high energy physics and hadrotherapy will be covered and the main detectors with their limitations, used for these purposes, will be described as well.

1.1 Beam Monitoring

During the data taking process, a large amount of parameters needs to be monitored in order to completely control an accelerating apparatus. There are three main classes of diagnostic requests at any of these facilities:

- Reliable, quick measurements to determine the basic parameters of a machine

setting, mainly used as a fast check of the general functionality. These devices should be non-destructive for the beam, yielding an online information and their readings should give a single number or simple plots.

- Instrumentation built for a daily check of performance and stability or for the control of wanted parameter changes in the accelerator setting. It can also be used for solving simple machine problems in case of any malfunction.

- Complex instrumentation for the commissioning of a new accelerator component, for the development of higher performance and for solving more serious problems in case of a malfunction. The devices can be more sophisticated to use and might be destructive for the beam. The main goal of these devices is the creation of reliable information about the complex beam behavior allowing a clear interpretation.

The monitoring of a beam profile is mainly included in the first category.

This kind of measurementment is exploited to control the beam width, as well as the transverse matching between different parts of an accelerator. Usually multiple devices, assembled for this purpose, are installed in the accelerating tunnel in order to control how quadrupole magnets bend, focus and correct the particle beam. Depending on the beam particles, their current and their energy, a very large variety of devices exists.

Some of the most popular techniques used for this goal are represented by scintillator screens, wire scanners, diamond detectors and scintillating fibres. A scintillator screen is the most common and easy way of measuring a beam profile consisting of intercepting the beam with a scintillating layer (a P43 phosphor layer for example) and viewing the emitted fluorescence with a CCD camera. Alternatively wire scanner devices are used where a high spatial resolution of the beam is needed. Normally the size of an electron beam is less than 1 mm, while proton or heavy ion beams have large diameters, up to some cm. Alternatively, diamond detectors are being studied in particle beam profile and dosimetry monitoring apparatuses. These devices have become more and more interesting for research purposes due to their qualities compared to the previously used silicon diodes. Finally scintillating fibres, filled with organic or inorganic liquids, are being used as multipurpose detectors.

1.2 Hadrotherapy Beam Profile

Nowadays, beams of photons, protons and other heavier particles are also being used in oncological therapies. In fact, in comparison to conventional therapies, high-energy beams, constituted by such particles, offer significant advantages for

cancer treatment. Their energy distribution in tissue is in fact characterized by a small deposit at the beginning of their dose release and a distinct high energy deposit, called Bragg peak, with a sharp fall-off at the end of their interaction path as can be seen in fig 1.1.

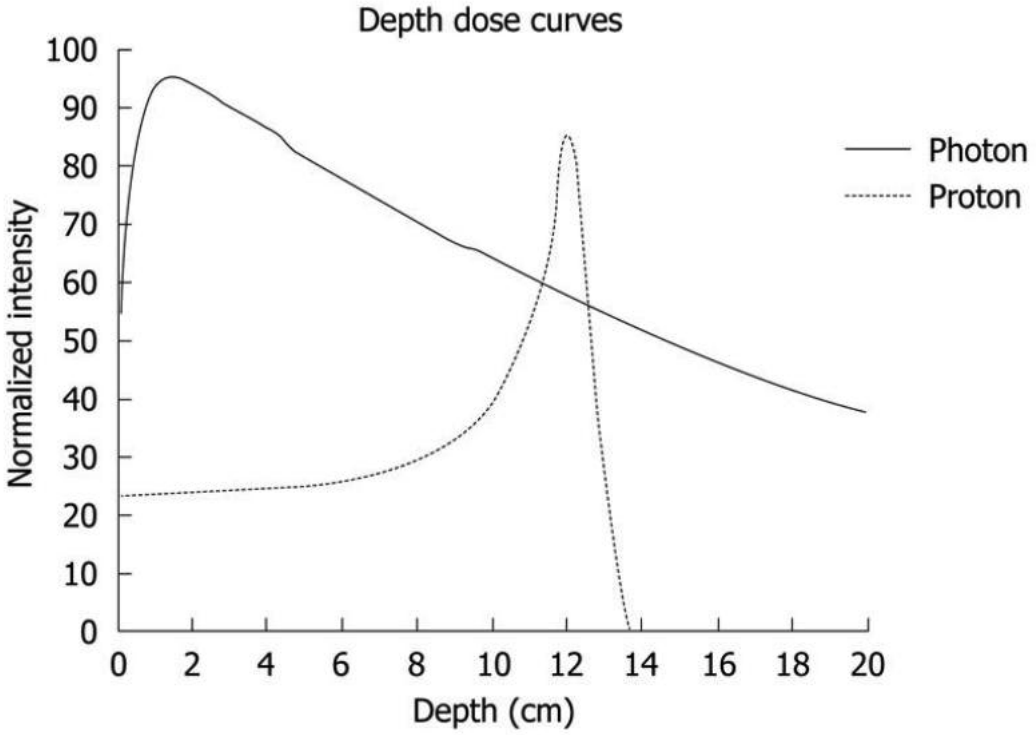


Figure 1.1. Bragg curve for photons and protons beams. [3].

Taking full advantage of this well-defined distribution and its small lateral spread, modern beam detectors allow a millimeter precision dose delivery. In addition, ions heavier than protons show an enhanced biological effectiveness in the Bragg peak region resulting in a reduced subsequent cellular repair; these characteristics are particularly attractive for radio-resistant tumors treatment that have to deal with targets localized near vital organs.

There are two main approaches to deliver the desired dose distribution to a certain patient:

- *Discrete scanning (spot or voxel)*: in this technique the dose is delivered by pencil beams applied in discrete steps. After each pencil beam is delivered, the source is interrupted, the beam steering elements are changed to deliver particles at a different position and/or energy and the beam is then turned back on; such process is repeated until the desired dose distribution and number of particles have been delivered.

- *Continuous scanning (raster scanning)*: this is a method in which a pencil beam is scanned continuously in a raster pattern. A variation in intensity as a function of the beam position is achieved by continuously controlling the proton-beam intensity and the scanning speed. Once one "layer" of particles of a particular energy has been delivered, the source is interrupted, the beam energy is changed, and the beam is then turned back on to irradiate the next layer.

In order to control such dose delivery techniques, beam monitor systems are needed. Beam monitor devices for medical applications have analogous features to high energy physics detectors, differing in their goals and control mechanisms since they are used for running-in, setting-up and maintaining the patient treatment beam. These devices need to be capable of monitoring various beams that differ from each other in intensity, energy range, transverse widths, longitudinal time-profile and particle constitution in order to satisfy all the various treatments' requirements. There are mainly three types of monitoring apparatuses that are required to handle these treatment beams:

- Profile detectors: they usually have a destructive interaction with the beam and, as their name suggests, they are used to determine the beam profile, in order to monitor the quality of the beam, similarly to the high energy physics case. A typical requirement for such type of detectors is a dimension of the order of $3 - 5\,cm$, with a spatial resolution of the order of $100\,\mu m$. These detectors are usually positioned in vacuum.

- Beam position detectors situated in treatment rooms: they monitor the position of the beam just before interacting with the patient. These devices need to have a non-destructive interference with the beam in order to prevent beam diffusion and Bragg peak position changes during the treatment. The maximum amount of material tolerated is usually of the order of $1\,mm$ water-equivalent. Moreover the size of the detector needs to be large enough to follow the beam rastering of the order of $20 \times 20\,cm^2$, while the readout system needs to be capable of following the beam raster frequency of the order of $100\,Hz$.

- Dosimetry detectors: they are exploited to measurement the quantity of energy delivered to the patient; such kind of detectors are usually placed inside a *human water phantom*, in order to measurement the total energy and the depth profile of the incident beam; because of that, these detectors need to be accurately calibrated.

A wide variety of detectors for these purposes are being studied in modern

hadrotherapy facilities and the different available possibilities that can be adopted will be shown in the following sections.

1.3 Existing Detectors

In this section the functioning principles of different available beam monitors will be shown with their strengths and weaknesses, in order to introduce the microcapillary scintillators covered in the following chapter.

1.3.1 Scintillator Screens

The most direct way to observe a particle beam consists in monitoring the light emitted from a scintillation screen through a commercial video or CCD camera. These devices are installed in nearly all accelerator apparatuses from the beam source up to the target and their general schematic structure is shown in figure 1.2.

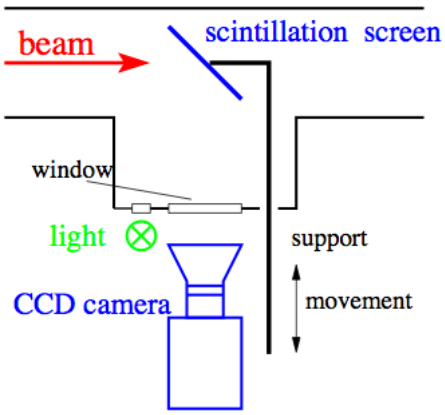


Figure 1.2. Schematic representation of a scintillation screen detector [3].

When a particle penetrates a scintillating material, the electronic energy loss generated by the collision between the beam particles and the target electrons create some fluorescence photons, later measured by a camera. The properties required and owned by a well designed detector of such kind are:

- High light output matched to the optical system of the CCD camera in the optical wavelength range between $450\,nm$ and $700\,nm$.
- High dynamic range and a good linearity between the incident particle flux and the light output. In particular, when a saturation of the light occurs a deformation of the recorded profile is generated.

- Radiation hardness to prevent permanent damage caused by the incident particles.
- No absorption of the emitted light to prevent artificial broadening caused by the stray light trapped in the material.
- Fast decay time in order to allow the observation of possible beam size variations.

Plastic scintillators have only low radiation hardness and for this reason various kinds of inorganic materials have been studied. In Figure 1.3 different proprieties of possible material used to fabricate the scintillator screens are shown.

Abbreviation	Material	Activator	max. emission	decay time
Quartz	SiO_2	none	470 nm	< 10 ns
	CsI	Tl	550 nm	1 μs
Chromolux	Al_2O_3	Cr	700 nm	100 ms
YAG	$Y_3Al_5O_{12}$	Ce	550 nm	0.2 μs
	Li glass	Ce	400 nm	0.1 μs
P11	ZnS	Ag	450 nm	3 ms
P43	Gd_2O_2S	Tb	545 nm	1 ms
P46	$Y_3Al_5O_{12}$	Ce	530 nm	0.3 μs
P47	$Y_2Si_5O_5$	Ce&Tb	400 nm	100 ns

Figure 1.3. Different proprieties of possible material used to fabricate the scintillator screens [3].

Figure 1.3 gives only approximated parameter values since the properties of doped materials are strongly dependent of the incident particle concentration.

For high intensity beams, it needs to be checked that the material is not destroyed by the power absorption; in particular, for slow heavy ions this restricts the use of these detectors. A disadvantage of the screens is related to the large amount of material present along the beam. In fact, the material used is usually several *mm* thick that it causes a large beam energy loss, excluding this detector from the possible devices used for the diagnostics of a circulating beam inside a synchrotron or for online monitoring measurements.

1.3.2 Wire Scanners

An alternative to scintillator screens is represented by wire scanners. This type of detector is composed of a single wire placed perpendicular to the beam path as shown in figure 1.4.

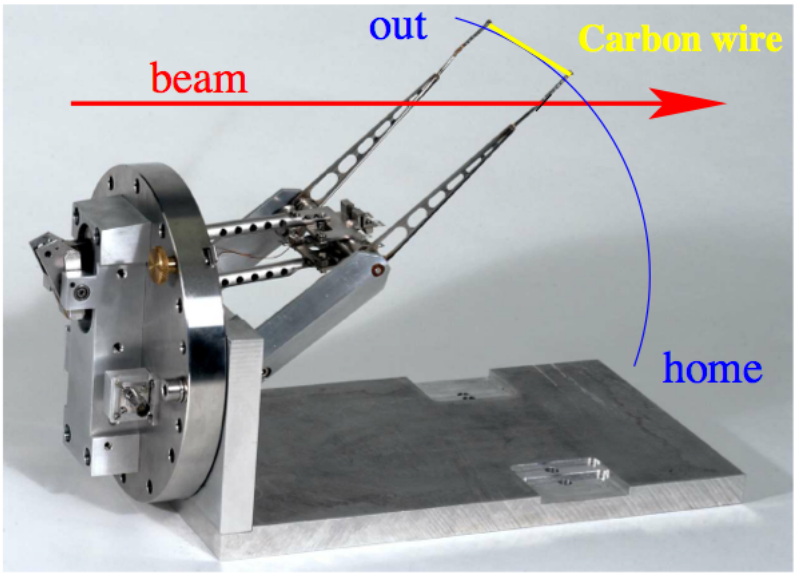


Figure 1.4. Arrangement of a straight wire on a fast pendulum mechanics [3].

The advantage of this technique is given by the resolution in the sub$-mm$ range making this device used in electron accelerators with a comparable beam size and proton synchrotrons due to the small amount of intercepting matter. With the devices controlled by a special pneumatic mechanism, scanning velocities up to $10\,m/s$ can be achieved.

The wire material used is usually carbon or SiC due to its low weight and low nuclear charge Z, resulting in a low energy deposition in the wire (see Bethe-Bloch equation in section 2.1). In addition, these materials can withstand high temperatures without melting and their thickness can be designed to be down to $10\,\mu m$. Because of the structure of the single wire scanned, even with high scanning velocity, the beam profile is not taken at a single instant, giving therefore only the possibility to probe the steady state distribution.

For the beam profile display, the wire scanner space location, determined by the position encoder, is plotted on the horizontal axis while the vertical axis beam signal can be deduced from the current generated by the emitted secondary electrons. Another way of measuring the beam profile on both axis can be achieved by mounting the wires in a crossed orientation, as shown in figure 1.5.

The strengths and weaknesses of this device can be schematized as follows:

- A moving wire samples the beam profile at different locations and at different times; therefore variations of the beam intensity in time will be mixed with transverse intensity variations.

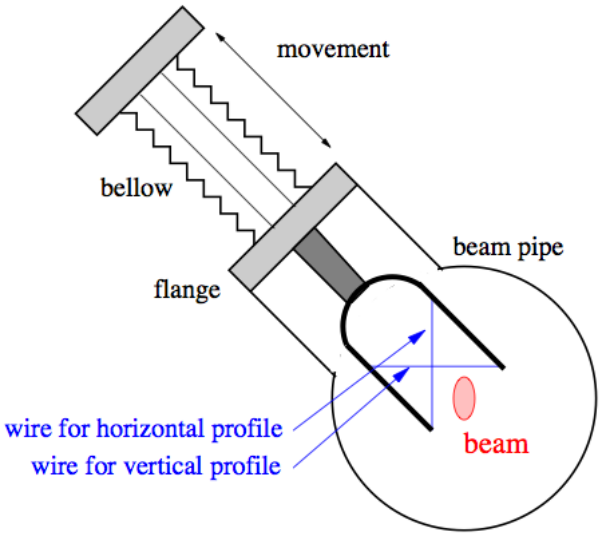


Figure 1.5. Wired scanners mounted in crossed orientation to measurement the beam profile on the x and y axis. [3].

- In case of pulsed beams further complications may arise from the demand for exact synchronization.
- A wire scanner can have much higher resolution compared to a scintillator scanner, down to $10\,\mu m$, due to its constant movement. (For high resolution mechanical vibration has to be avoided.)
- The electronics for data acquisition is cheap for a scanning system and it has a low energy deposition in the wire, depending on its material component.

1.3.3 Diamond Detectors

In radiotherapy, silicon diodes are the detectors which are most often used for the measurementment of relative dose distributions. A good alternative recently explored for particle beam monitoring is represented by diamond detectors [4]. Some studies on the use of natural diamond detectors for similar measurements have been published but, due to the high cost of natural diamonds, they had difficulties in establishing themselves as a solid alternative. Thanks to the progress made in the *chemical vapor deposition* (CVD) technology the production costs have decreased and this solution got a foothold in the particle beam monitoring panorama.

Due to the fact that diamond is almost equivalent to soft tissue, having an atomic number $Z = 6$ instead of $Z = 7.42$, the output signal of the detector is directly proportional to the absorbed dose rate of tissue and therefore the data produced can

be used without any correction. On the other hand in high energy particle physics these devices are being studied for experiments such as the BaBar, Belle, CMS and ATLAS [5].

The working principle of these devices is shown in figure 1.6; when a charged particle passes through the diamond crystal, it interacts with the material producing an electron-hole pair, subsequently accelerated by an electric field and extracted as an electric signal.

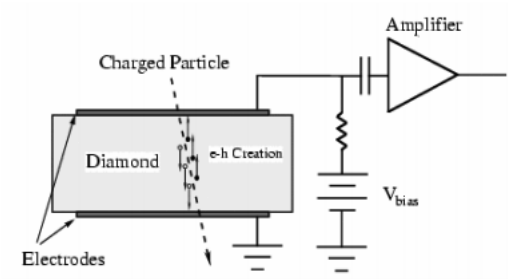


Figure 1.6. Schematic representation of a diamond detector [5].

These type of detectors have a high response, good time and spatial resolution (around $2\,\mu m$ [6]), good temperature stability (about $1\%\,K^{-1}$) and good radiation stability and tolerance (up to $2{\cdot}10^{15}$ particles per cm^2). Their drawback is represented by a fast readout and low noise requirements for single particle counting.

Chapter 2

A New Detector for Beam Monitoring

This chapter gives an overview of scintillator detectors, analyzing scintillating fibre and microcapillary scintillator characteristics. Such analysis is carried out covering the theory behind particle detectors, illustrating the effect produced and the basic reactions that occur when a particle encounters matter and interacts with it along its propagation path. These processes are the basis of all current particle detection technologies and, thus, they determine their peculiar characteristics. Finally, in the last section of this chapter, an explanation of how microcapillary tubes are made will be given.

2.1 Basic Processes

2.1.1 Energy Loss

The passage of charged particles through matters is generally characterized by an energy loss and a deflection from the incident initial direction; these effects are the results of respectively an inelastic collision with the target atomic electrons and an elastic scattering caused by the presence of the material nuclei.

Out of the two electromagnetic interactions, the inelastic collision is the main responsible for the energy loss during heavy particles passage through matter. These collisions are characterized by a transfer of energy from the incident particle to the target atoms causing an excitation or ionization of the latter; they have a cross section of $\sigma \simeq 10^{-17} \div 10^{-16}\, cm^2$ where σ is defined as the effective area which governs the probability of some scattering or absorption event and is mathematically calculated as:

$$\sigma(E) = \int \frac{d\sigma}{d\Omega} d\Omega \tag{2.1}$$

where $d\Omega$ is the differential of the scattering solid angle and $\frac{d\sigma}{d\Omega}$ the differential cross section defined as the ratio

$$\frac{d\sigma}{d\Omega}(E,\Omega) = \frac{1}{F}\frac{dN_s}{d\Omega} \tag{2.2}$$

with F the incident flux and N_s the average number of particle scattered per unit time.

The amount of energy transferred in each collision is generally a very small fraction of the total kinetic energy of the particle; however a substantial energy loss is observed even in relatively thin layers of target material. For example, a $10\,MeV$ proton loses all of its energy in $0.25\,mm$ of copper due to the large amount of particle collisions with the target atoms per unit path length.

Elastic scattering between incident particle and target nuclei occurs frequently too although not as often as electron collision; however, in general, a small fraction of the particle total kinetic energy is transferred in these collisions due to the bigger mass of the nuclei compared to the incident particle.

All collisions are statistical processes, occurring with a certain probability given by the quantum mechanics' theory; however, due to the high amount of collisions per macroscopic path length, the fluctuations in the total energy loss are small and calculations can be done using the average energy loss per unit path length, also known as *stopping power* or $\frac{dE}{dx}$.

The correct calculation [7] for stopping power using quantum mechanics was first performed by Hans Albrecht Bethe and Felix Bloch, from whom the analytical expression takes its name, and was later corrected introducing the *density effect* and the *shell* correction, giving it its final form expressed as:

$$-\frac{dE}{dx} = 2\pi N_a r_e^2 m_e c^2 \rho \frac{Z}{A}\frac{z^2}{\beta^2}\left[ln\left(\frac{2m_e\gamma^2 v^2 W_{max}}{I^2}\right) - 2\beta^2 - \delta - 2\frac{C}{Z}\right] \tag{2.3}$$

with

- r_e: classical electron radius ($2.817 \times 10^{-13}\,cm$)
- m_e: electron mass
- N_a: Avogadro's number ($6.022 \times 10^{23}\,mol^{-1}$)
- I: mean excitation potential
- Z: atomic number of the target material
- A: atomic weight of the target material
- ρ: density of the target material

- z: charge of the incident particle expressed in units of the elementary charge e
- $\beta = \frac{v}{c}$ of the incident particle
- $\gamma = \frac{1}{\sqrt{1-\beta^2}}$
- δ: density correction
- C: shell correction
- W_{max}: maximum energy transfer in a single collision.

The stopping power dependency from different particles' energy is shown in figure 2.1 for different materials; from this figure it can be seen that as a incident particle slows down in matter, its rate of energy loss will change. The graph of the stopping

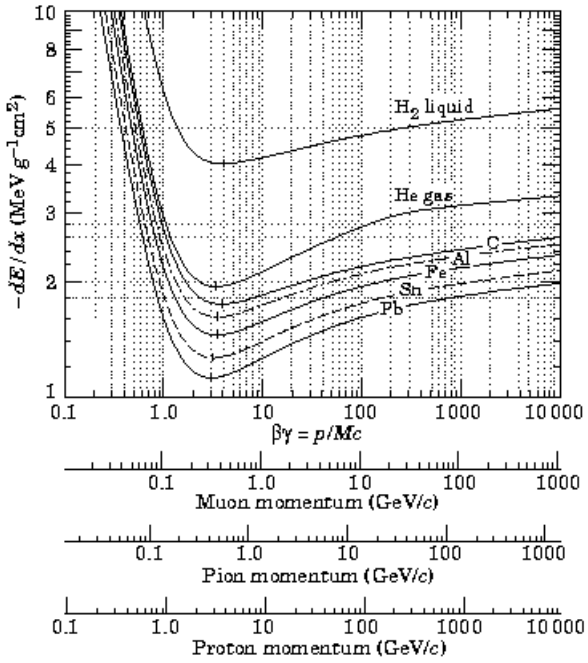


Figure 2.1. The stopping power dE/dx as a function of the energy of different particles [8].

power expressed as a function of the incident particle penetration depth is shown in figure 2.2; from this picture called the *Bragg curve*, the *Bragg peak* can be seen. The importance of this peak for radiotherapy can't be stressed enough: in fact, in medical application, there is a need to deliver a high dose of radiation to deep-rooted malignant cells without destroying too much of the coating healthy tissue; by choosing the incident particles and their energy this peak can be shaped to maximize the treatment effects on a patient.

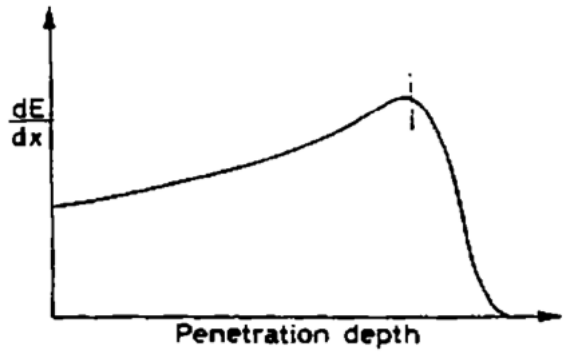


Figure 2.2. A general Bragg curve showing the variation of dE/dx as a function of the incident particle penetration depth in the target's matter [9].

2.1.2 Scintillation

Scintillation detection devices are one of the most widely and often used particle detectors in high energy physics today. This technology is based on the propriety of certain materials of emitting a small amount of photons, called scintillation, when they are struck by radiation. Coupling these devices with a photosensor and an amplifier, the scintillations produced can be converted in electrical signals and analyzed electronically to obtain information on the interacting particles.

Some of the characteristics required from a good scintillator are represented by sensitivity to energy, fast time response and pulse shape discrimination.

Sensitivity to energy means that the detector needs to give an output directly proportional to the deposited energy. Since the photodetectors can be designed to be linear too, the amplitude of the final signal produced will be proportional to the energy absorbed by the detector thus giving the possibility to use this device also for accurate energy measurements. Moreover a short response and recovery time are needed in order to obtain better timing information, greater precision and higher count rates compared to other detectors. Pulse shape discrimination can be used to distinguish between different types of incident particles by analyzing the shape of the device output light pulses [10]. This result can be achieved using the different fluorescence mechanisms of the scintillating material contained in the detector.

Some materials exhibit a propriety called fluorescence, meaning that they are capable of absorbing energy and reemit it in the form of photons. The time needed in order to absorb the energy and perform an atomic transition is on the order of $10^{-8}s$, moving the external atomic electrons to a metastable excited state. The delay time between the absorption and reemission may last from a few microseconds to hours depending on the scintillator material considered; thus, adding the delay and the absorption time, a device time response can be obtained.

A general characterization of a scintillating detector can be given through four parameters:

- the efficiency of the detector in converting the incident particles to photons.
- the grade of transparency of the scintillating material to its light in order to allow the generated light to escape the detector and to be detected by the coupled photosensors.
- the spectral range of the light emission that needs to be consistent with the spectral response of the chosen photosensors.
- the response time, given in terms of the decay constant τ. Two or more decay constants can be associated to a scintillating material.

A wide variety of scintillating materials exists, such as organic and inorganic, gases and glasses; in this section the attention will be focused on organic scintillators in order to understand the processes exploited by the studied microcapillary detector.

2.2 Organic Scintillators

Organic scintillators are "aromatic hydrocarbon compounds containing linked or condensed benzene-ring structures" [9]; they are widely used for their very short decay time of the order of nanoseconds. In these aggregate, fluorescent radiation arises from the transitions made by the non-bound valence atomic electrons. These electrons are in π-*molecular orbitals* and are not bound to any molecular atom. A typical energy spectrum for these particles is shown in figure 2.3, where spin singlet states have been separated from spin triplet states; the energy gap between each level is on the order of the eV.

When radiation penetrates the material the valence electrons are excited and a transition to one of the previously shown states occurs. From these states there is a high probability to make a radiative decay to one of the ground states, thus producing a number of photons in times on the order of nanoseconds. Thanks to the molecular nature of the scintillations, these organic materials can be exploited in many physical forms (i.e. crystals, mixtures of different compounds, liquids and solid forms, etc.) without loosing their scintillating properties. In the studied microcapillary detectors, liquid organic scintillators have been used and their qualities will be covered in the following sections.

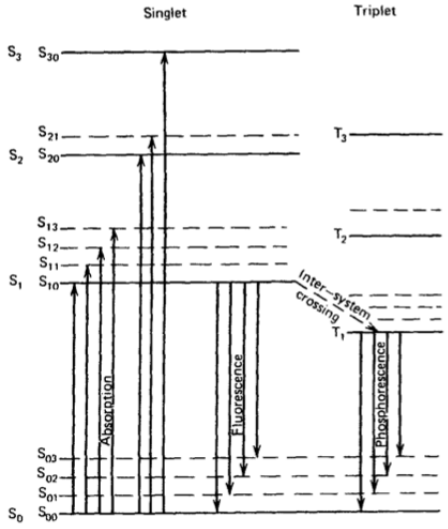


Figure 2.3. Typical energy spectrum of an organic scintillator molecule [11].

2.3 Scintillating fibres

Scintillating fibres are the main detectors that exploit the characteristics explained in the previous section. In fact, the energy deposited by ionizing particles into the active detector material is used in order to produce photons; once they are generated, thanks to a layer refraction index material that surrounds the scintillating material, these photons are guided to the photodetectors that are used to reveal the generated light. Alternatively, a thin layer of reflective material can be exploited to reflect the light through the fibre, allowing for signal transmission. These devices are multi-purpose detectors (such as calorimeters, time of flight measurement devices, tracking detectors, trigger counters etc.) and are used in a wide range of physics research fields [12] [13].

The general structure of an optical waveguide is represented in figure 2.4. The two indices of refraction for the core and the cladding, respectively n_2 and n_1, must minimize the relation given by Snell's law:

$$\theta_B = \arcsin(\frac{n_2}{n_1}); \;\; n_2 > n_1. \tag{2.4}$$

This request is being made in order to guide the light toward the readout systems and minimize the flaws given by the dispersion of light as shown in figure 2.5.

A typical configuration of a solid core scintillating fibre is composed by a PS core surrounded by a PMMA cladding, with an index of refraction respectively of 1.59 and 1.49 [10]. Typical dimensions of the fibres are 0.5 to 1 mm and the total fraction of light that is guided along the fibre amounts to few percents of the initial

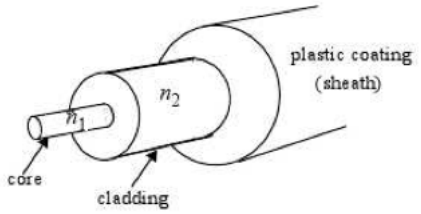


Figure 2.4. General structure of an optical waveguide used in the construction of scintillating fibres detectors [14].

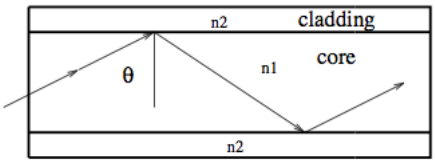


Figure 2.5. Working principle of an optical fibre waveguide [14].

generated photons. This technique is still a possible choice [15] but it presents a weak radiation-hardness characteristic and a difficulty of further miniaturization.

A solution to these flaws can be found in liquid scintillator capillaries. In fact, a glass capillary tube filled with liquid organic scintillator can behave as a scintillating fibre. This approach exhibits a high radiation-hardness characteristic, given by the possible circulation of the internal liquid, and an attenuation length of up to $3\,m$, thanks to the excellent light losses as low as 10^{-6} per reflection.

Thus, as it has been shown in this section, the final detector efficiency will be mainly affected by the choice of the scintillating material inside the fibre and its surrounding cladding or reflector. If during this material selection, the request of having $n_2 > n_1$ is not met, a reflective metal is deposited on the channels to increase the reflectivity. Moreover, in order to achieve the best detection results, the wavelength sensitivity peak of the readout system needs to be matched with the wavelength of the light yielded from the liquid, trying to keep the light reflection losses as low as possible. A list of possible scintillating organic materials and their characteristic is shown in figure 2.6. In the prototype studied [1] the microcapillary channels, filled with the EJ-305 material, have been covered with a thin layer of gold, due to the poor light guide performance of the two materials, having respectively $n1 = 1.50$ and $n2 = 0.99$.

Table A6.3 Properties of some organic scintillators

scintillator	density (g/cm^3)	index of refraction	wavelength of maximum emission (nm)	decay time constant (ns)	scintillation pulse height [1]	H/C ratio [2]	yield/ NaI
Monocrystals							
naphthalene	1.15	1.58	348	11	11	0.800	
anthracene	1.25	1.59	448	30-32	100	0.714	0.5
trans-stilbene	1.16	1.58	384	3-8	46	0.857	
p-terphenyl	1.23		391	6-12	30	0.778	
Plastics [3]							
NE 102 A	1.032	1.58	425	2.5	65	1.105	
NE 104	1.032	1.58	405	1.8	68	1.100	
NE 110	1.032	1.58	437	3.3	60	1.105	
NE 111	1.032	1.58	370	1.7	55	1.096	
Plastics [4]							
BC-400	1.032	1.581	423	2.4	65	1.103	
BC-404	1.032	1.58	408	1.8	68	1.107	
BC-408	1.032	1.58	425	2.1	64	1.104	
BC-412	1.032	1.58	434	3.3	60	1.104	
BC-414	1.032	1.58	392	1.8	68	1.110	
BC-416	1.032	1.58	434	4.0	50	1.110	
BC-418	1.032	1.58	391	1.4	67	1.100	
BC-420	1.032	1.58	391	1.5	64	1.100	
BC-422	1.032	1.58	370	1.6	55	1.102	
BC-422Q	1.032	1.58	370	0.7	11	1.102	
BC-428	1.032	1.58	480	12.5	50	1.103	
BC-430	1.032	1.58	580	16.8	45	1.108	
BC-434	1.049	1.58	425	2.2	60	0.995	

[1] relative to anthracene
[2] ratio of hydrogen to carbon atoms
[3] Nuclear Enterprises Ltd. Sighthill, Edinburgh, U.K.
[4] Bicron Corporation, Newbury, Ohio, USA

Figure 2.6. Possible scintillating organic materials and their characteristics. [16].

2.4 Microcapillary Scintillators

During the study and the development of an experimental scintillating fibres detector prototype for the LHC experiment, an upgrade of these devices has been derived by A. Mapelli [17]. The goal of his doctoral thesis, during which these detectors have been taken into account, was the fabrication of both scintillating fibres detectors and a new monitor device that needed to be easy to construct, to have a low material budget and have a high spatial resolution associated with a high radiation hardness. The characteristics required for this device derived by the necessity of a zero-degree calorimeter for feasibility studies at CERN, a beam condition monitoring for CMS and a better beam monitoring system for hadron facilities such as CNAO, Bern facility and MedAustron. From the study of the CMS beam radiation monitoring apparatus, the article [1] and the prototype on which this work is based have been derived.

The studied microcapillary detector consists of a single microfluidic channel filled with a liquid scintillator, designed to define a dense array of optically independent scintillating capillaries with dedicated photodetectors. Such devices allow the easy manipulation of fluids inside capillaries and give the possibility to circulate and renew the liquid scintillator, making the active medium of the detector intrinsically radiation hard; moreover, changing the type of scintillating liquid in the microchannels, the same device can be used for different types of measurements. This design is based on the assumption that there is low crosstalk between the different capillaries due to the right angles being necessary for fluidic circulation at the end of the straight sections. A schematic representation of the fabricated prototype detector is shown in figure 2.7.

The preliminary tests made on the microcapillary scintillators yielded a comparable performance of this device to scintillating fibres for thicknesses of the order of $500\,\mu m$ and some competitive results for thicknesses smaller than $300\,\mu m$. For lengths smaller than $100\,\mu m$, application for online beam monitoring can be envisaged having 100% of the beam directed towards the patient, a permanent monitoring of the radiation received by the patient and a circulation of the scintillator liquid that grants the needed radiation hardness.

2.5 Microcapillary tubes

The microcapillary tubes on which the studied detector is based represent the main innovation in this new kind of detector. In fact, thanks to the small size of these channels, a high spatial resolution can be achieved with a non-destructive effect on the beam. Conventionally, any channel or tube having hydraulic diameter less than

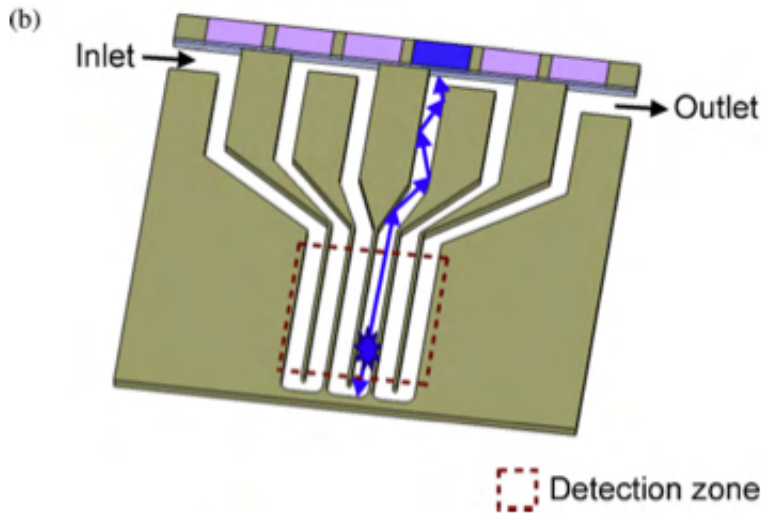


Figure 2.7. Schematic representation of the fabricated prototype detector [1].

1 mm is called a microchannel. Thus, fabrication becomes very important in this field of micro and nano domains. There are various fabrication techniques available for realizing these micro devices that can be divided into three main categories: photolithography, additive and subtractive techniques [18].

The photolithography technique consists of selectively removing parts of a thin photoresist film deposited on a top-oxidized substrate through light irradiation. Thanks to this process from a transparent mask a geometric pattern is transferred on the substrate, resulting in the desired microchannel fabrication.

The additive technique is instead a procedure where an either gaseous or liquid reactant is deposited on the substrate. This material deposition can occur due to the reaction between the added component and the heated substrate (CVD and PVD) or due to spin coating where the deposited material is rotated at high speed resulting in a homogenous spread on the substrate.

