Hydrogen Pellet Detection using Single-line CCD Cameras

Madhu Sudhana Rao Thelajala
Abstract

Hydrogen Pellet Detection Using single-line CCD Cameras

Madhu Sudhana Rao Thelajala

The PANDA Experiment will be one of the future experiments at the Facility for Antiproton and ion Research (FAIR), which has been officially founded in October 2010. The facility will be built on the area of the GSI Helmholtzzentrum fur Schwerionenforschung GmbH (GSI) in Darmstadt, Germany [1]. In the experiment, a stream of frozen hydrogen spheres (pellets) will traverse vertically an anti-proton beam circulating in the accelerator. Collisions of projectile anti-protons with the proton nuclei’s of the pellets will lead to the generation of new particles, which in turn will be measured in the PANDA detector facility. By using fast single-line CCD cameras, it is possible to capture the pellet course in all three spatial dimensions and calibrate the pellet interactions.

The aim of this thesis project is to describe the FPGA firmware for reading frames delivered by line-scan CCD cameras, detect and parameterize light reflections from pellets and deliver the results in the form of amplitude, position and a time stamp to the external readout system via an optical link. Additionally, the firmware supports slow control functions of the system as well as camera configuration.
Acknowledgement:

This thesis is carried out by me (Madhu Thelajala) and Geng Xiaoxiu. The following tasks have been performed by me.

1. Deserializer clock
2. Slow control transmitter
3. TXDATA format

The following parts have been carried out by Geng.

1. Deserializer data
2. Slow control receiver
3. RXDATA format

The following part is worked together

1. GTP transceiver

The previous work of “Pellet Tracking System” is performed by Malte Albrecht [16]. It provides a base platform for us to carry out our thesis work.

Foremost, I would like to express my sincere gratitude to my Supervisor Dr. Pawel Marciniewski for the continuous support of my thesis work, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me to perform my thesis work and in writing this thesis report.

My sincere thanks to Dr. Kjell Fransson for clarifying the doubts during my thesis work.

I am greatly indebted to Dr. Leif Gustafsson for having given me this splendid opportunity to be part of an active project of high research interest. His rigorous review comments helped me to the maximal extent in maintaining quality throughout my report. I would also like to take this opportunity to acknowledge him for the knowledge that he imparted through “VHDL course” during my master’s Degree.

I would like to thank my colleague Geng for sharing his knowledge and contributions.

I thank my friends and my family for their support and encouragement.
# Table of Contents

Table of Figures ..................................................................................................................................................... v

1 INTRODUCTION ............................................................................................................................................... 1

1.1 Overview of PANDA DETECTOR EXPERIMENT ........................................................................ 2

1.2 Pellet tracking – camera setup in UPTS........................................................................................ 3

1.3 Uppsala pellet target station (UPTS)............................................................................................. 4

1.4 Hardware setup and Motivation ..................................................................................................... 6

2 A single line CCD camera............................................................................................................................ 8

2.1 The camera Link .................................................................................................................................... 9

2.2 The Spartan 6 FPGA board description ..................................................................................... 11

2.3 An Overview of the design firmware .......................................................................................... 14

3 VHDL Implementation ............................................................................................................................. 16

4 Clocking design........................................................................................................................................... 19

5 Camera Link data de-serializer ............................................................................................................. 22

5.1 Configuration of the de-serializer clock and data reception............................................. 22

5.2 Data reception using PLL................................................................................................................ 22

6 VHDL design implementation for Hydrogen pellet recognition.............................................. 26

6.1 Automatic configuration ................................................................................................................. 27

6.2 Pedestal correction............................................................................................................................ 32

6.3 Pellet recognition............................................................................................................................... 32

6.4 Timestamp............................................................................................................................................ 35

6.5 Temporary Registers........................................................................................................................ 35

7 TXDATA FORMAT ...................................................................................................................................... 36

8 GTP Transceiver ......................................................................................................................................... 39

8.1 GTP Transmitter ................................................................................................................................ 39
8.1.1 8B/10B encoding technology ................................................................................................ 39

8.1.2 K Characters ....................................................................................................................... 40

8.2 GTP Receiver .......................................................................................................................... 41

8.2.1 RxPolarity ............................................................................................................................ 41

8.2.2 RX word alignment ............................................................................................................. 41

8.2.3 RX Loss-of-Sync State Machine ...................................................................................... 42

9 RX DATA FORMAT .................................................................................................................. 46

10 Slow Control ............................................................................................................................ 48

10.1 Slow control transmitter ...................................................................................................... 50

10.2 Slow control Receiver ......................................................................................................... 52

11 Performance ............................................................................................................................ 57

11.1 Test runs using "Fish line" (Optical fiber) setup .................................................................. 57

12 Conclusion and Future work .................................................................................................. 61

References ......................................................................................................................................... 62

Appendix A ......................................................................................................................................... 63
# Table of Figures

Figure 1: Setup of the PANDA experiment. The antiproton beam is entered from the left side .......... 2
Figure 2: Test setup for pellet tracking. The arrow points to a window through which the skimmer can be seen ................................................................................................................................................................................. 4
Figure 3: The UPTS setup schematic and CCD-cameras are located in red ........................................... 5
Figure 4: The data flow scheme of the readout system ............................................................................. 7
Figure 5: Camera schematic diagram ........................................................................................................... 8
Figure 6: integration time setting of free running mode ........................................................................... 9
Figure 7: The Camera Link interface .......................................................................................................... 10
Figure 8: Spartan 6 FPGA board ................................................................................................................ 11
Figure 9: Firmware design overview ......................................................................................................... 14
Figure 10: ISE Design flow .......................................................................................................................... 16
Figure 11: The ISE Navigator ........................................................................................................................ 18
Figure 12: Schematic design of the design clocks ..................................................................................... 19
Figure 13: Data Stream Using Clock with a 7:1 SerDes Ratio ...................................................................... 22
Figure 14: A general deserializer idea ........................................................................................................ 23
Figure 15: Schematic for deserializer ........................................................................................................ 24
Figure 16: Deserializer parallel data .......................................................................................................... 24
Figure 17: Schematic for Deserializer FIFO ............................................................................................. 25
Figure 18: Firmware design overview of the Hydrogen Pellet detection .................................................... 26
Figure 19: Raw data of ODD pixels from the camera and its MEM_DATA ................................................. 27
Figure 20: FSM for Conf0 ........................................................................................................................ 28
Figure 21: Conf0 mean values ................................................................................................................... 29
Figure 22: FSM for Conf1 .......................................................................................................................... 30
Figure 23: Structure of summation block memory .................................................................................... 31
Figure 24: simulation result for conf1 ..................................................................................................... 32
Figure 25: Raw data (red+blue) and pedestal corrected data (light red and light blue) in ChipScope ................................................................................................................................................................................. 33
Figure 26: Pellet Recognition FSM ......................................................................................................... 34
Figure 27: Chipscope result of pellet recognition ..................................................................................... 35
Figure 28: TXDATA_FIFO block diagram ............................................................................................... 36
Figure 29: FSM for TX data format .......................................................................................................... 38
Figure 30: GTP Transmitter Block Diagram ............................................................................................. 39
Figure 31: GTP Receiver Block Diagram ................................................................. 41
Figure 32: LOS state machine ............................................................................. 43
Figure 33: GTP block diagram ............................................................................ 43
Figure 34: FSM for GTP_Transmitter_TXDATA .................................................. 44
Figure 35: RXDATA_FIFO block diagram ............................................................ 46
Figure 36: FSM to store camera commands in the FIFO ...................................... 47
Figure 37: Slow control schematic ...................................................................... 48
Figure 38: Slow control transmitter schematic ................................................... 50
Figure 39: FSM of UART transmitter ................................................................. 51
Figure 40: slow transmitter data from Spartan 6 FPGA board ............................. 51
Figure 41: schematic for UART receiver ............................................................. 53
Figure 42: FSM for slow control UART receiver ................................................ 53
Figure 43: Slow control receiver data ............................................................... 54
Figure 44: FSM to store the camera response into the UART_RX_FIFO .............. 54
Figure 45: FSM for 50MHz ready signal ............................................................ 56
Figure 46: Test setup using an illuminated plastic .............................................. 57
Figure 47: GTP transmitter contains pellet data ............................................... 58
Figure 48: GTP transmitter contains pellet data and slow control data ............... 58
Figure 49: GTP transmitter contains slow control data ...................................... 58
1 INTRODUCTION

In accelerator-based physics, experiments are roughly divided into collider experiments and fixed target experiments. In collider-based experiments, two beams of high energy particles collide head on. The advantage of collider experiments is that two beams have significantly higher (useful) energy and higher mass particles can be produced.

In a fixed-target experiment, a charged particle such as an electron or a proton is accelerated by an electric field and collides with a target, which can be a solid, liquid, or gas. A detector measures the charge, momentum, mass, etc. of the resulting particles. An example of this process is the PANDA experiment (1), in which antiprotons are colliding with a perpendicular stream of hydrogen spheres. Developing detectors for these fixed-target experiments is a challenging task.

There are different ideas developed for fixed-target experiments. One idea is to use a cluster jet-target and another is to use a pellet-target. In the cluster-jet target, the produced clusters contain approximately $10^5$ atoms. The clusters size and shape are randomly distributed within a certain range, but the target beam has the same size and shape with equal volume density distribution and a sharp boundary. In the pellet target, the produced cluster also contains approximately $10^5$ atoms, but the pellet size and shape are evenly distributed in some certain range. A great benefit of the pellet target is that the distance between the pellet generator and the interaction point can be in the order of meters, which makes it possible to have the cryosystem needed for pellet formation placed outside of detector parts and feed the pellets into it through a narrow tube. For a cluster jet target the nozzle which creates the cluster stream has to be positioned closer to the interaction point in order to reach the same target thickness. Both systems can run with other gases like deuterium or in principle even nitrogen or argon.
So far, the idea using a pellet-target has been developed for the CELSIUS/WASA experiment in Uppsala in 1995 (2) and currently in operation in the WASA@COSY experiment in Forschungszentrum Jülich (FZJ) in Germany. The same idea will also be used in the PANDA experiment, Darmstadt, Germany (1). Right now, the Uppsala group is doing development for the PANDA experiment, for this purpose a copy of the WASA pellet target was built at The Svedberg Laboratory, Uppsala. This target is a stand alone system. It is called the Uppsala Pellet Test Station (UPTS) and is used for target development and studies, like the pellet tracking system presented in this work.

