

Alma Mater Studiorum · University of Bologna

SCHOOL OF ENGINEERING AND ARCHITECTURE · BOLOGNA
DEPARTMENT OF ELECTRICAL, ELECTRONIC AND INFORMATION ENGINEERING
- GUGLIELMO MARCONI -

LAUREA MAGISTRALE IN
INGEGNERIA ELETTRONICA

**DEVELOPMENT OF AN ENHANCED TRANSFER
DATA CHANNEL FOR A HYBRID SOC FPGA
USED IN A DAQ SYSTEM AIMED AT
IMPROVING HADRON THERAPY PROTOCOLS**

SUPERVISOR:

**CHIAR.MO PROF
MAURO VILLA**

CO-ADVISOR:

**DOTT.SSA
SILVIA BIONDI**

CANDIDATE:

ANDREA SAVARESE

SESSION I
ACADEMIC YEAR 2019/2020

Contents

| | |
|---|-----------|
| Sommario | 1 |
| Abstract | 3 |
| Introduction | 5 |
| 1 Tumors and hadrontherapy | 7 |
| 1.1 Tumors | 7 |
| 1.1.1 Historical background | 7 |
| 1.1.2 Classification | 8 |
| 1.1.3 Pathogenesis | 9 |
| 1.1.4 Causes and diffusion | 10 |
| 1.1.5 Treatment | 11 |
| 1.2 Hadrontherapy | 13 |
| 1.2.1 Features: operation, advantages and disadvantages | 14 |
| 1.2.2 Therapeutic beams | 18 |
| 1.2.3 Relative Biological Effectiveness (RBE) and Linear Energy Transfer (LET) | 19 |
| 1.2.4 Nuclear fragmentation | 19 |
| 2 The FOOT experiment | 23 |
| 2.1 Configuration | 24 |
| 2.1.1 Electronic Setup | 25 |
| 2.1.2 Emulsion Setup | 30 |
| 2.1.3 Control system and interfaces | 31 |

| | | |
|----------|---|-----------|
| 2.2 | Future previsions and other considerations | 32 |
| 3 | Acquisition system for the Microstrip Silicon Detector | 35 |
| 3.1 | The Microstrip Silicon Detector | 35 |
| 3.2 | Operation | 38 |
| 3.3 | The DE10-Nano board | 39 |
| 3.4 | FPGA and Hard Processor System | 42 |
| 3.4.1 | The HPS–FPGA Interfaces | 42 |
| 3.4.2 | The Platform Designer | 43 |
| 3.4.3 | Avalon–MM Interface | 44 |
| 3.5 | FPGA DAQ firmware | 48 |
| 3.5.1 | The finite state machine | 48 |
| 3.5.2 | The acquisition | 49 |
| 3.5.3 | Communication and data retrieval | 51 |
| 3.5.4 | SoC_system | 52 |
| 3.5.5 | Debug entities | 54 |
| 3.6 | DDR3_manager | 55 |
| 3.7 | HPS software | 59 |
| 3.8 | Reserved memory | 63 |
| 3.9 | Additional feature: ADC | 65 |
| 3.10 | Synthesis, compilation and testing | 66 |
| 4 | Performance | 77 |
| | Conclusioni | 85 |
| | Appendix A | 87 |
| | Bibliography | 93 |

Sommario

In questa tesi è presentato il lavoro svolto su un sistema di acquisizione utilizzato in un esperimento di fisica nucleare facente parte del progetto *FOOT*, volto ad ottenere ulteriori informazioni sulla frammentazione nucleare per migliorare i protocolli medici di adroterapia e le metodologie di radioprotezione spaziale. Il sistema si basa su una scheda Terasic DE10-Nano che monta un SoC FPGA Cyclone V. L'obiettivo principale del lavoro è stato aumentare il throughput del trasferimento dei dati acquisiti dai sensori verso la memoria principale: a tal fine è stata utilizzata direttamente la memoria RAM del processore integrato come buffer circolare temporaneo. È stata inoltre implementata l'interfaccia (realizzata dall'Università di Perugia) per la sensoristica e un controller per l'ADC della scheda. Il lavoro ha compreso sia lo sviluppo del firmware, quindi VHDL e Platform Designer, sia del software, con la scrittura di funzioni in C++ per l'interfacciamento all'hardware. È stata inoltre necessaria una modifica al Device Tree del kernel del sistema operativo Linux presente sul SoC. Il sistema è stato simulato e testato in laboratorio con esito positivo. La scheda DE10-Nano vanta un banda di trasmissione massima teorica di 60 MB/s, che però scende a circa 10 MB/s quando la scheda è installata nel sistema completo di acquisizione, limitazione dovuta a fattori esterni alla scheda, come lo stato della rete del laboratorio e l'overhead degli altri componenti. Questi risultati sono eccellenti e, inoltre, il massimo throughput di 60 MB/s supporterà future ottimizzazioni del sistema senza creare colli di bottiglia per gli altri dispositivi. Operazioni di ottimizzazione sull'infrastruttura sono tutt'ora in corso, quindi ci si aspetta un ulteriore incremento della performance in un vicino futuro.

Abstract

This thesis presents the work carried out on an acquisition system used in an experiment of nuclear physics which is part of the *FOOT* project, aimed at obtaining further information on nuclear fragmentation to improve medical protocols of hadrontherapy and deep space radiation protection methodologies. The system is based on a Terasic DE10-Nano board which mounts a Cyclone V FPGA SoC. The main focus of the work was to increase the throughput of the transfer of the data acquired by the sensors to the main memory: for this purpose, the RAM memory of the integrated processor was used as a temporary circular buffer. In addition, the interface (made by the University of Perugia) for the sensors and a controller for the on-board ADC were implemented. The work included both the development of the firmware, therefore VHDL and Platform Designer, and of the software. In fact, writing functions in C++ was necessary in order to allow the software to access hardware resources. Furthermore, a change to the device tree of the Linux Kernel running on the SoC was required. The system was simulated and successfully tested in laboratory. The DE10-Nano board boasts a theoretical maximum transmission bandwidth of 60 MB/s, which decreases to about 10 MB/s when the board is installed in the complete acquisition system, a limitation due to factors external to the board, such as the state of the laboratory network and the overhead of the other components. These results are excellent and will support future development without becoming a bottleneck for the other devices. Infrastructure optimization operations are still ongoing, thus further performance improvements are expected in the near future.

Introduction

The average lifespan has lengthened worldwide in the last decades, thanks to the continuous progress of medicine and the growing attention to personal safety. However, some diseases, that once did not have many opportunities to manifest, are now much more widespread. Among these, there are certainly tumors. In 2018 approximately 18.1 million malignant tumors were diagnosed and 9.6 million people died because of that. Medicine has been fighting this disease for a long time with high effort and increasingly safer and more effective methods of treatment are being discovered. One of the most innovative treatments is hadrontherapy which has been shown to have clear advantages over radiotherapy, both because it damages less healthy tissues and because it is more effective against types of radioresistant tumors. Unfortunately, the experimental data available are not sufficient for the application of a standard treatment protocol and therefore a lot of time is invested in the analysis of the single case. This methodology wastes a lot of money, efforts and time. One of the phenomenon which is not fully understood yet is the *fragmentation*. The *FOOT* experiment has been proposed to solve this questions. The *FOOT* experiment will consist in bombarding targets with proton beams or light nuclei used in hadrontherapy and evaluating the resulting products (fragments) due to nuclear interactions. The energy and quantity of these fragments are not only important for medical research, but also for the development of future radio protection shields for deep space exploration.

The main objective of the work described in this thesis is the development of an enhanced high-speed data channel for a board, based on an Intel SoC FPGA, used in the acquisition system of the microstrip silicon detector. The aim of this board is to acquire data from the sensors in specific conditions, elaborate and send them to the central management system of the experiment.

In chapter 1, tumors will be described from a biological and medical point of view, pointing out causes, development and the main used treatment, exploring more in depth hadrontherapy. In chapter 2, the *FOOT* experiment will be discussed in detail. More precisely, the objectives of the experiment, the single components and their configurations will be explained, including their operation and their main characteristics. In chapter 3, the internal architecture and operation of the acquisition board will be explained, including the firmware, the software and the simulation and final testing of the system. In addition, it will be included a detailed overview of the tool used to design the system, the hardware of the board and an overview concerning the actual microstrip silicon detector. Finally, in chapter 4, the performance of the board alone and integrated into the DAQ system will be discussed.

Chapter 1

Tumors and hadrontherapy

1.1 Tumors

A neoplasm is

«an abnormal mass of tissue the growth of which exceeds and is uncoordinated with that of the normal tissues and persists in the same excessive manner after the cessation of the stimuli which evoked the change.»¹ [1]

Fundamental to the origin of all neoplasms are heritable (genetic) changes that allow excessive and unregulated proliferation that is independent of physiologic growth-regulatory stimuli (see figure 1.1) [1]. It is important to know that the word "tumor" refers to the mass that often appears where a neoplasm is present. There are also neoplasm that doesn't form a "tumor" like leukemia.

1.1.1 Historical background

The term "tumor" comes from the latin *tumor*, which means "swelling", while the term "cancer" comes from Hippocrates [2] who called it with this name because of its physical appearance resembling a crab. Neoplasms were always present in human history: the most ancient existing written documentation dates back to about 3000 b. C. and it is found in the Edwin Smith papyrus, the most ancient

¹Definition given by oncologist Rupert Allan Willis, accepted internationally

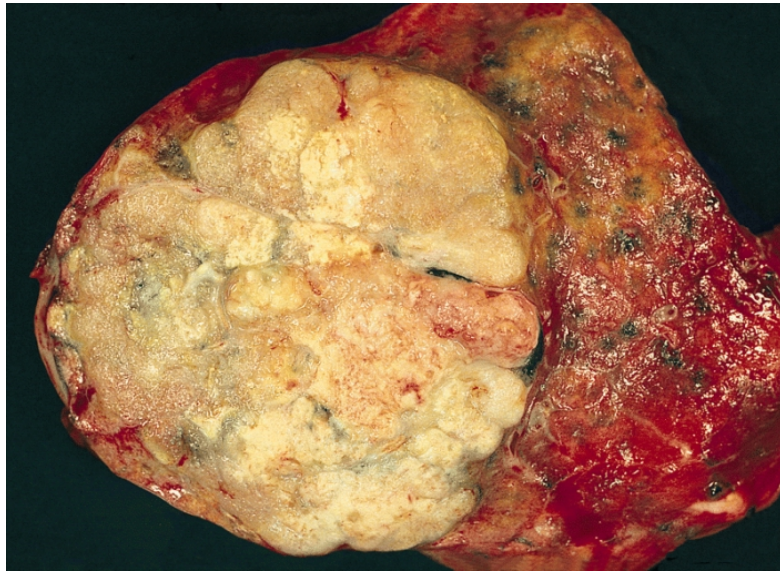


Figure 1.1: Pulmonary adenocarcinoma, a form of malignant neoplasm. Note the large peripheral mass, lobulated and translucent in appearance [3].

medical treaty that we have, in which the breast tumor is described [4]. There are even fossilized evidences of neoplasms: the most ancient case is an osteosarcoma (malignant tumor of the bones) which affected a hominid 1.7 million years ago [5].

1.1.2 Classification

Based on biologic behavior, tumors can be classified as:

- **benign**

their characteristic is their "expansive" behavior, that is, while maintaining some characteristics and functionality of the nearby healthy cells, they grow in an abnormal way, squeezing boundary tissues without damaging them directly²;

- **malignant or "cancer"**

these tumors are composed by cells that have lost every characteristics of the original tissue. Because of their abnormal and uncontrollable multiplication

²Even so, this doesn't mean that they are harmless.

rate, they infiltrate adjacent tissues until every healthy cells is replaced by cancerous material;

- **borderline**

neoplasms with an uncertain behavior or in middle ground between the above two categories.

1.1.3 Pathogenesis

A healthy cell, in order to become cancerous, needs to suffer genetic alterations that damage its duplication control system. The great majority of tumorous cells presents widespread damage to its chromosomal makeup. These damages span from changes of portions of DNA to complete absence (or overabundance) of specific chromosomes. These changes lead the cell to reproduce in an uncontrollable and pointless rate, squeezing nearby cells and altering its biological functions. A cell requires the following mutations to become a full-fledged cancer (malignant neoplasm):

- acquisition of a multiplicative autonomy as a result of permanent incapacity to submit to regulatory mechanisms of the organism,
- insensibility to multiplication inhibition caused by cellular density,
- reduced adhesion to other cells and tissues,
- digestion of extra cellular matrix, in order to promote proliferation,
- making of new blood vessels to supply oxygen and nutrients,
- loss of differentiation capacity,
- acquisition of unlimited replication ability,
- impossibility of programmed auto-induced death (apoptosis),
- loss of contact inhibition.

When these mutations occur, the tissue can be called "tumorous" and can expand and incorporate nearby tissues until reaching blood vessels and lymph nodes. From these locations the tumor is free to roam the entire host body. This process is called metastasis. The cancerous cells that detach from the main mass to travel to other places via blood vessels and lymph nodes are called *metastatic cells*. They are the origin of new tumorous masses far away from the main tumor called metastasis [6]. In figure 1.2 the different phases of the development of a neoplasm are shown.

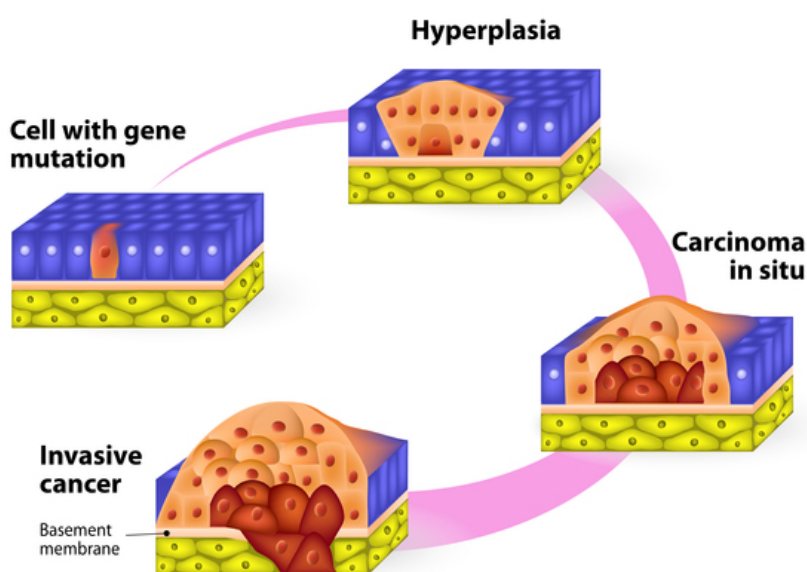


Figure 1.2: Development phases of cancer [7].

1.1.4 Causes and diffusion

Understanding the leading causes of a tumor in a single individual is, at the moment of writing, practically impossible. This is due to the fact that a person has a small probability to develop the illness because of a multitude of different factors: we can say, for example, that a smoker who has developed a pulmonary tumor has become ill *probably* because of his addiction, however this cannot be said with

absolute certainty as there are a lot of other factors that could have caused it (even if with a much lower probability). Neoplasms are generally environmental diseases with 90-95% of cases attributable to environment factors and 5-10% to genetic factors [8]. There is a large amount of environment causes which could increase the risk of cancer. Among them the most important are pollution, smoke (25-30%), unhealthy food and obesity (30-35%), infections (15-20%), lack of physical activity, ionizing radiations and even stress [8]. Some substances can increase the chances of a mutation at the genetic level and, for this reason, are called *mutagenic*: among them there are the compounds present in cigarette smoke, alcohol, benzene and asbestos. Even obesity and bad food habits can be causes of neoplasms: for example, diets low in fruit and vegetables increase chances of becoming ill. Other sources of cancer are infections that are responsible for a big part of the deaths in the most developed countries. Among the viruses capable of developing tumors, the *papillomavirus* and hepatitis B and C viruses emerge. Furthermore, an additional cause is represented by ionizing radiations which can have environmental origin, such as UV radiation from the sun and radon gas, or human origin, such as X and γ rays used for medical purposes. The latter are even used to treat neoplasms and this can lead, in some cases, to develop a *secondary tumor*.

According to recent estimates, 18.1 million new cases of cancer were diagnosed in the year 2018 in the world and 9.6 million people died because of that [9]. Over the years the cases of malignant neoplasm have increased mainly because of the growing aging population [10] and, in fact, one of the elements that favor the development of this disease the most is precisely old age, since a longer life span corresponds to a greater probability that a cell undergoes the mutations necessary to transform it into a tumor. With the further increase in life expectancy, we can only expect a further increase in the rate developments of malignancies.

1.1.5 Treatment

Cancer is a disease that is particularly difficult to treat also because of the large amount of forms which it can occur with. For this reason, experimental medicine is always looking for new increasingly effective and less invasive treatments. One of the biggest challenges lies in the ability to selectively and precisely target diseased

cells while leaving healthy ones intact, since they belong to the same organism and therefore have no substantial differences (except for those listed in the previous sections). There are various treatment methods available for a patient, whose adoptability depends on various factors such as, for example, the type of tumor, the age and the stage of progress of the disease:

- **surgery:**

the patient is accompanied to the operating room for the physical removal of the tumor mass. This operation can be aimed both at a total eradication of the mass, especially as regards the treatment of benign tumors confined on site and more easily reachable (for example skin and breast tumors), and a biopsy of the mass, for diagnostic purposes and in order to check the stage of advancement and/or evaluate further treatment techniques. It is not applicable in the case of eye, brain or inner ear tumors [11];

- **cancer chemotherapy:**

the patient is given medicines that can kill diseased cells. Based on the principle that cancer cells reproduce much faster than normal ones, the substances used for these treatments interfere with the mechanisms related to cell replication, killing them during this process (cytotoxic action), wherever they are found. Unfortunately, this method is one of the most debilitating because of the very high toxicity of the substances used. Furthermore, not all cancers are vulnerable to this treatment (chemoresistant tumors) [12];

- **immunotherapy:**

it is a type of experimental biological therapy, that is a type of treatment that uses substances made from living organisms to treat cancer. Generally, Immunotherapy helps the immune system to better act against cancer [13];

- **radiotherapy:**

the tumor is irradiated by ionizing radiations. X-rays and γ radiations are used which are capable of killing the cells, further and heavily damaging their chromosomes. The problem with this treatment is that it is not selective and as a result it also damages healthy cells in the area surrounding the tumor; in some cases where there would be damage to nearby vital organs,

the treatment is unusable. For these reasons, this treatment increases the chances of another tumor (called *secondary tumor*) emerging [14][15]. The figure 1.3 shows an example of radiotherapy equipment;



Figure 1.3: Radiotherapy equipment [16].

- **hadrontherapy:**

the tumor is bombarded by hadrons, that are protons or nuclei of heavier elements such as carbon or oxygen. The following section will be dedicated to this treatment.

1.2 Hadrontherapy

Hadrontherapy is a latest generation technique for the treatment of tumors which consists in bombarding the area affected by the disease with hadrons³, that are composite particles formed by quarks held together by strong interaction. This new technique was first proposed by the American physicist Robert Wilson in 1946 and, still today, it is the subject of research by many scientists, doctors and

³from the Greek *adrós*, that is "strong"

physicists in the world for its enormous potential. There are currently over 100 centers for hadrontherapy in the world and another 31 are under construction, a total of over 130,000 patients already treated [17].

1.2.1 Features: operation, advantages and disadvantages

The principle of operation of radiotherapy and hadrontherapy is the same. When an X or γ ray interacts with matter, it produces electrons which are given a fraction of the energy of the initial ray. Since the electrons thus produced are charged, in their motion they can ionize other atoms or molecules, breaking their chemical bonds. The hadrons used in hadrontherapy are also charged and therefore in their path inside the matter they can ionize atoms or molecules. Since the initial energies of electrons or hadrons are of the order of MeV (millions of electronvolts), while the binding energies of the molecules are of the order of electronvolts, each radiation or hadron has the possibility of breaking millions of molecules before being stopped in the body. When a DNA molecule is damaged (directly or indirectly) as shown in figure 1.4, it can lose its reproductive or functional capacity and therefore the growth of the tumor mass can be stopped. Since there is a direct link between ionization and energy loss of the particles, it is preferred to measure the radiation damage capacity in terms of energy lost (or deposited) in the tissues.

Although the operating principle is the same, the main feature that differentiates hadrontherapy from radiotherapy is precisely the different distribution of energy deposition as a function of the depth of penetration into the tissues. In radiotherapy most of the energy is released in the most superficial part of the tissue and then decays with an exponential trend with increasing depth. In this way, most of the energy hits the healthy cells placed in the tissues before the tumor and only a small percentage actually damages the diseased tissue (blue line in figure 1.5). On the contrary, in hadrontherapy, the distribution of energy is very different, since most of it is released in a small region called *Bragg peak* (green and red lines in figure 1.5), the position of which depends on the energy of the particle beam (figure 1.6). For this reason, hadrontherapy is much more selective than traditional radiotherapy, targeting almost only cancerous tissues and leaving healthy ones "intact", as shown in figure 1.7.

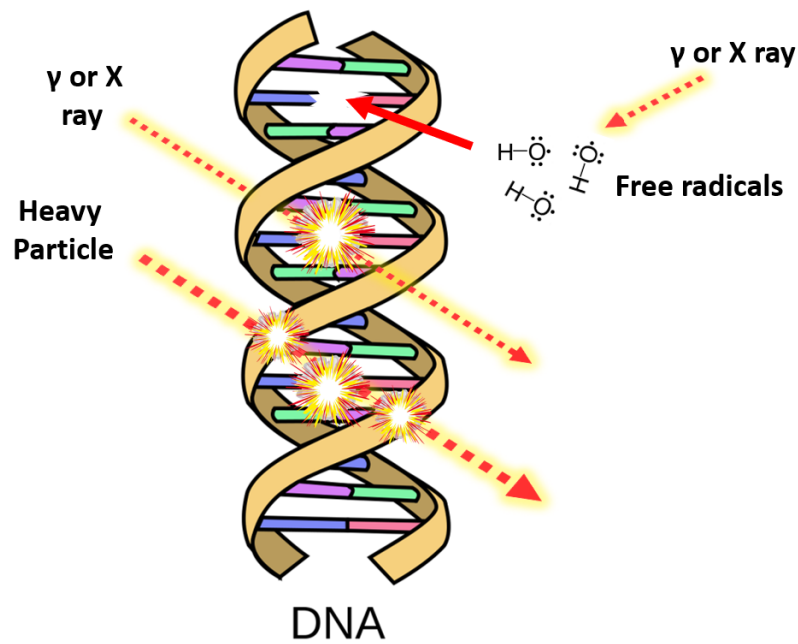


Figure 1.4: A visual representation of the possible ways in which DNA can be damaged.

