A multichannel digitizer for the PANDA experiment

Alvaro Serneguet Sorli

Institutionen för informationsteknologi
Department of Information Technology
The PANDA Experiment will be one of the future experiments at the Facility for Antiproton and Ion Research (FAIR), at Darmstadt, Germany. In the experiment, a stream of frozen hydrogen pellets will traverse vertically an antiproton beam circulating in the accelerator. Collisions of antiproton projectiles with frozen hydrogen pellets will lead to the generation of new particles, which in turn will be measured in the PANDA detector facility.

The aim of this thesis is to develop a readout firmware for an ADC (Analog to Digital Converter) prototype, which will be used in the PANDA detector to gather all the necessary information from the antiproton-proton collisions. The firmware takes out interesting parameters of the pulse produced by the collision like amplitude, integral, pedestal or arrival time. The work will include algorithms for pulse recognition, data buffering, data formatting and readout using a Gigabit Transceiver (GTP). Big efforts have been put to achieve a very efficient utilization of the resources and to minimize the latency.
Acknowledgement

I would like to express my gratitude to Dr. Pawel Marciniewski for giving me the opportunity to do my thesis about this amazing and huge project, PANDA experiment and for sharing his knowledge with me. I also appreciate the support and patience of Dr. Leif Gustafsson when he helped me to write the present report. Without them I could not have done my thesis, I am so grateful for their patience and dedication.
Thank you.
Contents

1. Introduction ........................................................................................................................................... 1
   1.1. PANDA experiment ......................................................................................................................... 1
   1.2. Tools ................................................................................................................................................ 3
   1.3. The signal digitizer (ADC) .............................................................................................................. 5
   1.4. Data acquisition system .................................................................................................................. 6

2. FPGA firmware implementation ........................................................................................................... 7
   2.1. FPGA design .................................................................................................................................... 7
   2.2. Deserializer ....................................................................................................................................... 8
       2.2.1. Automatic reset synchronization .............................................................................................. 8
   2.3. Pulse Detector Module ................................................................................................................... 10
       2.3.1. Average calculation ................................................................................................................... 11
       2.3.2. Storing the pedestal ................................................................................................................... 12
       2.3.3. Pulse detection ........................................................................................................................... 12
       2.3.4. Storing pulse and reading out the data ...................................................................................... 13
   2.4. FIFO ................................................................................................................................................. 14
   2.5. Data Merger ...................................................................................................................................... 14
   2.7. GTP transceiver ............................................................................................................................... 17
       2.7.1. GTP Transmitter ....................................................................................................................... 17
       2.7.2. GTP Receiver ............................................................................................................................ 18

3. Verification and testing ....................................................................................................................... 21

4. Limitations .............................................................................................................................................. 25

5. Conclusions and future work .............................................................................................................. 25

6. References .............................................................................................................................................. 28

7. Figures ................................................................................................................................................... 29
1. Introduction

1.1. PANDA experiment

The name PANDA stands for anti-Proton ANnihilations at DArmstadt. The PANDA Experiment is a facility for Antiproton and Ion Research (FAIR) 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 antiprotons in HESR collides with a fixed target (frozen hydrogen pellets) inside the PANDA detector. Investigations related to the strong force at medium energy are the primary focus of this experiment.

![Electromagnetic calorimeter (EMC)](image)

In the PANDA detector a high electromagnetic calorimeter (EMC) is required over a large energy range from a few MeV up to several GeV [2].

Figure 1 shows the barrel (left) and forward endcap part (right) of the EMC that gets information about the energy. The parts of the barrel calorimeter are cut to get a better view to the inside. Displayed are the PWO crystals, the support feet connecting to the support beam, held by the support rings at front and back.
Good identification and reconstruction of multi-photon and lepton-pair channels are of utmost importance for the success of the PANDA experiment. Low energy thresholds and good energy and spatial resolution are important assets to achieve high yield and good background rejection. Due to the high luminosity, fast response and radiation hardness are additional requirements that have to be fulfilled.

To achieve the required very low energy threshold, the light yield has to be maximized. For that reason the EMC consists of 15552 PbWO$_4$ crystals that are operated at -25ºC, which increases the light output by a factor of four.

