CRU Readout Simulation for the Upgrade of the ALICE Time Projection Chamber at CERN

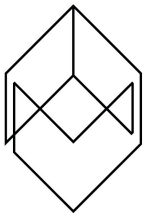
Damian K. Wejnerowski

Master's thesis in Software Engineering at

Department of Computing, Mathematics and Physics,
Bergen University College

Department of Informatics,
University of Bergen

June 2015



HØGSKOLEN
I BERGEN

Abstract

ALICE Collaboration at CERN is planning major upgrade of the Time Projection Chamber Detector to cope with an increase of frequency and energy of particle collisions after 2020. The first phase of this thesis involves contributing to the design of new hardware to be installed on the detector by developing a computer model of the readout electronics.

The important part of the readout electronics is CRU – Common Readout Unit module, which controls the readout process, and prepares data before sending them out to further parts of the readout chain.

The computer simulation was used to compare performance of several proposed implementations of CRU. Data readout was simulated with different input scenarios to estimate buffer size required by CRU. Obtained results present an important input for further improvement of the detector performance.

Acknowledgements

This thesis would not have happened without help and generosity of many people.

I would like to express my gratitude to my academic supervisor Prof. Håvard Helstrup for prompt and comprehensive feedback all along the process, and for tirelessly reading and commenting drafts of my thesis. I consider myself privileged to have had a supervisor with such breadth of experience and knowledge, who yet remains as friendly, reliable and available as you have been at all times.

Johan Alme, who has been my technical supervisor for the duration of this project. I am deeply grateful to him for the long discussions that helped me sort out the technical details of my work. You have always been helpful and patient and your knowledge of both electronics and physics, continues to impress me. Your work has been an important source of learning and inspiration.

Prof. Dr. Dieter Röhrich, for hosting me at the Department of Physics and Technology, and for giving me the possibility to partake in this interesting experiment and the valuable assistance he has given me while writing this thesis.

I am also grateful to Dr. Christian Lippmann for guidance, generous contribution of knowledge and experience, valuable comments and assisting me with the tasks I have performed for ALICE.

My gratitude also goes to Arild Velure, a PhD student from the University of Bergen, who was a great source of reliable informations related to SAMPA chip, and helped me to understand numerous concepts from the hardware world.

Special thanks to Bjarte Kileng and Pål Ellingsen for access to several powerful computers which speeded up running of simulation and made possible to test computer model of readout electronics in various conditions.

Furthermore, I would like to thank all my friends who have contributed in various ways, both with technical issues and social activities, for making my study time such a rewarding experience.

Last, but not at all least, I would like to thank my parents and my girlfriend who have provided invaluable love, support and encouragement all this time.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

A Large Ion Collider Experiment (ALICE) hosted at CERN, the European Laboratory for Nuclear Research, is devoted to analyse the outcome produced in collisions between heavy ions. Heavy-ion collisions make it possible to achieve pressure and temperature conditions corresponding to the origin of the universe shortly after the Big Bang.

Measuring the results of these collisions requires a large detector. The ALICE experiment is one of the four main experiments on the Large Hadron Collider (LHC) ring. It is specially designed to study the physics of strongly interacting matter at extreme energy densities, where a phase of matter called quark-gluon plasma forms. The ALICE detector is essentially composed of a variety of subdetectors that measure the different properties of the collision. One of these subdetectors is a Time Projection Chamber (TPC) detector. The TPC detector is a 5 metre diameter cylindrical chamber consisting of a gas-filled detection volume in an electric field with almost 600 000 channels distributed over two endplates. To read out data from each of these channels, sophisticated readout electronics are required. Currently FPGA based systems with embedded Linux are used to monitor and control the read out of data. This system is called a Readout Control Unit (RCU), and TPC detector has a total of 216 such cards.

After a successful run in 2009 - 2013, the Large Hadron Collider (LHC) at CERN has been shut down for maintenance and upgrades. Physicists and engineers have used this break to carry one of the most sophisticated experiments in history even further. The collider has been upgraded to increase its collision energy and frequency. In 2018 the LHC will be shut down again for further upgrades. To handle the increased data rate that is expected by 2020, the TPC detector system will be completely rebuilt. The

upgraded version of RCU will be replaced with the new Common Read-out Unit (CRU) which is based on a completely new FPGA. The system will be running embedded Linux, and a large design work is necessary to implement the new drivers and new applications on this platform. Such a system must be tested and verified thoroughly before being installed.

1.2 The Assignment

Designing and producing a new electronics for data acquisition system which will be installed under the second Long Shutdown is a time consuming and challenging process. It is almost impossible to produce hardware first and test it after that. This approach would be a recurrent process which would consume enormous economical and time resources.

To help to design electronics needed to upgrade the detector and as the most important part of this assignment - the computer simulation of the readout electronics has been developed. Each module of the simulated model has been implemented in detail and described in this report. The purpose of any modules of the hardware was defined before writing the master thesis but the design of some parts of the system was not given. Therefore a significant part of the work on master thesis was to propose the implementation of those modules. Every suggested implementation of any module was tested using simulation to check if it meets the requirements regarding to real time performance and quality.

1.3 Report Outline

- Chapter 2 will be a thorough presentation of background theory and will give introduction to the scientific context of the project by presenting the European Organization for Nuclear Research - CERN, most important experiments hosted at CERN and it will explain why such a huge detectors are needed to study the smallest building blocks of matter.
- Chapter 3 introduces problem description by describing ALICE Experiment, TPC detector and explaining what is the purpose of upgrading of the readout electronics installed on the detector and why it is required. The new electronics which will be used after the Long Shutdown 2 is also presented in this chapter.
- Chapter 4 introduces the methods, tools and techniques used to find the best solution to the given problem.

- Chapter 5 describes the structure, design and implementation of the system. This chapter will contain also evaluation of the implementation, its limitations and all assumptions which were made.
- Chapter 6 introduces and explains results generated using the computer simulation.
- Chapter 7 presents an evaluation and conclude the outcome of the work with this report.

Chapter 2

Scientific Context

The general purpose of the Large Hadron Collider is to improve our understanding of the Universe. Despite the great progress in various fields of science, many basic phenomena surrounding us remain unexplained. This chapter discusses what scientists from CERN are looking for and why it is so significant.

2.1 CERN

CERN - the European Organization for Nuclear Research is the world's largest physics laboratory established in 1954 as a joint venture between 12 European countries [1]. Meyrin site close to Geneva in Switzerland was selected to host the CERN Laboratory.

CERN from its beginning has been dedicated to study the fundamental structure of the Universe. It has been done by exploring the basic constituents of matter - the fundamental particles. Investigating the smallest building blocks and the fundamental laws of nature demands large and sophisticated scientific instruments like particle accelerators and detectors.

2.2 LHC

The Large Hadron Collider is located at CERN in Geneve. It is the world's largest and most powerful particle accelerator. It is also the most complicated device constructed by humans. The LHC is a kind of microscope that allows to explore the world in a very small scales. In the LHC two particle beams moving in opposite direction are accelerated close to the speed of light before they are made to collide with each other. An accelerator

can only accelerate certain kinds of particles. The LHC accelerates particles which fulfill the following two requirements:

1. The particle must be charged.
Electromagnetic devices used by the LHC to manipulate beams can have impact only on charged particles.

2. The particle need not to decay.

The requirements listed above limit the number of particles that can practically be accelerated to electrons, protons, ions, and all their antiparticles.

The LHC is not an independent construction. It depends on so called accelerating structure, as shown on figure 2.1, which gradually accelerates particles to achieve higher energy. For proton runs acceleration starts with hydrogen, whose atoms are composed of one proton and one electron. These atoms every few hours are taken from a small cylinder and ionized, or robbed of electrons.

Then, the obtained protons are directed to a linear accelerator, Linac 2, where it accelerates until about 30% of the speed of light. Then they go to the PS Booster accelerator and here their kinetic energy is increased almost 30-fold. From the Booster protons are transferred to the Proton Synchrotron PS, and then to the Super Proton SPS, at every stage of increasing energy approximately 20 times. Less than five minutes after leaving the cylinder protons finally get inside the Large Hadron Collider tunnel [2].

The acceleration of lead nuclei is slightly different then in the case of protons, however the final stages of their road to LHC run through the PS and SPS accelerators. Every day during the run at the LHC accelerates just a few nanograms of hydrogen [3].

2.2.1 Operational Specifications

The LHC reused the tunnel from previous LEP accelerator, dismantled in 2000. As a result of geological consideration and cost it has been decided that excavation of a tunnel to house a 27-km circumference machine was much cheaper rather then acquiring the land to build at the surface [5]. In addition the impact on the landscape was reduced to a minimum. The mean depth of the tunnel is 100 m, and its real depth varies between 175 m under the Jura and 50 m towards Lake Geneva.

The 27-kilometre accelerator ring consists of two adjacent parallel beam pipes surrounded by electromagnets. Both the pipes and magnets are kept in vacuum and are isolated by thermal shield.

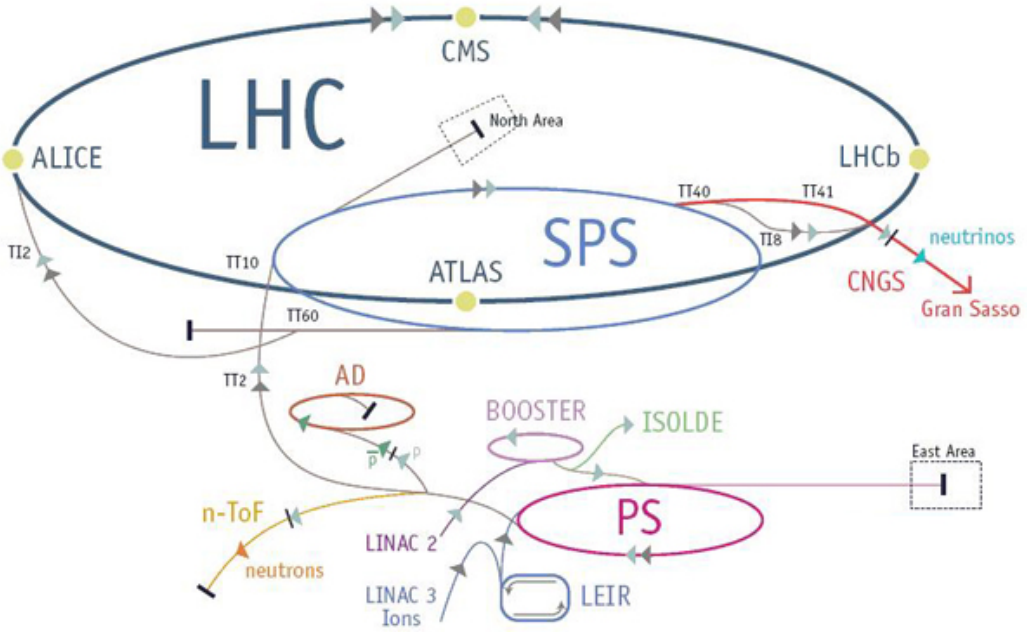


FIGURE 2.1: The accelerator complex is composed of a series of accelerators which work together to push particles to nearly the speed of light. [4]

The electromagnets are used to maintain a strong magnetic field which guides the beams around the accelerator ring. The beams travel in opposite directions in each pipe and just prior to collision, they are bend by the another type of magnet that focus the particle beams to increase the chances of collision.