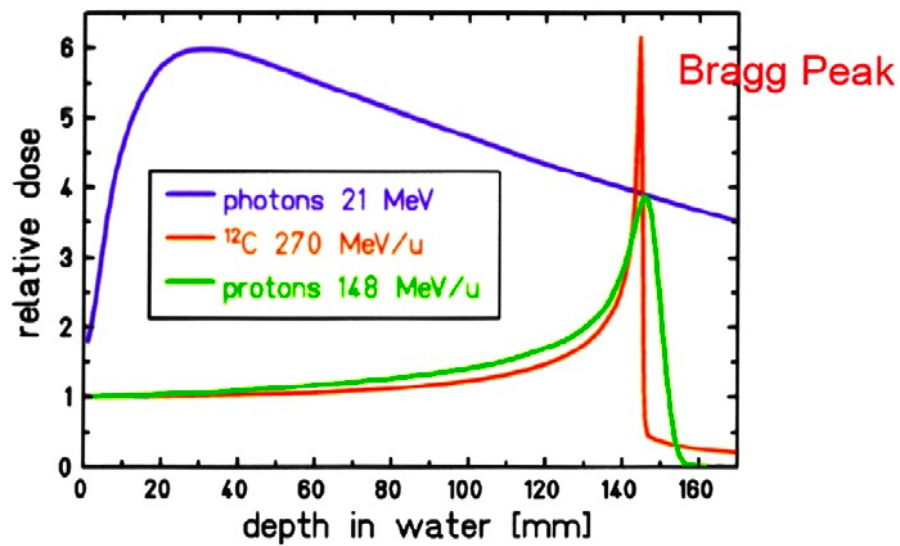


Figure 1.5: Graph showing relative dose when photons (normal radiotherapy), protons and ^{12}C are used [14].

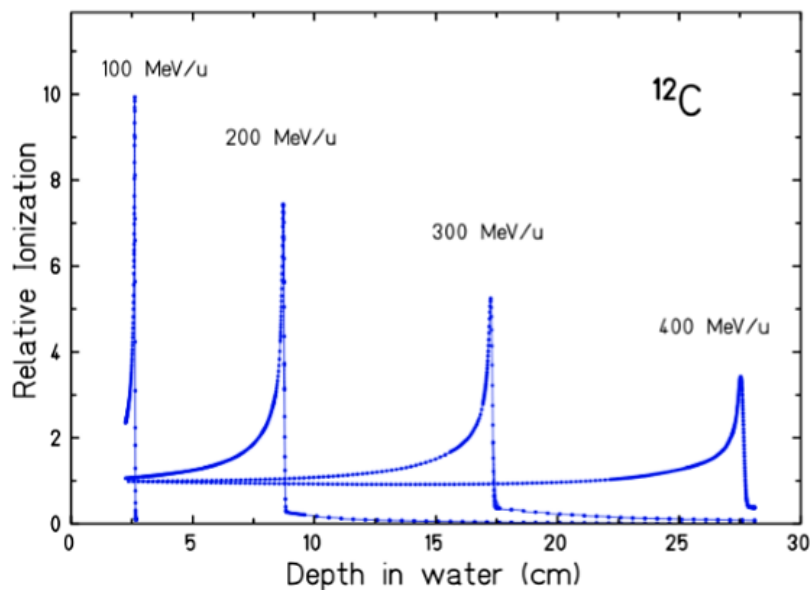


Figure 1.6: Relative ionization curves as a function of the depth of penetration caused by beams of ^{12}C at various energies [14].

A further advantage of this new treatment is the high efficacy against hypoxic tumors, which are tumors developed in an oxygen deficient environment. In fact, while ionizing radiation kills cancer cells mostly indirectly through the formation of ROS free radicals⁴ which require the presence of the oxygen, hadrons (especially the heaviest ones) can directly damage the DNA chains, making the oxygenation state of the tissue irrelevant [14][15][18][19].

Unfortunately, there are also disadvantages in using this technique. The most significant problems are the costs of equipment and therapy as powerful particle accelerators are needed, such as the one built by the CNAO foundation⁵ in Pavia (figure 1.8), while for radiation therapy, small, less bulky and lower cost accelerators are sufficient. This also leads to a lower availability. In fact, there are currently only two other hadrontherapy centers in Italy in addition to the CNAO: in Catania, at the laboratories of the National Institute for Nuclear Physics (INFN-LNS), and in Trento, at TIFPA (Trento Institute for Fundamental Physics Application).

⁴Reactive Oxygen Species.

⁵Centro Nazionale di Adroterapia Oncologica.

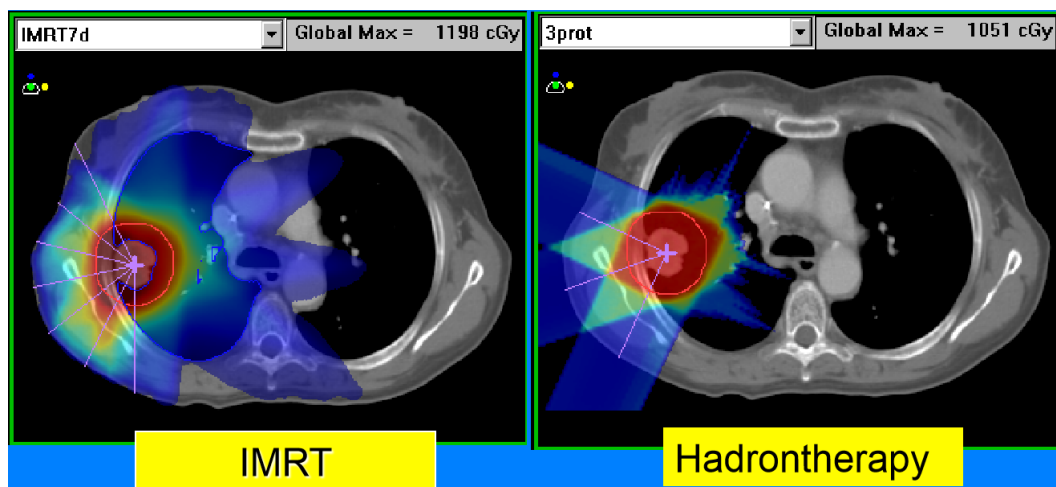


Figure 1.7: Energy distribution in conventional radiotherapy (on the left) and in hadrontherapy (on the right). Hadrontherapy allows to safeguard any vital organs in areas close to the tumor. Red corresponds to a large amount of energy released in the tissue, blue to a low one [18].

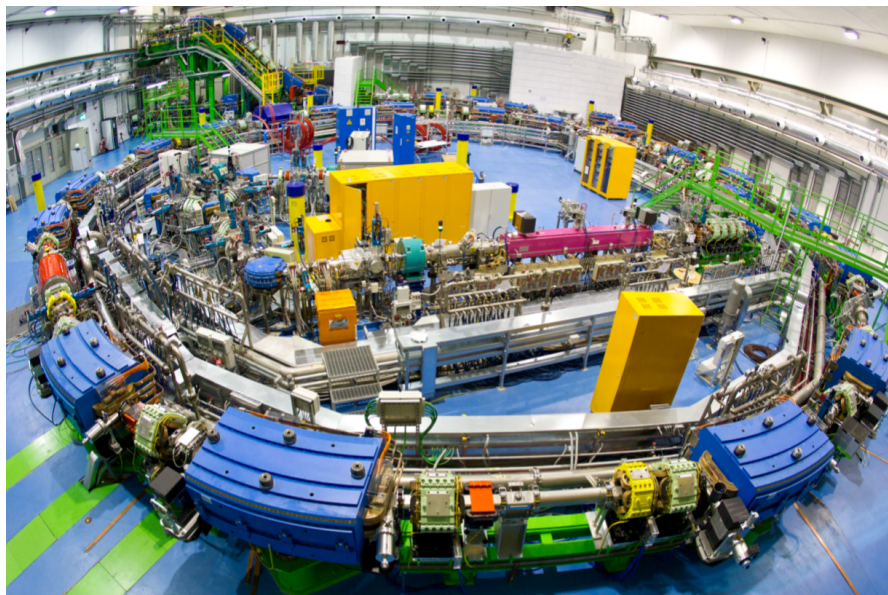


Figure 1.8: CNAO particle accelerator in Pavia used for hadrontherapy [18].

An additional problem is the uncertainty of some measurements and the lack of many experimental data which prevent the application of a standard treatment protocol. One of these doubts concerns the extent of the *fragmentation* process, which will be discussed in one of the next sections.

1.2.2 Therapeutic beams

The hadrons used in hadrontherapy can be of various types and sizes, each with particular properties that can make them more or less suitable for the treatment of a certain tumor. The most used and studied nuclei are:

- **Protons (p^+):**

the phenomenon of fragmentation is negligible, given that a proton cannot split into further parts and that the energies of the target fragments are extremely low;

- **Helium (${}^4\text{He}$):**

it shows a greater biological efficacy with respect to protons due to its double electric charge;

- **Carbon (${}^{12}\text{C}$):**

it has a much greater ability to damage cancer cells than protons. Furthermore, a beam of these particles disperses less than a beam of protons. This feature greatly improves the accuracy of the treatment, even if more bulky and expensive accelerators are needed, as they must be able to supply the energy needed to a more massive hadron. Fragmentation is not negligible;

- **Oxygen (${}^{16}\text{O}$):**

it has high specific ionization and its greater nuclear stability limits its fragmentation. It is indicated in cases where a very high deposited energy per unit length ratio is required.

Other hadrons have also been studied as probes for hadrontherapy, but these listed are those that give the best prospects, both because they are all biologically compatible materials with the human body (helium is inert, while protons, carbon and oxygen are present in quantity) and because they are particularly stable

nuclei, so the phenomenon of fragmentation is more contained than in other cases (nitrogen).

1.2.3 Relative Biological Effectiveness (RBE) and Linear Energy Transfer (LET)

The evaluation of the effects of ion beams in hadrontherapy is usually described in terms of two parameters known as *Relative Biological Effectiveness* (for the biological part) and *Linear Energy Transfer* for what regards the energy deposit.

The *Relative Biological Effectiveness* (RBE for short) characterizes the biological effects produced by a specific type of "projectile" used in the therapy with the same dose released in the tissue by an ionizing reference radiation. It is defined as:

$$RBE = \frac{D_{Rx}}{D_r} \quad (1.1)$$

where D_{Rx} is the dose of energy released by a reference ionizing radiation (usually γ radiation) to obtain a certain biological effect and D_r is the dose needed for the radiation or hadron to be tested to achieve the same effect. Protons have an RBE of about 1.1, while Carbon ions between 3 and 4 at maximum.

The *Linear Energy Transfer* (LET) quantifies the energy released to the tissue in a certain spatial interval. It is defined as:

$$LET = \frac{dE}{dx} \quad (1.2)$$

where dE is the amount of energy released by the particle (both because of the primary and secondary beam) over the distance dx .

Both RBE and LET are dependent on each other, in fact they have a very precise relationship shown in figure 1.9 [20].

1.2.4 Nuclear fragmentation

One of the most important problems concerning hadrontherapy is the already mentioned phenomenon of *fragmentation*: when hadrons impact the tissues they give rise to full-fledged nuclear fissions of the hadrons themselves or of the atoms

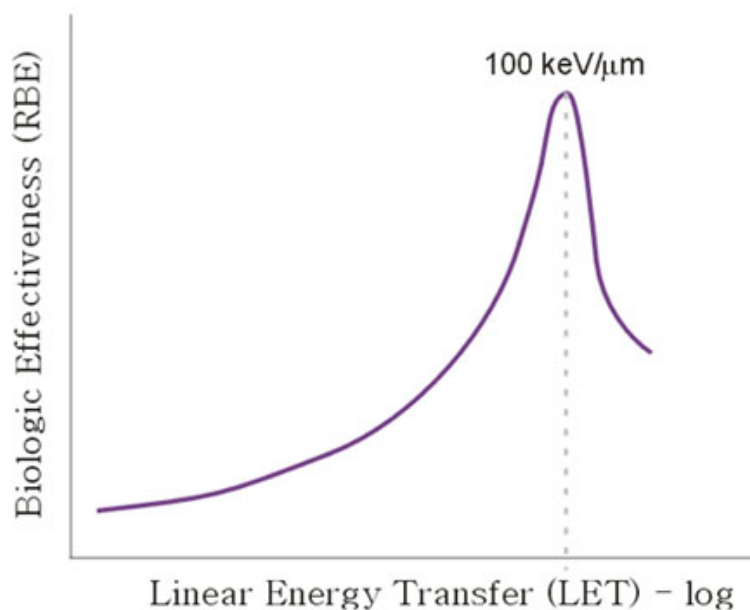


Figure 1.9: Relationship between LET and RBE [20].

of the target or both, which break into smaller atoms and particles. In other words, the nuclear interactions between the hadrons of the beam and the atoms of the tissues induce the fragmentation of both the former and the latter. These fragments can add a significant contribution to the released energy both before and after the Bragg peak (figure 1.10, blue line), especially when heavier particles such as ^{12}C are used. This means that the energy of these fragments may be sufficient to damage other cells in other locations than that of the tumor, undermining the accuracy of the therapy. Furthermore, radiotherapy cannot be used for those tumors located close to very sensitive organs, therefore hadrontherapy is preferred since it damages the tissues in a more targeted way (as shown in figure 1.7); on the other hand, fragmentation makes the use of those probes that appear to damage tumors better less appealing: carbon or oxygen ions (in figure 1.5 observe the difference of energy released between protons and ^{12}C after the Bragg Peak). In order to better understand what is the real amount of energy released in healthy tissues and in any neighboring vital organ due to fragmentation, the *FOOT* experiment has been designed. The main objective of the experiment is to acquire experimental data

regarding the entity and the main characteristics of this phenomenon in order to understand and, if possible, limit and control any side effects. The main goals and details will be documented and discussed in the next chapter.

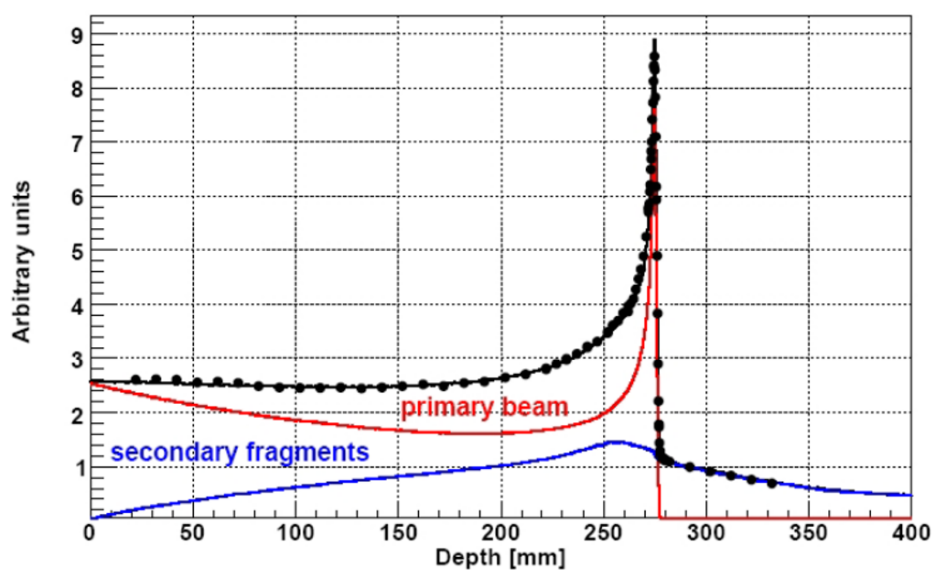


Figure 1.10: Graph showing energies deposited by the primary beam (in red) and secondary particles in fragmentation processes (in blue) [15].

Chapter 2

The FOOT experiment

Over the past decade, there has been an increase in the number of cancer patients treated with hadrontherapy [21]. In fact, as already said, most of the advantages of this new technique is precisely the different distribution of the energy released in the tissues in relation to the depth which allows to be more selective in the therapy and therefore reduces the damage to healthy tissues. Unfortunately, at present, the current experimental data are not sufficient for the application of a standard treatment protocol. The *FOOT* experiment, acronym that stands for *FragmentatiOn Of Target*, was proposed to fill some of these gaps, or, more precisely, to understand more about the fragmentation phenomenon.

Experiments have already been carried out on proton and carbon ion-induced fragmentation, but in a narrow band of beam energy, and therefore the data are not sufficient for an accurate risk analysis. In addition, the previous experiments were mainly focused on light fragments ($Z < 3$, in which Z is the atomic number, that is the number of protons found in the nucleus of that fragment) and there is a total lack of data on the production of heavier fragments [22][23].

The FOOT experiment aims to measure the fragmentation probability of heavy and light fragments at the desired energies, that are up to 250 MeV for protons and 400 MeV/u¹ for carbon ions, all this in order to gather new information and improve the accuracy of the data already present.

In the case of ¹²C the fragmentation is expected to manifest as a "tail" immedi-

¹megaelectron volts per unit of atomic mass.

ately after the Bragg peak due to highly energetic and fast fragments that continue to move beyond the peak; on the contrary, in the case of protons, the secondary beam is expected to be made up only of fragments of the target (a proton cannot split into further parts at these energies) with very low energies and speeds which give an almost constant contribution limited to the region before the peak (see figure 1.5).

The FOOT experiment is carried out in the same experimental rooms where patients are treated with hadrontherapy. In this way, we intend to exploit the beams available to the treatment energies and flows. Due to the small size of the rooms, the experiment must have small dimensions and must also be easily relocated to minimize the stop to treatments caused by FOOT measurements.

2.1 Configuration

One of the main problems of the experiment concerns the reconstruction of the fragments produced by the protons, which have low speed, low energy, an average path that is typically expressed in a few tens of micrometers and, consequently, a low probability of leaving the target, whose thickness is about 2 mm. Obviously if the fragments do not come out of the target they cannot be revealed. The solution adopted is to use an inverse kinematic approach: instead of "shooting" protons at a target made up of carbon, oxygen, calcium, etc. that simulates the patient's body, these heavier ions are "fired" at a target formed by a layer of carbon and a layer of a hydrocarbon with a high percentage of hydrogen (polyethylene $(C_2H_4)_n$ for example). In this way, by subtracting the cross sections obtained from the impacts with carbon from those with hydrocarbon, the data are obtained only for hydrogen, which is nothing more than a proton, that is exactly what we wanted [22][24]. The use of a target made up directly of Hydrogen is not possible, since it would be extremely dangerous. In addition, polyethylene is much cheaper. This problem does not concern the beams of particles heavier than simple proton (such as beams of ^{12}C or ^{16}O) which produce fast, energetic fragments that easily come out of the target.

The experiment will have two configurations, one measuring heavier fragments (*Electronic Setup*) and one measuring lighter ones (*Emulsion Setup*). This double

configuration is necessary because simulations have shown that the heavier fragments tend to distribute themselves in a cone after the target with a maximum polar angle of about 10° , while the lighter ones can have much wider angles [25]. The need for the experiment of having very detailed measurements in both cases prevents from having a single configuration.

2.1.1 Electronic Setup

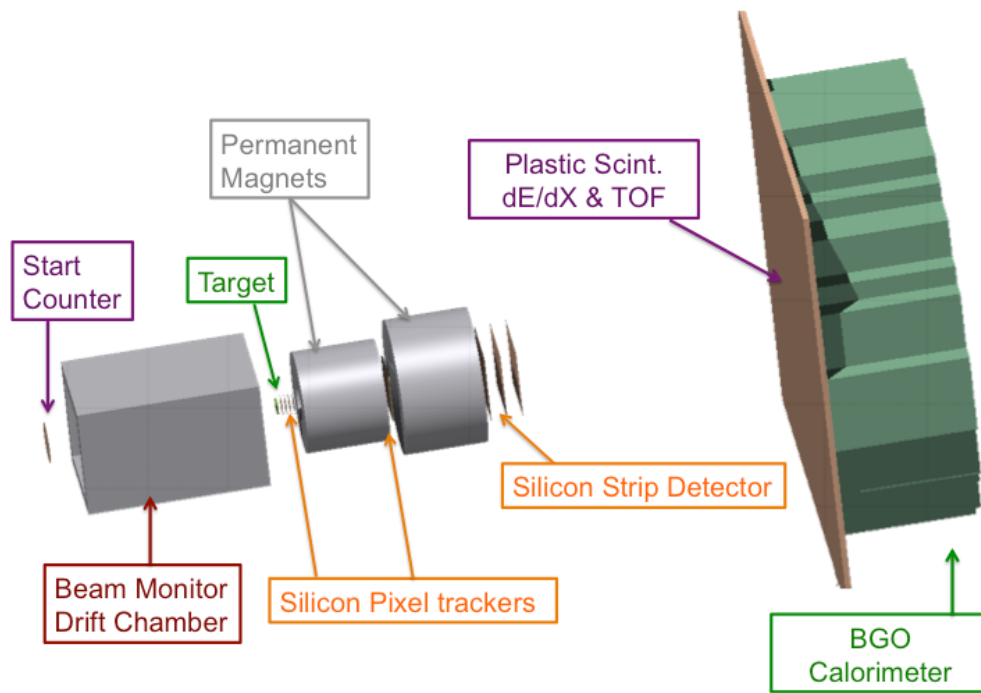


Figure 2.1: Visual representation of the configuration dedicated to heavy fragments [26].