The crystals produce short (20 ns) light pulses with intensity proportional to the energy deposited in them by projectile particles. The light pulses are then converted to electrical pulses by photo-sensors [3]. The electrical pulses in turn are amplified, shaped and delivered to Analog-to-Digital converters (ADC) followed by digital signal processing performed in Field Programmable Gate Arrays (FPGA). The FPGAs are used to parameterize signals and limit the data flow to the subsequent Data Acquisition (DAQ) System.

From the moment the experiment is triggerless, the event-building and data selection are performed on-line. All detector signals will need a continuous high-speed sampling and autonomous pulse detection and parameterization systems. The analog to digital conversion will require a dynamic range with an effective number of bits (EOB) reaching 14 corresponding to a 1MeV to 8GeV range of interest for the deposited energies. The ADCs will be located in a confined space within the detector, where the devices will perform the digitization and extraction of important features of signals, like pulse amplitudes, integrals and times to be sent to the Data Acquisition System (DAQ) over optical links (also called feature extraction)[4]. The ADC system will be exposed to ionizing radiation and a magnetic field of a flux density reaching 2 Tesla. The limited space provided in the detector for the ADC system poses high channel density and a necessity for liquid cooling of the devices.
1.2. Tools

The most useful tool for testing has been ChipScope [19]. It integrates a logic analyzer and measurement hardware components, making it possible to obtain and add values in real time in the FPGA using an ILA (Integrated Logic Analyzer) module [19].

Also in many situations to implement big functions it was necessary to split them in smaller functions and to simulate them in ISIM (ISE Simulator). It is obligatory to create a test file, and the problem creating this kind of file is that it needs to set every input.

Xilinx ISE Project Navigator has been used as main tool. It is an interface where it is possible to access all of the design entry and design implementation tools like the Core Generator [17]. The implementation in the FPGA has been written in VHDL (VHSIC Hardware Description Language). VHDL is a hardware description language used in electronic design automation to describe digital and mixed-signal systems, which can be implemented in FPGAs and other integrated circuits [15].

The ISE design flow comprises the following steps: design entry, design synthesis, design implementation, and Xilinx device programming (see figure 1.2). Design verification, which includes both functional verification and timing verification, takes place at different points during the design flow. This section describes what Xilinx ISE does during each step [14].

![Design flow overview](image-url)
Synthesis
The synthesizer converts HDL (VHDL/Verilog) code into a gate-level netlist. By default Xilinx ISE uses the built-in synthesizer XST (Xilinx Synthesis Technology). The Synthesis report contains very useful information. There is a maximum frequency estimate in the "timing summary" chapter. One should also pay attention to warnings since they can indicate hidden problems. After a successful synthesis one can run the "View RTL Schematic" task (RTL stands for register transfer level) to view a gate-level schematic produced by a synthesizer. The XST output is stored in NGC (Native Generic Constraints file) format.

Translation
The Translate process merges all the input netlists and design constraints and outputs a Xilinx native generic database (NGD) file (or EDIF netlist, depending on what synthesizer was used) that describes the logical design reduced to Xilinx primitives. The netlist produced by the NGDBUILD program contains some approximate information about switching delays.

Mapping
The Mapping process maps the logic defined by an NGD file into FPGA elements, such as CLBs and IOBs. The output design is a native circuit description (NCD) file that physically represents the design mapped to the components in the Xilinx FPGA. It contains precise information about switching delays, but no information about propagation delays (since the layout hasn’t been processed yet).

Placing and Routing
The Placing and Routing is the most important process and time-consuming step of the implementation. It defines how device resources are located and interconnected inside an FPGA. Placement is more important than routing because if the components are placed badly it will lead to bad routing. In order to provide a possibility for the FPGA designers to tweak placement, PAR (Place And Route) has many options such as Placer Effort Level, Starting Placer Cost Table and Timing Mode. The output of the PAR program is also stored in NCD format.

Timing Constrains
The Timing Constrains phase is the final check of the circuit. It checks the design to ensure that no timing violation (like period, setup or hold violation) will occur in the working design.