The electromagnets are built from coils of special electric cable that operates in a superconducting state which make them able to conduct electricity efficiently without resistance or loss of energy. To achieve this state magnets must be chilled to 1.9 K (-271.3 °C) – a temperature colder than outer space [5].

The liquid helium was chosen as a cooling medium because of its physical properties. Liquid helium boils at -268.93 °C and does not freeze at atmospheric pressure [6]. In addition in temperature below -271 °C liquid helium passes from the fluid to the superfluid state which provides very high thermal conductivity [7].

The maximum energy obtainable for an accelerator is related to its size. In case of a collider of a ring shape, energy is a function of the radius of the ring and the strength of the dipole magnetic field that keeps particles in their orbits. The LHC is designed to collide beams of protons at a total energy of 14 TeV (teraelectronvolt) per collision and beams of heavy ions at a total energy of 1150 TeV per collision [9]. Such collision energy has never been reached before in a lab [4].

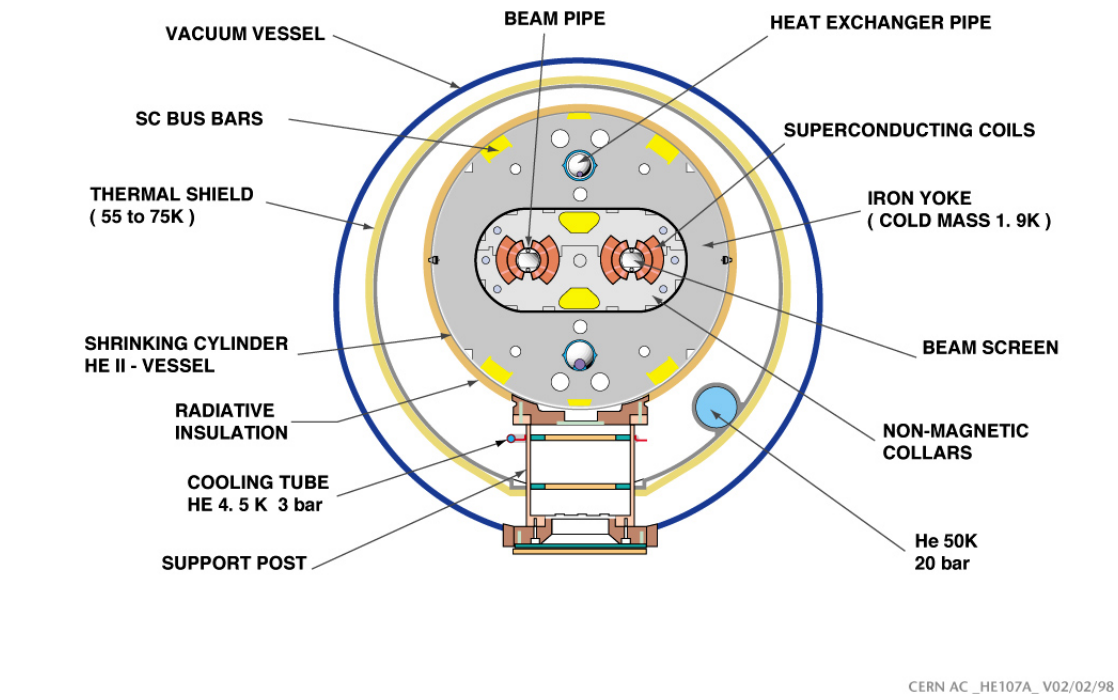


FIGURE 2.2: Cross section of LHC dipole. [8]

2.2.2 Experiments Installed at the LHC

The purpose of the detectors installed at Large Hadron Collider is to measure various properties of each collision. Data from detectors enable physicists to characterize all the different particles that were produced in collision. Some of them can be observed directly, some are short-lived and have to be reconstructed from their decay products and some particles like neutrinos escape without leaving any trace.

ALICE

ALICE, the acronym for A Large Ion Collider Experiment is an experiment specialized in analysing heavy-ion (Pb-Pb nuclei) collisions.

The resulting temperature and energy density of the collisions are expected to be high enough to produce quark–gluon plasma, a state of matter wherein quarks and gluons are no longer confined inside hadrons. Such a state of matter probably existed just after the Big Bang, before particles such as protons and neutrons were formed. The quark-gluon plasma has been described more detailed later in this section.

The ALICE Experiment is going to search for the answers to fundamental questions, using the extraordinary tools provided by the LHC:

- What happens to matter when it is heated to 100,000 times the temperature at the centre of the Sun?
- Why do protons and neutrons weigh 100 times more than the quarks they are made of?
- Can the quarks inside the protons and neutrons be freed?

The ALICE experiment involves an international collaboration of more than 1500 physicists, engineers and technicians, including around 350 graduate students, from 154 physics institutes in 37 countries across the world. The experiment continuously took data during the first physics campaign of the machine from fall 2009 until early 2013 [10].

Quark-Gluon Plasma

The elementary particles create a whole picture of the world we know today. Quarks, bound by gluons, group together to form hadrons. The example of the most common stable hadrons are protons and neutrons. They group together to form atomic nuclei. These atomic nuclei attract electrons, and they group to form atoms. Atoms form together molecular structures and shape the matter.

However, not always everything is and was so structured and ordered like it is described above. Physicists think that in the special conditions like extreme high temperatures and densities the fundamental particles were not formed but constituents were free to roam on their own.

The state of this matter is called a quark-gluon plasma. It is like a hot, dense soup which is made of all kinds of particles, mainly quarks and gluons, moving at near light speed [11].

According to the Big Bang theory, shortly after the creation of the Universe - Big Bang - the Universe was filled with mentioned quark-gluon plasma. Therefore by recreating the conditions like they were just after the Big Bang, physicists can go back in time and study the origin of the world.

ALICE is described more detailed in section 3.1.

The other experiments installed on LHC are: ATLAS, CMS, LHCb, LHCf, MOEDAL and TOTEM.

Chapter 3

Problem Description

This chapter introduces ALICE Experiment and TPC detector. It also presents a model of the upgraded readout electronics which is a part of the data acquisition system for ALICE TPC.

3.1 The ALICE Experiment at LHC

ALICE (A Large Ion Collider Experiment) is a general purpose, heavy-ion detector placed at one of the four collision points of the Large Hadron Collider. ALICE is optimized to study collisions between lead nuclei but, as a general purpose detector, it can collect data from proton-proton collisions as well [12]. High-energy nuclear collisions allow to reach high enough temperature and energy density to produce quark-gluon plasma (QGP), a state of matter where quarks and gluons are no longer kept inside hadrons but they can roam freely. Physicists believe that shortly after the Big Bang, from picoseconds to about 10 microseconds, the fundamental particles were not formed yet and the Universe was in state of QGP.

The QGP is required to study the physics of strongly interacting matter. The strong interaction is the force responsible for binding quarks together in protons and neutrons, and then protons and neutrons together in the atomic nucleus.

3.1.1 Layout and Technical Solutions

The energy of collisions in the center of the ALICE detector achieves 7000 GeV per proton, and 14000 GeV in the center of the mass of pair of two protons for the current

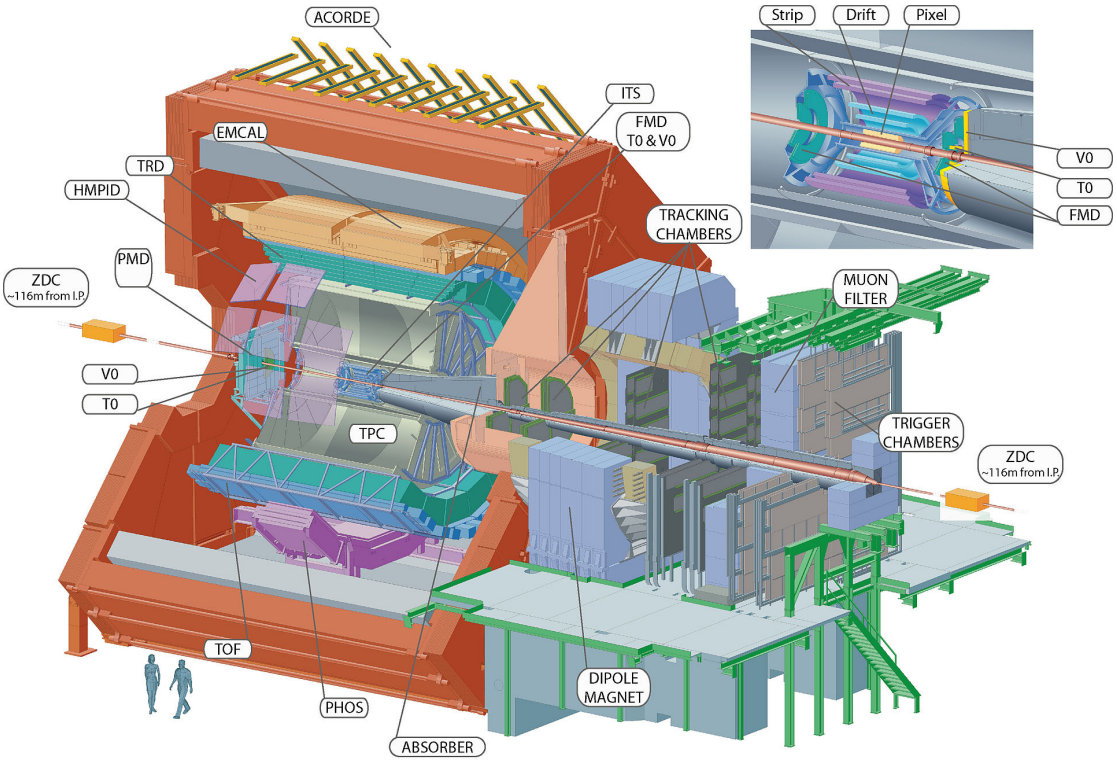


FIGURE 3.1: Computer generated cut-away view of ALICE showing the 18 detectors of the experiment. [13]

run - RUN 2. In case of the heavy-ions, the energy reaches 5500 GeV per pair of the nucleus [4].