This configuration is indicated for the study of the heaviest fragments that have a higher probability of exiting the target in a cone with a maximum angle of 10° . The system is designed to measure momentum, kinetic energy, time of flight (TOF) and LET of each fragment detected. In order to achieve this goal, the system, represented in figure 2.1, consists of several parts:

- **Start Counter**

it is the first element of the experiment. It provides the arrival time of a

particle and the trigger signal for the whole experiment as well as marking the start of the TOF. It is a plastic scintillator EJ-228, made by *Eljen Technology*, $5 \times 5 \text{ cm}^2$, $250 \mu\text{m}$ thick, encapsulated in an aluminium frame (figure 2.2). The read-out is performed by 48 SiPMs, organized in 8 groups of 6 SiPMs each.

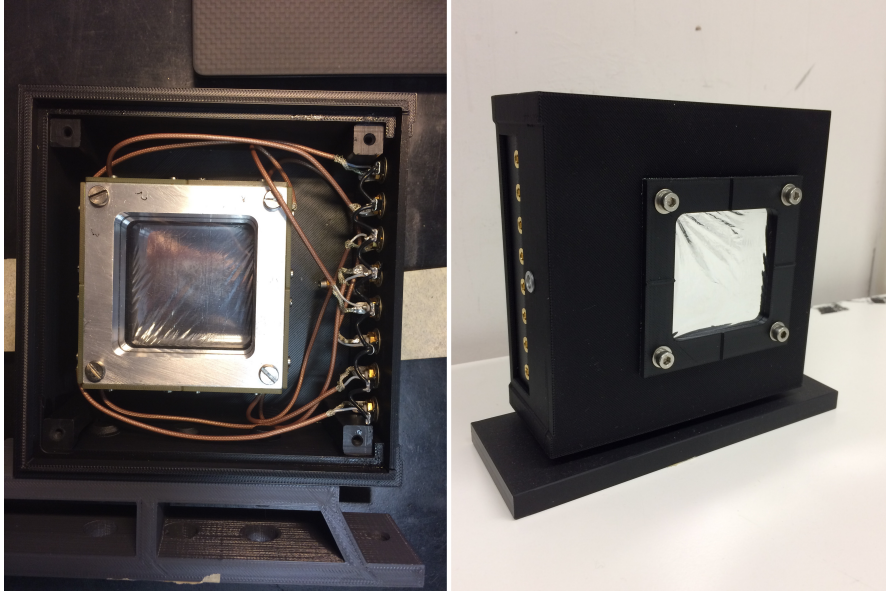


Figure 2.2: Start Counter photos.

- **Drift Chamber**

It is a fundamental component because it will be used to measure the direction and impinging point of the ion beam on the target. The drift chamber contains Argon, CO_2 and 12 layers of anodic wires capable of detecting the passage of charged particles. The wires are oriented alternately in the x and y direction, in order to reconstruct the beam profiles (figure 2.3);

- **Target**

it is the target on which, with a probability of the order of 1%, fragmentation occurs. The target, made of pure carbon (graphite) or polyethylene, has a typical thickness of the order of a millimeter;

- **Silicon pixel trackers**

there are two silicon pixel trackers in the experiment: the first one, located

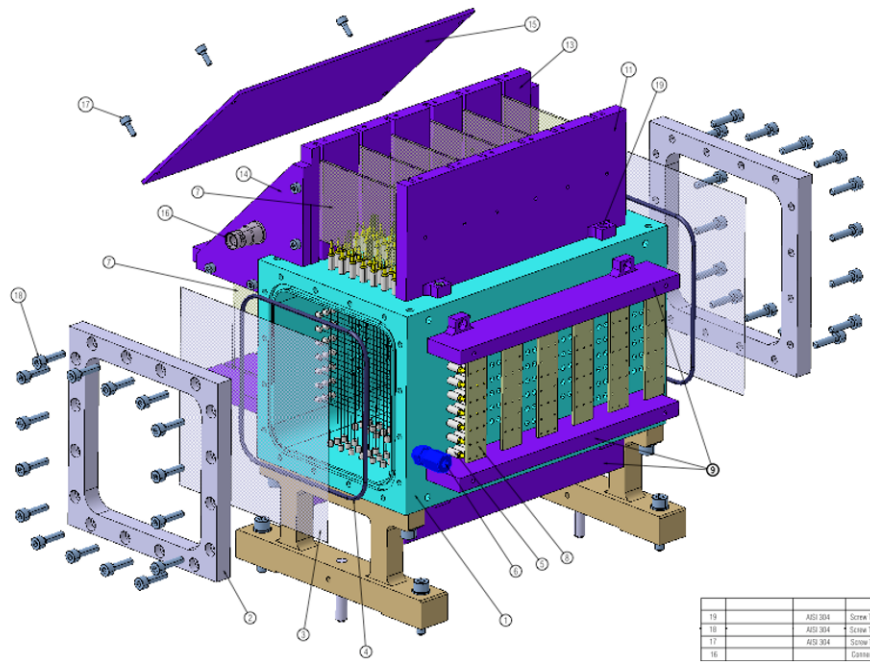


Figure 2.3: Schematic of the drift chamber [14].

5 mm from the target, is made up of 4 layers of pixel sensors MIMOSA28² (shown in figure 2.4) and it is responsible for the patch reconstruction, looking at the products immediately after fragmentation and therefore obtaining the point of impact; the second one is made up of 2 layers and, together with the silicon strip detector and the aid of magnets, it increases the precision of the measurement on the trajectory, allowing a precise calculation of the momentum of the particle;

- **Magnets**

permanent magnets with "Halbach array" geometry (figure 2.5) in order to have the most uniform magnetic field possible. A maximum uniform magnetic field of 0.8 T can be achieved with this geometry. This field bends the trajectory of the charged ions, allowing from this to derive the momentum;

- **Microstrip Silicon Detector (MSD)**

it is used to calculate the LET ($\frac{dE}{dx}$) and, as already said, the momentum.

²Acronym that stands for "Minimum Ionizing particle MOS Active pixel sensor".

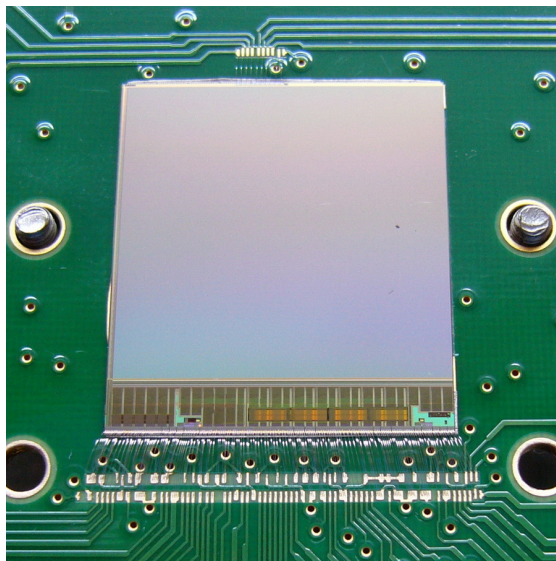


Figure 2.4: Pixel sensor MIMOSA28 [27].

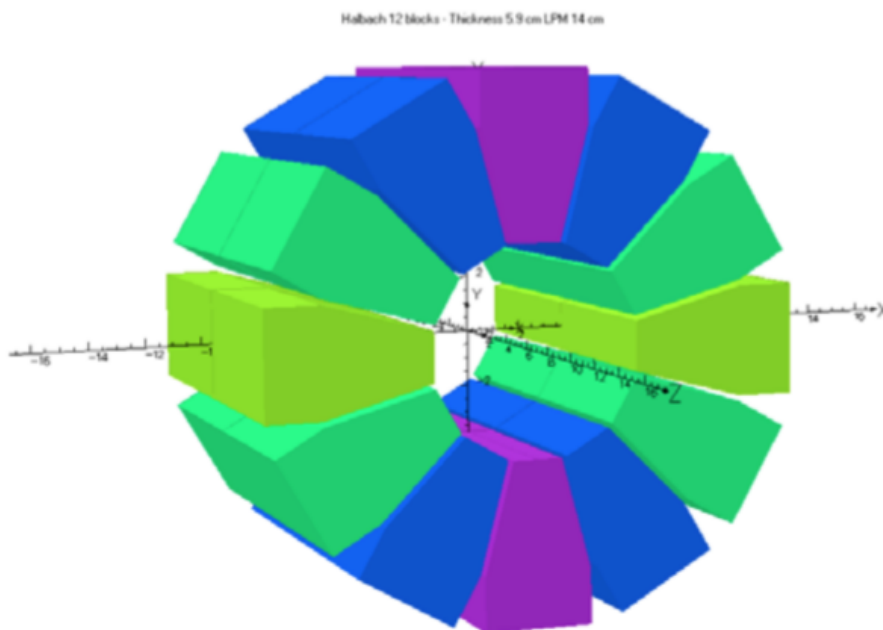


Figure 2.5: Schematic of one of the permanent magnets [28].

It consists of 3 layers each composed by 2 planes in which the microstrips are oriented orthogonally (one plane for the x axis and one plane for the y axis). Chapter 3 is dedicated to this component of the experiment and, more precisely, to the board which manages the acquisition of the data and the control of the sensors;

- **TOF wall**

at a distance of about one meter from the magnetic system the particle reaches another plastic scintillator and the TOF and again the LET are measured in order to have a redundant measurement and verify the data. This component consists of two orthogonal layers of 20 plastic scintillator bars, each 3 mm thick, 2 cm wide and 40 cm long. Each bar is connected at the two ends to two photomultipliers;

- **Calorimeter BGO³**

finally, inside the calorimeter, the kinetic energy of the particle is measured. The calorimeter is formed by an array of 350 2x2 cm² BGO crystals.

³BGO suggests the material of which the part is made, that is Bismuth Germanium Oxide, a luminescent material.

2.1.2 Emulsion Setup

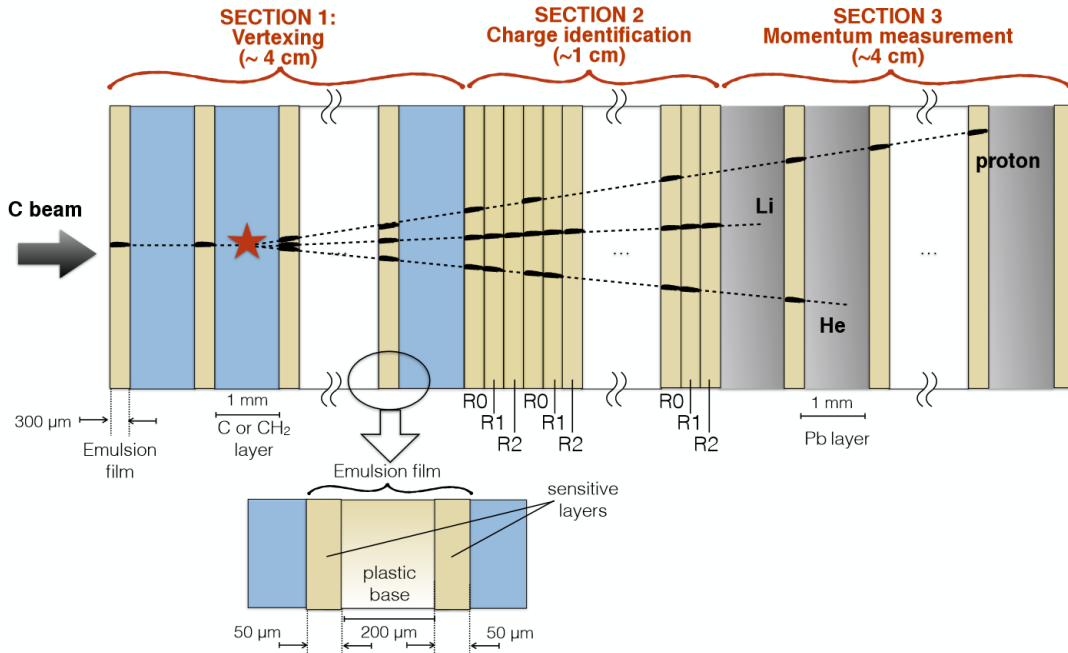


Figure 2.6: Visual representation of the configuration dedicated to light fragments with the Emulsion Spectrometer [26].

This configuration, shown in figure 2.6, is used to identify the lightest fragments that can be projected into a cone with a much wider angle of 10° . The maximum angle at which a deflected fragment is detectable is 70° . While the electronic control part remains the same (trigger board and remote control via PCs), all the electronic setup components, but the start counter and the drift chamber, are replaced by an emulsion spectrometer.

An emulsion chamber is a device made up of various layers of different composition and ordered according to the desired objectives: the key components are the emulsion sensitive layers which are formed by a plastic substrate on which two other thinner layers are present on both sides. These are composed by AgBr crystals scattered inside a gelatinous binder. These crystals are sensitive to the passage of any charged particle which leaves a "fingerprint" detectable under the microscope. The component is divided into three sections as shown in figure 2.6: a first part dedicated to identifying the position and direction of the fragments

immediately after impact, a second part dedicated to identifying the charge and a third part dedicated to measuring the momentum. In order to achieve these goals, the first section is formed by sensitive layers alternating with layers formed by C or CH₂, the second one is formed only by sensitive layers adjacent to each other and finally the third is formed by sensitive layers alternating with thick layers of lead [28].

2.1.3 Control system and interfaces

The FOOT detector is equipped with a parameterizable acquisition system that will allow different conditions and configurations of operation. The maximum acquisition frequency is of the order of 1 kHz and is due to the slower component, that are the pixel sensors. The system will be organized hierarchically and based on various Linux computers and VME crates⁴ with related cards that will communicate via standard interfaces such as USB, Ethernet and fiber optics. A main computer (Head PC) will be used to start, stop and configure the experiment and another computer called "Storage PC" will be used to collect the data from the sensors and save them on hard disks. Every sensor will have its own electronics to manage data and controls from the outside. Four signals are common: trigger signal, busy, *BCOclock* and *BCOreset*. The first one is the start signal of the experiment and will be provided to each sensors by the trigger system, which will be based on the signals coming from the start counter; the busy signals to the rest of the experiment that a specific device is busy for some reason (for example, still reading data or unable to respond to another trigger at that moment). The *BCOreset* and *BCOclock* (shown in figure 2.7) are used to synchronize the data between the various devices: the former is a reset that tells every device to reset their counters of rising edges at *BCOclock*, while the latter is a slow clock [28]. In addition, a series of custom "patch panel" cards are expected to be made, in order to better organize busy and trigger signals from and to sensor interfaces.

⁴Versabus Module Eurocard. It is a standard computer bus.

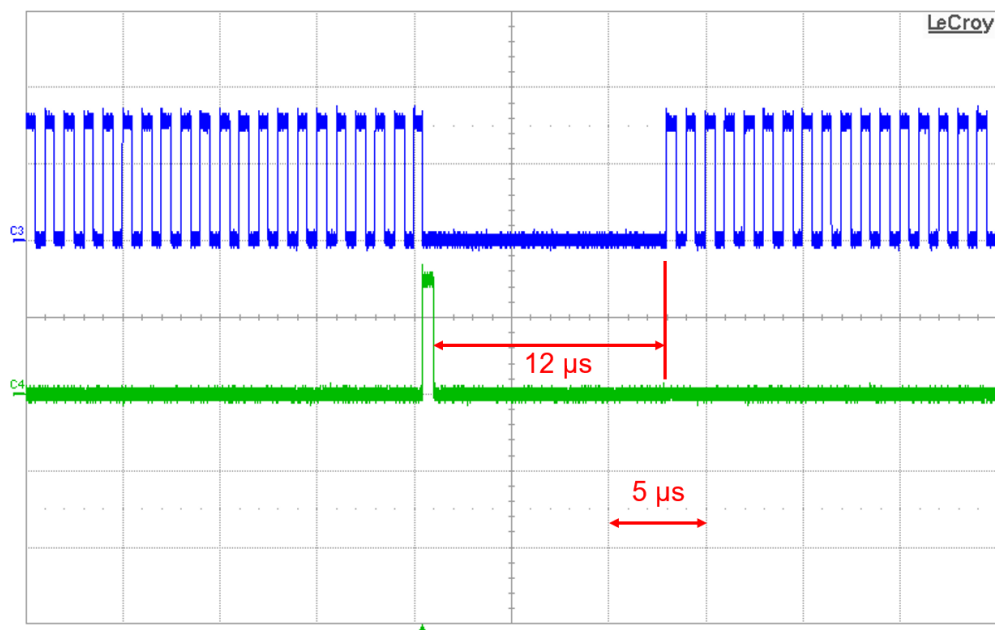


Figure 2.7: The *BCOClock* (on the top) and *BCOReset* (on the bottom) signals seen on the oscilloscope.

2.2 Future previsions and other considerations

There have been already two acquisition sessions in the configuration dedicated to light fragments in April 2019 and in February 2020 at GSI⁵, but the data have not been made public yet. A new data acquisition with the full configuration is expected to take place at CNAO in Pavia by the end of November 2020. The main objective of the experiment is to measure the extent of fragmentation in proton and carbon ion therapy, but this does not imply that the results are limited to the field of cancer treatment. In fact, the measurements performed with the proposed experiment could be also interesting for other applications, like radioprotection in space. This is the second main goal of the *FOOT* experiment. NASA and other space agencies have started since several years the study of the risk assessment for astronauts in view of long duration space missions in deep space, such for instance the travel to Mars [29]. The design and optimization of spacecraft shielding requires a detailed knowledge of fragmentation processes, which are very common in deep

⁵The GSI Helmholtzzentrum für Schwerionenforschung in Darmstadt operates a worldwide leading accelerator facility for research purposes.

space because of the continuous impacts of cosmic and sun particles on shuttles, capsules, rockets and, above all, astronauts.

Chapter 3

Acquisition system for the Microstrip Silicon Detector

This chapter is devoted to present the main topic of this Master's thesis, that is the data acquisition system for the Microstrip Silicon Detector (MSD) sensor. This component is a combination of FPGA firmware and C++ software development, but required also some changes to the structure of the Linux system device tree at the kernel level. In the following sections each one of these aspects will be covered, including some hints on the sensors themselves.

3.1 The Microstrip Silicon Detector

A microstrip silicon detector is a particle detector that consists of a large number of identical semiconductor strips laid out along one axis of a two-dimensional structure. In our experiment there are 3 measurement stations, each one made up of 2 other planes oriented orthogonally. The latter are called x and y planes (figure 3.1) and together will provide a space point along a nuclear fragment track. Every DAQ board DE10-Nano collects the data from a pair of x and y planes. The sensors have been made by *Hamamatsu Photonics K.K.*, a Japanese company that specializes in various products ranging from photomultiplier tubes, opto-semiconductors, and light sources, to image processing and measurement equipment [30]. In our specific case, the sensor is $96 \times 96 \text{mm}^2$ and the distance between two adjacent strips

is $50\ \mu\text{m}$, but only one every two is actually read from the outside, thus the actual readout pitch is $150\ \mu\text{m}$. The thickness of the active sensor is only $150\ \mu\text{m}$ but a layer of $5\ \text{mm}$ of light passive material was added to help mechanical handling. In total, each plane has 640 channels. When a charged particle hits or touches a strip, charges accumulate into the former, generating a faint analog signal on its channel, which is preamplified, shaped, sampled and held by an IDE1140 integrated circuit [31]. Since each plane has 640 channels and each IDE1140 has 64 inputs, 10 of these are required for each plane. This chip is designed by *IDEAS - Integrated Detector Electronics AS* -, which is a fabless supplier of integrated circuits and systems based in Oslo, Norway, which designs ICs and systems for radiation detection and imaging. The IDE1140 is a 64 channel low-noise/low power high dynamic range charge sensitive preamplifier-shaper circuit, with simultaneous sample and hold, multiplexed analogue readout, calibration facilities and internally generated biases. The Input charge range is $\pm 200\ \text{fC}$ [32]. When a trigger signal is received, the analog signals on the different channels, sampled by the previously mentioned IC, are passed one by one to two AD7476 ADCs that work in parallel on a single plane. The ADC used is a 12-bit high speed, low power, successive approximation ADC, which operates from a single $2.35\ \text{V}$ to $5.25\ \text{V}$ power supply and feature throughput rates up to $1\ \text{MSPS}$ [33]. The data converted to digital are finally sent serially to the DAQ board [34]. The entire path of the data from the raw sensors to the DAQ board is shown in the schematic in figure 3.2.

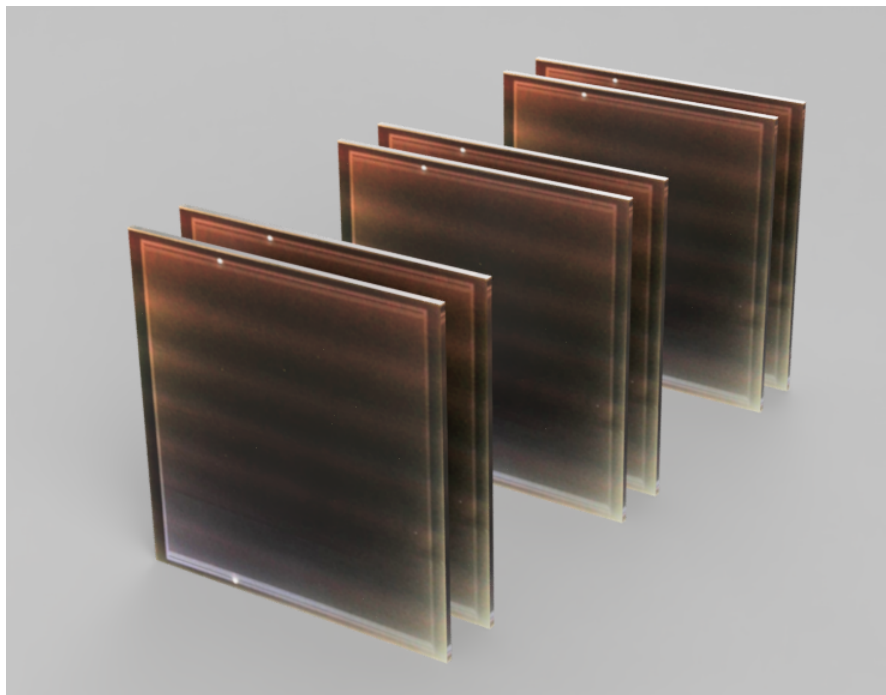


Figure 3.1: Schematic showing the arrangement of the 6 planes (not in scale).