Lastly, the subtractive technique consists of etching the substrate, meaning that a strong acid or mordant is used to transfer onto the unprotected parts of the material surface a predesigned pattern. Recently, instead of using a chemical etching, beam of ions, electrons or photons have been used to bombard the material surface and evaporate the dug material.

The fabrication of the studied detector microchannels starts with the spin coating for $100\,s$ at $950\,rpm$ of a homogeneous layer of SU-8 [19] [20] [21] with a $200\,\mu m$ thickness. The SU-8 material is a "negative, epoxy-type, near-UV photoresist based on EPON SU-8 epoxy resin" [22] that has been developed by IBM. Such material has a refractive index of the order of $n \simeq 1.6$ (fig 2.8 shows n as a function of the incident

wavelength [23]), a density of $\rho = 1.2\,g/cm^3$ and its chemical formula is $C_{22}H_{22}O_4$. Moreover, SU-8 is a radiation hard material, being used for ions photolithography,

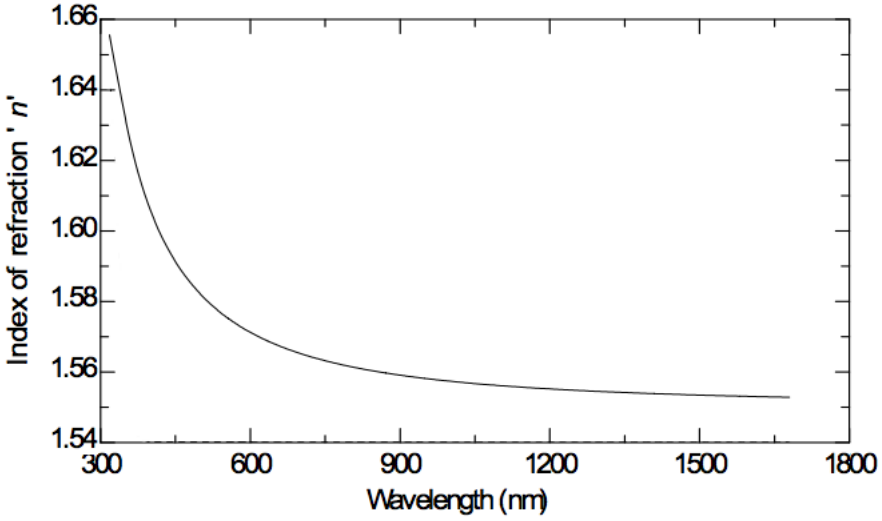


Figure 2.8. Refractive index as a function of the incident wavelength for the SU-8 material.

making it a good candidate for the case studied. From such values the radiation length can be expressed as:

$$\frac{M_0}{X_0} = \sum \frac{M_i}{X_i} \tag{2.5}$$

where M_0 and M_i are respectively the total mass of the sample and the mass of the individual components expressed in g while X_0 and X_i represent respectively the combined radiation length of the sample and the radiation length of the individual components expressed in g/cm^2; such equation yields a value of $41.65\,g/cm^2$ for the SU-8 material.

The layer of SU-8 is then deposited on a silicon wafer and it is slowly cooled down at a $4\,C/min$ rate, after being baked for $10\,min$ at $120\,C$; this fabrication process has been designed in order to prevent the formation of cracks in the microchannels. Having achieved this stage, in order to polymerize the coated substrate, this structure is exposed to UV light at a $500\,mJ/cm^2$ dose through a mask followed by a post exposure bake. Finally, in order to increase optical properties and prevent optical cross-talk between neighbouring microchannels, the surface of the channels are gold metallized through a sputtering technique [24]. A schematic representation of the cross-section of the microchannels at the level of the detection zone can be seen in figure 2.9 while the results of this procedure are shown in figure 2.10, 2.11 and 2.12.

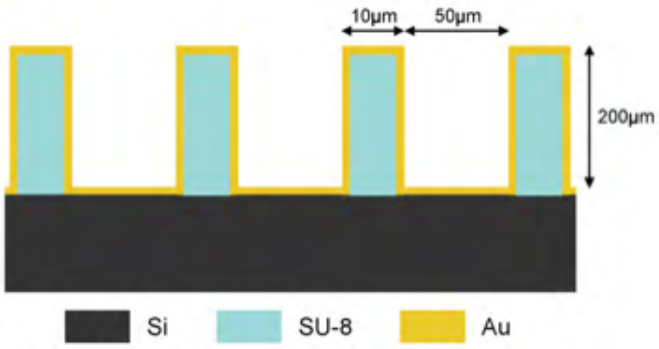


Figure 2.9. Schematic representation of the cross-section of the microchannels at the level of the detection zone [1].

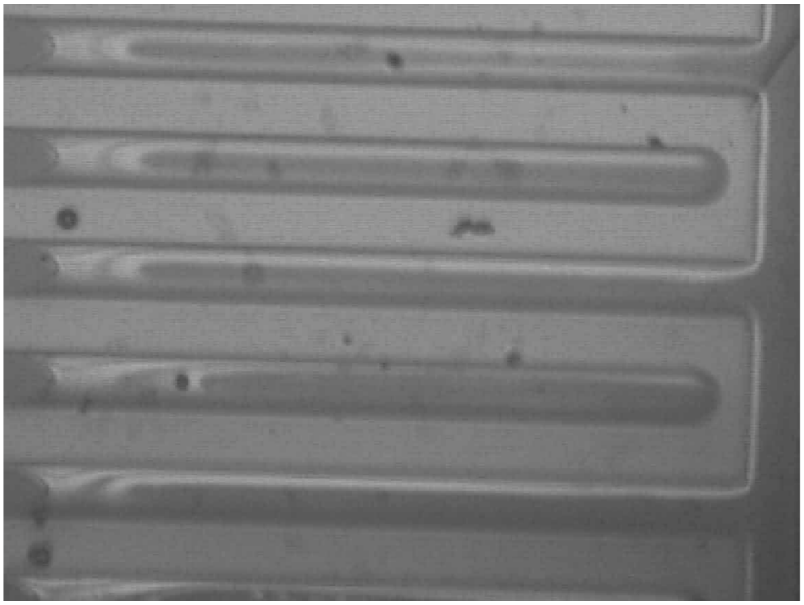


Figure 2.10. Side of a prototype array of SU-8 microchannels as built in 2012, where light is allowed to be transmitted and detected through the coupled photodetectors.

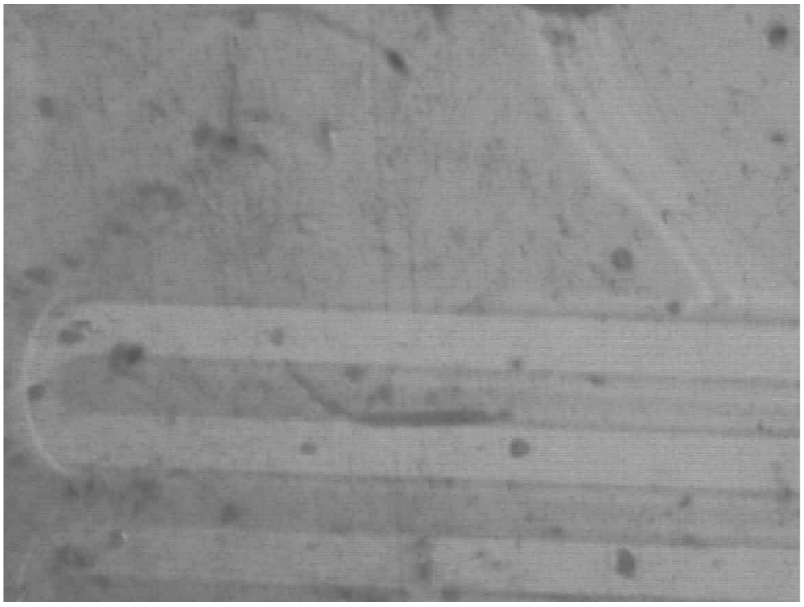


Figure 2.11. View of the insertion liquid microcapillary tube of a prototype array of SU-8 microchannels.

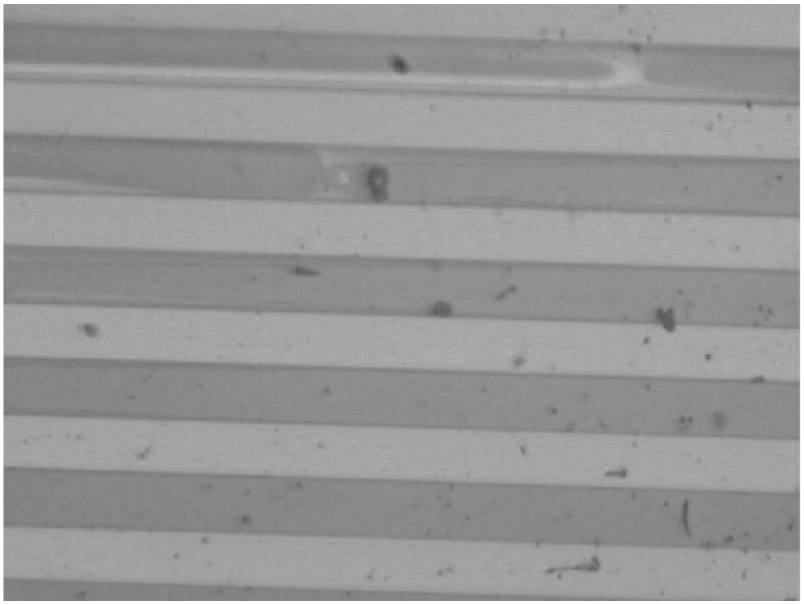


Figure 2.12. Top view of a prototype array of SU-8 microchannels.

2.6 The Detector Characteristics

Having covered in the previous section the theory needed to understand the detector, in the following part the microcapillary scintillator considered in this work will be characterized through the calculation of various physical parameters taking into account the readout system described in chapter four.

2.6.1 The Scintillator Detection Efficiency

The first physical value that can be theoretically calculated in order to give an idea of the potential of this device is represented by the scintillator detection efficiency; this value will be derived for a beam of protons with a typical CNAO energy (250 MeV momentum [25] [26]) and subsequently for a minimum ionizing particles (MIP). The liquid considered for the detection process is the EJ-305 produced by *Eljen Technology*.

Considering the Bethe-Bloch equation 2.3 and omitting the density effect and the shell correction due to their relatively small size for the considered energies, a stopping power of the liquid

$$dE_{CNAO}/dx = 15.87 MeV/cm \tag{2.6}$$

and

$$dE_{MIP}/dx = 1.8 MeV/cm \tag{2.7}$$

can be obtained for the two beams.

The depth of a scintillating channel can be considered on the order of 200 μm that, multiplied for the two stopping powers found, yields an energy deposited in the scintillating material of

$$\Delta E_{CNAO} = 0.32 MeV \tag{2.8}$$

and

$$\Delta E_{MIP} = 3.6 \times 10^{-2} MeV. \tag{2.9}$$

Finally, knowing from the EJ-305 datasheet [27] that every MeV of energy deposited in the scintillating material generates 12×10^3 photons with a wavelength of approximately 424 nm, the total number of photons produced in both cases can be calculated, obtaining $N_{\gamma CNAO} \simeq 3800$ and $N_{\gamma MIP} \simeq 430$; from these results a round number of $2 \cdot 10^3\, \gamma/mm\, per\, mip$ can be derived for quick approximated calculations.

From this result, assuming that the generated photons are guided in the microchannels by attenuated total internal reflection [28], the expected photoelectrons

produced can be expressed as follows [1]:

$$\overline{N}_{pe} = N\gamma \cdot \epsilon_{coll} \cdot \epsilon_{refl} \cdot \epsilon_{att} \cdot \epsilon_{in} \cdot \epsilon_{Qeff} \tag{2.10}$$

where $N\gamma$ represents the scintillations produced by an interacting proton, ϵ_{coll} is the collection efficiency of a rectangular metal-coated microchannel, ϵ_{refl} the gain caused by the reflective end of the channel opposite to the photodetector, ϵ_{att} the transport efficiency due to optical absorption, ϵ_{in} the transmission efficiency at the interface between the microchannel and the photodetector and ϵ_{Qeff} the quantum efficiency of the chosen readout system. For the calculations reported in this section, the previously cited parameters have been kept constant, taking their values from the prototype used in [1], but considering a different photodetector and varying the thickness of the scintillating material.

Estimating the efficiency ϵ_{coll} and ϵ_{refl} respectively 0.03 and 1.4 through Monte Carlo simulations [1], omitting ϵ_{att} due to its large value compared to the small size of the detector, considering ϵ_{in} to be around 0.99 and knowing the ϵ_{Qeff} of the photodetector to be 0.2 for photons of a $424\,nm$ wavelength, an expected photoelectric yield of approximately 22 and 2.5 is obtained respectively for the CNAO-like and the MIP beam.

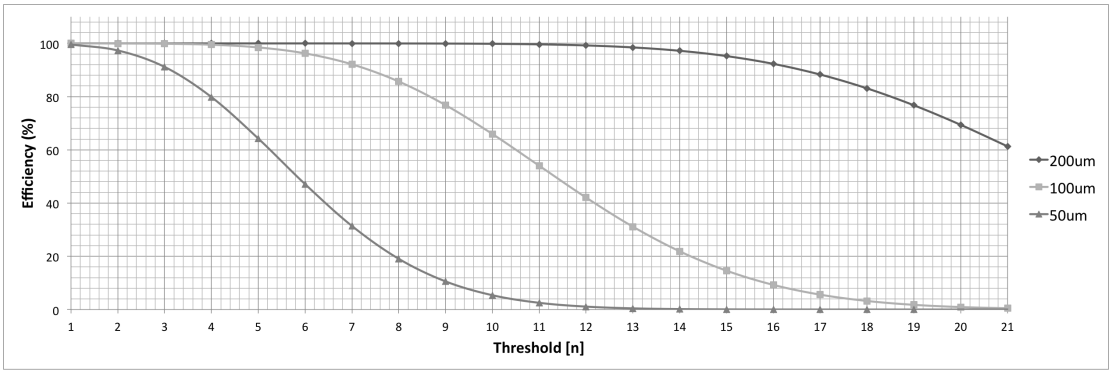
Modelling the interaction between the photodetector and the photodiodes as a poissonian statistical process, the final detection efficiency ϵ_{det} defined as the probability of observing one photon, will be given by one minus the probability of not detecting any scintillation, that is:

$$\epsilon_{det} \approx 1 - P(0, \overline{N}_{pe}) = 1 - e^{-\overline{N}_{pe}} \tag{2.11}$$

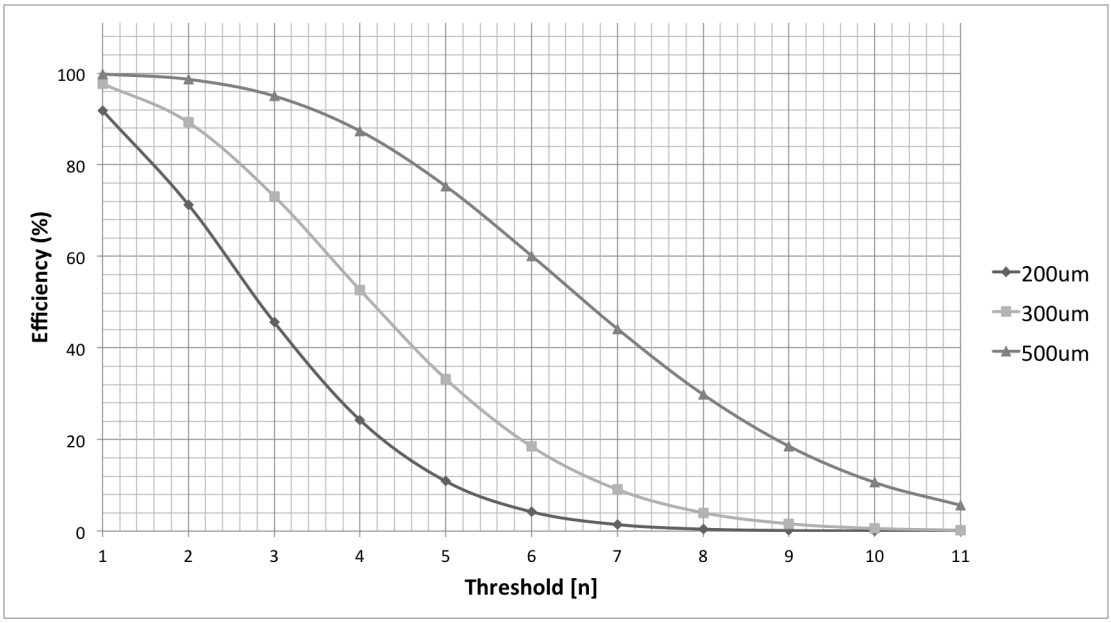
obtaining a 100% and 92% efficiency for the two beams on single photon counting.

Varying the depth of the scintillating channel and the minimum number n of photons detected by the photodetector per interaction, the graphs 2.13a and 2.13b have been produced, calculating the detection efficiency as:

$$\epsilon^{n}_{det} \approx 1 - \sum_{i=0}^{n-1} P(i, \overline{N}_{pe}) = 1 - e^{-\overline{N}_{pe}} \cdot \sum_{i=0}^{n-1} \left(\frac{\overline{N}_{pe}^{\,i}}{i!} \right). \tag{2.12}$$



(a)



(b)

Figure 2.13. Theoretical detection efficiency of the studied microcapillary scintillator detecting a CNAO-like (a) and MIP (b) proton beam, varying the depth of the scintillating channel and the minimum number of photoelectrons needed by the photodetector in order to reveal the interaction (called threshold in figure).

Chapter 3

Photosensors

The great diffusion of scintillating detectors for spectroscopy and high energy physics purposes has increased the usage and improved the technology of photodetectors. This kind of devices are particularly well designed for the conversion of the scintillators' output in electrical signals.

The photodetectors are capable of converting a really weak light signal, typically constituted by one hundred photons, in an electric signal, easily handled by a modern readout technology. Furthermore the quantity of current produced is directly proportional to the amount of light absorbed and the timing information carried by the photons are reproduced in the electric signal. For such reasons the photodetectors find a large use in optical spectroscopy, in measurements made through laser technologies, in astronomy, in high energy physics and in health applications.

In this chapter the available sensor technologies will be presented with their strengths and weaknesses in order to understand in which research field each devices is best suited.

3.1 The Photodiodes

Historically photomultipliers tubes (PMT) have been the first photodetectors designed; despite their diffusion in high energy physics this technology had quantum efficiency and resolution limitations. The quantum efficiency is defined as:

$$QE = \frac{number\ of\ photoelectrons\ emitted}{number\ of\ photons\ absorbed}. \tag{3.1}$$

For commercial photomultipliers tubes such quantum efficiency assumes a value between $20 - 40\%$ [29]. With the evolution of photodetectors technology, these devices have been replaced by photodiodes thanks to the progress made by solid

state research. These devices have a better quantum efficiency (typically around 80%), resulting in a better energy resolution; they also have a lower power consumption and their more compact structure is not affected by strong magnetic fields. Due to all these qualities the photodiode technology has almost completely replaced PMT devices in every application field. However, photodiodes have the drawback of needing a more intense light signal compared to the few photons required by a common PMT. Furthermore, the current response of a commercial photodiode requires a preamplifier circuit.

During the experiments made in the course of history, different types of photodiodes have been designed in order to stress some of the previously mentioned characteristics to the detriment of others; the structural outline of some commercial photodiodes are shown in figure 3.1.

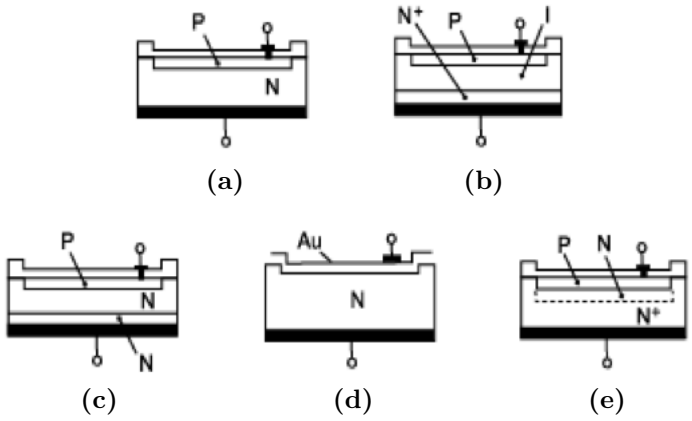


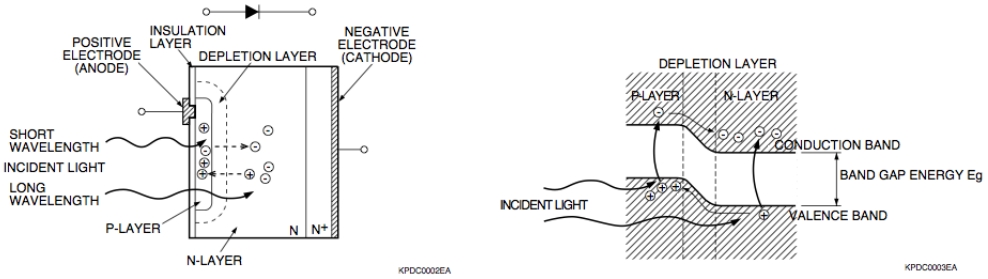
Figure 3.1. Structural outline of some commercial photodiodes [30].

In this section the working principles on which the photodiode technology is based will be covered; in particular the PN photodiode type will be explained due to its simplicity and good example of the mechanisms on which this technology is based.

3.1.1 The PN Photodiode

The PN Photodiode structural outline is shown in figure 3.2a. In this PN junction the positive layer is a photoelectric converter while the N layer is the a substrate. The intermediate zone is called *depletion layer* and the electrons produced in this area are accelerated due to the presence of an electric field. Varying the dimensions of these zones and their doping, it is possible to obtain the desired frequency spectral response of the designed device.

When the photons collide with the active surface of the photodiode, if their



(a) (b)

Figure 3.2. Structural outline of a PN Photodiode [30].

energy is greater than the gap energy of the device composing material, the atomic electrons are excited and some electron-hole pairs are generated (figure 3.2b). Due to the presence of an electric field in the depletion layer, the electrons are then accelerated toward the N-type layer while the holes are sped up toward the P-type. With this technique the P layer is positively charged while the N zone negatively; thanks to these charges it is possible to extract a weak current proportional to the incident photons on the device.

3.1.2 The Photodiode Equivalent Circuit and Noise

For the analytic coverage of this device a general outline is needed. One of the most common model found in literature is given by the equivalent circuit shown in figure 3.3.

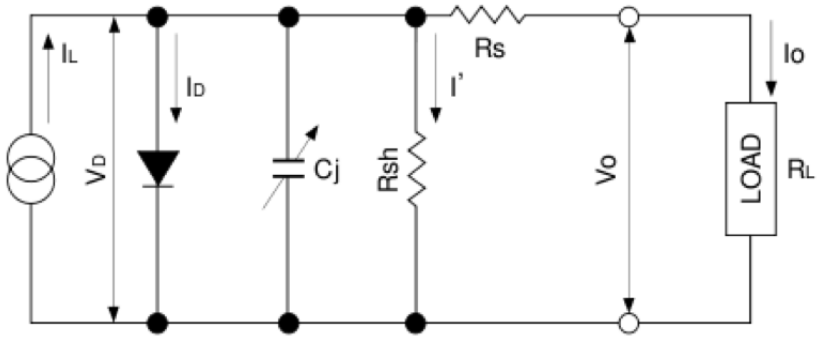


Figure 3.3. Equivalent circuit of a PN Photodiode [30].

In this model the various physical parameters represent:

- I_L: Current generated by the incident light. It can be analytically expressed in terms of the sensitivity and the incident photons' number as:

$$I_L = S \cdot N_\gamma \cdot h\nu \tag{3.2}$$

where S is the sensitivity of the photodiode expressed in A/W, N_γ the number of reacting photons per second, ν their frequency and h the Planck constant.

- I_D: Diode current or dark current; in other words it is the current that flows through the diode when the device is not irradiated by the photons beam. This current is the main noise term when the device is reversely polarized. Its analytical expression is given by the Shockley's law:

$$I_D = I_S(\exp \frac{V_D}{\eta V_T} - 1) \tag{3.3}$$

  where I_S is the saturation current, V_D the potential energy between the two diode's ends, η the diffusion coefficient depending on the constituent material and V_T the thermic tension.

- R_s: Series resistance that represent the ohmic junction resistance. In some photodiodes this resistance is increased by the presence of a high resistance material, placed in the diode for breakdown protection reasons when the device is reversely polarized.

- R_{sh}: Shunt resistance that is generated by the superficial leakage currents. This resistance is the main noise term when the device is not polarized while it becomes pretty small when the diode is reversely polarized.

- I': Current that flows through the shunt resistance.

- C_j: Junction capacity.

- V_D: Diode voltage.

- I_o: Output current.

- V_o: Output voltage.

From the equivalent circuit shown it is possible to derive the analytical expression for the output DC current as a function of the applied voltage, obtaining:

$$I_o = I_L - I_D - I' = I_L - I_s(\exp \frac{eV_D}{kT} - 1) - I' \tag{3.4}$$

where I_s is the saturation current, e the electron charge, k the Boltzmann constant and T the diode temperature expressed in Kelvin.

In figure 3.4 the obtained expression can be seen; it needs to be underlined how this graph represents the characteristic curve of a normal diode shifted to more negative current values as the incident light beam intensity increases.

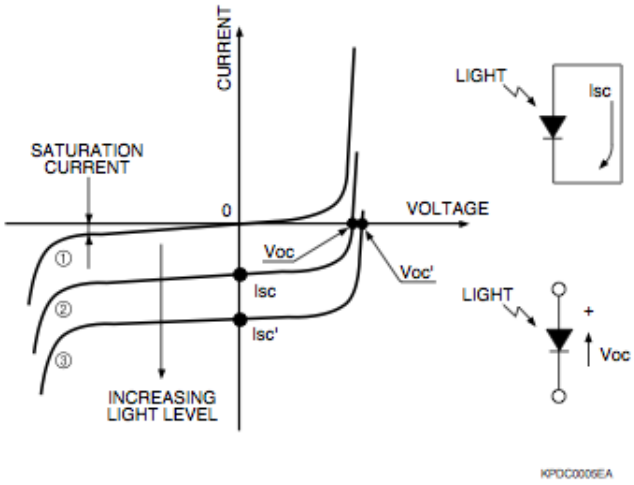


Figure 3.4. Analytical expression for the photodiode output current as a function of the applied voltage [30].

The main noise contribution to the photodiode device is given by the p-n junction. In order to model this noise a low frequency approximation ($f < \frac{1}{\tau_c}$, with τ_c mean life of the charge carriers) is used:

$$< n_D^2 >= 2q_e(I_D + 2I_S) = 2q_e I_S M^2(\exp \frac{eV_D}{kT} + 1) \tag{3.5}$$

where $< n_D^2 >$ is expressed in A^2/Hz.

3.2 The Avalanche Photodiodes

A photodetector alternative to the photodiodes is represented by the avalanche photodiodes (APD). This particular type of devices have been designed in order to read a weak scintillator output without changing the photodiode quantum efficiency; in fact, as we have seen in the previous section, photodiodes have a higher quantum efficiency compared to PMTs.

3.2.1 The APD Working Principle

The working principle on which the avalanche photodiodes are based is explained using the figure 3.5. The structure of this device is similar to the photodiode one showing two doped zones and a depletion region. The device is highly reversely polarized in order to diminish the width of the depletion region and to increase the internal electric field. Thanks to such electric field the generated electrons are accelerated and a high kinetic energy that lets them ionize the material's atoms

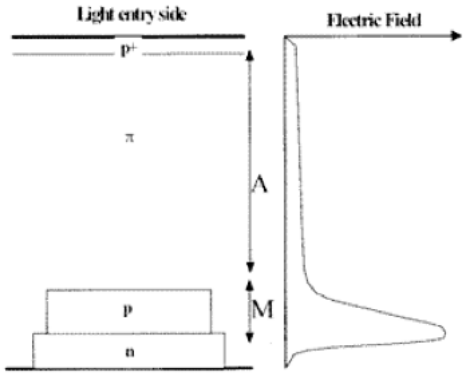


Figure 3.5. Structural outline and internal electric field of an avalanche photodiode [31].

is gained; the ionized electrons are then accelerated by the electric field causing themselves other ionization processes: an avalanche multiplication is therefore produced. Thanks to this process a current gain is obtained in the device, acquiring the possibility to detect small amounts of light, comparable to the ones detected by commercial PMTs. An example of the general characteristic of commercial APD can be seen in figure 3.6.

• •Electrical and optical characteristics (Typ. Ta=25 °C, unless otherwise noted)

Type No.	Spectral response range λ (nm)	Peak *3 sensitivity wavelength λp (nm)	Photo sensitivity S M=1 λ=420 nm (A/W)	Quantum efficiency QE M=1 λ=420 nm (%)	Breakdown voltage VBR ID=100 µA Typ. (V)	Breakdown voltage VBR ID=100 µA Max. (V)	Temperature coefficient of VBR (V/°C)	Dark *3 current ID Typ. (nA)	Dark *3 current ID Max. (nA)	Cut-off frequency fc (MHz)	Terminal *3 capacitance Ct (pF)	Excess *3 Noise index λ=420 nm	Gain M λ=420 nm
S8664-02K	320 to 1000	600	0.24	70	400	500	0.78	0.1	1	700	0.8	0.2	50
S8664-05K								0.2	1.5	680	1.6		
S8664-10K								0.3	3	530	4		
S8664-20K								0.6	6	280	11		
S8664-30K								1	15	140	22		
S8664-50K								3	35	60	55		
S8664-55								5	50	40	80		
S8664-1010								10	100	11	270		

*1: K: Borosilicate glass E: Epoxy resin
*2: Area in which a typical gain can be obtained.
*3: Values measured at a gain listed in the characteristics table.

Figure 3.6. Structural outline and internal electric field of an avalanche photodiode [32].

3.2.2 The APD Equivalent Circuit and Noise

Analogously to the photodiode technology, a general outline is needed for the analytic coverage of these devices. One of the most common models found in literature is given by the equivalent circuit shown in figure 3.7.

In this model the various physical parameters represent:

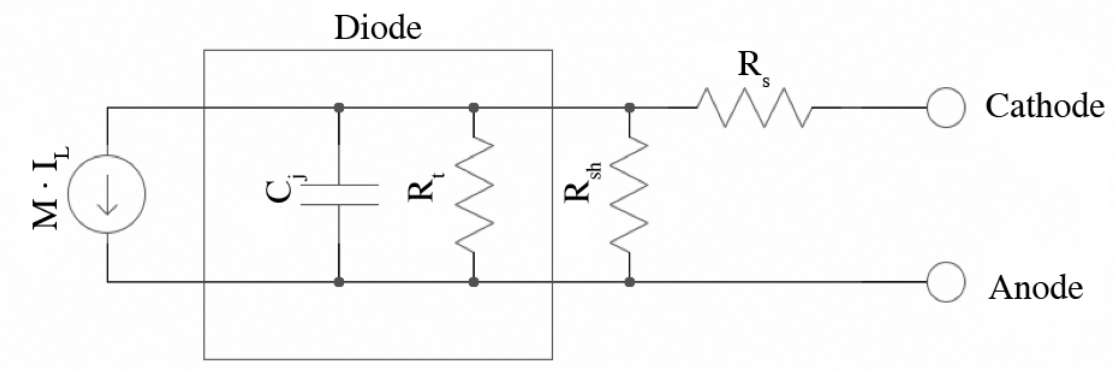


Figure 3.7. Equivalent circuit of an avalanche photodiode.

- $M \cdot I_L$: Current generated by the incident light. Compared to the photodiode generated light it has a multiplying factor M representing the avalanche process, typically of the order of 50.
- R_t: Differential resistance internal to the diode. It can be expressed as $R_t = \delta V_D / \delta I_D$ where V_D is the diode output voltage. It represents the generation of the dark current I_D amplified by the multiplying factor M; similarly to the photodiode, the analytical expression of the dark current is given by the formula:

$$I_D = M \cdot I_S(\exp \frac{V_D}{\eta V_T} - 1) \tag{3.6}$$

  where $M \cdot I_S$ is the saturation current, V_D the potential energy between the two diode's ends, η the diffusion coefficient depending on the constituent material and V_T the thermic tension.

- R_{sh}: Shunt resistance that is generated by the superficial leakage currents. It can be neglected in most of the cases.
- C_j: Junction capacity.
- R_s: Series resistance that represents the ohmic junction resistance.

From this electric circuit the main sources of noise can be derived and represented in three distinguished elements: the light shot noise, the diode shot noise and the series resistance thermic noise.

The light shot noise can be analytically express by the equation 3.7.

$$< n_L^2 >= 2 q_e I_L M^2 F \, [A^2/Hz] \tag{3.7}$$

In this equation $M \cdot q_e$ is the minimal charge variation due to the avalanche process, $M \cdot I_L$ the produced current and F the multiplying adimensional factor that takes into account the $1/f$ noise.

The diode shot noise can be calculated instead with the following equation:

$$< n_D^2 >= 2q_e I_S M^2 (\exp \frac{V_D}{\eta V_T} + 1) \, [A^2/Hz] \tag{3.8}$$

where, similarly to the previous case, $M \cdot q_e$ is the minimal charge variation and $M \cdot I_S(\exp \frac{V_D}{\eta V_T} + 1)$ represents the two currents producing the noise.

The last noise term is the series resistance thermic noise that can be represented with the equation 3.9.

$$< n_S^2 >= \frac{4k_B T}{R_S} \, [A^2/Hz] \tag{3.9}$$

In this equation k_B is the Boltzmann constant, R_S the series resistance and T the temperature expressed in kelvin.

Knowing these three noise terms the total APD noise can be calculated with the following equation:

$$< n_{APD}^2 >= \Sigma_i \frac{1}{2\pi} \int_0^\infty < n_i^2 > |H_i(j\omega)|^2 d\omega \, [A^2] \tag{3.10}$$

where $< n_i^2 >$ are the different noise contributes and $H_i(j\omega)$ the device's transfer function.

3.3 The Multi-Pixel Photon Counter

The last photosensor that will be covered in this chapter is represented by the Multi-Pixel Photon Counter. MPPCs are new devices designed to count single incident photons. They are based on the previously explained APD detectors placed in parallel but working in geiger mode. An APD working in geiger mode is defined as an avalanche photodiode reversely polarized with a tension greater than its breakdown threshold; in this way the device gain increases to values around $10^5 - 10^6$ losing its linearity and therefore becoming capable of detecting only a single photon. For such reason the APD working in geiger mode are shortly called *Single Photon Avalanche Diodes* or SPAD.

3.3.1 The MPPC Operating Principles

As in the avalanche photodiodes theory, due to the collision between the incident photon and the active surface of the device, one or more electrons are generated and accelerated by a strong electric field. Thanks to this strong acceleration the

generated electrons can start the avalanche process. Due to the presence of an electric field, a sudden flow of current through the device is triggered, with delays of the order of the ns, at a stable current of the order of $10\,\mu A$. This process is called *Geiger discharge*. Due to charge fluctuations in the depletion zone, if the breakdown current is limited by a resistance, the device has a high probability to fall in a metastable interdiction state, stopping the current flow (this phenomenon is called *quenching*). The photon interaction with the device surface therefore produces the starting avalanche electron that causes a pulse generation, indicating the interaction time. Placing more SPAD in parallel (figure 3.8) and measuring the output current, it is possible to know how many devices fired and consequently how many photons have been collected.

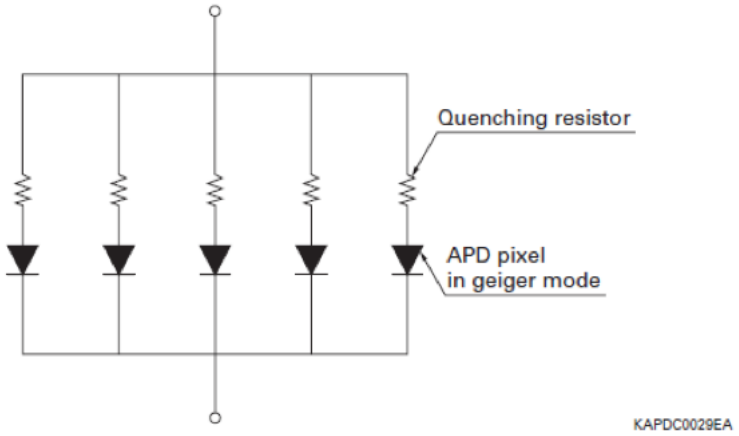


Figure 3.8. Structural outline of a MPPC [33].