The ALICE detector is 26 m long, 16 m high and 16 m wide. Its weight is approximately 10000 tonnes. The data acquisition system of ALICE requires data bandwidth of up to 2.5 GByte/s to record and select the steady stream of events resulting from central collisions. It results in storage consumptions of up to 1.25 GByte/s in real time when beams are collided in the accelerator.

The ALICE experiment consists of 18 different sub-detectors and their associated systems for detector control, trigger, data acquisitions, cooling, gas and power supply. Each of the detectors is characterized by its own specific technology choices and design developed to meet physics requirements and to works with experimental conditions expected at LHC [14].

The central barrel part of ALICE is placed inside a 7800 t solenoid magnet which was used under the L3 experiment at LEP. The basic purpose of the central part of the detector is to measure hadrons, electrons, photons and muons. This function is realized by sub-systems like: Inner Tracking System (ITS), a six-layer, silicon vertex detector,

and the Time Projection Chamber (TPC). To improve resolution of measurement at high momentum, the Transition Radiation Detector (TRD) is also used to track particles.

The other sub-detectors installed inside ALICE are: Inner Tracking System detectors (ITS), three particles identification arrays of Time-of-Flight detector (TOF), Ring Imaging Cherenkov (HMPID) and two electromagnetic calorimeters (PHOS and EMCal).

3.2 TPC

The Time Projection Chamber (TPC) is the main detector dedicated to perform tracking and identification of charged particles in the ALICE experiment. The ALICE TPC is composed of a cylindrical gas-filled detection volume divided in two half parts of equal size by a central high voltage electrode. The inner and the outer radius of the detector is respectively about 80 and 280 cm, with an overall length of 500 cm in the beam direction.

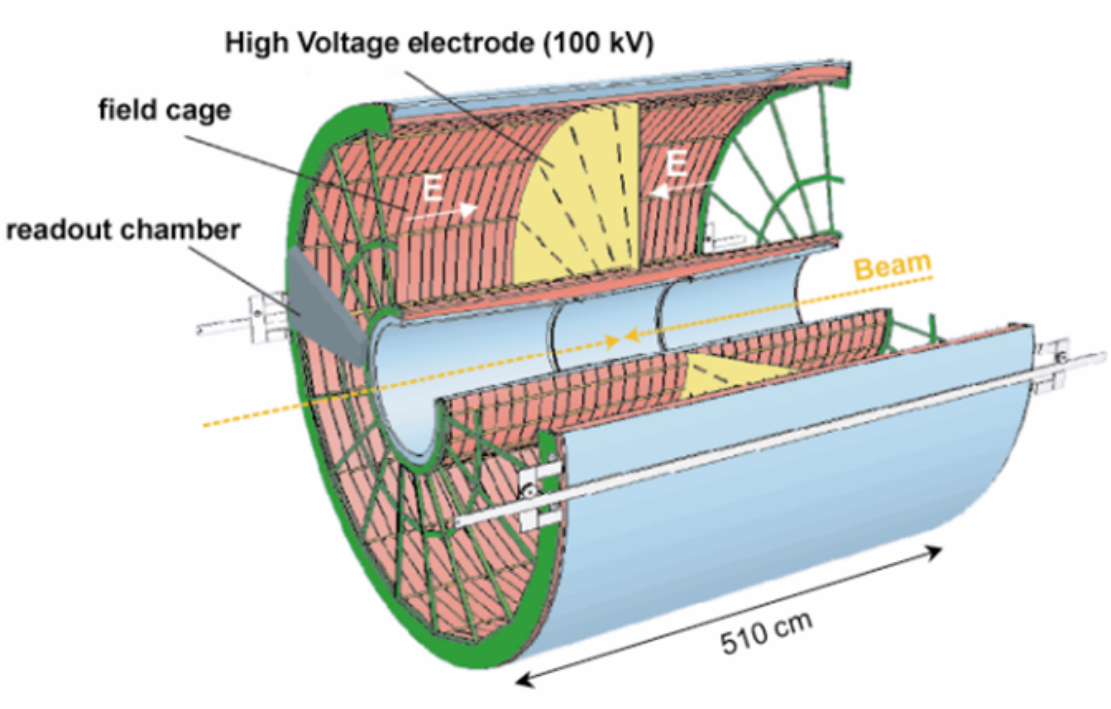


FIGURE 3.2: Model of the ALICE Time Projection Chamber detector. [15]

The basic principle of the ALICE TPC is as follows: once a charged particle is passing through the gas volume of the detector, it ionizes the gas liberating electrons along its passage. Depending on the momentum and identity of the particle, the density of the ionization will be weaker or stronger. Created electrons drift, under the influence of the

electric field, towards the end plates of the cylinder, ions drift to the central cathode. The arrival point to the endplates of the cylinder is precisely measured by multi-wire proportional chambers (MWPC) with wire planes and 560 000 electronics channels. Together with an accurate measurement of the arrival time the complete 3-dimensional trajectory of the charged particles traversing the ALICE TPC can be determined with high precision.

The cylindrical chamber is filled with gas mixture of 90% Ne and 10% CO_2. After extensive study, it was decided to use this mixture of gas under RUN 2 because the mixture was proved to be an optimal solution which provides low diffusion and large ion mobility [16]. There are still other solutions under consideration which can be used under RUN 3. One of the proposed solution assumes adding N_2 to the mixture which should provide higher gas gain stability and better control of the fraction and influence on the drift velocity.

3.2.1 Data Acquisition

The present ALICE TPC readout is based on Multi-Wire Proportional Chamber (MWPC). As mentioned in section 3.2 the charged particle traversing the detection volume ionizes the gas. It results in creating electrons that drift towards the readout end-plates due to an electric field applied by a high voltage (100 kV) electrode at the middle of the ALICE TPC. After a few microseconds the drifting electrons reach the Multi-Wire Proportional Chambers where they are amplified. These MWPCs are arranged with a plane of anode wires in the azimuthal direction, a plane of cathode wires and a plane of gating wires which prevent the ions from drifting back to the volume of the TPC detector. The pad plane is located under the named layers of wires, and it is responsible for reading the actual signal.

During the Long Shutdown 2 the Multi-Wire Proportional Chambers will be replaced with the GEM - Gas Electron Multiplier foils [17]. The GEM and the readout electronics which will be used under RUN 3 is introduced in the next section 3.3 dedicated to the upgrade of the ALICE TPC for RUN 3.

3.3 Upgrade for the RUN 3

3.3.1 Physics Background for the Long Shutdown 2

The life cycle of the Large Hadron Collider consists of runs, when the charged particles are colliding and data are collected, and long shutdowns - periods when there are no beams in accelerator and the devices installed at LHC are maintained and upgraded. Every detector during each long shutdown is upgraded to manage to operate in the new environment of LHC which introduces higher collision rate together with higher energy of each collision.

Since 2009, the LHC has successfully provided collisions to the four large experiments mentioned in previous chapters: ALICE, ATLAS, CMS and LHCb. After the RUN 1, the first long shutdown started in February 2013 and during the next 18 months many improvements were performed. The energy of pp collisions was increased to 13-14 TeV, it is significantly higher compared to the 7-8 TeV for the RUN 1 in 2012, and the luminosity for Pb-Pb collisions was increased to 1 - 4 x 10^{27} $cm^{-2}s^{-1}$ [18], it results in reaching an interaction rate of about 30 kHz for Pb beams under RUN 2.

The scope of this master thesis is to contribute to the upgrade of ALICE TPC which is planned to be performed under the second long shutdown. The Long Shutdown 2 is expected to start in July 2018 and it is planned to take 18 - 21 months. In this period of time, the LHC will be prepared to handle full luminosity of about L = 2 x 10^{34} $cm^{-2}s^{-1}$, for ALICE the instantaneous luminosity will reach L = 6 x 10^{27} $cm^{-2}s^{-1}$ which results in interaction rate of about 50 kHz for Pb-Pb collisions [19].