Device Programming
The Generate Programming File process produces a bitstream for Xilnx device configuration. After the design is completely routed, you must configure the device so it can execute the desired function.
1.3. The signal digitizer (ADC)

The ADC unit has 32-channel dual-range, symmetrizing active-filter amplifiers, 8 ADC with 80 Msps (each ADC has 8 channels with 14-bits per channel), 2 FPGAs Virtex-6 for signal processing and 2 optical link interfaces. (See figure 2). It was designed, manufactured and successfully tested at Uppsala University.

The signals are received by the 32 inputs and they are shaped and amplified. After that they go to the ADCs, in which they will be transformed from analog to digital signals. The digital signals will be received by the FPGA and there the code in the FPGA developed in this thesis will deseralize, analyze and extract the information that the signals contain. Finally the information already formatted will be transferred by optical links to a receiver.

The optical links are bi-directional and are used for data control, synchronization and configuration. The hardware structure features a partial dualism, improving the device robustness in case of communication failure over one of the optical links [5].
The dual-range configuration was chosen in order to facilitate high dynamic range. For the first prototype, gains of 1:1 and 16:1 (post-amplifier) were implemented. The resulting resolution was measured at KVI and the Bochum University, Germany with the full electronic track of the detector, including crystals, Vacuum Photo-tetrodes (VPTT) and pre-amplifiers reached 13.5 bit.

1.4. Data acquisition system

The board receive an electric pulse from the detector. After that the analog pulse is transformed by the ADCs in digital signals that can be processed by the FPGAs. The analog signal brings the information about the energy deposited in the crystals and the FPGAs will capture as many properties as possible. Moreover the signal will only be registered if it was received with another signal called trigger. The role of the trigger is basically to difference if the signal received was an interesting signal or just noise.
2. FPGA firmware implementation

2.1. FPGA design

The ADC interface receives the serialized data. Later the data in order to be deserialized go into a deserializer. N FIFOs and N Pulse Detector Modules receive the deserialized data, N is 8 (the number of channels). See figure 4.

A FIFO [9] per channel is required to store the pulse information. The FIFO is controlled by a Pulse Detector Module, which evaluates if there is a pulse in the signal. The Pulse Detector Module is also responsible for formatting the data. It needs to register the timestamp of the pulse using the reference time provided by the Clk-timestamp unit. It is a simple clock module running at the same frequency as the system (80MHz).

The information about the pulse is sent by optic fiber from the GTP transceiver, but it cannot be done directly because the frequency of the transceiver is different, only 50MHz. It is necessary to change the data frequency using a FIFO (GTP FIFO).

The GTP transceiver is a power-efficient transceiver, it is highly configurable and tightly integrated with the programmable logic resources of the FPGA, it has line rates from 600 Mb/s to 6.6 Gb/s [10]. The frequency used for data transmission is 50MHz * 40 bits, which is 2Gbit/s.
The ILA module contains a set of signals, which are used by ChipScope to debug. The task of the first ILA next to the VIO is to allow introducing values from ChipScope to the FPGA dynamically. It was necessary to create another ILA because the GTP works with a different frequency than it described above.

2.2. Deserializer

![Deserializer structure](image)

In the board there are 4 ADCs, in each ADC there are 8 channels, which have 14 bits each. The figure 5 shows the ADC0 that consist in two deserializers. Each of them receives a clock, a frame, 8 bits n and 8 bits p (because it receives an analog signal). The frame is set to "00001111" for aligning the incoming data. The frame is used like a pattern, when the deserializer receives correctly the frame it can be sure that the data will be aligned.

After 16 cycles at 80Mhz the deserializer transmits 64 bits to the output. The 64 bits are divided in four groups, channel 1, 4, 5 and 8 for the first deserializer and channel 2, 3, 6 and 7 for the second. Instead of send 16 bits per channel 14 bits are sent, this amount is due to the protocol used.

2.2.1. Automatic reset synchronization

When the board becomes warmer (55°C Celsius) there are synchronization problems. Those problems are due to the clock of the ADCs. When the temperature is high the clock signal does not arrive at the same time to the ADC components. The ADCs send
the data to the deserializer and the deserializer receives drifted data (see figures 14 and 15).

The first solution was simply to make an interface in Chip Scope to reset the ADCs one by one and go through all ADCs resetting several times until it works and is synchronized. In total there are 8 ADC (we have 4 per FPGA).

