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 vii

1 Introduction 1
  1.1 Beam Monitoring . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.2 Hadrotherapy Beam Profile . . . . . . . . . . . . . . . . . . . . . . 2
  1.3 Existing Detectors . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    1.3.1 Scintillator Screens . . . . . . . . . . . . . . . . . . . . . . . . 5
    1.3.2 Wire Scanners . . . . . . . . . . . . . . . . . . . . . . . . . . 6
    1.3.3 Diamond Detectors . . . . . . . . . . . . . . . . . . . . . . . . 8

2 A New Detector for Beam Monitoring 10
  2.1 Basic Processes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
    2.1.1 Energy Loss . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
    2.1.2 Scintillation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  2.2 Organic Scintillators . . . . . . . . . . . . . . . . . . . . . . . . . . 14
  2.3 Scintillating fibres . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  2.4 Microcapillary Scintillators . . . . . . . . . . . . . . . . . . . . . . 18
  2.5 Microcapillary tubes . . . . . . . . . . . . . . . . . . . . . . . . . . 18
  2.6 The Detector Characteristics . . . . . . . . . . . . . . . . . . . . . . 23
    2.6.1 The Scintillator Detection Efficiency . . . . . . . . . . . . . . 23

3 Photosensors 26
  3.1 The Photodiodes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
    3.1.1 The PN Photodiode . . . . . . . . . . . . . . . . . . . . . . . . 27
    3.1.2 The Photodiode Equivalent Circuit and Noise . . . . . . . . . 28
  3.2 The Avalanche Photodiodes . . . . . . . . . . . . . . . . . . . . . . . 30
    3.2.1 The APD Working Principle . . . . . . . . . . . . . . . . . . . 30
    3.2.2 The APD Equivalent Circuit and Noise . . . . . . . . . . . . . 31
  3.3 The Multi-Pixel Photon Counter . . . . . . . . . . . . . . . . . . . . 33
    3.3.1 The MPPC Operating Principles . . . . . . . . . . . . . . . . . 33
3.3.2 The MPPC Equivalent Circuit and Its Noise Sources

3.4 Detectors Readout

4 The Photodiode Readout

4.1 System Requirements and Specifications

4.2 A General Overview

4.2.1 The Photodiode Array

4.2.2 The ADC

4.2.3 The FPGA Board

4.2.4 The Readout Architecture

4.2.5 The Readout and Control Protocol

4.3 Simulation

4.3.1 Linear feedback shift register

4.3.2 The Photodiode Array Simulation

4.3.3 The ADC Simulation

4.4 The FPGA Logic

4.4.1 The Deserializer

4.4.2 The Clock Divider

4.4.3 The Internal Fifo

4.4.4 The Finite State Machine

4.4.5 The Logic Control Unit

5 Laboratory Tests

5.1 Laboratory Test Setup

5.1.1 Equipment

5.1.2 The Graphical User Interface

5.2 Expected Performances

5.3 Debugging Procedures

6 Measurements

6.1 Geometry

6.2 System Characterization

6.2.1 Data Errors

6.2.2 Linearity results

6.3 Measurements

7 Conclusions
A The VHDL Code
A.1 Photodiode Simulation ................. 81
A.2 ADC Simulation ........................ 83
A.3 FPGA Logic .............................. 84
A.4 Test Bench ............................. 108

Bibliography .............................. 122
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 µm, 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 measurement 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.

![Depth dose curves](image)

**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 \(3 \sim 5 \text{ cm}\), with a spatial resolution of the order of \(100 \mu \text{m}\). 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 \(1 \text{ mm}\) water-equivalent. Moreover the size of the detector needs to be large enough to follow the beam rastering of the order of \(20 \times 20 \text{ cm}^2\), while the readout system needs to be capable of following the beam raster frequency of the order of \(100 \text{ Hertz}\).

- 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].](image)

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.
1.3 Existing Detectors

- 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 properties of possible material used to fabricate the scintillator screens are shown.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Material</th>
<th>Activator</th>
<th>max. emission</th>
<th>decay time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quartz</td>
<td>SiO₂</td>
<td>none</td>
<td>470 nm</td>
<td>&lt; 10 ns</td>
</tr>
<tr>
<td></td>
<td>CsI</td>
<td>Tl</td>
<td>550 nm</td>
<td>1 μs</td>
</tr>
<tr>
<td>Chromolux</td>
<td>Al₂O₃</td>
<td>Cr</td>
<td>700 nm</td>
<td>100 ms</td>
</tr>
<tr>
<td>YAG</td>
<td>Y₃Al₅O₁₂</td>
<td>Ce</td>
<td>550 nm</td>
<td>0.2 μs</td>
</tr>
<tr>
<td></td>
<td>Li glass</td>
<td>Ce</td>
<td>400 nm</td>
<td>0.1 μs</td>
</tr>
<tr>
<td>P11</td>
<td>ZnS</td>
<td>Ag</td>
<td>450 nm</td>
<td>3 ms</td>
</tr>
<tr>
<td>P43</td>
<td>Gd₂O₃S</td>
<td>Tb</td>
<td>545 nm</td>
<td>1 ms</td>
</tr>
<tr>
<td>P46</td>
<td>Y₃Al₅O₁₂</td>
<td>Ce</td>
<td>530 nm</td>
<td>0.3 μs</td>
</tr>
<tr>
<td>P47</td>
<td>Y₂Si₅O₁₅</td>
<td>Ce&amp;Tb</td>
<td>400 nm</td>
<td>100 ns</td>
</tr>
</tbody>
</table>

Figure 1.3. Different properties 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.
1.3 Existing Detectors

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.
1.3 Existing Detectors

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 measurement 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.

![Schematic representation of a diamond detector](image)

**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 μm [6]), good temperature stability (about 1% K$^{-1}$) and good radiation stability and tolerance (up to 2·10$^{15}$ 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 \( \sigma \approx 10^{-17} \div 10^{-16} \text{cm}^2 \) where \( \sigma \) is defined as the effective area which governs the probability of some scattering or absorption event and is mathematically calculated as:

\[
\sigma(E) = \int \frac{d\sigma}{d\Omega} d\Omega
\]  
(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

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

(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:

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

(2.3)

with

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

- $z$: charge of the incident particle expressed in units of the elementary charge $e$
- $\beta = \frac{v}{c}$ of the incident particle
- $\gamma = \frac{1}{\sqrt{1 - \beta^2}}$
- $\delta$: density correction
- $C$: shell correction
- $W_{\text{max}}$: 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 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 2.1](image)

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

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 $\tau$. 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 $\pi$-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.
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:

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

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
2.3 Scintillating fibres

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 $n_1 = 1.50$ and $n_2 = 0.99$. 
### Table A6.3 Properties of some organic scintillators

<table>
<thead>
<tr>
<th>scintillator</th>
<th>density (g/cm³)</th>
<th>index of refraction</th>
<th>wavelength of maximum emission (nm)</th>
<th>decay time constant (ns)</th>
<th>scintillation pulse height ¹</th>
<th>H/C ratio ²</th>
<th>yield/NaI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monocrysalts</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>naphthalene</td>
<td>1.15</td>
<td>1.58</td>
<td>348</td>
<td>11</td>
<td>11</td>
<td>0.800</td>
<td></td>
</tr>
<tr>
<td>anthracene</td>
<td>1.25</td>
<td>1.59</td>
<td>448</td>
<td>30-32</td>
<td>100</td>
<td>0.714</td>
<td></td>
</tr>
<tr>
<td>trans-stilbene</td>
<td>1.16</td>
<td>1.58</td>
<td>384</td>
<td>3-8</td>
<td>46</td>
<td>0.857</td>
<td></td>
</tr>
<tr>
<td>p-terphenyl</td>
<td>1.23</td>
<td>1.58</td>
<td>391</td>
<td>6-12</td>
<td>30</td>
<td>0.778</td>
<td></td>
</tr>
<tr>
<td><strong>Plastics³⁴</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NE 102 A</td>
<td>1.032</td>
<td>1.58</td>
<td>425</td>
<td>2.5</td>
<td>65</td>
<td>1.105</td>
<td>0.5</td>
</tr>
<tr>
<td>NE 104</td>
<td>1.032</td>
<td>1.58</td>
<td>405</td>
<td>1.8</td>
<td>68</td>
<td>1.100</td>
<td></td>
</tr>
<tr>
<td>NE 110</td>
<td>1.032</td>
<td>1.58</td>
<td>437</td>
<td>3.3</td>
<td>60</td>
<td>1.105</td>
<td></td>
</tr>
<tr>
<td>NE 111</td>
<td>1.032</td>
<td>1.58</td>
<td>370</td>
<td>1.7</td>
<td>55</td>
<td>1.096</td>
<td></td>
</tr>
<tr>
<td><strong>Plastics³⁴</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC-400</td>
<td>1.032</td>
<td>1.581</td>
<td>423</td>
<td>2.4</td>
<td>65</td>
<td>1.103</td>
<td></td>
</tr>
<tr>
<td>BC-404</td>
<td>1.032</td>
<td>1.58</td>
<td>408</td>
<td>1.8</td>
<td>68</td>
<td>1.107</td>
<td></td>
</tr>
<tr>
<td>BC-408</td>
<td>1.032</td>
<td>1.58</td>
<td>425</td>
<td>2.1</td>
<td>64</td>
<td>1.104</td>
<td></td>
</tr>
<tr>
<td>BC-412</td>
<td>1.032</td>
<td>1.58</td>
<td>434</td>
<td>3.3</td>
<td>60</td>
<td>1.104</td>
<td></td>
</tr>
<tr>
<td>BC-414</td>
<td>1.032</td>
<td>1.58</td>
<td>392</td>
<td>1.8</td>
<td>68</td>
<td>1.110</td>
<td></td>
</tr>
<tr>
<td>BC-416</td>
<td>1.032</td>
<td>1.58</td>
<td>434</td>
<td>4.0</td>
<td>50</td>
<td>1.110</td>
<td></td>
</tr>
<tr>
<td>BC-418</td>
<td>1.032</td>
<td>1.58</td>
<td>391</td>
<td>1.4</td>
<td>67</td>
<td>1.100</td>
<td></td>
</tr>
<tr>
<td>BC-420</td>
<td>1.032</td>
<td>1.58</td>
<td>391</td>
<td>1.5</td>
<td>64</td>
<td>1.100</td>
<td></td>
</tr>
<tr>
<td>BC-422</td>
<td>1.032</td>
<td>1.58</td>
<td>370</td>
<td>1.6</td>
<td>55</td>
<td>1.102</td>
<td></td>
</tr>
<tr>
<td>BC-422Q</td>
<td>1.032</td>
<td>1.58</td>
<td>370</td>
<td>0.7</td>
<td>11</td>
<td>1.102</td>
<td></td>
</tr>
<tr>
<td>BC-428</td>
<td>1.032</td>
<td>1.58</td>
<td>480</td>
<td>12.5</td>
<td>50</td>
<td>1.103</td>
<td></td>
</tr>
<tr>
<td>BC-430</td>
<td>1.032</td>
<td>1.58</td>
<td>580</td>
<td>16.8</td>
<td>45</td>
<td>1.108</td>
<td></td>
</tr>
<tr>
<td>BC-434</td>
<td>1.049</td>
<td>1.58</td>
<td>425</td>
<td>2.2</td>
<td>60</td>
<td>0.995</td>
<td></td>
</tr>
</tbody>
</table>

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 µm and some competitive results for thicknesses smaller than 300 µm. For lengths smaller than 100 µm, 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
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 µm 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, making it a good candidate for the case studied. From such values the radiation length can be expressed as:

\[
\frac{M_0}{X_0} = \sum \frac{M_i}{X_i}
\]  

(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/cm² 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.
2.5 Microcapillary tubes

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 $dE_{CNAO}/dx = 15.87 MeV/cm$ (2.6) and $dE_{MIP}/dx = 1.8 MeV/cm$ (2.7) can be obtained for the two beams.

The depth of a scintillating channel can be considered on the order of 200 $\mu m$ that, multiplied for the two stopping powers found, yields an energy deposited in the scintillating material of $\Delta E_{CNAO} = 0.32 MeV$ (2.8) and $\Delta E_{MIP} = 3.6 \times 10^{-2} MeV$. (2.9)

Finally, knowing from the EJ-305 datasheet [27] that every $MeV$ of energy deposited in the scintillating material generates $12 \times 10^3$ photons with a wavelength of approximately 424 $nm$, the total number of photons produced in both cases can be calculated, obtaining $N_{\gamma, CNAO} \approx 3800$ and $N_{\gamma, MIP} \approx 430$; from these results a round number of $2 \times 10^3 \gamma/mm\ per\ mip$ 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 \[1\]:

\[
N_{pe} = N\gamma \cdot \epsilon_{coll} \cdot \epsilon_{refl} \cdot \epsilon_{att} \cdot \epsilon_{in} \cdot \epsilon_{Qeff}
\] (2.10)

where \(N\gamma\) represents the scintillations produced by an interacting proton, \(\epsilon_{coll}\) is the collection efficiency of a rectangular metal-coated microchannel, \(\epsilon_{refl}\) the gain caused by the reflective end of the channel opposite to the photodetector, \(\epsilon_{att}\) the transport efficiency due to optical absorption, \(\epsilon_{in}\) the transmission efficiency at the interface between the microchannel and the photodetector and \(\epsilon_{Qeff}\) 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 \(\epsilon_{coll}\) and \(\epsilon_{refl}\) respectively 0.03 and 1.4 through Monte Carlo simulations \[1\], omitting \(\epsilon_{att}\) due to its large value compared to the small size of the detector, considering \(\epsilon_{in}\) to be around 0.99 and knowing the \(\epsilon_{Qeff}\) 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 \(\epsilon_{det}\) defined as the probability of observing one photon, will be given by one minus the probability of not detecting any scintillation, that is:

\[
\epsilon_{det} \approx 1 - P(0, N_{pe}) = 1 - e^{-N_{pe}}
\] (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:

\[
\epsilon_{det}^n \approx 1 - \sum_{i=0}^{n-1} P(i, N_{pe}) = 1 - e^{-N_{pe}} \cdot \sum_{i=0}^{n-1} \left( \frac{N_{pe}^i}{i!} \right).
\] (2.12)
2.6 The Detector Characteristics

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:

\[
QE = \frac{\text{number of photoelectrons emitted}}{\text{number of photons absorbed}}.
\]  

(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.