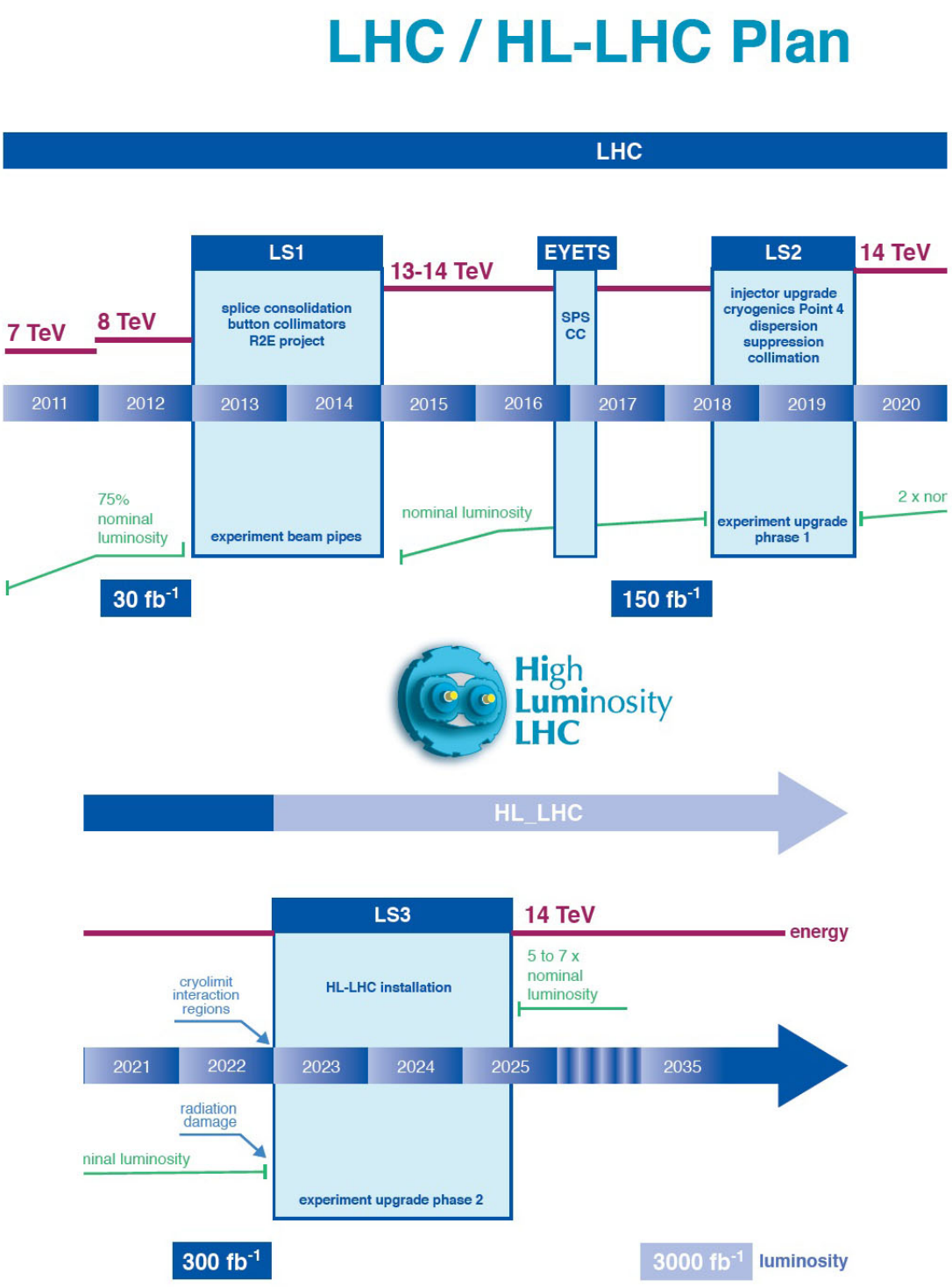


FIGURE 3.3: Timeline for LHC including the first three long shutdown periods. [20]

The Long Shutdown 2, opposite to the first Long Shutdown during which the read-out electronics were upgraded, will introduce completely new hardware, designed from

scratch. All electronics will be removed from the detector to make place for new, faster and more efficient solutions.

3.3.2 Gas Electron Multiplier - New Readout Technology for RUN 3

The Multi-Wire Proportional Chambers (MWPC), mentioned in 3.2.1, are the first part of the readout system. The MWPCs are responsible for reading and amplifying signals coming from the TPC detector. Unfortunately, the MWPC technology limits the sampling frequency which makes it impossible to record data with collision rate of 50 kHz which is planned for the Run 3.

To overcome this limitation, Multi-Wire Proportional Chambers will be replaced by Gas Electron Multiplier (GEM) planes. The GEM is essentially Kapton foils placed between two layers of copper and perforated through each layer with holes of approximately size of 40 - 70 um diameter and 140 um pitch [21]. Between two layers of copper, a voltage of 150 - 400 V is generated, to make large electric fields in the holes.

The electrons created by ionizing particles which are traversing the detection volume of the TPC detector, drift towards the end plates. It is enough that just one single electron enters any hole to create en avalanche containing hundreds of electrons. The resulting electrons infiltrate a cascaded structure of several GEM foils where they are amplified, collected and transferred to the readout electronics, placed just a few centimetres away, via flexible Kapton cables.

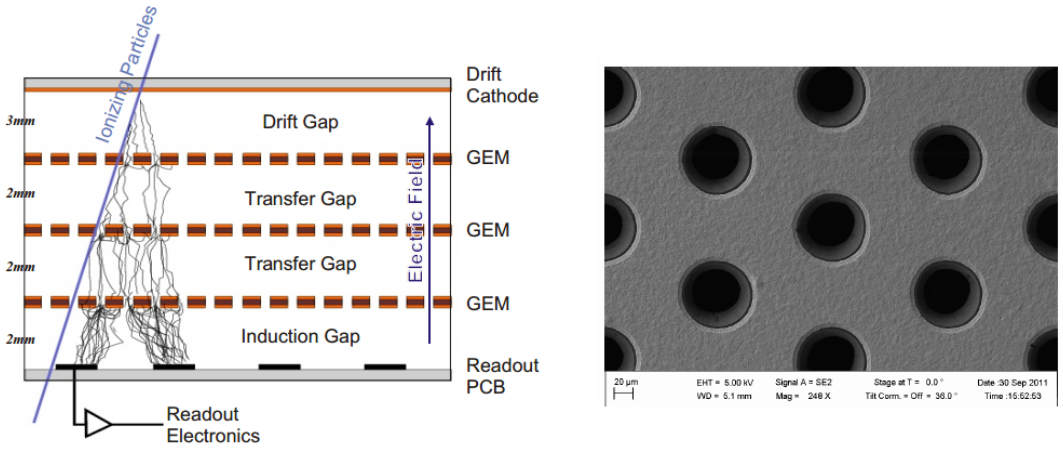


FIGURE 3.4: On the left: structure of GEM foils and avalanche of electrons, on the right: electron microscope photography of a GEM foil with hole pitch 140 um. [22]

3.3.3 Front-end Readout Electronics for ALICE TPC for RUN 3

Replacement of Multi-Wire Proportional Chambers by Gas Electron Multipliers introduces new challenges for front-end electronics. The new readout chain must be redesigned to comply to the new requirements related to continuous readout, opposite to trigged readout mode used in RUN 1 and RUN 2, which results in strongly increased data throughput.

The data collected from the GEMs are processed and digitalized by the Front-End Cards (FECs) installed on the detector. Each FEC consists of 5 SAMPA chips and 2 GBTx modules sending data to the Common Readout Unit (CRU). The Common Readout Unit is responsible for monitoring and controlling front-end electronics and it sorts the collected data and forwards them to the Online Farm System for further processing. The schematic picture of the TPC readout electronics is shown in the Figure 3.5.

Reading data from over 500000 channels of TPC detector requires over 280 CRU units which monitor and receive data from 3400 FECs equipped with 17000 SAMPA chips and 6800 GBTx [23]. However, different solutions for the CRU are still under consideration. Some of the solutions consider CRU which reads data from 12 FECs, other solution supports version of CRU which can serve 16 FECs. More information about the different designs of CRU can be found in section 3.3.7.

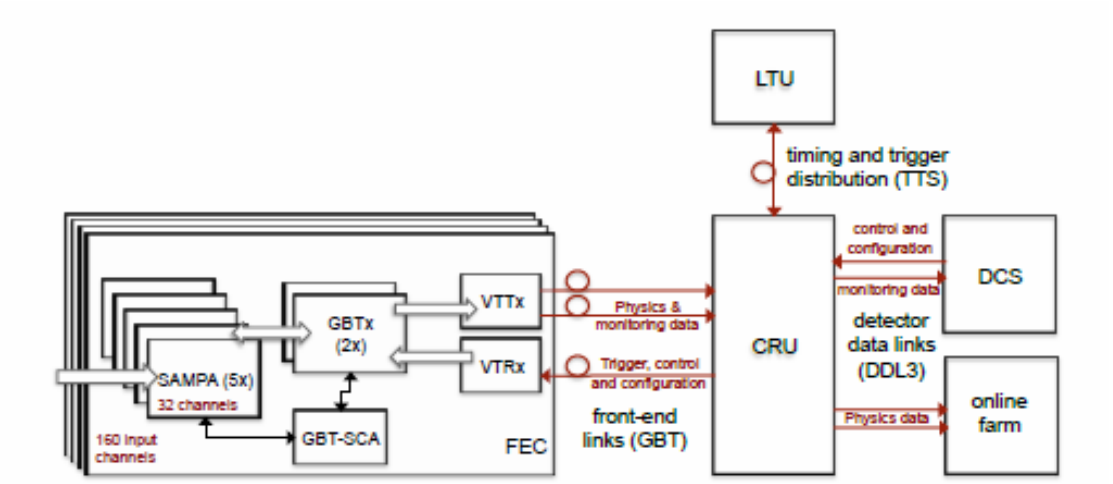


FIGURE 3.5: Schematic of the TPC readout electronics with the Front-End Card as a front-end electronics and CRU as central part. [24]

3.3.4 Front-End Cards

The replacement of the existing MWPC-based readout chambers by GEMs during the second Long Shoutdown involves also the necessity for new readout electronics that not only enable continuous data readout but also accommodate the opposite signal polarity [24]. The signals retrieved from the pads of end-plates are read by the front-end electronics installed in the form of Front-End Cards (FECs) in front of the end-plates. Front-End Cards are placed in a special frame, called Service Support Wheel (SSW), which support them mechanically, provides power supply and cooling facilities. FECs are connected to the pads using flexible Kapton cables mentioned in the section 3.3.2.

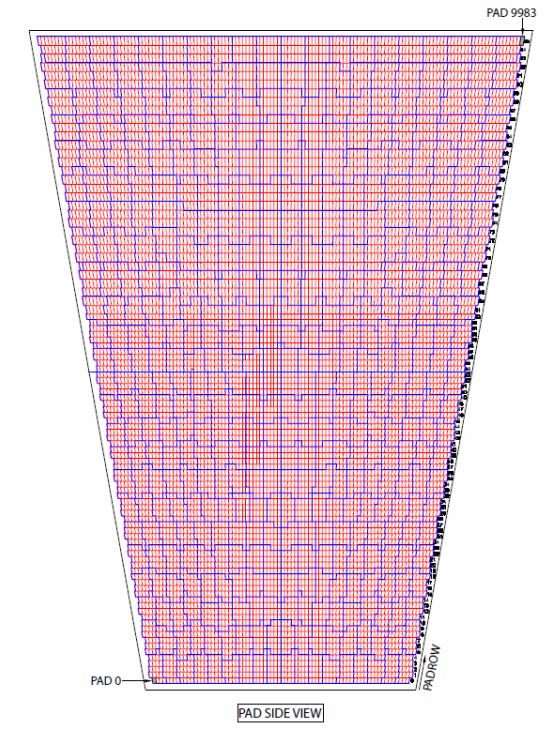


FIGURE 3.6: One sector of readout plate divided into pads and partitions connected to FECs. [25]

The Front-End Electronics are responsible for reading signals from 552960 channels [24] of the TPC detector. The electronics have to handle the sampling rate of 10 MHz, what together with the size of each sample equals to 10 bit and occupancy of pad around 15-27% results in data amount of about 1 TByte/s [24] for the TPC detector.

The front-end electronics distribute this amount of data through Front-End Cards. Each FEC supplies 160 input channels which can read signals from the same number of pads. The channels are distributed evenly on 5 SAMPA chips. Figure 3.7 presents the concept of the FEC and how the SAMPA chips are connected to GBTx.