1.1 Overview of PANDA DETECTOR EXPERIMENT

![Figure 1: Setup of the PANDA experiment. The antiproton beam is entered from the left side.](image)

The name PANDA stands for anti-Proton ANnihilations at DArmstadt. The PANDA Experiment is a facility for Antiproton and Ion Research (FAIR) founded in October 2010 (1). It is built on the area of the GSI Helmholtzzentrum für Schwerionenforschung GmbH (GSI) in Darmstadt, Germany. Further large experiments are planned at this facility, which is dedicated to do research on the fundamental structure of matter. A part of this facility will be the antiproton storage ring HESR which will deliver an antiproton beam with a momentum range of 1.5 GeV/c - 15 GeV/c. The storage antiproton in HESR collide with a fixed target (i.e hydrogen pellets) inside the PANDA detector. Investigations related to the strong force at medium energy are the primary focus of this experiment.
1.2 Pellet tracking – camera setup in UPTS

The physics program for the PANDA experiment includes charmonium spectroscopy, especially because the center of mass energy can reach up to 5.5 GeV, which is well above the open charm threshold (DD-breakup threshold: 3.73 GeV/c²). Normally, D-mesons will have a very short lifetime which is approximately equal to 1.040 ps and decay length of around 0.3 mm. In the PANDA experiment, with the help of a Micro-Vertex-Detector, it is possible to detect the displaced decay vertex of the D-meson and it is also possible to identify the D-meson. It would be better to reconstruct the decay length and therewith to identify a D-meson. But for this, a precise knowledge is needed about the position of the primary interaction vertex. However in many cases the reconstruction of this point in space using the information the detector delivers about the reaction products is not usable, because the reaction products either decay too fast to be detected or electrically neutral and thus cannot be tracked by the innermost detectors.

A method to determine the primary vertex without using the reaction products and the detector is to track the path of single hydrogen pellets. As explained the stream of hydrogen pellets traverse perpendicular to the anti-proton beam in the PANDA detector experiment. It is possible to trace the individual pellets because pellets are discrete objects and have a distance in the order of millimeters to one another. To determine the primary vertex, one has to determine the position of each pellet in the X-Z plane parallel to the accelerator beam (Z-axis) before and after the interaction point. To determine the position of these pellets, CCD cameras are used in the design. By illuminating the pellet stream with a laser light, pellets are made visible in the CCD cameras. By placing two of these cameras with a relative angle of 90° to each other, the x- and z-position can be determined. A test system with four cameras in total is being developed at the UPTS. In the figure 2, one can see that, there are two places to place the cameras. Wherever a camera is placed, a laser light is placed on the opposite side, so that the camera can see the hydrogen pellets. To these cameras, an FPGA board is interfaced to process the pellet data. The hardware setup is explained in the next section.
1.3 **Uppsala pellet target station (UPTS)**

In this experiment, all the tests and measurement are performed at the Uppsala pellet target station. The UPTS is an independent pellet target/generator built for the panda detector experiment. The UPTS is almost similar to the WASA with minor changes. We can say that; the UPTS is almost a copy of WASA. The picture of UPTS and its schematic is shown in figure 3.
In the schematic at the top a cold head is placed. In the cold head the hydrogen is in gas form. To liquefy the hydrogen from the gas, the system uses helium. The liquid hydrogen then passes down to a narrow opening of a glass nozzle. A cylindrical piezo-electric crystal is placed around the glass nozzle. A frequency generator is connected to this crystal, which is used to break the liquid hydrogen into droplets when leaving the nozzle. The broken hydrogen pellets now pass to a droplet information chamber. From the droplet information chamber the droplets pass to a glass capillary. Here the system creates a vacuum, where the droplets travel from the chamber to the glass capillary. In this process the droplets freeze to solid, because the pressure of the vacuum is below 1 microbar. In the vacuum process the path of the pellets are distorted. The stream of pellets which is coming from the glass capillary has large angular spread, which leads to a widening of the pellet beam. The system has a skimmer to collimate the beam about 1.5m below the glass capillary. This skimmer is made of a metallic cone with a circular opening of 2mm at the top. 20% to 70% of the pellets pass this skimmer and the remaining pellets bounce from the cone and stay
back in the vacuum chamber. Below the skimmer, the particle (Hydrogen pellets and antiprotons) interaction takes place. After that the hydrogen pellet travels further down and evaporates. In the system, the CCD cameras are placed at four places. The camera data is read out to a Spartan 6 FPGA. The hardware setup in this work will be explained further below.

1.4 Hardware setup and Motivation

Normally frame grabbers are used to capture the individual, digital still frames from an analog video signal or a digital video stream. In typical applications, the cameras are interfaced to frame grabbers located in a PC. These frame grabbers are recording the full frames to be displayed, stored or transmitted in raw or compressed digital form. In this project, there is no need to record full frames. Instead an algorithm for detecting and parameterizing is necessary to compress the dataflow. For the efficient accomplishment of this task for many cameras, a Spartan FPGA 6 board was constructed.

The advantage of using the Spartan 6 FPGA board is that it can simultaneously read 4 CCD cameras. The Spartan 6 FPGA is programmed to detect pellets and parameterize the pellet events according to an algorithm designed using a VHDL (Hardware Description Language) (3). The processed data on the Spartan 6 FPGA can be transmitted to a VIRTEX 5 FPGA board by an optical fiber for further processing. The optical-fiber link is connected to the GTP (Gigabit Transceiver Protocol) transceiver of the two FPGA boards. The VIRTEX 5 board has totally 16 optical link ports and it supports reading multiple processed camera data from several Spartan 6 FPGA boards. The VIRTEX 5 board is equipped with a VME (Versa Module Europa) bus to transfer the pellet data to a computer. The data flow scheme of the readout system is shown in figure 4.
Figure 4: The data flow scheme of the readout system
2 A single line CCD camera

In this design, an AVIIVA M2 CL single line CCD camera is used. These cameras are easy to interface and they provide accurate data. The advantage of this camera is, it provides anti blooming and it can operate in 0 to 60° Celsius (4). The resolution of the camera is 512 pixels divided into two, 256 are odd pixels and 256 even pixels. The light intensity resolution of each pixel data is 12bit.

Figure 5: Camera schematic diagram

There are two linear CCD taps and one chain to process 256 odd pixels and the other the 256 even pixels. During readout of a line image, the collected charge collected in analog sample and hold pixel memories in digitized pixel by pixel, one for even and the other for odd pixels, as shown in figure 5. The camera utilizes correlated double sampling, dark level correction, gain and offset correction and 12 bit analog to digital conversion. During each line scan the CCD is exposed over a period of minimum 5μs followed by a dead time of at least 1μs according to the specification when the charge is transferred from the CCD.

The camera can be configured in a Free Run Mode or External Trigger Mode. In Free Run mode the camera run independently and the output date rate can be 20, 30, 40 or 60 MHz (4). If the camera is running in External Trigger mode the integration and read out are controlled by signals TRIG1 and TRIG2. In this design, we used the free running mode at a rate of 30MHz. In the free running mode the integration time and readout periods will start automatically and immediately after previous period. The integration time setting in the free running mode is explained in the table 1 and Figure 6.
<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ti</td>
<td>Integration time duration</td>
<td>1</td>
<td>--</td>
<td>13ms</td>
</tr>
<tr>
<td>Tg</td>
<td>Consecutive integration period gap(at maximum frequency)</td>
<td>--</td>
<td>6microsec</td>
<td>--</td>
</tr>
<tr>
<td>Tt</td>
<td>Integration period stop to read out start delay</td>
<td>--</td>
<td>1microsec</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 1

**Timing diagram:**

![Timing diagram](image)

**Figure 6: integration time setting of free running mode**

In order to build a multiple camera system to monitor some big and complicated events, we need to name one camera as master camera and the rest are slave cameras. The master camera provides output clock and line valid data to all the slave cameras. All cameras share one clock and trigger signals to ensure synchronization.

### 2.1 The camera Link

The Camera Link is a standard designed for point to point computer vision applications. The Camera Link is an extention of the original technology from National Semiconductor. The purpose of designing this protocol is to standardize scientific and industrial video products including cameras, cables and frame grabbers. The Camera Link configuration can be selected depending on the amount of data transmitted. In the project the cameras use a so called "base" configuration of the Camera Link Standard. The camera can be read out and controlled via the Camera Link interface. The Camera Link interface has four serial data lines, a clock running at 60MHz and an asynchronous communication channel for camera configuration and control. The output data format of four data lines is depicted in figure 7.
Figure 7: The Camera Link interface

For configuration and control are later called a “Slow Control”. The serial communication based on the RS-232 protocol is used (6). The slow control is full duplex and without handshaking. The baud rate is 9600 baud and the format data is 8 bit data with one stop bit and no parity bit. This protocol is explained in the Slow Control Chapter 10.

<table>
<thead>
<tr>
<th>BIT</th>
<th>PIN NAME</th>
<th>BIT</th>
<th>PIN NAME</th>
<th>BIT</th>
<th>PIN NAME</th>
<th>BIT</th>
<th>PIN NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODD-00</td>
<td>TX0</td>
<td>ODD-07</td>
<td>TX5</td>
<td>EVEN-02</td>
<td>TX19</td>
<td>EVEN-09</td>
<td>TX14</td>
</tr>
<tr>
<td>ODD-01</td>
<td>TX1</td>
<td>ODD-08</td>
<td>TX7</td>
<td>EVEN-03</td>
<td>TX20</td>
<td>EVEN-10</td>
<td>TX10</td>
</tr>
<tr>
<td>ODD-02</td>
<td>TX2</td>
<td>ODD-09</td>
<td>TX8</td>
<td>EVEN-04</td>
<td>TX21</td>
<td>EVEN-11</td>
<td>TX11</td>
</tr>
<tr>
<td>ODD-03</td>
<td>TX3</td>
<td>ODD-10</td>
<td>TX9</td>
<td>EVEN-05</td>
<td>TX22</td>
<td>STROBE</td>
<td>TXCLK</td>
</tr>
<tr>
<td>ODD-04</td>
<td>TX4</td>
<td>ODD-11</td>
<td>TX12</td>
<td>EVEN-06</td>
<td>TX16</td>
<td>LVAL</td>
<td>TX24</td>
</tr>
<tr>
<td>ODD-05</td>
<td>TX6</td>
<td>EVEN-00</td>
<td>TX15</td>
<td>EVEN-07</td>
<td>TX17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODD-06</td>
<td>TX27</td>
<td>EVEN-01</td>
<td>TX18</td>
<td>EVEN-08</td>
<td>TX13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: The bit assignment of the 28 bits

During one clock pulse, the camera sends 4×7 bits of serialized data. In this design, the cameras are connected to the cameralink connector of the Spartan 6 FPGA Board. Upon receiving, this data will be deserialized for further processing. A detailed description of deserialization is given in next chapter. The bit assignment of the 28 bits is shown in the table 2.
2.2 The Spartan 6 FPGA board description

The entire project firmware is developed for the Spartan 6 FPGA board. A block diagram of the Spartan 6 FPGA board is shown in Figure 8.