![Photodiode Diagrams](image)

**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
3.1 The Photodiodes

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.

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:

$$I_L = S \cdot N_\gamma \cdot h\nu$$  \hspace{1cm} (3.2)
where $S$ is the sensitivity of the photodiode expressed in $A/W$, $N$, the number of reacting photons per second, $\nu$ 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:

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

(3.3)

where $I_S$ is the saturation current, $V_D$ the potential energy between the two diode’s ends, $\eta$ 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:

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

(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.
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.
3.2 The Avalanche Photodiodes

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.

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:
3.2 The Avalanche Photodiodes

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:

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

where $M \cdot I_S$ is the saturation current, $V_D$ the potential energy between the two diode’s ends, $\eta$ 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.

$$< n_L^2 > = 2q_0 I_L M^2 F [A^2/Hz]$$ \hspace{1cm} (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:

$$<n^2_D> = 2q_eI_SM^2(exp \frac{V_D}{\eta VT} + 1)[A^2/Hz] \quad (3.8)$$

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

$$<n^2_S> = \frac{4k_BT}{R_S}[A^2/Hz] \quad (3.9)$$

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:

$$<n^2_{APD}> = \sum_i \frac{1}{2\pi} \int_0^\infty <n^2_i> |H_i(j\omega)|^2d\omega [A^2] \quad (3.10)$$

where $<n^2_i>$ 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].](image)

### 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 50$\Omega$. 
3.3 The Multi-Pixel Photon Counter

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

$$\sigma_{\text{stat}} \propto \sqrt{N_{\text{count}}}$$  \hspace{1cm} (3.11)

where $N_{\text{count}}$ 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 $100 \text{kcps}$.

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:

\[
N_{\text{fired}} = N_{\text{tot}} \left[ 1 - \exp^{-\frac{N_{\text{ph}} \cdot PDE}{N_{\text{tot}}}} \right]
\]  

where \( N_{\text{fired}} \) is the number of SPADs activated, \( N_{\text{tot}} \) the total number of SPADs, \( N_{\text{ph}} \) 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.

![Table of parameters of some commercial MPPC](image)

**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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{phDet}$</td>
<td>40 ÷ 2000</td>
<td>kHz</td>
</tr>
<tr>
<td>$f_{adc}$</td>
<td>40</td>
<td>MHz</td>
</tr>
<tr>
<td>$n_{Bit}(ADC)$</td>
<td>14</td>
<td>Bit</td>
</tr>
<tr>
<td>Integration Time</td>
<td>$0.5 \div 1.64 \cdot 10^6$</td>
<td>$\mu$s</td>
</tr>
<tr>
<td>Input Voltage</td>
<td>4.8</td>
<td>V</td>
</tr>
<tr>
<td>FIFO Length</td>
<td>2056</td>
<td>32bits words</td>
</tr>
</tbody>
</table>

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 fulfill 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.

![Spectral response](image1.png)

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

![Timing chart](image2.png)

**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
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:

\[
V_{\text{sig}} = \frac{kSF_pN_{pe}A_{ch}hvT_{int}}{A_{pd}e} \tag{4.1}
\]

where

- \( k \) is a constant equal to 13.66 cd/W
- \( S \) represents the photodetector photosensitivity
- \( F_p \) stands for the incident beam flux
- \( N_{pe} \) is the expected photoelectrons produced by a single particle
- \( A_{ch} \) represents the cross section of a microcapillary channel
4.2 A General Overview

- $h$ stands for the Planck constant
- $\nu$ 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 $F_p = 0.25 \times 10^{14} \text{ particles m}^{-2}\text{s}^{-1}$ 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 $5 \times 10^{-7} \text{ m}^2$, a photodetector surface of $1.8 \times 10^{-7} \text{ m}^2$ and an integration time of $100 \text{ ms}$, a signal amplitude of approximately $737 \text{ mV}$ and $84 \text{ mV}$ is obtained.

In general, a value of approximately $0.35 \mu\text{V/mip}$ 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 $0.01 \text{ mV/ms}$ 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 $1 \text{ ms}$ and knowing that each photon generates a signal output of the order of $0.10 \mu\text{V}$, 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 $4.8 \text{ V}$ 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 $1.65 \text{ V}$, 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].](image1)

Figure 4.5. ADS7946 block diagram [37].

![Figure 4.6. ADS7946 timing diagram [37].](image2)

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 into multiple entities programmed with the Xilinx software suites ISE and EDK. The architecture implemented into the FPGA board is shown in Figure 4.9.

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.
4.2 A General Overview

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 : 0α : 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 bit 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.

![Diagram of Ethernet UDP header and frame](image)

**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.

![Diagram of Ethernet UDP frame](image)

**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.
4.3 Simulation

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 in 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.
4.3 Simulation

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.

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
from which the generalization of the generators:

\[ S_n = S_{n-j} \ast S_{n-k} \pmod{m}, \, 0 < j < k < n \]  

where \( m \) is usually a multiple of two (i.e. \( m = 2^M \)), the variable \( M,n,j \) and \( k \) are natural numbers and \( \ast \) 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:

\[ B_0 = B_{13} \oplus B_{12}, \; t > t_0 \]
\[ B_{13} = 1, \; B_i = 0, \; \forall i = 0, 1, 2...12, \; t = t_0 \]

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.
4.3 Simulation

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.

![Timing diagram of the photodiode array and its related simulation.](image)

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.
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 \( \text{datain} \) port equals 1, the internal \( \text{outputInteger} \) signal is set equals to:

\[
\text{outputInteger} = \text{outputInteger} + 2^{\text{width} - 3 - n}
\]  

(4.5)

where the variable \( \text{width} \) 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 \( \text{dataout} \) 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 $40 \text{ Mhz}$ while the photodiode one must be lower than $2 \text{ Mhz}$. The only other constraint given by the structure of the readout system is:

$$T_{\text{photo}} > 32T_{\text{adc}} + 80\text{ns}$$ (4.6)

where the $T_{\text{photo}}$ is the photodiode clock period and $T_{\text{adc}}$ 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 $\text{ns}$ 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 $\text{lengthDivisor}$ 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 \( \text{divisor}/2 \) the first time and to \( \text{divisor} - \text{divisor}/2 \) the second time, the \( \text{clkOut} \) output port is set equals to:

\[
\text{clkOut} = \text{NOT}(\text{clkOut}).
\] (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.

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\( \text{MHz} \) 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\( \text{MHz} \) FPGA clock while the latter by the 125\( \text{MHz} \) 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 \( \text{rd}_\text{en} \) 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.
4.4 The FPGA Logic

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 The FPGA Logic

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 additionally 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 $250 \text{MHz}$ 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.

![Clock Analyzer Schematic](image)

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

![Clock Analyzer Time Diagram](image)

**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:

$$T_{int} = n \cdot T_{photo}$$  \hspace{1cm} (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
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 4.28.** Schematic circuit diagram of the trigger analyzer logic block.

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

![Picture of laboratory setup]

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
5.2 Expected Performances

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 µW of power on a spot of 123 mm$^2$ of size. The number of photons reaching the photodiode array can be expressed as:

$$n_{\text{ph}}^\gamma = T_{\text{int}} \cdot f_{\text{led}} \cdot n_{\text{pul}}^\gamma \cdot \left(\frac{A_{PA}}{A_{LB}}\right)$$  \hspace{1cm} (5.1)$$

where $n_{\text{ph}}^\gamma$ is the total number of photons detected during every photodiode measurement, $T_{\text{int}}$ is the integration time of the photodetector, $f_{\text{led}}$ is the frequency of the LED pulses, $n_{\text{pul}}^\gamma$ 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

$$n_{\text{pul}}^\gamma = \frac{P_{\text{led}} \cdot T_{\text{pul}}}{h \nu_\gamma}$$  \hspace{1cm} (5.2)$$

where $P_{\text{led}}$ is the power generated by the EPLED per pulse, $T_{\text{pul}}$ the average time interval of each pulse, $h$ the Planck constant and $\nu_\gamma = 833 \text{ THz}$ the frequency of photons generated. The number of photons per pulse yielded is $n_{\text{pul}}^\gamma \simeq 1014$. Knowing most of these values from the EPLED and photodiode data sheets [41] [36] a $n_{\text{pul}}^\gamma \simeq 1.8 \cdot 10^5$ is yielded, using $T_{\text{int}} = 1 \text{ ms}$, $f_{\text{led}} = 10 \text{ MHz}$ and $A_{PA} = 2.24 \text{ mm}^2$.

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:

$$V_{\text{data}} = Kn_{\text{ph}}^\gamma = K T_{\text{int}} \cdot f_{\text{led}} \cdot \frac{P_{\text{led}} \cdot T_{\text{pul}}}{h \nu_\gamma} \cdot \left(\frac{A_{PA}}{A_{LB}}\right)$$  \hspace{1cm} (5.3)$$

where $V_{\text{data}}$ is the photodetector output signal, $n_{\text{ph}}^\gamma$ the total number of photons detected and $K$ the conversion factor of about 50 µV 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.
5.3 Debugging Procedures

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.
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.
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 $3 \text{ mm}$ 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.

![Graph](image)

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

### 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 $50 \mu V/\text{photon}$ 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 $1 \text{ms}$ a photodiode array frequency of about $2 \text{MHz}$ 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.
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).
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.

![Image of the setup used to take the first measurements with the microcapillary channels.](image)

**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.

![Picture of the channels filled with the Cargille and Rhodamine 6G solution.](image)

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

In such a setup different measurements have been taken by irradiating the previously described microchannel 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.

![Data Mean Constant Subtracted](image)

**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 from 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
6library IEEE;
7use IEEE.STD_LOGIC_1164.ALL;
8use IEEE.STD_LOGIC_ARITH.ALL;
9
10entity PhotodiodeArraySimulationTop is
11generic ( width : integer := 14 );
12Port ( clk : in STD_LOGIC;
13reset : in STD_LOGIC;
14video : out STD_LOGIC_VECTOR ( width-1 downto 0);
15trigger : out STD_LOGIC;
16eos : out STD_LOGIC );
17end PhotodiodeArraySimulationTop;
18
19architecture Behavioral of PhotodiodeArraySimulationTop is
20
21component PhotodiodeArraySimulation
22generic ( width : integer := 14 );
23Port ( Clk : in STD_LOGIC;
24Reset : in STD_LOGIC;
25random_video : in STD_LOGIC_VECTOR (width-1 downto 0);
26video : out STD_LOGIC_VECTOR (width-1 downto 0);
27trigger : out STD_LOGIC;
28eos : out STD_LOGIC );
29end component;
30
31component random
32generic ( width : integer := 14 );
33Port ( clk : in std_logic;
34random_num : out std_logic_vector (width-1 downto 0));
35end component;
36
37signal randomInterno : std_logic_vector (width-1 downto 0);
begin
PhotodiodeArraySimulation logic : PhotodiodeArraySimulation port map (clk=>clk,
40 reset=>reset,
video=>video,
trigger=>trigger,
random_video=>random Internal,
eos=>eos);
41
random logic : random port map (clk=>clk,
42 random_num=>random Internal);
43
end Behavioral;

PhotodiodeArraySimulation.vhd

begin

PhotodiodeArraySimulation logic : PhotodiodeArraySimulation port map (clk=>clk,
reset=>reset,
video=>video,
trigger=>trigger,
random_video=>random Internal,
eos=>eos);

random logic : random port map (clk=>clk,
random_num=>random Internal);

end Behavioral;