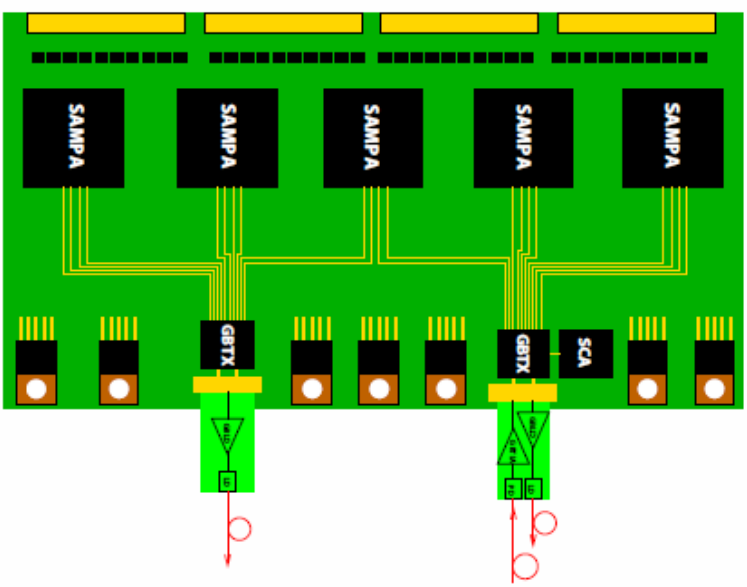


FIGURE 3.7: Schematic of the front-end card with SAMPA and GBTs. [24]

3.3.5 The SAMPA Chip

The main goal of the SAMPA project is to upgrade ALICE TPC Read-Out (ALTRO) chip which is responsible for reading and processing signals retrieved from pads. Under working on this master thesis, SAMPA chip, successor of ALTRO, was still under construction.

SAMPA is a mixed analog-digital custom integrated circuit dedicated to reading, digitalization and processing of detector signals. The SAMPA chip is developed as CMOS Application Specific Integrated Circuit (ASIC) in 0.13 μm technology with voltage supply of 1.2 V. It provides 32 channels, of either negative or positive polarity, what is especially important because the GEM based readout provides opposite polarity to the MWPC. The SAMPA ASIC can work in triggered and continuous read-out mode.

Right after the collision, the charged particles ionize the gas in the TPC detection volume, liberating electrons that drift towards the end-plates covered by pad planes. The electrons meeting a pad, create a voltage signal which results in changing of the charge at the pad. The changing of the charge is observed by the first block of SAMPA chip, named Charge Sensitive Amplifier (CSA) which amplifies the signal induced on the pad. The next block of SAMPA is called Shaper, and it is responsible for transforming the incoming pulse into a signal with long enough duration to make it readable with the sampling rate used by the SAMPA. Finally, the Analog-to-Digital converter (ADC)

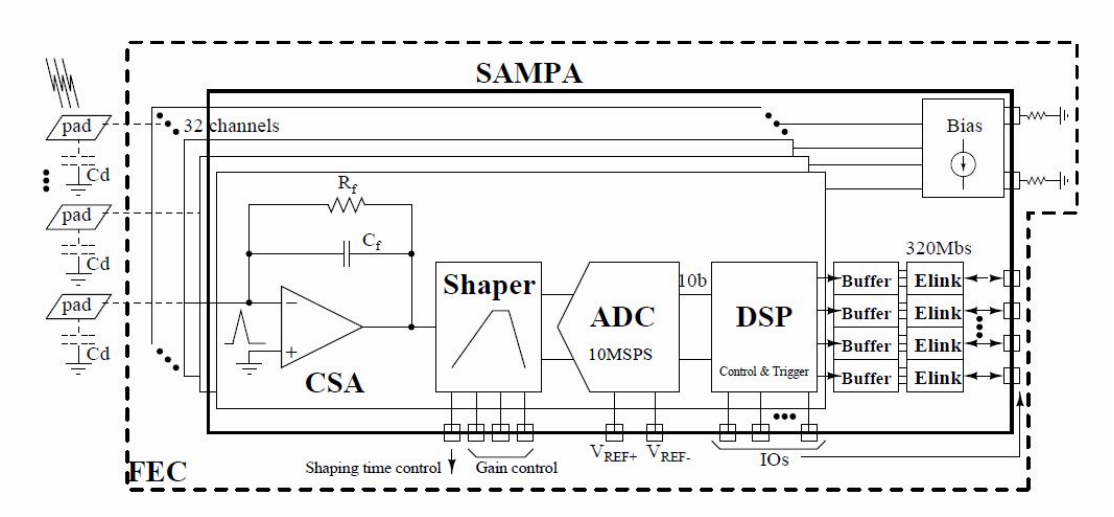


FIGURE 3.8: Block scheme of SAMPA chip. [24]

generates 10 bits data, called samples, with the frequency of 10 MHz, and forward it to the digital part of the SAMPA chip.

Each sample leaving the analog part of the SAMPA is passed through digital filters responsible for compression, formatting and buffering of data. Data compression is done using zero-suppression algorithm but Huffman encoding is also considered.

Zero-Suppression

The zero-suppression filter reduces the amount of data stored in the buffers by removal of redundant data like noise. The omitted data consist of samples with signal strength below the given threshold. Zero-suppression implemented in the SAMPA chip is configurable, meaning that it is possible to change the value of the threshold and even to deactivate zero-suppression completely to obtain all of the data. It can be useful in some cases, for example for testing purpose.

The data reduction decreases the amount of needed buffer memory. Alternatively, using the same buffer size, a higher frequency of collisions can be handled by the readout chain. Reduced throughput results also in lower demand on bandwidth between each of modules in the readout electronics.

It is usual that many particles hit a neighbouring pad while arriving to the end-plates of the detector after a collision. In this case the most central pads detect a stronger signal and surrounding pads get lower signal but still strong enough to be distinguished from noise. Such a collection of particles bombarding certain area of detection plate is called

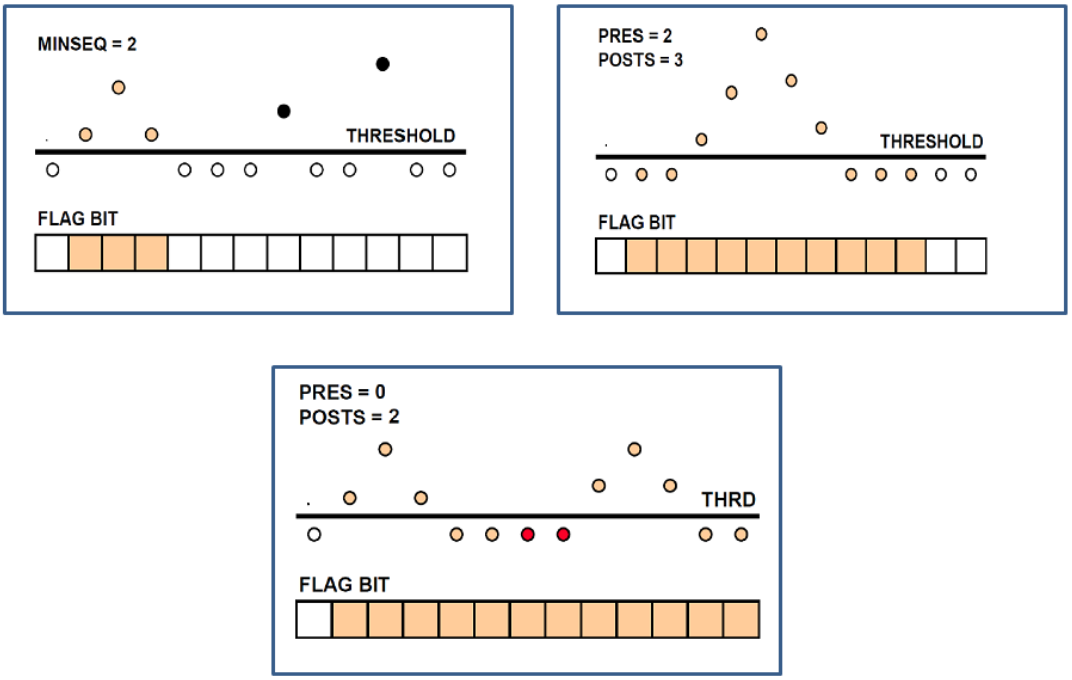


FIGURE 3.9: Illustration of the applied zero-suppression algorithm on different scenarios of input signals. The last box shows example of cluster merging. [26]

a particle shower, and the pads hit by a shower are called clusters. A typical cluster has a Gaussian shape, it spreads over 3 pads and has a width in time of 5 time-bins.

The zero-suppression algorithm which is going to be implemented in SAMPA can recognize clusters and handle them in the same way as a valid signals. Figure 3.9 shows how the SAMPA handles different scenarios of input signals using different configuration of the zero-suppression algorithm.

Data Format

Digital data are sent out through four serial output links connected to the GBTx. Every link can send 1 bit of data with the rate of 320 MHz, which results in throughput of 320 Mbps per serial link [27].

Digitalized samples are stored to the Ring Buffer, before they are sent out. There, they are waiting up to the end of the current event, called a time window. There are two buffers per input channel. One of them stores data which are composed of samples, and the second one stores headers which describe data and are used to construct packets of data.

When all samples for the current time window arrive to the buffer, SAMPA creates a data packet. One packet is created for each of 32 read-out channels. Each packet contains data and header. Data are stored as a linked list with 10-bit words. The linked list contains all samples for one channel read during one time window. The size of the data part of the packet can vary and depends on how many samples were written after zero-suppression filter to the buffer, but it cannot exceed maximal number of samples per time window which is a constant number equal to 1021 samples.

After the last sample for the entire channel and time window is collected, SAMPA creates a header. The main goal of the header is to describe the data stored in the data buffer.

A newly created header is a signal for the output link that the data collected in data buffer are ready to be sent. The four output channels are connected by default to the channel buffers in the following way:

- The first output link sends data from channels number 1 - 8
- The second one is connected to channels 9 - 16
- The third link is connected to channels 17 - 24
- And the last one takes channels from 25 to 32

However, the connection of buffers to the output links is configurable and can be changed.

The output links send data independently and simultaneously. It can happen that one output link must send a much bigger amount of data than another one. The reason is that some channels can have high momentary occupancy while other channels can have only a few samples. It is a very important case because in this scenario samples coming from one time window can be delayed and sent after the next time window. Therefore, packets must be sorted in runtime to prevent disorder of data in the next parts of the readout chain. The sorting of packets will take place in the CRU, which is described in section 3.3.7, but the initial sorting which can help CRU to sort data more efficiently is also considering to be implement already in SAMPA.