3.3.2 The MPPC Equivalent Circuit and Its Noise Sources

Analogously to the other cases, an analytical treatment of these devices is needed. In order to do so the equivalent circuit shown in figure 3.9 is usually used.

In this model the elements of the circuit represent the following aspects of the device:

- V_s: Supply voltage which is usually set around $70\,V$.
- V_B: Breakdown voltage which usually equals $69\,V$.
- R_L: Load resistance that is responsible of the quenching process.
- R_s: Matching resistance; it is used in order to match the output impedance, typically of the order of 50Ω.

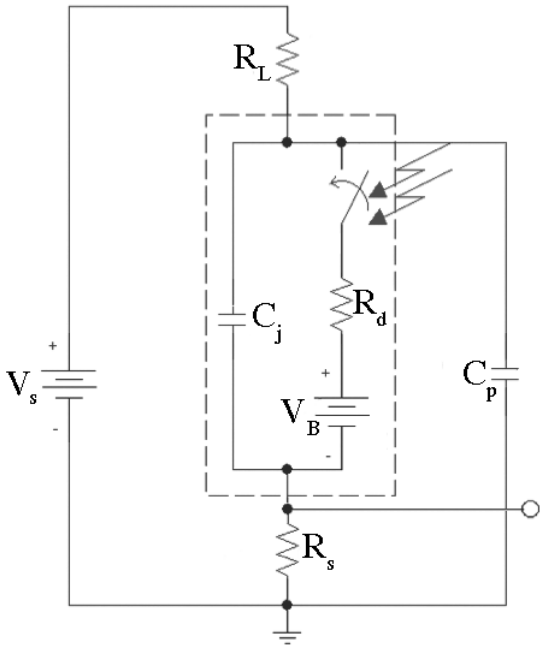


Figure 3.9. Equivalent circuit of a MPPC.

- R_d: Series resistance that represent the ohmic diode junction resistance.
- C_j: Junction capacity.
- C_p: Parasitic capacity.

Having defined these variables, the MPPC's working process can be characterized by three physical parameters: the quenching current, the quenching period and the restoring period. The quenching current is defined as the current threshold after which the restoring period can start. The quenching period is determined as the time interval during which the MPPC can not reveal any other interacting photon. Finally the restoring period is defined as the time interval spent by the device reverting its tension to the initial value V_s. An image of a typical progress of these three parameters is shown in figure 3.10.

In literature, since the MPPCs have a large gain, instead of being described through a current noise analysis, this kind of devices are more often characterized by a spurious counts study. Spurious counts can derive from five different contributes: the statistical counting uncertainty, the dark current extra counts, the cross-talk phenomenon, all the incident photons that are not counted and the afterpulse contribute.

The statistical counting uncertainty comes from the probabilistic nature of the interaction between the incident photons and the device's surface. These interactions

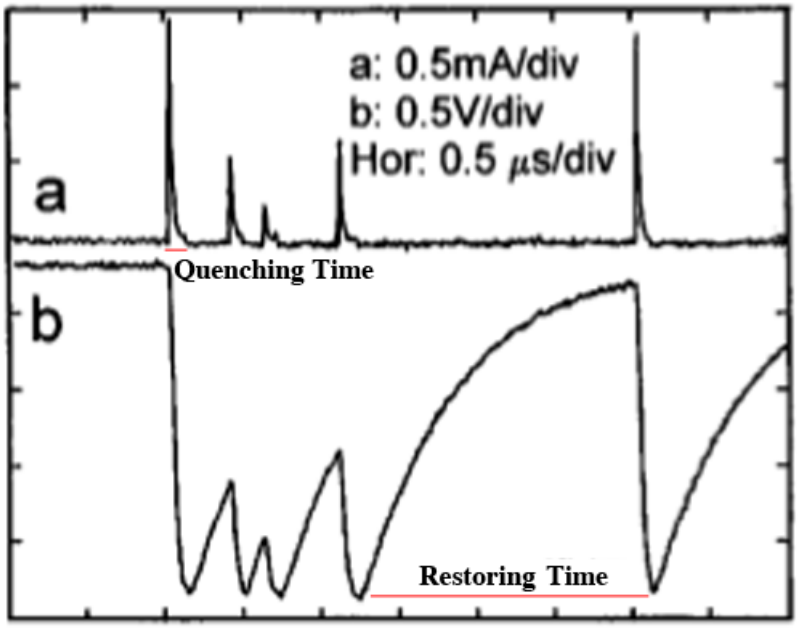


Figure 3.10. Typical progress of a MPPC device [33] where the current is expressed in $0.5\,mA/div$ and the voltage in $0.5\,V/div$.

can be modelled as a Poissonian process, giving a mean variation of:

$$\sigma_{stat} \propto \sqrt{N_{count}} \tag{3.11}$$

where N_{count} is the number of photons counted.

The dark currents extra counts are generated by the thermoelectric charges produced by the APD pixel leakage current that can start an avalanche process. The value for these extra counts is usually written in the device's datasheet and has often a value of the order of $100\,kcps$.

The cross-talk phenomenon between different pixels can generate some other extra counts. One of the causes of this process is due to the second electron emission of the activated APD.

The charges generated from the interaction between the photons and the surface of the device can sometimes remain trapped in the depletion region impurities and escape from it during the APD interdiction state, reactivating it. This additional counts are often called afterpulses.

The last spurious counts contribution is given by the nonlinearity of the device; in fact, being the APDs only capable of detecting single photons, if a MPPC detects two photons that simultaneously impact the same cell, a wrong photons count will be yielded. The proportion between the activated SPADs and the incident photons

will therefore be given by:

$$N_{fired} = N_{tot}[1 - \exp^{-\frac{N_{ph} \cdot PDE}{N_{tot}}}] \tag{3.12}$$

where N_{fired} is the number of SPADs activated, N_{tot} the total number of SPADs, N_{ph} the number of photons in the beam, and PDE is the photodiode efficiency. In figure 3.11 multiple parameters of some commercial MPPCs are shown.

Specifications (Typ. Ta=25 °C, unless otherwise noted)

Parameter	Symbol	S10362-11 series			Unit
		-025U, -025C, -025P	-050U, -050C, -050P	-100U, -100C, -100P	
Effective active area	-	1 × 1			mm
Number of pixels	-	1600	400	100	-
Pixel size	-	25 × 25	50 × 50	100 × 100	μm
Fill factor *1	-	30.8	61.5	78.5	%
Spectral response range	λ	320 to 900			nm
Peak sensitivity wavelength	λp	440			nm
Photon detection efficiency *2 (λ=λp)	PDE	25	50	65	%
Operating voltage range	-	70 ± 10 *3			V
Dark count *4	-	300	400	600	kcps
Dark count Max. *4	-	600	800	1000	kcps
Terminal capacitance	Ct	35			pF
Time resolution (FWHM) *5	-	200 to 300			ps
Temperature coefficient of reverse voltage	-	56			mV/°C
Gain	M	2.75×10^5	7.5×10^5	2.4×10^6	-

*1: Ratio of the active area of a pixel to the entire area of the pixel
*2: Photon detection efficiency includes effects of crosstalk and afterpulses.
*3: For the recommended operating voltage of each product, refer to the data attached to each product.
*4: 0.5 p.e. (threshold level)
*5: Single photon level
Note: Each value was measured at recommended operating voltage (refer to the data attached to each product).
The last letter of each type number indicates package materials (U: metal, C: ceramic, P: SMD).

Figure 3.11. Table of parameters of some commercial MPPC [34].

3.4 Detectors Readout

Since microcapillary scintillators are being designed for both hadrotherapy and high energy experiments, they need to be readout by different photodetectors, depending on the requests of each experiment. In fact, using a MPPC readout, these devices could be used, for example, as single particle trackers or time flight measurement systems; such a choice would be lead by interactions that these devices can detect a signal generated by just a few photons. In other experiments instead, where the time integration of a large number of particles is needed, the designer choice would probably end up on the photodiode technology; examples of such applications could derive from a beam profile or dosimetry monitoring system.

In this master thesis work, a photodiode array readout technology has been chosen

to investigate the beam monitoring capability of the detector. Other readout systems are being studied in order to realize a detector suitable for all the cases previously explained. In the next chapter the electronics developed for the photodiode array readout technology will be discussed together with the first results obtained.

Chapter 4

The Photodiode Readout

In this chapter the photodiode array readout system structure will be discussed in detail. The system shown has been built in order to manipulate the scintillator output signal and convert it into physical data statistically analyzable.

4.1 System Requirements and Specifications

The readout system described in this chapter has been realized having in mind well-defined requirements. Since the studied detector has been conceived as a multi purpose device, the readout system needed to be designed as versatile as possible. In particular, the working frequencies needed to be easily varied in order to quickly adapt the monitor system to different kind of beams. The developed device has therefore been realized with the following technical specifications:

Parameter	*Value*	*Units*
f_{phDet}	$40 \div 2000$	kHz
f_{adc}	40	MHz
n_{Bit}(ADC)	14	Bit
Integration Time	$0.5 \div 1,64 \cdot 10^6$	μs
Input Voltage	4.8	V
FIFO Length	2056	32bits words
Ethernet based control and readout (Gigabit Ethernet)		

where f_{adc} is the frequency of the adc clock, f_{phDet} is the frequency of the photodetector clock and n_{Bit}(ADC) the number of bits used by the ADC to digitalize the data.

4.2 A General Overview

The front end electronics, assembled for the studied microcapillary scintillators, is capable of converting light in an electronic signal, by sampling the analog stream from the photodiode array with a high resolution $14 - bit$ ADC. Having formatted the digital signal into a pre-established format, the final data is then sent through an ethernet network to a storage system. A block diagram of the system is shown in figure 4.1.

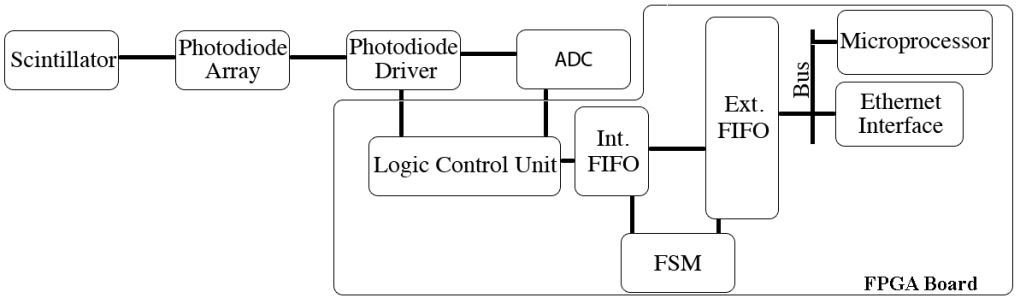


Figure 4.1. Logical outline of the operations made on the microcapillary detector output.

As it can be seen from the picture above, the first operation to be made is the conversion of the light produced by the scintillator into a digital signal through a photodetector and an ADC. Such conversion needs to be synchronized with the readout internal clock and its integration time needs to be programmable, in order to fulfil the versatility requirement. The readout system chosen is a photodiode array that will be described in detail in the next section. The analog signal produced is then digitized. The analog to digital conversion is performed by an ADC that serially sends its output to a deserializer contained in the Logic Control Unit. Having digitally obtained the measurement, a final data is composed, formatting the data in a form that will be shown in section 4.2.5. Having achieved this stage, the word is then sent to a microprocessor through a two FIFOs system controlled by a finite state machine (FSM). Afterwards, the microprocessor encapsulates the received data in a UDP message and sends the final message to a computer interfaced through an ethernet network. The structure and the working mode of each device used in this data processing flow will be covered in the upcoming sections.

4.2.1 The Photodiode Array

The photosensor chosen is the amplified photodiode array S8865 and its driver circuit C9118, commercialized by the *Hamamatsu Photonics* company. These photodetectors

are Si photodiode arrays with a combined signal processing IC chip. Such chip is in CMOS technology and incorporates a timing generator, a shift register, a charge amplifier array and a clamp and hold circuit, making the external configuration simple. Using the appropriate driver circuit, it is possible to arrange multiple arrays in a row in order to configure a long image sensor. This photodiode array has been chosen for its simplicity, reliability and low cost. The spectral response of this device is shown in figure 4.2 while its timing chart in figure 4.3 and its driver circuit diagram in figure 4.4.

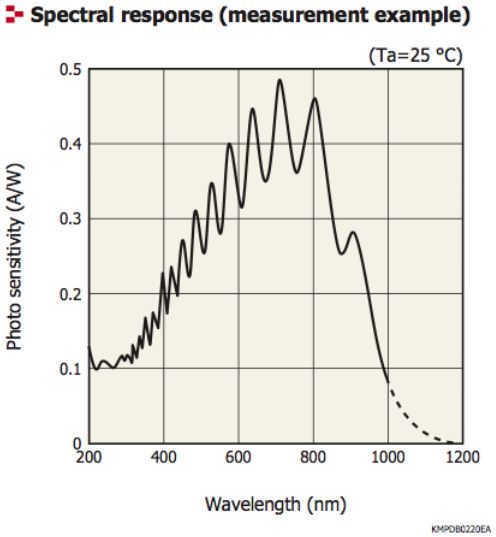


Figure 4.2. Spectral response of the S8865 photodiode array [35].

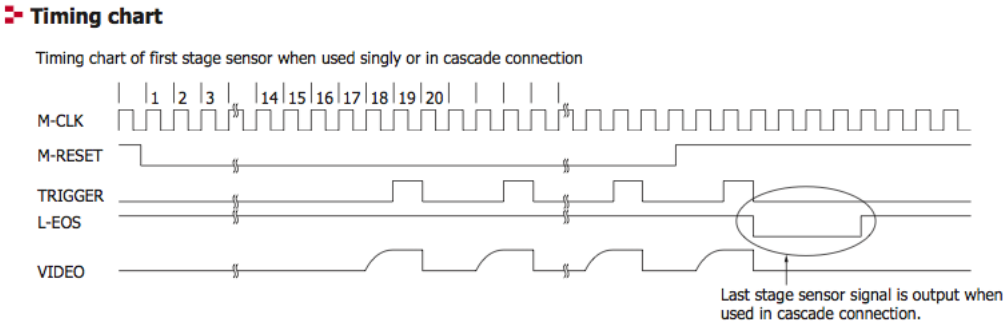


Figure 4.3. C9118 driver circuit timing chart [36].

As it can be seen from fig 4.2, the photosensitivity peak does not match exactly the input light wavelength of 424 nm of this scintillator. Due to such mismatching, a different photodetector would be needed to achieve better performances. However, during the development period in which this work has been carried out, the

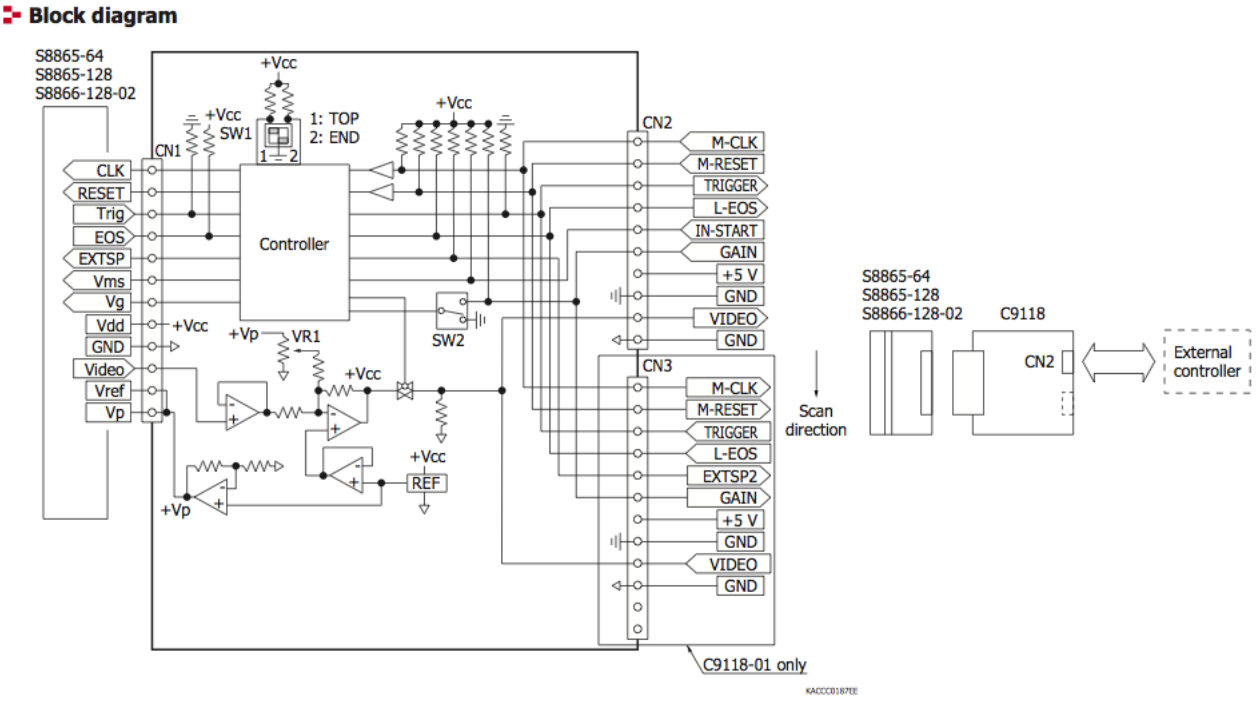


Figure 4.4. C9118 driver circuit block diagram [36].

Hamamatsu photodiode arrays were the best photodetectors available to our group.

During the data acquisition process, two different photodiode arrays of the S8865 family have been used. In particular, the S8865-128 and the S8865-256 have been mounted one at a time on the readout system. In order to calculate the signal to noise ratio and give an estimate of the quality of the readout system, the signal photodiode amplitude has been calculated for different types of light input; this physical parameter can be mathematically expressed as:

$$V_{sig} = \frac{kSF_p\overline{N}_{pe}A_{ch}h\nu T_{int}}{A_{pd}e} \tag{4.1}$$

where

- k is a constant equal to $13.66\,cd/W$
- S represents the photodetector photosensitivity
- F_p stands for the incident beam flux
- $\overline{N}_{pe}$ is the expected photoelectrons produced by a single particle
- A_{ch} represents the cross section of a microcapillary channel

- h stands for the Planck constant
- ν is the scintillations' frequency
- T_{int} represents the photodetector integration time
- A_{pd} stands for the surface of the photodetector window
- e is the electron elementary charge.

Taking the photosensitivity value from the S8865-128 photodiodes data sheet [35], considering a CNAO-like particle beam with a flux of $F_p = 0.25 \times 10^{14}$ particles $m^{-2}s^{-1}$ and assuming an expected photon number of 22 and 2.5 respectively for a $250\,Mev$ and a MIP protons (such numbers have been derived in section 2.6.1 for a thickness of $200\,\mu m$), a cross section channel of $5 \times 10^{-7}\,m^2$, a photodetector surface of $1.8 \times 10^{-7}\,m^2$ and an integration time of $100\,ms$, a signal amplitude of approximately $737\,mV$ and $84\,mV$ is obtained.

In general, a value of approximately $0.35\,\mu V/mip$ can be derived for this particular scintillator-photodetector system for a thickness of $200\,\mu m$. Moreover, in the data sheet [35] a value of $0.01\,mV/ms$ for the dark output voltage per integration time is reported. Therefore, the number of photons that can be distinguished from the dark current by the photodiode can be derived. In particular, considering an integration time of $1\,ms$ and knowing that each photon generates a signal output of the order of $0.10\mu V$, the lower boundary results in a value less than 100 photons. From such result, it can be said that the dark contribute is negligible compared to the measured data taken during the tests described in chapter 5, since they typically are two order of magnitude greater.

4.2.2 The ADC

The *analog-to-digital converter* chosen for this work is the ADS7946 device, produced by the *Texas Instruments* company. The ADS7946 is a 14-bit analog-to-digital converter, with unipolar inputs. The device operates at a 2 MSPS sample rate with a standard 16-clock data frame and it features a sufficient dc precision and dynamic performance. This device includes a two-channel input multiplexer and a low-power successive approximation register (SAR) ADC with an inherent sample-and-hold (S/H) input stage. The ADS7946 support a wide analog supply range that allows the full-scale input range to extend to $4.8\,V$ single-ended and has an automatic power-down feature that can be enabled when operating at slower speeds to reduce power consumption. A simple Serial Peripheral Interface Bus (SPI), with a digital supply that can operate as low as $1.65\,V$, allows for an easy interfacing to a wide variety of digital controllers. This device has been chosen as a good compromise between

its cost and performance. The ADS7946 time and block diagram are respectively shown in figure 4.6 and 4.5.

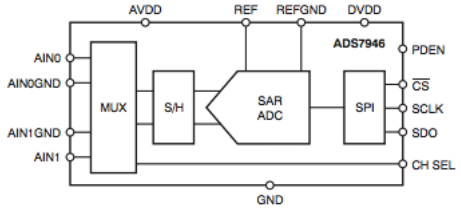


Figure 4.5. ADS7946 block diagram [37].

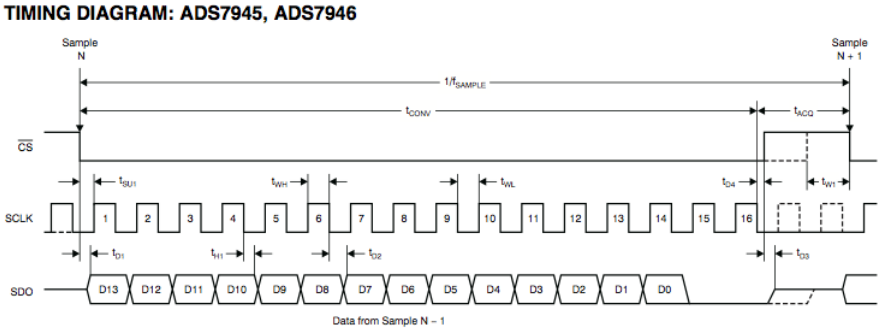


Figure 4.6. ADS7946 timing diagram [37].

4.2.3 The FPGA Board

The FPGA used in the development of the front end electronics is the Virtex-5 LXT ML505 evaluation platform produced by the *Xilinx* company and, as the name suggests, it is based on the Virtex-5 technology. The device is a general purpose FPGA development board with a good balance of high-performance logic, serial connectivity, signal processing, and embedded processing resources. It provides a feature-rich general purpose evaluation and development platform, including on-board memory and industry standard connectivity interfaces and delivering a versatile development framework for embedded applications. In particular, in addition to the embedded RISC processor blocks, integrated system-level hard-IP blocks for Peripheral Component Interconnect Express (PCIe), Tri-mode Ethernet, and advanced high-speed RocketIO GTP and GTX serial transceivers are also provided through the Virtex-5 FPGA family. A block diagram and a picture of the board can be seen respectively in fig 4.7 and 4.8.

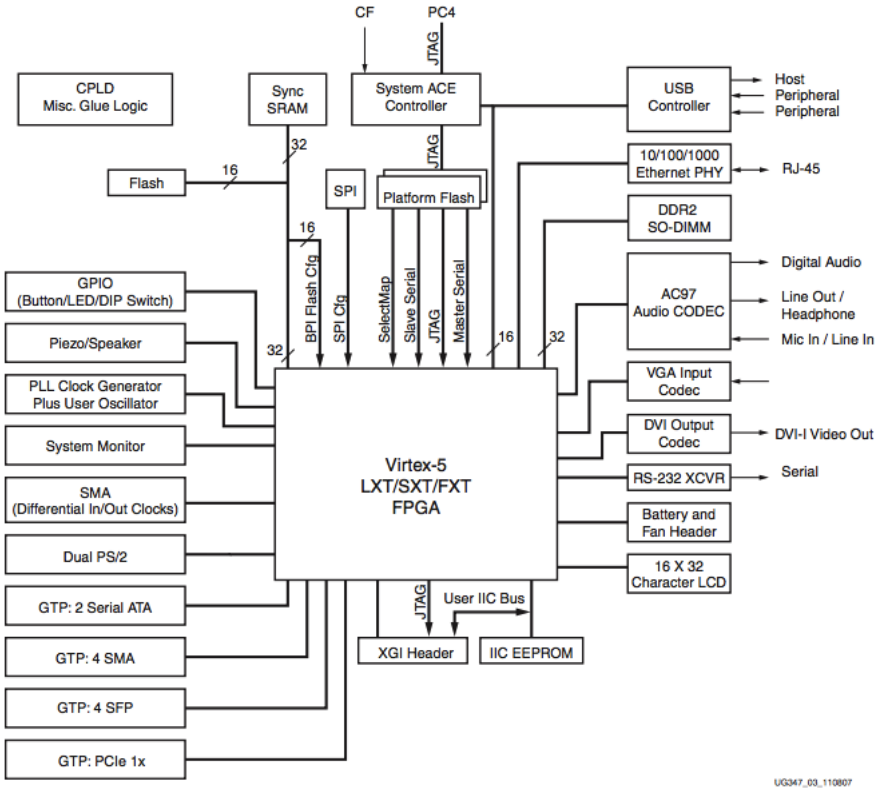


Figure 4.7. Picture of a Xilinx Virtex-5 LXT ML505 board [38].

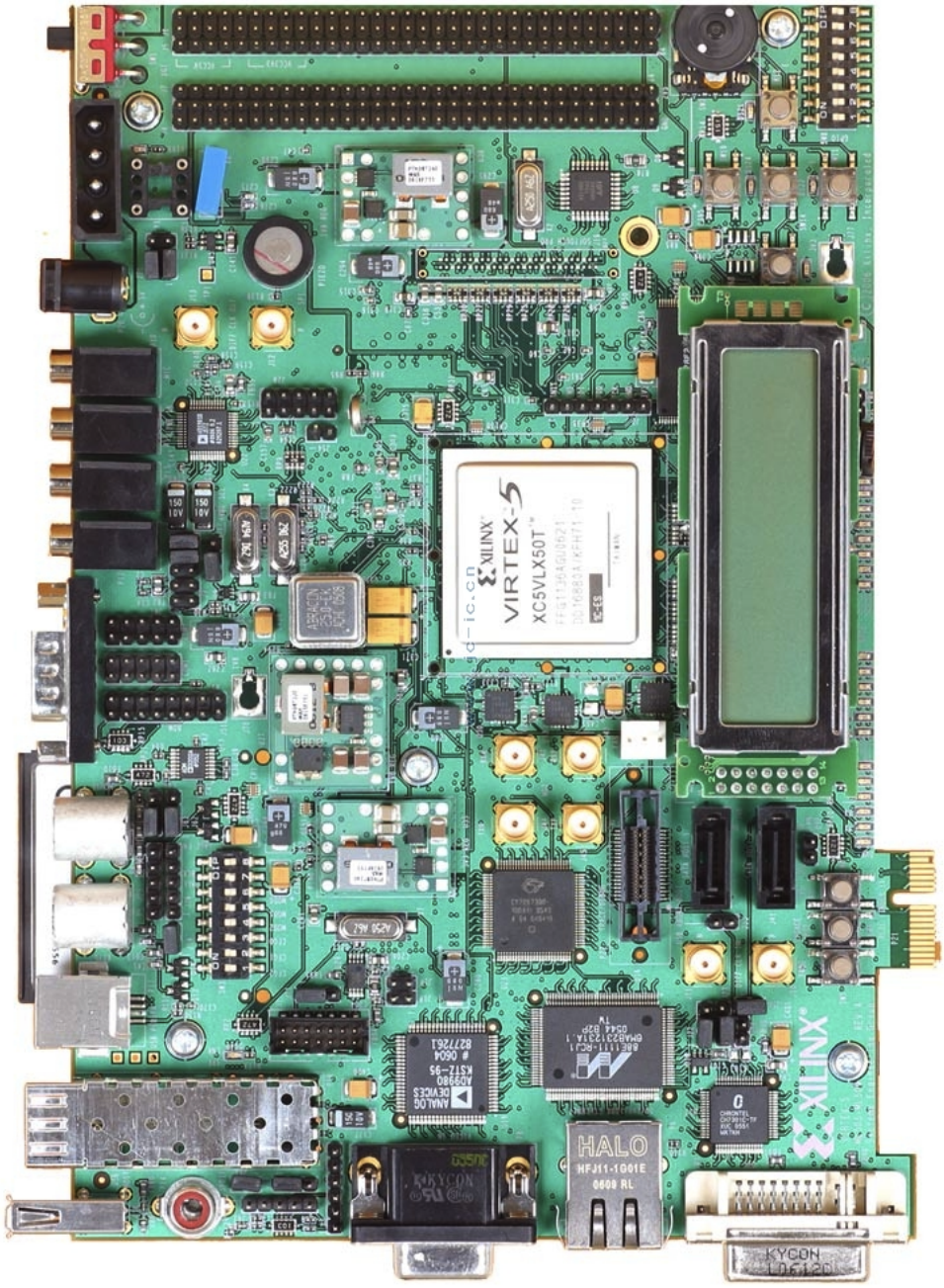


Figure 4.8. Picture of a Xilinx Virtex-5 LXT ML505 board [38].

4.2.4 The Readout Architecture

The ML505 board has been divided in multiple entities programmed with the *Xilinx* software suits ISE and EDK. The architecture implemented into the FPGA board is shown in figure 4.9.

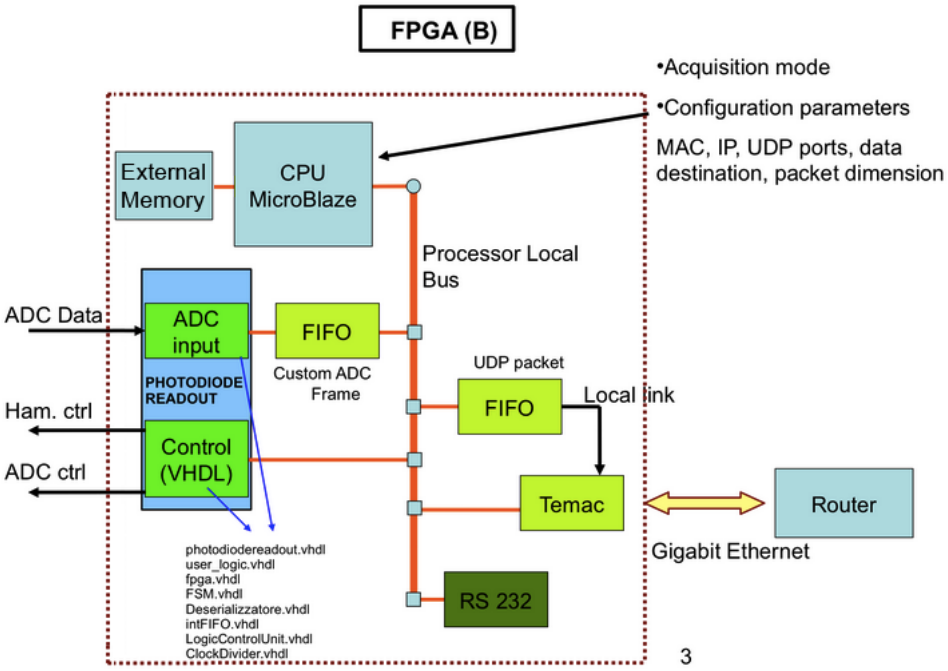


Figure 4.9. Outline of the entities contained in the FPGA [39].

In this scheme, the MicroBlaze CPU is a soft RISC processor core specifically designed for Xilinx FPGAs and implemented entirely in their general-purpose memory and logic fabric. Such block controls the entire board, decoding the UDP packets received from an interfaced computer and sending the decoded commands to the other entities. A $1\,GB$ external memory has been attached to this microprocessor to save and restore the FPGA initial configuration parameters and the software.

The TEMAC logic block, instead, interfaces the FPGA logic to the ethernet network and uses the same common bus, used to connect the CPU to the different ports (i.e. ethernet, usb, sata etc.). TEMAC is an acronym for Tri-Mode Ethernet Media Access Controller and is a reference to the three speed (10, 100, and 1000 Mb/s) capable Ethernet MAC function available in this core.

The FPGA also contains an RS-232 serial port, allowing the board to communicate with a terminal. A null modem cable is normally required to connect the board to the serial port on a computer. Such serial port is designed to operate up to $230400\,Bd$ and an interface chip is exploited in order to shift the voltage level

between FPGA and RS-232 signals.

The data packets are transmitted between the logic blocks through three FIFOs: one placed between the data bus and the TEMAC and two placed between the local bus and the readout logic unit; the motivation for this design structure will be given in the upcoming sections.

Finally, the logic control block manages the ADC and the photodiode physical devices through the generation of their control signals; the structure of this last logic block will be discussed in section 4.4.

4.2.5 The Readout and Control Protocol

The front end electronics has been connected to a computer through an ethernet network in order to control it and send out and analyze the collected data. In order to obtain such feature, a readout and control protocol was needed. Such system allow a parameters and commands exchange between the two interfaced devices. In particular, network and specific board parameters as well as reading and writing commands are needed to be transmitted through the network.

In order to establish the communication protocol, all the devices involved in the communication process need to have some parameters defined in order to communicate. Such parameters are the following:

- *The remote MAC address*: such MAC is needed by the readout node when is asked to send the measured datas to the interfaced computer.
- *The remote IP address*: it is the IP at which the data are sent by the readout node.
- *The local MAC address*: the first four bytes of this address are fixed to the hexadecimal values of $00 : 0a : 35 : 01$ while the last two can be set through the eight general-purpose active-high DIP switches present on the board.
- *The local IP address*: this IP is needed by the computer in order to know at which address to send the commands. Currently such address is manually associated to the FPGA MAC address through a terminal *arp* command. In the future, the board will be modified to avoid such manual process.
- *The communication port number*: it is the port through which the computer and the board communicate.

Moreover, having established the communication protocol, the board also needs to know other three parameters: the integration time, expressed in number of photodetector clock periods, and the frequency of the adc and photodetector clocks,

both expressed in number of divisions of the $250\,MHz$ FPGA clock. All these parameters, together with other monitoring information, can be accessed through an addressing scheme that manages a node and a board space:

- *Node Space:* composed by five registers that contain the information needed for the communication processes. In particular, there is a register for each of the five parameters previously described and one for storing the packets flags, fragment offset and time to live.
- *Board Space:* composed by four 32 but registers that can be externally read and written and additional four that are read only. The first four registers contain the readout parameters that the system uses during the data taking process while the last four contain different internal flags of the readout system, used for monitoring purposes

After configuring all these registers, an acquisition data run can be started (all the board internal registers can be read and written through the dedicated commands). All these communication processes occur thanks to the User Datagram Protocol (UDP).

The UDP packets, called frames, have been formatted in a predetermined form, understood by both the board microprocessor and the interfaced computer software. A general ethernet IP UDP frame is divided into three parts: the frame header, the frame payload and the frame footer. The frame header is subsequently divided in other seven parts as shown in figure 4.10:

- The preamble: composed by seven bytes of the value of 10101010. It is used to communicate to the receiving parts that the transmission has started and to synchronize the network clocks.
- The start frame delimiter (SFD): constituted by one byte of the value of 10101011 indicating the following transmission of important data.
- The Destination MAC address: formed by six bytes containing the destination LAN MAC address of the frame. If the address does not correspond to any machine in the network the frame is discarded.
- The source mac address: composed by six bytes containing the sender LAN MAC address.
- The payload length: constituted by two bytes indicating the length of the following payload.

- The payload: formed by a variable number of bytes from a minimum of 46 up to 1500 containing the actual data needed to be transmitted. If the message sent is bigger than the maximum 1500 bytes, the frame is divided in multiple smaller packets. Otherwise, if the message is smaller than 46 bytes a *padding* is added in order to reach the minimum length.
- The frame check sequence (FCS): composed by four bytes, it is used for transmission error checking.

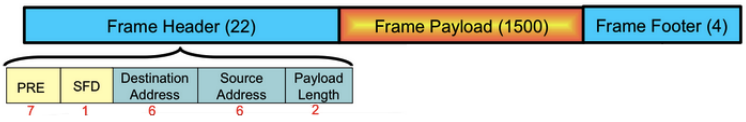


Figure 4.10. Subdivision of the ethernet UDP header. All the numbers represent the length of each frame subdivision expressed in bytes. [39].