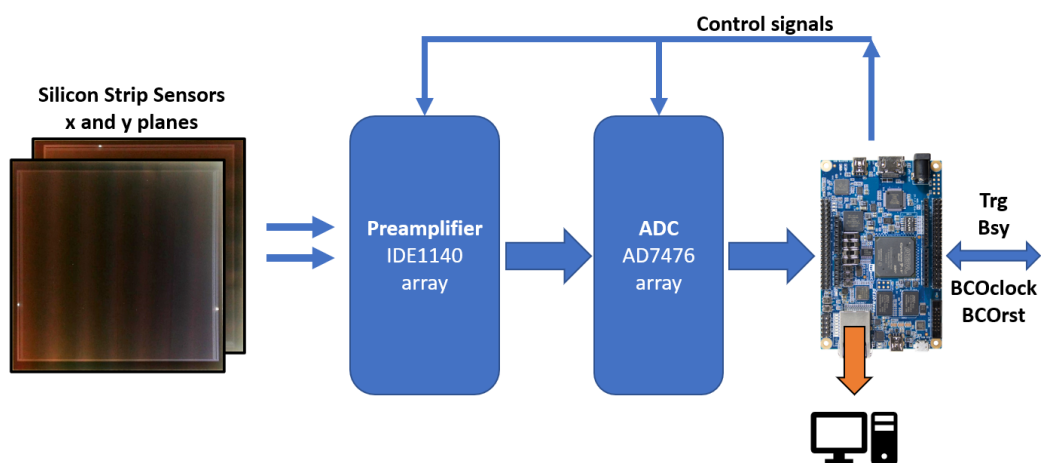


Figure 3.2: Schematic showing the system from the raw sensors to the acquisition board and Ethernet link.

3.2 Operation

The main purpose of the entire system (see figure 3.3) is to perform the acquisition of data from the silicon microstrip sensor, when a trigger signal arrives from the experiment control. As described in the previous section, a DAQ board model DE10-Nano will be dedicated to a pair of sensor planes (x -plane and y -plane), so there will be a total of 3 DE10-Nano for the microstrip sensor. While the board is acquiring data, it must set to high its busy signal. The board will stream the raw data acquired and other metadata, including the value of the *BCOclock* counter at the moment of the arrival of the trigger (beginning of an *event*, in Physics terms), via an Ethernet interface to the storage PC. The board has three main states: IDLE, CONFIG and RUN. In the first one, the board is idling, the second state is used to configure the board if needed to operate with special behaviours and the RUN state is its normal operation status during data taking.

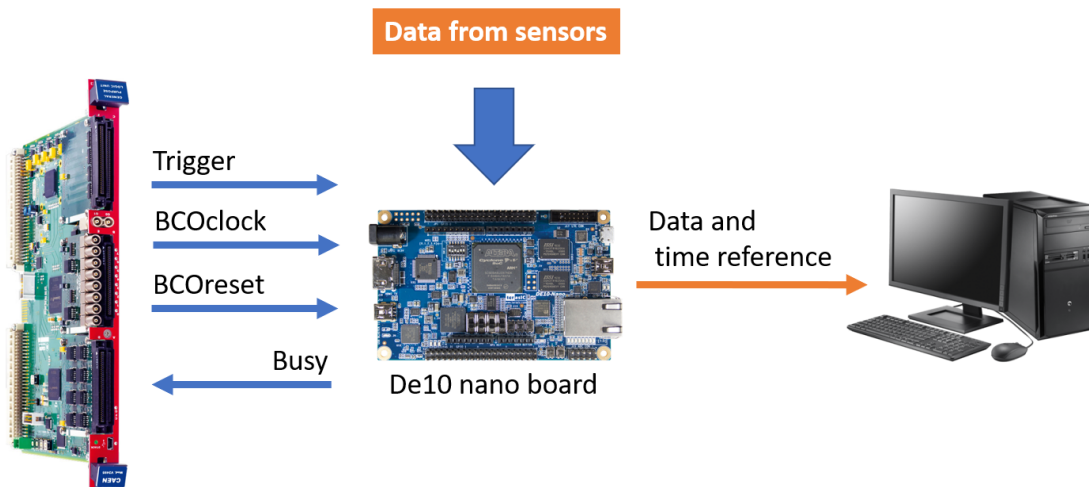


Figure 3.3: Generic schematic showing the main interconnections of the DAQ system for the Microstrip Silicon Detector.

3.3 The DE10-Nano board

The DAQ board used is a DE Series from Terasic, model DE10-Nano, shown in figures 3.4 and 3.5. The DE10-Nano Development Kit presents a robust hardware design platform built around the Intel System-on-Chip (SoC) FPGA, which combines the latest dual-core Cortex-A9 embedded cores with industry-leading programmable logic for ultimate design flexibility [35].

The SoC FPGA is a Cyclone V SoC 5CSEBA6U23I7NDK with 110,000 logic elements, 5,570 kilobits of on-chip memory, 224 18 x 19 multipliers, 112 variable precision DSP blocks, 6 phased-locked loops (PLL) and 145 User defined I/O. In addition to the standard Altera/Intel logic, the FPGA is accompanied by an Hard Processor System (HPS) which is made up of a dual core ARM Cortex A9 MPCore clocked at 800 MHz with 32 KB L1 instruction cache, 32 KB L1 data cache and 512 KB shared L2 cache. Furthermore, it is equipped with 64 KB of on-chip SRAM and 1 GB DDR3 SDRAM (32-bit data). The latter is necessary to run the *Ångström* Linux Distribution and Linux kernel 4.1.33 LTSI. The HPS and FPGA communicate with each other through bus interfaces that bridge the two distinct portions. The board is equipped with a generous amount of peripherals, some tied to the FPGA, some to the HPS (see figures 3.4, 3.5 and the schematic in figure 3.6). The peripherals tied to the FPGA are 2 push buttons, 4 slide switches, 8 LEDs, two 40-pin expansion headers with diode protection (36 GPIO usable, plus two GND, a 5 V and 3.3 V pins), one Arduino expansion header with Arduino UNO R3 compatibility and an 8-channel, 12-bit A/D converter, 500 ksps, 4-pin SPI. The peripherals tied to the HPS are a Gigabit Ethernet PHY with RJ45 connector, a USB 2.0 On-The-Go (OTG) port, a microSD card interface and socket (this is used to store files and the image of the *Ångström* operating system), an accelerometer with I2C interface and interrupt, a UART to USB, USB Mini-B connector, a warm reset button, a user button and a user LED and finally an expansion header for use with Linear Technology DC934A daughter board [36]. The firmware and software can be uploaded in various ways which will be explained in the following sections.

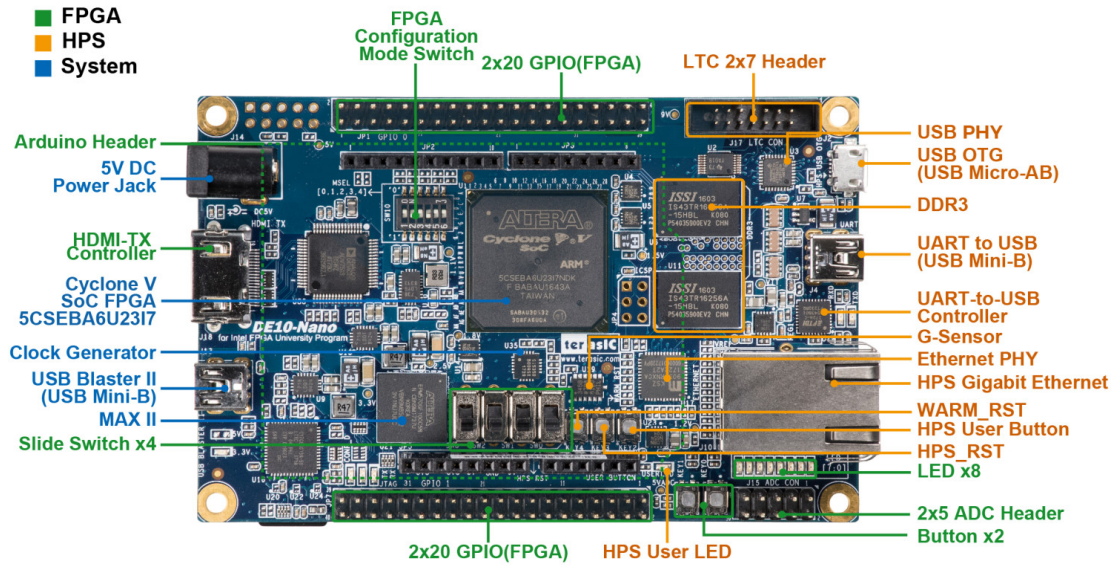


Figure 3.4: Top view of the DE10 nano board with highlighted main components [37].

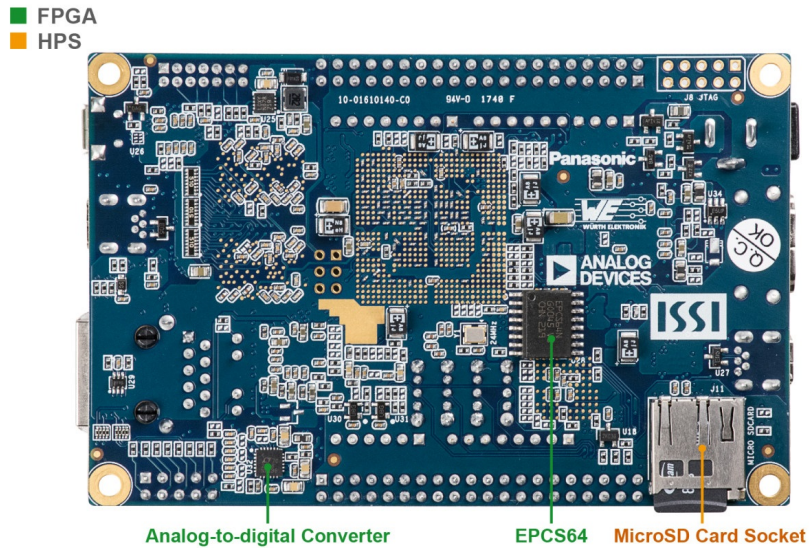


Figure 3.5: Bottom view of the DE10 nano board with highlighted main components [37].

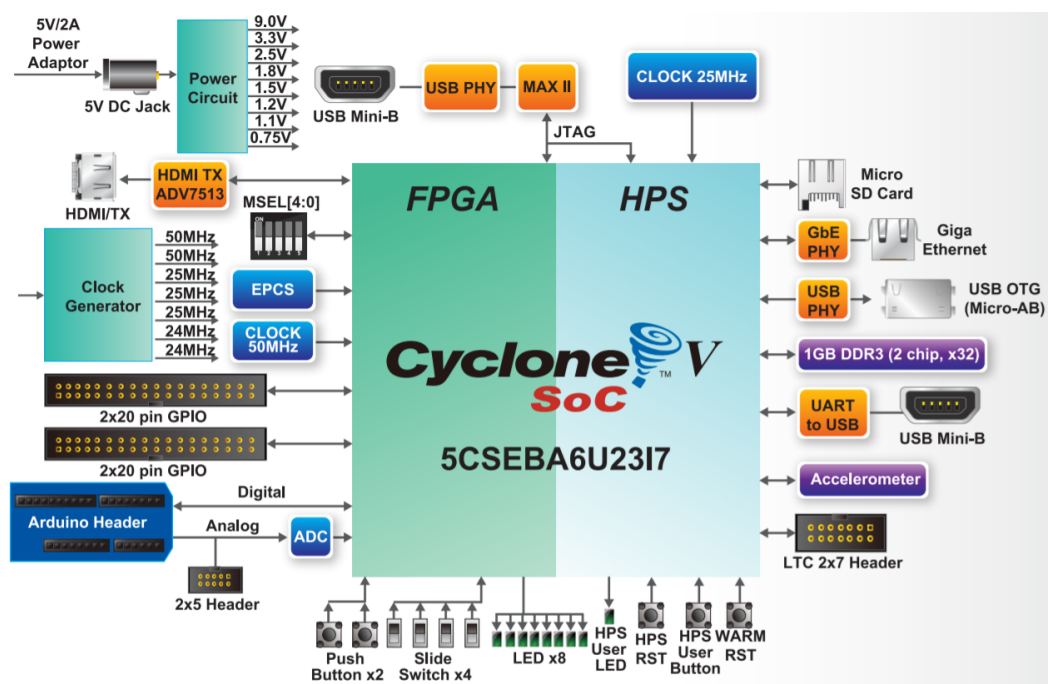


Figure 3.6: Abstract depiction of the different components on the DE10 Nano board [36].

3.4 FPGA and Hard Processor System

As already said, the HPS and FPGA communicate with each other through specific interfaces which connect the two distinct blocks. These interfaces are called *HPS-to-FPGA interfaces* (figure 3.7) and provide a variety of communication channels between the HPS and the FPGA fabric. The HPS is highly integrated with the FPGA fabric, resulting in thousands of connecting signals.

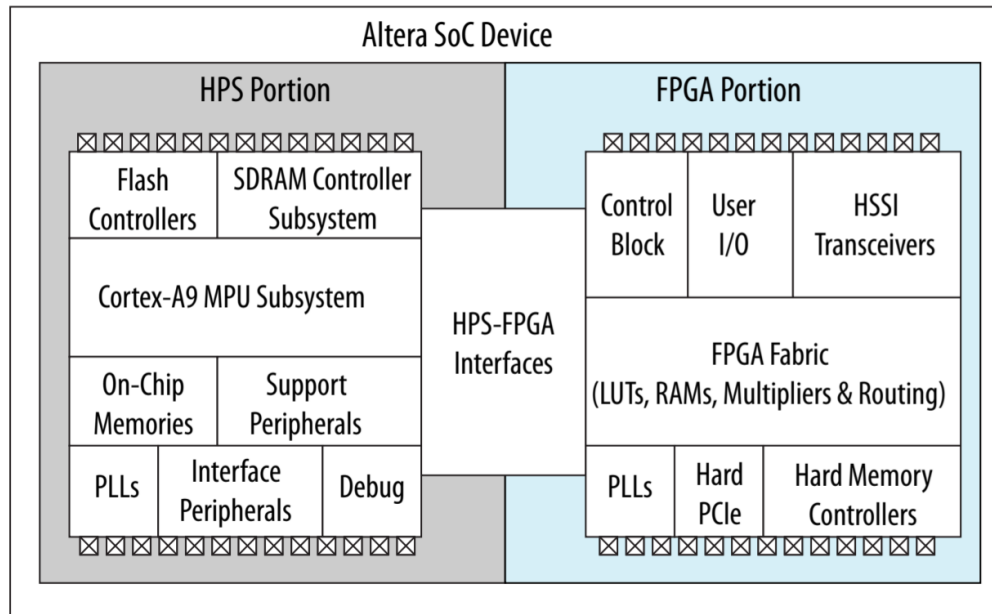


Figure 3.7: The two different portions of the Cyclone V SoC mounted on the DE10 Nano board. Note the HPS-FPGA interfaces in the middle [38].

3.4.1 The HPS–FPGA Interfaces

The HPS–FPGA *memory-mapped* interfaces provide the major communication channels between the HPS and the FPGA fabric [38]. There are many different channels, however only the most important ones will be discussed in detail:

- **FPGA–to–HPS bridge:**

a high–performance bus with a configurable data width of 32, 64, or 128 bits, allowing the FPGA fabric to master transactions to the slaves in the HPS. This interface allows the FPGA fabric to have full visibility into the HPS

address space. This interface also provides access to the coherent memory interface [38];

- **HPS-to-FPGA bridge:**

a high-performance interface with a configurable data width of 32, 64, or 128 bits, allowing the HPS to master transactions to slaves in the FPGA fabric [38];

- **Lightweight HPS-to-FPGA bridge:**

a low-performance interface with a 32-bit fixed data width, allowing the HPS to master transactions to slaves in the FPGA fabric. The lightweight bridge is useful for low-bandwidth traffic, such as memory-mapped register accesses to FPGA peripherals. This approach diverts traffic from the high-performance HPS-to-FPGA bridge, and can improve both register access latency and overall system performance [38];

- **FPGA-to-HPS SDRAM interface:**

the SDRAM controller on HPS contains a multiport frontend (MPFE) that accepts requests from HPS masters and from soft logic in the FPGA fabric through the FPGA-to-HPS SDRAM interface. As a result, the FPGA has access to a low-latency, high performance direct communication channel to the on board 1 GB SDRAM memory. It is important to underline that this access is not cache-coherent, so discretion is advised [38].

3.4.2 The Platform Designer

The *HPS-FPGA interfaces* require instantiation and configuration. Furthermore, the user needs a tool in order to connect these bridges to peripherals on the FPGA. All of this is done through the *Platform Designer* (formerly *Qsys*). This tool is the next generation system integration tool in the Intel Quartus Prime software. It saves significant time and effort in the FPGA design process by automatically generating interconnect logic to connect intellectual property (IP) functions and subsystems. Furthermore, it makes use of a powerful hierarchical framework to offer fast response times for interconnecting large systems, while also providing support for blackbox entities [39]. This means that the Platform Designer allows

a simplified way of linking together complex HDL designs, and this is thanks to the standard *Avalon* interface, whose family defines interfaces appropriate for streaming high-speed data, reading and writing registers and memory, and controlling off-chip devices. Components available in Platform Designer incorporate these standard interfaces. Additionally, it can incorporate Avalon interfaces in custom components, enhancing the interoperability of designs. Thanks to this versatility, the Platform designer allows us to use custom HDL code to interact with the already mentioned HPS-FPGA *memory-mapped* interfaces. The specifications define seven different varieties of the *Avalon* interface, but only the *Avalon-MM* will be explored more in depth, since it is the one used in the DAQ board.

3.4.3 Avalon-MM Interface

The *Avalon Memory Mapped Interface* (*Avalon-MM*) is an address-based read/write interface typical of master-slave connections. It is synchronized to an associated clock interface. Signals may be combinational if they are driven from the outputs of registers that are synchronous to the clock signal. This specification does not dictate how or when signals transition between clock edges. This specification does not require all signals to exist in an Avalon-MM interface. There is no signal that is always required. The most important are (figure 3.8):

- **address:**

by default, the *address* signal represents a byte address. The value of the address must align to the data width. From the point of view of the slave, by default, the interconnect translates the byte address into a word address in the slave's address space. Each slave access is for a word of data. For example, *address* = 0 selects the first word of the slave, while *address* = 1 selects the second word of the slave;

- **readdata:**

The data driven from the slave to the master in response to a read transfer;

- **read:**

asserted to indicate a read transfer;

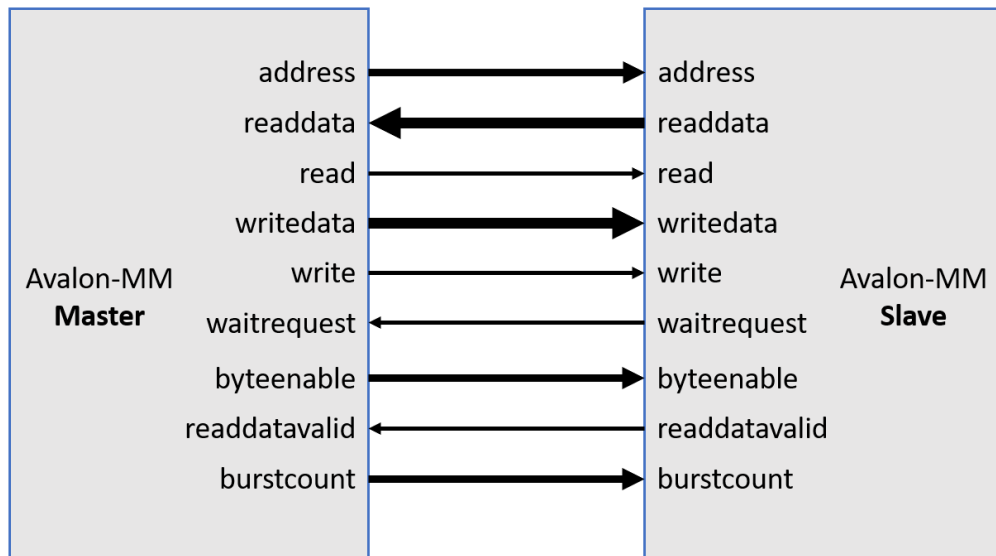


Figure 3.8: Simple schematic showing the main signals of the Avalon-MM interface.

- writedata:**
 data for write transfers. The width must be the same as the width of *readdata* if both are present;
- write:**
 asserted to indicate a write transfer;
- waitrequest:**
 a slave asserts *waitrequest* when unable to respond to a read or write request. In this case, it forces the master to wait until the interconnect is ready to proceed with the transfer. At the start of all transfers, a master initiates the transfer and waits until *waitrequest* is deasserted. When *waitrequest* is asserted, master control signals to the slave must remain constant (except for *beginbursttransfer*, a deprecated signal). A master must make no assumption about the assertion state of *waitrequest* when the master is idle, since *waitrequest* may be high or low, depending on system properties. An Avalon-MM slave may assert *waitrequest* during idle cycles. An Avalon-MM master may initiate a transaction when *waitrequest* is asserted and wait for that signal to be deasserted;

- **byteenable:**

enables one or more specific byte lanes during transfers on interfaces of width greater than 8 bits. Each bit in *byteenable* corresponds to a byte in *writedata* and *readdata*. The master bit $\langle n \rangle$ of *byteenable* indicates whether byte $\langle n \rangle$ is being written to. During writes, *byteenable* specifies which bytes are being written to. Other bytes should be ignored by the slave. During reads, *byteenable* indicates which bytes the master is reading;

- **readdatavalid:**

used for variable-latency, pipelined read transfers. When asserted, indicates that the *readdata* signal contains valid data. For a read burst with *burstcount* value $\langle n \rangle$, the *readdatavalid* signal must be asserted $\langle n \rangle$ times, once for each *readdata* item. There must be at least one cycle of latency between acceptance of the read and assertion of *readdatavalid*. A slave may assert *readdatavalid* to transfer data to the master independently of whether the slave is stalling a new command with *waitrequest*;

- **burstcount:**

used by bursting masters to indicate the number of transfers in each burst. The value of the maximum *burstcount* parameter must be a power of 2. A *burstcount* interface of width $\langle n \rangle$ can encode a max burst of size $2^{\langle n \rangle - 1}$. For example, a 4-bit *burstcount* signal can support a maximum burst count of 8. The minimum *burstcount* is 1.