The second and smartest solution was to create a state machine (not shown) at the beginning to analyse the ADCs and check if they are synchronised or not. In case that there is one of them unsynchronized it will be reset until there will be no errors. The goal in the first state of the state machine is to get a median of the channel signal, taking 1000 samples. In next state a threshold is set to distinguish between two possible situations either it has found an unexpected pulse or it is not well synchronized and several wrong samples are coming.

![Figure 14 and 15: First picture shows unsynchronized channel and the second synchronized.](image)

The process to distinguish between a pulse and noise takes 25 samples and counts how many direction changes occurred and how many clock cycles the signal were over the threshold. If there was more than 10 samples over threshold and more than 5 direction changes we can assume the deserializer was not receiving any pulse. It was receiving an unsynchronized signal, the ADC will be reset until no more unsynchronized signals are found. To make sure the ADC was reset successfully it skips 1000 samples before to start to check the same channel again. In the same way it will check every channel and the channel will be reset if it is necessary.
2.3. Pulse Detector Module

To recognize the pulse a state machine was created (see figure 6), which also put together other values such as averages and timestamps.

Figure 6: Pulse Detector state machine
2.3.1. Average calculation

The state machine shown in the figure 6 starts at S0. Here it creates four averages as a reference point each one with size equal to the FIFO size, after this state it goes to S1 where the averages are compared, it is necessary to get three of them equal (not exactly equal but without significant differences) only then it can be assumed that the average is created correctly.

For example in the figure 7 there are four averages. When the algorithm compare them it will conclude that only avg0, Avg2 and Avg3 are equal therefore one of them is selected, for example Avg0. Avg1 was discarded cause it has different average provoked by noise.

![Figure 7: Signal averages](image)

In next states it will store the pedestal [18], which is a set of samples previous to the pulse; this number is dynamic (it is possible to modify it from ChipScope). The pedestal must be stored before receiving a trigger and the pulse.

One of the challenges was how to obtain the pedestal. The first logical thought is to make a FIFO to store the pedestal and another FIFO to store the data but it is possible to achieve that with just one FIFO as well.

The first thing was to create something called LIMIT in the FIFO, which is only a variable that we will use to delimit a zone of the buffer. We will be able to fill the FIFO until that LIMIT, only if the parameter DATA_COUNT of the FIFO is activated we can use the LIMIT variable. We also need to activate the parameters FULL and EMPTY to know when we have finished to write and read respectively. After the FIFO is filled with data until LIMIT we will set READ_ENABLE to always keep the same amount of samples in the FIFO. Then when the pulse arrives we will already have the PEDESTAL stored and we only have to unset the READ_ENABLE to allow continuing storing data.
2.3.2. Storing the pedestal

In state S2, if the trigger signal is received next state will be S8, which is a reset state, because the trigger signal comes always after the pulse and in this case that means that at least the beginning of the pulse was lost. If instead there is no trigger signal, it will compare the in-coming ADC data with the threshold. The threshold is a variable that indicate the limit of common noise, when the ADC data exceeds that limit it possible that the pulse is coming.

Figure 8 and 9: Threshold and average; pulse and trigger

2.3.3. Pulse detection

The task of state S3 is to check if there is a real pulse coming, the state must distinguish between a pulse and noise. The mechanism to know if there is a pulse is to count how many cycles the pulse is over the threshold, it is known that the pulse is always more than five clock cycles over it. Then if it is receiving a signal is less or equal to five cycles it can safely be assumed that it is not a pulse and it is a peak of noise. It will come back to the previous state S2 and reset the FIFO deleting stored data.

The number of cycles can easily be modified using a generic variable, and ChipScope can also provide it.
Figure 10: Differences between noise and pulse

Figure 10 shows that there is a big difference between a peak of noise and a pulse, and it is easy to see that a pulse has more time over threshold (TOT) than a peak of noise.

The status of the algorithm in this point, state S4, is next; in the FIFO the pedestal, and the pulse have been stored, but it is necessary to receive the acknowledgment, i.e. the trigger signal. If it does not appear in the next DIST_TRIGGER samples (Generic variable, number of samples permitted from the pulse to the trigger) the pulse will be qualified as SKIPED_PULSE and returns to S2.
Note that both S3 and S4 when going back to state S2 will reset the FIFO to start again storing pedestal. If a trigger signal is detected it will go to S5.