PhotodiodeArraySimulation.vhd

--- PhotodiodeArraySimulation.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity PhotodiodeArraySimulation is

generic (width : integer := 14);

port (Clk : in STD_LOGIC;
Reset : in STD_LOGIC;
random_video : in STD_LOGIC_VECTOR (width-1 downto 0);
Video : out STD_LOGIC_VECTOR (width-1 downto 0);
Trigger : out STD_LOGIC;
EOS : out STD_LOGIC);

end PhotodiodeArraySimulation;

architecture Behavioral of PhotodiodeArraySimulation is

constant maxChannelNumber : integer := 255;
signal count : integer range 0 to 6;
signal channelCounter : integer range 0 to maxChannelNumber := 0;
signal output : std_logic_vector (width-1 downto 0);
signal trig : std_logic;
signal started : std_logic;
signal eosint : std_logic;
signal counterFromReset : integer range 0 to 19;
signal stopping : std_logic := '0';

begin

process(Reset, Clk)
begin
if (Reset='event and Reset='0') then
   count<=0;
   channelCounter<=0;
   stopping<= '0';
   output<=(others=>>'0');
   counterFromReset<=1;
   trig<='0';
   eosint<='1';
   counterFromReset<=0;
elsif(Reset='0' and rising_edge(Clk) and counterFromReset<17) then
   counterFromReset<=counterFromReset+1;
elsif(channelCounter=(maxChannelNumber-1) and stopping='0') then
   stopping<='1';
elsif (rising_edge(Clk) and counterFromReset = 17) then
   if count=0 then
      count<=count+1;
   output<=>(random_video);
   elsif count=1 then
      count<=count+1;
      channelCounter<=channelCounter+1;
   end if;
end if;
A.2 ADC Simulation

```
58     trig='1';
59   elsif count=2 then
60     count<count+1;
61     output<=(others=>'0');
62     trig='0';
63   elsif (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 end process;
79
80 end Behavioral;
```

```
Random.vhd

-- ---------------------------
-- Random.vhd
-- ---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Random is
  generic ( width : integer := 14 );
  Port ( clk : in std_logic;
         random_num : out std_logic_vector (width-1 downto 0));
end Random;

architecture Behavioral of Random is
begin
  process(clk)
  variable rand_temp : std_logic_vector (width-1 downto 0):=(width-1=>'1', OTHERS=>'0');
  variable temp : std_logic := '0';
  begin
    if(rising_edge(clk)) then
      temp := rand_temp(width-1) xor rand_temp(width-2);
      rand_temp(width-1 downto 1) := rand_temp(width-2 downto 0);
      rand_temp(0) := temp;
    end if;
    random_num <= rand_temp;
  end process;
end Behavioral;
```

```
A.2     ADC Simulation

-- ---------------------------
-- AdcSimulation.vhd
-- ---------------------------
```
A.3 FPGA Logic

userlogic.vhd

-- userlogic.vhd -- entity/architecture pair

-- -------------------------------
--
-- Copyright (c) 1995-2012 Xilinx, Inc. All rights reserved.
--
-- **
-- -- Xilinx, Inc.
-- **
-- -- XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
-- **
-- -- AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND
-- **
-- -- SOLUTIONS FOR XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE,
-- **
-- -- OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,
-- **
-- -- APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION
-- **
-- -- THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
-- **
-- -- AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
-- **
-- -- FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY
-- **
-- -- WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
-- **
-- -- IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
-- **
-- -- REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
-- **

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

generic ( width : integer := 14 );

entity AdcSimulation is
  Port ( input : in STD_LOGIC_VECTOR (width-1 downto 0);
        clk : in STD_LOGIC;
        cs_n : in STD_LOGIC;
        sdo : out STD_LOGIC);
end AdcSimulation;

architecture Behavioral of AdcSimulation is

signal n : integer range 0 to width+3 :=0;
signal internal_data : std_logic_vector (width-1 downto 0);
signal waiting : std_logic ;
signal start : std_logic ;

begin
  process(clk , cs_n)
  begin
    if (cs_n='event and cs_n = '0') then
      sdo<='Z';
      start<='1';
      waiting<='0';
      internal_data<input;
      n<=0;
    elsif (rising_edge ( clk) and start = '1') then
      if (waiting = '0') then
        sdo<=internal_data(width-1-n);
      end if;
      if (n=width-1) then
        waiting<='1';
        internal_data<=('others' => '0');
      elsif (n=width) then
        sdo<='0';
      elsif (n=(width+2)) then
        n<=0;
        start<='0';
        waiting<='0';
      end if;
      n<=n+1;
    end if;
  end process;
end Behavioral;
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS **
-- FOR A PARTICULAR PURPOSE. **
-- **
-- ***************************************************************************
-- Filename : user_logic.vhd
-- Version : 2.00. a
-- Description : User logic.
-- Date : Fri Jun 22 16:08:56 2012 (by Create and Import Peripheral Wizard)
-- VHDL Standard : VHDL '93
-- Naming Conventions :
-- active low signals : "* _n"
-- clock signals : "clk", "clk_div#", "clk_#"
-- reset signals : "rst", "rst_n"
-- generics : "C_ *"
-- user defined types : "* _TYPE"
-- state machine next state : "* _ns"
-- state machine current state : "* _cs"
-- combinational signals : "* _ce"
-- pipelined or register delay signals : "* _d#"
-- counter signals : "* _cnt#"
-- clock enable signals : "* _ce"
-- internal version of output port : "* _i"
-- device pins : "* _pin"__
-- ports:
-- " Names begin with Uppercase"
-- processes:
-- " *PROCESS"
-- component instantiations:
-- "<ENTITY >_I_<#| FUNC>"
-- DO NOT EDIT BELOW THIS LINE --------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;
-- DO NOT EDIT ABOVE THIS LINE --------------------
-- USER libraries added here
library unisim;
use unisim.vcomponents.all;
-- Entity section
--
-- Definition of Generics :
-- C_SLV_DWIDTH -- Slave interface data bus width
-- C_NUM_REG -- Number of software accessible registers
-- C_RDFIFO_DEPTH -- Read FIFO depth
--
-- Definition of Ports :
-- Bus2IP_Clk -- Bus to IP clock
-- Bus2IP_Reset -- Bus to IP reset
-- Bus2IP_Data -- Bus to IP data bus
-- Bus2IP_BE -- Bus to IP byte enables
-- Bus2IP_WrCE -- Bus to IP write chip enable
-- Bus2IP_RdCE -- Bus to IP read chip enable
-- IP2Bus_Data -- IP to Bus data bus
-- IP2Bus_WrCE -- IP to Bus write chip enable
-- IP2Bus_RdCE -- IP to Bus read transfer acknowledgement
-- IP2Bus_Error -- IP to Bus error response
-- IP2RFIFO_WrReq -- IP to RFIFO : IP write request
-- IP2RFIFO_Data -- IP to RFIFO : IP write data bus
-- IP2RFIFO_WrCE -- IP to RFIFO write acknowledge
-- RFIFO2IP_WrAck -- RFIFO to IP : RFIFO write acknowledge
-- RFIFO2IP_AlmostFull -- RFIFO to IP : RFIFO almost full
-- RFIFO2IP_Full -- RFIFO to IP : RFIFO full
-- RFIFO2IP_Vacancy -- RFIFO to IP : RFIFO vacancy
-- Entity section
--
entity user_logic is
  generic
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    -- USER generics added here
    dataLength : integer := 16;
    integrationTimeBitMax : integer := 8;
    channelLength : integer := 8;
    fifoDepth : integer := 2048;
    hamDivisorLength : integer := 12;
    adcDivisorLength : integer := 6;
    -- ADD USER GENERICS ABOVE THIS LINE ---------------
  port
    -- ADD USER PORTS BELOW THIS LINE ------------------
    -- USER ports added here
    Ham_Eos : in std_logic;
    Ham_Trigger : in std_logic;
    Ham_ClkIn : in std_logic;
    Ham_ClkOut : out std_logic;
    Ham_Reset_n : out std_logic;
    Ham_Gain : out std_logic;
    Ham_InStart : out std_logic;
    Adc_SDO : in std_logic;
    Adc_ClkOut : out std_logic;
    Adc_CS_n : out std_logic;
    Adc_PDEn : out std_logic;
    Adc_ChSel : out std_logic;
    Gen_Clk : in std_logic;
    Gen_L0_p : in std_logic;
    Gen_L0_n : in std_logic;
    Gen_L1_p : in std_logic;
    Gen_L1_n : in std_logic;
    Gen_L2_p : out std_logic;
    Gen_L2_n : out std_logic;
    Gen_L3_p : out std_logic;
    Gen_L3_n : out std_logic;
    Gen_C0 : in std_logic;
    Gen_C1 : in std_logic;
    Gen_C2 : out std_logic;
    Gen_C3 : out std_logic;
    Gen_ClkOut_p : out std_logic;
    Gen_ClkOut_n : out std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------
    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH : integer := 32;
    C_NUM_REG : integer := 8;
    C_RDFIFO_DEPTH : integer := 16384
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
);
attribute MAX_FANOUT : string;
attribute SIGIS : string;
attribute SIGIS of Bus2IP_Clk : signal is "CLK";
attribute SIGIS of Bus2IP_Reset : signal is "RST";

end entity user_logic;

-- Architecture section

architecture IMP of user_logic is

component FPGA

generic ( wordLength : integer :=32;
dataLength : integer :=16;
integrationTimeBitMax : integer :=8;
-- max integration time expressed in Hackclk*2^integrationTimeBitMax
channelLength : integer :=8;
hamDivisorLength : integer :=12;
adcDivisorLength : integer := 6;
fifoDepth : integer :=2048);

Port ( rst_n : in std_logic;
trigger : in std_logic;
FPGA_Clk : in std_logic;
integTime : in std_logic_vector ( integrationTimeBitMax -1 downto 0);
FSM_Clk : in std_logic;
Ham_Eos : in std_logic;
extTrigger : in std_logic;
deseInput : in std_logic;
Test : in std_logic;
extFIFOFull : in std_logic;
triggerDaInvRst_n : in std_logic;
divisorHam : in std_logic_vector (0 to hamDivisorLength -1);
divisorAdc : in std_logic_vector (0 to adcDivisorLength -1);
triggerDaInv : out std_logic;
extTrigOut : out std_logic_vector ( wordLength -2 downto 0);
wordOut : out std_logic_VECTOR ( wordLength -1 downto 0);
photodiodeClkOut : out std_logic;
photodiodeStart : out std_logic;
ADC_ClkOut : out std_logic;
intFIFOFull : out std_logic;
intFIFOEmpty : out std_logic;
busy : out std_logic;
adcPDEN : out std_logic;
ADCStart : out std_logic);

end component;

signal L_int : std_logic_vector ( 3 downto 0);
signal L_int_n : std_logic_vector ( 3 downto 0);
signal sFIFOFull : std_logic ;
signal sFIFOEmpty : std_logic ;
signal sFIFOProgFull : std_logic ;
signal gen_C0_enable : std_logic ;
signal triggerAnsw : std_logic ;
signal integTime : std_logic_vector( integrationTimeBitMax-1 downto 0);
signal xHamDivisor : std_logic_vector(0 to hamDivisorLength-1);
signal xAdcDivisor : std_logic_vector(0 to adcDivisorLength-1);

end component;

signal sFIFOFull : std_logic ;
signal sFIFOEmpty : std_logic ;
signal sFIFOProgFull : std_logic ;
signal gen_C0_enable : std_logic ;

--USER signal declarations added here, as needed for user logic

----------------------------------------
-- Signals for user logic slave model a/w accessible register example
----------------------------------------
signal slv_reg5 : std_logic_vector (0 to C_SLV_DWIDTH-1);
signal slv_reg6 : std_logic_vector (0 to C_SLV_DWIDTH-1);
signal slv_reg7 : std_logic_vector (0 to C_SLV_DWIDTH-1);
signal slv_reg_write_sel : std_logic_vector (0 to 7);
signal slv_reg_read_sel : std_logic_vector (0 to 7);
signal slv_ip2bus_data : std_logic_vector (0 to C_SLV_DWIDTH-1);
signal slv_write_ack : std_logic;
signal slv_read_ack : std_logic;
signal orTrigExt : std_logic;
signal Test : std_logic;
signal extTrigCount : std_logic_vector (0 to C_SLV_DWIDTH-2);
signal TrigExt : std_logic;
signal sGen_C0 : std_logic;
signal FIFOWrEn : std_logic;

begin
-- USER logic implementation added here
FPGAlogic : FPGA
    generic map
        ( wordLength => C_SLV_DWIDTH ,
          dataLength => dataLength ,
          integrationTimeBitMax => integrationTimeBitMax ,
          channelLength => channelLength ,
          hamDivisorLength => hamDivisorLength ,
          adcDivisorLength => adcDivisorLength ,
          fifoDepth => fifoDepth )
    port map
        ( rst_n => rst_n ,
          trigger => Ham_Trigger ,
          FPGA_Clk => Gen_Clk ,
          integTime => integTime ,
          Test => Test ,
          Ham_Eos => Ham_Eos ,
          F3N_Clk => Bus2IP_Clk ,
          extTrigger => TrigExt ,
          deseInput => Adc_SDO ,
          extFIFOFull => RFIFO2IP_Full ,
          triggerDaIn => triggerDaIn ,
          extFifoWrEn => FIFOWrEn ,
          extTrigOut => extTrigCount ,
          divisorHam => sHamDivisor ,
          divisorAdc => sAdcDivisor ,
          wordOut => IP2RFIFO_Data ,
          photodiodeClkOut => Ham_ClkOut ,
          --photodiodeStart => Ham_InStart ,
          photodiodeStart => Ham_Reset_n ,
          AOC_ClkOut => Adc_ClkOut ,
          intFIFOFull => sIFIFOFull ,
          intFIFOEmpty => sIFIFOEmpty ,
          extFifoWrEn => FIFOWrEn ,
          busy => FIFOProgFull ,
          adcPDEN => Adc_PDn ,
          ADCStart => Adc_CS_n);

-- Example code to read/write user logic slave model s/w accessible registers
--
-- Note:
-- The example code presented here is to show you one way of reading/writing
-- software accessible registers implemented in the user logic slave model.
-- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
-- to one software accessible register by the top level template. For example,
-- if you have four 32 bit software accessible registers in the user logic,
-- you are basically operating on the following memory mapped registers:
--
-- Bus2IP_WrCE/Bus2IP_RdCE Memory Mapped Register
--  "1000" C_BASEADDR + 0x0
--  "0100" C_BASEADDR + 0x4
--  "0010" C_BASEADDR + 0x8
--  "0001" C_BASEADDR + 0xC
--
-- slv_reg_write_sel <= Bus2IP_WrCE(0 to 7);
-- slv_reg_read_sel <= Bus2IP_RdCE(0 to 7);
-- slv_write_ack <= Bus2IP_WrCE(0 to 1) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or Bus2IP_WrCE(3)
-- implement slave model software accessible register(s)