Further on, the packet's payload part is divided in a substantial number of parts formatted in a standard way; such subdivision is shown in figure 4.11.

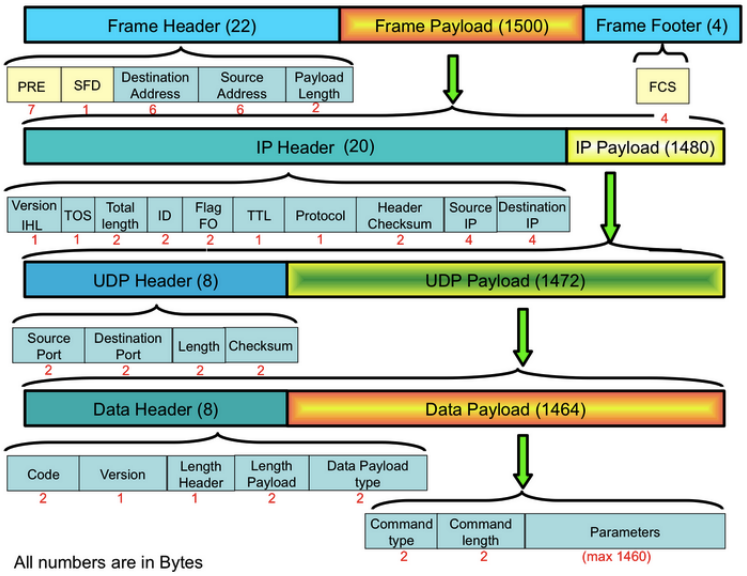


Figure 4.11. Subdivision of the ethernet UDP frame [39].

The only non standard formatted part of this specific subdivision is represented by the data payload type, contained in the data header, and the data payload, contained in the UDP payload. The data header type contains one of the words shown in figure 4.12; if a command word is inserted in this part, the data payload will be empty while, if some other parameters are inserted, the data payload will contain the specification of the written word.

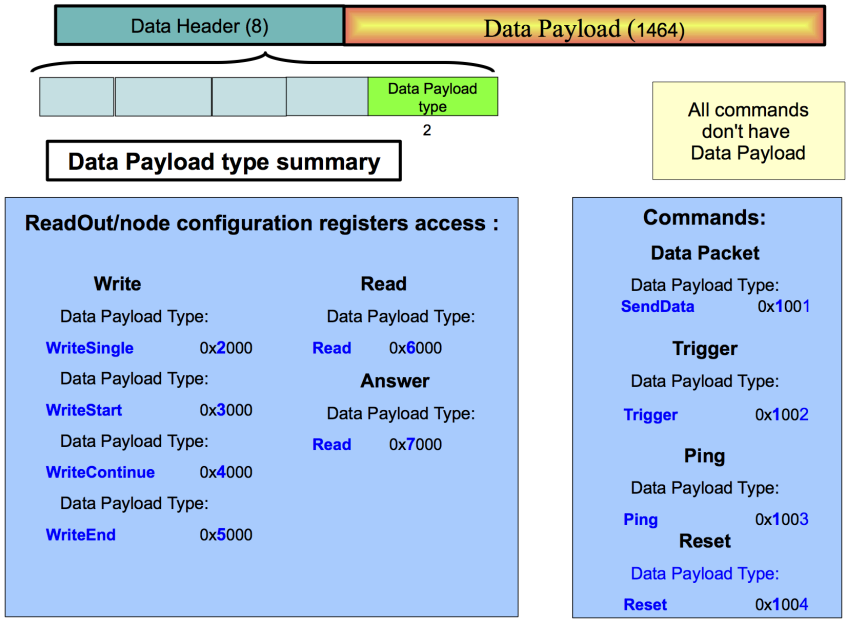


Figure 4.12. Format of the possible words contained in the data header type [39].

The data payload has been formatted in a similar way in order to make the MicroBlaze understand the messages sent and access the board internal registers. Finally the data packet structure has been formatted as shown if figure 4.13. The information contained in such part of the UDP packet can report the acquisition running ID, a footer representing the end of the packet or the measured data of a photodiode channel; these are the data that will be analyzed in order to understand the physical processes detected.

4.3 Simulation

The firmware of the FPGA has been written in VHDL. VHDL is an acronym for Very-high-speed integrated circuits Hardware Description Language, originally developed at the behest of the U.S Department of Defense. Such programming language is used in electronic design automation to describe digital and mixed-signal systems.

The first step taken in the coding process of the board logic is represented by the simulation of the physical devices, namely the photodiode array and the adc. During this stage, the digital emulation of the devices inputs processing has been coded. The entity that simulates the ADC behaviour is a stand alone logic block while the photodiode array simulation entity is composed of two logical blocks; one is exploited to produce a pseudorandom series of natural numbers representing the measured datas, while the other generates and controls the output signals. The

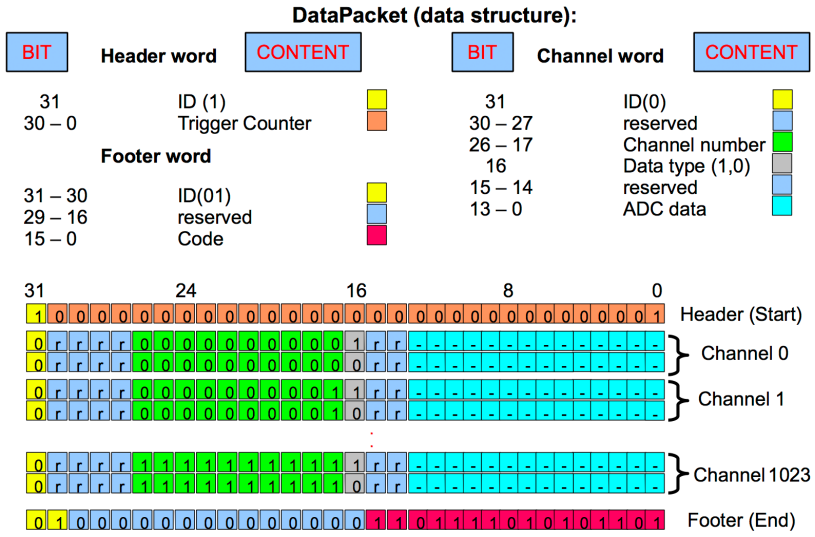


Figure 4.13. Format of the data packet structure containing the physical data that need to be analyzed [39].

structural outline of these two logic circuits is shown in figure 4.14 and 4.15.

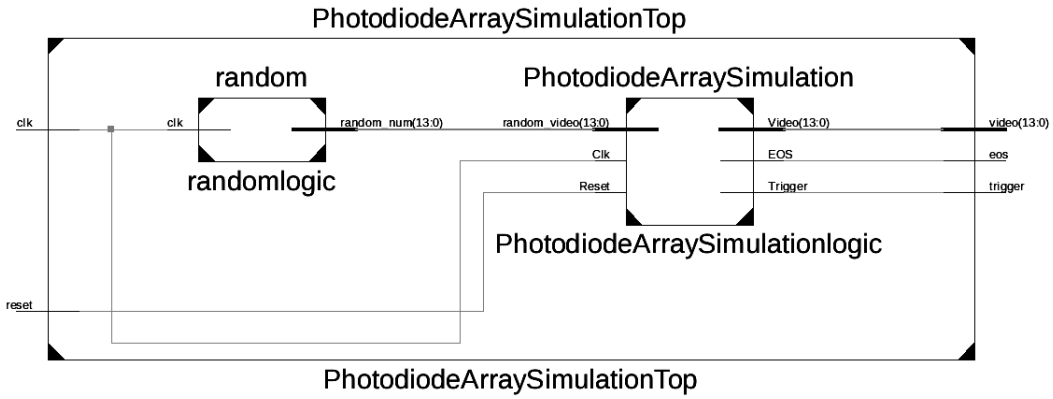


Figure 4.14. Structural outline of the entity coded for the photodiode array simulation.

4.3.1 Linear feedback shift register

The logic block written for the pseudorandom generation of a series of natural numbers has been coded following the linear feedback shift register technique; such technique is based on the lagged Fibonacci generators [40]. The algorithm has been chosen for its easy circuital implementation, being possible to realize it only with a shift register and some binary operators.

The lagged Fibonacci generator method takes his name from the Fibonacci series from which it has been derived; this series can be expressed through the recurrence

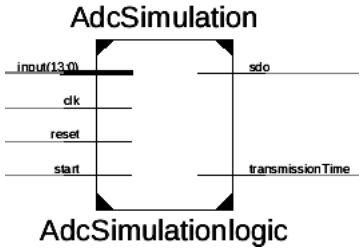


Figure 4.15. Structural outline of the entity coded for the adc simulation.

relation:

$$S_n = S_{n-1} + S_{n-2} \tag{4.2}$$

from which the generalization of the generators:

$$S_n = S_{n-j} * S_{n-k} \ (mod\ m), 0 < j < k < n \tag{4.3}$$

where m is usually a multiple of two (i.e. $m = 2^M$), the variable M,n,j e k are natural numbers and $*$ is a binary operator; like all pseudorandom number generators the series obtained is periodic, with a period dependent on the initial seed and the binary operator chosen. The linear feedback shift register are based on such generators; they are constituted by a shifter register, having in input a function of its internal flip-flops values. In particular the coded linear feedback shift register has been realized with a shift register described by the following equation:

$$\begin{aligned} B_0 &= B_{13} \oplus B_{12},\ t > t_0 \\ B_{13} &= 1,\ B_i = 0,\ \forall i = 0, 1, 2...12,\ t = t_0 \end{aligned} \tag{4.4}$$

where B_i is the i-th bit of the shift register and $\oplus$ the exclusive or binary operator. The minimal shift register length required and the bit on which the operator acts have been chosen based on the 14-bit adc resolution, keeping the circuit as simple as possible. A graphical representation of this entity is shown in figure 4.16 while its code has been reported in appendix A.1.

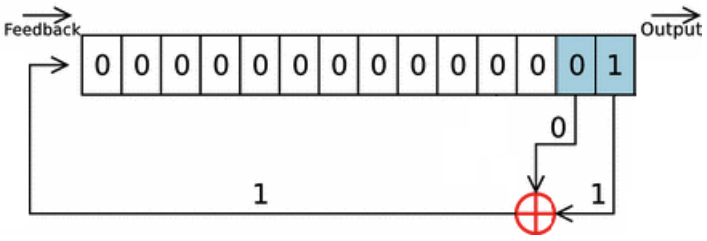


Figure 4.16. Graphical representation of the linear feedback shift register used at $t = t_0$.

4.3.2 The Photodiode Array Simulation

As previously stated in section 4.2.1, the photodetector used in this system is the S8865 photodiode array, commercialized by the *Hamamatsu Photonics* company. In order to simulate this device an entity sensible to the input clock rising edge has been coded. Such logic block counts the number of rising edges, subsequently to having received a starting trigger signal sent by the logic control unit. Having implemented this part, the code for the output generation has been written. The entity emulates the output of the physical circuit using the pseudorandom number generator logic block previously described. The timing diagram of this device and its simulation are shown in figure 4.17, while the code has been reported in the appendix A.1.

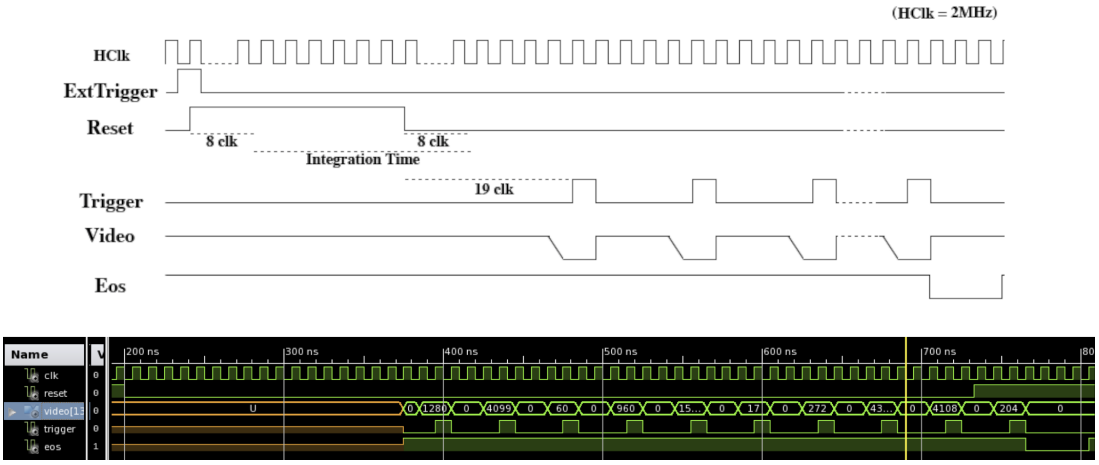


Figure 4.17. Timing diagram of the photodiode array and its related simulation.

4.3.3 The ADC Simulation

The *Texas Instruments* ADS7945 adc circuit has been simulated through the code written in the appendix A.2. In this logical block the FPGA has been programmed to set the output pin *sdo* equals to the $(13 - n)$-th bit at every input clock rising edge, where n is the number of edges counted. The *transmissionTime* output that is shown in the block diagram in figure 4.15, has been implemented in order to see the effective conversion time of the adc and to monitor more precisely the device. The timing diagram and the simulation of this logical block are shown in figure 4.18.

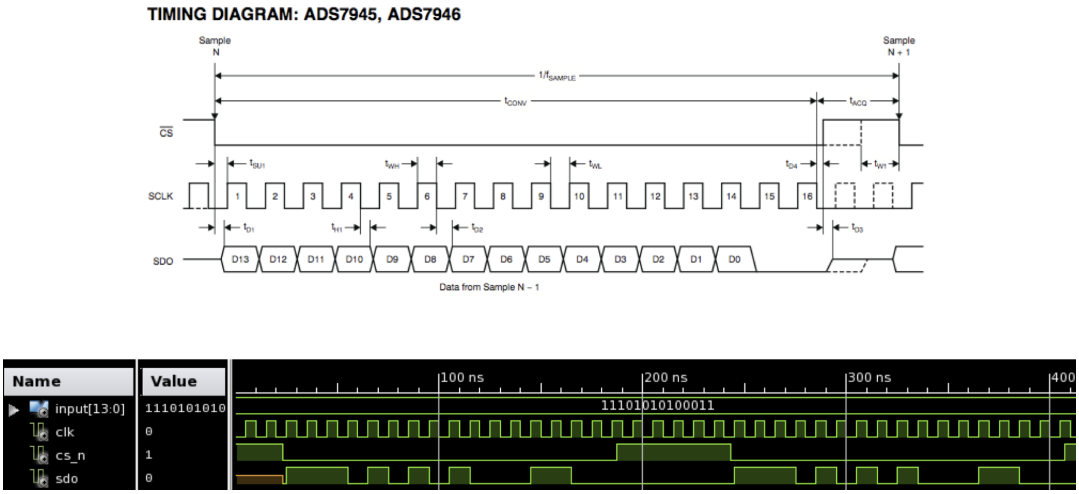


Figure 4.18. Timing diagram of the adc and its related simulation.

4.4 The FPGA Logic

The FPGA logic has the goal of deserializing the adc output, formatting the data into a pre-established form, sending it to the external fifo and monitoring all the photodiode and adc control signals. This logic block is subdivided into five entities: the deserializer, the clock divider,the internal fifo, the finite state machine and the logic control unit. The latter entity is additionally formed by other two logic blocks: the clock and the trigger analyzers. The general outline of the FPGA logic is shown in figure 4.19 and in the following sections each logic subdivision will be covered in detail.

4.4.1 The Deserializer

As suggested by its name, the deserializer entity aims at converting the serial adc output into a 16-bits word that is sent to the logic control unit. The schematic block of such entity is shown in figure 4.20. The deserializer is synchronous with respect to the adc, being both devices controlled by the same clock. When an adc clock rising edge is detected by the deserializer, if the *datain* port equals 1, the internal *outputInteger* signal is set equals to:

$$outputInteger = outputInteger + 2^{width-3-n} \tag{4.5}$$

where the variable *width* equals 16 representing the word length. As previously seen in the section 4.2.2, the adc produces a 14-bits serial data stream; therefore, when the deserializer has counted 14 clock rising edges, the *dataout* port is set to the value

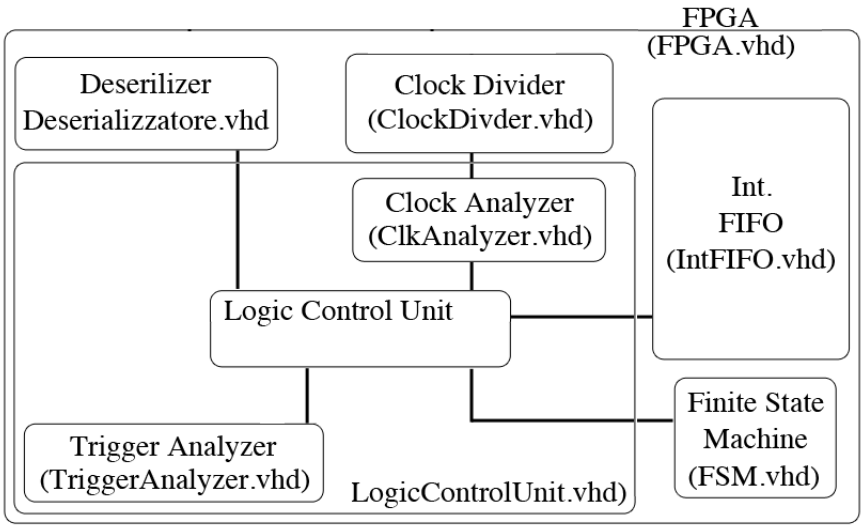


Figure 4.19. Schematic circuit diagram of the entity coded for the adc simulation working at $250\,MHz$.

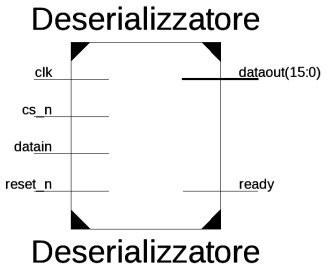


Figure 4.20. Schematic circuit diagram of the deserializer entity.

of "00" + *outputInteger* and rises the *ready* flag in order to communicate to the logic control unit that the word is ready and can be formatted. The timing diagram simulation of this device is shown in figure 4.21.

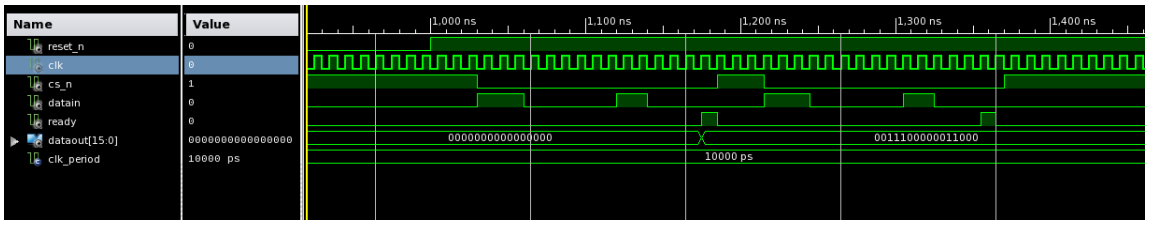


Figure 4.21. Timing simulation of the deserializer entity.

4.4.2 The Clock Divider

The clock divider entity has been created in order to generate the adc and photodiode clocks with the same phase of the fpga clock. The constraints of these two clocks are given by the hardware specification. In particular, the adc clock frequency must be lower than $40\,Mhz$ while the photodiode one must be lower than $2\,Mhz$. The only other constraint given by the structure of the readout system is:

$$T_{photo} > 32T_{adc} + 80ns \tag{4.6}$$

where the T_{photo} is the photodiode clock period and T_{adc} the adc's. Giving the fact that the analogic signal can be sampled only during the photodiode trigger signal, the previously stated constraint is generated by the adc need of having at least eighty ns for the signal acquisition and sixteen clock periods to sample the photodiode video output. The considered logic block is sensitive to the fpga rising edge and it takes as an input a *divisor* integer, made up of *lengthDivisor* bits, which is an entity's parameter. The schematic circuit of this block is shown in figure 4.22. Starting

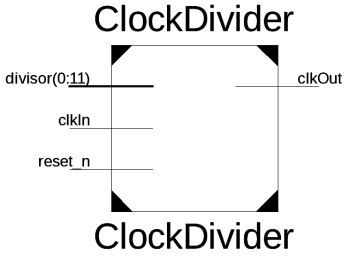


Figure 4.22. Schematic circuit diagram of the clock divider entity.

from one and counting up to $divisor/2$ the first time and to $divisor - divisor/2$ the second time, the *clkOut* output port is set equals to:

$$clkOut = NOT\,(clkOut). \tag{4.7}$$

Such differentiation between the two clock phases has been made in order to allow the entity to take odd number as divisor integers. Finally, the timing diagram simulation of the clock divider is shown in figure 4.23.

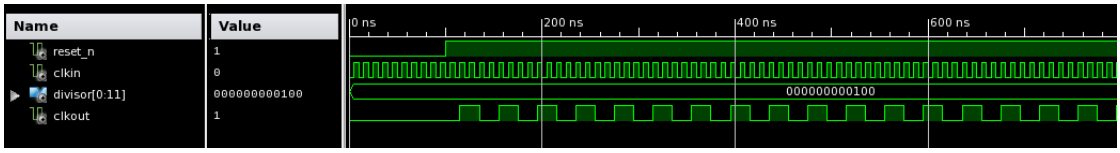


Figure 4.23. Timing simulation of the clock divider entity.

4.4.3 The Internal Fifo

A system of two FIFOs, one internal and one external with respect to the FPGA logic entity, has been chosen due to the limited storage capacity of the external one and due to the incapacity of the internal FIFO of interacting with the FPGA's BUS. The latter logic block has been realized using the ISE IP generation wizard instead of manually coding it in order to speed up the development process.

The external FIFO, embedded in the FPGA, has a length of 512 words of 32-bits each and is driven by the $125\,Mhz$ BUS clock. The internal FIFO, instead, has a length of 2048 words of 32-bits each with two different writing and reading clocks; the former is driven by the $250\,Mhz$ FPGA clock while the latter by the $125\,Mhz$ BUS clock. This FIFO has been also programmed with an almost full flag activated when it's filled more than two-thirds of its whole length. The schematic circuit of this block is shown in figure 4.24.

4.4.4 The Finite State Machine

Once the structure of two FIFOs has been chosen, a machine capable of handling this system has been needed. Such entity is a simple machine that checks if the external FIFO is full and the internal FIFO empty. If one of these two conditions are met during the FPGA clock rising edge the *rd_en* output port is set to zero, in all the other cases to one. The VHDL code written for this logic block is reported in appendix A.3 while the schematic block in figure 4.25.

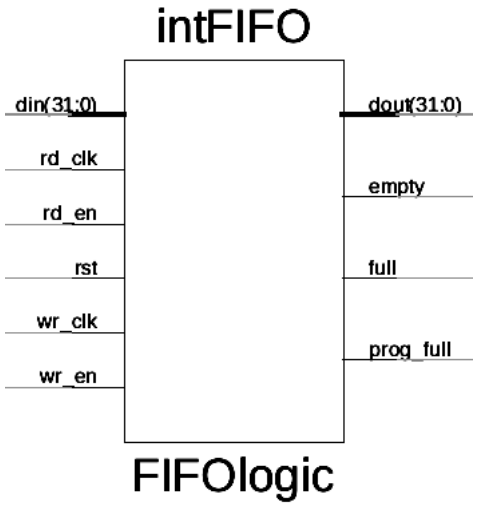


Figure 4.24. Schematic circuit diagram of the internal FIFO entity generated through the ISE wizard.

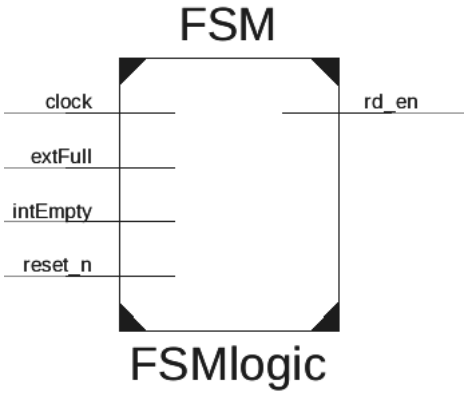


Figure 4.25. Schematic circuit diagram of the finite state machine logic block.

4.4.5 The Logic Control Unit

The logic control unit entity is the most important and advanced entity of the FPGA logic part. The aim of such logic block is the formatting of the data, their transmission through the network and the handling of the control signals of the whole detector front end electronics. The logic block additionnally includes two entities used to sample the adc clock, the photodiode clock, the external trigger and the photodiode trigger. The clock analyzer logic entity sends out a short pulse, lasting one $250\,MHz$ clock period, on the *rising* port when the input clock rising edge is detected; similarly the trigger analyzer entity sends out a spike when the input clock rising or falling edges are detected respectively on the *rising* and *falling* output ports. Their schematic and time simulation diagrams are shown respectively in figure 4.26, 4.27, 4.28 and 4.29.

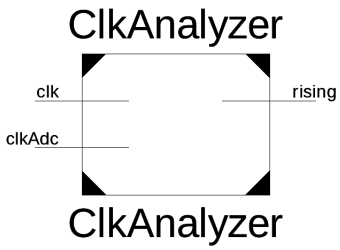


Figure 4.26. Schematic circuit diagram of the clock analyzer logic block.

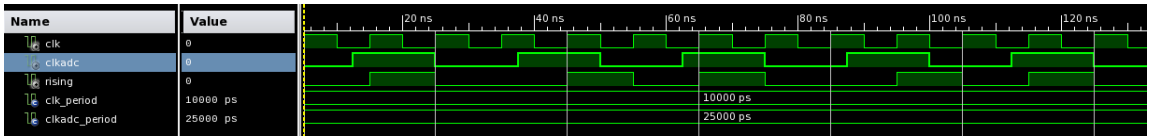


Figure 4.27. Simulation time diagram of the clock analyzer entity.

Having sampled the four stimuli, the logic control unit can control the whole system. When it's turned on by setting its *rst_n* port to the higher logical level, the entity waits for an external trigger. Once this external trigger has arrived, the control logic sends the trigger counting word to the internal FIFO and activates the *photodiodeStart* output, setting it to one, for a period lasting:

$$T_{int} = n \cdot T_{photo} \tag{4.8}$$

where the T_{int} is the integration time (i.e. the time during which the *photodiodeStart* needs to stay equals to one), T_{photo} the period of the photodiode clock and n a value

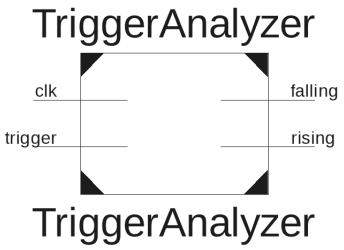


Figure 4.28. Schematic circuit diagram of the trigger analyzer logic block.

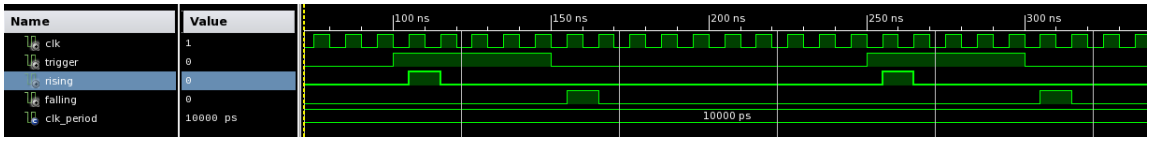


Figure 4.29. Simulation time diagram of the trigger analyzer entity.

passed to the entity by the *integTime* port. Having given the start signal to the photodiode, a trigger input is then awaited in order to start the adc conversion, setting the *adcStart* to zero. The adc start signal needs to stay in the zero state for sixteen adc rising edges counted by the logic control unit through the previously described clock analyzer block. The same process is repeated when the trigger falls, in order to measurement the pedestal of the apparatus. As previously described, when the adc edges counter equals fourteen, a flag is set to one on the *dataReadyFromDeserializer* port by the deserializer entity, signalling that the word on the *datain* port is ready to be sent to the internal FIFO. The whole control process is repeated until the *Ham_Eos* input equals zero, meaning that the photodiode has sent all its channel datas and that the logic control unit can wait for another external trigger. The VHDL code written for this logic block is reported in appendix A.3 while the schematic block in figure 4.30 and its time simulation in figure 4.31.

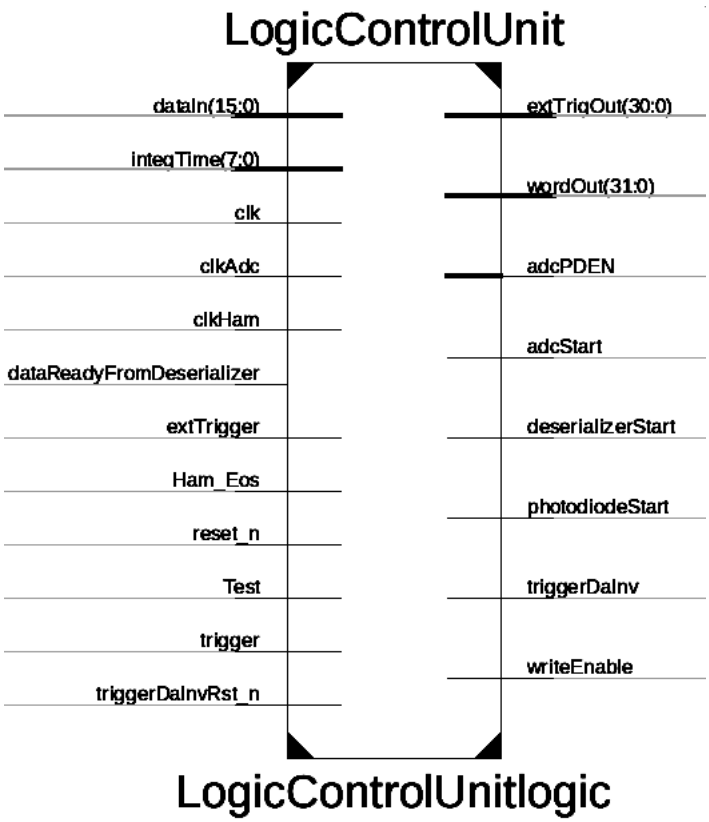


Figure 4.30. Schematic circuit diagram of the logic control unit entity.

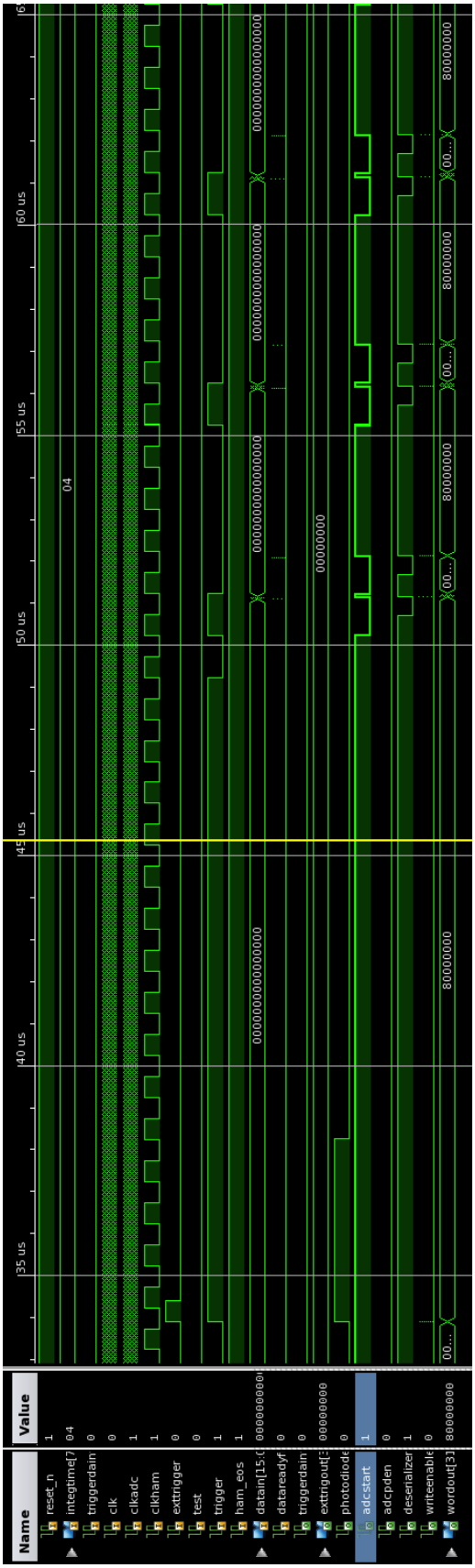


Figure 4.31. Time simulation diagram of the logic control unit entity.

Chapter 5

Laboratory Tests

In this chapter the main tests made on the developed device will be covered. Specifically, a general overview of the test setup will be given in the first section, followed by some physics calculation on the expected measurements and finally showing the debugging procedures adopted.

5.1 Laboratory Test Setup

5.1.1 Equipment

The laboratory setup, where most of the tests were made, was composed by:

- A computer running Windows XP. This machine has been used to load the firmware and the software inside the FPGA. Such operating system was needed in order to run the Xilinx proprietary software.
- A laptop running Unix. This machine has been used in order to run the custom developed Graphical User Interface and to acquire and elaborate the datas produced by the board.
- The acquisition Board containing a custom interface board with a $2\,MHz$ sampling rate, 14 SAR ADC that samples the photodiode video signal.
- A pulsed led with a variable pulse frequency. This device has been used to illuminate the photodiode array and to perform some preliminary measurements.
- A PVC box containing the photodiode array and the led. A picture of such box is shown in figure 5.1.
- An oscilloscope used to monitor the board input and output signals.
- A router in order to link all the devices through a Gigabit ethernet network.

A picture of the laboratory setup is shown in figure 5.2.

Figure 5.1. Box containing the photodiode array and the pulsed led.

5.1.2 The Graphical User Interface

In order to interact with the board and monitor its activity, a graphical user interface has also been developed. The GUI is a java application capable of sending and receiving UDP or TCP messages through an ethernet network; the main window of such application is displayed in figure 5.3. The interfacing application had a fundamental role during the testing and debugging board processes, allowing writing and reading operations on the board internal registers.

5.2 Expected Performances

In order to properly debug the readout system, a theoretical calculation of the expected measurements was needed. Taking the specifics of the EPLED360 pulsed UV-LED produced by the *Laser2000* company, the number of photons detected by the photodiode array was estimated. The EPLED360 is a pulsed led capable of generating calibrated light pulses of 700 ps width at programmable frequencies less

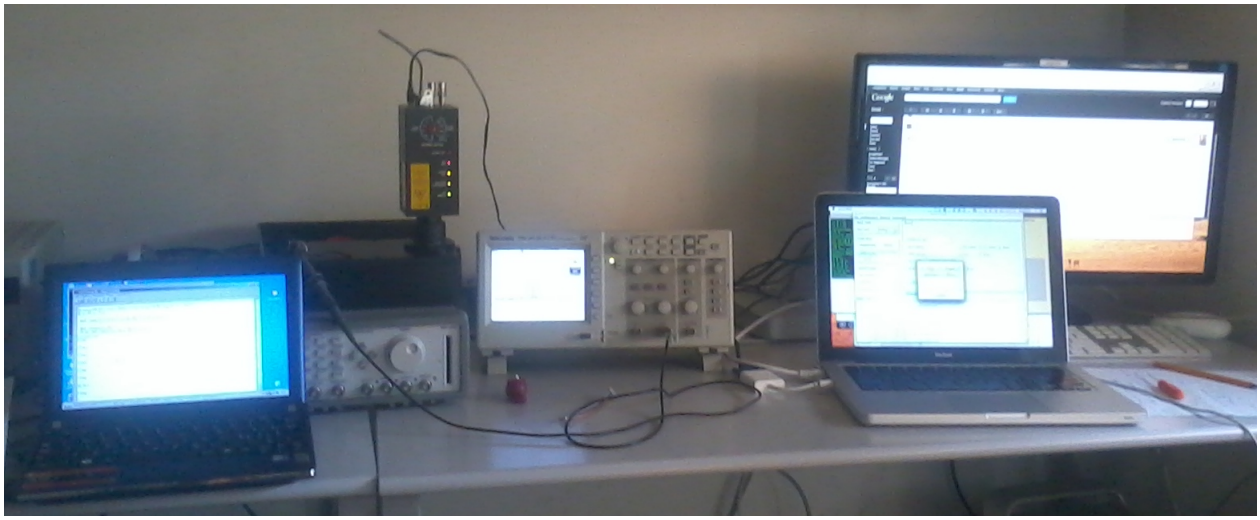


Figure 5.2. Picture of the laboratory setup where most of the tests were made.