The main components of the Spartan 6 FPGA board are:

1. XC6SLX100T FPGA
2. Cam link connectors- 4 pieces
3. SFP cage
4. USB to RS232 converter

Figure 8: Spartan 6 FPGA board
1. XC6SLX100T FPGA:

The board is equipped with a XC6SLX100T FPGA device. It is a Xilinx Spartan 6 from the LX family. The device features:

- 101,261 logic cells
- 15,822 slices (each Spartan 6 FPGA contains 4 LUTs and 8 Flip-Flops)
- 126,576 Flip-Flops
- Configured for up to 976kb of distributed RAM
- 180 DSP48A1 blocks and 268 of 16kb BlockRAMs
- 6 clock management Tiles (CMT)
- Four Memory Controller Blocks (MCB) and up to 498 user I/ Os

More details on the Spartan 6 FPGA can be found in the product specification (7).

2. Camera Link connectors:

The board has four 26 pin cam link connectors of the type MDR (Mini D Ribbon) for interfacing to cameras.

3. SFP cage:

The SFP stands for Small Form Factor pluggable cage. The board has one SFP cage and each cage contains a small form factor pluggable transceiver. The transceiver is interfaced to an optical link for full duplex data communication supports data rate up to 3.125 GB/s.

4. USB to RS232 converter:

The CP2102 is a USB-to-UART Bridge Controller. It is providing a simple solution for updating RS-232 designs to USB using a minimum of components and PCB space. The device includes a USB 2.0 full-speed function controller, USB transceiver, oscillator, EEPROM, and asynchronous serial data bus (UART) with a complete modem control signals. The EEPROM is used to customize the USB vendor ID, product ID, Product Description String, Power Descriptor, Device Release Number, and Device Serial Number as desired for OEM applications. The EEPROM is
programmed on-board via the USB, allowing the programming step to be easily integrated into the product manufacturing and testing procedure.

5. Configuration Memory:

The board has an In-system programmable PROM for configuration of an FPGA. An XCf32PFS48C PROM [11] device is used on the Spartan 6 FPGA board with a size of 32MB. The device is programmed via JTAG.

6. JTAG

The JTAG stands for Joint Test Action Group (8). The JTAG is used to program the In-system programmable PROM device in the design. JTAG interface is providing a convenient and fast way to test the FPGA project.

7. NIM input/output:

The board has four NIM standard input/output pins. It is used to check the behavior of the signals by the user at runtime.

8. Crystal oscillator:

There is only one internal clock on the Spartan 6 FPGA board, in the form of a crystal oscillator with a frequency of 100MHz. Other FPGA clocks can be derived internally from DCM and PLL components.

9. Power supply:

The board is powered by a 5V power supply. The board is using the converters to generate the required voltages (3.3V, 2.5V and 1.2V) for the FPGA and other components.
2.3 An Overview of the design firmware

The firmware design is split up into several parts, which is implemented in separate files. The overview of the firmware design can be seen in figure 9.
When powering up the system the Deserializer block is performed. In the Deserializer block, the data coming from the cameras are deserialised and formatted. The formatted data are stored in a Deserializer_FIFO. After the deserializer design operation, a two stage automatic pedestal extraction is performed. Every pixel of the camera shows a pedestal amplitude depending on the ambient light. One has to correct this effect in order to detect real pellet events efficiently. The two stages conf0 and conf1 of the automatic pedestal correction generate pedestal values for each single pixel and store these values in a block memory. Once the automatic pedestal correction process has been completed, a flag is set to enable the pellet recognition part. This block also contains a timestamp generator for each valid camera line. The recognized pellet events are stored in the corresponding FIFOs. A clocking is implemented for the above part and the slow control part. After this, a GTP (Multi-gigabit Transceiver Protocol) part transmits the pellet events serially using 8b/10b coding technology to the VIRTEX 5 FPGA board. Additionally, a slow control part is implemented for four cameras to configure the cameras and to configure readout. A configuration readout data is stored in the corresponding FIFO. The configuration readout data is also sent to the VIRTEX 5 FPGA by using the GTP part.

All the code components explained above are instantiated and connected to each other in the top level entity file called **Cam_link_int4.vhd**. The top level entity also contains an instantiation of a Xilinx ChipScope instance. The chipscope components are generated using the Xilinx LogiCore generator.
3 VHDL Implementation

The firmware for the Spartan 6 FPGA was written in VHDL. VHDL stands for VHSIC hardware description language, which is used to describe the digital and mixed signal systems like integrated circuits and FPGAs (3). The VHDL language looks very similar to other conventional programming languages such as BASIC, C and assembly code. VHDL is inherently a parallel programming language, where all instructions run concurrently. The advantage of VHDL is that the compiled code will process all the signals in parallel. It is also possible to simulate the source code with the help of a test bench and simulation software, before it is downloaded to an FPGA.

The Xilinx ISE tool is used for the project to develop the source code in VHDL. The Xilinx ISE system is an integrated Software Environment produced by the Xilinx Company (9). It consists of a set of programs to create, simulate and implement the digital designs in an FPGA or CPLD target devices. The tool has a graphical user interface (GUI) that allows the programs to execute from the toolbars, menus or icons. The ISE design flow is shown below.

![Figure 10: ISE Design flow](image-url)
The design flow is explained in the following section.

**Create Design:**

The first step is to create an ISE (Integrated Software Environment) design project and add or create source files for this project. A source file can be created in different formats such as HDL, schematic, EDIF/NGC, intellectual property (IP), embedded processor and digital signal processing. An ISE project design will consist of a top-level source file and various lower-level files.

**Synthesis Design:**

This step creates a net list from the source files. The net list can serve as input to the implementation module.

**Simulate Design:**

This is an important step that should be done at various points during the design flow. The design can be verified by using a simulator, which is provided by the ISE design suite. The simulator verifies the functionality of the design, the behavior and the timing of the circuit.

**Constraints Entry:**

One can specify the timing, placement and other design requirements by using design constraints. The ISE design suite has constraint editors to facilitate constraints entry for timing constraints as well as I/O pin and layout constraints.

**Implement design:**

After the synthesis step, the implementation converts the logic design to a physical design file that can be downloaded to the target devices. The step is involved in three sub steps: Translating the Netlist, Mapping and Place and Route.

**Implementation Analysis:**

After the implementation, one can analyze the design performance against the constraints, device resource utilization, timing performance, and power usage. We can also analyze the timing and power results by using the Time and Xpower Analyzer tools.

**Implementation Improvement:**

One can make changes in the design sources, process properties based on the analysis of the design results and rerun the synthesis and implementation to achieve the design requirements.
Device Configuration and Programming:

After generating the bit file, the file can be downloaded from the host computer to the target FPGA.

The above steps are handled through a central ISE project Navigator window shown in figure 11.

Figure 11: The ISE Navigator

Chipscope Pro Analyzer:

Chipscope (10) is a tool, which allows the user to put one or more integrated logic analyzers into an FPGA. Chipscope is used to observe input and output signals of the design and can also observe the internal signals used in, for example, a state machine in a running system. The benefit of chipscope is a possibility to debug the design while it is running on the hardware. Chipscope is widely used for debugging the code and to check the behavior of components after implementations. In this project all results are taken from the chipscope pro analyzer.

ISim:

ISim stands for the Xilinx ISE simulator (11). It is a Hardware Description Language simulator that performs both behavior and timing simulations in VHDL or Verilog language designs.
4 Clocking design

A proper clocking is an important issue for the entire design. Four cameras are in use in the development system; one has to take care of the synchronization between different pieces of the hardware (e.g. More cameras). An external clock is required for synchronization between different parts of the hardware. To drive several VHDL based designs in the project, an external 100 MHz clock on the Spartan 6 FPGA board is used. But in the project, different VHDL designs are running at different frequencies. For example, a 30MHz clock is used for the deserializer design and pellet detection design. For slow control, a 9600 baud clock and a sampling clock (16 times faster than the baud clock) is used. A LogiCORE IP Clocking Wizard (13) provided by Xilinx library is used to generate the required frequencies. The schematic diagram of the design clocks is shown in figure 12.