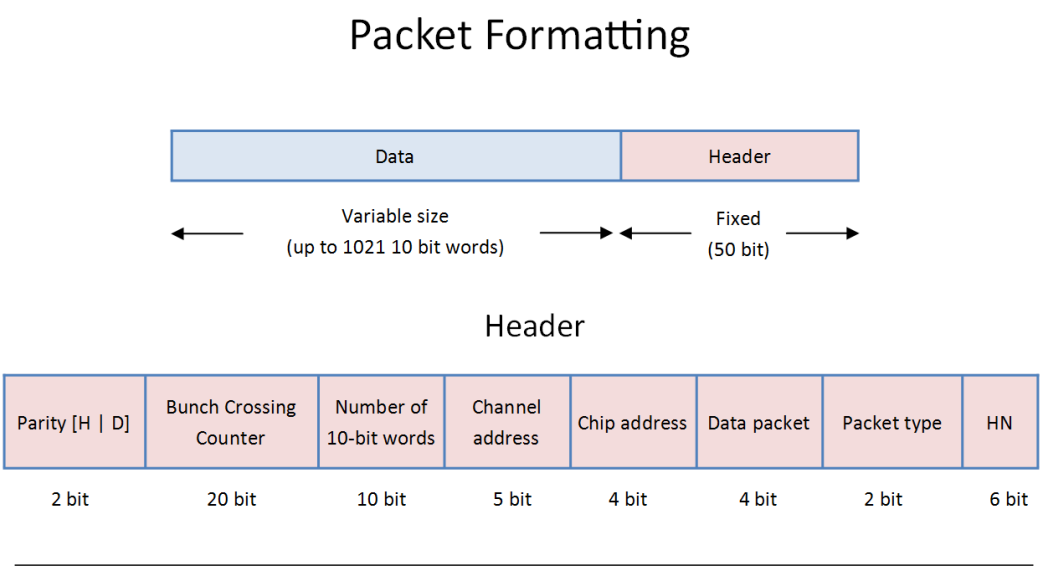


FIGURE 3.10: Illustration of packet structure composed by SAMPA.

The SAMPA chip can create different types of packets:

- Data

  Data packet consists of header and linked list of samples.

- Neighbour

  It can happen that certain SAMPA chips are more loaded with samples than other chips on the same Front-End Card. Therefore, SAMPA can be configured to send data from its neighbour chips. The neighbour data packet is the same as data packet but it contains information in header about containing data from neighbours chips.

- Heartbeat

  Heartbeat is a packet indicating that the detector electronics are operational.

- Channel fill

  This type of packet does not contain any data but only header. It is sent if there is no data in the channel to keep the readout chain synchronized.

Each type of packet has the same header but with different properties. Figure 3.10 shows the structure of the packet composed by the SAMPA. The header of the packet contains additional information about the data which is specific to the way the hardware works and how it has been designed. This information is out of the scope of this master thesis

as it is not relevant for the simulation of readout electronics and can be omitted during implementation without any impact on efficiency and performance of the readout model.

The first two bits of the packet contain data about parity of the data section and the header section of the packet. The Bunch Crossing Counter field is used to store information about bunch collisions of the LHC and is used to keep the whole front-end electronics system synchronized. The next 10 bits of the header store the number of samples in the data section of the packet. The next following fields specify the hardware address of the SAMPA chip and the input channel of this chip. This information is very important to map samples to the right sector of the detection end-plates of the TPC detector. The Data Packet and Packet Type bits inform what packet type, described above, this is. The last one field in header makes place for redundant bits which can be used to correct the header of the package.

3.3.6 GBT Module

The SAMPA ASICs are connected to the Common Read-Out Unit through the Gigabit Transceiver (GBT) system via optical links. The GBT system is a dedicated ASIC implemented in 120 nm CMOS technology, designed specially to be resistant against radiation. Each Front-End Card (FEC) contains 2 GBTs. To read data from around 500 000 channels of the TPC detector, the readout electronics foresee to use approximately 6800 GBTs in total.

GBT ASICs provide physics and monitoring data gathered from SAMPAs to the CRU and lets the CRU send control commands to the SAMPA chips using I2C protocol. Each GBT provides 10 input e-links which means that it can serve 2.5 SAMPA chips since, as mentioned in subsection 3.3.5, each SAMPA chip has four output links. The acquired data from SAMPAs are multiplexed and transmitted to the CRU.

The communication between GBT and CRU is realized through the bi-directional optical transceivers (VTRx) and uni-directional twin transmitters (VTTx) which results in effective bandwidth of 2 x 3.2 Gbit/s. The data transmission uses a special transport protocol which using forward error correction makes the communication robust and radiation tolerant.

3.3.7 CRU - Common Readout Unit

The Common Readout Unit (CRU) acts as an interface between the front-end electronics, placed on detector, and the Trigger System, On-line Farm and the Data Control System (DCS). The CRU is placed outside the radiation area in the control room and receives

data from the front-end electronics through optical, radiation tolerant fibers. The CRU is a successor of RCU2 card used during RUN 2 and its predecessor RCU which was used under RUN 1.

The CRU is responsible for data readout from the front-end electronics and for steering, monitoring and controlling the configuration of readout chain.

The input data arrive to CRU via GBTx ASICs with the effective bandwidth of 24 x 3.2 Gbit/s. Received data packets coming from SAMPA chips are sorted in CRU and sent further consecutively pad-row-by-pad-row.

The biggest challenge related to work with CRU under this master thesis, is that the final design of CRU is not determined yet. It is known, what CRU is supposed to do, but nobody knows how it will be realized. There are many ideas around CRU and the design of CRU is still evolving. It resulted in many specifications of CRU to be considered during this project.

FIGURE 3.11: Altera Arria 10 GX card with Dual-core ARM Cortex-A9 MPCore processor. [28]

The end-plates of the TPC detector are divided into sections, called pad planes. Depending on the design choice, one CRU could read data from up to 1920 or up to 2560 pads. The proposed design of the CRU assumes implementation of one input fifo buffer per one pad. Once each fifo receives data packets containing samples for entire time window, CRU will send data in a predefined order. It will result in an easy to implement way for sorting data in the CRU.

The application of specific functionalities, requires Common Read-out Units to be implemented as electronics boards with custom designed, programmable functionality based on up-to-date FPGA technology. A FPGA (Field-Programmable Gate Array) is an integrated circuit designed to be programmed by a customer.

The PCI40 card with the Altera Arria 10 GX is a major candidate to be used as a base for developing the CRU. The Altera Arria 10 is a high-performance FPGA developed in 20 nm technology [29]. On the board of GX version of Arria 10 is installed Dual-core ARM Cortex-A9 MPCore processor with clock frequency of 1.5 GHz. The board offers 32 GB of embedded memory which can be extended with external memory. It supports also PICe x8 interface compatible with the 3rd generation.

However, if it turns out that this board will have not sufficient performance for requirements imposed in relation to CRU, an alternative solution will have to be found.

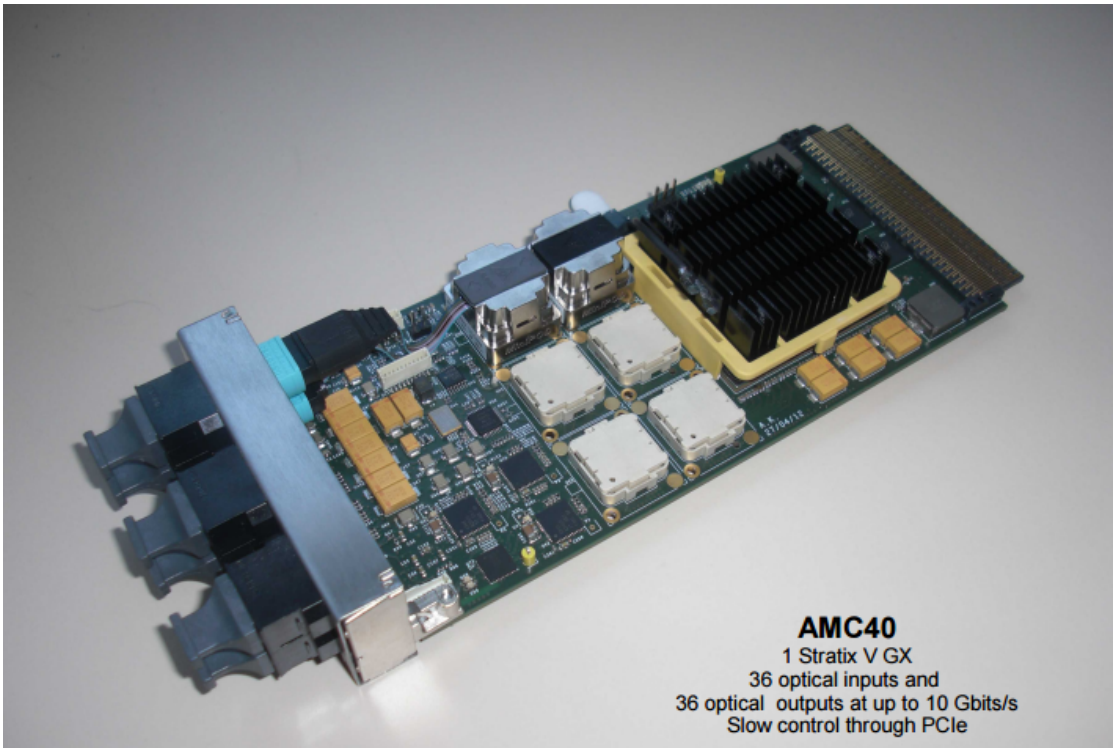


FIGURE 3.12: AMC40 board with Altera Stratix V processor. [30]

Different Designs of CRU

The CRU will read data from 12 FECs, process them and send them out via one PCIe x16 3rd generation link. The alternative solutions assume that the CRU would read data not from 12 FECs but from 16 FECs. It would result in demand for bigger memory size implemented in CRU.