SLAVE_REG_WRITE_PROC : process ( Bus2IP_Clk ) is
begin

if Bus2IP_Clk ' event and Bus2IP_Clk = '1' then
  if Bus2IP_Reset = '1' then
    slv_reg0 <= ( others =>'0' );
    slv_reg1 <= ( others =>'0' );
    slv_reg2 <= ( others =>'0' );
    slv_reg3 <= ( others =>'0' );
    slv_reg4 <= ( others =>'0' );
    slv_reg5 <= ( others =>'0' );
    slv_reg6 <= ( others =>'0' );
    slv_reg7 <= ( others =>'0' );
  else
    case slv_reg_write_sel is
      when "10000000" =>
        for byte_index in 0 to ( C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg0(bytes_index*8 to byte_index*8+7) <= Bus2IP_Data(bytes_index*8 to byte_index*8+7);
          end if;
          end loop;
      when "01000000" =>
        for byte_index in 0 to ( C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg1(bytes_index*8 to byte_index*8+7) <= Bus2IP_Data(bytes_index*8 to byte_index*8+7);
          end if;
          end loop;
      when "00100000" =>
        for byte_index in 0 to ( C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg2(bytes_index*8 to byte_index*8+7) <= Bus2IP_Data(bytes_index*8 to byte_index*8+7);
          end if;
          end loop;
      when "00010000" =>
        for byte_index in 0 to ( C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg3(bytes_index*8 to byte_index*8+7) <= Bus2IP_Data(bytes_index*8 to byte_index*8+7);
          end if;
          end loop;
      when "00001000" =>
        for byte_index in 0 to ( C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg4(bytes_index*8 to byte_index*8+7) <= Bus2IP_Data(bytes_index*8 to byte_index*8+7);
          end if;
          end loop;
      when "00000100" =>
        for byte_index in 0 to ( C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg5(bytes_index*8 to byte_index*8+7) <= Bus2IP_Data(bytes_index*8 to byte_index*8+7);
          end if;
          end loop;
      when "00000010" =>
        for byte_index in 0 to ( C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg6(bytes_index*8 to byte_index*8+7) <= Bus2IP_Data(bytes_index*8 to byte_index*8+7);
          end if;
          end loop;
      when "00000001" =>
        for byte_index in 0 to ( C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg7(bytes_index*8 to byte_index*8+7) <= Bus2IP_Data(bytes_index*8 to byte_index*8+7);
          end if;
          end loop;
    when others => null;
  end case;
end if;
end if;
end process SLAVE_REG_WRITE_PROC;
-- implement slave model software accessible register(s) read mux

SLAVE_REG_READ_PROC : process(extTrigCount, RIIFO2IP_AlmostFull, RIIFO2IP_Full, sFIFOProgFull, 
sFIFOEmpty, sFIFOFull, stringerDaInv, slv_reg_read_sel, slv_reg0, slv_reg1, 
slv_reg2, slv_reg3, slv_reg4, slv_reg5, slv_reg6, slv_reg7, L_int) is

begin

case slv_reg_read_sel is

when "10000000" => slv_ip2bus_data <= slv_reg0;
when "01000000" => slv_ip2bus_data <= slv_reg1;
when "00100000" => slv_ip2bus_data <= slv_reg2;
when "00010000" => slv_ip2bus_data <= slv_reg3;
when "00001000" => slv_ip2bus_data <= \L_int(1) & \L_int(0) & Gen_C1 & Gen_C0 & "0000" & "0000" & "0000" & \striggerDaInv & "000" & "0000" & "0000" & "0000" & "0000";
when "00000100" => slv_ip2bus_data <= '0' & extTrigCount;
when "00000010" => slv_ip2bus_data <= slv_reg6;
when "00000001" => slv_ip2bus_data <= sFIFOFull & sFIFOEmpty & sFIFOProgFull & RIIFO2IP_Full & RIIFO2IP_AlmostFull & "000" & "0000" & "0000" & "0000" & "0000" & "0000" & "0000";
when others => slv_ip2bus_data <= (others => '0');
end case;

end process SLAVE_REG_READ_PROC;

-- ----------------------------------------
-- Example code to drive IP to Bus signals
-- ----------------------------------------

IP2Bus_Data <= slv_ip2bus_data when slv_read_ack = '1' else (others => '0');

IP2Bus_WrAck <= slv_write_ack;
IP2Bus_RdAck <= slv_read_ack;
IP2Bus_Error <= '0';

--
-- all output configuration bits must be connected here to slv_regx
--

L_int(2) <= slv_reg0(31); -- bit 0 for readout
L_int(3) <= slv_reg0(30);
rst_n <= slv_reg0(29);
triggerAns <= slv_reg0(28);

sHamDivisor(11) <= slv_reg0(27);
sHamDivisor(10) <= slv_reg0(26);
sHamDivisor(9) <= slv_reg0(25);
sHamDivisor(8) <= slv_reg0(24);
sHamDivisor(7) <= slv_reg0(23);
sHamDivisor(6) <= slv_reg0(22);
sHamDivisor(5) <= slv_reg0(21);
sHamDivisor(4) <= slv_reg0(20);
sHamDivisor(3) <= slv_reg0(19);
sHamDivisor(2) <= slv_reg0(18);
sHamDivisor(1) <= slv_reg0(17);
sHamDivisor(0) <= slv_reg0(16);

Gen_C2 <= slv_reg0(15);
Gen_C3 <= slv_reg0(14);

sAdcDivisor(5) <= slv_reg0(13);
sAdcDivisor(4) <= slv_reg0(12);
sAdcDivisor(3) <= slv_reg0(11);
sAdcDivisor(2) <= slv_reg0(10);
sAdcDivisor(1) <= slv_reg0(9);
sAdcDivisor(0) <= slv_reg0(8);
A.3 FPGA Logic

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity FPGA is
  generic (
    wordLength : integer :=32;
    dataLength : integer :=16;
    integrationTimeBitMax : integer :=8;
    channelLength : integer :=8;
    hamDivisorLength : integer :=12;
    adcDivisorLength : integer :=6;
    fifoDepth : integer :=2048);
  Port ( rst_n : in std_logic;
          trigger : in std_logic;
          FPGA_Clk : in std_logic;
          integTime : in std_logic_vector(integrationTimeBitMax-1 downto 0);
          FSM_Clk : in std_logic;
          Ham_Eos : in std_logic;
          Test : in std_logic;
          extTrigger : in std_logic;
          deseInput : in std_logic;
          divisorHam : in std_logic_vector(0 to hamDivisorLength-1);
          divisorAdc : in std_logic_vector(0 to adcDivisorLength-1);
          extFIFOFull : in std_logic;
          triggerDaInvRst_n : in std_logic;
          extTrigOut : out std_logic_vector(wordLength-2 downto 0);
          triggerDaInv : out std_logic;
          wordOut : out std_logic_vector(wordLength-1 downto 0);
          photodiodeCLKOut : out std_logic;
          photodiodeStart : out std_logic;
          ADC_CLKOut : out std_logic;
          intFIFOFull : out std_logic;
          );
```

```
end FPGA;
```

```vhdl
-- FPGA.vhd

entity FPGA is
  generic ( wordLength : integer :=32;
    dataLength : integer :=16;
    integrationTimeBitMax : integer :=8;
    channelLength : integer :=8;
    hamDivisorLength : integer :=12;
    adcDivisorLength : integer :=6;
    fifoDepth : integer :=2048);
  Port ( rst_n : in std_logic;
          trigger : in std_logic;
          FPGA_Clk : in std_logic;
          integTime : in std_logic_vector(integrationTimeBitMax-1 downto 0);
          FSM_Clk : in std_logic;
          Ham_Eos : in std_logic;
          Test : in std_logic;
          extTrigger : in std_logic;
          deseInput : in std_logic;
          divisorHam : in std_logic_vector(0 to hamDivisorLength-1);
          divisorAdc : in std_logic_vector(0 to adcDivisorLength-1);
          extFIFOFull : in std_logic;
          triggerDaInvRst_n : in std_logic;
          extTrigOut : out std_logic_vector(wordLength-2 downto 0);
          triggerDaInv : out std_logic;
          wordOut : out std_logic_vector(wordLength-1 downto 0);
          photodiodeCLKOut : out std_logic;
          photodiodeStart : out std_logic;
          ADC_CLKOut : out std_logic;
          intFIFOFull : out std_logic;
        );
```

```
end FPGA;
```

```vhdl
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity FPGA is
  generic ( wordLength : integer :=32;
    dataLength : integer :=16;
    integrationTimeBitMax : integer :=8;
    channelLength : integer :=8;
    hamDivisorLength : integer :=12;
    adcDivisorLength : integer :=6;
    fifoDepth : integer :=2048);
  Port ( rst_n : in std_logic;
          trigger : in std_logic;
          FPGA_Clk : in std_logic;
          integTime : in std_logic_vector(integrationTimeBitMax-1 downto 0);
          FSM_Clk : in std_logic;
          Ham_Eos : in std_logic;
          Test : in std_logic;
          extTrigger : in std_logic;
          deseInput : in std_logic;
          divisorHam : in std_logic_vector(0 to hamDivisorLength-1);
          divisorAdc : in std_logic_vector(0 to adcDivisorLength-1);
          extFIFOFull : in std_logic;
          triggerDaInvRst_n : in std_logic;
          extTrigOut : out std_logic_vector(wordLength-2 downto 0);
          triggerDaInv : out std_logic;
          wordOut : out std_logic_vector(wordLength-1 downto 0);
          photodiodeCLKOut : out std_logic;
          photodiodeStart : out std_logic;
          ADC_CLKOut : out std_logic;
          intFIFOFull : out std_logic;
        );
```

```
end FPGA;
```

```vhdl
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity FPGA is
  generic ( wordLength : integer :=32;
    dataLength : integer :=16;
    integrationTimeBitMax : integer :=8;
    channelLength : integer :=8;
    hamDivisorLength : integer :=12;
    adcDivisorLength : integer :=6;
    fifoDepth : integer :=2048);
  Port ( rst_n : in std_logic;
          trigger : in std_logic;
          FPGA_Clk : in std_logic;
          integTime : in std_logic_vector(integrationTimeBitMax-1 downto 0);
          FSM_Clk : in std_logic;
          Ham_Eos : in std_logic;
          Test : in std_logic;
          extTrigger : in std_logic;
          deseInput : in std_logic;
          divisorHam : in std_logic_vector(0 to hamDivisorLength-1);
          divisorAdc : in std_logic_vector(0 to adcDivisorLength-1);
          extFIFOFull : in std_logic;
          triggerDaInvRst_n : in std_logic;
          extTrigOut : out std_logic_vector(wordLength-2 downto 0);
          triggerDaInv : out std_logic;
          wordOut : out std_logic_vector(wordLength-1 downto 0);
          photodiodeCLKOut : out std_logic;
          photodiodeStart : out std_logic;
          ADC_CLKOut : out std_logic;
          intFIFOFull : out std_logic;
```
intFIFOEmpty : out std_logic;
extFifoWrEn : out std_logic;
busy : out std_logic;
adcPDEN : out std_logic;
ADCStart : out std_logic);
end FPGA;

architecture Behavioral of FPGA is

component LogicControlUnit
  generic ( width : integer;
dataLength : integer;
integrationTimeBitMax : integer;
channelLength : integer);
Port ( reset_n : in std_logic;
extTrigger : in std_logic;
clk : in std_logic;
integTime : in std_logic_vector (integrationTimeBitMax-1 downto 0);
Ham_Eos : in std_logic;
clkAdc : in std_logic;
clkHam : in std_logic;
Test : in std_logic;
trigger : in std_logic;
dataIn : in std_logic_vector (dataLength-1 downto 0);
dataReadyFromDeserializer : in std_logic;
triggerDaInvRst_n : in std_logic;
extTrigOut : out std_logic_vector (width-2 downto 0);
triggerDaInv : out std_logic;
adcStart : out std_logic;
photodiodeStart : out std_logic;
deserializerStart : out std_logic;
writeEnable : out std_logic;
adcPDEN : out std_logic;
wordOut : out std_logic_vector (width-1 downto 0));
end component;

component Deserializzatore
  generic ( width : integer);
Port ( reset_n : in STD_LOGIC;
clk : in STD_LOGIC;
cs_n : in STD_LOGIC;
datain : in STD_LOGIC;
ready : out STD_LOGIC;
dataout : out STD_LOGIC_VECTOR (width-1 downto 0));
end component;

component intFIFO
  port( wr : in std_logic;
       wr_clk : in std_logic;
       rd_clk : in std_logic;
       din : in std_logic_vector (31 downto 0);
       wr_en : in std_logic;
       rd_en : in std_logic;
       dout : out std_logic_vector (31 downto 0);
       full : out std_logic;
       empty : out std_logic;
       prog_full : OUT std_logic);
end component;

component FSM
  port( intEmpty : in std_logic;
       extFull : in std_logic;
       clock : in std_logic;
       reset_n : in std_logic;
       wr_en : out std_logic;
       rd_en : out std_logic);
end component;

component ClockDivider
  generic ( lengthDivisor : integer :=12);
Port ( reset_n : in std_logic;
       clkIn : in std_logic;
       divisor : in std_logic_vector (0 to lengthDivisor-1));
A.3 FPGA Logic

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 adcIntClk : std_logic;
123  signal hamIntClk : std_logic;
124
125  begin
126
127  LogicControlUnitlogic : LogicControlUnit generic map ( width => wordLength,
128    dataLength => dataLength,
129    integrationTimeBitMax => integrationTimeBitMax,
130    channelLength => channelLength)
131  port map ( reset_n => rst_n,
132    clk=> FPGA_Clk,
133    Ham_Eos=>Ham_Eos,
134    clkAdc=>adcIntClk,
135    clkHam=>hamIntClk,
136    integTime => integTime,
137    extTrigger=>extTrigger,
138    Test=>Test,
139    trigger=>trigger,
140    dataIn=>deserOut,
141    dataReadyFromDeserializer=>deserReady,
142    triggerDaInvRst_n=>triggerDaInvRst_n,
143    triggerDaInv=>triggerDaInv,
144    adcStart=>ADCStart,
145    photodiodeStart=>photodiodeStart,
146    deserializerStart=>deserializerStart,
147    writeEnable=>LogicWriteEn,
148    adcPDEN=>adcPDEN,
149    wordOut=>LogicWordOut);
150
151  Deserializzatorelogic : Deserializzatore generic map( width => dataLength)
152  port map ( reset_n => rst_n,
153    clk=>adcIntClk,
154    cs_n=>deserStart,
155    datain=>deseInput,
156    ready=>deserReady,
157    dataout=>deserOut);
158
159  ClockHamDividerlogic : ClockDivider generic map( lengthDivisor => hamDivisorLength)
160  port map ( reset_n => rst_n,
161    clkIn=>FPGA_Clk,
162    divisor=>divisorHam,
163    clkOut=>hamIntClk);
164
165  ClockADCDividerlogic : ClockDivider generic map( lengthDivisor => adcDivisorLength)
166  port map ( reset_n => rst_n,
167    clkIn=>FPGA_Clk,
168    divisor=>divisorADC,
169    clkOut=>adcIntClk);
170
171  FIFOlogic : intFIFO
172  port map ( rst=>not(rst_n),
173    wr_clk=>FPGA_Clk,
174    rd_clk=>FMO_Clk,
175    din=>LogicWordOut,
176    wr_en=>LogicWriteEn,
177    rd_en=>FIFOReadEn,
178    dout=>wordOut,
179    full=>intFIFOFull,
180    empty=>intFIFOEmpt,
181    prng_full=>busy);
182
183  FSMlogic : FSM port map ( intEmpty=>intFIFOEmpt,
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

LIBRARY XilinxCoreLib;

ENTITY intFIFO IS
PORT ( rst : IN STD_LOGIC;
wr_clk : IN STD_LOGIC;
rd_clk : IN STD_LOGIC;
din : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
wr_en : IN STD_LOGIC;
rd_en : IN STD_LOGIC;
dout : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
full : OUT STD_LOGIC;
empty : OUT STD_LOGIC;
prog_full : OUT STD_LOGIC);

ARCHITECTURE FIFO_a OF intFIFO IS

BEGIN

end Behavioral;

intFIFO.vhd
A.3 FPGA Logic