![Figure 12: Schematic design of the design clocks](image)

It is easy to create HDL source wrappers for the clock circuits customized to our clocking requirements by using this LogiCORE IP Clocking Wizard. Here, the clock can be generated in two modes, one is an automatic mode and another is a manual mode. In automatic mode, the clock wizard selects appropriate clocking configures buffering, timing and feedback multiplier, divisor parameters for our desired clocking network. In the manual mode, all clock primitives can be configured manually. For this design, automatic mode is selected for generating the desired clocks. The automatic mode, just needs to assign the desired clocks while using the Clocking Wizard. The LogiCORE IP Clocking Wizard has generated three clocks named as 30MHz, 12MHz and 15.385MHz by using a system clock 100MHz. This LogiCORE IP Clocking Wizard uses a PLL to generate multiple clocks. The table for the clock divisor/multiplier is given below.
The general formula for PLL output clock frequency is

\[
\text{Frequency of } \text{CLKOUTn} = \text{Frequency of } \text{CLKIN} \times \frac{(\text{CLKFBOUT\_MULT} / \text{DIVCLK\_DIVIDE})}{\text{CLKOUTn\_DIVIDE}}
\]

It is not possible to generate the baud clock and sampling clock by using a PLL, because the PLL can only generate clocks above 3.2MHz. A simple VHDL code is written for generating the baud clock and sampling clock which is shown below. The P1 process used a 12MHz clock for generating a baud clock of 9600 baud, the clk_out_1 is changing its value 0 ↔1 or 1↔0 for every 624 clock cycles. The P2 process used a 15.385MHz clock for generating a sampling clock, the clk_out_2 is changing its value 0↔1 or 1↔0 for every 50 clock cycles.

```
P1:process(clk_int_1)
    variable divider:integer:=0;
    begin
        if (clk_int_1'event and clk_int_1='1') then
            if (divider/=624) then
                divider:=divider+1;
            else
                divider:=0;
                clk_out_1<=not clk_out_1;
            end if;
        end if;
    end process;
```
P2:process(clk_int_2)
    variable divider:integer:=0;
    begin
        if (clk_int_2'event and clk_int_2='1') then
            if (divider/=49) then
                divider:=divider+1;
            else
                divider:=0;
                clk_out_2<=not clk_out_2;
            end if;
        end if;
    end process;
5 Camera Link data de-serializer

In the design, the Spartan 6 FPGA board contains four cameralink connectors, which are used to capture the serialized data from the cameras. In order to deserialize the camera link data, the Spartan 6 FPGA utilizes SERDES IP Cores (provided by Xilinx) (14). The name SERDES stands for serialization and deserilization. Before using an IP Core directly in the design, the clock and data reception needs to be configured in order to comply with the design requirements.

5.1 Configuration of the de-serializer clock and data reception

The deserializer design depends on the incoming data stream format. Normally, the data stream rate is a multiple of the incoming clock. The clock signal is used for framing signals. A multiple change in the state of data lines can occur during one clock pulse. Widely used in cameras, TVs and monitors are the 1:7 interface (14). The data format of 1:7 is shown in figure 13.

![Figure 13: Data Stream Using Clock with a 7:1 SerDes Ratio](image)

In this configuration, the received clock is multiplied by the PLL giving a high speed sampling clock. In the design, the camera is giving a 30MHz clock to the deserializer design. The 30MHz input clock accompanying the 7:1 data requires a PLL to operate at 210MHz, by using a high speed capture clock to clock the received data to the input to the deserializer. The deserializer process the received data and produce parallel output data at the speed of the original clock.

5.2 Data reception using PLL

The phase of the sampling clock has to be accurately adjusted to the data bit phase. The adjustment is done by controlling a PLL (delay Locked Loop) containing a chain of electrically programmable delay components. The phase calibration is performed at the power up of the device. In the design, the receiver clock is multiplied by 7 in the PLL and generates a new internal single data capture clock. The block diagram of the data reception using PLL is shown in figure 14.
As described in 2.1, the Camera Link has four serial input data lines. The data reception takes place with the sampling clock (210MHz) with reference to the receiver clock. The four data reception blocks will produce an output as 4x7 parallel data. A Bitslip operation is used to align the parallel data in correct order with reference to the input frame clock. The detailed explanation of the deserializer can be found on Source-Synchronous Serialization and Deserialization (14) or Spartan-6 FPGA SelectIO Resources (15).

The output 28bits contain the 12bit amplitude for even pixels-12 bit and odd pixels-12 bit, one line valid signal and three control signals which are not used in this design.
The component declaration of the deserializer is shown below. Once a deserializer was designed for one camera, it was multiplied four times to serve for four cameras. A “FOR GENERATE” statement is used to generate the same component for four cameras.

The deserializer design parallel data are shown using chipscope pro logic analyzer in Figure 16. The red and blue data line indicates even and odd pixel’s parallel data from the camera. In the chipscope the parallel data are shown in the form of bus plot and waveform.

The deserialized data from the four cameras are source synchronous and may slightly differ in frequency and phase. In order to use a common clock domain for further processing, Deserializer_FIFO were implemented. The FIFO is used to eliminate the data skews and to provide one clock for all feature extractions. The schematic diagram of the Deserializer_FIFO is shown in Figure 17.
A camera clock is used to write the deserializer data in the FIFO. The write enable pin is kept asserted, so that the deserializer keeps on writing the parallel data into the Deserializer_FIFO. The next step is to find out at which pixel (position of the pixel) the hydrogen pellet passed the camera and also find out the amplitude of the shadow from the pellet. The VHDL Implementation for the pellet calibration will be explained in next chapter.
6 VHDL design implementation for Hydrogen pellet recognition

The firmware overview of the Hydrogen pellet recognition part is shown in Figure 18. This chapter will explain how a two stage automatic pedestal correction and pellet recognition is working (16).

![Figure 18: Firmware design overview of the Hydrogen Pellet detection](image)

As explained in 1.3, the laser lights are placed opposite to each camera. A camera can see a certain statistical noise and every pixel has its own gain, which also acts on an externally appeared offset. In complete darkness the average value of the pixel depends on the gain and offset, while the actual value additionally contains the noise. In order to efficiently detect a relatively small signal from pellet reflection one has to correct for the systematic offset. Before finding the real pellets a two stage (Conf0 and Conf1) automatic pedestal correction is performed. The Conf0 stage computes the mean values for all odd and all even pixels. Once the Conf0 stage has calculated the mean values of odd and even pixels for 16 valid exposures, a flag is set to enable the second stage (Conf1). The output signals from the first stage are named init_mean_even and init_mean_odd (also be called global mean values). The global mean values are used in the second stage to detect lines, which probably contains a pellet event from the camera exposure. The Conf1 stage calculates the mean values for odd and even pixel of 1024 valid exposures. In the Conf1 stage a block memory is used to store the summed values. The block memory data width is 44bit and the depth is 256 locations. Each location contains odd and even pixel summation values. Each time, a new value from the camera is summed with corresponding block memory cell and stored in the same location. Once the memory is filled with the sum of 1024 valid
exposures, a new process reads the data from the summation block memory, truncates 10 bits from the odd and even pixels and store new mean values in the external block memory (Pedestal BRAM). This process also finds the largest pedestal (MEAN_MAX) value among all the pixels. By doing the automatic pedestal correction, it’s possible to detect the real pellets by applying only one threshold value for all pixels. The raw data of odd pixels from the camera and its block memory mean values (MEM_DATA) are depicted in figure 19. After two stage of pedestal correction, a flag signal is set to start the pellet recognition part. In the pellet recognition part an error correction for each pixel coming from the camera is performed with the help of pedestal values and output real pellet events. A detailed explanation of each part is done below.

![Figure 19: Raw data of ODD pixels from the camera and its MEM_DATA](image)

6.1 Automatic configuration

The automatic pedestal correction fills a Pedestal BRAM with pedestal values for each pixel of the camera as it has been described above. Later, when the design is used while running a pellet target, there could be pellet events already when powering up the system. These events should not influence the pedestal values, but should be ignored until the pellet recognition starts operating. Thus, the first stage just calculates separate mean values for all odd and all even pixels. These two 'global' mean values are needed in the second stage in order to be able to reject events with pellets.

6.1.1 Stage Conf0

The purpose of Conf0 Finite State Machine (FSM) is to determine the mean values for the 16 lines exposure data from a camera. The finite state machine for conf0 is shown in figure 20.
At first, the state is at state “000” which also called “reset state”. In this state i.e. “000”, all flags and counters which are needed for conf0 are reset. The FSM starts operating when the deserializer submits the data to the deserializer FIFO. In the “000” state the FSM waits for the Line Valid signal (LVAL) to turn to ‘0’ to go to the next state. In the “001” state, the counters are initialized and waits for the line valid signal to turn to ‘1’ from the deserializer FIFO. Once the FSM sees a Line Valid signal asserted from the deserializer FIFO the state goes to the next state i.e. “010”. In the “010” state, the flag sum_en will be set to ‘1’ to start the SUM UP DATA process. The FSM reads the digitized content of two pixels from the deserializer FIFO in every clock cycle. So it needs 256 clock cycles to read one camera exposure data from the deserializer FIFO. The LVAL signal is asserted for 256 clock cycles. The SUM UP DATA process adds up the incoming data for the odd and even registers separately. The reason behind splitting the odd and even values is that a camera is using two different ADCs for odd and even pixels, which might show a slightly different behavior. Once the full line reads the data from the deserializer FIFO, the FSM jumps to the “011” state. In this state the line counter is being increased by one and the SUM UP DATA process
is stopped. For some reason, if the line valid signal is not asserted for 256 clock cycles the FSM jump from “011” state to the “001” state and resets the summation counter. This procedure is repeated until the line counter reaches 16. The line counter overflow checks in the “100” state, if a line counter overflow happens the FSM jump to the “101” state. Now two summation registers contain the sum of 16×256=4096 values each 12bit wide. After accumulating the 16 lines, global mean values are calculated by using a simple bit shift and the output of mean values (init_mean_even and init_mean_odd) are filled with the global values. A FOR GENERATE statement is used to implement the same design for the remaining cameras. The chipscope result for conf0 is shown in figure 21. The red and blue data lines of RX are the incoming data from the deserializer FIFO; the two other data lines init_mean_even and init_mean_odd are the output of conf0.

![Figure 21: Conf0 mean values](image)

6.1.2 Stage Conf1

The purpose of the conf1 finite state machine is, to sum up incoming data from the deserialazer FIFO and calculate the mean values out of the sums. The main difference to the previous stage is, the summation is not only done for odd and even pixels separately, but for every single pixel. In this stage the global values from the previous stage are used to detect lines, which may contain a pellet event from the camera exposure. The finite state machine for Conf1 is shown in figure 22.
At first, the state is in “000” also called reset state. In this state, all the flags, counters and memory address registers are reset. In “000”, the FSM waits for the line valid flag to turn to ‘0’ and also for conf1_go to turn to ‘1’. The conf1_go turns to ‘1’ only when the first stage has gone through at least one time, and global mean values are available from the conf0 stage. The line valid signal and conf1_go are ‘1’ in the “000” state, then the FSM jumps to the “001” state. In “001” state a summation block memory write enable bit is set to one, to store the sums of incoming data for each pixel.

Here the summation block memory has a depth of 256 rows is shown in figure 23. Each row holds the sum for one even and one odd pixel, so the depth of the memory covers one-line exposure of a camera (512 pixels).