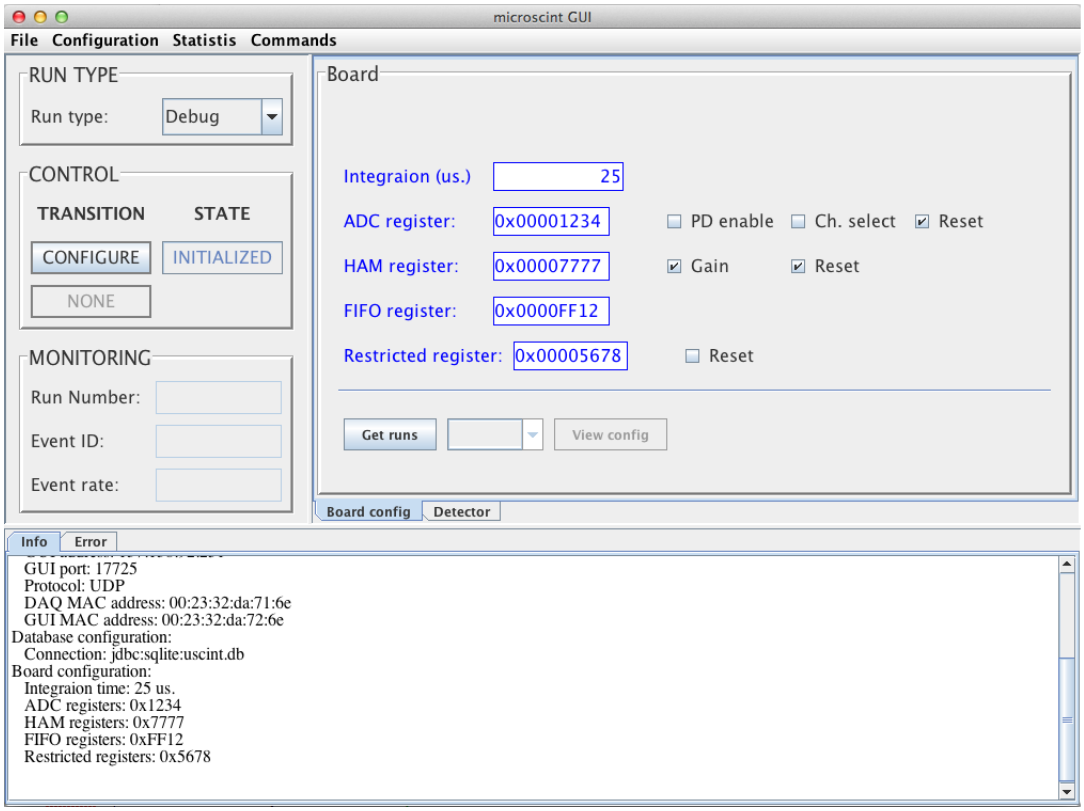


Figure 5.3. Main window of the GUI application.

than $10\,MHz$. Each pulse delivers $0.8\,\mu W$ of power on a spot of $123\,mm^2$ of size. The number of photons reaching the photodiode array can be expressed as:

$$n_\gamma^{ph} = T_{int} \cdot f_{led} \cdot n_\gamma^{pul} \cdot (\frac{A_{PA}}{A_{LB}}) \tag{5.1}$$

where n_γ^{ph} is the total number of photons detected during every photodiode measurementment, T_{int} is the integration time of the photodetector, f_{led} is the frequency of the LED pulses, n_γ^{pul} is the number of photons in a single pulse and A_{PA}/A_{LB} is the ratio between the area of the photodiodes irradiated and the area covered by the LED beam. Such equation can be further expanded knowing that

$$n_\gamma^{pul} = \frac{P_{led} \cdot T_{pul}}{h\nu_\gamma} \tag{5.2}$$

where P_{led} is the power generated by the EPLED per pulse, T_{pul} the average time interval of each pulse, h the Planck constant and $\nu_\gamma = 833\,THz$ the frequency of photons generated. The number of photons per pulse yielded is $n_\gamma^{pul} \simeq 1014$. Knowing most of these values from the EPLED and photodiode data sheets [41] [36] a $n_\gamma^{ph} \simeq 1.8 \cdot 10^5$ is yielded, using $T_{int} = 1\,ms$, $f_{led} = 10\,MHz$ and $A_{PA} = 2.24\,mm^2$. Finally, having derived the conversion factor between the number of detected photons and the signal generated by the amplified photodetector, an estimate of the expected measured datas can be derived. Such conversion factor has been calculated by measuring the output voltage of the integrated light and dividing it for the number of photons detected. The analytical relation between these two values can be expressed as:

$$V_{data} = K n_\gamma^{ph} = K T_{int} \cdot f_{led} \cdot \frac{P_{led} \cdot T_{pul}}{h\nu_\gamma} \cdot (\frac{A_{PA}}{A_{LB}}) \tag{5.3}$$

where V_{data} is the photodetector output signal, n_γ^{ph} the total number of photons detected and K the conversion factor of about $50\,\mu V$ extrapolated from the measurements.

5.3 Debugging Procedures

The first test done in order to debug the readout system was the development of a testing mode for the fpga logic. In this mode, enabled through a bit of a board register, the logic sends only the header and the footer of each trigger, avoiding the transmission of the measured data. Thanks to such feature, the two FIFO memories have been tested without the rest of the readout chain, allowing a first interface between the board and the GUI. A time simulation representing a test mode run can be seen in figure 5.4.

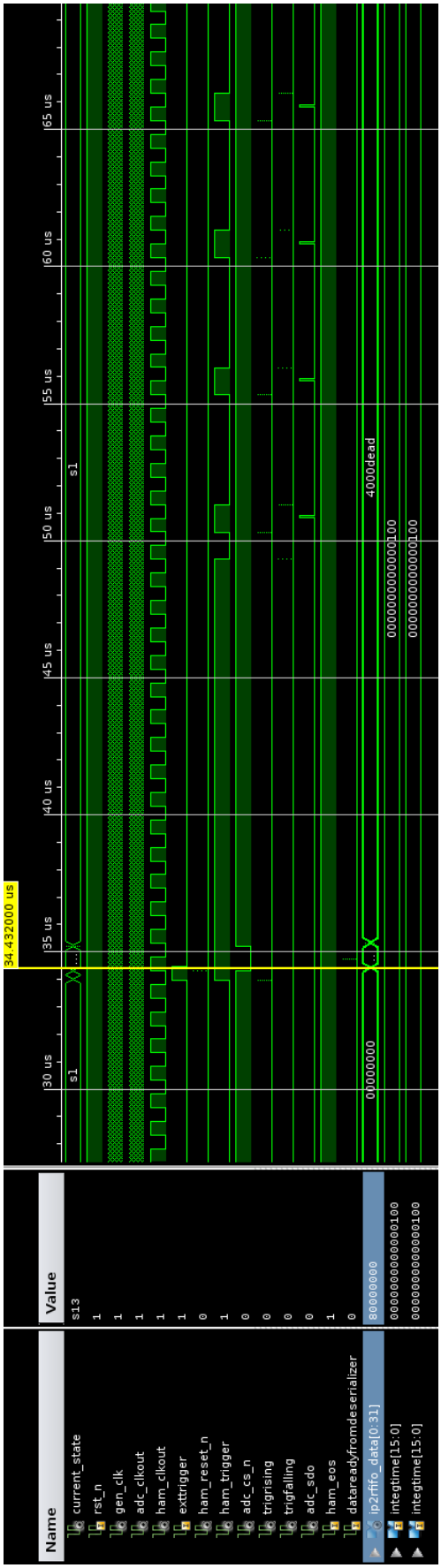


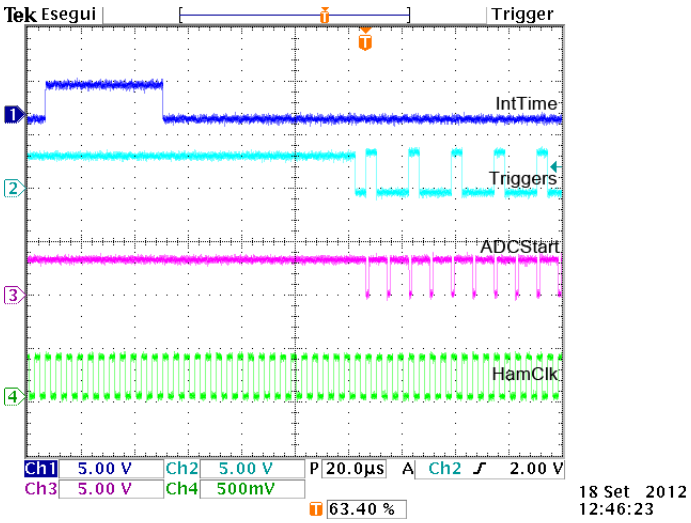
Figure 5.4. Time test simulation diagram of the whole logic entity.

Having received on the computer the first packets containing the headers and footers, the board output signals have been checked through an oscilloscope; an image of these signals can be seen in figure 5.5.

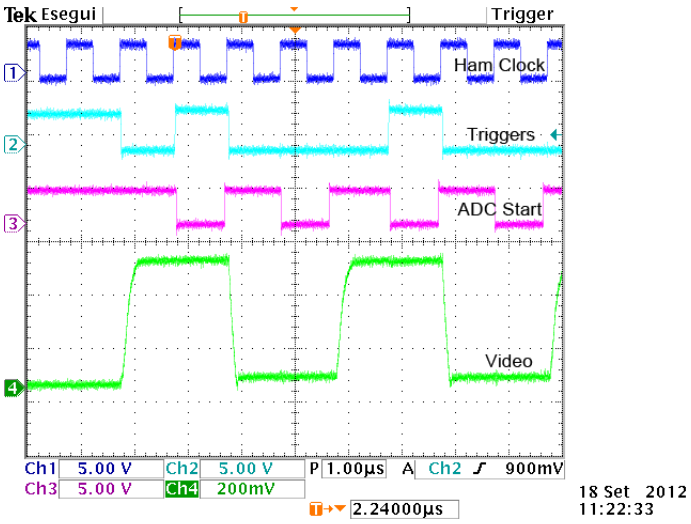
In particular, this figure 5.5 pictures the timing between the ADC and photodetector signals, an ADC conversion period and the beginning of a data acquisition run. In order to acquire such image, a time integration of $48\,\mu s$, a photodetector frequency of $250\,kHz$, and an ADC frequency of $35\,MHz$ have been used. From the picture the signals simulated during the process described in part 4.4.5 can be seen.

Finally, the EPLED has been mounted onto the box where the photodetector were placed, the readout board was connected to the photodiodes and the first data acquisition test has been run. The result of such procedure can be seen in figure 5.6 where the output signal of the photodiode array illuminated by the LED and the serial data output of the used ADC are represented. In this measurement a 3 mm hole is used to limit the light emitted by the led, illuminating approximately 4 pixels in accordance with a pitch size of 0.8 mm.

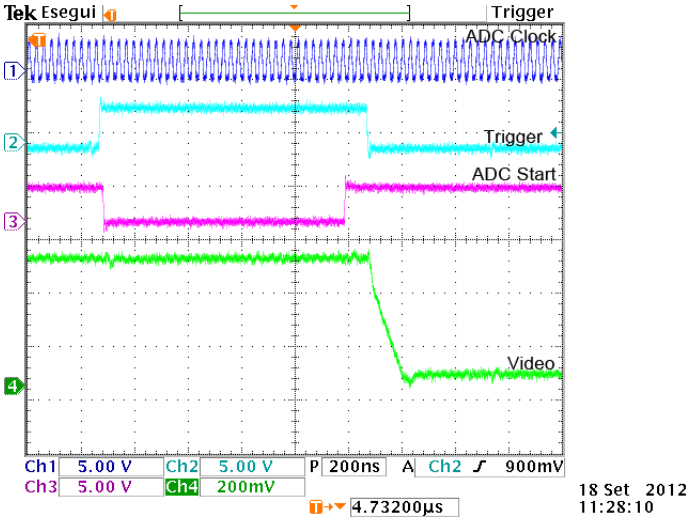
Thanks to these procedures a considerable amount of problems were solved. As a first step, problems regarding the logic and the synchronisation of the two FIFOs have been solved modifying the FPGA firmware. Moreover, different circuit assembly errors were detected, measuring the output signals through the oscilloscope. Such errors were solved analyzing the device pins' solderings and their routing through a microscope. Finally, measuring subsequents data taking runs, multiple errors have been solved on both the software and firmware behaviour, time synchronisation and the FPGA interface with the computer. Having tested the correct functioning of the whole readout system, a characterization of such device has been undertaken; such process is described in the following chapter.



(a)

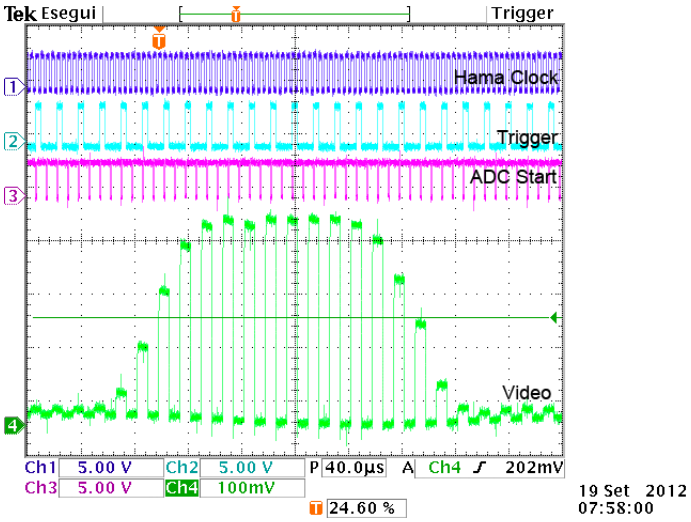


(b)

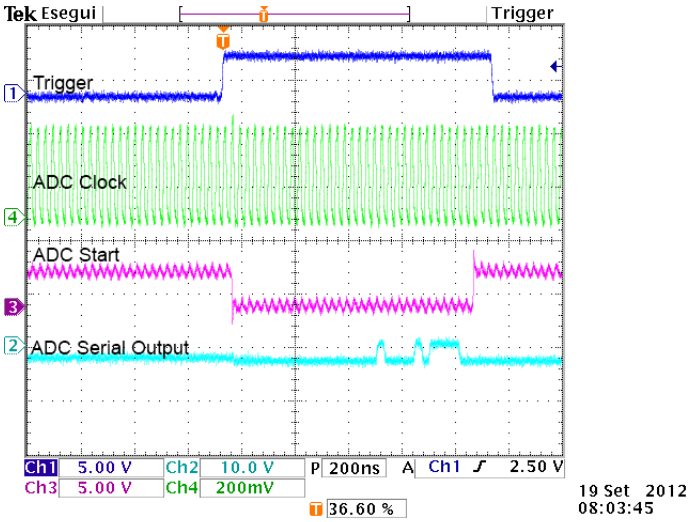


(c)

Figure 5.5. Readout signals measured through an oscilloscope. In fig (a) the beginning of a data acquisition run is shown. In fig (b) the timing of the main signals of the photodiode array and the ADC have been measured. Finally in fig (c) a single ADC conversion period is shown.



(a)



(b)

Figure 5.6. Readout signals measured through an oscilloscope. A 3 mm hole is used to limit the light emitted by the led, reaching then the photodiode surface. About 4 pixels are illuminated in accordance with a pitch size of 0.8 mm. In fig (a) the output signal of the photodiode array illuminated by a pulsed LED is shown. In fig (b) the serial data output of the used ADC has been measured. 32 clock cycles are necessary to convert and send the serialized digital data

Chapter 6

Measurements

In this chapter the first results from the developed readout system, irradiated with different light sources, will be shown. In particular, the geometry of the acquisition process, the linearity tests made on the device and finally the result obtained will be described in the following sections.

6.1 Geometry

As shown in section 5.1, the photodiode array has been placed in a PVC box in order to irradiate its pixels with a pulsed LED. In particular, one side and the top of such box have been pierced to form a circular hole of $3\,mm$ of diameter, where different light generators have been mounted through dedicated adapters, as shown in figure 6.1.

Figure 6.1. Picture of the hole made into the top side of the box and its dedicated adapters.

The top hole has been created in order to directly irradiate the photodetector, while the one on the side has been designed to directly illuminate the scintillating channels described in section 2.4. Specifically, the distance between the top hole and the photodetector plane is approximately $4\,cm$, while the distance between the side hole and the scintillating channels plane is about $2.5\,cm$.

6.2 System Characterization

In order to further characterize the detector readout system, its linearity has been tested by varying the integration time and the frequency of the photodetector; the same pulsed LED, as described in section 5.1, has been used in order to make such tests.

6.2.1 Data Errors

A typical photodiode array output can be seen in figure 6.2 where the photodetector has a clock frequency of about $2\,MHz$ and an integration time of approximately $1\,ms$. From such image a peak with a width of approximately 4 pixels, as calculated in section 5.2, can be seen. In order to get such graph, multiple data operations have been needed. As previously stated, the data have been measured varying the integration time and the frequency of the photodetector; for each combination of these two parameters, the measurements have been taken 2000 times.

During the first measurements taken a fluctuation of the channels values has been noted. Due to the presence of this phenomenon, a high measuring error was being introduced. Having therefore considerably high errors on the measurements acquired, an error reducing algorithm has been studied.

A first step studied to obtain a statistical valid result is represented by the operation of plotting a histogram of the values assumed by each channel during the total number of acquisition runs (figure 6.3(a)). From such histogram a distribution similar in each channel was found and the noise was noted to be correlated at low frequencies. Because of that, a gaussian fit has been made on the histogram, obtaining a mean value for each channel. Thanks to this operation, a graph of the channel mean values versus the number of such channels could be derived as shown in figure 6.3(c).

At this point, a pedestal value still needed to be subtracted from the obtained values. In order to achieve such goal, for each trigger a histogram of the channels values was analyzed. During such analysis the histogram has been fitted with a gaussian curve centred in the mode point and the mean value found with such fit has been subtracted to each channel mean value, as can be seen in figure 6.3(d).

The result obtained from such operation is shown in figure 6.3(e).

Finally, in order to select the pixels illuminated by the LED light, the first and last thirty channels of the array, that were noted to have the most amount of noise, have been fitted with a constant (figure 6.3(f)): subtracting the calculated constant and excluding the results that where less than three times the standard deviation of such fit, the final graphs 6.2 has been produced for a photodiode array with 128 channels.

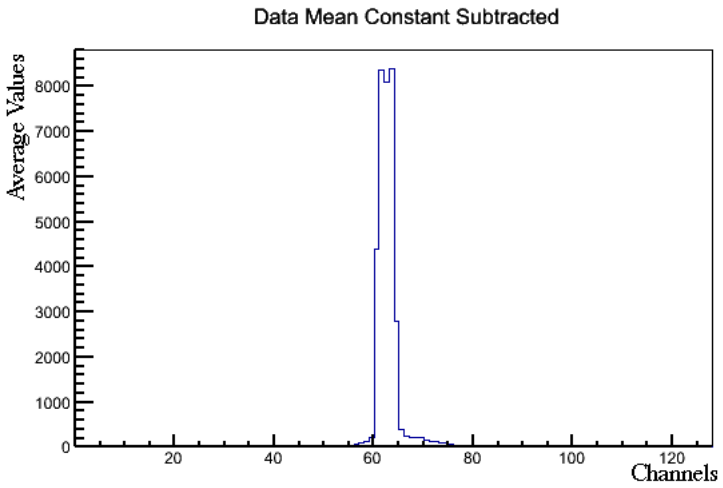


Figure 6.2. Final result of the studied error reducing algorithm for a set of measurements acquired with a photodetector frequency of about $2\,MHz$ and an integration time of approximately $1\,ms$.

6.2.2 Linearity results

Having obtained the data with the process described in the previous part, a linearity fit for each photodiode frequency has been made, varying the integration time and excluding from the fitted points the ones that saturated the photodetector: such exclusion has been made by setting a threshold, calculated through the saturated channels. Graphs of such linearity curves are shown in figure 6.4.

From the fitted graphs, a value representing the output volt per photon of approximately $50\,\mu V/photon$ can be derived. Moreover, from the same graphs, a curve representing the difference between the values extrapolated from the fitted points and the actual values measured can also be drawn. The result of such an operation can be seen in figure 6.5.

As it can be seen from the shown graphs, the photodetector is linear exception made for its saturation points. From this fact it can be stated that for integration times between $1\,\mu s$ and $1\,ms$ a photodiode array frequency of about $2\,MHz$ can be

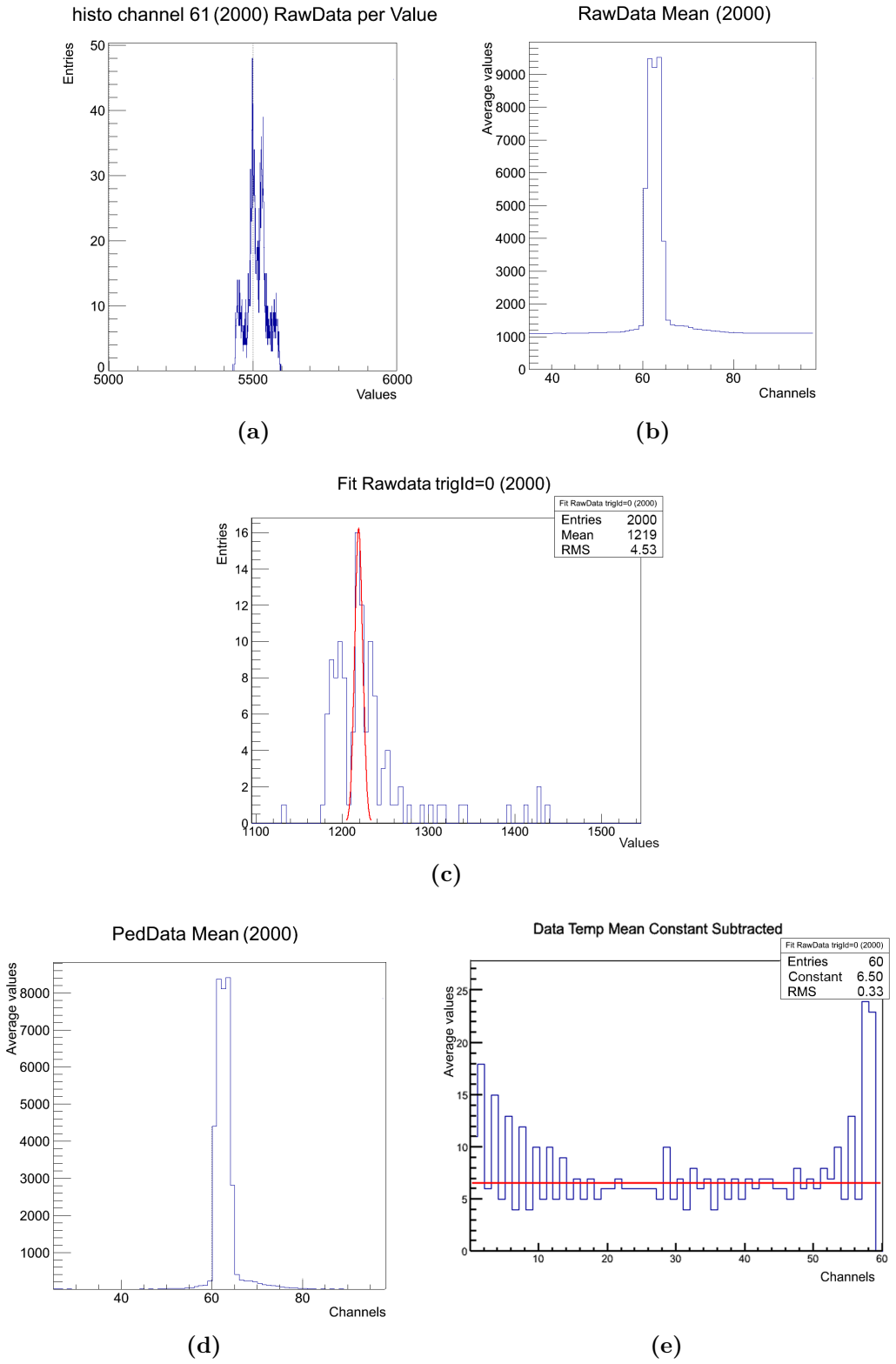
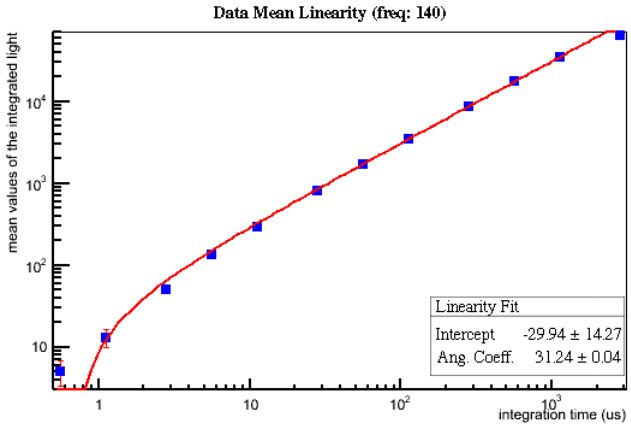
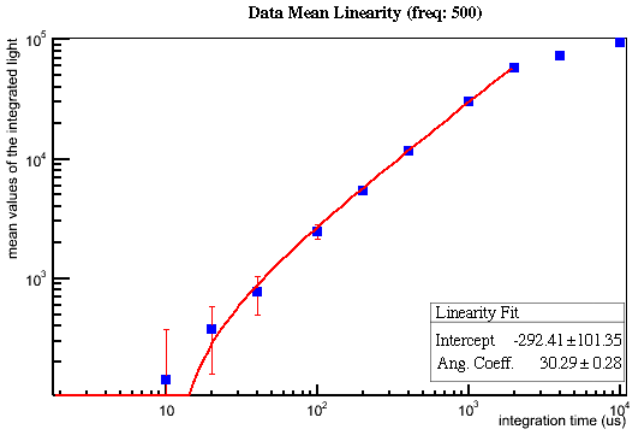


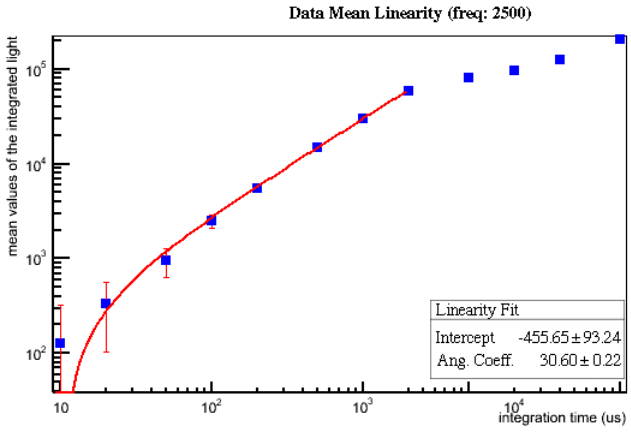
Figure 6.3. Operation performed on the data measured with a photodetector frequency of about $2\,MHz$ and an integration time of approximately $1\,ms$. In fig (a) a histogram of the values assumed by channel 61 during the 2000 acquisition runs is shown while in fig (b) a graph of the channels mean value versus the number of such channels is pictured. Moreover in fig (c) an histogram of the channels values fitted with a gaussian curve can be seen. Finally in fig (d) the result of the subtraction of the fitted pedestal is drawn and in fig (e) the constant fit of the first and last thirty channels values is shown.



(a)

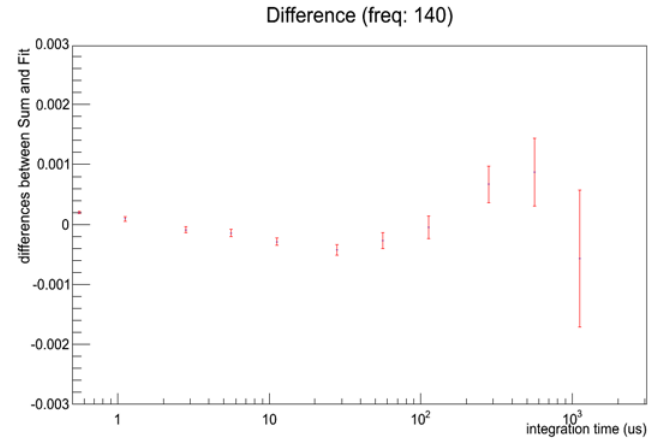


(b)

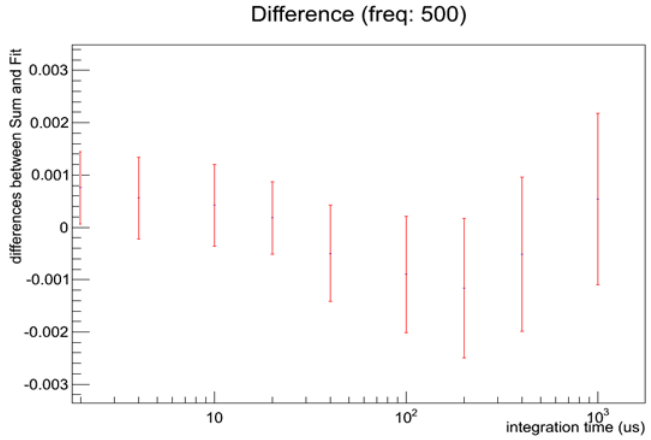


(c)

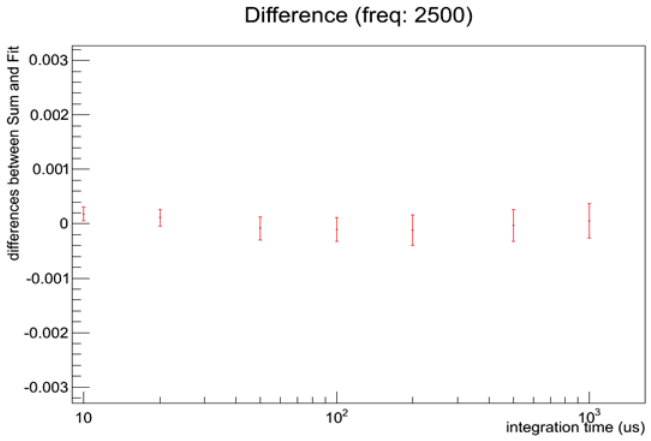
Figure 6.4. Linearity curves obtained by varying the photodetector frequency and the integration time. The three curves pictured show a linear curve for a photodetector frequency of approximately $2\,MHz$ (a), $0.5\,MHz$ (b) and $0.1\,MHz$ (c).



(a)



(b)



(c)

Figure 6.5. Curves representing the difference between the values extrapolated from the fitted points and the actual values measured excluding the saturated measurements normalized with the value assumed by the maximum non saturated measurement. The three curves pictured have been realized setting the photodetector frequency at approximately $2\,MHz$ (a), $0.5\,MHz$ (b) and $0.1\,MHz$ (c).

used, while for higher integration times a frequency of approximately $50\,kHz$ should be chosen.

6.3 Measurements

Having achieved this point, the first measurements with the microcapillary channels have been taken. In order to do so, the light has been introduced in the box through the side hole, irradiating the channels that have been directly laid on the photodiode array; the setup used is shown in figure 6.6.

Figure 6.6. Image of the setup used to take the first measurements with the microcapillary channels.

In this setup, multiple data measurements have been taken filling the microchannels with a fluorescent liquid. The liquid was a saturated solution of Cargille, which is a certified diiodomethane sulfur tin iodide refractive index liquid (n=1.80), and Rhodamine 6G. A picture of the channels filled with such solution is shown in figure 6.7.

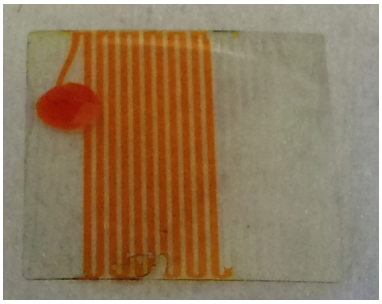


Figure 6.7. Picture of the channels filled with the Cargille and Rhodamine 6G solution.

In such a setup different measurements have been taken by irradiating the previously described michrochannel with a green light generated by different LED and laser sources. In figure 6.8 a typical signal measured from a green light LED source can be seen.

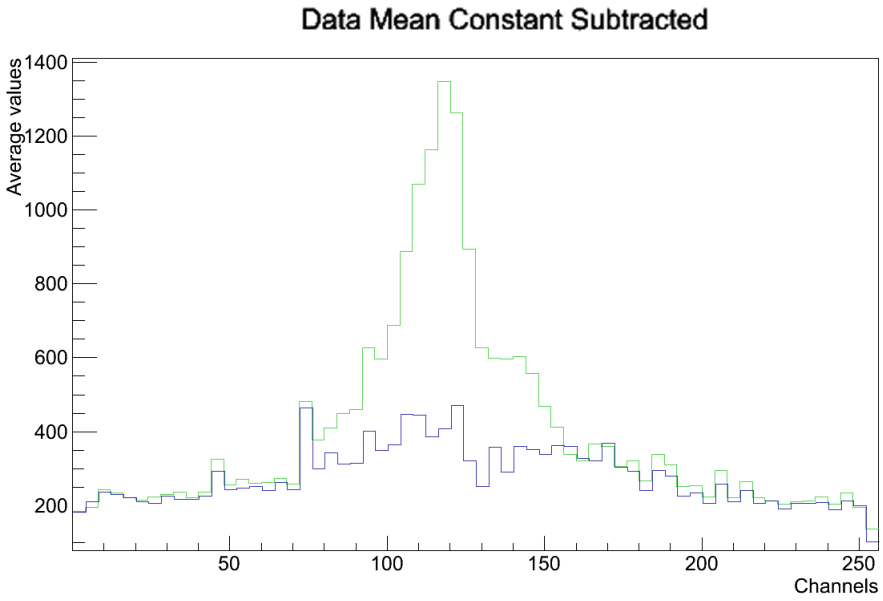


Figure 6.8. Typical signal measured from a green light LED source and its pedestal measured with only the PVC block at a frequency of $250\,kHz$ with an integration time of $0.24s$ using a 256 channels photodiode array.

Chapter 7

Conclusions

The presented thesis has shown the work undertaken in order to develop a readout electronics for a new microcapillary scintillator detector studied for high energy physics and hadrotherapy applications. In particular, different photodetectors and analog to digital converters have been studied to collect the light generated form the described microcapillary channels and to digitalize the measures acquired.

Having selected a Hamamatsu photodiode array and a Texas Instruments 14-bit ADC, a dedicated device has been realized by programming a Xilinx FPGA and by creating a communication protocol to interface a computer with it. Peculiarly, the FPGA board has been programmed to monitor and synchronize the photodetector and the ADC as well as to process the data acquired. Having developed a first prototype of such a device, multiple tests have been performed in order to check the correct functioning of the whole readout system. Finally the linearity of the device has been tested, identifying the best working photodetector frequencies for different integration times, and an initial measure has been taken by illuminating with a green LED the microchannels filled with a fluorescent liquid.

The work undertaken produced a readout system capable of measuring a maximum input of $4.8\,V$ at a maximum frequency of $2\,MHz$ that keeps its linearity features in an integration time range between $1\,\mu s$ and $10\,ms$ and that can be controlled via a gigabit ethernet network.

On this specific detector readout system, further studies should be carried on. In fact, the device needs to be further tested by filling the microchannels with a scintillating material and by irradiating them with protons or other heavy particles; moreover, the software coded to interface the readout with a computer should be improved. Finally, the results obtained from such a developing process should be installed on an accelerating apparatus and different measurements should be taken in such setup.

Appendix A

The VHDL Code

In this section the VHDL code written for the physical devices simulation, the FPGA logic functioning and for their test bench is shown.