```vhdl
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_mgc_constraint => 0,
76  c_application_type_axis => 0,
77  c_application_type_rdch => 0,
78  c_application_type_vach => 0,
79  c_application_type_vdch => 0,
80  c_application_type_vrch => 0,
81  c_axi_addr_width => 32,
82  c_axi_aruser_width => 1,
83  c_axi_awuser_width => 1,
84  c_axi_buser_width => 1,
85  c_axi_data_width => 64,
86  c_axi_id_width => 64,
87  c_axi_type => 0,
88  c_axis_tdata_width => 64,
89  c_axis_tdest_width => 4,
90  c_axis_tkeep_width => 4,
91  c_axis_tstrb_width => 4,
92  c_axis_tuser_width => 4,
93  c_axis_type => 0,
94  c_common_clock => 0,
95  c_count_type => 0,
96  c_data_count_width => 11,
97  c_default_value => "BlankString",
98  c_din_width => 32,
99  c_din_width_axis => 1,
100 c_din_width_rach => 32,
101 c_din_width_rdch => 64,
102 c_din_width_vach => 32,
103 c_din_width_vdch => 64,
104 c_din_width_vrch => 2,
105 c_dout_rst_val => "0",
106 c_dout_width => 32,
107 c_enable_rlocs => 0,
108 c_enable_rst_sync => 1,
109 c_error_injection_type_axis => 0,
110 c_error_injection_type_rdch => 0,
111 c_error_injection_type_vach => 0,
112 c_error_injection_type_vdch => 0,
113 c_error_injection_type_vrch => 0,
114 c_family => "virtex5",
115 c_full_flags_rst_val => 1,
116 c_has_almost_empty => 0,
117 c_has_almost_full => 0,
118 c_has_axi_aruser => 0,
119 c_has_axi_awuser => 0,
120 c_has_axi_buser => 0,
121 c_has_axi_rd_channel => 0,
122 c_has_axi_wr_channel => 0,
123 c_has_axis_tdata => 0,
124 c_has_axis_tdest => 0,
```
c_has_axis_tid => 0,
c_has_axis_tkeep => 0,
c_has_axis_tlast => 0,
c_has_axis_tready => 1,
c_has_axis_tstrb => 0,
c_has_axis_tuser => 0,
c_has_backup => 0,
c_has_data_count => 0,
c_has_data_counts_axis => 0,
c_has_data_counts_rach => 0,
c_has_data_counts_rdch => 0,
c_has_data_counts_wach => 0,
c_has_data_counts_wdch => 0,
c_has_data_counts_wrch => 0,
c_has_int_clk => 0,
c_has_master_ce => 0,
c_has_meminit_file => 0,
c_has_overflow => 0,
c_has_mif_file_name => "BlankString",
c_has_meminit_file => 0,
c_has_overflow => 0,
c_has_overflow => 0,
c_has_overflow => 0,
c_has_program_flags_axis => 0,
c_has_program_flags_rach => 0,
c_has_program_flags_rdch => 0,
c_has_program_flags_wach => 0,
c_has_program_flags_wdch => 0,
c_has_program_flags_wrch => 0,
c_has_rd_data_count => 0,
c_has_rd_rst => 0,
c_has_rst => 1,
c_has_slave_ce => 0,
c_has_valid => 0,
c_has_wr_ack => 0,
c_has_wr_data_count => 0,
c_has_wr_rst => 0,
c_has_underflow => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
c_has_valid => 0,
A.3 FPGA Logic

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_synchromizer_stage => 2,
227    c_underflow_low => 0,
228    c_use_common_overlow => 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_valid_low => 0,
240    c_use_embedded_reg => 0,
241    c_use_fifo16_flags => 0,
242    c_use_fwft_data_count => 0,
243    c_valid_low => 0,
244    c_use_default_settings => 0,
245    c_use_dout_rst => 1,
246    c_use_ecc => 0,
247    c_use_ecc_axis => 0,
248    c_use_ecc_rach => 0,
249    c_use_ecc_rdch => 0,
250    c_use_ecc_wach => 0,
251    c_use_ecc_wdch => 0,
252    c_use_ecc_wrch => 0,
253    c_use_fifo16_flags => 0,
254    c_use_fwft_data_count => 0,
255    c_valid_low => 0,
256    c_use_default_settings => 0,
257    c_use_dout_rst => 1,
258    c_use_ecc => 0,
259    c_use_ecc_axis => 0,
260    c_use_ecc_rach => 0,
261    c_use_ecc_rdch => 0,
262    c_use_ecc_wach => 0,
263    c_use_ecc_wdch => 0,
264    c_use_ecc_wrch => 0,
265    c_valid_low => 0,
266    c_use_default_settings => 0,
267    c_use_dout_rst => 1,
268    c_use_ecc => 0,
269    c_use_ecc_axis => 0,
270    c_use_ecc_rach => 0,
271    c_use_ecc_rdch => 0,
272    c_use_ecc_wach => 0,
273    c_use_ecc_wdch => 0,
274    c_use_ecc_wrch => 0,
275    c_valid_low => 0,
276    c_use_default_settings => 0,
277    c_use_dout_rst => 1,
278    c_use_ecc => 0,
279    c_use_ecc_axis => 0,
280    c_use_ecc_rach => 0,
281    c_use_ecc_rdch => 0,
282    c_use_ecc_wach => 0,
283    c_use_ecc_wdch => 0,
284    c_use_ecc_wrch => 0,
285    c_valid_low => 0,
286    c_use_default_settings => 0,
287    c_use_dout_rst => 1,
288    c_use_ecc => 0,
289    c_use_ecc_axis => 0,
290    c_use_ecc_rach => 0,
291    c_use_ecc_rdch => 0,
292    c_use_ecc_wach => 0,
293    c_use_ecc_wdch => 0,
294    c_use_ecc_wrch => 0,
295    c_valid_low => 0,
296    c_use_default_settings => 0,
297    c_use_dout_rst => 1,
298    c_use_ecc => 0,
299    c_use_ecc_axis => 0,
300    c_use_ecc_rach => 0,
301    c_use_ecc_rdch => 0,
302    c_use_ecc_wach => 0,
303    c_use_ecc_wdch => 0,
304    c_use_ecc_wrch => 0,
305    c_valid_low => 0,
306    c_use_default_settings => 0,
307    c_use_dout_rst => 1,
308    c_use_ecc => 0,
309    c_use_ecc_axis => 0,
310    c_use_ecc_rach => 0,
311    c_use_ecc_rdch => 0,
312    c_use_ecc_wach => 0,
313    c_use_ecc_wdch => 0,
314    c_use_ecc_wrch => 0,
-- synthesis translate_on
END FIFO_a;

FSM.vhd

library ieee;
use ieee.std_logic_1164.all;

entity FSM is
port( intEmpty : in std_logic;
extFull : in std_logic;
clock : in std_logic;
reset_n : in std_logic;
wr_en : out std_logic;
rd_en : out std_logic);
end FSM;

architecture Behavioral of FSM is
begin

state_type is (S0, S1, S2, S3);
signal next_state, current_state : state_type;
begin

state_reg : process(clock, reset_n)
begin
if (reset_n = '0') then
current_state <= S0;
elsif (clock'event and clock = '1') then
current_state <= next_state;
end if;
end process;

comb_logic : process(current_state, intEmpty, extFull)
begin
case current_state is
when S0 => rd_en <= '0';
wr_en <= '0';
if (intEmpty = '0') then
next_state <= S1;
else
next_state <= S0;
end if;
when S1 => rd_en <= '1';
wr_en <= '0';
next_state <= S2;
when S2 => rd_en <= '0';
wr_en <= '0';
if (extFull = '0') then
next_state <= S3;
else
next_state <= S2;
end if;
when S3 => rd_en <= '0';
wr_en <= '1';
if (intEmpty = '0') then
next_state <= S1;
else
next_state <= S0;
end if;
when others =>
end case current_state is
end process;
end FSM;
Deserializzatore.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

device Deserializzatore is
  generic ( width : integer :=16);
  Port ( reset_n : in STD_LOGIC;
         clk : in STD_LOGIC;
         cs_n : in STD_LOGIC;
         datain : in STD_LOGIC;
         ready : out STD_LOGIC;
         dataout : out STD_LOGIC_VECTOR ( width-1 downto 0));
end Deserializzatore;

architecture Behavioral of Deserializzatore is
  type state_type is (S0, S1, S2, S3);
  signal current_state : state_type;
  signal clkCounterInternal : unsigned (4 downto 0);
  constant maxClkCounterInternal : unsigned (4 downto 0):= (4= > '0 ', others= > '1 ');
  signal outputInternal : unsigned ( width-1 downto 0);

begin
  comb_log: process(clk, reset_n)
  begin
    if (reset_n= '0 ')
      current_state <= S0;
      clkCounterInternal <= (others= > '0 ');
      outputInternal <= (others= > '0 ');
    elsif (clk=' event and clk='1 ')
      if (current_state = S0)
        clkCounterInternal <= (others= > '0 ');
        outputInternal <= (others= > '0 ');
        current_state <= S1;
      elsif (current_state = S1 and cs_n='0 ')
        if (clkCounterInternal + 1 = maxClkCounterInternal)
          current_state <= S2;
          outputInternal <= outputInternal+ 2**(width-3-TG_INTEGER(clkCounterInternal)));
        clkCounterInternal <= clkCounterInternal + 1;
      else
        clkCounterInternal <= clkCounterInternal + 1;
        end if;
      elsif (current_state = S2)
        current_state <= S3;
      elsif (current_state = S3)
        if (cs_n='1 ')
          clkCounterInternal <= (others= > '0 ');
          outputInternal <= (others= > '0 ');
          current_state <= S1;
          end if;
      end if;
    end if;
  end process;
end Behavioral;
end process;

output : process(current_state)
begin
  case current_state is
    when S0 => ready <= '0';
    dataout <= (others=>'0');
    when S2 => ready <= '1';
    dataout <= std_logic_vector(outputInternal);
    when others => ready <= '0';
    dataout <= std_logic_vector(outputInternal);
  end case;
end process;
end Behavioral;

ClockDivider.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ClockDivider is
generic ( lengthDivisor : integer :=12);
Port ( reset_n : in std_logic;
clkIn : in std_logic;
divisor : in std_logic_vector (lengthDivisor-1 downto 0);
clkOut : out std_logic);
end ClockDivider;

architecture Behavioral of ClockDivider is
signal clockState : std_logic := '0 ';
signal n : unsigned ( lengthDivisor-1 downto 0);

begin
  process(clkIn, reset_n)
  begin
    if(reset_n='0') then
      n(0)<'1';
      n(lengthDivisor-1 downto 1)<=('0'&unsigned(divisor(lengthDivisor-1 downto 1)));
      clockState<'1';
    elsif(rising_edge(clkIn)) then
      if(n='0' & unsigned(divisor(lengthDivisor-1 downto 1))) then
        clockState<'0';
      elsif (n(unsigned(divisor))) then
        clockState<'1';
      end if;
      if (n(unsigned(divisor))) then
        n(0)<'1';
        n(lengthDivisor-1 downto 1)<=('0'&unsigned(divisor(lengthDivisor-1 downto 1)));
      else
        n<=unsigned(n)+1;
      end if;
    end if;
  end process;
LogicControlUnit.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity LogicControlUnit is
  generic (
    width : integer;
    dataLength : integer;
    channelLength : integer;
    integrationTimeBitMax : integer
  );
  Port ( reset_n : in std_logic;
       extTrigger : in std_logic;
       clk : in std_logic;
       clkAdc : in std_logic;
       clkHam : in std_logic;
       Test : in std_logic;
       Ham_Eos : in std_logic;
       trigger : in std_logic;
       dataIn : in std_logic_vector (dataLength-1 downto 0);
       dataReadyFromDeserializer : in std_logic;
       integTime : in std_logic_vector (integrationTimeBitMax-1 downto 0);
       triggerDaInvRst_n : in std_logic;
       extTrigOut : out std_logic_vector (width-2 downto 0);
       triggerDaInv : out std_logic;
       adcStart : out std_logic;
       photodiodeStart : out std_logic;
       deserializerStart : out std_logic;
       writeEnable : out std_logic;
       adcPDEN : out std_logic;
       wordOut : out std_logic_vector (width-1 downto 0));
end LogicControlUnit;

architecture Behavioral of LogicControlUnit is

-- ------------------------------------ Component Declaration -------------------------

component TriggerAnalyzer
  Port ( clk : in std_logic;
       reset_n : in std_logic;
       trigger : in std_logic;
       rising : out std_logic;
       falling : out std_logic);
end component;

component ClkAnalyzer
  Port ( clk : in std_logic;
       reset_n : in std_logic;
       clkAdc : in std_logic;
       rising : out std_logic);
end component;

-- -----------------------------------------------------------------------------------

type state_type is (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12);

signal current_state : state_type;

signal extTrigOutInternal : unsigned (width-2 downto 0);
signal photodiodeStartCounterInternal : unsigned (integrationTimeBitMax-1 downto 0);
signal adcStartCounterInternal : unsigned (5 downto 0);
signal channelCounter : unsigned (channelLength-1 downto 0);

clkOut <= clockState;

end Behavioral;
constant maxExtTrigOutInternal : unsigned (width-2 downto 0) := (others=>'1');
class intermediateADCStartCounterInternal : unsigned (5 downto 0) := (6=>'1', others=>'0');
class maxChannelCounterInternal : unsigned (channelLength-1 downto 0) := (others=>'1');
--signal adcPDENInternal : std_logic;
signal triggerDAInvInternal : std_logic;
signal trigRising : std_logic;
signal trigFalling : std_logic;
signal extTrigRising : std_logic;
signal adcRising : std_logic;
signal hamRising : std_logic;
signal intermediateADCStartCounterReached : std_logic;
signal maxADCStartCounterReached : std_logic;
signal maxChannelCounterReached : std_logic;
signal maxExtTrigOutReached : std_logic;

begin
  TriggerAnalyzerLogic : TriggerAnalyzer port map(
    clk=>clk,
    reset_n=>reset_n,
    trigger=>trigger,
    rising=>trigRising,
    falling=>trigFalling);
  ExtTriggerAnalyzerLogic : ClkAnalyzer port map(
    clk=>clk,
    reset_n=>reset_n,
    clkAdc=>extTrigger,
    rising=>extTrigRising);
  AdcClkAnalyzerLogic : ClkAnalyzer port map(
    clk=>clk,
    reset_n=>reset_n,
    clkAdc=>clkAdc,
    rising=>adcRising);
  HamClkAnalyzerLogic : ClkAnalyzer port map(
    clk=>clk,
    reset_n=>reset_n,
    clkAdc=>clkHam,
    rising=>hamRising);

  integTimeReached_setting : process (clk)
  begin
    if (clk'event and clk='1') then
      if (std_logic_vector(photodiodeStartCounterInternal - 1) = integTime) then
        integTimeReached <= '1';
      else
        integTimeReached <= '0';
      end if;
    end if;
  end process;

  intermediateADCStartCounterReached_setting : process (clk)
  begin
    if (clk'event and clk='1') then
      if (adcStartCounterInternal = intermediateADCStartCounterInternal) then
        intermediateADCStartCounterReached <= '1';
      else
        intermediateADCStartCounterReached <= '0';
      end if;
    end if;
  end process;

  maxADCStartCounterReached_setting : process (clk)
  begin
    if (clk'event and clk='1') then
      if (adcStartCounterInternal = maxADCStartCounterInternal) then
        maxADCStartCounterReached <= '1';
      else
        maxADCStartCounterReached <= '0';
      end if;
    end if;
maxChannelCounterReached_setting : process (clk)
begin
if (clk' event and clk = '1') then
  if (channelCounter = maxChannelCounterInternal) then
    maxChannelCounterReached <= '1';
  else
    maxChannelCounterReached <= '0';
  end if;
end if;
end process;

maxExtTrigOutReached_setting : process (clk)
begin
if (clk' event and clk = '1') then
  if (extTrigOutInternal = maxExtTrigOutInternal) then
    maxExtTrigOutReached <= '1';
  else
    maxExtTrigOutReached <= '0';
  end if;
end if;
end process;

comb_log : process (clk, reset_n)
begin
if (reset_n = '0') then
  current_state <= S0;
  extTrigOutInternal <= (others => '0');
  photodiodeStartCounterInternal <= (others => '0');
  adcStartCounterInternal <= (others => '0');
  channelCounter <= (others => '1');
  --adcPDENInternal <= '0';
  triggerDaInvInternal <= '0';
elsif (clk' event and clk = '1') then
  case current_state is
    when S0 =>
      if (triggerDaInvRst_n = '0') then
        triggerDaInvInternal <= '0';
      else
        triggerDaInvInternal <= '1';
      end if;
      current_state <= S1;
    when S1 =>
      if (extTrigRising = '1') then
        channelCounter <= (others => '1');
        --adcPDENInternal <= '0';
        current_state <= S2;
      end if;
    when S2 =>
      if (Test = '1') then
        current_state <= S10;
      else
        current_state <= S3;
      end if;
    when S3 =>
      if (integTimeReached = '1') then
        photodiodeStartCounterInternal <= (others => '0');
        current_state <= S4;
      elsif (hamRising = '1') then
        photodiodeStartCounterInternal <= photodiodeStartCounterInternal +1;
      end if;
    when S4 =>
      if (triggerRising = '1') then
        current_state <= S5;
      end if;
  end case;
end if;
end process;
when S5 =>
  if (intermediateAdcStartCounterReached = '1') then
    if (maxChannelCounterReached = '1') then
      channelCounter <= (others => '0');
      end if;
    else
      channelCounter <= channelCounter + 1;
      end if;
  else
    current_state <= S6;
  end if;
when S6 =>
  if (maxAdcStartCounterReached = '1') then
    adcStartCounterInternal <= adcStartCounterInternal + 1;
    current_state <= S7;
  else
    adcRising = '1';
    adcStartCounterInternal <= adcStartCounterInternal + 1;
    end if;
when S7 => current_state <= S8;
when S8 =>
  if (trigFalling = '1') then
    -- if (Ham_Eos = '0') then
    --   adcPDENInternal <= '0';
    -- else
    --   adcPDENInternal <= '0';
    -- end if;
    current_state <= S9;
  end if;
when S9 =>
  if (intermediateAdcStartCounterReached = '1') then
    current_state <= S10;
  else
    adcRising = '1';
    adcStartCounterInternal <= adcStartCounterInternal + 1;
    end if;
when S10 =>
  if (maxAdcStartCounterReached = '1') then
    adcStartCounterInternal <= (others => '0');
    current_state <= S11;
  else
    adcRising = '1';
    adcStartCounterInternal <= adcStartCounterInternal + 1;
    end if;
when S11 =>
  if (Ham_Eos = '0') then
    current_state <= S12;
  else
    current_state <= S4;
    end if;
when S12 =>
  if (maxExtTrigOutReached = '1') then
    extTrigOutInternal <= (others => '0');
  else
    extTrigOutInternal <= (extTrigOutInternal + 1);
    end if;
    if (triggerDaInvRst_n = '0') then
    triggerDaInvInternal <= '0';
  else
    triggerDaInvInternal <= '1';
  end if;
  current_state <= S1;
end case;
end if;
end process;

output : process(current_state)
begin
  case current_state is
    when S0 => extTrigOut <= std_logic_vector(extTrigOutInternal);
    triggerDaInv <= triggerDaInvInternal;
    adcStart <= '1';
    photodiodeStart <= '0';
    deserializerStart <= '1';
    writeEnable <= '0';
    adcPDEN <= '0';
    wordOut <= (others=>'0');

    when S1 => extTrigOut <= std_logic_vector(extTrigOutInternal);
    triggerDaInv <= triggerDaInvInternal;
    adcStart <= '1';
    photodiodeStart <= '0';
    deserializerStart <= '1';
    writeEnable <= '1';
    adcPDEN <= '0';
    wordOut <= '1' & std_logic_vector(extTrigOutInternal);

    when S2 => extTrigOut <= std_logic_vector(extTrigOutInternal);
    triggerDaInv <= triggerDaInvInternal;
    adcStart <= '1';
    photodiodeStart <= '1';
    deserializerStart <= '1';
    writeEnable <= '1';
    adcPDEN <= '0';
    wordOut <= '1' & std_logic_vector(extTrigOutInternal);

    when S3 => extTrigOut <= std_logic_vector(extTrigOutInternal);
    triggerDaInv <= triggerDaInvInternal;
    adcStart <= '1';
    photodiodeStart <= '0';
    deserializerStart <= '1';
    writeEnable <= '0';
    adcPDEN <= '0';
    wordOut <= '1' & std_logic_vector(extTrigOutInternal);

    when S4 => extTrigOut <= std_logic_vector(extTrigOutInternal);
    triggerDaInv <= triggerDaInvInternal;
    adcStart <= '0';
    photodiodeStart <= '0';
    deserializerStart <= '0';
    writeEnable <= '0';
    adcPDEN <= '0';
    wordOut <= '1' & std_logic_vector(extTrigOutInternal);

    when S5 => extTrigOut <= std_logic_vector(extTrigOutInternal);
    triggerDaInv <= triggerDaInvInternal;
    adcStart <= '0';
    photodiodeStart <= '0';
    deserializerStart <= '0';
    writeEnable <= '0';
    adcPDEN <= '0';
    wordOut <= '1' & std_logic_vector(extTrigOutInternal);

    when S6 => extTrigOut <= std_logic_vector(extTrigOutInternal);
    triggerDaInv <= triggerDaInvInternal;
    adcStart <= '0';
    photodiodeStart <= '0';
    deserializerStart <= '0';
    writeEnable <= '0';
    adcPDEN <= '0';
    wordOut <= '1' & std_logic_vector(extTrigOutInternal);

    when S7 => extTrigOut <= std_logic_vector(extTrigOutInternal);
    triggerDaInv <= triggerDaInvInternal;
    adcStart <= '1';
    photodiodeStart <= '0';
    deserializerStart <= '1';
A.3 FPGA Logic