In this stage, one needs to configure the design to collect data for a number of exposures. The maximum can add up from one camera is set to be 1024 lines. The result’s width of the memory is 44bits. Hence a 12bit value is integrated 1024 times, the resulting sum will have a 22 bit representation (1024 *12 bits, which gives 44bit and in that, 22bits are odd and next 22bits are even).
In the state “001” the FSM waits for the line valid signal to turn to ‘1’, if so the state jumps to the “010” state. In the “010” state, the FSM stays until one full line is transmitted. In every clock cycle, new data of two pixels are added to the content of the corresponding memory cell and the result is stored in the next cell. In the “010” state, the block memory address is increased by one for every clock cycle when the content in the memory is being read, new data is added to the memory read data and a sum is stored in the same place. It might be a situation that there is a pellet event while running this procedure. That means, in the line few pixels contain more than their pedestal values. That’s why before storing the new value in the summation memory, the values are checked for actual pixel values larger than the global pedestal values from the stage Conf0 init_mean_even, init_mean_odd values and an arbitrarily chosen ‘pellet-threshold’. If the line is an incomplete line then the FSM jump back to the reset state “000”. If the line is correctly transmitted then the FSM jump to the “011” state. In the “011” state, the line counter is increased by one and the FSM goes to the state “100”. If the line counter overflows the FSM jump back to the state “010” accumulating new exposure data from the deserializer FIFO. If the FSM jump to the “100” state, it means summation memory is filled with the sum of 1024 lines for each pixel. Now we need to calculate mean values for every pixel stored in different block memory, outside VHDL entity Conf1.

The summation memory (Figure 23) is read out for every clock cycle and separates the value into ODD and EVEN pixels. The pixel values are truncated by 10bits, which again creates mean values. The mean values are writing to an external block memory. This external memory is similar to the summation memory; the data width is 24bits instead of 44bits. To fill the external block memory, this operation takes 256 clock cycles. Once this operation is performed, a filled memory is cleared and the FSM jumps to the “010” state for accumulating new data. The automatic pedestal correction conf1 also keeps on running all the time like the previous pedestal correction conf0. It also means that, the external memory is updated for every at 1025 exposures of the camera. One has to take care while
setting up the cameras; because the pedestal values might change because of ambient light, reactions etc.

The behavior of the conf1 code was simulated in ISim and tested with the chipscope. The simulation result of conf1 is shown in figure 24. The simulation is only for 8 lines instead of 1024 lines of summation data. The simulation shown in figure 24, one can see that the state stays in “010” almost all the time during accumulation. In the figure, the red line indicates an overflow line counter and truncated data are transferred to the pedestal block memory from the summation memory (MEM_WE turns ‘1’, MEM_DATA is containing valid content). In figure 20 the red line shows the ODD pixels delivered by the camera and blue line shows the MEM_DATA for the ODD pixels. The MEM_DATA is stored in the pedestal BRAM (Block Random-Access Memory).

![Figure 24: simulation result for conf1](image)

6.2 Pedestal correction

The Pedestal correction is using the pedestal values (MEM_data and MEAN_MAX) to perform an offset (pedestal) error correction for each incoming pixel. The reason is, there might be a visible difference between ODD and EVEN pixels, because two ADCs are used in the camera. One can correct the data after knowing the pedestal value for each pixel. Once the data are corrected all the pixels are on equal level and by using these values, it is easier to detect a pellet using a simple threshold.

6.3 Pellet recognition

The pellet recognition code detects pellet events and outputs the necessary information about every pellet. The pellet recognition uses pedestal corrected data.
To start the pellet recognition code, the pedestal correction part has to complete the following steps:

Both automatic configuration Conf0 and Conf1 has executed at least once and a pellet recognition start signal is set to ‘1’ to start the pellet recognition part. At this stage, the pedestal memory is filled with the pedestal values. Additionally to above signals, the stage Conf1 is providing an additional signal to the pellet recognition part named MEAN_MAX. This MEAN_MAX signal contains the largest pedestal value. All values should be corrected to a level of MEAN_MAX + Δmin, which means that all values will be corrected upward. For new incoming data, the steps are as follows:

For every pixel ‘i’

- DC(i)=DR(i)+Corr(i)
- Corr(i)=MEAN_MAX+ Δmax− P(i)

Where

DC → data corrected
DR → data raw (actual camera value)
Corr → correction
MEAN_MAX → the highest mean value from all pixels
Δmax → Security margin
P(i) → pedestal value

Figure 25 shows the chipscope result of raw data of ODD and EVEN pixels (red indicates ODD and blue indicates EVEN); the light red and light blue are the pedestal corrected data.

Figure 25: Raw data (red+blue) and pedestal corrected data (light red and light blue) in ChipScope
The pellet recognition part is also controlled by a finite state machine (FSM). The FSM for the pellet recognition is shown in the figure 26. At first, the state should be at the “000” state. In “000” state, the FSM waits for a pellet recognition start signal (RECO_GO=’1’), this only happens once the two stages conf0 and conf1 are finished. In the “001” state, the FSM waits for the line valid flag (LVAL=’0’) then jumps to next state “010”. In the “010” state all flags are reset and the FSM waits for the line valid flag (LVAL=’0’) then jumps to the “011” state. Basically, from state “000” to state “010” are reset states. In the “011” state, a simple pellet detection algorithm was implemented to detect real pellets. A real pellet detection is carried out at the “011” state. If the pixel is greater than the MEAN_MAX value plus a globally defined pellet threshold, then it is considered as a real pellet event. This pellet event is referred to be a pellet_candidate. In the line, if some other pixel is not greater than the pellet_candidate then the pellet_candidate is the output of pellet detection. If some other pixel in the line is greater than the pellet_candidate then this new pixel is the pellet_candidate. The algorithm works to detect one pellet per a line. In case two pixels are visible in a line, then the one with larger amplitude is sent out. The outputs of the pellet recognition entity contain position in the line, the amplitude of the pellet event and the
pellet status flag. The pellet status flag is 1 bit wide, indicating that the pellet has been found. The position is 9 bits wide, which contains the maximum amplitude pixel position in the line. The amplitude is 12 bits wide, which contains the largest value of the pixel in the line. This implementation was designed for four cameras. The chipscope result of the pellet recognition part is shown below in figure 27. The red arrow mark indicates the pellet found a location in the line and the corresponding pellet values are displayed in the waveform.

![Chipscope result of pellet recognition](image)

Figure 27: Chipscope result of pellet recognition

Now we have the pellet information of the four cameras namely pellet amplitude (12 bit) and pellet position (9 bit). Now we need to generate timestamps for the pellet events. The explanation of the timestamp generation is following.

### 6.4 Timestamp

A simple process statement is used to generate the timestamp. The width of the timestamp is a 44 bit wide number. A simple counter is used in the process; this counter is increased by one every time, once the full line is read from a camera. Every line will get a unique number, since the FPGA system was powered up. Each camera has an individual timestamp.

### 6.5 Temporary Registers

The pellet information is resampled with a speed of the optical data transmitter (50MHz) and stored in the temporary registers for further processing.
7 TXDATA FORMAT

As soon as the system is powered up the cameras are connected to the Spartan 6 FPGA board, the hydrogen pellet detection part starts searching for pellet events. The pellet information and the slow control data are transmitted via an optical link to the Virtex 5 FPGA board. The optical link is connected to the GTP transceiver of the two FPGA boards as shown in figure 4. Before transmitting the data to the Virtex 5 FPGA board, the data is formatted in the following way and stored in the FIFO, named this FIFO “TXDATA_FIFO”.

<table>
<thead>
<tr>
<th>Pellet information</th>
<th>Slow control data</th>
<th>Slow control status signal</th>
<th>Pellet found signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:10]</td>
<td>[9:2]</td>
<td>[1]</td>
<td>[0]</td>
</tr>
</tbody>
</table>

The reason for using the FIFOs is, in case the camera finds pellet events and these events are transmitted to the Virtex 5 FPGA board. The GTP transmitter needs five clock pulses in order to transmit the single pellet information. Meanwhile if the same camera finds new pellet events, the pellet temporary registers are overridden with new values. To eliminate the data loss, FIFOs are used. The instance of the TXDATA_FIFO is shown in figure 28.

The GTP transmitter reads the data from this TXDATA_FIFO and transmit it to the Virtex board via an optical link. The width of the FIFO_TXDATA is 32bit. At each clock pulse, a 32bit data is stored in the FIFO. We have three kinds of information about the pellet: pellet position, pellet amplitude and timestamp.

Storing pellet data and slow control rx data in the TXDATA_FIFO is done in the following order.

<table>
<thead>
<tr>
<th>First word</th>
<th>Sending the camera ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second word</td>
<td>pellet position plus slow control receiver data</td>
</tr>
<tr>
<td>Third word</td>
<td>pellet amplitude plus slow control receiver data</td>
</tr>
<tr>
<td>Fourth word</td>
<td>pellet timestamp first 22bits plus slow control receiver data</td>
</tr>
<tr>
<td>Fifth word</td>
<td>pellet timestamp last 22bits plus slow control receiver data</td>
</tr>
</tbody>
</table>
A Finite State Machine (FSM) for the TXDATA formatting is shown in figure 29. The state machine works for four cameras. Initially, it is at the “000” state also called reset state. In the “000” state the FSM waits for a pellet found signal and the slow control rx valid signal from the four cameras. If the FSM finds that if any of the cameras have two high signals then the FSM enables the corresponding FIFO write bit and writes the 32bit append data i.e. camera ID, slow control rx data, slow control rx valid signal and pellet found in the TXDATA_FIFO. Then the FSM jumps to the “001” state. In case, if the FSM finds only when the pellet found signal is high then, it only writes camera id and pellet found data in the FIFO and then jumps to the “001” state. The FSM writes null values if the slow control data is not available for the FSM. If the two signals (pellet found and slow control valid) are zero then the FSM remains in the same state. In the “000” state, if the FSM finds only when the slow control signal is high then the FSM enables the corresponding FIFO write bit and it will write only camera id and slow control rx data in the TXDATA_FIFO and waits in the same state i.e. “000”. In the “001” state, if the FSM finds that slow control rx valid signal is high then the FSM writes pellet position data along with slow control rx data in the TXDATA_FIFO, if not it will write only pellet position data and then jump to the “010” state. In the “010” state, the FSM writes the pellet amplitude data along with the slow control in the TXDATA_FIFO, if the FSM finds the slow control rx valid signal is high otherwise it will write only pellet amplitude data and then jump to the “011” state. In the “011” state, the FSM writes the first 22bit timestamp data in the TXDATA_FIFO along with slow control data, if the FSM finds the slow control rx valid is high otherwise it will write the timestamp data and then jump to the “100”. In the “100” state, the FSM writes remaining 22bit data along with slow control data into the FIFO, if the FSM finds the slow control rx valid signal is high otherwise it will write the timestamp data and then jump to the reset state. The above procedure run until the system is turned off.
Figure 29: FSM for TX data format
8 GTP Transceiver

In the project, the GTP transceiver modules are used for communication between the Spartan6 board and a 16-channel optical data collector board (Virtex 5 FPGA board). The GTP transceivers are based on fast serial communication technology. A GTP transmitter serializes the data and sends it through an optical link to a GTP receiver. Once the receiver gets the data, it deserializes it for further processing. Slow control commands are sent from the data collection board, though the GTP and optical link to the Spartan6 FPGA board.