FPGA Family Name	Xilinx Virtex 6	Altera Stratix V GX	Xilinx Virtex 7	Altera Arria 10 GX	Xilinx Virtex Ultrascale	Altera Stratix 10	CRU Requirements (without SAMPA sorting)
Status		available	available	ES available from Q2 '15	available	end of 2017	
FPGA Part Number	XC6VLX240T	5SGXEA7	XC7VX690T	10AX115	XCVU190	10SG280	
Used in...	C-RORC	AMC40	MP7	PCIe40			
Logic Elements / Cells [M]	0.241	0.622	0.693	1.15	1.9	2.8	
FFs [M]	0.3	0.939	0.866	1.7	2.14		
LUTs [M]	0.15	0.235	0.433	0.425	1.07		
18/20 Kb RAM Bloks	832	2560	2940	2713	7560	11721	1920 / 2560
Total Block RAM (Mbit)	15	50	53	53	133	229	40 / 53
>= 10 Gb/s Transceivers	24	48	80	96	60	144	48
PLLs	12	28	20	32	60	48	
PCIe x8, Gen3	2 (Gen2)	4	3	4	6	6	

FIGURE 3.13: Different hardware solutions considered to be base for implementation of CRU. [31]

In addition, the PCIe could be replaced with 8 DDL3 links. DDL3 is a 3rd version of Detector Data Link which is characterized by its bandwidth of 10 Gb/s.

One of the considered alternatives, during this master thesis, was use of the AMC40 board with Altera Stratix V processor. The AMC40 card is developed to perform data readout in LHCb experiment. The card has 36 optical input links which means that is could serve up to 18 FECs.

Alternative FPGAs which can be use to implement CRU, are shown in the table presented in figure 3.13. The table shows desired buffers structure for CRU and how different FPGAs fulfill this requirement.

Location of CRU

The location of the CRU is one of the factors affecting the final design. Two ideas of location of the CRU were considered by the scientists from CERN and each of the solutions has its advantages and disadvantages. One of the ideas was to locate the CRU in the cavern, directly on the front-end electronics, installed on the detector. This solution foresees usage of 10 GbE long fiber links connecting the CRU to the Online Farm and further systems. It would result in lower number of used links. Unfortunately, this solution would require designing CRU based on radiation tolerant FPGAs which are characterized by comparatively low performance [23].

Finally, it was decided to place the CRU units in the control room, outside the radiation area. This solution presents a more robust and clean system with more processing power and not limited access during LHC operation.

The location of the CRU affects its design which also has impact on this master thesis. The simulation developed during this work was used to compare two different implementations of output links of the CRU. The usage of eight DDL3 links with bandwidth of approximately 10 Gb/s each, was compared to the performance reached by the CRU using one PCIe output link with the third generation of PCI Express x16 and the results are presented in chapter 6.

Chapter 4

Method Discussion

This chapter introduces major methods and computer tools used to find solution for the described problem in the previous chapter.

4.1 Research Method

Designing new hardware is a challenging and time consuming process. Many parts of the readout electronics for TPC detector for RUN 3 are still under consideration. The computer model of the hardware should help to complete the final design. When developing a new hardware several questions arise:

- How can we evaluate the correctness and quality of the designed hardware?
  - Will the designed hardware work properly? Is the hardware performance enough to satisfy increasing data rate?
- Is it a better way to design the required hardware? The simulation will help to find answers to many additional questions, like:
  - What buffer size is needed on particular modules? How long will data processing take?

By using software development tools and a framework to simulate a hardware, executable models can be created in order to simulate, validate, and optimize the system being designed. The simulation will help not only to verify correctness of the design but also to propose improvements. This thesis supports not only the electronic engineers in designing the hardware but it contributes to the field of physics as well. If the

performance of TPC can be improved through experimental optimization, this enables more careful analysis of raw data during the given time frame. The end result should be higher quality data from the experiment to the particle physics community, a very important goal for such an enormous experiment as ALICE.

4.1.1 Simulation

The first part of the project is to develop a virtual implementation of the proposed hardware model. The implementation will be used to simulate readout electronics with the samples of real data from the previous experiments. The challenge is to understand the model and learn how to use a proper framework for computer modelling and evaluation of hardware. The simulation can help not only to evaluate the model but also to find answers to many questions around design of the model. It will also help to plan those parts of the model which are not designed yet.

4.1.2 Measurement

Observation of the data flow in the system will help to estimate parameter values on certain modules. A good example here is a buffer size needed by particular elements in the system. The correct size of the buffers cannot be too small or too large. If the size of the buffer is too small, it will result in insufficient performance or even worse, it could affect the results of the experiment. On the other hand, if the buffer size is overestimated, it will result in unnecessary cost of the hardware, what is quite important when we realize that the high speed memory is a very expensive part of the hardware.

4.1.3 Test Bench

The development of a test bench is needed to experiment with the model. Since design of some parts of the hardware is not accomplished yet, these parts will be implemented in the simulation as black boxes. The SystemC framework [32] will be used to implement the test bench. SystemC is a library for C/C++ programming language. It provides specialized data types and classes which help to develop an event-driven simulation.

4.1.4 Evaluation

The main goal of the research is to contribute to the ALICE experiment and the research should be evaluated against it. The virtual model should help to solve the problems

occurring while designing the hardware for upgrade of TPC. If the time will allow it, the computer model will be used directly to find solutions for the problems. If not, the simulation should be an important support tool for designing and testing the desirable readout electronics.

4.2 SystemC Framework

SystemC is a framework used to develop computer models of hardware systems. SystemC is delivered as an open source library containing structures for modelling hardware components and their interactions. An important part of SystemC is a simulation kernel which allows to evaluate behaviour of the designed hardware model through simulations.

SystemC is based on the C++ programming language. It extends the capabilities of C++ to enable hardware description by adding such important concepts as concurrency, events and data types [33].

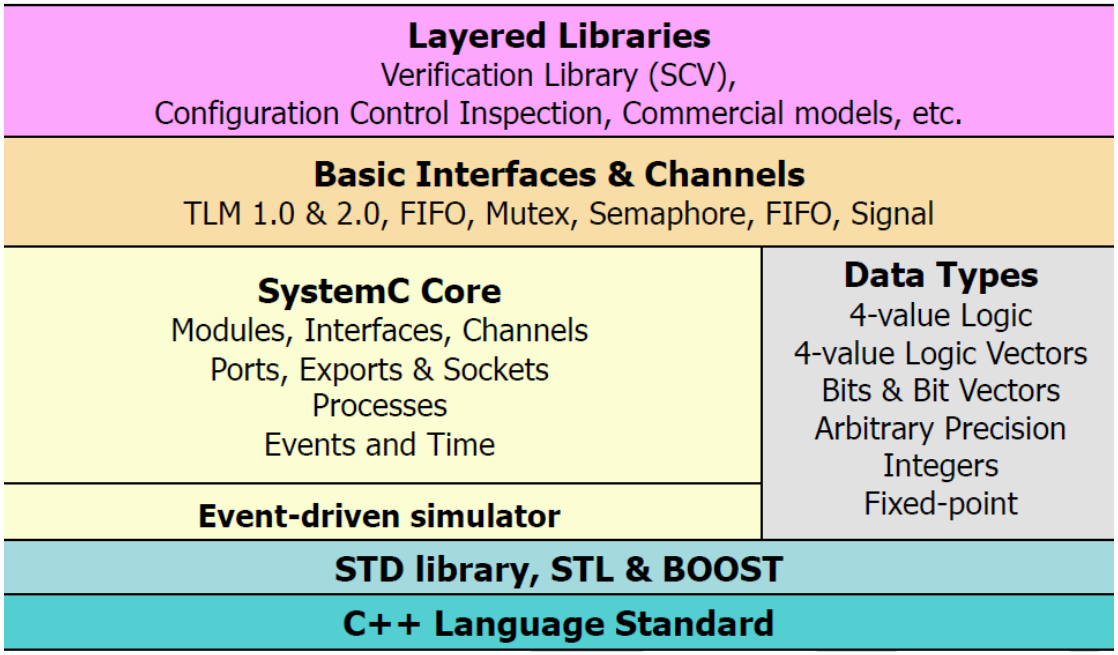


FIGURE 4.1: Structure of SystemC framework. [34]

By using C/C++ development tools and the SystemC library, executable models can be created in order to simulate, validate, and optimize the system being designed.

The executable model is essentially a C++ program that exhibits the same behaviour as the system when executed.

The simulated system is broken up into smaller, more manageable pieces like:

- **Modules**

  Modules are the basic building blocks of a SystemC design hierarchy. A SystemC model usually consists of several modules which communicate via ports. The modules can be thought of as a building block of SystemC.

- **Ports**

  Ports allow communication from inside a module to the outside (usually to other modules) via channels.

- **Processes**

  Processes are the main computation elements. They are concurrent.

- **Channels**

  Channels are the communication elements of SystemC. They can be either simple wires or complex communication mechanisms like FIFOs or bus channels.

  Different types of channels are: signal (the equivalent of a wire), buffer, fifo, mutex and semaphore.

- **Interfaces**

  Ports use interfaces to communicate with channels.

- **Events**

  Events allow synchronization between processes and must be defined during initialization.

- **Data types**

  SystemC introduces several data types which support the modelling of hardware.

4.2.1 SystemC - Code Example

The classical example of SystemC usage is a simple system consisting of two modules: Producer and Consumer. One module, called Producer, is a module which generates data and sends them to the second module, called Consumer. The Consumer module reads data and write information about them to an output.

This simple Producer-Consumer system is used to show how the SystemC framework can be used in practice. The following code listings show implementation of the example model.

The most fundamental methods, used in the example and used very often in this master thesis, are:

- sc_start(value , [sc_time_unit]);

  Initializes simulation for a given running time

- sc_stop();

  Stops simulation

- sc_time_stamp();

  Returns current simulation time

- wait(time);

  Waits for a given period of time

- write();

  Writes data to a port

- read();

  Reads data from a port

The listing 4.1 shows the implementation of the Producer module.

```
1 SC_MODULE(Producer) {
2   void t_source(void);
3   sc_port < sc_fifo_out_if<int> > out_port;
4   // Constructor
5   SC_CTOR(Producer) {
6     SC_THREAD(t_source);
7  }
8 }
```

Listing 4.1: Implementation of Producer module using SystemC.

Modules are the basic building block within SystemC. The keyword SC_MODULE(Producer) is used to initialize a new module named Producer. The macro SC_MODULE can be replaced with the pure C++ syntax like:

```
1 class Producer : sc_module {
2 //Module body
3 }
```

Listing 4.2: Alternative implementation of Producer module using SystemC.