The minimum requirements for an Avalon-MM interface are *readdata* for a read-only interface, or *writedata* and *write* for a write-only interface.

The figure 3.9 shows a typical read and write transfers using *waitrequest*. A slave typically receives *address*, *byteenable*, *read* and/or *write*, and *writedata* after the rising edge of the clock. A slave asserts *waitrequest* before the rising clock edge to hold off transfers. When the slave asserts *waitrequest*, the transfer is delayed. While *waitrequest* is asserted, the *address* and other control signals are held constant. Transfers complete on the rising edge of the first clock after the slave interface deasserts *waitrequest*. There is no limit on how long a slave interface can stall. Therefore, the user must ensure that a slave interface does not assert *waitrequest* indefinitely [40].

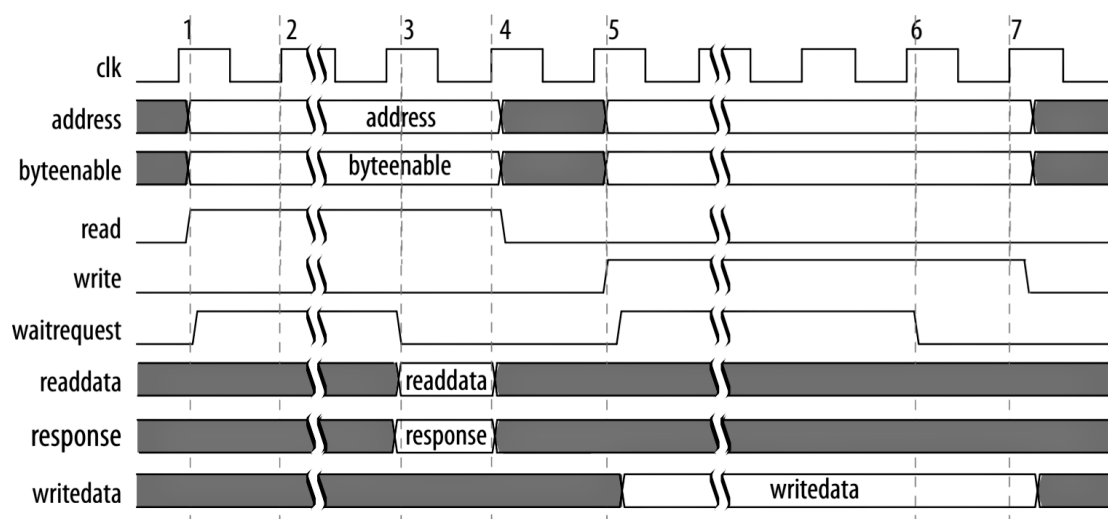


Figure 3.9: A typical read and write transfer using *waitrequest* [40].

3.5 FPGA DAQ firmware

In this section, the architecture and functionality of the firmware downloaded to the FPGA will be explained in detail. The code is based on the work of Enrico Vezzali [41], which was improved in data throughput and responsiveness with some additional features, which will be shown later. Furthermore, the readability was improved thanks to the addition of numerous comments to the code. This document refers to the firmware version 0xDE100209.

The system is ideally made up of 3 macroblocks, that are the *Acquisition*, the *Communication* and the *SoC_system*, the entity¹ generated by the Platform Designer (figure 3.10). The operation of each macroblock is controlled by a finite state machine, called *Main_FSM*, whose states are controlled externally from the Head PC (see section 2.1.3). The *Acquisition* block manages the retrieval and formatting of data from the sensors and redirects them to the *Communication* block, which handles the data and send them to the third block, the *SoC_system*. In addition, the *Communication* block handles the commands and queries from the Head PC and sends the appropriate responses. Finally, the *SoC_system* is an entity generated by the Platform Designer which includes the interconnections between the FPGA fabric and the HPS. Two entity used only for debug purposes are also present, *BeamSimu* and *Trigger_generator*. All these blocks will be discussed in the next sections.

3.5.1 The finite state machine

This entity *Main_FSM* is the main block which governs the operations of all the other entities and systems. It contains 6 states: Idle, Config, PrepareForRun, Run, EndOfRun and WaitingEmptyFifo. When the firmware is loaded into the FPGA, the finite state machine is in the Idle state: here the board is idling and waiting for a specific command from a connected PC. When this is received, the machine transits to the Config state. Here the board waits for other configurations, if needed, and waits for another transition command. The next state is PrepareForRun, which is a transition state just for synchronization reasons. After this, the

¹Entities are the main blocks which compose VHDL code.

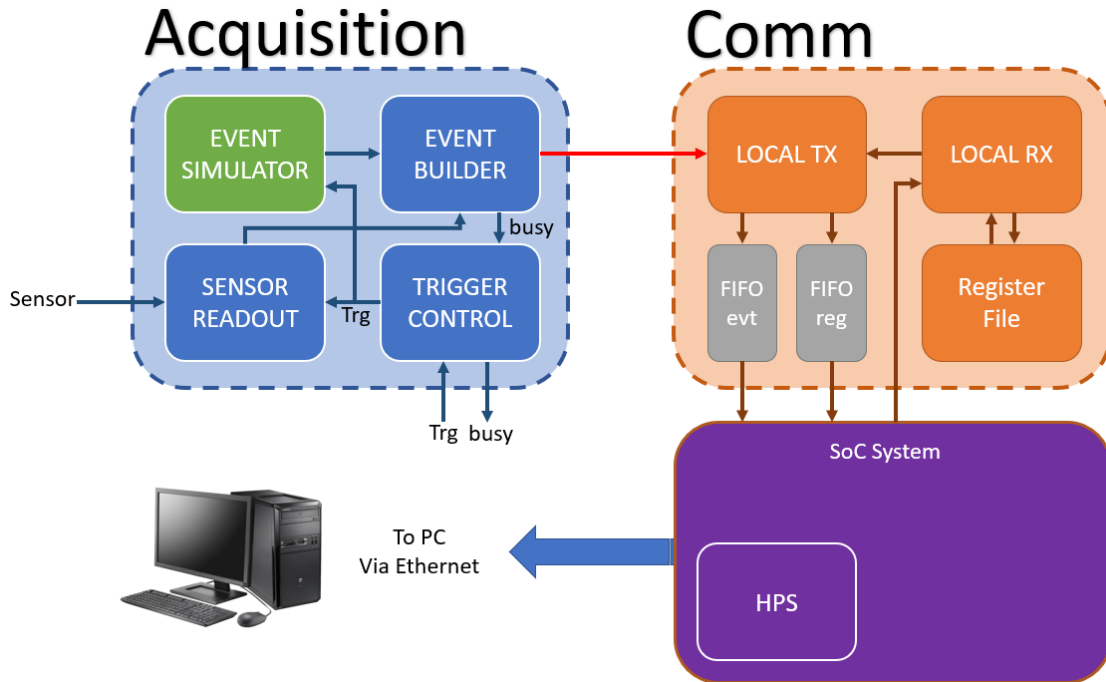


Figure 3.10: Schematic showing the main macro-blocks.

Run state is reached: the board is active and ready to receive triggers from the experiment trigger control, manage data and send them to the PC. When another command is received, the board transits to EndOfRun. In this state the board checks if an event is being read from the sensor and, in this case, waits. The last state is WaitingEmptyFifo in which the board waits for all the data in the FIFOs to be dumped on the storage PC. The next state is Config and the cycle can restart from it.

3.5.2 The acquisition

The firmware which handles and formats every event in the appropriate manner in order to be passed to the HPS (figure 3.11) consists of 3 main entities plus a simulation entity:

- **Event_Simulator:**
this entity emulates the sensor. When a trigger is received it starts to send dummy data to the *Event_Builder* via a 32 bit bus interface. Every block of

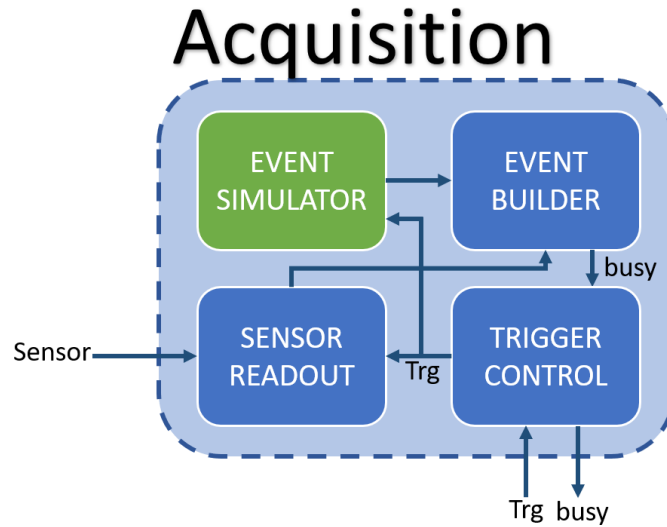


Figure 3.11: The entities dedicated to data management.

data starts with a word which indicates the length (in words) of the transmission. After this, some header words are sent and then the simulated data. The latter are composed by random numbers and other values which can help during debugging. Every event terminates with some trailing words. During the whole transmission the *Data_Valid* signal is held high, while the *EndOfEvent* signal goes high only while the last word of the event is present on the bus;

- **Sensor_Readout:**

it has the same inputs and outputs towards the rest of the firmware as the *Event_Simulator*. The only difference is that it interacts with the actual front-end chips reading the sensor data. It was made by a team of the University of Perugia;

- **Event_Builder:**

this entity receives the data from the *Event_Simulator* or the *Sensor_Readout* and saves them in a FIFO called *SC_FIFO*. At the rising edge of the *Data_Valid* signal, the values of some significant signals are latched and saved in another FIFO called *metadata_Fifo*, including the value of the counter of rising edges of *BCOclock* and the length of the incoming event.

During a normal reading process this block asserts the busy signal, but the board can be configured to assert the busy until the entire event has been dumped from the *SC_FIFO* and the *metadata_Fifo* and not only during the reading sequence. This mode of operation was used for debug purposes;

- **Trigger_Control:**

the *Trigger_Control* entity receives the trigger signal from the trigger control of the experiment and forwards it to the *Event_Simulator* and the *Sensor_Readout* only if the *Event_Builder* is not busy and the finite state machine is in the Run state.

3.5.3 Communication and data retrieval

All the data acquired from the sensors need to be sent to the PC storage and this is done through a chain of various blocks and FIFO memories which lead to the *SoC_system* instantiation block (described in the following section). Furthermore, the board is capable of receiving commands and queries and sending responses on the status of the system to the HPS.

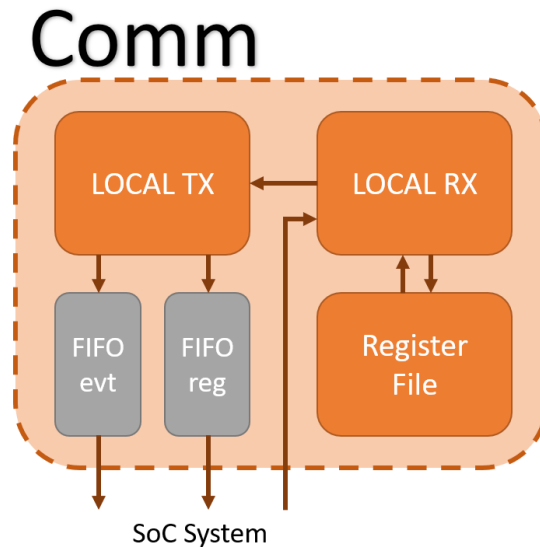


Figure 3.12: The entities devoted to communication management.

The previously mentioned blocks of the acquisition system are directly connected to 2 other entities, that are *Local_RX* and *Local_TX* (figure 3.12):

- **Local_RX:**

this entity is responsible for the management of the commands and queries which arrive from the HPS. When a message is received, it is stored in a FIFO called *FifoRX*. On the other side, the messages are retrieved and decoded. The messages could be a reset signal, a change-state signal for the main finite state machine, a write request or a read request on a register. In the last two cases, *Local_RX* accesses to another entity called *Register_File*, which comprehends all the necessary logic to read or write from or to a set of registers, which are divided into two main groups: the *monitor registers*, used to inspect the status of the FPGA (read only) and the *control registers*, used to control the behaviour of the board (read and write). Details about the registers and relative contents are provided in the appendix. In the specific case of a read request, the read value is formatted and stored in another FIFO called *Internal_Fifo*;

- **Local_TX:**

this block manages the data which are supposed to be sent to the HPS, thus it has two read interfaces: one is connected to the *Local_RX*, whose data are retrieved and stored into a FIFO called *Fifo_TX_Regs*, while the other is connected to the *Event_Builder*, whose data are stored into another FIFO called *fifo_data*. The outputs of these two FIFOs are connected to two other FIFOs in one of the top entities: these two are the last FIFOs and are called *EventFifo* and *RegEventFifo*. The outputs of these two last FIFOs are directly connected to the *SoC_system*, which will be explained in the following section.

3.5.4 SoC_system

The connection between the FIFOs of the events and registers data and the HPS is implemented via the *SoC_system*, the special entity generated by the *Platform Designer*. Every block instantiated into the Platform Designer uses one of the *HPS-FPGA interfaces* mentioned above to exchange informations with the HPS according to its needs. These components are:

- **hps:**

this component represents the *HPS-FPGA interfaces* (figure 3.7). It is the access gate to the real HPS. The previously discussed interfaces can be configured, enabled and disabled, but also other features can be managed, such as HPS-FPGA GPIOs and external SDRAM parameters;
- **sysid_qsys:**

it is a read-only register connected to the lightweight HPS-to-FPGA bus. When read, it returns the firmware revision;
- **dipsw_pio:**

it is a read-only register containing the status of the FPGA DIP switches of the board. It is connected to the Lightweight HPS-to-FPGA bus;
- **button_pio:**

it is a read-only register containing the status of the FPGA-side buttons of the board. It is connected to the Lightweight HPS-to-FPGA bus;
- **led_pio:**

it is a read/write register containing the turn on/off status of the 8 LEDs of the board. It is connected to the Lightweight HPS-to-FPGA bus;
- **iofifocontrol:**

this component handles the fifo interface between the HPS and the FPGA. It provides access to three communication FIFOs and their status information. It receives the outputs of *EventFifo* (the actual data from the sensors) and *RegEventFifo* (responses of the FPGA to HPS queries) FIFOs and the signals that indicate the status of these two. If a write access is performed on this component at the offset 0, the data on the bus are sent to the *Local_RX* entity as commands or queries from the HPS. If a read access is performed at the offset 0, the last command received from the HPS is sent back to the HPS. If a read access is performed at the offset 1, the status of the FIFO of *Local_RX* is sent. If a read access is performed at the offset 2, the event data on the output of the *EventFifo* are sent. If a read access is performed at the offset 3, the status of the *EventFifo* is sent. If a read access is performed

at the offset 4, the data on the output of the *RegEventFifo* are sent. Finally, if a read access is performed at the offset 5, the status of the *RegEventFifo* is sent. *iofifocontrol* is connected to the Lightweight HPS-to-FPGA bus and, for this reason, the maximum theoretical bandwidth of the transmission of the event data with this system was about 11 kHz or about 6 MB/s [41];

- **DDR3_manager:**

it is connected directly to the *port0* of the FPGA-to-HPS SDRAM interface, which provides full access to the on board 1 GB SDRAM memory. It is used as a fast, low-latency preferential path for the event data coming from the *EventFifo* to the HPS. This block is the major upgrade to the board system, thus it will be discussed more in detail in one of the following section.

3.5.5 Debug entities

These entities are not actually part of the DAQ system, but were used for debug and test purposes during the design phase. The board is provided with 2 simulation entities, *Trigger_generator* and *BeamSimu*. The former is a simple pulse generator, which can be used as a trigger generator to act as the trigger system of the experiment if this one is unable to operate. The latter is again another pulse generator, but, while *Trigger_generator* is fixed at a specific rate, this actually simulates the variable rate of a real beam of particles. The trigger rate varies from less than a hertz to about 5 kHz in few seconds. As it will be explained in section 3.10, we never used *Trigger_generator* since the trigger system was operational and we used an external bench pulse generator, while we used *BeamSimu* to simulate in a more realistic way the real beam of incoming particles.

3.6 DDR3_manager

The original version of the DAQ firmware used the *iofifocontrol* component to send command, receive responses and, above all, transmit the data from the sensors. As a consequence, since every communication was managed by the Lightweight HPS-to-FPGA bridge and the data were moved in chunks of just 32 bit only when the HPS was requesting them, the system was bottlenecked, which caused the bandwidth of the whole system to be reduced at about 11 kHz of DAQ rate or about 6 MB/s of transfer rate [41]. This component was created in order to increase the bandwidth, driving high priority traffic away from the performance bridge and allowing the HPS to access data at the much faster clock speed of the DDR3 SDRAM. It was designed with the purpose of implementing a data transfer that should be as reliable and versatile as possible, thus it is totally independent from the data which receives and sends. This should facilitate a future upgrade of the system, if needed.

The component has 4 main groups of signals. The first is composed by *clock* and *reset*. The second is the Avalon-MM master interface, the "sink" of the transmission, which is made up of a 32 bit *address* signal, *read* and *readdata* (both unused, present for future development), *write*, *writedata* (32 bit), *byteenable* (4 bit, all connected to high state, not necessary but present for future development), *waitrequest*, *burstcount* (fixed to 1, unused) and *readdatavalid* (unused). It is connected directly to the *port0* of the FPGA-to-HPS SDRAM interface, which provides full access to the on board 1 GB DDR3 SDRAM memory. The third category is the "conduit_FIFO": this interface connects to the "source" of the data to transfer, that is the output of *EventFifo*. The port is composed by *acknowledge*, *empty* flag of the FIFO and a 32 bit data bus. The fourth category is used for synchronization with the HPS, and is made up of two 32 bit buses (one input driven by the HPS GPIOs called *hps_side_RAM_ctrl_reg* and one output to the HPS GPIOs called *fpga_side_RAM_ctrl_reg*) and an *enable* signal. The pinout of the block and its connection inside the Platform Designer are shown in figure 3.13 and 3.14 respectively.

DDR3_manager is provided with a specific reserved space of 16 MiB in the 1 GB HPS DDR3 RAM. It must write only in this space, otherwise it could

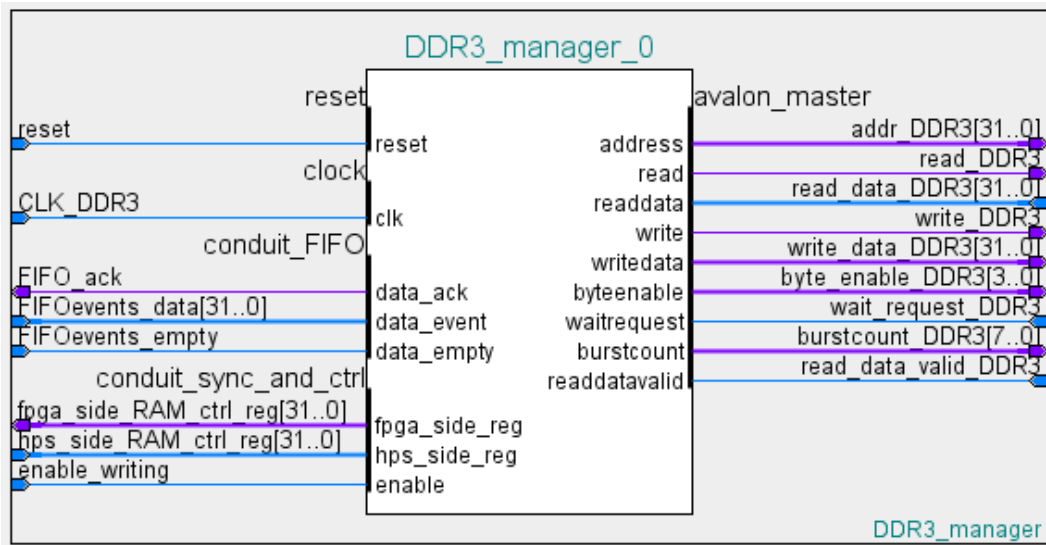


Figure 3.13: The DDR3_Manager custom component.