8.1 GTP Transmitter

The GTP transmitter block diagram is shown in figure 30. The GTP transmitter consists of two main blocks: a front end and far end. The front end is used for signal serializing and transmission and the far end block is used for data coding and adjusting (17).

![GTP Transmitter Block Diagram](image)

**Figure 30: GTP Transmitter Block Diagram**

### 8.1.1 8B/10B encoding technology

In telecommunications, 8b/10b coding (17) is often used to map 8-bit symbols in 10-bit symbols to achieve DC-balance and bounded disparity, and yet provide enough state changes to allow reasonable clock recovery. The 8b/10b encoding 8-bit data into two groups, one of them has 3-bits and the other has 5-bits. The 3-bits group is encoded into 4-bits and the 5-bits group is encoded into 6-bits.

Suppose the data we want to send is “10110101”:
First we divide it into “101” and “10101”

10101= (21), 101= (5)

So “10110101”=D21.5, then we check a lookup table of 8b/10b as shown below, we get the encoded value.

<table>
<thead>
<tr>
<th>Data Byte Name</th>
<th>Bits HGF EDCBA</th>
<th>Current RD – abcdei fghj</th>
<th>Current RD + abcdei fghj</th>
</tr>
</thead>
<tbody>
<tr>
<td>D21.5</td>
<td>101 10101</td>
<td>101010 1010</td>
<td>101010 1010</td>
</tr>
</tbody>
</table>

The 8b/10b assigns two different symbols to each value. In most cases, one of the symbol contains 6 ‘0’ and 4 ‘1’, another is composed of 4 ‘0’ and 6 ‘1’. The encoder will keep on checking the number of 0 and 1 from the data and select the symbol according to the demand, to ensure the DC balance on the line. These two symbols are usually referred as ‘+’ and ‘-’ as can be seen from the table above. The encoding function is integrated in the GTP transmitter module so it will not use any FPGA logic.

### 8.1.2 K Characters

In the 8b/10b tables, there are some special characters which we can use for control. Like comma detection, word alignment. They are called K characters.

<table>
<thead>
<tr>
<th>input</th>
<th>RD = –1</th>
<th>RD = +1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HGF EDCBA</td>
<td>abcdei fghj</td>
</tr>
<tr>
<td>K. 28. 0</td>
<td>000 11100</td>
<td>001111 0100</td>
</tr>
<tr>
<td>K. 28. 1</td>
<td>001 11100</td>
<td>001111 1001</td>
</tr>
<tr>
<td>K. 28. 2</td>
<td>010 11100</td>
<td>001111 0101</td>
</tr>
<tr>
<td>K. 28. 3</td>
<td>011 11100</td>
<td>001111 0011</td>
</tr>
<tr>
<td>K. 28. 4</td>
<td>100 11100</td>
<td>001111 0010</td>
</tr>
<tr>
<td>K. 28. 5</td>
<td>101 11100</td>
<td>001111 1010</td>
</tr>
<tr>
<td>K. 28. 6</td>
<td>110 11100</td>
<td>001111 0110</td>
</tr>
<tr>
<td>K. 28. 7</td>
<td>111 11100</td>
<td>001111 1000</td>
</tr>
<tr>
<td>K. 27. 7</td>
<td>111 11011</td>
<td>111010 1000</td>
</tr>
<tr>
<td>K. 29. 7</td>
<td>111 11011</td>
<td>110110 1000</td>
</tr>
<tr>
<td>K. 30. 7</td>
<td>111 11110</td>
<td>011110 1000</td>
</tr>
</tbody>
</table>
8.2 GTP Receiver

The block diagram for the GTP receiver is shown in figure 31. In the first stage the SIPO (serial in parallel out) will convert the input serial data into parallel for FPGA to process. Later on the Comma Detection and Alignment module will reorganize the data and tell us where the useful data starts. Then the useful data will go through the 10b/8b decoding module and the output of this module.

![Figure 31: GTP Receiver Block Diagram](image)

8.2.1 RxPolarity

The GTP RX module has a polarity function. The engineer may accidentally put the differential receiver RXP and RXN pin in wrong position when designing the PCB. By using the polarity function, it is possible to invert the order of RXP and RXN just by setting RXPOLARITY port high.

8.2.2 RX word alignment

A special pattern COMMA is used to detect the boundaries of incoming data. COMMA K28.5 is a special character in 8B/10B coding.
The figure above tells us that once we find the COMMA in the data stream we get to know the boundary of incoming data.

### 8.2.3 RX Loss-of-Sync State Machine

The 8B/10B protocol requires us to implement a loss of sync mechanism if the channel of data is not working correctly. Every GTP is using a state machine to recover the problem and the procedures of the state machine is shown in figure 32.
From the state above we can find out that there are two attributes RX_LOS_INVALID_INCR and RX_LOS_THRESHOLD. We can change the Sensitivity of the state machine by adjusting these two parameters.
A GTP IP Core is used to send the data to the Virtex 5 FPGA board via an optical fiber link. Before making use of an IP core, it needs to be configured in order to comply with the design requirements. This IP core is a dual tile; a dual tile is a set of two GTP transceivers. However, in the design only one tile is used and another is disabled. The block diagram for the GTP design is shown in figure 33. The basic explanation for the GTP transceiver is explained above (8-8.2.3). The detailed information can be found on the Spartan-6 FPGA GTP Transceivers (17). The GTP transceiver module has its own clock 100MHz. The IP core has an internal PLL design; by using this PLL, two clocks 50MHz and 200MHz were generated in this GTP design. The GTP transmitter transmits the input parallel data serially to the Virtex 5 FPGA by using a 50MHz clock. The data reception takes place with a high speed clock (200MHz) with reference to the 50MHz clock. As explained in chapter 7, the GTP transmitter transmits the pellet data and the slow control data. These data are stored in the TXDATA_FIFO. The FSM is designed to read the data from the TXDATA_FIFO and provides input to the GTP transmitter. The GTP transmitter transmits the parallel input data serially via an LVDS optical link to the Virtex 5 FPGA board. The FSM for the GTP_Transmitter_TXDATA is shown in figure 34.

Figure 34: FSM for GTP_Transmitter_TXDATA

Initially the FSM is at reset state i.e. “000”. In this state all the cameras’ TXDATA_FIFO read bits are disabled, TX_DIS is asserted to ‘0’ (enable pin for the GTP transmitter) and the FSM sends a K character i.e. “000000BC” as input data to the GTP transmitter and then jumps to
the “001” state. In order to transmit input data as a k character, we have to assert txcharisk0 to “0001” (we are sending only one character so first bit is set to ‘1’). In the “001” state, the FSM checks if any of the camera FIFO’s fifo_txdata_empty signals are low. If the FSM finds that first camera fifo_txdata_empty signal is low then the FSM jumps to the “010” state. In the “010” state, the FSM enables the corresponding TXDATA_FIFO read bit (fifo_txdma_rd_en) and reads the data from fifo_txdma_out signal and transfer it to the input transmitter signal (TXDATA_IN). In the “010” state, the txcharisk0 is set to “0000” because this signal is set high only when the TXDATA_IN signal has a K character data. In the “010” state, the FSM checks for the fifo_txdma_empty signal, if it’s low then the FSM remains in the same state and reads the data from the TXDATA_FIFO. If the fifo_txdma_empty signal is high then the FSM asserts the TXDATA_FIFO read bit (fifo_txdma_rd_en) to low and then it Jumps to the reset state. The same procedure works for the remaining cameras. Whenever the camera data is available in the corresponding TXDATA_FIFO’s then the FSM jumps to corresponding state, enabled corresponding read bit and provides data to the TXDATA_IN signal.
9 RX DATA FORMAT

The Virtex 5 FPGA board sends the camera configuration commands to the Spartan 6 FPGA board, which is sent by the user from the PC to the Virtex 5 FPGA board. The GTP receiver on the Spartan 6 FPGA board receives LVDS serial data and converts this to parallel form. The received parallel data is in the following format.

RXDATA=x"00000"&"00" & (camera_command_data) & camera_command_valid & '0';

The receiver first receives the camera ID data and then the camera configuration commands. A state machine is designed to store the configuration commands in individual FIFOs named “RXDATA_FIFO”, based on the received camera IDs. The schematic diagram for a RXDATA_FIFO is shown in figure 35. In the project, an IP Core FIFO is used, which is provided by Xilinx library. The width of the input and output data of a FIFO is 8 bits and the depth is 64. The WR_clk for the FIFO is 50MHz (GTP clk) and RD_clk has baudrate of about 9600.

![Figure 35: RXDATA_FIFO block diagram](image)

The FSM is used for storing camera commands in an individual RXDATA_FIFO as shown in figure 36. At first, the FSM is in state “0000” also called the reset state. In the state “0000”, if the FSM receives Camera_Command_Valid='1' and Camera_Command_Data=x"00" (first camera ID) then the FSM jumps to the “0001” state. In the “0001” state, the FSM enables the RXDATA_FIFO write bit signal of the first camera. The Camera_Command_Data signal keeps writing in its FIFO until the Camera_Command_Data signal receives a carriage return(0D). If the signal receives a carriage return (0D) then the RXDATA_FIFO write bit signal is disabled and the FSM jumps to the “0101” state. In the “0101” state, the FSM checks for the camera ID on the Camera_Command_Data signal. If the signal contains a camera ID data, then the FSM jumps to the “0001” state and will repeat the same procedure as explained above, otherwise it jumps to the reset state. In the reset state, the FSM waits again for the camera ID and the Camera_Command_Valid signal. Based on the camera ID, the FSM jumps to corresponding state, enables the camera RXDATA_FIFO for writing the
data. Based on the camera ID information, the FSM enables the corresponding RXDATA_FIFO to store the camera commands. All the camera commands end with a carriage return (0D). Once the carriage return is received on the Camera_Command_Data signal, then the FIFO write bit signal is disabled.