A.1 Photodiode Simulation

PhotodiodeArraySimulationTop.vhd

```
1  ------------------------------
2
3  --PhotodiodeArraySimulationTop.vhd
4
5  ------------------------------
6
7  library IEEE;
8  use IEEE.STD_LOGIC_1164.ALL;
9  use IEEE.STD_LOGIC_ARITH.ALL;
10
11 entity PhotodiodeArraySimulationTop is
12 generic ( width : integer :=  14 );
13     Port (  clk : in  STD_LOGIC;
14             reset : in  STD_LOGIC;
15             video : out  STD_LOGIC_VECTOR (width-1 downto 0);
16             trigger : out  STD_LOGIC;
17             eos : out  STD_LOGIC);
18 end PhotodiodeArraySimulationTop;
19
20 architecture Behavioral of PhotodiodeArraySimulationTop is
21
22   component PhotodiodeArraySimulation
23   generic ( width : integer :=  14 );
24     Port (  Clk : in  STD_LOGIC;
25             Reset : in  STD_LOGIC;
26          random_video : in STD_LOGIC_VECTOR (width-1 downto 0);
27             Video : out  STD_LOGIC_VECTOR (width-1 downto 0);
28             Trigger : out  STD_LOGIC;
29             EOS : out  STD_LOGIC);
30   end component;
31
32   component random
33    generic ( width : integer :=  14 );
34   Port (  clk : in std_logic;
35          random_num : out std_logic_vector (width-1 downto 0));
36   end component;
37
38   signal randomInterno : std_logic_vector (width-1 downto 0);
39
```

```
40  begin
41
42    PhotodiodeArraySimulationlogic: PhotodiodeArraySimulation port map(clk=>clk,
43                                reset=>reset,
44                                video=>video,
45                                trigger=>trigger,
46                                random_video=> randomInterno,
47                                eos=>eos);
48
49    randomlogic: random port map( clk=>clk,
50                    random_num=>randomInterno);
51
52  end Behavioral;
```

PhotodiodeArraySimulation.vhd

```
1   ------------------------------
2
3   --PhotodiodeArraySimulation.vhd
4
5   ------------------------------
6
7   library IEEE;
8   use IEEE.STD_LOGIC_1164.ALL;
9   use IEEE.STD_LOGIC_ARITH.ALL;
10
11
12  entity PhotodiodeArraySimulation is
13      generic (   width : integer :=  14);
14      Port (  Clk : in  STD_LOGIC;
15              Reset : in  STD_LOGIC;
16        random_video : in STD_LOGIC_VECTOR (width-1 downto 0);
17              Video : out  STD_LOGIC_VECTOR (width-1 downto 0);
18              Trigger : out  STD_LOGIC;
19              EOS : out  STD_LOGIC);
20  end PhotodiodeArraySimulation;
21
22  architecture Behavioral of PhotodiodeArraySimulation is
23
24  constant maxChannelNumber : integer :=255;
25  signal count : integer range 0 to 6;
26  signal channelCounter : integer range 0 to maxChannelNumber :=0;
27  signal output : std_logic_vector (width-1 downto 0);
28  signal trig : std_logic;
29  signal started : std_logic;
30  signal eosint : std_logic;
31  signal counterFromReset : integer range 0 to 19;
32  signal stopping : std_logic :='0';
33
34  begin
35
36    process(Reset, Clk)
37    begin
38      if (Reset'event and Reset='0') then
39        count<=0;
40        channelCounter<=0;
41        stopping<= '0';
42        output<=(others=>'0');
43        counterFromReset<=1;
44        trig<='0';
45        eosint<='1';
46        counterFromReset<=0;
47      elsif(Reset='0' and rising_edge(Clk) and counterFromReset<17) then
48        counterFromReset<=counterFromReset+1;
49      elsif(channelCounter=(maxChannelNumber-1) and stopping='0') then
50        stopping<= '1';
51      elsif (rising_edge(Clk) and counterFromReset = 17) then
52        if count=0 then
53          count<=count+1;
54          output<=(random_video);
55        elsif count=1 then
56          count<=count+1;
57          channelCounter<=channelCounter+1;
```

```
58          trig<='1';
59        elsif count=2 then
60          count<=count+1;
61          output<=(others=>'0');
62          trig<='0';
63          if (stopping='1' and eosint='1') then
64            eosint<='0';
65            count<=4;
66          end if;
67        elsif count=3 then
68          count<=0;
69        elsif count=7 then
70          eosint<='1';
71        else
72          count<=count+1;
73        end if;
74      Video <= output;
75      Trigger <= trig;
76      EOS <= eosint;
77      end if;
78
79    end process;
80
81
82  end Behavioral;
```

Random.vhd

```
1   ------------------------------
2
3   --Random.vhd
4
5   ------------------------------
6
7   library IEEE;
8   use IEEE.STD_LOGIC_1164.ALL;
9
10  entity Random is
11      generic ( width : integer :=  14 );
12    Port (  clk : in std_logic;
13        random_num : out std_logic_vector (width-1 downto 0));
14  end Random;
15
16  architecture Behavioral of Random is
17  begin
18    process(clk)
19
20    variable rand_temp : std_logic_vector(width-1 downto 0):=(width-1=>'1', OTHERS=>'0');
21    variable temp : std_logic := '0';
22
23    begin
24
25      if(rising_edge(clk)) then
26        temp := rand_temp(width-1) xor rand_temp(width-2);
27        rand_temp(width-1 downto 1) := rand_temp(width-2 downto 0);
28        rand_temp(0) := temp;
29      end if;
30      random_num <= rand_temp;
31
32    end process;
33
34  end Behavioral;
```

A.2 ADC Simulation

```
1   ------------------------------
2
3   --AdcSimulation.vhd
4
5   ------------------------------
6
```

```
7  library IEEE;
8  use IEEE.STD_LOGIC_1164.ALL;
9  use IEEE.NUMERIC_STD.ALL;
10
11 entity AdcSimulation is
12   generic ( width : integer :=  14 );
13     Port (  input : in  STD_LOGIC_VECTOR (width-1 downto 0);
14             clk : in STD_LOGIC;
15       cs_n : in STD_LOGIC;
16             sdo : out  STD_LOGIC);
17 end AdcSimulation;
18
19 architecture Behavioral of AdcSimulation is
20
21 signal n : integer range 0 to width+3 :=0;
22 signal internal_data : std_logic_vector (width-1 downto 0);
23 signal waiting : std_logic;
24 signal start : std_logic;
25
26 begin
27   process(clk, cs_n)
28   begin
29     if (cs_n'event and cs_n ='0') then
30       sdo<='Z';
31       start<='1';
32       waiting<='0';
33       internal_data<=input;
34       n<=0;
35     elsif(rising_edge(clk) and start='1') then
36       if(waiting='0') then
37         sdo<=internal_data(width-1-n);
38       end if;
39       if (n=width-1) then
40         waiting<='1';
41         internal_data<=(others=>'0');
42       elsif(n=width) then
43         sdo<='0';
44       elsif(n=(width+2)) then
45         n<=0;
46         start<='0';
47         waiting<='0';
48       end if;
49       n<=n+1;
50     end if;
51   end process;
52
53 end Behavioral;
```

A.3 FPGA Logic

userlogic.vhd

```
1  ------------------------------------------------------------------------------
2  -- user_logic.vhd - entity/architecture pair
3  ------------------------------------------------------------------------------
4  --
5  -- ***************************************************************************
6  -- ** Copyright (c) 1995-2012 Xilinx, Inc.  All rights reserved.            **
7  -- **                                                                       **
8  -- ** Xilinx, Inc.                                                         **
9  -- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"        **
10 -- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND      **
11 -- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,       **
12 -- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,       **
13 -- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION          **
14 -- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,    **
15 -- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE     **
16 -- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY             **
17 -- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE              **
18 -- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR       **
19 -- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF      **
```

```
20  -- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
21  -- ** FOR A PARTICULAR PURPOSE.                                             **
22  -- **                                                                       **
23  -- ***************************************************************************
24  --
25  -------------------------------------------------------------------------------
26  -- Filename:          user_logic.vhd
27  -- Version:           2.00.a
28  -- Description:       User logic.
29  -- Date:              Fri Jun 22 16:08:56 2012 (by Create and Import Peripheral Wizard)
30  -- VHDL Standard:     VHDL'93
31  -------------------------------------------------------------------------------
32  -- Naming Conventions:
33  --   active low signals:                    "*_n"
34  --   clock signals:                         "clk", "clk_div#", "clk_#x"
35  --   reset signals:                         "rst", "rst_n"
36  --   generics:                              "C_*"
37  --   user defined types:                    "*_TYPE"
38  --   state machine next state:              "*_ns"
39  --   state machine current state:           "*_cs"
40  --   combinatorial signals:                 "*_com"
41  --   pipelined or register delay signals:   "*_d#"
42  --   counter signals:                       "*cnt*"
43  --   clock enable signals:                  "*_ce"
44  --   internal version of output port:       "*_i"
45  --   device pins:                           "*_pin"
46  --   ports:                                 "- Names begin with Uppercase"
47  --   processes:                             "*_PROCESS"
48  --   component instantiations:              "<ENTITY_>I_<#|FUNC>"
49  -------------------------------------------------------------------------------
50
51  -- DO NOT EDIT BELOW THIS LINE ----------------------
52  library ieee;
53  use ieee.std_logic_1164.all;
54  use ieee.std_logic_arith.all;
55  use ieee.std_logic_unsigned.all;
56
57  library proc_common_v3_00_a;
58  use proc_common_v3_00_a.proc_common_pkg.all;
59
60  -- DO NOT EDIT ABOVE THIS LINE ----------------------
61
62  --USER libraries added here
63
64  library unisim;
65  use unisim.vcomponents.all;
66
67  -------------------------------------------------------------------------------
68  -- Entity section
69  -------------------------------------------------------------------------------
70  -- Definition of Generics:
71  --   C_SLV_DWIDTH                 -- Slave interface data bus width
72  --   C_NUM_REG                    -- Number of software accessible registers
73  --   C_RDFIFO_DEPTH               -- Read FIFO depth
74  --
75  -- Definition of Ports:
76  --   Bus2IP_Clk                   -- Bus to IP clock
77  --   Bus2IP_Reset                 -- Bus to IP reset
78  --   Bus2IP_Data                  -- Bus to IP data bus
79  --   Bus2IP_BE                    -- Bus to IP byte enables
80  --   Bus2IP_RdCE                  -- Bus to IP read chip enable
81  --   Bus2IP_WrCE                  -- Bus to IP write chip enable
82  --   IP2Bus_Data                  -- IP to Bus data bus
83  --   IP2Bus_RdAck                 -- IP to Bus read transfer acknowledgement
84  --   IP2Bus_WrAck                 -- IP to Bus write transfer acknowledgement
85  --   IP2Bus_Error                 -- IP to Bus error response
86  --   IP2RFIFO_WrReq               -- IP to RFIFO : IP write request
87  --   IP2RFIFO_Data                -- IP to RFIFO : IP write data bus
88  --   RFIFO2IP_WrAck               -- RFIFO to IP : RFIFO write acknowledge
89  --   RFIFO2IP_AlmostFull          -- RFIFO to IP : RFIFO almost full
90  --   RFIFO2IP_Full                -- RFIFO to IP : RFIFO full
91  --   RFIFO2IP_Vacancy             -- RFIFO to IP : RFIFO vacancy
92  -------------------------------------------------------------------------------
93
```

```
94  entity user_logic is
95    generic
96    (
97      -- ADD USER GENERICS BELOW THIS LINE ---------------
98      --USER generics added here
99    dataLength  : integer := 16;
100   integrationTimeBitMax : integer:=8;
101   channelLength : integer := 8;
102   fifoDepth   : integer := 2048;
103   hamDivisorLength : integer :=12;
104   adcDivisorLength : integer := 6;
105     -- ADD USER GENERICS ABOVE THIS LINE ---------------
106
107     -- DO NOT EDIT BELOW THIS LINE ----------------------
108     -- Bus protocol parameters, do not add to or delete
109     C_SLV_DWIDTH                   : integer              := 32;
110     C_NUM_REG                      : integer              := 8;
111     C_RDFIFO_DEPTH                 : integer              := 16384
112     -- DO NOT EDIT ABOVE THIS LINE ----------------------
113   );
114   port
115   (
116     -- ADD USER PORTS BELOW THIS LINE ------------------
117     --USER ports added here
118
119     Ham_Eos                       : in  std_logic;
120     Ham_Trigger                   : in  std_logic;
121     Ham_ClkIn                     : in  std_logic;
122     Ham_ClkOut                    : out  std_logic;
123     Ham_Reset_n                   : out std_logic;
124     Ham_Gain                      : out std_logic;
125     Ham_InStart                   : out std_logic;
126     Adc_SDO                       : in  std_logic;
127     Adc_ClkOut                    : out  std_logic;
128     Adc_CS_n                      : out std_logic;
129     Adc_PDEn                      : out std_logic;
130     Adc_ChSel                     : out std_logic;
131     Gen_Clk                       : in  std_logic;
132     Gen_L0_p                      : in  std_logic;
133     Gen_L0_n                      : in  std_logic;
134     Gen_L1_p                      : in  std_logic;
135     Gen_L1_n                      : in  std_logic;
136     Gen_L2_p                      : out std_logic;
137     Gen_L2_n                      : out std_logic;
138     Gen_L3_p                      : out std_logic;
139     Gen_L3_n                      : out std_logic;
140     Gen_C0                        : in  std_logic;
141     Gen_C1                        : in  std_logic;
142     Gen_C2                        : out std_logic;
143     Gen_C3                        : out std_logic;
144     Gen_ClkOut_p                  : out std_logic;
145     Gen_ClkOut_n                  : out std_logic;
146     -- ADD USER PORTS ABOVE THIS LINE ------------------
147
148     -- DO NOT EDIT BELOW THIS LINE ----------------------
149     -- Bus protocol ports, do not add to or delete
150     Bus2IP_Clk                     : in  std_logic;
151     Bus2IP_Reset                   : in  std_logic;
152     Bus2IP_Data                    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
153     Bus2IP_BE                      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
154     Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
155     Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
156     IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
157     IP2Bus_RdAck                   : out std_logic;
158     IP2Bus_WrAck                   : out std_logic;
159     IP2Bus_Error                   : out std_logic;
160     IP2RFIFO_WrReq                 : out std_logic;
161     IP2RFIFO_Data                  : out std_logic_vector(0 to C_SLV_DWIDTH-1);
162     RFIFO2IP_WrAck                 : in  std_logic;
163     RFIFO2IP_AlmostFull            : in  std_logic;
164     RFIFO2IP_Full                  : in  std_logic;
165     RFIFO2IP_Vacancy               : in  std_logic_vector(0 to log2(C_RDFIFO_DEPTH))
166     -- DO NOT EDIT ABOVE THIS LINE ----------------------
167   );
```

```
168
169   attribute MAX_FANOUT : string;
170   attribute SIGIS : string;
171
172   attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
173   attribute SIGIS of Bus2IP_Reset  : signal is "RST";
174
175 end entity user_logic;
176
177 ------------------------------------------------------------------------------
178 -- Architecture section
179 ------------------------------------------------------------------------------
180
181 architecture IMP of user_logic is
182
183   component FPGA
184   generic (   wordLength  : integer :=32;
185         dataLength  : integer :=16;
186         integrationTimeBitMax : integer:=8; --max integration time expressed in hamclk*2^(integrationTimeBitMax)
187         channelLength : integer :=8;
188         hamDivisorLength : integer :=12;
189         adcDivisorLength : integer := 6;
190         fifoDepth   : integer :=2048);
191   Port (    rst_n : in std_logic;
192         trigger : in std_logic;
193         FPGA_Clk : in std_logic;
194         integTime : in std_logic_vector(integrationTimeBitMax-1 downto 0);
195         FSM_Clk : in std_logic;
196         Ham_Eos : in std_logic;
197         extTrigger : in std_logic;
198         deseInput : in std_logic;
199         Test : in std_logic;
200         extFIFOFull : in std_logic;
201         triggerDaInvRst_n : in std_logic;
202         divisorHam : in  std_logic_vector(0 to hamDivisorLength-1);
203         divisorAdc : in  std_logic_vector(0 to adcDivisorLength-1);
204         triggerDaInv : out std_logic;
205         extTrigOut : out std_logic_vector(wordLength-2 downto 0);
206         wordOut : out  std_logic_VECTOR (wordLength-1 downto 0);
207         photodiodeClkOut : out std_logic;
208         photodiodeStart : out std_logic;
209         ADC_ClkOut : out std_logic;
210         intFIFOFull : out std_logic;
211         extFifoWrEn : out std_logic;
212         intFIFOEmpty : out std_logic;
213         busy : out std_logic;
214         adcPDEN : out std_logic;
215         ADCStart : out std_logic);
216   end component;
217
218   signal L_int   : std_logic_vector ( 3 downto 0);
219   signal L_int_n : std_logic_vector ( 3 downto 0);
220
221   signal sFIFOFull        : std_logic;
222   signal sFIFOEmpty       : std_logic;
223   signal sFIFOProgFull      : std_logic;
224   signal striggerDaInv      : std_logic;
225   signal rst_n            : std_logic;
226   signal triggerAnsw      : std_logic;
227   signal integTime            : std_logic_vector(0 to integrationTimeBitMax-1);
228   signal sHamDivisor      : std_logic_vector(0 to hamDivisorLength-1);
229   signal sAdcDivisor      : std_logic_vector(0 to adcDivisorLength-1);
230   signal Gen_C0_enable    : std_logic;
231
232   --USER signal declarations added here, as needed for user logic
233
234   ------------------------------------------
235   -- Signals for user logic slave model s/w accessible register example
236   ------------------------------------------
237   signal slv_reg0                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
238   signal slv_reg1                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
239   signal slv_reg2                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
240   signal slv_reg3                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
241   signal slv_reg4                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
```

```
242   signal slv_reg5                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
243   signal slv_reg6                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
244   signal slv_reg7                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
245   signal slv_reg_write_sel              : std_logic_vector(0 to 7);
246   signal slv_reg_read_sel               : std_logic_vector(0 to 7);
247   signal slv_ip2bus_data                : std_logic_vector(0 to C_SLV_DWIDTH-1);
248   signal slv_read_ack                   : std_logic;
249   signal slv_write_ack                  : std_logic;
250   signal orTrigExt            : std_logic;
251   signal Test             : std_logic;
252   signal extTrigCount         : std_logic_vector(0 to C_SLV_DWIDTH-2);
253   signal TrigExt            : std_logic;
254   signal sGen_C0            : std_logic;
255   signal FIFOWrEn           : std_logic;
256
257 begin
258
259   --USER logic implementation added here
260
261   FPGAlogic: FPGA generic map ( wordLength => C_SLV_DWIDTH,
262                                 dataLength => dataLength,
263                                 integrationTimeBitMax => integrationTimeBitMax,
264                                 channelLength => channelLength,
265                                 hamDivisorLength => hamDivisorLength,
266                adcDivisorLength => adcDivisorLength,
267                                 fifoDepth => fifoDepth)
268     port map( rst_n         => rst_n,
269               trigger       => Ham_Trigger,
270               FPGA_Clk      => Gen_Clk,
271               integTime     => integTime,
272               Test        => Test,
273               Ham_Eos       => Ham_Eos,
274               FSM_Clk         => Bus2IP_Clk,
275               extTrigger      => TrigExt,
276               deseInput       => Adc_SDO,
277               extFIFOFull     => RFIFO2IP_Full,
278               triggerDaInvRst_n => triggerAnsw,
279               triggerDaInv    => striggerDaInv,
280               extTrigOut      => extTrigCount,
281               divisorHam      => sHamDivisor,
282               divisorAdc      => sAdcDivisor,
283               wordOut       => IP2RFIFO_Data,
284               photodiodeClkOut  => Ham_ClkOut,
285               --photodiodeStart   => Ham_InStart,
286         photodiodeStart   => Ham_Reset_n,
287               ADC_ClkOut      => Adc_ClkOut,
288               intFIFOFull     => sFIFOFull,
289               intFIFOEmpty    => sFIFOEmpty,
290               extFifoWrEn     => FIFOWrEn,
291               busy        => sFIFOProgFull,
292               adcPDEN       => Adc_PDEn,
293               ADCStart      => Adc_CS_n);
294
295   ------------------------------------------------------------
296   -- Example code to read/write user logic slave model s/w accessible registers
297   --
298   -- Note:
299   -- The example code presented here is to show you one way of reading/writing
300   -- software accessible registers implemented in the user logic slave model.
301   -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
302   -- to one software accessible register by the top level template. For example,
303   -- if you have four 32 bit software accessible registers in the user logic,
304   -- you are basically operating on the following memory mapped registers:
305   --
306   --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
307   --                     "1000"   C_BASEADDR + 0x0
308   --                     "0100"   C_BASEADDR + 0x4
309   --                     "0010"   C_BASEADDR + 0x8
310   --                     "0001"   C_BASEADDR + 0xC
311   --
312   ---------------------------------------
313   slv_reg_write_sel <= Bus2IP_WrCE(0 to 7);
314   slv_reg_read_sel  <= Bus2IP_RdCE(0 to 7);
315   slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or Bus2IP_WrCE(3)
```

```
316                or Bus2IP_WrCE(4) or Bus2IP_WrCE(5) or Bus2IP_WrCE(6) or Bus2IP_WrCE(7);
317    slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or Bus2IP_RdCE(3)
318                or Bus2IP_RdCE(4) or Bus2IP_RdCE(5) or Bus2IP_RdCE(6) or Bus2IP_RdCE(7);
319
320    -- implement slave model software accessible register(s)
321    SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
322    begin
323
324      if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
325        if Bus2IP_Reset = '1' then
326          slv_reg0 <= (others => '0');
327          slv_reg1 <= (others => '0');
328          slv_reg2 <= (others => '0');
329          slv_reg3 <= (others => '0');
330          slv_reg4 <= (others => '0');
331          slv_reg5 <= (others => '0');
332          slv_reg6 <= (others => '0');
333          slv_reg7 <= (others => '0');
334        else
335          case slv_reg_write_sel is
336            when "10000000" =>
337              for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
338                if ( Bus2IP_BE(byte_index) = '1' ) then
339                  slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
340                end if;
341              end loop;
342            when "01000000" =>
343              for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
344                if ( Bus2IP_BE(byte_index) = '1' ) then
345                  slv_reg1(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
346                end if;
347              end loop;
348            when "00100000" =>
349              for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
350                if ( Bus2IP_BE(byte_index) = '1' ) then
351                  slv_reg2(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
352                end if;
353              end loop;
354            when "00010000" =>
355              for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
356                if ( Bus2IP_BE(byte_index) = '1' ) then
357                  slv_reg3(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
358                end if;
359              end loop;
360            when "00001000" =>
361              for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
362                if ( Bus2IP_BE(byte_index) = '1' ) then
363                  slv_reg4(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
364                end if;
365              end loop;
366            when "00000100" =>
367              for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
368                if ( Bus2IP_BE(byte_index) = '1' ) then
369                  slv_reg5(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
370                end if;
371              end loop;
372            when "00000010" =>
373              for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
374                if ( Bus2IP_BE(byte_index) = '1' ) then
375                  slv_reg6(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
376                end if;
377              end loop;
378            when "00000001" =>
379              for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
380                if ( Bus2IP_BE(byte_index) = '1' ) then
381                  slv_reg7(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
382                end if;
383              end loop;
384            when others => null;
385          end case;
386        end if;
387      end if;
388
389    end process SLAVE_REG_WRITE_PROC;
```

```
390
391   -- implement slave model software accessible register(s) read mux
392   SLAVE_REG_READ_PROC : process(extTrigCount, RFIFO2IP_AlmostFull, RFIFO2IP_Full, sFIFOProgFull,
393               sFIFOEmpty, sFIFOFull, striggerDaInv, slv_reg_read_sel, slv_reg0, slv_reg1,
394               slv_reg2, slv_reg3, slv_reg4, slv_reg5, slv_reg6, slv_reg7, L_int) is
395
396   begin
397
398     case slv_reg_read_sel is
399       when "10000000" => slv_ip2bus_data <= slv_reg0;
400       when "01000000" => slv_ip2bus_data <= slv_reg1;
401       when "00100000" => slv_ip2bus_data <= slv_reg2;
402       when "00010000" => slv_ip2bus_data <= slv_reg3;
403       when "00001000" => slv_ip2bus_data <= L_int(1) & L_int(0) & Gen_C1 & Gen_C0 &
404                                             "0000" &
405                                             "0000" &
406                                             "0000" &
407                                             striggerDaInv & "000" &
408                                             "0000" &
409                                             "0000" &
410                                             "0000";
411       when "00000100" => slv_ip2bus_data <= '0' & extTrigCount;
412       when "00000010" => slv_ip2bus_data <= slv_reg6;
413       when "00000001" => slv_ip2bus_data <= sFIFOFull & sFIFOEmpty & sFIFOProgFull & RFIFO2IP_Full &
414                                             RFIFO2IP_AlmostFull & "000" &
415                                             "0000" &
416                                             "0000" &
417                                             "0000" &
418                                             "0000" &
419                                             "0000" &
420                                             "0000";
421
422       when others => slv_ip2bus_data <= (others => '0');
423     end case;
424
425   end process SLAVE_REG_READ_PROC;
426
427   ----------------------------------------
428   -- Example code to drive IP to Bus signals
429   ----------------------------------------
430   IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
431                   (others => '0');
432
433   IP2Bus_WrAck <= slv_write_ack;
434   IP2Bus_RdAck <= slv_read_ack;
435   IP2Bus_Error <= '0';
436
437 --
438 -- all output configuration bits must be connecetd here to slv_regx
439 --
440   L_int(2)     <= slv_reg0(31);         --bit 0 for readout
441   L_int(3)     <= slv_reg0(30);
442   rst_n        <= slv_reg0(29);
443   triggerAnsw  <= slv_reg0(28);
444   sHamDivisor(11) <= slv_reg0(27);
445   sHamDivisor(10) <= slv_reg0(26);
446   sHamDivisor(9)  <= slv_reg0(25);
447   sHamDivisor(8)  <= slv_reg0(24);
448   sHamDivisor(7)  <= slv_reg0(23);
449   sHamDivisor(6)  <= slv_reg0(22);
450   sHamDivisor(5)  <= slv_reg0(21);
451   sHamDivisor(4)  <= slv_reg0(20);
452   sHamDivisor(3)  <= slv_reg0(19);
453   sHamDivisor(2)  <= slv_reg0(18);
454   sHamDivisor(1)  <= slv_reg0(17);
455   sHamDivisor(0)  <= slv_reg0(16);
456   Gen_C2          <= slv_reg0(15);
457   Gen_C3          <= slv_reg0(14);
458   sAdcDivisor(5)  <= slv_reg0(13);
459   sAdcDivisor(4)  <= slv_reg0(12);
460   sAdcDivisor(3)  <= slv_reg0(11);
461   sAdcDivisor(2)  <= slv_reg0(10);
462   sAdcDivisor(1)  <= slv_reg0(9);
463   sAdcDivisor(0)  <= slv_reg0(8);
```

```
464     Test       <= slv_reg0(0);
465
466     Ham_Gain     <= slv_reg1(31);
467     --Ham_Reset_n  <= slv_reg1(30);
468     --Ham_InStart not connected
469     integTime(7) <= slv_reg1(29);
470     integTime(6) <= slv_reg1(28);
471     integTime(5) <= slv_reg1(27);
472     integTime(4) <= slv_reg1(26);
473     integTime(3) <= slv_reg1(25);
474     integTime(2) <= slv_reg1(24);
475     integTime(1) <= slv_reg1(23);
476     integTime(0) <= slv_reg1(22);
477     Gen_C0_enable <= slv_reg1(1);
478     orTrigExt    <= slv_reg1(0);
479
480     --Adc_PDEn    <= slv_reg2(31);
481     Adc_ChSel   <= slv_reg2(30);
482
483     sGen_C0   <= Gen_C0 and Gen_C0_enable;
484     TrigExt   <= (sGen_C0 or orTrigExt);
485     IP2RFIFO_WrReq    <= FIFOWrEn;
486
487   -- VHDL instantiation:
488
489   L0_1 : IBUFDS  port map (I=>Gen_L0_p, IB=> Gen_L0_n, O=>L_int(0));
490   L1_1 : IBUFDS  port map (I=>Gen_L1_p, IB => Gen_l1_n, O=>L_int(1));
491   L2_1 : OBUFDS  port map (I=>L_int(2), O=>Gen_L2_p, OB=> Gen_L2_n);
492   L3_1 : OBUFDS  port map (I=>L_int(3), O=>Gen_L3_p, OB=> Gen_L3_n);
493   C0_1 : OBUFDS  port map (I=>Gen_Clk, O=>Gen_ClkOut_p, OB=> Gen_ClkOut_n);
494
495   end IMP;
```

FPGA.vhd

```
1   ----------------------------------------------------------------------------------
2
3   --FPGA.vhd
4
5   ----------------------------------------------------------------------------------
6   library IEEE;
7   use IEEE.std_logic_1164.ALL;
8   use IEEE.NUMERIC_STD.ALL;
9
10
11  entity FPGA is
12    generic (   wordLength  : integer :=32;
13          dataLength  : integer :=16;
14          integrationTimeBitMax : integer:=8;
15          channelLength : integer :=8;
16          hamDivisorLength : integer :=12;
17          adcDivisorLength : integer := 6;
18          fifoDepth   : integer :=2048);
19    Port (    rst_n : in std_logic;
20          trigger : in std_logic;
21          FPGA_Clk : in std_logic;
22          integTime : in std_logic_vector(integrationTimeBitMax-1 downto 0);
23          FSM_Clk : in std_logic;
24          Ham_Eos : in std_logic;
25          Test : in std_logic;
26          extTrigger : in std_logic;
27          deseInput : in std_logic;
28          divisorHam : in  std_logic_vector(0 to hamDivisorLength-1);
29          divisorAdc : in  std_logic_vector(0 to adcDivisorLength-1);
30          extFIFOFull : in std_logic;
31          triggerDaInvRst_n : in std_logic;
32          extTrigOut : out std_logic_vector(wordLength-2 downto 0);
33          triggerDaInv : out std_logic;
34          wordOut : out  std_logic_VECTOR (wordLength-1 downto 0);
35          photodiodeClkOut : out std_logic;
36          photodiodeStart : out std_logic;
37          ADC_ClkOut : out std_logic;
38          intFIFOFull : out std_logic;
```

```
39          intFIFOEmpty : out std_logic;
40          extFifoWrEn : out std_logic;
41          busy : out std_logic;
42          adcPDEN : out std_logic;
43          ADCStart : out std_logic);
44  end FPGA;
45
46  architecture Behavioral of FPGA is
47
48
49    component LogicControlUnit
50          generic (   width : integer;
51            dataLength : integer;
52            integrationTimeBitMax : integer;
53            channelLength : integer);
54    Port (  reset_n : in std_logic;
55        extTrigger : in std_logic;
56        clk : in std_logic;
57        integTime : in std_logic_vector(integrationTimeBitMax-1 downto 0);
58        Ham_Eos : in std_logic;
59        clkAdc : in std_logic;
60        clkHam : in std_logic;
61        Test : in std_logic;
62        trigger : in std_logic;
63        dataIn : in std_logic_vector(dataLength-1 downto 0);
64        dataReadyFromDeserializer : in std_logic;
65        triggerDaInvRst_n : in std_logic;
66        extTrigOut : out std_logic_vector(width-2 downto 0);
67        triggerDaInv : out std_logic;
68        adcStart : out std_logic;
69        photodiodeStart : out std_logic;
70        deserializerStart : out std_logic;
71        writeEnable : out std_logic;
72        adcPDEN : out std_logic;
73        wordOut : out std_logic_vector(width-1 downto 0));
74    end component;
75
76    component Deserializzatore
77    generic ( width : integer);
78    Port (    reset_n : in  STD_LOGIC;
79          clk : in STD_LOGIC;
80          cs_n : in STD_LOGIC;
81          datain : in  STD_LOGIC;
82          ready : out STD_LOGIC;
83          dataout : out  STD_LOGIC_VECTOR (width-1 downto 0));
84    end component;
85
86    component intFIFO
87      port(    rst : in  std_logic;
88           wr_clk : in  std_logic;
89           rd_clk : in  std_logic;
90           din : in  std_logic_vector(31 downto 0);
91           wr_en : in  std_logic;
92           rd_en : in  std_logic;
93           dout : out  std_logic_vector(31 downto 0);
94           full : out  std_logic;
95           empty : out  std_logic;
96               prog_full : OUT  std_logic);
97    end component;
98
99    component FSM
100   port( intEmpty:   in std_logic;
101       extFull:    in std_logic;
102       clock:    in std_logic;
103       reset_n:    in std_logic;
104       wr_en:      out std_logic;
105       rd_en:    out std_logic);
106   end component;
107
108   component ClockDivider
109   generic (   lengthDivisor   : integer :=12);
110   Port (  reset_n : in std_logic;
111       clkIn : in std_logic;
112       divisor : in std_logic_vector (0 to lengthDivisor-1);
```

```
113         clkOut : out std_logic);
114     end component;
115 
116     signal deserStart: std_logic;
117     signal deserReady: std_logic;
118     signal deserOut : std_logic_vector(15 downto 0);
119     signal LogicWordOut : std_logic_vector(31 downto 0);
120     signal LogicWriteEn : std_logic;
121     signal intFIFOEmp : std_logic;
122     signal FIFORdEn : std_logic;
123     signal zFifoFull : std_logic;
124     signal adcIntClk : std_logic;
125     signal hamIntClk : std_logic;
126 
127   begin
128 
129     LogicControlUnitlogic: LogicControlUnit generic map (   width => wordLength,
130                                 dataLength => dataLength,
131                                 integrationTimeBitMax => integrationTimeBitMax,
132                                 channelLength => channelLength)
133                         port map( reset_n=>rst_n,
134                                 clk=>FPGA_Clk,
135                                 Ham_Eos=>Ham_Eos,
136                                 clkAdc=>adcIntClk,
137                                 clkHam=>hamIntClk,
138                                 integTime => integTime,
139                                 extTrigger=>extTrigger,
140                                 Test=>Test,
141                                 trigger=>trigger,
142                                 dataIn=>deserOut,
143                                 dataReadyFromDeserializer=>deserReady,
144                                 triggerDaInvRst_n=>triggerDaInvRst_n,
145                                 triggerDaInv=>triggerDaInv,
146                                 extTrigOut=>extTrigOut,
147                                 adcStart=>ADCStart,
148                                 photodiodeStart=>photodiodeStart,
149                                 deserializerStart=>deserStart,
150                                 writeEnable=>LogicWriteEn,
151                                 adcPDEN=>adcPDEN,
152                                 wordOut=>LogicWordOut);
153 
154     Deserializzatorelogic: Deserializzatore generic map( width => dataLength)
155                           port map( reset_n=>rst_n,
156                                 clk=>adcIntClk,
157                                 cs_n=>deserStart,
158                                 datain=>deseInput,
159                                 ready=>deserReady,
160                                 dataout=>deserOut);
161 
162     ClockHamDividerlogic: ClockDivider generic map( lengthDivisor => hamDivisorLength)
163                         port map(   reset_n=>rst_n,
164                                 clkIn=>FPGA_Clk,
165                                 divisor=>divisorHam,
166                                 clkOut=>hamIntClk);
167 
168     ClockADCDividerlogic: ClockDivider generic map( lengthDivisor => adcDivisorLength)
169                         port map(   reset_n=>rst_n,
170                                 clkIn=>FPGA_Clk,
171                                 divisor=>divisorADC,
172                                 clkOut=>adcIntClk);
173 
174     FIFOlogic : intFIFO
175                port map(           rst=>not(rst_n),
176                                 wr_clk=>FPGA_Clk,
177                                 rd_clk=>FSM_Clk,
178                                 din=>LogicWordOut,
179                                 wr_en=>LogicWriteEn,
180                                 rd_en=>FIFORdEn,
181                                 dout=>wordOut,
182                                 full=>intFIFOFull,
183                                 empty=>intFIFOEmp,
184                                 prog_full=>busy);
185 
186     FSMlogic : FSM port map(              intEmpty=>intFIFOEmp,
```

```
187                             extFull=>extFIFOFull,
188                             clock=>FSM_Clk,
189                             reset_n=>rst_n,
190                             wr_en=>extFifoWrEn,
191                             rd_en=>FIFORdEn);
192
193
194    intFIFOEmpty  <= intFIFOEmp;
195    ADC_ClkOut    <= adcIntClk;
196    photodiodeClkOut<= hamIntClk;
197
198  end Behavioral;
```