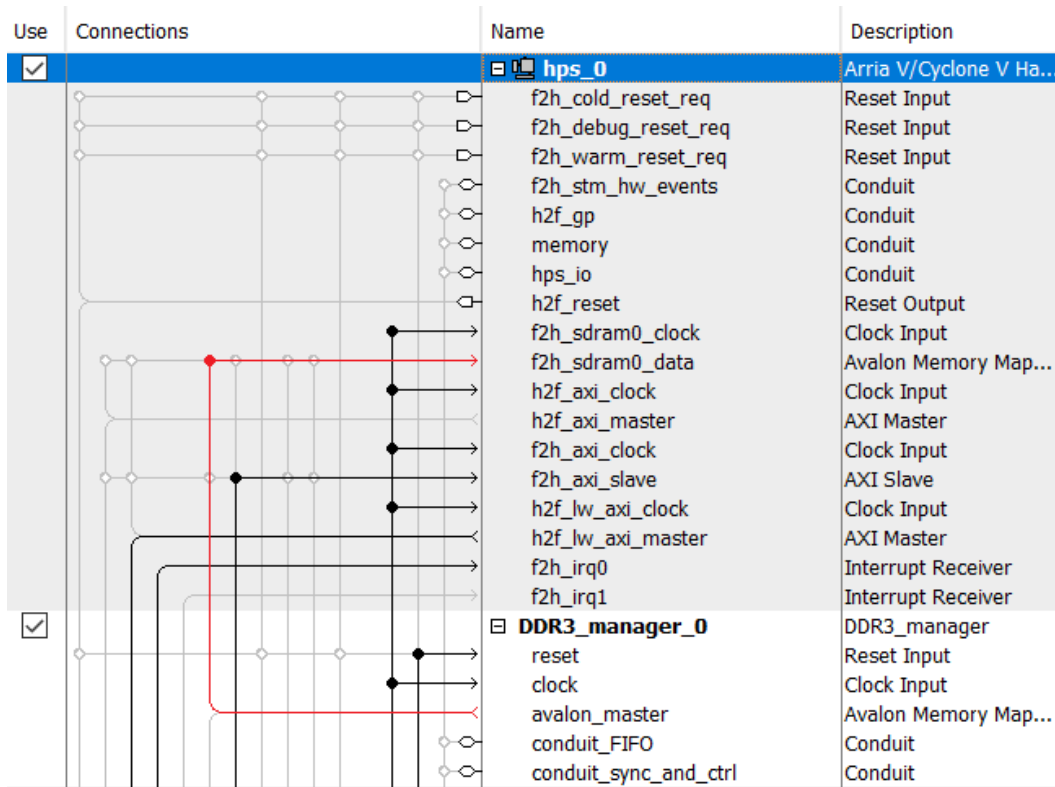


Figure 3.14: Detail of the interconnection between the DDR3_Manager custom component and the HPS. The red lines highlights the direct connection to the SDRAM port.

cause kernel panic². This memory is used as a circular buffer, which is written by *DDR3_manager* and read by the software running into the HPS. The lowest 30 bits of the *hps_side_RAM_ctrl_reg* and *fpga_side_RAM_ctrl_reg* buses are pointers (called *p_hps* and *p_fpga* respectively), the 31st bits are flags called *wrapped_around_hps*, in the first case, and *wrapped_around_fpga*, in the second. The 32th bits are not used. *p_hps* points to the address which will be read in the next HPS read access, while *p_fpga* points to the address which will be written in the next *DDR3_manager* write access (figure 3.15). After every bus operation, the pointer related to the master who performed the operation is incremented. The two *wrapped_around* flags toggle when the respective pointer wraps around and points again to the base address of the reserved memory. This is necessary, since *p_hps* cannot overcome *p_fpga*, and this is achieved thanks to the "xor" of the two *wrapped_around* flags. In fact, the latter is high only when one of the two pointer has wrapped around.

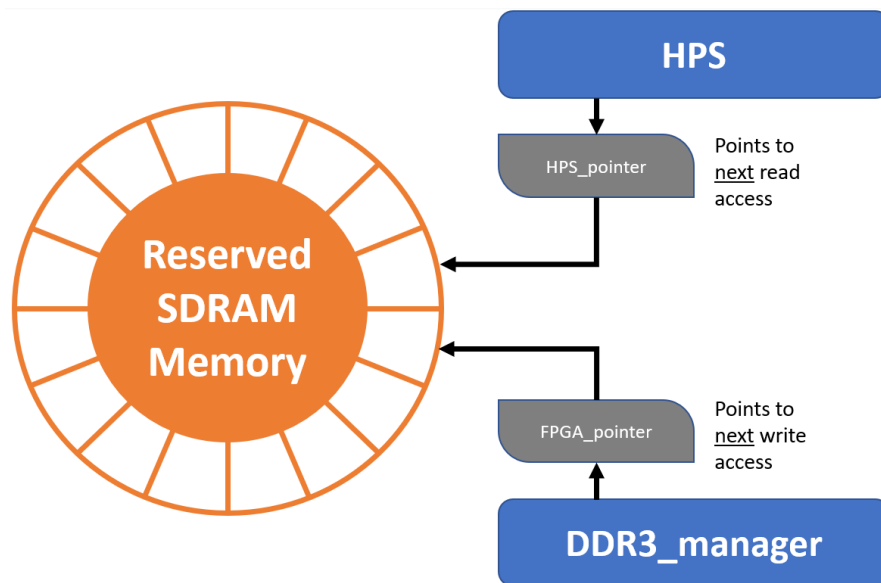


Figure 3.15: Schematic showing the circular buffer and the two pointers.

This is how *DDR3_manager* operates: if the *enable* signal is low, the module is disabled. This signal is set or reset inside a control register in the FPGA. If the

²A kernel panic is a safety measure taken by an operating system's kernel upon detecting an internal fatal error in which it either is unable to safely recover or cannot have the system continue to run without having a much higher risk of major data loss.

enable is high the block checks continuously the *empty* flag of the *EventFifo* to probe if event data are present. When *empty* goes low, the module checks if there is enough space in the predefined reserved RAM memory using *hps_side_RAM_ctrl_reg* and *fpga_side_RAM_ctrl_reg* and, if so, starts an Avalon write access to the SDRAM at the address pointed by *p_fpga*. When the transmission is completed, *p_fpga* is incremented by 4 (32 bits are 4 bytes and the RAM is byte-addressable) and if the result of the operation is outside the defined reserved memory, the *p_fpga* is set to the base address and the *wrapped_around_fpga* is toggled.

3.7 HPS software

The FPGA firmware would be useless without the management software running in the operating system *Ångström* Linux distribution 2016.12. The OS, based on Linux kernel 4.1.33 LTSI [36], despite having some speed, latency and responsiveness disadvantages compared to running the HPS bare-metal, allows us to maintain a high-level approach and save much time, with very little effect on performance for our specific application. The software, written in *C++* by Silvia Biondi and others, can receive commands from the Head PC via Ethernet interface and send back FPGA register values, the data of the events from the sensors and responses to the queries of the Head PC. The structure of the code is hierarchical, with custom, specific classes designed to handle the hardware of the FPGA and to interact with the already discussed blocks inside the *Platform Designer*.

The code is based on three main threads, which run in parallel during the "running" state (see chapter 3) of the DAQ system (figure 3.16). The first one is the starting thread which handles the creation and destruction of the other two threads and sets up the configurations of the board. The other two threads are used solely to transfer event data from the reserved RAM filled by the FPGA to the Ethernet interface: the *producer* thread retrieves event data from the RAM and stores them in a "soft" circular buffer, while the *consumer* thread recovers these data and sends them through Ethernet. The chosen protocol for data transmission is TCP/IP, which is slower, but safer than UDP, thus more suitable to our purposes.

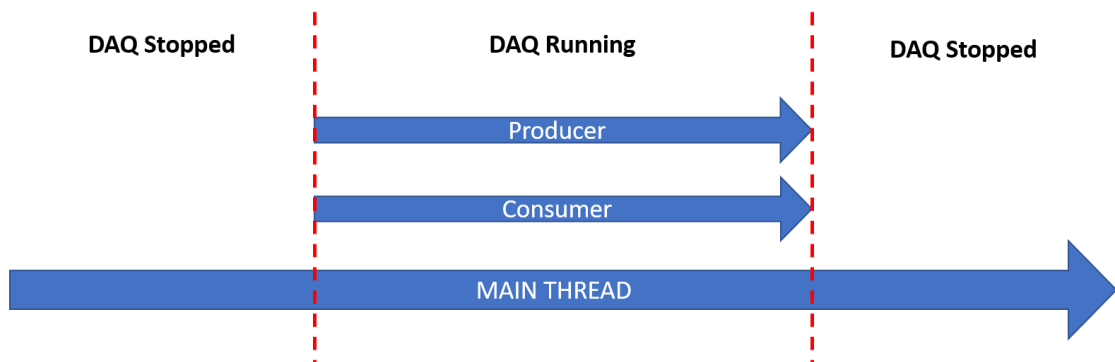


Figure 3.16: Threads creation and destruction during the operation of the DAQ.

The base class used by the threads to communicate with the FPGA is called *FpgaIOBase* (files are called *FpgaIOBase.h* and *FpgaIOBase.cpp*), while the derived class is called *FpgaInterface* (files called *FpgaInterface.h* and *FpgaInterface.cpp*). A schematic representation of the system is shown in figure 3.17. Most of the system improvements were made in these two classes, in order to allow the software to exploit the new transmission system using the RAM DDR3 as a circular buffer. Only the added features and functions will be discussed.

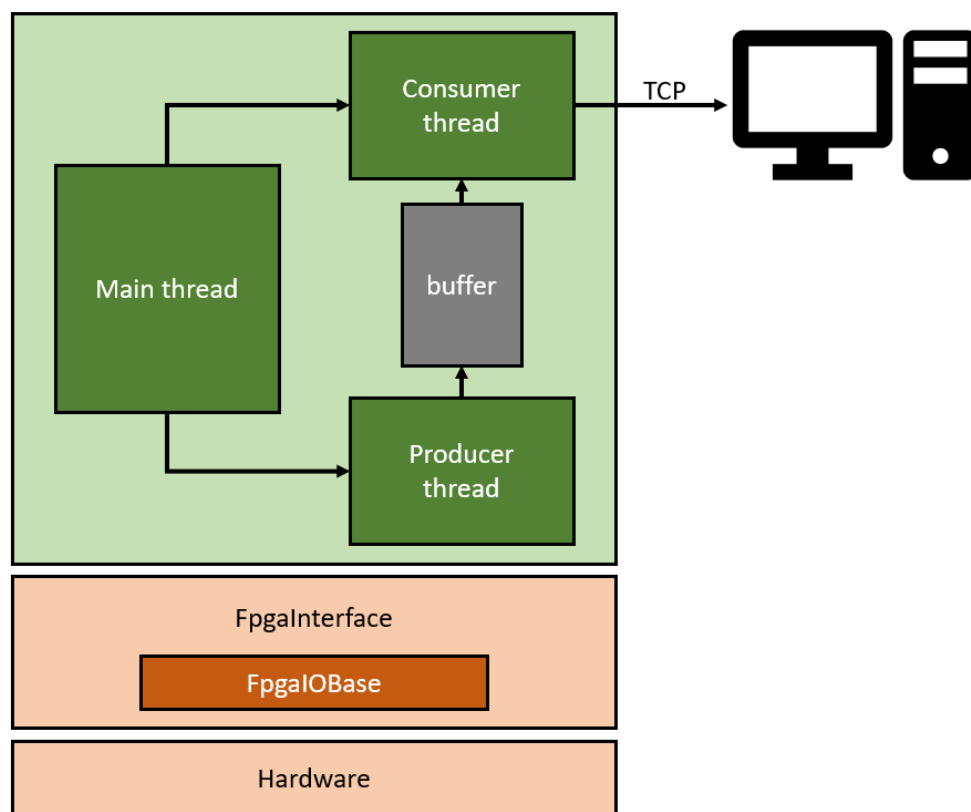


Figure 3.17: The class composed by *FpgaIOBase* and *FpgaInterface* offers an "abstraction layer" of the hardware to facilitate the development of the main software.

As expected, the lowest level functions have been added in the base class, therefore inside *FpgaIOBase.h* and *FpgaIOBase.cpp*. In these classes, the following functions can be found:

- **int openMem_RAM()**
first of all, this function accesses physical OS memory using the *open* function at */dev/mem*. If this operation is successful, it maps the reserved memory, that is the memory used exclusively by FPGA and HPS to exchange event data, inside the virtual address space of the process. Furthermore, GPIOs of the HPS (used as the already mentioned pointers for the transfer system) and some control registers of the SDRAM controller are mapped in this function. After the mapping, the SDRAM bridge is resetted and the calibration flag of the RAM is checked. Returns 0 if successful;
- **int closeMem_RAM()**
this function is used when the program is closing to deallocate memory mapped inside *openMem_RAM()*. Returns 0 if successful;
- **uint32_t Read_32bit_RAM(uint32_t addr_offset)**
returns the 32 bit word at the specified offset *addr_offset* inside the reserved RAM memory;
- **void Write_32bit_RAM(uint32_t data uint32_t addr_offset)**
writes the specified 32 bit word *data* at the specified offset *addr_offset* inside the reserved RAM memory;
- **uint8_t Read_byte_RAM(uint32_t addr_offset)**
returns the byte at the specified offset *addr_offset* inside the reserved RAM memory;
- **void init_RAM(uint32_t data)**
initializes every 32 bit word of the reserved memory to the value specified by *data*;
- **void Write_HPS_RAM_control(uint32_t addr_offset, uint32_t wrapped_around)**
sets the 32 bit general purpose output pins of the HPS with the passed values of the *HPS_pointer* (*addr_offset*) and HPS wrapped_around flag (*wrapped_around*). This function writes what is called *hps_side_RAM_ctrl_reg* in the FPGA firmware;

- **void Read_FPGA_RAM_control(volatile uint32_t* addr_offset, volatile uint32_t* wrapped_around)**

stores inside *addr_offset* and *wrapped_around* the values of the *FPGA_pointer* and *FPGA wrapped_around* flag, respectively. It reads what is called *fpga_side_RAM_ctrl_reg* in the FPGA firmware.

Inside the daughter class, *FpgaInterface.cpp* and *FpgaInterface.h*, the function **void readOfffo_on_RAM(std::vector<uint32_t> & data , uint32_t nmax)** was added first. It reads the available data inside the FIFO of events till a maximum of *nmax* words (default is 2000). The data are returned inside the *data* sequence container that encapsulates a dynamic size array of *uint32_t*. Specifying a too high *nmax* may lead to Kernel panic. First of all, the function calls *Read_FPGA_RAM_control* in order to read the current values of the pointer and *wrapped_around* flag of the FPGA. Then a "xor" operation is made between the *wrapped_around* flags of the FPGA and HPS and, based on this and on the relative position of the pointers of FPGA and HPS, the RAM memory is read using the *Read_32bit_RAM* function. When the pointers point at the same RAM cell (so there is nothing to read) or the maximum number of read data words (*nmax*) is reached, the function update its pointer and *wrapped_around* flag using the function *Write_HPS_RAM_control* and returns.

At a later time, another function was added to the *FpgaInterface* class, that is **void readOfffo_on_RAM_fast(std::vector <uint32_t> & data, uint32_t nmax)**. Its operation is the same, but the main difference with the first one is that the latter has the visibility of the internal pointers and flags and this allows it to access directly chunks of data and move them atomically to another memory location, reducing the overhead caused by calling several times other functions, such as the *Read_32bit_RAM*. Furthermore, it keeps track of the events read and only reads intact events, leaving on the RAM those which are being still transferred. This function will be used in the real system and it is the one used to evaluate the performance of the board as it will be discussed in chapter 4.

3.8 Reserved memory

As already mentioned, the new system of transmission between the FPGA and HPS, handled by software in the case of the latter and by the *DDR3_manager* block in the former, uses a reserved part of the main 1 GB DDR3 SDRAM memory on the board. This "shared" or "reserved" memory is necessary because the two portions of the SoC, which are the HPS and FPGA, are independent from each other, therefore an incautious write operation of the FPGA fabric could overwrite vital data of the OS running on the HPS causing kernel panic and the reset of the board, at best. For this reason, a small part of this memory is reserved and, in this way, the operating system ignore it and the FPGA can write without compromising the OS functionality.

In order to tell the operating system to ignore a specific part of the memory for its normal operation, some modifications of the *Device Tree* were needed. A *Device Tree* is a tree data structure with nodes that describe the physical devices in a system. It is loaded by the bootloader at start-up together with the kernel image [42]. After the node defining the main memory was found, a new node was added in which we declared a part of that memory as reserved. Furthermore, the additional attribute "no-map" was added, in order to indicate that the operating system must not create a virtual mapping of the region as part of its standard mapping of system memory, nor permit speculative access to it under any circumstances other than under the control of the device driver using the region [43]. A small snippet of code is shown in the following page, which includes what has been added. The upper "memory" node was already present and indicates to the kernel the amount of memory available (1 GB). On the bottom, the node "reserved-memory" was added and specifies the attributes of the reserved address space: it starts at address 0xC0_0000 and it has a width of 16 MiB.

```
memory {
    device_type = "memory";
    reg = <0x0 0x40000000>;
};

reserved-memory {
    #address-cells = <0x1>;
    #size-cells = <0x1>;
    ranges;

    buffer@0x00c00000 {
        no-map;
        reg = <0xc00000 0x1000000>;
    };
};
```

3.9 Additional feature: ADC

The implementation of the analog-to-digital functionalities of the DE10-Nano board was a later addition to the system. At the moment, none of the 8 channels of the LTC2308 ADC have a defined application, but they are expected to be used to monitor temperatures and significant voltages inside the DAQ chain, specifically from the silicon sensors to the board itself.

For the implementation, the *ADC Controller for DE-series Boards IP Core* by Intel was used, which provides access to the analog-to-digital converters found on the DE-series boards. It controls all required digital signals both to and from the ADC, and provides the user with 8 12-bit signals which represent the updated outputs of the corresponding channels which are continuously sampled. These signals are routed to the *Register_File* entity and mapped on 4 monitor registers, each containing the values of 2 channels (see Appendix A). In such a way, the ADC values can be retrieved via the *iofifocontrol* entity together with the other monitor and control registers of the DAQ board.

For this purpose, the function `uint16_t read_ADC_channel(uint8_t ch_index)` was created: when called, it returns the value of the channel specified by `ch_index`.

3.10 Synthesis, compilation and testing

The *DDR3_manager* block has been written in VHDL using the software *Quartus Prime 18.1 Standard Edition* and simulated using *ModelSim Intel Starter Edition 10.5b*. A screenshot of the simulation is shown in figure 3.18.

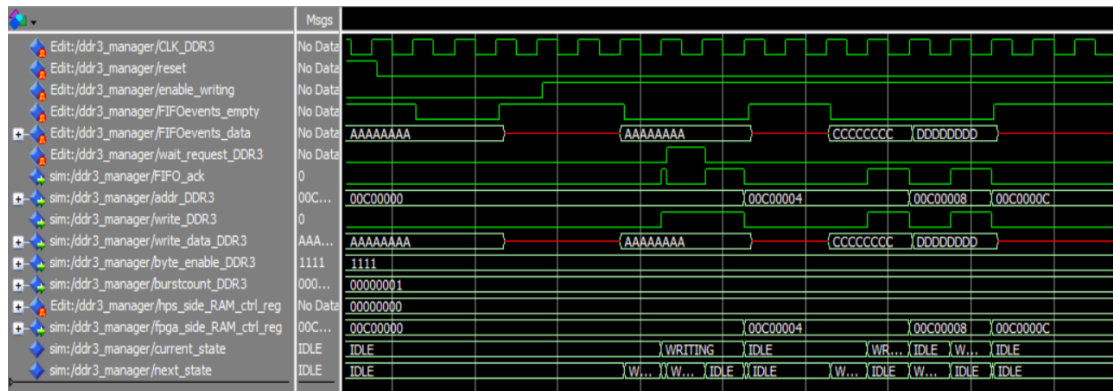


Figure 3.18: A small portion of the simulation of the *DDR3_manager* block. Three write transitions are shown: the first *empty* pulse is ignored as the *enable* signal is low; the first write transition have a single cycle of *waitrequest*; the last two are two simple transitions very close to each other.

After the success of the single block simulation, the additional firmware was not implemented immediately inside the main experiment code at first. On the contrary, in order to have a cleaner and more controllable workspace and test the firmware and the software together, a specific simple firmware was created and the improvements were tested inside this controlled and simulated "space". Specifically, the *DDR3_manager* block has been instantiated inside a "simulation" entity in which the logic was programmed to act as a FIFO. This false memory could be programmed to appear empty or full and the data were a combination of counters and fixed numbers.

Regarding the C++ code, the software was compiled using the *arm-linux-gnueabi* cross compiler for *armhf* architecture via the *Intel SoC FPGA Embedded Development Suite (SoC EDS)*, which is a comprehensive tool suite for embedded software development on Intel SoC FPGAs. The approach followed to debug the firmware has been used also for the software: a simple software has been written to read commands from the terminal and execute simple task for debug purposes,

such as reading and writing the status of the pointers or simulate a full reading and writing transition on RAM as it happens on the real system. This code used the same C++ classes of the real software in order to test the new functions which were added.

The output of one of these tests is shown here: the FPGA pointer is 40, while the HPS pointer is 30. Combined with the fact that the two wrapped around flags are both 1, this indicates that the FPGA has written $40 - 30 = 10$ words on the RAM which are waiting to be read. Therefore the read words are exactly 10 and, after the reading, the HPS and FPGA pointers point at the same word cell.

```
-Before reading- HPS_wrapped_around = 1 | HPS pointed offset = 30
-Before reading- FPGA_wrapped_around = 1 | FPGA pointed offset = 40

Reading and printing new data:
8888883C
9999993D
AAAAAA3E
BBBBBB3F
CCCCCC40
DDDDDD41
EEEEEE42
FFFFFF43
44
11111145

Current HPS_wrapped_around = 1 | Current HPS pointed offset = 40
Current FPGA_wrapped_around = 1 | Current FPGA pointed offset = 40
```

For what concerns the reserved memory, the added node in the device tree was accepted by the kernel at boot and using the shell command `cat /proc/iomem` it was possible to confirm that the modification was successful, as shown in figure 3.19.

```
root@de10nano3:~# cat /proc/iomem
00000000-00bfffffff : System RAM
  00008000-00aa80b3 : Kernel code
  00b2e000-00bf8f0b : Kernel data
01c00000-2f7fffffff : System RAM
ff702000-ff703fff : /soc/ethernet@ff702000
ff704000-ff704fff : /soc/dwmmc0@ff704000
ff706000-ff706fff : /soc/fpga-mgr@ff706000
ff708000-ff708fff : /soc/gpio@ff708000
ff709000-ff709fff : /soc/gpio@ff709000
ff70a000-ff70afff : /soc/gpio@ff70a000
ffb40000-ffb4ffff : /soc/usb@ffb40000
ffb90000-ffb9001f : /soc/fpga-mgr@ff706000
ffc02000-ffc0201f : serial
ffc03000-ffc0301f : serial
ffc04000-ffc04fff : /soc/i2c@ffc04000
ffd02000-ffd02fff : /soc/watchdog@ffd02000
ffd05000-ffd05fff : /soc/rstmgr@ffd05000
ffe01000-ffe01fff : /soc/amba/pdma@ffe01000
  ffe01000-ffe01fff : /soc/amba/pdma@ffe01000
ffff0000-ffffffff : ffff0000.sram
root@de10nano3:~#
```

Figure 3.19: Using the command `cat /proc/iomem`, the address mapping can be seen. Notice the "hole" from `0x00BF_FFFF` to `0x01C0_0000` in the RAM addresses, which is where the reserved memory resides.