2.3.4. Storing pulse and reading out the data

In state S5 all samples are stored in the FIFO. Then when the FIFO is full it will go to S6 and wait until START_READ is 1. This signal is used to read out the pulse from the FIFO in state S7.
Finally when all the samples have been read out and the FIFO is empty it goes to the last state S8 to reset.

There are two kinds of reset; the reset that S8 performs is a simple reset that resets everything except the average and for this reason it can start from S2. The other reset is called Full_Reset, is triggered by ChipScope and this kind of reset resets everything including the average and goes to S0.
2.4. FIFO

The FIFO size has been set to 1024 based on the number of samples of the signal to be stored. The word width is 16 bits. The FIFO has been created using Core Generator.

2.5. Data Merger

The main task of the Data Merger Module is to serialize the information provided by Pulse Detector Modules.

An interesting feature of this algorithm is that it is able to find the next no empty channel without any extra latency (within a clock cycle) of this module. In figures 11a and 11b the green channels are not empty and the red channels are empty. In figure 11a before the first channel finishes the data transmission the system knows that the next channel with a pulse is the number 6. In figure 11b from the beginning the system knows that the first channel with a pulse is channel 5.

Also the communication with the Pulse Detector Module does not introduce any extra latency. As soon as the values are available the data are sent to the GTP-FIFO.

During the first state S0 in the Data Merger state machine shown in figure 12 the event number and the trigger timestamp are sent. At the same clock cycle the state machine checks which channel has a pulse. If there is not channel with a pulse the state machine goes to state S6, where a word counter is sent. When a channel is not empty the state machine goes to S1, where the channel number is sent. After that the state machine goes to S2 to send the pulse timestamp and after that to S3, where it sends the least significant part of the integral created by a Pulse Detector Module. The integral consists of 28 bits, which are enough to store it. In the
next state it sends the most significant part of the integral and in state S5 the raw samples data are sent.

The amount of raw samples data is always 1024 words, while the number of pedestal samples preceding pulse samples is a user parameter defined in ChipScope. In state S5 the raw samples data and the amplitude are sent. The amplitude is the maximum value of raw samples data minus the average.

The raw samples data are sent until the corresponding FIFO buffer is empty (Finish==1). The state machine will check if there are more pulses to send otherwise it changes to state S6 to send the word count, which is incremented after to send any word. After the word count is sent the state machine goes to state S7, which perform a reset and the state machine starts again with default values.

Figure 12: Data serializer and formatter state machine
2.6. Data format

The Data Merger Module sends the data in the structure shown below:

![Figure 13: Data format](image)

The TAG field consisting of 8 bits is used to describe the meaning of the data, which are transmitted, the next field consists of 14 bits for data, the next 8 bits contain control values such as checksum, and the two last bits indicate if there is any useful information in the data or control fields.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>F0</td>
</tr>
<tr>
<td></td>
<td>F1</td>
<td>F2</td>
</tr>
<tr>
<td></td>
<td>F3</td>
<td>F4</td>
</tr>
<tr>
<td></td>
<td>F5</td>
<td>F7</td>
</tr>
</tbody>
</table>

Table 1: Tags

The tag field starts with 0xF because other tags such as 0xE are already used in other projects and it was necessary to avoid problems and format conflicts. Anyway there are still many tags available it can also used 0xA and 0xB.

The transmission data and flow control is done as shown in table below.

| State | Word count | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |
|-------|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| S0    | 1          |    | F0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| S0    | 2          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | F2 |
| S1    | 3          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| S2    | 4          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| S3    | 5          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| S4    | 6          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| S5    | 7 to 1024  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| S5    | 1025       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| S6    | 1026       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Table 2: Word flow
The Integral is the sum of 1024 samples of the signal and the result reaches 20 bits. Therefore it needs two data words to get transferred.

Only words belonging to the Event Number, Trigger Timestamp and Word Count will be sent once. The words 3 to 1025 form a data block, which is sent for every non-empty channel.