Figure 36: FSM to store camera commands in the FIFO
10 Slow Control

The slow control is designed for configuring the cameras. A set of configuration commands is given in the AViiVA M2 CL (4) camera data sheet. By using the commands, we can do the following things.

1. Configure the settings of the camera.
2. Readout camera configuration.

The camera link interface provides two LVDS signal pairs for communication between the camera and the Spartan 6 FPGA board. The communication is using the RS232 protocol (6). The slow control is designed for four cameras. The schematic diagram of the slow control is shown in figure 37.

![Figure 37: Slow control schematic](image)

As explained in the previous chapter, the received slow control data are stored in the individual FIFO’s. These data are read by the slow control transmitter and data is sent serially to the corresponding camera. Based on the transmitted command type, the camera will send the response data to the Spartan 6 FPGA board. The slow controller receiver
receives the serial data from a camera, converts them to parallel data and stores it in the corresponding FIFO.

The configuration for the serial communication line is set as follows.

- This interface operates in a full duplex manner without handshaking.
- The transmitter clock uses a clock of 9600 baud and transmits the data in the following order: 1 start bit, 8 bit data and 1 stop bit.

![Serial Communication Diagram]

From the Spartan 6 FPGA board, we can send two different types of commands: write and read. The basic command structure is as follows.

"S = n(CR)"

Where S: command character

n: setting value

CR: Carriage Return

Every command ends with the carriage return (OD).

An example of a valid command is

G = 3(CR)

The above command sets the camera gain value to 3.

The above command is transmitted byte by byte from the Spartan 6 FPGA board to the camera. The camera processes the received command. If the command is a valid command, then the camera will send response data to the Spartan 6 FPGA board.

- If the command is a write command, then it sets the camera and sends a camera response data as >OK(CR).
- If the command type is a read command, then the camera sends the response data separated by a carriage return (CR).

In case of a not valid command, the camera will not change its settings and send a camera response in the following form:

1. >1 = out of range;
2. >2, >3 = illegal command;
3. >4, >5, >6 = internal error;
4. >4, >6, >7 = internal error;
2. >2 = syntax error;  5. >5 undefined function;  
3. >3 = command too long;  

An example of the camera response to a read command is given below.

Received command of the camera: "! = 3(CR)"

The above command is for reading the current settings of the camera. The camera response data are as follows.

- A = 0(CR); B = 0(CR); ...; E = 0(CR); >OK(CR).

**10.1 Slow control transmitter**

The slow control transmitter reads the parallel data from the RXDATA_FIFO (ch: 9) and transmits these data serially to the camera. The schematic diagram for the slow control transmitter is shown in figure 38.

![Figure 38: Slow control transmitter schematic](image)

A state machine is designed to read the data from the “RXDATA_FIFO” and transmit this data serially to the camera. An FSM for the slow control transmitter is shown in figure 39. When the FIFO is empty, a signal uart_fifo_empty is asserted (high). If any of the Empty signals indicate low, it means the data is available in the FIFO. These data are sent to the camera using a slow control transmitter FSM. At first, the FSM is at the idle state “000”. If any of the RXDATA_FIFO’s empty signal indicates low then the FSM jumps to the “001” state. In the “001” state, the FSM enables the corresponding RXDATA_FIFO read bit signal (uart_rd_en), loads the 8bit data into the register and then jumps to the next state “010”. In the “010” state, the corresponding RXDATA_FIFO read bit is disabled and the 8bit data is encapsulated by a start and stop bit and then the FSM jumps to the “011” state. In the “011” state, 10bit data is sent serially by using a shift operation. Here a counter is used to
determine the number of bits transmitted by the transmitter. If the counter reaches 9, the FSM jumps to the “100” state. In the “100” state, the transmitter sends the stop bit i.e ‘0’ and a uart_tx_done signal is asserted for one clock cycle. This signal is asserted for one baud clock pulse after every byte transmission. In the “100” state, the FSM checks for four RXDATA_FIFO’s empty signals. If any one of the RXDATA_FIFO’s empty signals indicates low, then the FSM jumps to the “001” state and repeats the same procedure as explained above. If the FIFO’s empty signal is high in the “100” state then the FSM jumps to the reset state (000). The transmitted serial data from the Spartan 6 FPGA board to the first camera is shown below from using the chipscope pro analyzer.

Figure 39: FSM of UART transmitter

Figure 40: slow transmitter data from Spartan 6 FPGA board

An example command which we sent from the PC is “213D330D” is shown in figure 40. This data travels from Virtex 5 FPGA board to the Spartan 6 FPGA board and is stored in the
“RXDATA_FIFO”. The slow control transmitter reads the “RXDATA_FIFO” data and sends the data serially to the camera.

The above command is for “Camera configuration readout”. The camera response data is sent serially to the Spartan 6 FPGA board. On the Spartan 6 FPGA board, another FSM is designed to convert serial data to parallel form. This FSM is called the slow control receiver.

10.2 Slow control Receiver

The serial data sent from the camera at the particular baud rate is converted to parallel using a receiver module. During the transmission every byte is encapsulated by a start and a stop bit. A start bit is represented by a zero signal and a stop bit by a high signal for one baud clock cycle.

Since no clock information is conveyed from the transmitted signal, the receiver can retrieve the data bits only by using the predetermined parameters (baud rate). An over sampling scheme is used to estimate the mid points of transmitting bits and then retrieve the rest of data bits in subsequent clock cycles.

Over sampling method:

In the over sampling method the received data on the Spartan 6 FPGA board is sampled using sampling clock (153KHz), which is 16 times faster than the baud clock. To detect the start of transmission the incoming signal is sampled to detect a ‘0’, which is the beginning of the start bit. At this point a counter clocked at the sampling rate is started. When the counter reaches 7, the incoming signal reaches the middle of the start bit.

Once the midpoint of the start bit is detected, each of the data bits is retrieved at baud clock frequency. The retrieved value is shifting into a register. This method ensures that the data bits received from the receiver port are read near the midpoint of each bit duration to avoid detection errors.
The schematic diagram for the slow control receiver is shown in figure 41.

![Schematic Diagram for UART Receiver](image1)

**Figure 41: Schematic for UART Receiver**

The FSM for the slow control receiver is shown in figure 42. At first, the FSM is at the '000' state which is also called 'Idle state'. When a start bit is detected at the "000" state then the FSM jumps to the "001" state. In the "001" state, a counter clocked at sampling rate is started. When the counter reaches 7, the incoming signal reaches the middle of the start bit then the FSM jumps to the "010" state. In the "010" state, the data bit counter starts from zero. In the "010" state, the counter checked for 15 to read the received data at the midpoint.
of the duration. When the data bit counter reaches 8 then the FSM jumps to the “011” state. In the “011” state, the FSM detects the stop bit, the data valid signal is asserted and then the FSM jumps back to the “000” state (Idle state).

The commands are sent to one camera at a time and the camera response data are received. The chipscope result of the slow control receiver for the command “213D330D”, which is sent by the slow control transmitter, is shown below.

![Figure 43: Slow control receiver data](image1)

In the figure 43, for transmitter command “213D330D”, the camera sent a response data serially to the Spartan 6 FPGA board. Later this information is sent to the Virtex 5 FPGA board from the GTP transmitter of the Spartan 6 FPGA board via an optical link. So, four camera response data are stored individually for further processing. To store individual camera response data another four FIFO’s are used, named FIFO’s “UART_RX_FIFO”. The FSM is designed to store a camera response data in individual UART_RX_FIFO’s.

![Figure 44: FSM to store the camera response into the UART_RX_FIFO](image2)
Figure 44 shows, the FSM to store the camera response data individually in the UART_RX_FIFO’s. The FSM works for four cameras. Initially, the FSM is at the “000” state, which is also called “reset state”. In this state, the FSM checks for a valid signal (uart_rx_valid) in all the four cameras. This valid signal indicates high whenever the slow control receiver receives data from the camera. Among the four cameras, whichever camera sends a data to the slow control receiver, the corresponding valid signal is high. In any of the slow control receivers, if the valid signal is set high, the FSM jumps to the “001” state. In the “001” state, the corresponding FIFO write enable bit is set. In this state, the slow control receiver data are written to its corresponding FIFO and the FSM jumps to the “010” state. In the “010” state, the UART_RX_FIFO write bit is disabled and the FSM checks, if any of the slow control valid signals (uart_rx_valid) is high then the FSM jumps to the “001” state. If the valid signal is not high in the “010” state, the FSM checks for carriage return from the slow control receiver data output signal (uart_cam_fb_data). If the slow control receiver receives a carriage return, the FSM jumps to the “011” state. In this state, the FSM checks for two signals, one is camera’s valid signal (uart_rx_valid) and another is a carriage return. If the camera’s valid signal is high, then the FSM jumps to the “001” state, otherwise the FSM checks if the slow control receiver data output is a carriage return and it is the same for 200 clock pulses then the FSM jumps to the reset state. The carriage return is the end of the camera response data. In the “011” state, the fifo_ready_153KHz signal is asserted for corresponding FIFOs, which indicates to the GTP module to read the data from a slow control FIFO. This procedure runs unless the system is turned OFF.