When every new feature was simulated and tested in this controlled environment, these were embedded inside the main firmware and software and adapted to the already existing code. New control and monitor registers were added, in order to configure and operate with these features, such as those which contain the raw values of the 8 analog channels of the ADC, those which contain the FPGA and HPS pointers and the register to select the input of the *Event_Builder* (real sensors or simulated data).

The complete firmware uses 4808 out of 41910 ALM³ (11%), 7536 registers, 751872 out of 5662720 block memory bits (13%) and 1 DLL (figure 3.20). It is about 4000 lines long (about 15 entities). The firmware is clocked at 50 MHz, but the timing analyser integrated inside Quartus specifies that the system could run at 80.11 MHz without problems, even at 100 °C, at -40 °C and with slow transistors. In figure 3.21, a visual representation of the chip utilisation is shown. The two ARM cores are clocked by default at 800 MHz, while the DDR3 RAM memory is clocked at 400 MHz by default.


| Flow Summary | |
|--|---|
|  <<Filter>> | |
| Flow Status | Successful - Thu Jun 11 19:03:11 2020 |
| Quartus Prime Version | 18.1.0 Build 625 09/12/2018 SJ Standard Edition |
| Revision Name | DE10DAQ |
| Top-level Entity Name | DE10DAQ_TOP |
| Family | Cyclone V |
| Device | 5CSEBA6U2317 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 4,808 / 41,910 (11 %) |
| Total registers | 7536 |
| Total pins | 230 / 314 (73 %) |
| Total virtual pins | 0 |
| Total block memory bits | 751,872 / 5,662,720 (13 %) |
| Total DSP Blocks | 0 / 112 (0 %) |
| Total HSSI RX PCSs | 0 |
| Total HSSI PMA RX Deserializers | 0 |
| Total HSSI TX PCSs | 0 |
| Total HSSI PMA TX Serializers | 0 |
| Total PLLs | 0 / 6 (0 %) |
| Total DLLs | 1 / 4 (25 %) |

Figure 3.20: Compilation report of the firmware.

³Adaptive Logic Module

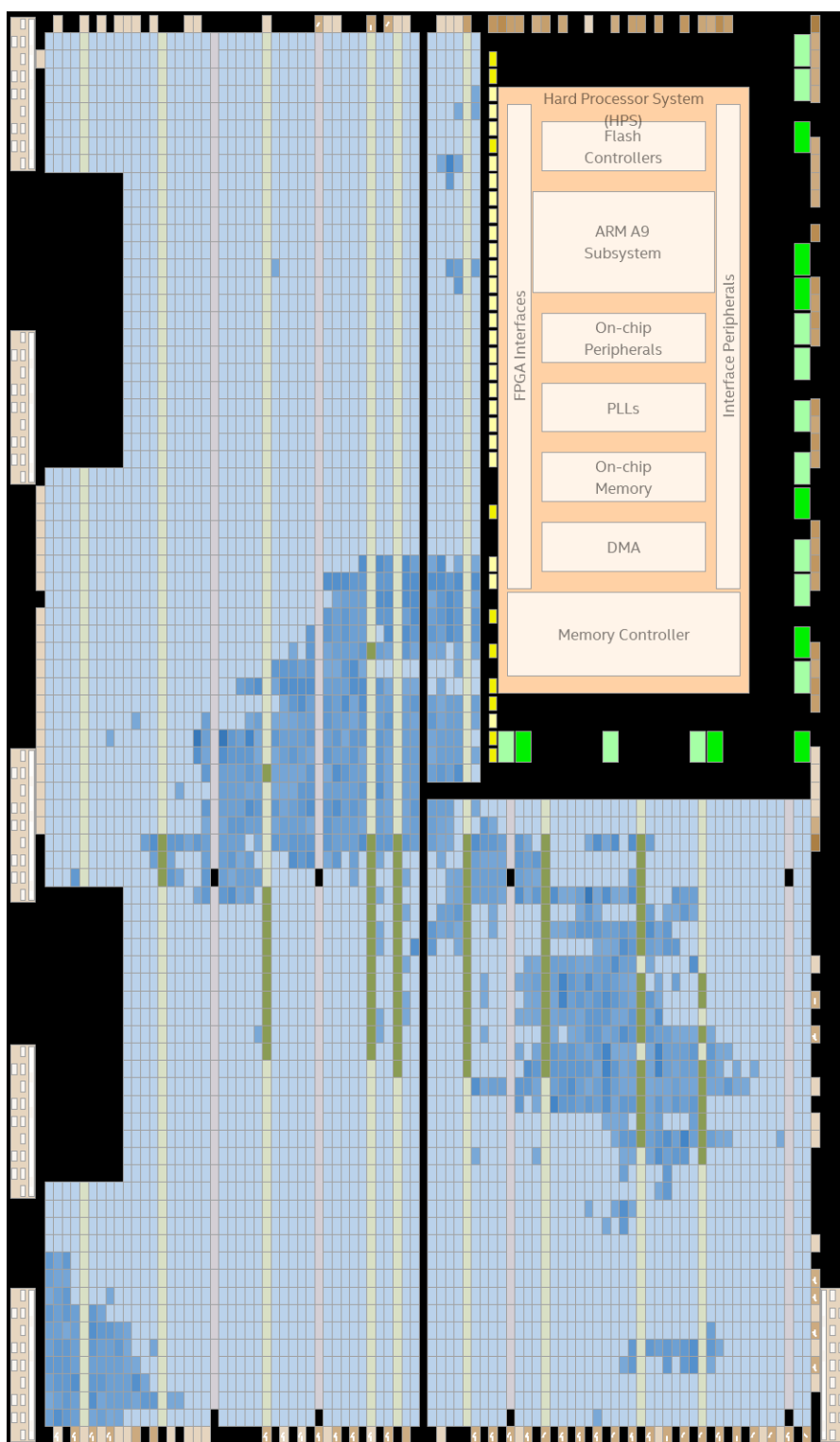


Figure 3.21: Representation of the internal cells of the FPGA chip. The intensity of the blue color increases the more the cell is used.

At this point, the entire experiment firmware and software had to be loaded on the FPGA. The Quartus output file .sof was converted to a .rbf file using the terminal command `% QUARTUS_ROOTDIR %//bin64//quartus_cpf -c -o bitstream_compression=on DE10DAQ.sof DE10DAQ.rbf pause` and copied to the directory `/lib/firmware`. Since the chosen flashing method was to use the Linux OS as the "programmer" of the FPGA, a special set of steps was required. First of all, a file .dtbo was created and put in the same directory in which the firmware .rbf resides: this creates a new overlay node in the live device tree that represents the FPGA programmable logic in which the firmware .rbf will be dumped. The code of this file in a readable format before the conversion in the .dtbo extension is shown here:

```
/dts-v1/;
/plugin/;

/{
    fragment@0 {
        target-path = "/soc/base-fpga-region";
        #address-cells = <1>;
        #size-cells = <1>;
        __overlay__ {
            firmware-name = "DE10DAQ.rbf";
        };
    };
};
```

At this point, a script `mount.sh` is created. This script is executed in the Linux terminal when the firmware needs to be downloaded on the FPGA logic. The script is shown here:

```
#!/bin/sh
mkdir /config
mount -t configfs configfs /config
rmdir /config/device-tree/overlays/test
mkdir /config/device-tree/overlays/test
```

```
ls -l /config/device-tree/overlays

cd /lib/firmware

echo ./DE10DAQ.dtbo > /config/device-tree/overlays/test/path
```

The software is compiled including all the provided libraries for the SoC and the file *hps_0.h*, which contains the correct Platform Designer constants used by the lower abstraction layer C++ classes. The former is created from Quartus using a script called *generate_hps_qsys_header.sh*, which is shown here:

```
#!/bin/sh
sopc-create-header-files \
"./soc_system.sopcinfo" \
--single hps_0.h \
--module hps_0
```

The DE10-Nano board loaded with the firmware and software has been tested in the laboratory. The test setup was composed by an external pulse generator to simulate the signals coming from the start counter, the trigger system (board CAEN V2495), a DE10-Nano board, another experiment board (a TDC⁴) and a PC connected to everything, which acted as both the Head PC and the Storage PC. With the aid of an oscilloscope and the management software of the PC, the internal signals and event data were checked and tested and after some troubleshooting and the correction of few bugs, the system was working as intended. In figure 3.22 the main signals as seen from the oscilloscope are shown.

⁴Time to digital converter.



Figure 3.22: The yellow signal is the pulse coming from the pulse generator, the green one is the trigger generated by the trigger control, which is forwarded to the DE-10. The reddish signal is the busy generated by the DE-10, while the blue one is the busy generated by the trigger control (experiment or global busy).

An example of event as seen from the debug console of the Head PC is shown here. This is event number 2 (Event # 2), which is 138 words long (size: 138). The first word, which is the length of the event in words (0000008a), then two fixed headers (eadebaba and 00eade00) follow, then the event number (00000002), the value of the *BCOCounter* (00130249) and a time information (clock ticks, 1d1a1383). At the end of the event two footers have been placed, that are fafefafe and bacca000. These headers and footers are added by the *Event_Builder*. All the data between these words, from 00000082 up to ffffffff, are the payload, which is the event coming from the sensors (or *Event_Simulator*, in this case) that has been acquired by the DE10 and has to be shipped to the central DAQ. In this case the format of the payload is a size word, three header words, actual sensor data and three footer words.

| | | | | | |
|------------|-----------|----------|----------|----------|----------|
| Event # 2, | size: 138 | | | | |
| 0000008a | eadebaba | 00eade00 | 00000002 | 00130249 | 1d1a1383 |
| 00000082 | aaaaaaaa | bbbbbbbb | ccccccc | 000238e5 | 06127dd4 |
| 0712ff8c | 0812ff19 | 0912fe32 | 0a12fc64 | 0b12f8c9 | 0c12f192 |
| 0d12e324 | 0e12c649 | 0f128c92 | 10121924 | 1112366c | 121268fc |
| 1312d5dc | 1412abb9 | 15125772 | 1612aac1 | 17125582 | 1812af21 |
| 19125e42 | 1a12b8a1 | 1b127142 | 1c12e6a1 | 1d12cd42 | 1e129a84 |
| 1f123509 | 20126e37 | 2112d84a | 2212b094 | 23126129 | 2412c677 |
| 25128cef | 261219df | 2712379a | 28126b11 | 2912d207 | 2a12a40f |
| 2b12481f | 2c12941a | 2d122834 | 2e12544c | 2f12acbc | 30125979 |
| 3112b6d7 | 32126daf | 3312df7a | 3412bef4 | 35127de9 | 3612fff7 |
| 3712ffef | 3812ffdf | 3912ffbf | 3a12ff7f | 3b12feff | 3c12fdff |
| 3d12fbff | 3e12f7ff | 3f12efff | 4012dfff | 4112bfff | 42127fff |
| 4312fbda | 4412f7b4 | 4512ef69 | 4612ded2 | 4712bda4 | 48127b49 |
| 4912f2b7 | 4a12e56f | 4b12cadf | 4c1295bf | 4d122b7f | 4e1252da |
| 4f12a191 | 50124322 | 51128261 | 521204c2 | 53120da1 | 54121f67 |
| 55123aea | 561271f1 | 5712e7c7 | 5812cf8f | 59129f1f | 5a123e3f |
| 5b12785a | 5c12f491 | 5d12e922 | 5e12d244 | 5f12a489 | 60124912 |
| 61129601 | 62122c02 | 63125c21 | 6412bc67 | 651278cf | 6612f5ba |
| 6712eb74 | 6812d6e9 | 6912add2 | 6a125ba4 | 6b12b36c | 6c1266d9 |
| 6d12c997 | 6e12932f | 6f12265f | 7012489a | 71129511 | 72122a22 |
| 73125061 | 7412a4e7 | 751249cf | 761297ba | 77122f74 | 78125acc |
| 7912b1bc | 7a126379 | 7b12c2d7 | 7c1285af | 7d120b5f | 7e12129a |
| 7f122111 | 55555555 | eeeeeeee | fffffff | fafefafe | bacca000 |

After the system started to work as expected, the focus has been placed on the

evaluation of the performance of the board using the new transfer system based on the DDR3 RAM, that is described in the next chapter.

Chapter 4

Performance

The performance of the board has been evaluated taking into consideration every possible parameter in a controlled environment, with different configurations of the system. In fact, the performance of the DE10 Nano depends on a multitude of different parameters of the whole DAQ system, such as the protocols used for communication, the physical networking, the number of devices connected to the Head PC and, most of all, the methodology used to retrieve the data from the RAM and send it to the PC. Even the management software running on the Head PC could potentially have an impact on performance. In order to have the picture of the situation as complete as possible and measure the performance with precision, we tested the board in a combination of configurations. In all the tests performed, the average length of one event was 142.5 32-bit words.

First of all, in order to find the actual maximum performance of the acquisition system on the DE-10 Nano and how much the new transmission channel is effective, we made various tests without the transmission to the Head PC of the events. In the first case, the HPS was reading data from the RAM only one event at a time, only when a full event (from the headers to the footers) was present. The read event was then saved in another memory location and finally discarded instead of being transmitted. With this approach, the maximum data rate achieved was 4-5 MB/s (trigger rate of about 8 kHz). Reading one event at a time is an inefficient way of operation, since the HPS overhead is heavily present. In the second case, we decided to test the board working in its intended operation, thus reading from the RAM

as much data as possible regardless of the amount of events read. When reading chunks of data of maximum 1 MB, the bandwidth reached 60 MB/s (about 105 kHz). Since the block `DDR3_manager` is designed with a maximum throughput of 100 MB/s, we can conclude that, even in this case, the limitation resides in the HPS overhead.

At this point, two DE10 Nano (shown in figure 4.1) were embedded inside a simplified DAQ system, consisting of the boards themselves, the Head/Storage PC and the trigger board CAEN V2495 (trigger control). The acquisition of the data from the trigger board was disabled and the maximum chunk size was set to 20000 words. In this condition, the trigger rate was 17.6 kHz (data rate of about 10 MB/s) (figure 4.2). The decrease in performance is expected, since the board now is limited by the overhead of the DAQ system. In a second test, we found out how the maximum chunk size could affect performance: the external pulse generator was set to a very high frequency (30 kHz) and the data rate was measured changing the block size every time. In figure 4.3, the result can be seen: the saturation at 17.6 kHz is reached with a chunk size of about 5000 words.

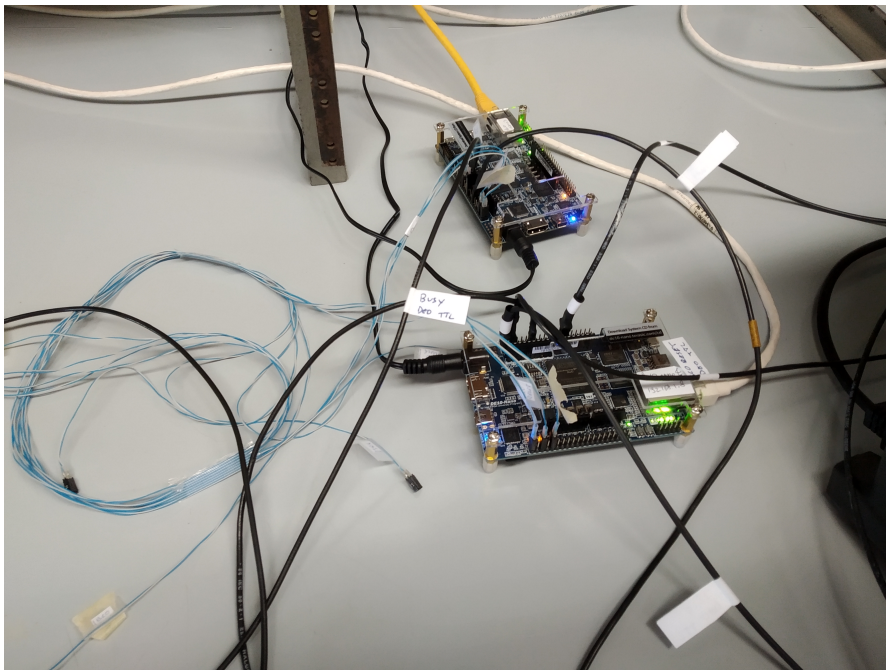


Figure 4.1: The two boards DE10 Nano used in the tests.

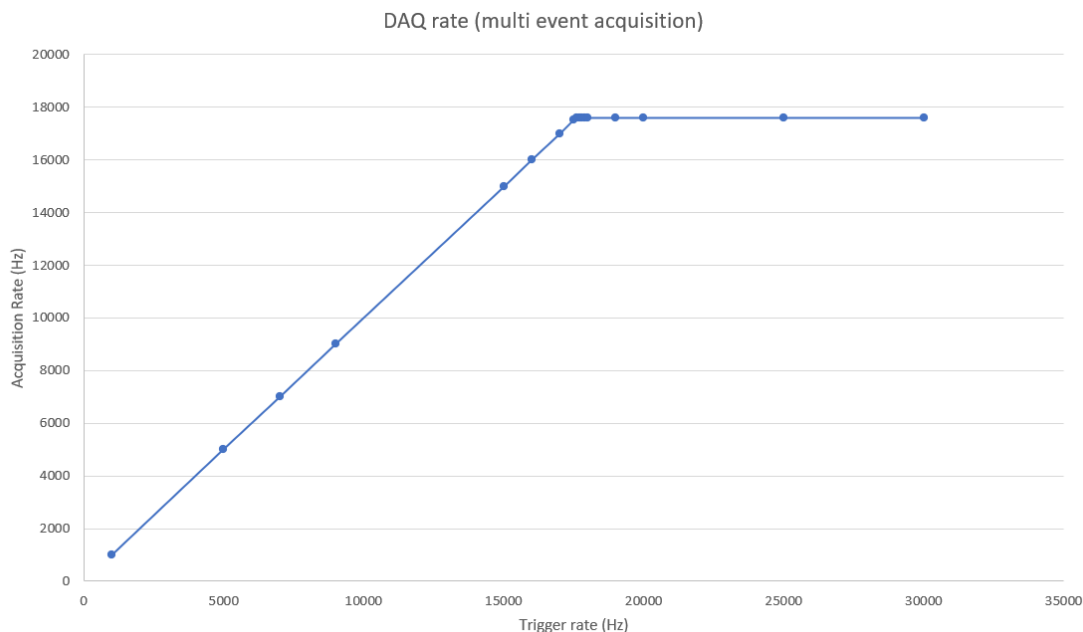


Figure 4.2: Graph showing the actual DAQ rate in the case in which two boards DE10 Nano are connected to the DAQ system and the data are retrieved only from these.

In another test only one DE10 Nano was connected, but the acquisition from the trigger board V2495 was enabled. The Head PC was configured to read from the board only one event at a time. The maximum trigger rate achieved was 1.44 kHz and this resulted in a data rate of about 821 kB/s (figure 4.4). As shown in figure 4.5, the DE-10 Nano board was easily managing the rate, but the system could not keep up. From this setup, we could deduce that the board V2495 was the slowest board on the system, creating the highest overhead in our tests. Finally, in the last test, two DE10 Nano were connected to the experiment and the Head PC was configured to read from the boards in chunks of 20000 words. The acquisition from the trigger board V2495 was enabled. This resulted in an acquisition rate of 3.4 kHz, or about 2 MB/s. As expected, even in this case, the DE10 Nano boards were easily managing the sustained rate, but the trigger board was limiting the acquisition speed. The work of optimization on the DAQ system, especially on the management software and on the networking infrastructure, is still continuing, therefore an additional increase of the performance is expected in the future.

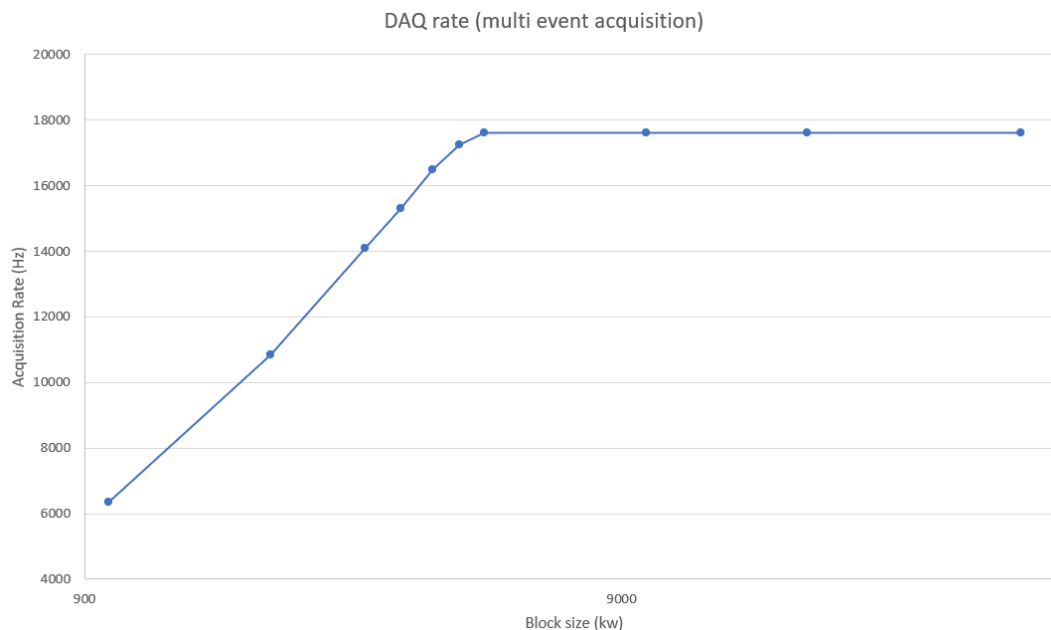


Figure 4.3: Graph showing the relationship between the maximum chunk size of the data read from the RAM and the actual DAQ rate.