2.7. GTP transceiver

To achieve a communication by an optical fiber a GTP transceiver is used [12]. A GTP transmitter serializes the data and sends them through an optical link to a optical receiver. Once the receiver gets the data, it deserializes it for further processing.

2.7.1. GTP Transmitter

In this chapter the configuration and the functional blocks inside a GTP transceiver transmitter (TX) are described. Each GTP transceiver includes an independent transmitter, which consists of a PCS (near end) and a PMA (far end). Figure 16 shows the functional blocks of the transmitter. Parallel data flows from the FPGA into the FPGA TX interface, through the PCS and PMA, and then out to the TX driver as high-speed serial data.

Figure 16: TX transceiver overview
**TX 8B/10B Encoder**
Many protocols use 8B/10B encoding on outgoing data like USB v3, PCI Express and Gigabit Ethernet. 8B/10B is an industry-standard encoding scheme that adds two bits of overhead per byte for improved performance \[20\]. The GTP transceiver includes an 8B/10B encoder to encode TX data without consuming FPGA resources. If encoding is not needed, the block can be disabled to minimize latency. In this project it is used because it ensures the correct function of the link by providing the following:
- A minimum transition density extract a clock from the data stream and synchronize with the receiver.
- A DC-balance in the signalling to reduce data baseline wander
- A bit error detection.

The 8B/10B encoding includes special characters (K characters) that are often used for control functions. This feature is utilized to send the COMMA character and to perform word alignment.

**Other modules**
The TX transceiver contains also other modules available, which are not implemented in this project.
The TX gearbox provides support for 64B/66B and 64B/67B header and payload combining.
TX Pattern Generator pseudo-random bit sequences (PRBS) that are commonly used to test the signal integrity of high-speed links. These sequences appear random but have specific properties that can be used to measure the quality of a link.
The digital oversampling circuit takes parallel data from the user interface at five times the desired line rate and replicates the data bit five times before advancing to next bit.

**2.7.2. GTP Receiver**
The block diagram for the GTP receiver is shown in figure 17. In the first stage the SIPO (serial in parallel out) converts the input High-speed serial data into parallel for FPGA processing. Later the Comma Detection and Alignment module reorganizes the data and indicates where the useful data start. Then the useful data flow through the 10b/8b decoding module and finally into the FPGA logic.
RX word alignment
Serial data must be aligned to symbol boundaries before they can be used as parallel data. To make alignment possible, transmitters send a recognizable sequence, usually called a comma. The receiver searches for the comma in the incoming data. When it finds a comma, it moves the comma to a byte boundary so the received parallel words match the transmitted parallel words.

Figure 18 shows the alignment to a 10-bit comma. The TX parallel data is on the left. The serial data with the comma is highlighted in the middle. The RX receiving unaligned bits are on the right.
**RX Loss-Of-Sync State Machine**

RX Loss-Of-Sync (LOS) is used in 8B/10B encoding to detect if the channel is malfunctioning. Each GTP receiver includes a LOS state machine that can be activated or deactivated.

**RX Elastic Buffer**

The GTP transceiver RX datapath has two internal parallel clock domains used in the PCS: the PMA parallel clock domain (XCLK) and the RXUSRCLK domain. XCLK is needed to keep the frequency of data with COMMA and coded by 10B/8B and RXUSRCLK to keep the frequency of the raw data. To receive data, the PMA parallel rate must be sufficiently close to the RXUSRCLK rate, and all phase differences between the two domains must be resolved. Figure 17 shows the two parallel clock domains, XCLK and RXUSRCLK. The GTP transceiver includes an RX elastic buffer to resolve differences between the PMACLK and RXUSRCLK domains.
3. Verification and testing

In the verification of this project the tools explained in the introduction (ChipScope, ISIM, waveform generator and oscilloscope) were used. There were different parts to verify such as pulse detector, trigger capture, GTP transceiver and data format. Two ways were used to verify the functionality of the pulse and trigger capture. The first ISIM simulator was used, in which test-benches are written in VHDL. A disadvantage of ISim is that it needs all the inputs contained in the test-bench whereas ChipScope can get real signals from the FPGA. The other way was to upload the code to the FPGA and run ChipScope Pro analyzer to show the raw samples data. At the same time to modify the signal properties such as amplitude, frequency, phase, delay, etc. with a waveform generator was very useful.