```vhdl
writeEnable <= '1';
adcPDEN <= '0';
wordOut <= "0"&"000000"&std_logic_vector(channelCounter) & '0' & dataIn;

when S8 => extTrigOut <= std_logic_vector(extTrigOutInternal);
triggerDalv <= triggerDalvInternal;
adcStart <= '1';
photodiodeStart <= '0';
deserializerStart <= '1';
writeEnable <= '0';
adcPDEN <= '0';
wordOut <= "0"&"000000"&std_logic_vector(channelCounter) & '0' & dataIn;

when S9 =>
  extTrigOut <= std_logic_vector(extTrigOutInternal);
  triggerDalv <= triggerDalvInternal;
  adcStart <= '0';
  photodiodeStart <= '0';
  deserializerStart <= '0';
  writeEnable <= '0';
  adcPDEN <= '0';
  wordOut <= "0"&"000000"&std_logic_vector(channelCounter) & '0' & dataIn;

when S10 =>
  extTrigOut <= std_logic_vector(extTrigOutInternal);
  triggerDalv <= triggerDalvInternal;
  adcStart <= '0';
  photodiodeStart <= '0';
  deserializerStart <= '0';
  writeEnable <= '0';
  adcPDEN <= '0';
  wordOut <= "0"&"000000"&std_logic_vector(channelCounter) & '0' & dataIn;

when S11 =>
  extTrigOut <= std_logic_vector(extTrigOutInternal);
  triggerDalv <= triggerDalvInternal;
  adcStart <= '1';
  photodiodeStart <= '0';
  deserializerStart <= '1';
  writeEnable <= '1';
  adcPDEN <= '0';
  wordOut <= "0"&"000000"&std_logic_vector(channelCounter) & '1' & dataIn;

when S12 =>
  extTrigOut <= std_logic_vector(extTrigOutInternal);
  triggerDalv <= triggerDalvInternal;
  adcStart <= '1';
  photodiodeStart <= '0';
  deserializerStart <= '1';
  writeEnable <= '1';
  adcPDEN <= '0';
  wordOut <= "010000000000001101111010101101";

when others =>
  extTrigOut <= std_logic_vector(extTrigOutInternal);
  triggerDalv <= triggerDalvInternal;
  adcStart <= '1';
  photodiodeStart <= '0';
  deserializerStart <= '1';
  writeEnable <= '0';
  adcPDEN <= '0';
  wordOut <= "0"&"000000"&std_logic_vector(channelCounter) & '0' & dataIn;

end case;
end process;
end Behavioral;