intFIFO.vhd

```
1   ------------------------------------------------------------------------------
2   --    This file is owned and controlled by Xilinx and must be used solely     --
3   --    for design, simulation, implementation and creation of design files     --
4   --    limited to Xilinx devices or technologies. Use with non-Xilinx          --
5   --    devices or technologies is expressly prohibited and immediately         --
6   --    terminates your license.                                                --
7   --                                                                            --
8   --    XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" SOLELY    --
9   --    FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR XILINX DEVICES.  BY    --
10  --    PROVIDING THIS DESIGN, CODE, OR INFORMATION AS ONE POSSIBLE             --
11  --    IMPLEMENTATION OF THIS FEATURE, APPLICATION OR STANDARD, XILINX IS      --
12  --    MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION IS FREE FROM ANY      --
13  --    CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE FOR OBTAINING ANY       --
14  --    RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY       --
15  --    DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE   --
16  --    IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR          --
17  --    REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF         --
18  --    INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A   --
19  --    PARTICULAR PURPOSE.                                                     --
20  --                                                                            --
21  --    Xilinx products are not intended for use in life support appliances,    --
22  --    devices, or systems.  Use in such applications are expressly            --
23  --    prohibited.                                                             --
24  --                                                                            --
25  --    (c) Copyright 1995-2012 Xilinx, Inc.                                    --
26  --    All rights reserved.                                                    --
27  ------------------------------------------------------------------------------
28  ------------------------------------------------------------------------------
29  -- You must compile the wrapper file FIFO.vhd when simulating
30  -- the core, FIFO. When compiling the wrapper file, be sure to
31  -- reference the XilinxCoreLib VHDL simulation library. For detailed
32  -- instructions, please refer to the "CORE Generator Help".
33
34  -- The synthesis directives "translate_off/translate_on" specified
35  -- below are supported by Xilinx, Mentor Graphics and Synplicity
36  -- synthesis tools. Ensure they are correct for your synthesis tool(s).
37
38  LIBRARY ieee;
39  USE ieee.std_logic_1164.ALL;
40  -- synthesis translate_off
41  LIBRARY XilinxCoreLib;
42  -- synthesis translate_on
43  ENTITY intFIFO IS
44    PORT (  rst : IN STD_LOGIC;
45        wr_clk : IN STD_LOGIC;
46        rd_clk : IN STD_LOGIC;
47        din : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
48        wr_en : IN STD_LOGIC;
49        rd_en : IN STD_LOGIC;
50        dout : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
51        full : OUT STD_LOGIC;
52        empty : OUT STD_LOGIC;
53        prog_full : OUT STD_LOGIC);
54  END intFIFO;
55  ARCHITECTURE FIFO_a OF intFIFO IS
56  -- synthesis translate_off
57  COMPONENT wrapped_FIFO
58    PORT (
```

```
59      rst : IN STD_LOGIC;
60      wr_clk : IN STD_LOGIC;
61      rd_clk : IN STD_LOGIC;
62      din : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
63      wr_en : IN STD_LOGIC;
64      rd_en : IN STD_LOGIC;
65      dout : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
66      full : OUT STD_LOGIC;
67      empty : OUT STD_LOGIC;
68      prog_full : OUT STD_LOGIC
69    );
70  END COMPONENT;
71
72  -- Configuration specification
73    FOR ALL : wrapped_FIFO USE ENTITY XilinxCoreLib.fifo_generator_v8_4(behavioral)
74      GENERIC MAP (
75        c_add_ngc_constraint => 0,
76        c_application_type_axis => 0,
77        c_application_type_rach => 0,
78        c_application_type_rdch => 0,
79        c_application_type_wach => 0,
80        c_application_type_wdch => 0,
81        c_application_type_wrch => 0,
82        c_axi_addr_width => 32,
83        c_axi_aruser_width => 1,
84        c_axi_awuser_width => 1,
85        c_axi_buser_width => 1,
86        c_axi_data_width => 64,
87        c_axi_id_width => 4,
88        c_axi_ruser_width => 1,
89        c_axi_type => 0,
90        c_axi_wuser_width => 1,
91        c_axis_tdata_width => 64,
92        c_axis_tdest_width => 4,
93        c_axis_tid_width => 8,
94        c_axis_tkeep_width => 4,
95        c_axis_tstrb_width => 4,
96        c_axis_tuser_width => 4,
97        c_axis_type => 0,
98        c_common_clock => 0,
99        c_count_type => 0,
100       c_data_count_width => 11,
101       c_default_value => "BlankString",
102       c_din_width => 32,
103       c_din_width_axis => 1,
104       c_din_width_rach => 32,
105       c_din_width_rdch => 64,
106       c_din_width_wach => 32,
107       c_din_width_wdch => 64,
108       c_din_width_wrch => 2,
109       c_dout_rst_val => "0",
110       c_dout_width => 32,
111       c_enable_rlocs => 0,
112       c_enable_rst_sync => 1,
113       c_error_injection_type => 0,
114       c_error_injection_type_axis => 0,
115       c_error_injection_type_rach => 0,
116       c_error_injection_type_rdch => 0,
117       c_error_injection_type_wach => 0,
118       c_error_injection_type_wdch => 0,
119       c_error_injection_type_wrch => 0,
120       c_family => "virtex5",
121       c_full_flags_rst_val => 1,
122       c_has_almost_empty => 0,
123       c_has_almost_full => 0,
124       c_has_axi_aruser => 0,
125       c_has_axi_awuser => 0,
126       c_has_axi_buser => 0,
127       c_has_axi_rd_channel => 0,
128       c_has_axi_ruser => 0,
129       c_has_axi_wr_channel => 0,
130       c_has_axi_wuser => 0,
131       c_has_axis_tdata => 0,
132       c_has_axis_tdest => 0,
```

```
133         c_has_axis_tid => 0,
134         c_has_axis_tkeep => 0,
135         c_has_axis_tlast => 0,
136         c_has_axis_tready => 1,
137         c_has_axis_tstrb => 0,
138         c_has_axis_tuser => 0,
139         c_has_backup => 0,
140         c_has_data_count => 0,
141         c_has_data_counts_axis => 0,
142         c_has_data_counts_rach => 0,
143         c_has_data_counts_rdch => 0,
144         c_has_data_counts_wach => 0,
145         c_has_data_counts_wdch => 0,
146         c_has_data_counts_wrch => 0,
147         c_has_int_clk => 0,
148         c_has_master_ce => 0,
149         c_has_meminit_file => 0,
150         c_has_overflow => 0,
151         c_has_prog_flags_axis => 0,
152         c_has_prog_flags_rach => 0,
153         c_has_prog_flags_rdch => 0,
154         c_has_prog_flags_wach => 0,
155         c_has_prog_flags_wdch => 0,
156         c_has_prog_flags_wrch => 0,
157         c_has_rd_data_count => 0,
158         c_has_rd_rst => 0,
159         c_has_rst => 1,
160         c_has_slave_ce => 0,
161         c_has_srst => 0,
162         c_has_underflow => 0,
163         c_has_valid => 0,
164         c_has_wr_ack => 0,
165         c_has_wr_data_count => 0,
166         c_has_wr_rst => 0,
167         c_implementation_type => 2,
168         c_implementation_type_axis => 1,
169         c_implementation_type_rach => 1,
170         c_implementation_type_rdch => 1,
171         c_implementation_type_wach => 1,
172         c_implementation_type_wdch => 1,
173         c_implementation_type_wrch => 1,
174         c_init_wr_pntr_val => 0,
175         c_interface_type => 0,
176         c_memory_type => 1,
177         c_mif_file_name => "BlankString",
178         c_msgon_val => 1,
179         c_optimization_mode => 0,
180         c_overflow_low => 0,
181         c_preload_latency => 1,
182         c_preload_regs => 0,
183         c_prim_fifo_type => "2kx18",
184         c_prog_empty_thresh_assert_val => 2,
185         c_prog_empty_thresh_assert_val_axis => 1022,
186         c_prog_empty_thresh_assert_val_rach => 1022,
187         c_prog_empty_thresh_assert_val_rdch => 1022,
188         c_prog_empty_thresh_assert_val_wach => 1022,
189         c_prog_empty_thresh_assert_val_wdch => 1022,
190         c_prog_empty_thresh_assert_val_wrch => 1022,
191         c_prog_empty_thresh_negate_val => 3,
192         c_prog_empty_type => 0,
193         c_prog_empty_type_axis => 5,
194         c_prog_empty_type_rach => 5,
195         c_prog_empty_type_rdch => 5,
196         c_prog_empty_type_wach => 5,
197         c_prog_empty_type_wdch => 5,
198         c_prog_empty_type_wrch => 5,
199         c_prog_full_thresh_assert_val => 1534,
200         c_prog_full_thresh_assert_val_axis => 1023,
201         c_prog_full_thresh_assert_val_rach => 1023,
202         c_prog_full_thresh_assert_val_rdch => 1023,
203         c_prog_full_thresh_assert_val_wach => 1023,
204         c_prog_full_thresh_assert_val_wdch => 1023,
205         c_prog_full_thresh_assert_val_wrch => 1023,
206         c_prog_full_thresh_negate_val => 1533,
```

```
207         c_prog_full_type => 1,
208         c_prog_full_type_axis => 5,
209         c_prog_full_type_rach => 5,
210         c_prog_full_type_rdch => 5,
211         c_prog_full_type_wach => 5,
212         c_prog_full_type_wdch => 5,
213         c_prog_full_type_wrch => 5,
214         c_rach_type => 0,
215         c_rd_data_count_width => 11,
216         c_rd_depth => 2048,
217         c_rd_freq => 1,
218         c_rd_pntr_width => 11,
219         c_rdch_type => 0,
220         c_reg_slice_mode_axis => 0,
221         c_reg_slice_mode_rach => 0,
222         c_reg_slice_mode_rdch => 0,
223         c_reg_slice_mode_wach => 0,
224         c_reg_slice_mode_wdch => 0,
225         c_reg_slice_mode_wrch => 0,
226         c_synchronizer_stage => 2,
227         c_underflow_low => 0,
228         c_use_common_overflow => 0,
229         c_use_common_underflow => 0,
230         c_use_default_settings => 0,
231         c_use_dout_rst => 1,
232         c_use_ecc => 0,
233         c_use_ecc_axis => 0,
234         c_use_ecc_rach => 0,
235         c_use_ecc_rdch => 0,
236         c_use_ecc_wach => 0,
237         c_use_ecc_wdch => 0,
238         c_use_ecc_wrch => 0,
239         c_use_embedded_reg => 0,
240         c_use_fifo16_flags => 0,
241         c_use_fwft_data_count => 0,
242         c_valid_low => 0,
243         c_wach_type => 0,
244         c_wdch_type => 0,
245         c_wr_ack_low => 0,
246         c_wr_data_count_width => 11,
247         c_wr_depth => 2048,
248         c_wr_depth_axis => 1024,
249         c_wr_depth_rach => 16,
250         c_wr_depth_rdch => 1024,
251         c_wr_depth_wach => 16,
252         c_wr_depth_wdch => 1024,
253         c_wr_depth_wrch => 16,
254         c_wr_freq => 1,
255         c_wr_pntr_width => 11,
256         c_wr_pntr_width_axis => 10,
257         c_wr_pntr_width_rach => 4,
258         c_wr_pntr_width_rdch => 10,
259         c_wr_pntr_width_wach => 4,
260         c_wr_pntr_width_wdch => 10,
261         c_wr_pntr_width_wrch => 4,
262         c_wr_response_latency => 1,
263         c_wrch_type => 0
264       );
265   -- synthesis translate_on
266   BEGIN
267   -- synthesis translate_off
268   U0 : wrapped_FIFO
269     PORT MAP (
270       rst => rst,
271       wr_clk => wr_clk,
272       rd_clk => rd_clk,
273       din => din,
274       wr_en => wr_en,
275       rd_en => rd_en,
276       dout => dout,
277       full => full,
278       empty => empty,
279       prog_full => prog_full
280     );
```

```
281  -- synthesis translate_on
282
283  END FIFO_a;
```

FSM.vhd

```
1   ----------------------------------------------------------------------------------------
2
3   --FSM.vhd
4
5   ----------------------------------------------------------------------------------------
6
7
8   library ieee ;
9   use ieee.std_logic_1164.all;
10
11  entity FSM is
12  port( intEmpty:     in std_logic;
13      extFull:      in std_logic;
14      clock:      in std_logic;
15      reset_n:    in std_logic;
16      wr_en:      out std_logic;
17      rd_en:      out std_logic);
18  end FSM;
19
20  architecture Behavioral of FSM is
21
22      type state_type is (S0, S1, S2, S3);
23      signal next_state, current_state: state_type;
24
25  begin
26
27      state_reg: process(clock, reset_n)
28      begin
29      if (reset_n='0') then
30          current_state <= S0;
31      elsif (clock'event and clock='1') then
32        current_state <= next_state;
33      end if;
34      end process;
35
36      comb_logic: process(current_state, intEmpty, extFull)
37      begin
38      case current_state is
39        when S0 =>  rd_en <= '0';
40              wr_en <='0';
41          if (intEmpty='0') then
42            next_state <= S1;
43          else
44            next_state <= S0;
45          end if;
46
47        when S1 =>  rd_en <= '1';
48              wr_en <='0';
49            next_state <= S2;
50
51        when S2 =>  rd_en <= '0';
52              wr_en <='0';
53          if (extFull='0') then
54            next_state <= S3;
55          else
56            next_state <= S2;
57          end if;
58
59        when S3 =>  rd_en <= '0';
60              wr_en <= '1';
61          if (intEmpty='0') then
62            next_state <= S1;
63          else
64            next_state <= S0;
65          end if;
66
67        when others =>
```

```
68          rd_en <= '0';
69          wr_en <='0';
70          next_state <= S0;
71      end case;
72      end process;
73  end Behavioral;
```

Deserializzatore.vhd

```
1   ----------------------------------------------------------------------------------------
2
3   --Deserializzatore.vhd
4
5   ----------------------------------------------------------------------------------------
6
7   library IEEE;
8   use IEEE.STD_LOGIC_1164.ALL;
9   use IEEE.NUMERIC_STD.ALL;
10
11
12  entity Deserializzatore is
13  generic ( width : integer :=16);
14  Port (    reset_n : in  STD_LOGIC;
15        clk : in STD_LOGIC;
16        cs_n : in STD_LOGIC;
17        datain : in  STD_LOGIC;
18        ready : out STD_LOGIC;
19        dataout : out  STD_LOGIC_VECTOR (width-1 downto 0));
20  end Deserializzatore;
21
22  architecture Behavioral of Deserializzatore is
23
24  type state_type is (S0, S1, S2, S3);
25  signal current_state: state_type;
26
27  signal clkCounterInternal : unsigned (4 downto 0);
28  constant maxClkCounterInternal : unsigned (4 downto 0):= (4=>'0', others=>'1');
29  signal outputInternal : unsigned (width-1 downto 0);
30
31  begin
32
33  comb_log: process(clk, reset_n)
34      begin
35      if (reset_n='0') then
36        current_state <= S0;
37        clkCounterInternal <= (others=>'0');
38        outputInternal <= (others=>'0');
39
40      elsif (clk'event and clk='1') then
41
42        if (current_state = S0) then
43            clkCounterInternal <= (others=>'0');
44            outputInternal <= (others=>'0');
45            current_state <= S1;
46        elsif (current_state = S1 and cs_n='0') then
47          if (clkCounterInternal + 1 = maxClkCounterInternal) then
48            current_state <= S2;
49          elsif (datain = '1') then
50            outputInternal <= outputInternal+ (2**(width-3-TO_INTEGER(clkCounterInternal)));
51            clkCounterInternal <= clkCounterInternal + 1;
52          else
53            clkCounterInternal <= clkCounterInternal + 1;
54          end if;
55        elsif (current_state = S2) then
56          current_state <= S3;
57        elsif (current_state = S3) then
58          if (cs_n='1') then
59            clkCounterInternal <= (others=>'0');
60            outputInternal <= (others=>'0');
61            current_state <= S1;
62          end if;
63        end if;
64      end if;
```

```
65      end process;
66
67
68  output: process(current_state)
69      begin
70      case current_state is
71        when S0 =>  ready <= '0';
72              dataout <= (others=>'0');
73
74        when S2 =>  ready <= '1';
75              dataout <= std_logic_vector(outputInternal);
76
77        when others =>  ready <= '0';
78                dataout <= std_logic_vector(outputInternal);
79
80      end case;
81      end process;
82
83  end Behavioral;
```

ClockDivider.vhd

```
1   ----------------------------------------------------------------------------------------
2
3   --ClockDivider.vhd
4
5   ----------------------------------------------------------------------------------------
6
7   library IEEE;
8   use IEEE.STD_LOGIC_1164.ALL;
9   use IEEE.NUMERIC_STD.ALL;
10
11
12  entity ClockDivider is
13    generic (   lengthDivisor   : integer :=12);
14    Port (  reset_n : in std_logic;
15        clkIn : in std_logic;
16        divisor : in std_logic_vector (lengthDivisor-1 downto 0);
17        clkOut : out std_logic);
18  end ClockDivider;
19
20  architecture Behavioral of ClockDivider is
21
22    signal clockState : std_logic:='0';
23    signal n : unsigned (lengthDivisor-1 downto 0);
24
25
26  begin
27    process(clkIn, reset_n)
28
29    begin
30      if(reset_n='0') then
31        n(0)<='1';
32        n(lengthDivisor-1 downto 1)<=(others=>'0');
33        clockState<='1';
34
35      elsif(rising_edge(clkIn)) then
36
37        if(n=('0' & unsigned(divisor(lengthDivisor-1 downto 1)))) then
38          clockState<='0';
39        elsif (n=unsigned(divisor)) then
40          clockState<='1';
41        end if;
42
43        if (n=unsigned(divisor)) then
44           n(0)<='1';
45           n(lengthDivisor-1 downto 1)<=(others=>'0');
46        else
47             n<=unsigned(n)+1;
48        end if;
49
50      end if;
51    end process;
```

```
52
53      clkOut <= clockState;
54
55  end Behavioral;
```

LogicControlUnit.vhd

```
1   ----------------------------------------------------------------------------
2
3   --LogicControlUnit.vhd
4
5   ----------------------------------------------------------------------------
6
7   library IEEE;
8   use IEEE.STD_LOGIC_1164.ALL;
9   use IEEE.NUMERIC_STD.ALL;
10
11  entity LogicControlUnit is
12      generic (   width : integer;
13            dataLength : integer;
14            channelLength : integer;
15            integrationTimeBitMax : integer);
16    Port (  reset_n : in std_logic;
17        extTrigger : in std_logic;
18        clk : in std_logic;
19        clkAdc : in std_logic;
20        clkHam : in std_logic;
21        Test : in std_logic;
22        Ham_Eos : in std_logic;
23        trigger : in std_logic;
24        dataIn : in std_logic_vector(dataLength-1 downto 0);
25        dataReadyFromDeserializer : in std_logic;
26        integTime : in std_logic_vector(integrationTimeBitMax-1 downto 0);
27        triggerDaInvRst_n : in std_logic;
28        extTrigOut : out std_logic_vector(width-2 downto 0);
29        triggerDaInv : out std_logic;
30        adcStart : out std_logic;
31        photodiodeStart : out std_logic;
32        deserializerStart : out std_logic;
33        writeEnable : out std_logic;
34        adcPDEN : out std_logic;
35        wordOut : out std_logic_vector(width-1 downto 0));
36  end LogicControlUnit;
37
38  architecture Behavioral of LogicControlUnit is
39
40  ------------------------------------Component Declaration-------------------------
41
42  component TriggerAnalyzer
43    Port (  clk : in std_logic;
44        reset_n : in std_logic;
45        trigger : in std_logic;
46        rising : out std_logic;
47        falling : out std_logic);
48    end component;
49
50  component ClkAnalyzer
51    Port (  clk : in std_logic;
52        reset_n : in std_logic;
53        clkAdc : in std_logic;
54        rising : out std_logic);
55    end component;
56
57  ------------------------------------------------------------------------------------
58
59  type state_type is (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12);
60  signal current_state: state_type;
61
62  signal extTrigOutInternal : unsigned (width-2 downto 0);
63  signal photodiodeStartCounterInternal : unsigned (integrationTimeBitMax-1 downto 0);
64  signal adcStartCounterInternal : unsigned (5 downto 0);
65  signal channelCounter : unsigned (channelLength-1 downto 0);
66
```

```
67  constant maxExtTrigOutInternal : unsigned (width-2 downto 0) := (others=>'1');
68  constant intermediateAdcStartCounterInternal : unsigned (5 downto 0) := (4=>'1', others=>'0');
69  constant maxAdcStartCounterInternal : unsigned (5 downto 0) := (5=>'1', others=>'0');
70  constant maxChannelCounterInternal : unsigned (channelLength-1 downto 0) := (others=>'1');
71
72  --signal adcPDENInternal : std_logic;
73
74  signal triggerDaInvInternal : std_logic;
75  signal trigRising : std_logic;
76  signal trigFalling : std_logic;
77  signal extTrigRising : std_logic;
78  signal adcRising : std_logic;
79  signal hamRising : std_logic;
80  signal integTimeReached : std_logic;
81  signal intermediateAdcStartCounterReached : std_logic;
82  signal maxAdcStartCounterReached : std_logic;
83  signal maxChannelCounterReached : std_logic;
84  signal maxExtTrigOutReached : std_logic;
85
86
87  begin
88
89  TriggerAnalyzerlogic: TriggerAnalyzer port map( clk=>clk,
90                          reset_n=>reset_n,
91                          trigger=>trigger,
92                          rising=>trigRising,
93                          falling=>trigFalling);
94
95  ExtTriggerAnalyzerlogic: ClkAnalyzer port map(  clk=>clk,
96                          reset_n=>reset_n,
97                          clkAdc=>extTrigger,
98                          rising=>extTrigRising);
99
100 AdcClkAnalyzerlogic : ClkAnalyzer port map( clk=>clk,
101                       reset_n=>reset_n,
102                       clkAdc=>clkAdc,
103                       rising=>adcRising);
104
105 HamClkAnalyzerlogic : ClkAnalyzer port map( clk=>clk,
106                       reset_n=>reset_n,
107                       clkAdc=>clkHam,
108                       rising=>hamRising);
109
110 integTimeReached_setting : process (clk)
111 begin
112   if (clk'event and clk='1') then
113     if(std_logic_vector(photodiodeStartCounterInternal - 1) = integTime) then
114       integTimeReached<='1';
115     else
116       integTimeReached<='0';
117     end if;
118   end if;
119 end process;
120
121 intermediateAdcStartCounterReached_setting : process (clk)
122 begin
123   if (clk'event and clk='1') then
124     if(adcStartCounterInternal = intermediateAdcStartCounterInternal) then
125       intermediateAdcStartCounterReached<='1';
126     else
127       intermediateAdcStartCounterReached<='0';
128     end if;
129   end if;
130 end process;
131
132 maxAdcStartCounterReached_setting : process (clk)
133 begin
134   if (clk'event and clk='1') then
135     if(adcStartCounterInternal = maxAdcStartCounterInternal) then
136       maxAdcStartCounterReached<='1';
137     else
138       maxAdcStartCounterReached<='0';
139     end if;
140   end if;
```

```
141 end process;
142
143 maxChannelCounterReached_setting : process (clk)
144 begin
145   if (clk'event and clk='1') then
146     if(channelCounter = maxChannelCounterInternal) then
147       maxChannelCounterReached<='1';
148     else
149       maxChannelCounterReached<='0';
150     end if;
151   end if;
152 end process;
153
154 maxExtTrigOutReached_setting : process (clk)
155 begin
156   if (clk'event and clk='1') then
157     if(extTrigOutInternal = maxExtTrigOutInternal) then
158       maxExtTrigOutReached<='1';
159     else
160       maxExtTrigOutReached<='0';
161     end if;
162   end if;
163 end process;
164
165
166 comb_log: process(clk, reset_n)
167     begin
168     if (reset_n='0') then
169       current_state <= S0;
170       extTrigOutInternal <= (others=>'0');
171       photodiodeStartCounterInternal <= (others=>'0');
172       adcStartCounterInternal <= (others=>'0');
173       channelCounter <= (others=>'1');
174       --adcPDENInternal <= '0';
175       triggerDaInvInternal <='0';
176
177     elsif (clk'event and clk='1') then
178
179       case current_state is
180         when S0 =>
181           if(triggerDaInvRst_n='0') then
182             triggerDaInvInternal <= '0';
183           else
184             triggerDaInvInternal <= '1';
185           end if;
186           current_state <= S1;
187
188         when S1 =>
189           if(extTrigRising='1') then
190             channelCounter <= (others=>'1');
191             --adcPDENInternal <= '0';
192             current_state <= S2;
193           end if;
194
195         when S2 =>
196           if (Test='1') then
197             current_state <= S10;
198           else
199             current_state <= S3;
200           end if;
201
202         when S3 =>
203           if (integTimeReached = '1') then
204             photodiodeStartCounterInternal <= (others=>'0');
205             current_state <= S4;
206           elsif(hamRising = '1') then
207             photodiodeStartCounterInternal<=photodiodeStartCounterInternal+1;
208           end if;
209
210         when S4 =>
211           if (trigRising='1') then
212             current_state <= S5;
213           end if;
214
```

```
215           when S5 =>
216             if (intermediateAdcStartCounterReached = '1') then
217               if(maxChannelCounterReached = '1') then
218                 channelCounter <= (others=>'0');
219               else
220                 channelCounter <= channelCounter + 1;
221               end if;
222               current_state <= S6;
223             elsif(adcRising = '1') then
224               adcStartCounterInternal <= adcStartCounterInternal+1;
225             end if;
226
227           when S6 =>
228             if (maxAdcStartCounterReached = '1') then
229               adcStartCounterInternal <= (others=>'0');
230               current_state <= S7;
231             elsif(adcRising = '1') then
232               adcStartCounterInternal <= adcStartCounterInternal+1;
233             end if;
234
235           when S7 => current_state <= S8;
236
237           when S8 =>
238             if (trigFalling ='1') then
239               --if (Ham_Eos = '0') then
240                 --adcPDENInternal <= '0';
241               --else
242                 --adcPDENInternal <= '0';
243               --end if;
244               current_state <= S9;
245             end if;
246
247           when S9 =>
248             if (intermediateAdcStartCounterReached = '1') then
249               current_state <= S10;
250             elsif(adcRising = '1') then
251               adcStartCounterInternal <= adcStartCounterInternal+1;
252             end if;
253
254           when S10 =>
255             if (maxAdcStartCounterReached = '1') then
256               adcStartCounterInternal <= (others=>'0');
257               current_state <= S11;
258             elsif(adcRising = '1') then
259               adcStartCounterInternal <= adcStartCounterInternal+1;
260             end if;
261
262           when S11 =>
263             if (Ham_Eos = '0') then
264               current_state <= S12;
265             else
266               current_state <= S4;
267             end if;
268
269           when S12 =>
270             if(maxExtTrigOutReached = '1') then
271               extTrigOutInternal <= (others=>'0');
272             else
273               extTrigOutInternal<=(extTrigOutInternal+1);
274             end if;
275
276             if(triggerDaInvRst_n='0') then
277               triggerDaInvInternal<='0';
278             else
279               triggerDaInvInternal<='1';
280             end if;
281
282             current_state <= S1;
283
284           when others => current_state <= S0;
285
286         end case;
287
288       end if;
```

```
289     end process;
290
291
292 output: process(current_state)
293     begin
294     case current_state is
295       when S0 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
296             triggerDaInv <= triggerDaInvInternal;
297             adcStart <= '1';
298             photodiodeStart <= '0';
299             deserializerStart <= '1';
300             writeEnable <= '0';
301             adcPDEN <= '0';
302             wordOut <= (others=>'0');
303
304       when S1 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
305             triggerDaInv <= triggerDaInvInternal;
306             adcStart <= '1';
307             photodiodeStart <= '0';
308             deserializerStart <= '1';
309             writeEnable <= '0';
310             adcPDEN <= '0';
311             wordOut <= (others=>'0');
312
313       when S2 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
314             triggerDaInv <= triggerDaInvInternal;
315             adcStart <= '1';
316             photodiodeStart <= '1';
317             deserializerStart <= '1';
318             writeEnable <= '1';
319             adcPDEN <= '0';
320             wordOut <= '1' & std_logic_vector(extTrigOutInternal);
321
322       when S3 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
323             triggerDaInv <= triggerDaInvInternal;
324             adcStart <= '1';
325             photodiodeStart <= '1';
326             deserializerStart <= '1';
327             writeEnable <= '0';
328             adcPDEN <= '0';
329             wordOut <= '1' & std_logic_vector(extTrigOutInternal);
330
331       when S4 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
332             triggerDaInv <= triggerDaInvInternal;
333             adcStart <= '1';
334             photodiodeStart <= '0';
335             deserializerStart <= '1';
336             writeEnable <= '0';
337             adcPDEN <= '0';
338             wordOut <= '1' & std_logic_vector(extTrigOutInternal);
339
340       when S5 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
341             triggerDaInv <= triggerDaInvInternal;
342             adcStart <= '0';
343             photodiodeStart <= '0';
344             deserializerStart <= '1';
345             writeEnable <= '0';
346             adcPDEN <= '0';
347             wordOut <= '1' & std_logic_vector(extTrigOutInternal);
348
349       when S6 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
350             triggerDaInv <= triggerDaInvInternal;
351             adcStart <= '0';
352             photodiodeStart <= '0';
353             deserializerStart <= '0';
354             writeEnable <= '0';
355             adcPDEN <= '0';
356             wordOut <= '1' & std_logic_vector(extTrigOutInternal);
357
358       when S7 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
359             triggerDaInv <= triggerDaInvInternal;
360             adcStart <= '1';
361             photodiodeStart <= '0';
362             deserializerStart <= '1';
```

```
363             writeEnable <= '1';
364             adcPDEN <= '0';
365             wordOut <= '0'&"000000"&std_logic_vector(channelCounter) &'0'& dataIn;
366
367       when S8 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
368             triggerDaInv <= triggerDaInvInternal;
369             adcStart <= '1';
370             photodiodeStart <= '0';
371             deserializerStart <= '1';
372             writeEnable <= '0';
373             adcPDEN <= '0';
374             wordOut <= '0'&"000000"&std_logic_vector(channelCounter) &'0'& dataIn;
375
376       when S9 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
377             triggerDaInv <= triggerDaInvInternal;
378             adcStart <= '0';
379             photodiodeStart <= '0';
380             deserializerStart <= '1';
381             writeEnable <= '0';
382             adcPDEN <= '0';
383             wordOut <= '0'&"000000"&std_logic_vector(channelCounter) &'0'& dataIn;
384
385       when S10 =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
386             triggerDaInv <= triggerDaInvInternal;
387             adcStart <= '0';
388             photodiodeStart <= '0';
389             deserializerStart <= '0';
390             writeEnable <= '0';
391             adcPDEN <= '0';
392             wordOut <= '0'&"000000"&std_logic_vector(channelCounter) &'0'& dataIn;
393
394
395       when S11 => extTrigOut <= std_logic_vector(extTrigOutInternal);
396             triggerDaInv <= triggerDaInvInternal;
397             adcStart <= '1';
398             photodiodeStart <= '0';
399             deserializerStart <= '1';
400             writeEnable <= '1';
401             adcPDEN <= '0';
402             wordOut <= '0'&"000000"&std_logic_vector(channelCounter) &'1'& dataIn;
403
404       when S12 => extTrigOut <= std_logic_vector(extTrigOutInternal);
405             triggerDaInv <= triggerDaInvInternal;
406             adcStart <= '1';
407             photodiodeStart <= '0';
408             deserializerStart <= '1';
409             writeEnable <= '1';
410             adcPDEN <= '0';
411             wordOut<="01000000000000000110111101010110 1";
412
413       when others =>  extTrigOut <= std_logic_vector(extTrigOutInternal);
414               triggerDaInv <= triggerDaInvInternal;
415               adcStart <= '1';
416               photodiodeStart <= '0';
417               deserializerStart <= '1';
418               writeEnable <= '0';
419               adcPDEN <= '0';
420               wordOut <= (others=>'0');
421
422     end case;
423     end process;
424
425 end Behavioral;
```

AdcClkAnalyzer.vhd

```
1  ----------------------------------------------------------------------------
2
3  --AdcClkAnalyzer.vhd
4
5  ----------------------------------------------------------------------------
6
7  library IEEE;
```

```
8  use IEEE.STD_LOGIC_1164.ALL;
9  use IEEE.NUMERIC_STD.ALL;
10
11 entity AdcClkAnalyzer is
12   Port (  clk : in std_logic;
13       clkAdc : in std_logic;
14       rising : out std_logic);
15   end AdcClkAnalyzer;
16
17 architecture Behavioral of AdcClkAnalyzer is
18
19 signal rised : std_logic :='0';
20 signal falled : std_logic :='1';
21
22 begin
23   process(clk, clkAdc)
24
25   begin
26     if(rising_edge(clk)) then
27       if(falled='1' and clkAdc='1' and rised='0') then
28         rising<='1';
29         rised<='1';
30         falled<='0';
31       elsif(rised='1' and clkAdc='0' and falled='0') then
32         falled<='1';
33         rising<='0';
34         rised<='0';
35       else
36         rising<='0';
37       end if;
38     end if;
39
40
41   end process;
42
43 end Behavioral;
```

TriggerAnalyzer.vhd

```
1  ----------------------------------------------------------------------------------
2
3  --TriggerAnalyzer.vhd
4
5  ----------------------------------------------------------------------------------
6
7  library IEEE;
8  use IEEE.STD_LOGIC_1164.ALL;
9  use IEEE.NUMERIC_STD.ALL;
10
11 entity TriggerAnalyzer is
12   Port (  clk : in std_logic;
13       reset_n : in std_logic;
14       trigger : in std_logic;
15       rising : out std_logic;
16       falling : out std_logic);
17 end TriggerAnalyzer;
18
19 architecture Behavioral of TriggerAnalyzer is
20
21 signal rised : std_logic;
22 signal falled : std_logic;
23
24 begin
25   process(clk)
26
27   begin
28     if (reset_n='0') then
29       rised<='0';
30       falled<='1';
31       falling<='0';
32       rising<='0';
33     elsif(rising_edge(clk)) then
34       if(falled='1' and trigger='1' and rised='0') then
```

```
35          rising<='1';
36          rised<='1';
37          falled<='0';
38          falling<='0';
39        elsif(rised='1' and trigger='0' and falled='0') then
40          falling<='1';
41          falled<='1';
42          rised<='0';
43          rising<='0';
44        else
45          rising<='0';
46          falling<='0';
47        end if;
48      end if;
49
50
51    end process;
52
53  end Behavioral;
```

A.4 Test Bench

TestAdcClkAnalyzer.vhd

```
1   --------------------------------------------------------------------------------
2
3   --TestAdcClkAnalyzer.vhd
4
5   --------------------------------------------------------------------------------
6   LIBRARY ieee;
7   USE ieee.std_logic_1164.ALL;
8
9   ENTITY TestAdcClkAnalyzer IS
10  END TestAdcClkAnalyzer;
11
12  ARCHITECTURE behavior OF TestAdcClkAnalyzer IS
13
14
15      COMPONENT AdcClkAnalyzer
16      PORT(
17           clk : IN  std_logic;
18           clkAdc : IN  std_logic;
19           rising : OUT  std_logic
20          );
21      END COMPONENT;
22
23
24     signal clk : std_logic := '0';
25     signal clkAdc : std_logic := '0';
26     signal rising : std_logic;
27     constant clk_period : time := 10 ns;
28     constant clkAdc_period : time := 25 ns;
29
30  BEGIN
31
32     uut: AdcClkAnalyzer PORT MAP (
33            clk => clk,
34            clkAdc => clkAdc,
35            rising => rising
36          );
37
38     clk_process :process
39     begin
40      clk <= '0';
41      wait for clk_period/2;
42      clk <= '1';
43      wait for clk_period/2;
44     end process;
45
46     clkAdc_process :process
47     begin
```

```
48      clkAdc <= '0';
49      wait for clkAdc_period/2;
50      clkAdc <= '1';
51      wait for clkAdc_period/2;
52    end process;
53
54
55    stim_proc: process
56    begin
57       wait for 100 ns;
58
59       wait for clk_period*10;
60
61       wait;
62    end process;
63
64  END;
```