Figure 19: ISim simulation

Figure 19 shows a simulation of the Pulse Detector Module, data_count is the FIFO counter, actual_st is the current state and pedestal is the number of samples to store before the signal. At the beginning when the state machine is in state S2 data_count is 3 that corresponds to the pedestal and the parameters rd_en (read enable) and wr_en (write enable) are activated. They are activated to allow to the FIFO keep the pedestal and to update it in each clock cycle. When the signal rd_en of the FIFO is deactivated, the FIFO stops to keep the pedestal and continues storing the pulse. The output is 0 because until the state S7 the data does not go through the output. It is also possible to see the current time (signal current_time) belonging to Clk_Timestap Module.
A waveform generator and an oscilloscope were used to generate and monitor physical input and output signals, thus it was possible to check exactly what the FPGA was receiving. The dual channel wave generator was delivering a trigger signal and a pulse signal.

In order to test the hardware with more signals the channel that contains the pulse was copied and modified to obtain 4 test signal, coupled to other channels in the FPGA. The signals were modified in phase, amplitude and DC level as it is shown in figure 20.

![Figure 20: Waveform generator signal (blue), output (green), and Trigger (red).](image)

Figure 20 shows the input data of 8 channels (blue), the trigger signal (red) that is correlated with the first channel and the output signal of the Data Serializer and Formatter unit (green). The output signal is composed of 14 bits of the data field, the event number, timestamps and the integral. This is why the output signal has a complex shape.

To evaluate the capability of the Pulse Detector Module of linking the pulse with its trigger and evaluate the valid trigger time window (dist_Trigger, X in the figure 20) the trigger pulse delay was modified in the waveform generator.

The GTP was tested using 5 different loop modes, from the most internal loop in which the optic fiber wire is not necessary to the most external loop using a fiber connection to a ATLB optical transceiver module (see figure 21)[5]. Loopback modes are specialized configurations of the transceiver datapath where the traffic stream is folded back to the source. Typically, a specific traffic pattern is transmitted and then compared to check for errors.
Loopback test modes fall into two broad categories:

- Near-end loopback modes loop transmit data back in to the transceiver closest to the traffic generator.
- Far-end loopback modes loop received data back in to the transceiver at the far end of the link.

Loopback testing can be used either during development or in deployed equipment for fault isolation. The traffic patterns used can be either application traffic patterns or specialized pseudo-random bit sequences. Each GTP transceiver features several loopback modes to facilitate testing:

- Near-End PCS Loopback (path 1 in Figure 21)
  - Parallel, one clock domain.
- Near-End PMA Loopback (path 2 in Figure 21)
  - Serial, high-speed domain.
- Far-End PMA Loopback (path 3 in Figure 21)
  - Serial, high-speed and hardware receiver, 3 domains.
- Far-End PCS Loopback (path 4 in Figure 21)
  - Serial, 4 domains, full test.
- Near-End optic fiber wire (path 5 in Figure 21)
  - Serial, 2 domains optical fiber cable.
The picture 22 shows how to make a loop with the fifth path.

Figure 22: External loop

Figure 23 shows a data transfer from 5 non-empty channels over a fiber loop (loopback 5). When there is no valid data to transmit, idle characters (comma) are sent.

The comma character is 0x000000BC but in the waveform appears in the Control field as 0x2F because its least significant bits are truncated (valid_bits).

Figure 23: ChipScope loop
4. Limitations

ISIM Xilinx Simulator does not allow for a full verification of the design, making it necessary to test the code directly on the board. This implies going through each of the code compilation steps to upload the project to the FPGA. The computer used in this project had an I3 processor and the process to upload the code to the FPGA takes approximately 30 minutes. It limits the number of iterations of the project. One solution could be to use a computer with better performance.

The waveform generator has only two channels, which are used to transmit the trigger and the pulse signal. Only one channel was available for testing, therefore only a partial verification was possible.

5. Conclusions and future work

An efficient VHDL code for the Data Acquisition System prototype needed for the PANDA project has been developed and described in this thesis. Another prototype will be built for a PANDA project soon, but almost all the work developed in this thesis will be useful even there.