Regardless of the overheads, therefore considering the tests which are reasonable to value for estimating the DE10 Nano firmware and software performance, the bandwidth of 17.6 kHz (data rate of about 10 MB/s) is a really good result for the experiment, since the networking protocols used in the experiment do not allow a perfect utilization of the Gigabit Ethernet mounted on the board, which will peak at about 10-20 MB/s (a comparable value to what a single DE-10 Nano is capable of). The maximum limit of the board alone of 60 MB/s will surely support future optimizations of the infrastructure if needed, without becoming a bottleneck. To increase the throughput, one proposal was to increase the data word length from 32 bit to 64 or even 128, but, given the circumstances, this revealed to be unnecessary.

In addition, some tests were made to ensure the reliability of the board during long acquisition sessions. Various status transition tests were performed, changing the status of the FSM remotely and checking if the board was responding correctly (config→start→stop→start→stop→unconfig→config→start→stop, etc) and during an integrity test, on 40 million events transferred from the FPGA to the RAM

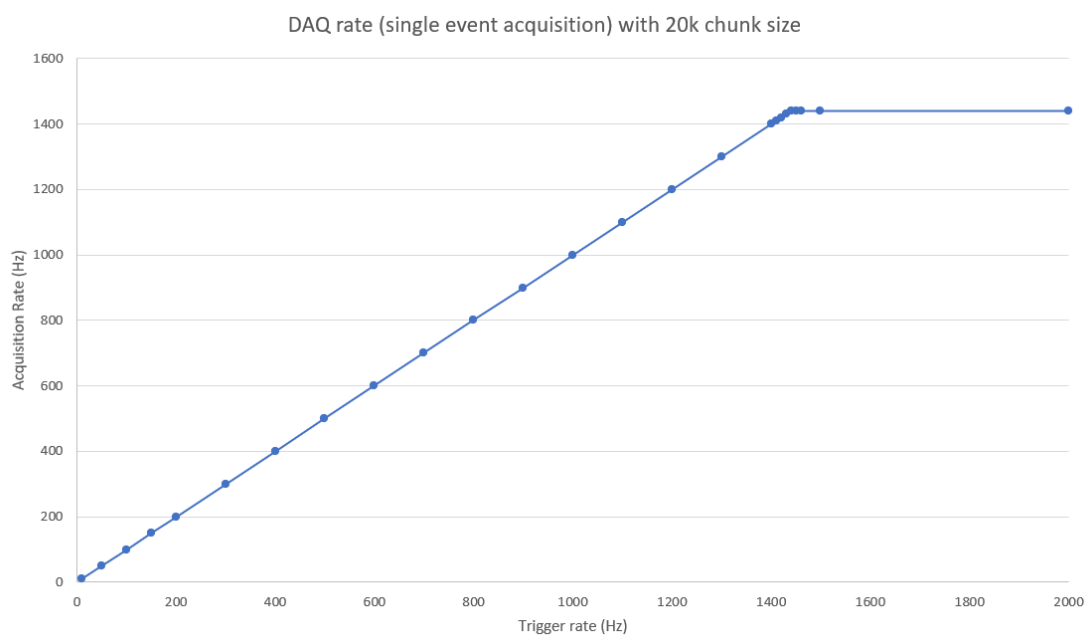


Figure 4.4: Graph showing the actual DAQ rate in the case in which the board is connected to the DAQ system, one event at a time is read and the data retrieval is made also from the trigger board.

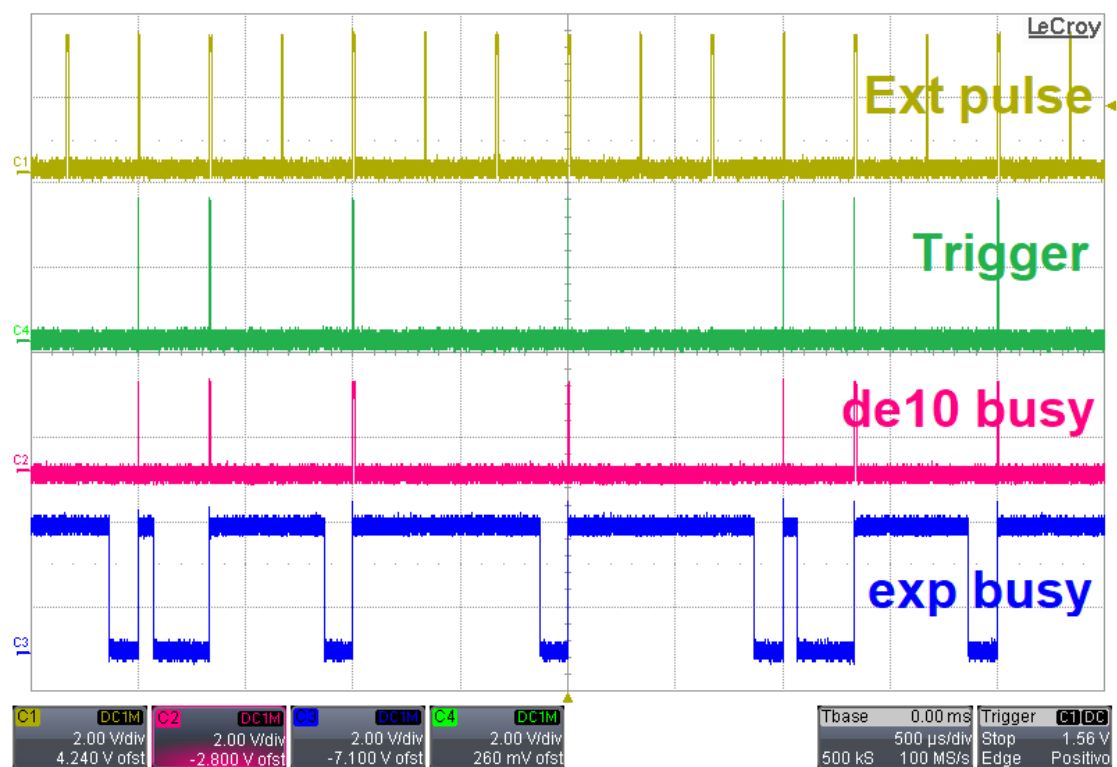


Figure 4.5: The main signals while acquiring single events. In this image it is clear that the DAQ is bottlenecking the system. In fact, the busy pulses of the Nano board (reddish) are a lot shorter than the main experiment ones (blue), which are still high when a new "particle" (yellow) arrives. This prevents the trigger control from forwarding the trigger signal (green) to the sensors.

and then read by the Head PC, no errors were found. During another long test run, with two DE10 and the acquisition from the trigger board V2495 enabled, which lasted about 60 hours (figure 4.6), on about 780 millions events, no errors were found at an average rate of 3.4 kHz. Considering the fact that a real acquisition session will last maximum 1 hour, this is an excellent result.

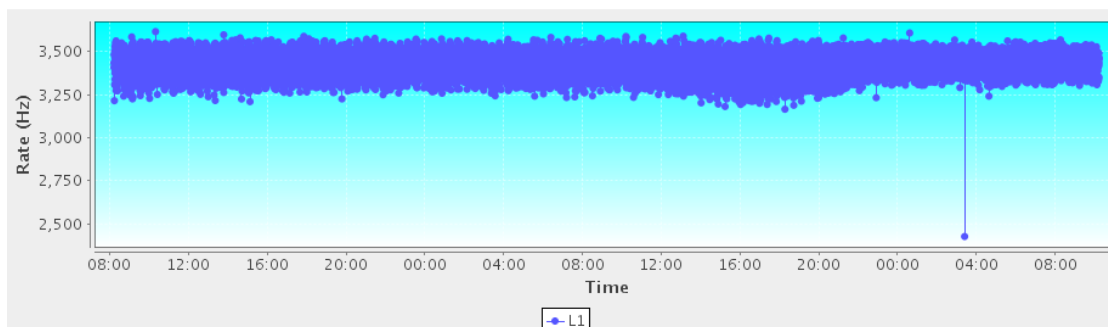


Figure 4.6: Graph showing all the readings of the DAQ in the stability test run which lasted about 60 hours.

Conclusion

This thesis presents the development of the new enhanced high-speed data channel which uses the RAM as a temporary circular buffer to increase the throughput of event data from the FPGA logic to the HPS inside a board DE-10 Nano used in the acquisition system of one of the detectors of the *FOOT* experiment. The board, based on an Intel SoC FPGA Cyclone V, is capable of managing the sensors and their circuitry or simulating dummy data, format and send them to the Head PC for further analysis, together with other data such as clock ticks since the beginning of the experiment or the value of *BCOcounter*. The acquisition starts when the trigger control of the experiment sets to high the trigger signal. During the acquisition, the board sets to high its busy signal, to tell the trigger control that it is not ready to receive more triggers. The board is equipped with a set of control and monitor registers, which can be read or written from the Head PC to view the status of a variety of subsystem of the board, such as status of the FIFOs or FSM state, or to configure the behaviour of the board.

The firmware has been written with the software Quartus II and simulated with ModelSim. It is written in VHDL and it is about 4000 lines long or about 15 entities. The software was compiled with the tool *arm-linux-gnueabi* cross compiler for *armhf* architecture via the *Intel SoC FPGA Embedded Development Suite (SoC EDS)*. The system was tested in laboratory alone and with other components of the experiment (trigger control, TDC, Head and Storage PC) and, after the correction of some minor bugs, it worked as intended, without timing or resources problems. The firmware is clocked at 50 MHz, while the ARM cores are clocked at 800 MHz. The DDR3 RAM is clocked at 400 MHz. The DE-10 Nano board boasts a theoretical maximum transmission bandwidth of 60 MB/s, which decreases to about 10 MB/s when the board is installed in the complete acquisi-

tion system, a limitation due to factors external to the board. These results are excellent and will support future development without becoming a bottleneck for the other systems.

In addition to the implementation of the new transfer system, a controller for the ADC and other accessory features were added, such as new required registers and the addition of several comments to increase the readability of the code in order to facilitate future development.

Two acquisition sessions have been already run in the configuration dedicated to light fragments in April 2019 and in February 2020 at GSI, but the data have not been published yet. A new data acquisition with the full configuration is expected to take place at CNAO in Pavia by the end of November 2020. We hope that the *FOOT* experiment will bring medicine to develop more effective, less invasive and safer treatment against cancer and help bring humans to Mars and, maybe, one day, to new worlds.

Appendix A

A.1 Control Registers

- **Dummy_Reg (offset 0):**
reserved.

| Bit | Description |
|--------|-------------|
| [31-0] | Reserved |

- **ToState_Reg (offset 1):**
this register is written with specific commands to order a change of state of the main finite state machine.

| Bit | Description |
|--------|--------------------------|
| [31-0] | FSM change state command |

- **VariableHeader_Reg (offset 2):**
customizable at run-time header for the *Event_Builder*.

| Bit | Description |
|--------|-------------|
| [31-0] | Header |

- **BusyMode_Reg (offset 3):**

sets the generation mode of the busy signal inside the *Event_Builder*. If '1', the block waits for the event to be dumped from the local FIFOs before resetting the busy signal.

| Bit | Description |
|-----|-------------|
| [0] | Busy mode |

- **FSMTimeOut_Reg (offset 4):**

minimum waiting time in clock cycles before the main finite state machine can pass from the EndOfRun state to the WaitingEmptyFifo state.

| Bit | Description |
|-------|-------------|
| [7-0] | FSM timeout |

- **RAM_interface_En_Reg (offset 5):**

enable of the DDR3_manager block.

| Bit | Description |
|-----|-------------|
| [0] | ENABLE |

- **simulated_acquisition_reg (offset 6):**

when active, the real interface to the sensor is excluded and the incoming data are simulated by the entity *Event_Simulator*.

| Bit | Description |
|-----|-------------|
| [0] | ENABLE |

A.2 Monitor Registers

- **Firmware_Reg (offset 0):**
firmware version.

| Bit | Description |
|--------|------------------|
| [31-0] | Firmware version |

- **FSM_StatusSignals_Reg (offset 1):**
it contains debug and status values related to the main finite state machine.

| Bit | Description |
|-------|------------------------|
| [28] | DAQ_IsRunning_Flag |
| [27] | DAQ_Reset_Flag |
| [26] | DAQ_Config_Flag |
| [25] | ReadingEvent_Flag |
| [2-0] | Main FSM current state |

- **Errors_Reg (offset 2):**
it contains values concerning errors or unexpected behaviours of the system.

| Bit | Description |
|-----|----------------------|
| [3] | InOutBothActive_Flag |
| [2] | InvalidAddress_Flag |
| [1] | ErrorNotRunning_Flag |
| [0] | ErrorBusy_Flag |

- **TriggerCounter_Reg (offset 3):**
the number of trigger pulses since the board went into the running state.

| Bit | Description |
|--------|----------------|
| [31-0] | TriggerCounter |

- **BCOCounter_Reg (offset 4):**

the number of rising edges of the experiment clock *BCOclock*.

| Bit | Description |
|--------|-------------|
| [31-0] | BCOCounter |

- **ClkCounter_Reg (offset 5):**

MSB of the main clock cycles counter.

| Bit | Description |
|--------|----------------|
| [31-0] | MSB ClkCounter |

- **LSB_ClkCounter_Reg (offset 6):**

LSB of the main clock cycles counter.

| Bit | Description |
|-------|----------------|
| [5-0] | LSB ClkCounter |

- **EB_Fifos_Reg (offset 7):**

it contains useful flags concerning the status of the two FIFOs inside *Event_Builder*.

| Bit | Description |
|---------|--------------------------|
| [31] | EBFull_Flag |
| [30] | EBAlmostFull_Flag |
| [29] | EBEmpty_Flag |
| [28] | EBMetadataFull_Flag |
| [20] | EBMetadataFull_Flag |
| [19-12] | Fifo metadata used words |
| [11] | EBFull_Flag |
| [10-0] | Fifo data used words |

- **LocalTX_Fifo_Reg (offset 8):**

it contains useful flags concerning the status of the two FIFOs inside *Local_TX*.

| Bit | Description |
|---------|---------------------------|
| [31] | TXFull_Flag |
| [30] | TXAlmostFull_Flag |
| [29] | TXEmpty_Flag |
| [28] | TXRegFifoFull_Flag |
| [27] | TXRegFifoAlmostFull_Flag |
| [26] | TXRegFifoEmpty_Flag |
| [25] | TXRegFifoFull_Flag |
| [24-16] | Registers Fifo used words |
| [11] | TXFull_Flag |
| [10-0] | Fifo data used words |

- **LocalRX_Fifos_Reg (offset 9):**

it contains useful flags concerning the status of the two FIFOs inside *Local_RX*.

| Bit | Description |
|-------|-----------------------------------|
| [31] | RXFull_Flag |
| [30] | RXAlmostFull_Flag |
| [29] | RXEmpty_Flag |
| [28] | RX_outFifo_Full_Flag |
| [27] | RX_outFifo_Empty_Flag |
| [9] | RXFull_Flag |
| [8-0] | Received commands Fifo used words |

- **ADC registers (offsets 10 to 13):**

these registers contain the raw values of the 8 analog channels of the ADC.

| Offset | Bits [23-12] [11-0] |
|--------|-----------------------|
| [10] | Ch 1 Ch 0 |
| [11] | Ch 3 Ch 2 |
| [12] | Ch 5 Ch 4 |
| [13] | Ch 7 Ch 6 |

- **FPGA_p_RAM (offset 14):**

register which contains the pointer of the FPGA and its wrapped_around flag.

| Bit | Description |
|--------|------------------------|
| [31-0] | fpga_side_RAM_ctrl_reg |

- **HPS_p_RAM (offset 15):**

register which contains the pointer of the HPS and its wrapped_around flag.

| Bit | Description |
|--------|-----------------------|
| [31-0] | hps_side_RAM_ctrl_reg |

Bibliography

- [1] Robbins, *Basic Pathology*, 8th edition, Saunders/Elsevier, 2007, cap. 6.
- [2] Elizabeth M. Craik, *The ‘Hippocratic’ Corpus: Content and Context*, 1^a edizione, Routledge, 2014.
- [3] PEIR Digital Library (Pathology image database)
- [4] cancer.org, <https://www.cancer.org/cancer/cancer-basics/history-of-cancer/what-is-cancer.html>.
- [5] E.J. Odes et al., *Earliest hominin cancer: 1.7-million-year-old osteosarcoma from Swartkrans Cave, South Africa*, South african Journal of Science, 2016, <http://dx.doi.org/10.17159/sajs.2016/20150471>.
- [6] *Enciclopedia della scienza e della tecnica*, Istituto dell’Enciclopedia Italiana, 2007-2008, [http://www.treccani.it/enciclopedia/cellula-neoplastica_\(Enciclopedia-della-Scienza-e-della-Tecnica\)](http://www.treccani.it/enciclopedia/cellula-neoplastica_(Enciclopedia-della-Scienza-e-della-Tecnica)).
- [7] NIH U.S. National Library of Medicine, <https://www.nlm.nih.gov/>
- [8] Anand P, Kunnumakkara AB, Kunnumakara AB, Sundaram C, Harikumar KB, Tharakan ST, Lai OS, Sung B, Aggarwal BB, *Cancer is a preventable disease that requires major lifestyle changes*, in Pharm. Res., vol. 25, n^o 9, settembre 2008, pp. 2097–116, DOI:10.1007/s11095-008-9661-9, PMC 2515569, PMID 18626751.
- [9] *Latest global cancer data: Cancer burden rises to 18.1 million new cases and 9.6 million cancer deaths in 2018*, www.iarc.fr, 5 December 2018.

- [10] William Coleman, Gregory Tsongalis, *Molecular Pathology: The Molecular Basis of Human Disease*, Amsterdam, Elsevier Academic Press, 2009, p. 66, ISBN 0-12-374419-9.
- [11] Associazione Italiana per la Ricerca sul Cancro, <http://www.airc.it/cancro/terapia-tumori/asportazione-tumore/>
- [12] Associazione Italiana per la Ricerca sul Cancro, <http://www.airc.it/cura-del-tumore/chemioterapia.asp>
- [13] National Cancer Institute at the National Institutes of Health, <https://www.cancer.gov/about-cancer/treatment/types/immunotherapy#how-does-immunotherapy-work-against-cancer>
- [14] I. Mattei, *A novel on line dose monitoring technique tailored for particle therapy applications*, PhD thesis, University of Roma 3, 2016.
- [15] M. Senzacqua, *Progettazione di un sistema di schermatura per un acceleratore lineare a protoni per adroterapia*, Tesi, University of Roma La Sapienza, 2013.
- [16] A. Chendi, *Commissioning di un dispositivo real-time per dosimetria in vivo*, University of Bologna, Tesi Magistrale, 2017.
- [17] Particle Therapy Co-Operative Group, <https://www.ptcog.ch/index.php/>
- [18] G. Battistoni, *Nuclear Physics and Hadrontherapy*, slides, <https://agenda.infn.it/conferenceDisplay.py?confId=10727>, 2016.
- [19] A. Zoccoli, *Hadron therapy for cancer treatment*, slides, <https://agenda.infn.it/conferenceDisplay.py?confId=10231>, 2015.
- [20] University of Virginia, <https://www.med-ed.virginia.edu/courses/rad/radbiol/02bio/bio-02-06.html>
- [21] M. Jermann, *International Journal of Particle Therapy: Summer 2015*, 2015.
- [22] J. Dudouet et al., *Physical Review C* **88** no. 2, 2013.

- [23] M. Toppi et al., *Physical Review C* **93**, 2016.
- [24] W.R. Webber et al., *Physical Review C* **41** no. 2, 1990.
- [25] A. Ferrari et al., CERN 2005-10, INFN/TC_05/11, SLAC-R-773L (2005);
T.T. Böhlen et al., *Nuclear Data Sheets* **120**, 2014.
- [26] G. Battistoni et al., *The FOOT (Fragmentation Of Target) Experiment*, 55th International Winter Meeting on Nuclear Physics, Bormio, Italy, 2017.
- [27] Institut Pluridisciplinaire Hubert CURIE (IPHC), <http://www.iphc.cnrs.fr/>
- [28] A. Alexandrov et al., *FOOT Conceptual Design Report*, 2017.
- [29] J. W. Norbury et al., *Radiation Measurements* **12**, 315-363, 2012.
- [30] Hamamatsu Photonics K.K. web site,
<https://www.hamamatsu.com/eu/en/index.html>
- [31] Zhang, Fei, et al., *A prototype silicon detector system for space cosmic-ray charge measurement*, *Chinese Physics C*, Vol. 38, No. 6, 066101 (2014)
[doi:10.1088/1674-1137/38/6/066101]
- [32] Integrated Detector Electronics AS – IDEAS web site, <https://ideas.no/>
- [33] AD7476/AD7477/AD7478 datasheet, <https://www.analog.com/>
- [34] L. Servoli, *Status of the MSD subsystem*, slides,
https://agenda.infn.it/event/20171/contributions/106234/attachments/69198/85892/Roma_Meeting_2019_12_04-06.pdf, 2019.
- [35] Terasic web site, <https://www.terasic.com.tw/en/>
- [36] *DE10-Nano Product Brief*, <https://www.terasic.com.tw/en/>, 2017.
- [37] DE10-Nano User Manual, 2019.
- [38] *Cyclone V Hard Processor System Technical Reference Manual cv_5v4*, 2018.
- [39] Platform Designer (formerly Qsys) feature page, <https://www.intel.com/>

-
- [40] *Avalon® Interface Specifications MNL-AVABUSREF*, Intel, 2020.
 - [41] Enrico Vezzali, *Sistema di DAQ per studi sulla terapia adronica: progetto in VHDL e implementazione su FPGA*, University of Bologna, Tesi, 2017.
 - [42] Thomas Petazzoni, *Device Tree for Dummies*, conference slides, https://elinux.org/ELC_2014_Presentations, 2014.
 - [43] Kernel documentation - Reserved memory regions -, <https://www.kernel.org/>