Now the UART_RX_FIFO data are sent to the “TXDATA_FIFO”. The “TXDATA format” FSM reads the data from the UART_RX_FIFO to the TXDATA_FIFO with the help of the fifo_ready_153KHz signal. However, here a problem arises, because the clock signals of the two modules are different (i.e. UART_RX_FIFO, TXDATA_FIFO). The “TXDATA FORMAT” FSM is using a 50MHz clock which is 326 times faster than the slow control receiver clock. The “TXDATA FORMAT” FSM reads the same data 326 times from the UART_RX_FIFO, if the data is read by using the fifo_ready_153KHz signal. To solve this problem, we designed the FSM to enable a fifo_ready_153KHz signal for one GTP clock cycle. For synchronization purpose, first the fifo_ready_153KHz signal is stored into a register, named fifo_ready_50MHz, with the help of GTP clock. The FSM for the 50MHz ready signal is shown in figure 45. This FSM works for four cameras. Initially, the FSM is at the “00” state. In this state, the FSM waits for fifo_ready_50MHz to become one. When this signal is asserted then a 50MHz ready signal named fifo_ready_50MHz_1 is set one for one GTP clock pulse, and then the FSM jumps to the “01” state. In the “01” state, the fifo_ready_50MHz_1 is set to zero and FSM waits for the fifo_ready_50MHz signal to become zero. When the fifo_ready_50MHz signal is set to zero, the FSM jumps to the “10” state. In the “10” state, the FSM waits for the fifo_ready_50MHz_1 signal to become one, to
assert the **fifo_ready_50MHz** signal and the FSM jumps to “01” state otherwise the FSM remains in the same state.

![Diagram](image)

**Figure 45: FSM for 50MHz ready signal**

By using **fifo_ready_50MHz_1** signal, the “TXDATA format” FSM reads the data from the slow control FIFO (UART_RX_FIFO) and stores the data in the “TXDATA_FIFO”.

56
11 Performance

11.1 Test runs using "Fish line" (Optical fiber) setup:

The whole design was tested with a simple setup shown in figure 46. In the figure, one can see that a camera was mounted on an adjustable aluminum table. The same setup can also be used in Uppsala Pellet Test Station (UPTS). The exact position (height and position) of the camera to the pellet beam can be adjusted by using the two micrometer screws. The camera was facing toward transparent plastic fiber, which was illuminated by a white LED. The LED was connected to a pulse generator which generates short pulses with an adjustable repetition frequency of 12 KHz. If the position of the camera exactly faces the illuminated plastic fiber, it captures the plastic fiber. The signals from the camera were very similar to the real pellet events.

![Figure 46: Test setup using an illuminated plastic](image)
The above setup was used for tests with chipscope pro analyzer during the development of the VHDL code. The integration of the VHDL code was tested on the Spartan 6 FPGA board used during the whole design. Some results of the calculated plastic fiber amplitude, position and timestamp are shown in Chipscope Pro analyzer window.

![Graph showing camera ID, pellet position, amplitude, and time stamp](image)

**Figure 47:** GTP transmitter contains pellet data

The pellet data are in hex format. The above data are the input data for the GTP transmitter. These data are transmitted via an optical link fiber to the Virtex 5 FPGA board. In the figure 47, the pellet data are transmitted. A situation, where both the pellet data and slow control data are transmitted is shown in Figure 48.

![Graph showing both pellet data and slow control data](image)

**Figure 48:** GTP transmitter contains pellet data and slow control data

A situation, where the slow control data are transmitted is shown in figure 49.

![Graph showing only slow control data](image)

**Figure 49:** GTP transmitter contains slow control data

The data received by the Virtex 5 FPGA board are stored in a FIFO. The VHDL code for data buffering and a VME interface inside the Virtex 5 FPGA board was developed by Pawel Marciniewski (Supervisor). A VME Readout controller (SBS Technologies) reads the data from the Virtex 5 FPGA board and sends them to a PC via a proprietary optical link (see figure 37). A command line interface was already developed in PC, for sending the
commands and to read the Virtex 5 FPGA data. A picture of the command line interface is shown in figure 50.

```
Address relativeto csr = 300000
Value to write = 0000213d
Execute number of times (> 5 = infinite) = 1
Write and read back:
What would you like to do?
e : exit
c : read csr
r : read register relative to csr address
w : write address relative to csr address
z : clear
g : get data
p : print data to screen
d : dump data to binary file
x : write formatted data to file
Address relativeto csr = 300000
Value to write = 330d0000
Execute number of times (> 5 = infinite) = 1
Write and read back:
What would you like to do?
e : exit
c : read csr
r : read register relative to csr address
w : write address relative to csr address
z : clear
g : get data
p : print data to screen
d : dump data to binary file
x : write formatted data to file
```
A VME readout controller reads the data from the Virtex 5 FPGA FIFO into PC via optical link is shown below.

<table>
<thead>
<tr>
<th>CameraID</th>
<th>Pellet Position</th>
<th>Pellet Amplitude</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ff000001</td>
<td>3d401</td>
<td>2b401</td>
<td>be7df801</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd00235e</td>
<td>ff000001</td>
<td>3d401</td>
</tr>
<tr>
<td>2bc01</td>
<td>be9a1801</td>
<td>1f0401</td>
<td>fd00235f</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d401</td>
<td>2b001</td>
<td>be9fb801</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd002360</td>
<td>ff000001</td>
<td>3d001</td>
</tr>
<tr>
<td>2b001</td>
<td>beb6d801</td>
<td>1f0401</td>
<td>fd002361</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d401</td>
<td>2ac01</td>
<td>bec17801</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd002362</td>
<td>ff000001</td>
<td>3d401</td>
</tr>
<tr>
<td>2b801</td>
<td>bee33801</td>
<td>1f0401</td>
<td>fd002363</td>
</tr>
<tr>
<td>ff000001</td>
<td>3e801</td>
<td>2b001</td>
<td>beee7801</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd002364</td>
<td>ff000001</td>
<td>3d401</td>
</tr>
<tr>
<td>2b001</td>
<td>bf04f801</td>
<td>1f0401</td>
<td>fd002365</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d401</td>
<td>2b001</td>
<td>bf0e9801</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd002366</td>
<td>ff000001</td>
<td>3d401</td>
</tr>
<tr>
<td>2ac01</td>
<td>bf26b801</td>
<td>1f0401</td>
<td>fd002367</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d001</td>
<td>2ac01</td>
<td>bf2c5801</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd002368</td>
<td>ff000001</td>
<td>3d001</td>
</tr>
<tr>
<td>2b801</td>
<td>bf487c01</td>
<td>1f0401</td>
<td>fd002369</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d001</td>
<td>2b401</td>
<td>bf649c01</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd00236a</td>
<td>ff000001</td>
<td>3d001</td>
</tr>
<tr>
<td>2b801</td>
<td>bf6a3c01</td>
<td>1f0401</td>
<td>fd00236b</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d001</td>
<td>2ac01</td>
<td>bf6fadc01</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd00236c</td>
<td>ff000001</td>
<td>3d401</td>
</tr>
<tr>
<td>2b801</td>
<td>bf8bc01</td>
<td>1f0401</td>
<td>fd00236d</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d401</td>
<td>2ac01</td>
<td>bfa81c01</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd00236e</td>
<td>ff000001</td>
<td>3d001</td>
</tr>
<tr>
<td>2b401</td>
<td>bfadc01</td>
<td>1f0401</td>
<td>fd00236f</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d401</td>
<td>2bc01</td>
<td>bf7c7c01</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd002370</td>
<td>ff000001</td>
<td>3d401</td>
</tr>
<tr>
<td>2ac01</td>
<td>bfebc01</td>
<td>1f0401</td>
<td>fd002371</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d401</td>
<td>2ac01</td>
<td>bff13c01</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd002372</td>
<td>ff000001</td>
<td>3d001</td>
</tr>
<tr>
<td>2ac01</td>
<td>c012c01</td>
<td>1f0401</td>
<td>fd002373</td>
</tr>
<tr>
<td>ff000001</td>
<td>3d401</td>
<td>2ac01</td>
<td>c02f1c01</td>
</tr>
<tr>
<td>1f0401</td>
<td>fd002374</td>
<td>ff000001</td>
<td>3d401</td>
</tr>
</tbody>
</table>
12 Conclusion and Future work

Through this thesis, VHDL code for a Spartan 6 FPGA board was developed, which is capable of reading from four CCD line scan cameras. The deserializer design (Chapter 5) on the Spartan 6 FPGA board converts the serial data into parallel form. The hydrogen pellet detection design (Chapter 6) takes the parallel data and performs the real pellet recognition. The detected pellets on the Spartan6 FPGA board are transferred to the Virtex 5 FPGA board via a GTP-optical fiber link (Chapter 8). The Virtex 5 FPGA board data are read by a VME readout controller and transferred to a PC via an optical link. Currently, detected pellet events are stored in the PC. Cameras are configured by the Slow control design (Chapter 10) and output data are also stored in the PC along with Pellet data.

The firmware can be developed further to calculate the trajectory of the hydrogen pellet by using the Pellet position, amplitude and timestamp values. The whole design can be extended to work with more than four cameras by modifying the VHDL code. The software can be improved further by reducing the number of FIFO’s used in the project and consequently minimizing the hardware cost. Currently, the developed software only calibrates position, amplitude and time stamp of each camera and there is no synchronization between the cameras. There is further scope for improvement as the software can be modified to improve the synchronization of all the four cameras by providing an external clock from the Spartan 6 FPGA board. Since the current FPGA on the Spartan supports more camera link connectors, the hardware can additionally be improved by adding more cameralink connectors and GTP transceiver modules.
References


2. Ekström, C. THE WASA FACILITY AT THE CELSIUS STORAGE RING.


Appendix A: Abbreviations

ASIC           Application Specific Integrated Circuit
BRAM           Block Random Access Memory
CCD            Charge Coupled Device
CELSIUS        Cooling with ELectrons and Storing of Ions
CPLD           Complex Programmable Logic Devices
CR             Carriage Return
EEPROM         Electrically Erasable PROM
FIFO           First In First Out
FPGA           Field-Programmable Gate Array
FSM            Finite State Machine
GAL            Generic Array Logic
GSI            GSI Helmholtzzentrum für Schwerionenforschung GmbH
GTP            Giga-Bit Transceiver Protocol
GUI            Graphical User Interface
IP             Intellectual Property
ISE            Integrated Software Environment
JTAG           Joint Test Action Group
PAL            Programmable Array Logic
PANDA          anti-Proton ANnihilations at DArmstadt
PCB            Printed circuit Board
PLL            Phased locked Loop
PROM           Programmable Read-Only Memory
SERDES         Serializer/Deserializer
SFP            Small Form Factor Pluggable
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
<tr>
<td>UPTS</td>
<td>Uppsala Pellet Test Station</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
</tr>
<tr>
<td>WASA</td>
<td>Wide Angle Shower Apparatus</td>
</tr>
</tbody>
</table>