The system can be extended in some aspects. It can be modified to get more pulse properties and a more accurate filter like Wiener filter [13] for noise filtration (figure 24) can be developed.

![Figure 24: Example of a TES noisy signal (cyan), reference signal (green), SG (blue) and Wiener (red) filter's outputs.](image)
This project has achieved the initial goals of latency and resources minimization. As it was demonstrated in this paper and the code developed offers a huge range of possibilities.

A weak point of the Pulse Detector is that when the pedestal is stored and if the pulse does not come the algorithm resets the FIFO deleting the pedestal. This can be a problem if the pulse intervals are shorter than the pedestal size.

One solution to that problem can be to create other FIFO to store the pedestal but if we are looking for resources minimization other solution is keeping the FIFO full instead of reset it.

In figure 25 is shown this last solution. When the state machine of the Pulse Detection (figure 6) is in state S2 and comes from S3 or S4 the FIFO will be full but with WR and RD set to 1. In that way the pedestal can be stored (state 1 in figure 25). When the state machine detects a pulse, the FIFO will store size_of_FIFO minus pedestal (1024 - 4) samples then of that pulse therefore we will keep the pedestal and store the pulse (state 2, 3 and 4 in figure 25).

The next prototype consists of 2 Kintex-7 instead of 2 Virtex-6 FPGAs. The new FPGAs are cheaper and less power consuming. This prototype is been evaluated by my supervisor Dr. Pawel Marciniewski has achieved a gain of almost 30% in energy saving. The input connections type have also been changed to a Direct Cable Connection ERCD, which provide a comfortable and easy installation of the board in crates at the EMC frame (see figure 26). 22 ADC modules of the thickness of 15-20mm will be housed in each crate. Other tasks for the coming years are to build the structure of encapsulation and cooling of the ADC modules. The total amount of ADC modules will be approximately 1000 pieces.
The present prototype will be sent to The Svedberg Lab, Uppsala, Sweden to test the radiation resistance. The tests will be with high radiation of at least $10^5$-$10^6$ protons/s/cm$^2$ and a beam energy of 150 MeV even more if it pass the tests [16]. Next year 2015, it is expected that the first production batch will be ordered with more than 1000 ADC modules.
6. References

1. PANDA collaboration, http://www-panda.gsi.de
3. Karoly Makonyi, PWO readout with VPTs, Presentation at Stockholm University (2012)
16. Test Radiation of the Trigger-less Readout of the PANDA EMC – Presentation at KVI, University of Groningen http://indico.gsi.de/getFile.py/access?contribId=9&resId=0&materialId=slides&confId=2030
7. Figures

Figure 1.1: Electromagnetic calorimeter (EMC) .........................................................1
Figure 1.2: Design flow overview .............................................................................3
Figure 2: ADC hardware structure ...........................................................................5
Figure 3: The prototype ..............................................................................................6
Figure 4: Main scheme ...............................................................................................7
Figure 5: Deserializer structure .................................................................................8
Figure 14 and 15: First picture shows unsynchronized channel and the second
synchronized. ............................................................................................................9
Figure 6: Pulse Detector state machine ....................................................................10
Figure 7: Signal averages ..........................................................................................11
Figure 8 and 9: Threshold and average; pulse and trigger ........................................12
Figure 10: Differences between noise and pulse ....................................................13
Figure 11: Skipping channels ....................................................................................14
Figure 12: Data Merger state machine .....................................................................15
Figure 13: Data format ................................................................................................16
Table 1: Tags ................................................................................................................16
Table 2: Word flow ......................................................................................................16
Figure 16: TX transceiver overview .........................................................................17
Figure 17: RX transceiver overview .........................................................................19
Figure 18: Block alignment .......................................................................................19
Figure 19: ISim simulation .........................................................................................21
Figure 20: Waveform generator signal (blue), output (green), and Trigger (red) ....22
Figure 21: Loopback levels .......................................................................................23
Figure 22: External loop .............................................................................................24
Figure 23: ChipScope loop .......................................................................................24
Figure 24: Example of a TES noisy signal (cyan), reference signal (green), SG (blue)
and Wiener (red) filter’s outputs. ............................................................................25
Figure 25: FIFO transitions of the pedestal solution ...............................................26
Figure 26: Crates ........................................................................................................27