TestClockDivider.vhd

```
1   --------------------------------------------------------------------------------
2
3   --TestClock7Divider.vhd
4
5   --------------------------------------------------------------------------------
6   LIBRARY ieee;
7   USE ieee.std_logic_1164.ALL;
8
9   -- Uncomment the following library declaration if using
10  -- arithmetic functions with Signed or Unsigned values
11  --USE ieee.numeric_std.ALL;
12
13  ENTITY TestClockDivider IS
14  END TestClockDivider;
15
16  ARCHITECTURE behavior OF TestClockDivider IS
17
18      -- Component Declaration for the Unit Under Test (UUT)
19
20      COMPONENT ClockDivider
21      PORT(
22           reset_n : IN  std_logic;
23           clkIn : IN  std_logic;
24           divisor : IN  std_logic_vector(0 to 11);
25           clkOut : OUT  std_logic
26          );
27      END COMPONENT;
28
29
30     --Inputs
31     signal reset_n : std_logic := '0';
32     signal clkIn : std_logic := '0';
33     signal divisor : std_logic_vector(0 to 11) := (9=>'1', 11=>'1', others => '0');
34
35    --Outputs
36     signal clkOut : std_logic;
37
38     -- Clock period definitions
39     constant clkIn_period : time := 10 ns;
40     constant clkOut_period : time := 10 ns;
41
42  BEGIN
43
44     uut: ClockDivider PORT MAP (
45            reset_n => reset_n,
46            clkIn => clkIn,
47            divisor => divisor,
48            clkOut => clkOut
49          );
50
51     clkIn_process :process
52     begin
53      clkIn <= '0';
```

```
54      wait for clkIn_period/2;
55      clkIn <= '1';
56      wait for clkIn_period/2;
57     end process;
58
59
60
61     stim_proc: process
62     begin
63        wait for 100 ns;
64      reset_n<='1';
65        wait for clkIn_period*1000;
66          divisor(11)<='1';
67
68
69        wait;
70     end process;
71
72  END;
```

TestDeserializzatore.vhd

```
1   ----------------------------------------------------------------------------------
2
3   --TestDeserializzatore.vhd
4
5   ----------------------------------------------------------------------------------
6   LIBRARY ieee;
7   USE ieee.std_logic_1164.ALL;
8
9
10  ENTITY TestDeserializzatore IS
11  END TestDeserializzatore;
12
13  ARCHITECTURE behavior OF TestDeserializzatore IS
14
15
16      COMPONENT Deserializzatore
17      generic ( width : integer := 16);
18    Port (  reset_n : in  STD_LOGIC;
19         clk : in STD_LOGIC;
20         cs_n : in STD_LOGIC;
21         datain : in  STD_LOGIC;
22         ready : out STD_LOGIC;
23         dataout : out  STD_LOGIC_VECTOR (width-1 downto 0));
24      END COMPONENT;
25
26
27     signal reset_n : std_logic := '0';
28     signal clk : std_logic := '0';
29     signal cs_n : std_logic := '0';
30     signal datain : std_logic := '0';
31     signal ready : std_logic;
32     signal dataout : std_logic_vector(15 downto 0);
33
34     constant clk_period : time := 10 ns;
35
36  BEGIN
37
38     uut: Deserializzatore PORT MAP (
39            reset_n => reset_n,
40            clk => clk,
41            cs_n => cs_n,
42            datain => datain,
43            ready => ready,
44            dataout => dataout
45          );
46
47     clk_process :process
48     begin
49      clk <= '0';
50      wait for clk_period/2;
51      clk <= '1';
```

```
52      wait for clk_period/2;
53     end process;
54
55
56     stim_proc: process
57     begin
58        wait for clk_period/2;
59      reset_n<='0';
60      cs_n<='1';
61          wait for clk_period*100;
62      reset_n<='1';
63      wait for clk_period*3;
64      cs_n<='0';
65      datain<='1';
66      wait for clk_period*3;
67      datain<='0';
68      wait for clk_period*6;
69      datain<='1';
70      wait for clk_period*2;
71      datain<='0';
72      wait for 45 ns;
73      cs_n<='1';
74      wait for clk_period*3;
75      cs_n<='0';
76      datain<='1';
77      wait for clk_period*5;
78      datain<='0';
79      wait for clk_period*4;
80      datain<='1';
81      wait for clk_period*2;
82      datain<='0';
83      wait for clk_period*5;
84      cs_n<='1';
85        wait;
86     end process;
87
88  END;
```

TestFIFO.vhd

```
1   ----------------------------------------------------------------------------------
2
3   --TestFIFO.vhd
4
5   ----------------------------------------------------------------------------------
6   LIBRARY ieee;
7   USE ieee.std_logic_1164.ALL;
8
9
10  ENTITY TestFIFO IS
11  END TestFIFO;
12
13  ARCHITECTURE behavior OF TestFIFO IS
14
15
16      COMPONENT intFIFO
17      PORT(
18           rst : IN  std_logic;
19           wr_clk : IN  std_logic;
20           rd_clk : IN  std_logic;
21           din : IN  std_logic_vector(31 downto 0);
22           wr_en : IN  std_logic;
23           rd_en : IN  std_logic;
24           dout : OUT  std_logic_vector(31 downto 0);
25           full : OUT  std_logic;
26           empty : OUT  std_logic;
27           prog_full : OUT  std_logic
28          );
29      END COMPONENT;
30
31
32     signal rst : std_logic := '0';
33     signal wr_clk : std_logic := '0';
```

```
34      signal rd_clk : std_logic := '0';
35      signal din : std_logic_vector(31 downto 0) := (others => '0');
36      signal wr_en : std_logic := '0';
37      signal rd_en : std_logic := '0';
38
39      signal dout : std_logic_vector(31 downto 0);
40      signal full : std_logic;
41      signal empty : std_logic;
42      signal prog_full : std_logic;
43
44      constant wr_clk_period : time := 10 ns;
45      constant rd_clk_period : time := 10 ns;
46
47  BEGIN
48
49      uut: intFIFO PORT MAP (
50             rst => rst,
51             wr_clk => wr_clk,
52             rd_clk => rd_clk,
53             din => din,
54             wr_en => wr_en,
55             rd_en => rd_en,
56             dout => dout,
57             full => full,
58             empty => empty,
59             prog_full => prog_full
60           );
61
62      wr_clk_process :process
63      begin
64       wr_clk <= '0';
65       wait for wr_clk_period/2;
66       wr_clk <= '1';
67       wait for wr_clk_period/2;
68      end process;
69
70      rd_clk_process :process
71      begin
72       rd_clk <= '0';
73       wait for rd_clk_period/2;
74       rd_clk <= '1';
75       wait for rd_clk_period/2;
76      end process;
77
78
79      stim_proc: process
80      begin
81           rst<='1';
82           wait for 100 ns;
83         rst<='0';
84           wait for wr_clk_period*10;
85         wr_en<='1';
86       wait for wr_clk_period*2;
87         wr_en<='0';
88         din<=(5=>'1', 2=>'1', others => '0');
89       wait for wr_clk_period*10;
90         wr_en<='1';
91       wait for wr_clk_period;
92         wr_en<='0';
93       wait for wr_clk_period*10;
94         wr_en<='1';
95       wait for wr_clk_period;
96         wr_en<='0';
97       wait for wr_clk_period*10;
98         wr_en<='1';
99       wait for wr_clk_period;
100        wr_en<='0';
101      wait for wr_clk_period*10;
102        rd_en<='1';
103      wait for wr_clk_period;
104        rd_en<='0';
105      wait for wr_clk_period*10;
106        rd_en<='1';
107      wait for wr_clk_period;
```

```
108       rd_en<='0';
109     wait for wr_clk_period*10;
110       rd_en<='1';
111     wait for wr_clk_period;
112       rd_en<='0';
113       wait;
114    end process;
115 
116 END;
```

TestFPGA.vhd

```
1   ----------------------------------------------------------------------------
2   
3   --TestFPGA.vhd
4   
5   ----------------------------------------------------------------------------
6   
7   LIBRARY ieee;
8   USE ieee.std_logic_1164.ALL;
9   
10  
11  ENTITY TestFPGA IS
12  END TestFPGA;
13  
14  ARCHITECTURE behavior OF TestFPGA IS
15  
16  
17      COMPONENT FPGA
18      PORT(
19           rst_n : IN  std_logic;
20           trigger : IN  std_logic;
21           FPGA_Clk : IN  std_logic;
22           FSM_Clk : IN  std_logic;
23           extTrigger : IN  std_logic;
24           deseInput : IN  std_logic;
25           extFIFOFull : IN  std_logic;
26           wordOut : OUT  std_logic_vector(31 downto 0);
27           photodiodeClkOut : OUT  std_logic;
28           photodiodeStart : OUT  std_logic;
29           intFIFOFull : OUT  std_logic;
30           busy : OUT  std_logic;
31           ADCStart : OUT  std_logic
32          );
33      END COMPONENT;
34  
35  
36     signal rst_n : std_logic := '0';
37     signal trigger : std_logic := '0';
38     signal FPGA_Clk : std_logic := '0';
39     signal ADC_Clk : std_logic := '0';
40     signal FSM_Clk : std_logic := '0';
41     signal extTrigger : std_logic := '0';
42     signal deseInput : std_logic := '0';
43     signal extFIFOFull : std_logic := '0';
44     signal wordOut : std_logic_vector(31 downto 0);
45     signal photodiodeClkOut : std_logic;
46     signal photodiodeStart : std_logic;
47     signal intFIFOFull : std_logic;
48     signal busy : std_logic;
49     signal ADCStart : std_logic;
50     constant FPGA_Clk_period : time := 10 ns;
51     constant FSM_Clk_period : time := 8 ns;
52     constant ADC_Clk_period : time := 25 ns;
53  
54  BEGIN
55  
56     uut: FPGA PORT MAP (
57            rst_n => rst_n,
58            trigger => trigger,
59            FPGA_Clk => FPGA_Clk,
60            ADC_Clk => ADC_Clk,
61            FSM_Clk => FSM_Clk,
```

```
62            extTrigger => extTrigger,
63            deseInput => deseInput,
64            extFIFOFull => extFIFOFull,
65            wordOut => wordOut,
66            photodiodeClkOut => photodiodeClkOut,
67            photodiodeStart => photodiodeStart,
68            intFIFOFull => intFIFOFull,
69            busy => busy,
70            ADCStart => ADCStart);
71
72     FPGA_Clk_process :process
73     begin
74      FPGA_Clk <= '0';
75      wait for FPGA_Clk_period/2;
76      FPGA_Clk <= '1';
77      wait for FPGA_Clk_period/2;
78     end process;
79
80     ADC_Clk_process :process
81     begin
82      ADC_Clk <= '0';
83      wait for ADC_Clk_period/2;
84      ADC_Clk <= '1';
85      wait for ADC_Clk_period/2;
86     end process;
87
88     FSM_Clk_process :process
89     begin
90      FSM_Clk <= '0';
91      wait for FSM_Clk_period/2;
92      FSM_Clk <= '1';
93      wait for FSM_Clk_period/2;
94     end process;
95
96
97     stim_proc: process
98     begin
99        rst_n<='0';
100       wait for 100 ns;
101         rst_n<='1';
102       wait for FPGA_Clk_period*100;
103    extTrigger<='1';
104    wait for FPGA_Clk_period*10;
105      extTrigger<='0';
106    wait for FPGA_Clk_period*500;
107      Trigger<='1';
108    wait for FPGA_Clk_period;
109      deseInput<='1';
110    wait for FPGA_Clk_period*3;
111      deseInput<='0';
112    wait for FPGA_Clk_period*46;
113      Trigger<='0';
114    wait for FPGA_Clk_period*500;
115      Trigger<='1';
116    wait for FPGA_Clk_period;
117      deseInput<='1';
118    wait for FPGA_Clk_period*3;
119      deseInput<='0';
120    wait for FPGA_Clk_period*46;
121      Trigger<='0';
122
123       wait;
124    end process;
125
126 END;
```

TestFSM.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY TestFSM IS
5  END TestFSM;
```

```
6
7   ARCHITECTURE behavior OF TestFSM IS
8
9
10      COMPONENT FSM
11      PORT(
12           intEmpty : IN  std_logic;
13           extFull : IN  std_logic;
14           clock : IN  std_logic;
15           reset_n : IN  std_logic;
16           wr_en : OUT  std_logic;
17           rd_en : OUT  std_logic
18          );
19      END COMPONENT;
20
21
22     signal intEmpty : std_logic := '1';
23     signal extFull : std_logic := '0';
24     signal clock : std_logic := '0';
25     signal reset_n : std_logic := '0';
26
27    --Outputs
28     signal wr_en : std_logic;
29     signal rd_en : std_logic;
30
31     -- Clock period definitions
32     constant clock_period : time := 10 ns;
33
34  BEGIN
35
36    -- Instantiate the Unit Under Test (UUT)
37     uut: FSM PORT MAP (
38            intEmpty => intEmpty,
39            extFull => extFull,
40            clock => clock,
41            reset_n => reset_n,
42            wr_en => wr_en,
43            rd_en => rd_en
44          );
45
46     -- Clock process definitions
47     clock_process :process
48     begin
49      clock <= '0';
50      wait for clock_period/2;
51      clock <= '1';
52      wait for clock_period/2;
53     end process;
54
55
56     -- Stimulus process
57     stim_proc: process
58     begin
59        -- hold reset state for 100 ns.
60        wait for 100 ns;
61      reset_n<='1';
62        wait for clock_period*10;
63      intEmpty<='0';
64      wait for clock_period*20;
65        extFull<='1';
66      wait for clock_period*20;
67        extFull<='0';
68      wait for clock_period*20;
69      intEmpty<='1';
70      wait for clock_period*20;
71        extFull<='1';
72
73        wait;
74     end process;
75
76  END;
```

TestTriggerAnalyzer.vhd

```
1  ----------------------------------------------------------------------------
2
3  --TestTriggerAnalyzer.vhd
4
5  ----------------------------------------------------------------------------
6  LIBRARY ieee;
7  USE ieee.std_logic_1164.ALL;
8
9
10 ENTITY TestTriggerAnalyzer IS
11 END TestTriggerAnalyzer;
12
13 ARCHITECTURE behavior OF TestTriggerAnalyzer IS
14
15     COMPONENT TriggerAnalyzer
16     Port (  clk : in std_logic;
17         trigger : in std_logic;
18       rising : out std_logic;
19       falling : out std_logic);
20     END COMPONENT;
21
22
23    signal trigger : std_logic := '0';
24    signal rising : std_logic;
25    signal clk : std_logic :='0';
26    signal falling : std_logic;
27   constant clk_period : time := 10 ns;
28 BEGIN
29
30    clk_process :process
31    begin
32     clk <= '0';
33     wait for clk_period/2;
34     clk <= '1';
35     wait for clk_period/2;
36    end process;
37
38    uut: TriggerAnalyzer PORT MAP (
39         clk=>clk,
40           trigger => trigger,
41           rising => rising,
42           falling => falling
43         );
44
45    stim_proc: process
46    begin
47       wait for 100 ns;
48     trigger<='1';
49       wait for 50 ns;
50     trigger<='0';
51     wait for 100 ns;
52     trigger<='1';
53       wait for 50 ns;
54     trigger<='0';
55
56       wait;
57    end process;
58
59 END;
```

TestUserlogic.vhd

```
1  ----------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:   16:07:29 06/26/2012
6  -- Design Name:
7  -- Module Name:   /Network/Servers/elegen04.roma1.infn.it/Users/guglielmogemignani/IseProject/TestUser_Logic.vhd
8  -- Project Name:  User_Logic_2.0
9  -- Target Device:
10 -- Tool versions:
11 -- Description:
```

```
12  --
13  -- VHDL Test Bench Created by ISE for module: user_logic
14  --
15  -- Dependencies:
16  --
17  -- Revision:
18  -- Revision 0.01 - File Created
19  -- Additional Comments:
20  --
21  -- Notes:
22  -- This testbench has been automatically generated using types std_logic and
23  -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
24  -- that these types always be used for the top-level I/O of a design in order
25  -- to guarantee that the testbench will bind correctly to the post-implementation
26  -- simulation model.
27  --------------------------------------------------------------------------------
28  library ieee;
29  use ieee.std_logic_1164.all;
30  use ieee.std_logic_arith.all;
31  use ieee.std_logic_unsigned.all;
32
33  library proc_common_v3_00_a;
34  use proc_common_v3_00_a.proc_common_pkg.all;
35
36  -- Uncomment the following library declaration if using
37  -- arithmetic functions with Signed or Unsigned values
38  --USE ieee.numeric_std.ALL;
39
40  ENTITY TestUser_Logic IS
41  END TestUser_Logic;
42
43  ARCHITECTURE behavior OF TestUser_Logic IS
44
45      -- Component Declaration for the Unit Under Test (UUT)
46
47      COMPONENT user_logic
48      PORT(
49           Ham_Eos : IN  std_logic;
50           Ham_Trigger : IN  std_logic;
51           Ham_ClkIn : IN  std_logic;
52           Ham_ClkOut : OUT  std_logic;
53           Ham_Reset_n : OUT  std_logic;
54           Ham_Gain : OUT  std_logic;
55           Ham_InStart : OUT  std_logic;
56           Adc_SDO : IN  std_logic;
57           Adc_ClkOut : OUT  std_logic;
58           Adc_CS_n : OUT  std_logic;
59           Adc_PDEn : OUT  std_logic;
60           Adc_ChSel : OUT  std_logic;
61           Gen_Clk : IN  std_logic;
62           Gen_L0_p : IN  std_logic;
63           Gen_L0_n : IN  std_logic;
64           Gen_L1_p : IN  std_logic;
65           Gen_L1_n : IN  std_logic;
66           Gen_L2_p : OUT  std_logic;
67           Gen_L2_n : OUT  std_logic;
68           Gen_L3_p : OUT  std_logic;
69           Gen_L3_n : OUT  std_logic;
70           Gen_C0 : IN  std_logic;
71           Gen_C1 : IN  std_logic;
72           Gen_C2 : OUT  std_logic;
73           Gen_C3 : OUT  std_logic;
74           Gen_ClkOut_p : OUT  std_logic;
75           Gen_ClkOut_n : OUT  std_logic;
76           Bus2IP_Clk : IN  std_logic;
77           Bus2IP_Reset : IN  std_logic;
78           Bus2IP_Data : IN  std_logic_vector(0 to 31);
79           Bus2IP_BE : IN  std_logic_vector(0 to 3);
80           Bus2IP_RdCE : IN  std_logic_vector(0 to 7);
81           Bus2IP_WrCE : IN  std_logic_vector(0 to 7);
82           IP2Bus_Data : OUT  std_logic_vector(0 to 31);
83           IP2Bus_RdAck : OUT  std_logic;
84           IP2Bus_WrAck : OUT  std_logic;
85           IP2Bus_Error : OUT  std_logic;
```

```
86          IP2RFIFO_WrReq : OUT  std_logic;
87          IP2RFIFO_Data : OUT  std_logic_vector(0 to 31);
88          RFIFO2IP_WrAck : IN  std_logic;
89          RFIFO2IP_AlmostFull : IN  std_logic;
90          RFIFO2IP_Full : IN  std_logic;
91       RFIFO2IP_Vacancy : IN  std_logic_vector(0 to log2(16384))
92         );
93     END COMPONENT;
94
95
96    --Inputs
97    signal Ham_Eos : std_logic := '1';
98    signal Ham_Trigger : std_logic := '0';
99    signal Ham_ClkIn : std_logic := '0';
100   signal Adc_SDO : std_logic := '0';
101   signal Gen_Clk : std_logic := '0';
102   signal Gen_L0_p : std_logic := '0';
103   signal Gen_L0_n : std_logic := '0';
104   signal Gen_L1_p : std_logic := '0';
105   signal Gen_L1_n : std_logic := '0';
106   signal Gen_C0 : std_logic := '0';
107   signal Gen_C1 : std_logic := '0';
108   signal Bus2IP_Clk : std_logic := '0';
109   signal Bus2IP_Reset : std_logic := '0';
110   signal Bus2IP_Data : std_logic_vector(0 to 31) := (others => '0');
111   signal Bus2IP_BE : std_logic_vector(0 to 3) := (others => '0');
112   signal Bus2IP_RdCE : std_logic_vector(0 to 7) := (others => '0');
113   signal Bus2IP_WrCE : std_logic_vector(0 to 7) := (others => '0');
114   signal RFIFO2IP_WrAck : std_logic := '0';
115   signal RFIFO2IP_AlmostFull : std_logic := '0';
116   signal RFIFO2IP_Full : std_logic := '0';
117   signal RFIFO2IP_Vacancy : std_logic_vector(0 to log2(16384)) := (others => '0');
118
119   --Outputs
120   signal Ham_ClkOut : std_logic;
121   signal Ham_Reset_n : std_logic;
122   signal Ham_Gain : std_logic;
123   signal Ham_InStart : std_logic;
124   signal Adc_ClkOut : std_logic;
125   signal Adc_CS_n : std_logic;
126   signal Adc_PDEn : std_logic;
127   signal Adc_ChSel : std_logic;
128   signal Gen_L2_p : std_logic;
129   signal Gen_L2_n : std_logic;
130   signal Gen_L3_p : std_logic;
131   signal Gen_L3_n : std_logic;
132   signal Gen_C2 : std_logic;
133   signal Gen_C3 : std_logic;
134   signal Gen_ClkOut_p : std_logic;
135   signal Gen_ClkOut_n : std_logic;
136   signal IP2Bus_Data : std_logic_vector(0 to 31);
137   signal IP2Bus_RdAck : std_logic;
138   signal IP2Bus_WrAck : std_logic;
139   signal IP2Bus_Error : std_logic;
140   signal IP2RFIFO_WrReq : std_logic;
141   signal IP2RFIFO_Data : std_logic_vector(0 to 31);
142
143   -- Clock period definitions
144   constant Gen_Clk_period : time := 4 ns;
145   signal count_adc: integer := 0;
146  signal count_i2p: integer := 0;
147
148  constant EXTDELAY: time := 1ns;
149
150 BEGIN
151
152   -- Instantiate the Unit Under Test (UUT)
153    uut: user_logic PORT MAP (
154          Ham_Eos => Ham_Eos,
155          Ham_Trigger => Ham_Trigger,
156          Ham_ClkIn => Ham_ClkIn,
157          Ham_ClkOut => Ham_ClkOut,
158          Ham_Reset_n => Ham_Reset_n,
159          Ham_Gain => Ham_Gain,
```

```
160           Ham_InStart => Ham_InStart,
161           Adc_SDO => Adc_SDO,
162           Adc_ClkOut => Adc_ClkOut,
163           Adc_CS_n => Adc_CS_n,
164           Adc_PDEn => Adc_PDEn,
165           Adc_ChSel => Adc_ChSel,
166           Gen_Clk => Gen_Clk,
167           Gen_L0_p => Gen_L0_p,
168           Gen_L0_n => Gen_L0_n,
169           Gen_L1_p => Gen_L1_p,
170           Gen_L1_n => Gen_L1_n,
171           Gen_L2_p => Gen_L2_p,
172           Gen_L2_n => Gen_L2_n,
173           Gen_L3_p => Gen_L3_p,
174           Gen_L3_n => Gen_L3_n,
175           Gen_C0 => Gen_C0,
176           Gen_C1 => Gen_C1,
177           Gen_C2 => Gen_C2,
178           Gen_C3 => Gen_C3,
179           Gen_ClkOut_p => Gen_ClkOut_p,
180           Gen_ClkOut_n => Gen_ClkOut_n,
181           Bus2IP_Clk => Bus2IP_Clk,
182           Bus2IP_Reset => Bus2IP_Reset,
183           Bus2IP_Data => Bus2IP_Data,
184           Bus2IP_BE => Bus2IP_BE,
185           Bus2IP_RdCE => Bus2IP_RdCE,
186           Bus2IP_WrCE => Bus2IP_WrCE,
187           IP2Bus_Data => IP2Bus_Data,
188           IP2Bus_RdAck => IP2Bus_RdAck,
189           IP2Bus_WrAck => IP2Bus_WrAck,
190           IP2Bus_Error => IP2Bus_Error,
191           IP2RFIFO_WrReq => IP2RFIFO_WrReq,
192           IP2RFIFO_Data => IP2RFIFO_Data,
193           RFIFO2IP_WrAck => RFIFO2IP_WrAck,
194           RFIFO2IP_AlmostFull => RFIFO2IP_AlmostFull,
195           RFIFO2IP_Full => RFIFO2IP_Full,
196           RFIFO2IP_Vacancy => RFIFO2IP_Vacancy
197         );
198
199     Gen_Clk_process :process
200      begin
201       Gen_Clk <= '1';
202       wait for Gen_Clk_period/2;
203       Gen_Clk <= '0';
204       wait for Gen_Clk_period/2;
205      end process;
206
207      Bus2IP_Clk_process :process(Gen_Clk)
208      begin
209       if(rising_edge(Gen_Clk)) then
210         if (count_i2p = 1) then
211           count_i2p <= 0;
212         else
213           count_i2p <= count_i2p +1;
214         end if;
215         if (count_i2p = 0) then
216           Bus2IP_Clk <= NOT Bus2IP_Clk;
217         end if;
218       end if;
219      end process;
220
221
222      stim_proc: process
223      begin
224          wait for Gen_Clk_period*20;
225         Bus2IP_WrCE<="01000000";
226         Bus2IP_BE(1)<='1';
227         Bus2IP_BE(2)<='1';
228         Bus2IP_BE(0)<='1';
229         Bus2IP_BE(3)<='1';
230           Bus2IP_Data(27)<='1';
231
232         wait for Gen_Clk_period*20;
233           Bus2IP_WrCE<="10000000";
```

```
234           Bus2IP_Data(27)<='0';
235           Bus2IP_Data(26)<='1';
236           Bus2IP_Data(25)<='0';
237           Bus2IP_Data(24)<='1';
238           Bus2IP_Data(23)<='1';
239           Bus2IP_Data(22)<='1';
240           Bus2IP_Data(21)<='1';
241           Bus2IP_Data(20)<='1';
242           Bus2IP_Data(13)<='1';
243         Bus2IP_Data(12)<='1';
244           Bus2IP_Data(11)<='1';
245         wait for Gen_Clk_period*20;
246           Bus2IP_WrCE<="10000000";
247           Bus2IP_Data(29)<='1';
248       wait for Gen_Clk_period*2000;
249           Bus2IP_WrCE<="10000000";
250           Bus2IP_Data(0)<='0';
251         wait for Gen_Clk_period*20;
252           Bus2IP_WrCE<="01000000";
253           Bus2IP_Data(27)<='1';
254           Bus2IP_Data(29)<='0';
255           Bus2IP_Data(26)<='0';
256           Bus2IP_Data(25)<='0';
257           Bus2IP_Data(24)<='0';
258           Bus2IP_Data(23)<='0';
259           Bus2IP_Data(22)<='0';
260           Bus2IP_Data(21)<='0';
261           Bus2IP_Data(20)<='0';
262           Bus2IP_Data(13)<='0';
263         Bus2IP_Data(12)<='0';
264         Bus2IP_Data(11)<='0';
265         Bus2IP_Data(0)<='0';
266           Bus2IP_Data(1)<='1';
267         wait for Gen_Clk_period*6395;
268       Gen_C0<='1' after EXTDELAY;
269       Ham_Trigger<='1' after EXTDELAY;
270       wait for Gen_Clk_period*126;
271         Gen_C0<='0' after EXTDELAY; Bus2IP_BE(1)<='0';
272         Bus2IP_BE(2)<='0';
273         Bus2IP_BE(0)<='0';
274         Bus2IP_BE(3)<='0';
275       wait for Gen_Clk_period*3579;
276         Ham_Trigger<='1' after EXTDELAY;
277       wait for Gen_Clk_period*128;
278         Ham_Trigger<='0'  after EXTDELAY;
279       wait for Gen_Clk_period*250;
280         Ham_Trigger<='1'  after EXTDELAY;
281       wait for Gen_Clk_period*128;
282         Adc_SDO<='1' after EXTDELAY;
283       wait for Gen_Clk_period*22;
284         Adc_SDO<='0' after EXTDELAY;
285       wait for Gen_Clk_period*100;
286         Ham_Trigger<='0'  after EXTDELAY;
287       wait for Gen_Clk_period*1000;
288         Ham_Trigger<='1'  after EXTDELAY;
289       wait for Gen_Clk_period*128;
290         Adc_SDO<='1' after EXTDELAY;
291       wait for Gen_Clk_period*22;
292         Adc_SDO<='0' after EXTDELAY;
293       wait for Gen_Clk_period*100;
294         Ham_Trigger<='0'  after EXTDELAY;
295       wait for Gen_Clk_period*1000;
296         Ham_Trigger<='1'  after EXTDELAY;
297       wait for Gen_Clk_period*128;
298         Adc_SDO<='1' after EXTDELAY;
299       wait for Gen_Clk_period*22;
300         Adc_SDO<='0' after EXTDELAY;
301       wait for Gen_Clk_period*100;
302         Ham_Trigger<='0'  after EXTDELAY;
303       wait for Gen_Clk_period*1000;
304         Ham_Trigger<='1'  after EXTDELAY;
305       wait for Gen_Clk_period*128;
306         Adc_SDO<='1' after EXTDELAY;
307       wait for Gen_Clk_period*22;
```

```
308         Adc_SDO<='0' after EXTDELAY;
309       wait for Gen_Clk_period*100;
310         Ham_Trigger<='0'  after EXTDELAY;
311       wait for Gen_Clk_period*1000;
312         Ham_Trigger<='1'  after EXTDELAY;
313       wait for Gen_Clk_period*128;
314         Adc_SDO<='1' after EXTDELAY;
315       wait for Gen_Clk_period*22;
316         Adc_SDO<='0' after EXTDELAY;
317       wait for Gen_Clk_period*100;
318         Ham_Trigger<='0'  after EXTDELAY;
319         Ham_Eos<='0'  after EXTDELAY;
320       wait for Gen_Clk_period*250;
321       Ham_Eos<='1'  after EXTDELAY;
322         wait;
323     end process;
324
325 END;
```

Bibliography

[1] A. Mapelli et al. "Scintillation particle detection based on microfluidics". In: *Sensors and Actuators A: Physical* 162.2 (2010), pp. 272–275.

[2] MJ Madou. *Fundamentals of microfabrication: the science of miniaturization*. CRC, 2002.

[3] P Forck. *Lecture notes on beam instrumentation and diagnostics*. Joint University Accelerator School. 2008.

[4] RJ Tapper. "Diamond detectors in particle physics". In: *Reports on Progress in Physics* 63 (2000), p. 1273.

[5] William Trischuk. *Recent Advances in Diamond Detectors*. RD42 Collaboration. 2004.

[6] H. Kagan. *ATLAS and CVD Diamond Detectors*. Ohio State University. 1999.

[7] JF Ziegler. "Stopping of energetic light ions in elemental matter". In: *Journal of applied physics* 85 (1999), p. 1249.

[8] WM Yao et al. "Review of particle physics". In: *Journal of Physics G: Nuclear and Particle Physics* 33 (2006), p. 1.

[9] W.R. Leo. *Techniques for nuclear and particle physics experiments: a how-to approach*. Springer Verlag, 1994.

[10] Alessandro Mapelli. "Scintillation Particle Detectors Based on Plastic Optical Fibres and Microfluidics". PhD thesis. ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2011.

[11] G.F. Knoll. *Radiation detection and measurement*. New York, John Wiley and Sons, Inc., 1979.

[12] C. Cianfarani et al. "A high-resolution detector based on liquid-core scintillating fibres with readout via an electron-bombarded charge-coupled device". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 339.3 (1994), pp. 449–455.

[13] C. D'Ambrosio et al. "Particle tracking with scintillating fibers". In: *Nuclear Science, IEEE Transactions on* 43.3 (1996), pp. 2115–2127.

[14] Daniel C. Konnoff. *SSPM-Based Optical Fiber Radiation Dosimeter*. 2012.

[15] A first study of a scintillating fibre detector for a muon ionisation cooling experiment. "McKigney, Edward and Gruber, Peter". In: *CERN Neutrino Factory Note 79, IC/HEP/01-1* (2001).

[16] Christian Joram. *Scintillation + Photo Detection*. CERN Summer Student Lectures, Particle Detectors. 2002.

[17] A. Mapelli. *Microfluidic Scintillation Detectors for Hadrotherapy Beam Monitoring*. KT fund. 2011.

[18] N. Prabhat and D. Gupta. "Fabrication and Analysis of Microchannels". In: *Department of Mechanical Engineering Indian Institute of Technology Bombay* (2006).

[19] MicroChem. *NANO SU-8, Negative Tone Photoresist*. Tech. rep. 1254 Chestnut Street Newton, MA 02464, 2002.

[20] FJ Blanco et al. "Novel three-dimensional embedded SU-8 microchannels fabricated using a low temperature full wafer adhesive bonding". In: *Journal of Micromechanics and Microengineering* 14 (2004), p. 1047.

[21] F.E.H. Tay et al. "A novel micro-machining method for the fabrication of thick-film SU-8 embedded micro-channels". In: *Journal of Micromechanics and Microengineering* 11 (2001), p. 27.

[22] F. Chollet. *SU-8: Thick Photo-Resist for MEMS*. 2012. URL: memscyclopedia.org/su8.html (visited on 05–2009).

[23] MicroChem. *SU-8 3000 Permanent Epoxy Negative Photoresist*.

[24] DM Mattox. *The foundations of vacuum coating technology*. William Andrew, 2003.

[25] B. Schlitt, A. Reiter, Breitenberger, et al. "Linac Commissioning at the Italian Hadrontherapy Centre CNAO". In: *Proceedings of the IPAC* 10 (2011).

[26] M. Pullia. "Status Report of the Cnao Construction and Commissioning". In: *IPAC2011, San Sebastián, Spain* (2011).

[27] ELJEN TECHNOLOGY. *EJ-305 Liquid Scintillator Highest Light Output*. 1300 W Broadway Sweetwater TX 79556 USA.

[28] R. Guerre. "Guided-wave micro-electro-mechanical systems (MEMS) optical switches for telecommunication applications". PhD thesis. EPFL, 2005.

[29] M. Teshima R. Mirzoyan M. Laatiaoui. "Very high quantum efficiency PMTs with bialkali photo-cathode". In: *Nuclear Instruments and Methods in Physics Research A* 567 (2006), pp. 230–232.

[30] C Hamamatsu Photonics. *Photodiode Technical Information*. 2010.

[31] P. Elmer. *Avalanche Photodiodes: A User's Guide*. 1998.

[32] C Hamamatsu Photonics. *S8664-series*. 2012.

[33] S. Cova et al. "Avalanche photodiodes and quenching circuits for single-photon detection". In: *Applied optics* 35.12 (1996), pp. 1956–1976.

[34] C Hamamatsu Photonics. *MPPC - Multi-Pixel Photon Counter, S10362-11 series*. 2010.

[35] C Hamamatsu Photonics. *Photodiode array combined with signal processing IC (S8865)*. 2011.

[36] C Hamamatsu Photonics. *Compact, easy-to-use driver circuit (C9118 series)*. 2011.

[37] C Texas Instruments. *14-Bit, 2-MSPS, Dual-Channel, Differential/Single-Ended, Ultralow-Power ADCs (Rev. B)*. 2011.

[38] Xilinx. *ML505/ML506/ML507 Evaluation Platform User Guide*. 2011.

[39] Stefano Veneziano and Paolo Bagiacchi. *Baord - FPGA - Ethernet protocol - List of commands*. 2012.

[40] R.P. Brent. "Uniform random number generators for supercomputers". In: *Proc. Fifth Australian Supercomputer Conference, Melbourne*. 1992, pp. 95–104.

[41] Laser2000. *NEW Picosecond Pulsed UV-LEDs*.

[42] Walter Scandale. *Introduction to Particle Accelerators*. CERN. 2010.

[43] R. Lauck et al. "Low-afterglow, high-refractive-index liquid scintillators for fast-neutron spectrometry and imaging applications". In: *Nuclear Science, IEEE Transactions on* 56.3 (2009), pp. 989–993.

[44] HR Krall, FA Helvy, and DE Persyk. "Recent Developments in GaP (Cs)-Dynode Photomultipliers". In: *Nuclear Science, IEEE Transactions on* 17.3 (1970), pp. 71–74.

[45] C. Radio Corporation of America. *RCA photomultiplier manual*. Technical series. RCA Electronic Components, 1970.

[46] M Gad el Hak. *The MEMS handbook*. CRC, 2002.

[47] C EMI Industrial Electronics. *EMI Photomultiplier Catalog*. Ltd. 1979.

[48] C. Burle Industries. *Photodiode Handbook*. USA, 1980.

[49] Agostino Di Francesco. *Personal Notes on APDs (Avalanche Photo-Diodes)*. 2012.