AdcClkAnalyzer.vhd

--------------------
-- AdcClkAnalyzer.vhd
--------------------
library IEEE;
```
A.3 FPGA Logic

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 begin
25 if(rising_edge(clk)) then
26 if(falled='1' and clkAdc='1' and rised='0') then
27 rising<='1';
28 rised<='1';
29 falled<='0';
30 elsif(rised='1' and clkAdc='0' and falled='0') then
31 falled<='1';
32 rising<='0';
33 rised<='0';
34 else
35 rising<='0';
36 end if;
37 end if;
38 end process;
39 
40 end Behavioral;

TriggerAnalyzer.vhd

1 -- TriggerAnalyzer.vhd
2 
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5 use IEEE.NUMERIC_STD.ALL;
6 
7 entity TriggerAnalyzer is
8 Port ( clk : in std_logic;
9 reset_n : in std_logic;
10 trigger : in std_logic;
11 rising : out std_logic;
12 falling : out std_logic);
13 end TriggerAnalyzer;
14 
15 architecture Behavioral of TriggerAnalyzer is
16 
17 signal rised : std_logic;
18 signal falled : std_logic;
19 
20 begin
21 process(clk)
22 begin
23 if (reset_n='0') then
24 rised<='0';
25 falled<='1';
26 falling<='0';
27 elsif(rising_edge(clk)) then
28 if(falled='1' and trigger='1' and rised='0') then
29 rising<='0';
30 falled<='1';
31 falling<='0';
32 elsif(falled='0' and trigger='0' and rised='1') then
33 rising<='1';
34 falled<='0';
35 falling<='0';
36 else
37 if(falled='1' and trigger='0' and rised='0') then
38 rised<='1';
39 falled<='1';
40 falling<='0';
41 elsif(falled='1' and trigger='1' and rised='0') then
42 rising<='0';
43 falled<='0';
44 else
45 rising<='1';
46 falled<='1';
47 end if;
48 end if;
49 end process;
50 
51 end Behavioral;
A.4 Test Bench

TestAdcClkAnalyzer.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY TestAdcClkAnalyzer IS
END TestAdcClkAnalyzer;

ARCHITECTURE behavior OF TestAdcClkAnalyzer IS
COMPONENT AdcClkAnalyzer
PORT(
  clk : IN std_logic;
  clkAdc : IN std_logic;
  rising : OUT std_logic
);
END COMPONENT;

signal clk : std_logic := '0';
signal clkAdc : std_logic := '0';
signal rising : std_logic := '0';
constant clk_period : time := 10 ns;
constant clkAdc_period : time := 25 ns;
BEGIN
uut : AdcClkAnalyzer PORT MAP (  
  clk => clk,
  clkAdc => clkAdc,
  rising => rising
);
clk_process : process
begin
  clk <= '0';
  wait for clk_period/2;
  clk <= '1';
  wait for clk_period/2;
  end process;
clkAdc_process : process
begin
end if;
end if;
else
  rising <= '0';
  falling <= '0';
end if;
end process;
end Behavioral;

A.4 Test Bench

TestAdcClkAnalyzer.vhd
A.4 Test Bench

-- Test Clock Divider

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using arithmetic functions with Signed or Unsigned values
-- USE ieee.numeric_std.ALL;

ENTITY TestClockDivider
IS
END TestClockDivider;

ARCHITECTURE behavior OF TestClockDivider IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT ClockDivider
PORT(
    reset_n : IN std_logic;
    clkIn : IN std_logic;
    divisor : IN std_logic_vector(0 to 11);
    clkOut : OUT std_logic
);
END COMPONENT;

-- Inputs
signal reset_n : std_logic := '0';
signal clkIn : std_logic := '0';
signal divisor : std_logic_vector(0 to 11) := (9=>>'1', 11=>>'1', others=>>'0');

-- Outputs
signal clkOut : std_logic;

-- Clock period definitions
constant clkIn_period : time := 10 ns;
constant clkOut_period : time := 10 ns;

BEGIN

uut : ClockDivider PORT MAP (
    reset_n => reset_n,
    clkIn => clkIn,
    divisor => divisor,
    clkOut => clkOut
);

clkIn_process : process
begin
    clkIn <= '0';

END;
wait for clkIn_period/2;
clkIn <= '1';
wait for clkIn_period/2;
end process;

stim_proc : process
begin
wait for 100 ns;
reset_n <= '1';
wait for clkIn_period*1000;
divisor(11)<= '1';
wait;
end process;
END;

TestDeserializzatore.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY TestDeserializzatore IS
END TestDeserializzatore;

ARCHITECTURE behavior OF TestDeserializzatore IS

COMPONENT Deserializzatore
generic ( width : integer := 16);
Port ( reset_n : in STD_LOGIC;
clk : in STD_LOGIC;
cs_n : in STD_LOGIC;
datain : in STD_LOGIC;
ready : out STD_LOGIC;
dataout : out STD_LOGIC_VECTOR (width-1 downto 0));
END COMPONENT;

signal reset_n : std_logic := '0';
signal clk : std_logic := '0';
signal cs_n : std_logic := '0';
signal datain : std_logic := '0';
signal ready : std_logic;
signal dataout : std_logic_vector(15 downto 0);
constant clk_period : time := 10 ns;

BEGIN

uut : Deserializzatore PORT MAP ( reset_n => reset_n,
clk => clk,
cs_n => cs_n,
datain => datain,
ready => ready,
dataout => dataout );

clk_process : process
begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
A.4 Test Bench

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

---

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY TestFIFO IS
END TestFIFO;

ARCHITECTURE behavior OF TestFIFO IS

COMPONENT intFIFO
PORT(
    rst : IN std_logic;
    wr_clk : IN std_logic;
    rd_clk : IN std_logic;
    din : IN std_logic_vector(31 downto 0);
    wr_en : IN std_logic;
    rd_en : IN std_logic;
    dout : OUT std_logic_vector(31 downto 0);
    full : OUT std_logic;
    empty : OUT std_logic;
    prog_full : OUT std_logic
);
END COMPONENT;

signal rst : std_logic := '0';
signal wr_clk : std_logic := '0';
```
A.4 Test Bench

```vhdl
signal rd_clk : std_logic := '0';
signal din : std_logic_vector(31 downto 0) := (others => '0');
signal wr_en : std_logic := '0';
signal dout : std_logic_vector(31 downto 0);
signal full : std_logic;
signal empty : std_logic;
signal prog_full : std_logic;
constant wr_clk_period : time := 10 ns;
constant rd_clk_period : time := 10 ns;
BEGIN
  uut : intFIFO PORT MAP (  
    rst => rst,  
    wr_clk => wr_clk,  
    rd_clk => rd_clk,  
    din => din,  
    wr_en => wr_en,  
    rd_en => rd_en,  
    dout => dout,  
    full => full,  
    empty => empty,  
    prog_full => prog_full  
  );
  wr_clk_process : process  
  begin  
    wr_clk <= '0';  
    wait for wr_clk_period /2;  
    wr_clk <= '1';  
    wait for wr_clk_period /2;  
    end process;
  rd_clk_process : process  
  begin  
    rd_clk <= '0';  
    wait for rd_clk_period /2;  
    rd_clk <= '1';  
    wait for rd_clk_period /2;  
    end process;
  stim_proc : process  
  begin  
    rst<= '1';  
    wait for 100 ns;  
    rst<= '0';  
    wait for wr_clk_period*10;  
    wr_en<= '1';  
    wait for wr_clk_period*2;  
    wr_en<= '0';  
    din<=(5=>'1', 2=>'1', others => '0');  
    wait for wr_clk_period*10;  
    wr_en<= '1';  
    wait for wr_clk_period;  
    wr_en<= '0';  
    wait for wr_clk_period*10;  
    wr_en<= '1';  
    wait for wr_clk_period;  
    wr_en<= '0';  
    wait for wr_clk_period*10;  
    wr_en<= '1';  
    wait for wr_clk_period;  
    wr_en<= '0';  
    wait for wr_clk_period*10;  
    wr_en<= '1';  
    wait for wr_clk_period;  
    rd_en<= '1';  
  end process;
```
TestFPGA.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY TestFPGA IS
END TestFPGA;

ARCHITECTURE behavior OF TestFPGA IS

COMPONENT FPGA
PORT(
  rst_n : IN std_logic;
  trigger : IN std_logic;
  FPGA_Clk : IN std_logic;
  extTrigger : IN std_logic;
  deseInput : IN std_logic;
  extFIFOFull : IN std_logic;
  wordOut : OUT std_logic_vector(31 downto 0);
  photodiodeClkOut : OUT std_logic;
  photodiodeStart : OUT std_logic;
  intFIFOFull : OUT std_logic;
  busy : OUT std_logic;
  ADCStart : OUT std_logic);
END COMPONENT;

BEGIN
  uut : FPGA PORT MAP ( 
    rst_n => rst_n ,
    trigger => trigger ,
    FPGA_Clk => FPGA_Clk ,
    extTrigger => extTrigger ,
    deseInput => deseInput ,
    extFIFOFull => extFIFOFull ,
    wordOut => wordOut ,
    photodiodeClkOut => photodiodeClkOut ,
    photodiodeStart => photodiodeStart ,
    intFIFOFull => intFIFOFull ,
    busy => busy ,
    ADCStart => ADCStart );

  constant FPGA_Clk_period : time := 10 ns;
  constant FSM_Clk_period : time := 8 ns;
  constant ADC_Clk_period : time := 25 ns;

END
A.4 Test Bench

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY TestFSM IS
END TestFSM;

刺激过程：
```
```
ARCHITECTURE behavior OF TestFSM IS

COMPONENT FSM

PORT(
    intEmpty : IN std_logic;
    extFull : IN std_logic;
    clock : IN std_logic;
    reset_n : IN std_logic;
    wr_en : OUT std_logic;
    rd_en : OUT std_logic
);

END COMPONENT;

signal intEmpty : std_logic := '1';
signal extFull : std_logic := '0';
signal clock : std_logic := '0';
signal reset_n : std_logic := '0';

-- Outputs
signal wr_en : std_logic;
signal rd_en : std_logic;

-- Clock period definitions
constant clock_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)
uc : FSM PORT MAP(
    intEmpty => intEmpty,
    extFull => extFull,
    clock => clock,
    reset_n => reset_n,
    wr_en => wr_en,
    rd_en => rd_en
);

-- Clock process definitions
clock_process : process
begin
    clock <= '0';
    wait for clock_period / 2;
    clock <= '1';
    wait for clock_period / 2;
end process;

-- Stimulus process
stim_proc : process
begin
    -- Hold reset state for 100 ns.
    wait for 100 ns;
    reset_n <= '1';
    wait for clock_period * 10;
    intEmpty <= '0';
    wait for clock_period * 20;
    extFull <= '1';
    wait for clock_period * 20;
    extFull <= '0';
    wait for clock_period * 20;
    intEmpty <= '1';
    wait for clock_period * 20;
    extFull <= '1';
end process;

END;

TestTriggerAnalyzer.vhd
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY TestTriggerAnalyzer IS
END TestTriggerAnalyzer;

ARCHITECTURE behavior OF TestTriggerAnalyzer IS

COMPONENT TriggerAnalyzer
Port ( clk : in std_logic;
trigger : in std_logic;
rising : out std_logic;
falling : out std_logic);
END COMPONENT;

signal trigger : std_logic := '0';
signal rising : std_logic;
signal clk : std_logic := '0';
signal falling : std_logic;
constant clk_period : time := 10 ns;
BEGIN

clk_process : process
begin
  clk <= '0';
  wait for clk_period/2;
  clk <= '1';
  wait for clk_period/2;
end process;

uut : TriggerAnalyzer PORT MAP (
  clk => clk,
  trigger => trigger,
  rising => rising,
  falling => falling
);

stim_proc : process
begin
  wait for 100 ns;
  trigger<= '1';
  wait for 50 ns;
  trigger<= '0';
  wait for 100 ns;
  trigger<= '1';
  wait for 50 ns;
  trigger<= '0';
  wait;
end process;

END;

TestUserlogic.vhd

-- Company:
-- Engineer:
-- Create Date: 16:07:29 06/26/2012
-- Module Name: /Network/Servers/eigen04.romal.tmfns.it/Users/guglielmogemignani/IseProject/TestUser_logic.vhd
-- Project Name: User_logic_2.0
-- Target Device:
-- Tool versions:
-- Description:
A.4 Test Bench

12 --
13 -- VHDL Test Bench Created by ISE for module: user_logic
14 --
15 -- Dependencies:
16 --
17 -- Revision: 0.01 - File Created
18 --
19 -- Notes:
20 --
21 --
22 --
23 -- This testbench has been automatically generated using types std_logic and
24 -- std_logic_vector for the ports of the unit under test. Xilinx recommends
25 -- that these types always be used for the top-level I/O of a design in order
26 -- to guarantee that the testbench will bind correctly to the post-implementation
27 -- simulation model.
28 --
29 library ieee;
30 use ieee.std_logic_1164.all;
31 use ieee.std_logic_arith.all;
32 use ieee.std_logic_unsigned.all;
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
49 PORT( 
50 Ham_Eos : IN std_logic;
51 Ham_Trigger : IN std_logic;
52 Ham_ClkIn : IN std_logic;
53 Ham_ClkOut : OUT std_logic;
54 Ham_Reset_n : OUT std_logic;
55 Ham_Gain : OUT std_logic;
56 Ham_InStart : OUT std_logic;
57 Adc_SDO : IN std_logic;
58 Adc_ClkOut : OUT std_logic;
59 Adc_CS_n : OUT std_logic;
60 Adc_PDEn : OUT std_logic;
61 Adc_ChSel : OUT std_logic;
62 Gen_Clk : IN std_logic;
63 Gen_L0_p : IN std_logic;
64 Gen_L0_n : IN std_logic;
65 Gen_L1_p : IN std_logic;
66 Gen_L1_n : IN std_logic;
67 Gen_L2_p : OUT std_logic;
68 Gen_L2_n : OUT std_logic;
69 Gen_L3_p : OUT std_logic;
70 Gen_L3_n : OUT std_logic;
71 Gen_C0 : IN std_logic;
72 Gen_C1 : IN std_logic;
73 Gen_C2 : OUT std_logic;
74 Gen_C3 : OUT std_logic;
75 Gen_ClkOut_p : OUT std_logic;
76 Gen_ClkOut_n : OUT std_logic;
77 BusZIP_Clk : IN std_logic;
78 BusZIP_Reset : IN std_logic;
79 BusZIP_Data : IN std_logic_vector(0 to 31);
80 BusZIP_RdCE : IN std_logic_vector(0 to 7);
81 BusZIP_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 );
87
88 END COMPONENT;
89
90 ARCHITECTURE implementation OF TestUser_Logic IS
91
92 COMPONENT user_logic
93
94 PORT( 
95 Ham_Eos : IN std_logic;
96 Ham_Trigger : IN std_logic;
97 Ham_ClkIn : IN std_logic;
98 Ham_ClkOut : OUT std_logic;
99 Ham_Reset_n : OUT std_logic;
100 Ham_Gain : OUT std_logic;
101 Ham_InStart : OUT std_logic;
102 Adc_SDO : IN std_logic;
103 Adc_ClkOut : OUT std_logic;
104 Adc_CS_n : OUT std_logic;
105 Adc_PDEn : OUT std_logic;
106 Adc_ChSel : OUT std_logic;
107 Gen_Clk : IN std_logic;
108 Gen_L0_p : IN std_logic;
109 Gen_L0_n : IN std_logic;
110 Gen_L1_p : IN std_logic;
111 Gen_L1_n : IN std_logic;
112 Gen_L2_p : OUT std_logic;
113 Gen_L2_n : OUT std_logic;
114 Gen_L3_p : OUT std_logic;
115 Gen_L3_n : OUT std_logic;
116 Gen_C0 : IN std_logic;
117 Gen_C1 : IN std_logic;
118 Gen_C2 : OUT std_logic;
119 Gen_C3 : OUT std_logic;
120 Gen_ClkOut_p : OUT std_logic;
121 Gen_ClkOut_n : OUT std_logic;
122 BusZIP_Clk : IN std_logic;
123 BusZIP_Reset : IN std_logic;
124 BusZIP_Data : IN std_logic_vector(0 to 31);
125 BusZIP_RdCE : IN std_logic_vector(0 to 7);
126 BusZIP_WrCE : IN std_logic_vector(0 to 7);
127 IP2Bus_Data : OUT std_logic_vector(0 to 31);
128 IP2Bus_RdAck : OUT std_logic;
129 IP2Bus_WrAck : OUT std_logic;
130 IP2Bus_Error : OUT std_logic;
131 );
132 END COMPONENT;
133
134 -- Component Definition for the UUT
135
136 ENTITY user_logic IS
137
138 END ENTITY user_logic;
139
140 ARCHITECTURE definition OF user_logic IS
141
142 SIGNAL Ham_Eos, Ham_Trigger, Ham_ClkIn, Ham_ClkOut, Ham_Reset_n, Ham_Gain, Ham_InStart, 
143 Adc_SDO, Adc_ClkOut, Adc_CS_n, Adc_PDEn, Adc_ChSel, Gen_Clk, Gen_L0_p, Gen_L0_n, Gen_L1_p, 
144 Gen_L1_n, Gen_L2_p, Gen_L2_n, Gen_L3_p, Gen_L3_n, Gen_C0, Gen_C1, Gen_C2, Gen_C3, 
145 Gen_ClkOut_p, Gen_ClkOut_n, BusZIP_Clk, BusZIP_Reset, BusZIP_Data, BusZIP_RdCE, 
146 BusZIP_WrCE, IP2Bus_Data, IP2Bus_RdAck, IP2Bus_WrAck, IP2Bus_Error; 
147
148 BEGIN
149
150 -- Entity body proceeds here
151
152 END ARCHITECTURE definition;
153
154 END ARCHITECTURE behavior;
155
156 END ENTITY TestUser_Logic;
157
158 ARCHITECTURE simulation OF TestUser_Logic IS
159
160 BEGIN
161
162 -- Simulating body proceeds here
163
164 END ARCHITECTURE simulation;
165
166 END;}
IP2RFIFO_WrReq : OUT std_logic;
IP2RFIFO_Data : OUT std_logic_vector(0 to 31);
RFIFOD2IP_WrAck : IN std_logic;
RFIFOD2IP_AlfMostFull : IN std_logic;
RFIFOD2IP_Vacancy : IN std_logic_vector(0 to log2(16384));
END COMPONENT;

-- Inputs
signal Ham_Eos : std_logic := '1';
signal Ham_Trigger : std_logic := '0';
signal Ham_ClkIn : std_logic := '0';
signal Adc_SDO : std_logic := '0';
signal Gen_Clk : std_logic := '0';
signal Gen_L0_p : std_logic := '0';
signal Gen_L0_n : std_logic := '0';
signal Gen_L1_p : std_logic := '0';
signal Gen_L1_n : std_logic := '0';
signal Gen_C0 : std_logic := '0';
signal Gen_C1 : std_logic := '0';
signal Bus2IP_Clk : std_logic := '0';
signal Bus2IP_Reset : std_logic := '0';
signal Bus2IP_Data : std_logic_vector(0 to 31) := (others => '0');
signal Bus2IP_BE : std_logic_vector(0 to 3) := (others => '0');
signal Bus2IP_RdCE : std_logic_vector(0 to 7) := (others => '0');
signal Bus2IP_WrCE : std_logic_vector(0 to 7) := (others => '0');
signal RFIFOD2IP_WrAck : std_logic := '0';
signal RFIFOD2IP_AlfMostFull : std_logic := '0';
signal RFIFOD2IP_Vacancy : std_logic_vector(0 to log2(16384)) := (others => '0');

-- Outputs
signal Ham_ClkOut : std_logic;
signal Ham_Reset_n : std_logic;
signal Ham_Gain : std_logic;
signal Ham_InStart : std_logic;
signal Adc_ClkOut : std_logic;
signal Adc_CS_n : std_logic;
signal Adc_PDEn : std_logic;
signal Adc_ChSel : std_logic;
signal Gen_L2_p : std_logic;
signal Gen_L2_n : std_logic;
signal Gen_L3_p : std_logic;
signal Gen_L3_n : std_logic;
signal Gen_C2 : std_logic;
signal Gen_C3 : std_logic;
signal Gen_ClkOut_p : std_logic;
signal Gen_ClkOut_n : std_logic;
signal IP2Bus_Data : std_logic_vector(0 to 31);
signal IP2Bus_RdACK : std_logic;
signal IP2Bus_WrACK : std_logic;
signal IP2Bus_Error : std_logic;
signal IP2RFIFO_WrReq : std_logic;
signal IP2RFIFO_Data : std_logic_vector(0 to 31);
constant Gen_Clk_period : time := 4 ns;
constant EXTDELAY : time := 1 ns;

BEGIN
-- Instantiate the Unit Under Test (UUT)
use user_logic PORT MAP (}
Ham_Eos => Ham_Eos,
Ham_Trigger => Ham_Trigger,
Ham_ClkIn => Ham_ClkIn,
Ham_ClkOut => Ham_ClkOut,
Ham_Reset_n => Ham_Reset_n,
Ham_Gain => Ham_Gain,
A.4 Test Bench

Ham_InStart => Ham_InStart,
Adc_SDO => Adc_SDO,
Adc_CLKOut => Adc_CLKOut,
Adc_CS_n => Adc_CS_n,
Adc_PDEn => Adc_PDEn,
Adc_ChSel => Adc_ChSel,
Gen_Clk => Gen_Clk,
Gen_L0_p => Gen_L0_p,
Gen_L0_n => Gen_L0_n,
Gen_L1_p => Gen_L1_p,
Gen_L1_n => Gen_L1_n,
Gen_L2_p => Gen_L2_p,
Gen_L2_n => Gen_L2_n,
Gen_L3_p => Gen_L3_p,
Gen_L3_n => Gen_L3_n,
Gen_C0 => Gen_C0,
Gen_C1 => Gen_C1,
Gen_C2 => Gen_C2,
Gen_C3 => Gen_C3,
Gen_CLK_out_p => Gen_CLK_out_p,
Gen_CLK_out_n => Gen_CLK_out_n,
Bus2IP_CLK => Bus2IP_CLK,
Bus2IP_Reset => Bus2IP_Reset,
Bus2IP_Data => Bus2IP_Data,
Bus2IP_BE => Bus2IP_BE,
Bus2IP_RdCE => Bus2IP_RdCE,
Bus2IP_WrCE => Bus2IP_WrCE,
IP2Bus_Data => IP2Bus_Data,
IP2Bus_RdAck => IP2Bus_RdAck,
IP2Bus_WrAck => IP2Bus_WrAck,
IP2Bus_Error => IP2Bus_Error,
IP2RFifo_WrReq => IP2RFifo_WrReq,
IP2RFifo_Data => IP2RFifo_Data,
RFIFO2IP_WrAck => RFIFO2IP_WrAck,
RFIFO2IP_AlmostFull => RFIFO2IP_AlmostFull,
RFIFO2IP_Full => RFIFO2IP_Full,
RFIFO2IP_Vacancy => RFIFO2IP_Vacancy;

Gen_CLK_process : process
begin
  Gen_CLK <= '1';
  wait for Gen_CLK_period/2;
  Gen_CLK <= '0';
  wait for Gen_CLK_period/2;
end process;

Bus2IP_CLK_process : process(Gen_CLK)
begin
  if(rising_edge(Gen_CLK)) then
    if (count_i2p = 1) then
      count_i2p <= 0;
    else
      count_i2p <= count_i2p +1;
    end if;
    if (count_i2p = 0) then
      Bus2IP_CLK <= NOT Bus2IP_CLK;
    end if;
  end if;
end process;

stim_proc : process
begin
  wait for Gen_CLK_period*20;
  Bus2IP_WrCE<="01000000";
  Bus2IP_BE(1)<="1";
  Bus2IP_BE(2)<="1";
  Bus2IP_BE(0)<="1";
  Bus2IP_BE(3)<="1";
  Bus2IP_Data(27)<="1";
  wait for Gen_CLK_period*20;
  Bus2IP_WrCE<="10000000";

A.4 Test Bench

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  wait for Gen_Clk_period *20;
244  Bus2IP_WrCE <= "10000000";
245  Bus2IP_Data(29) <= '1';
246  wait for Gen_Clk_period *2000;
247  Bus2IP_WrCE <= "10000000";
248  Bus2IP_Data(0) <= '0';
249  wait for Gen_Clk_period *20;
250  Bus2IP_WrCE <= "01000000";
251  Bus2IP_Data(11) <= '0';
252  Bus2IP_Data(0) <= '0';
253  wait for Gen_Clk_period *6396;
254  Gen_C0 <= '1' after EXTDELAY;
255  Ham_Trigger <= '1' after EXTDELAY;
256  wait for Gen_Clk_period *126;
257  Bus2IP_Data(12) <= '0';
258  Bus2IP_Data(11) <= '0';
259  wait for Gen_Clk_period *3579;
260  Ham_Trigger <= '1' after EXTDELAY;
261  wait for Gen_Clk_period *128;
262  Ham_Trigger <= '0' after EXTDELAY;
263  wait for Gen_Clk_period *250;
264  Ham_Trigger <= '1' after EXTDELAY;
265  wait for Gen_Clk_period *1000;
266  Ham_Trigger <= '1' after EXTDELAY;
267  wait for Gen_Clk_period *128;
268  wait for Gen_Clk_period *1000;
269  Ham_Trigger <= '1' after EXTDELAY;
270  wait for Gen_Clk_period *128;
271  Gen_C0 <= '0' after EXTDELAY;
272  Bus2IP_BE(1) <= '0';
273  Bus2IP_BE(2) <= '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  wait for Gen_Clk_period *22;
283  wait for Gen_Clk_period *1000;
284  Ham_Trigger <= '0' after EXTDELAY;
285  wait for Gen_Clk_period *128;
286 wait for Gen_Clk_period *100;
287  Ham_Trigger <= '0' after EXTDELAY;
288  wait for Gen_Clk_period *1000;
289  Ham_Trigger <= '1' after EXTDELAY;
290  wait for Gen_Clk_period *128;
291  wait for Gen_Clk_period *22;
292  wait for Gen_Clk_period *1000;
293  Ham_Trigger <= '0' after EXTDELAY;
294  wait for Gen_Clk_period *128;
295  wait for Gen_Clk_period *22;
296  wait for Gen_Clk_period *1000;
297  Ham_Trigger <= '1' after EXTDELAY;
298  wait for Gen_Clk_period *128;
299  wait for Gen_Clk_period *22;
300  wait for Gen_Clk_period *1000;
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  wait for Gen_Clk_period *22;
307  wait for Gen_Clk_period *22;
Adc_SDO<= '0' after EXTDELAY;
wait for Gen_Clk_period*100;
Ham_Trigger<= '0' after EXTDELAY;
wait for Gen_Clk_period*1000;
Ham_Trigger<= '1' after EXTDELAY;
wait for Gen_Clk_period*128;
Adc_SDO<= '1' after EXTDELAY;
wait for Gen_Clk_period*22;
Adc_SDO<= '0' after EXTDELAY;
wait for Gen_Clk_period*100;
Ham_Trigger<= '0' after EXTDELAY;
Ham_Eos<= '0' after EXTDELAY;
wait for Gen_Clk_period*250;
Ham_Eos<= '1' after EXTDELAY;
wait;
end process;
END;
Bibliography


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


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