Any module in basic should contain ports, constructor, and methods to work on the ports [35]. There are three types of ports:

- in - Input Ports
- out - Output Ports
- inout - Bi-direction Ports

In general, a port is declared using the class sc_port. Both Producer and Consumer use this class and specify an interface. The producer uses output interface of type fifo, it declares also that this interface will be used to send integers: sc_fifo_out_if <int>. The Consumer specifies an input interface in an analogous manner, as shown in the listing 4.4.

```
1 void Producer::t_source(void) {
2   int val = 0;
3   for (int packetNumber = 0; packetNumber < 10; packetNumber++) {
4     out_port->write(val);
5     cout << sc_time_stamp() << ": Producer(): Wrote " << val << ", to
      port " << i << endl;
6     val++;
7     wait(1, SC_NS);
8   }
9 };
```

Listing 4.3: Implementation of method sending data to port.

The Producer declares the method t_source which is used by the thread declared in constructor on the end of the module. Thread is a kind of process which when called keeps executing or waiting for some event to occur.

The listing 4.3 shows the implementation of the method t_source used by the Producer to send data to the Consumer module. The code for sending data out is nested inside the for-loop which repeats a transmission for a given number of times.

```
1 SC_MODULE(Consumer) {
2   void readInput(void);
3   sc_port < sc_fifo_in_if<int> > in_port;
4
5   // Constructor
6   SC_CTOR(Consumer) {
7     SC_THREAD(readInput);
8   }
9 };
```

Listing 4.4: Implementation of Consumer module.

After each transmission, the thread waits 1 nanosecond. The waiting time and unit is specified as parameters for the wait(...) method. In real scenarios, it is natural to calculate a real time it takes to send data, based on data size and speed of the link between modules.

```
1  void Consumer::readInput(void) {
2    int val;
3    while(true) {
4
5      if (in_port->read(val)) {
6        cout << sc_time_stamp() << ": Consumer(): Received "
7        << val << ", on port " << i
8        << endl;
9      } else {
10       cout << sc_time_stamp() << ": Consumer(): FIFO empty." << endl;
11     }
12   // Check back in 5ns
13   wait(5, SC_NS);
14   }
15 };
```

Listing 4.5: Implementation of method reading data from port.

The Consumer module uses the readInput method to receive data from the Producer module. The part of implementation responsible for reading data from channel, is nested inside a while-loop which is executed all time during running of simulation.

```
1  int sc_main(int argc, char* argv[]) {
2    Consumer consumer ("Consumer");
3    Producer producer ("Producer");
4
5    sc_fifo<int> fifo_channel = new sc_fifo<int>(5);
6
7    producer.out_port(fifo_channel);
8    consumer.in_port(fifo_channel);
9
10    sc_start(100, SC_NS);
11    return 0;
12 }
```

Listing 4.6: Creation and connecting together modules in the main method.

Data are read from a channel by the SystemC read(...) method and are saved into the val variable. This method provided by SystemC library returns true if data were read from the channel and false if the channel was empty. The returned boolean value is used to decide if the received data should be written out to the screen or not. The SystemC method sc_time_stamp() is used as part of the output string which writes information

about current, real time in the simulation. Before data is received, the Consumer waits 5 nanoseconds.

In the main method, shown in the listing 4.6, all modules are connected together. The modules Consumer and Producer are instantiated. The channel between the output port for Producer module, and the input port for the Consumer module is created and then used to connect them together.

At the end, the simulation is started for 100 nanoseconds. The program terminates when the computer simulation is done.

Chapter 5

Implementation

This chapter describes the structure, design and implementation of computer model of the readout electronics and test bench used to run simulation.

5.1 Simulation of the Readout Electronics

The programming language C++ and SystemC framework were used to develop a computer model of the readout electronics. As mentioned already in section 4.2, SystemC is a framework used to develop computer models of hardware systems. SystemC is based on the C++ programming language and it extends the capabilities of C++ to enable hardware description by adding such important concepts as concurrency, events and data types.

The simulation which enables to test the model is essentially a C++ console application. It can be compiled and then run both on Linux and Microsoft Windows operating systems.

During the early phases of the project, the source code was developed in Microsoft Visual Studio. This tool was used to experiment and to explore the capabilities delivered by the SystemC framework.

Running the simulation is time consuming. Depending on the input scenario, it takes 2-3 days to run simulation over tens of time windows. The number of time windows to be simulated has a great impact on running time, but this is not the only relevant parameter. The running time is affected largely by the complexity of the model which is supposed to be simulated. For instance, to test the SAMPA chip, one SAMPA module, and generated input data to its 32 channels is required. Testing the CRU in realistic

conditions requires to connect all 12 or 16 FECs, depending of design choice, which results in 60-80 SAMPA chips and from 1920 to 2560 readout channels.

Many tests and experiments which allowed to discover and remove programming bugs and finally to check correctness of developed software, were executed on a simplified model which consisted of only 1 FEC.

On this stage of the project it was clear, that it would not be possible to generate any final results using the simulation before the deadline of the master thesis without optimization.

The code optimization and a few tricks allowed to reduce running time of the simulation to about one day for about 50 time windows without any impact on the final results.

In addition, the simulation was run in parallel in a several virtual machines on several physical data machines. The virtual machines helped to clone the development environment together with all required libraries and tools to run the simulation. The source project was ported from Microsoft Windows environment to Linux environment to avoid the limitation imposed by the licensed software.

The freeware version of WMware Player was used to create and run virtual machines. Linux Mint was chosen to be installed as a guest operating system. The source code has not used any system calls specific to the operating system, and therefore the porting process was easy and it was not time consuming.

To make the simulation operational on a Linux system, a makefile was created. A makefile instructs a compiler to compile source code in a proper way. There were a few problems during porting related to use of threads by SystemC and to support of the new C++11 standard by the compiler. To solve these problems, a few extra parameters had to be added to the final makefile to compile code using SystemC correctly.

The simulation uses a configuration file to setup the model and to specify the running parameters. Using a configuration file it is possible to set many properties of the simulation, like for example: number of time windows to simulate, number of samples in each time window, sampling rate, many properties of the readout electronics model and each module like for instance: number of FECs, clock time for each chip, number of channels between modules, and much more.

It turned out that this approach is very flexible and running of the simulation can be automated. The simulation can be started by a custom developed bash script, which can change the properties of the simulation and setup the model. Then, when simulation is done, the script can take care for results, change simulation parameters and run it again.

5.2 Data Generator

The Data Generator, implemented in the DataGenerator class, acts as a detector in the computer model of the readout electronics, which supplies the model with input data. The Data Generator is implemented as SystemC module which is connected with each of the SAMPA chips by ports. The number of output ports Data Generator has, depends on the number of SAMPA chips used in the simulation and on the number of input channels each SAMPA chip has. These values are easily customizable by changing the parameters in the configuration file. There are also other parameters in the configuration file which control the behaviour of the Data Generator. The code listing 5.1 shows the part of the configuration file used to configure Data Generator. The five parameters are:

1. NUMBER_TIME_WINDOWS_TO_SIMULATE

   This parameter is used to manipulate the number of time windows, during which input data are generated. The simulation can run longer than the set number of time windows by this parameter, but after this time the model will not be supplied with input data.

   The value of this parameter, and the next one described, has directly impact on the real running time of the simulation.

2. NUMBER_OF_SAMPLES_IN_EACH_TIME_WINDOW

   The parameter determines the number of samples which can be read by the SAMPA during one time window. The standard value in the real model of the readout electronics is 1021 samples per time window and it was no need to change this parameter often. However, it is a good practice to avoid hardcoding any numbers in the code. It would result in source code which is difficult to maintain and to understand.

3. DG_WAIT_TIME

   The time between each series of new generated samples is set in this parameter. The waiting time must be given in nanoseconds. The waiting time for the Data Generator is a calculated time based on the sampling rate of the SAMPA chips. The time when the Data Generator creates a new series of samples must be synchronized well with the time when the SAMPA reads data from its channels.

4. DG_OCCUPANCY

   This parameter is used by the Static Data Generator, described later in this chapter. The occupancy parameter determines average occupancy of data for all input channels over each time window. For instance, if there is maximum 1021 samples

per channel in each time window and occupancy is set to 30%, so in average each channel during each time window receives approximately 306 samples.

5. DG_GENERATE_OUTPUT

   This parameter simply decides if there will be generated log entries in the log file by Data Generator module. The parameter is a boolean value.

```
1  //Data Generator
2  const int NUMBER_TIME_WINDOWS_TO_SIMULATE = 20;
3  const int NUMBER_OF_SAMPLES_IN_EACH_TIME_WINDOW = 1021;//1021 std value
4  const int DG_WAIT_TIME = 100; //ns, 10MHz
5  const int DG_OCCUPANCY = 30; //%
6  const bool DG_GENERATE_OUTPUT = false;//writing to logfile
```

LISTING 5.1: Part of the configuration file showing parameters for the Data Generator module.

5.2.1 Implementation Details

The Data Generator module is implemented in two files: header file (.h) and source file (.cpp). The header file declares the module with all its data structures and methods, while the source file defines the implementation of the module.

```
1  SC_MODULE(DataGenerator)
2  {
3  public:
4
5    void t_sink(void);
6    void write_log_to_file_sink(int _packetCounter, int _port, int
       _currentTimeWindow);
7
8    sc_port < sc_fifo_out_if<Sample> > porter_DG_to_SAMPA[
9    constants::NUMBER_OF_SAMPA_CHIPS*constants::SAMPA_NUMBER_INPUT_PORTS
10   ];//number of sampa * number input ports per sampa
11
12   // Constructor
13   SC_CTOR(DataGenerator)
14   {
15     SC_THREAD(t_sink);
16   }
17 };
```

LISTING 5.2: Part of the header file of Data Generator.

The computer model of the readout electronics has been tested with three different input scenarios and for that reason, three different editions of the Data Generator were created. The general implementation shared by the three Data Generator modules is discussed below. The implementation details, specific for each Data Generator variant, are described in the following subsections.

The base part of the header file, common for each Data Generator module, initializes two methods: "t_sink" and "write_log_to_file_sink", an array of output ports: "ports_DG_to _SAMPA" and the constructor which initializes a thread based on "t_sink" method as shown in the listings 5.2.

The "t_sink" method in the Data Generator is a main method responsible for creating of samples which are injected to the channels connecting the Data Generator with the SAMPA chips. The implementation of the method consists of the while-loop which is executed one time for each time window being simulated.

Inside the while-loop a for-loop which enumerates input channels for each SAMPA chip is implemented. In this place of the model, a simplification which makes implementation much less complex and much more clear was made. Because the concept of Front-End Cards was not introduced into computer model, the connection between the data generator and SAMPAs was much easier to implement. The SAMPA chips do not need to be addressed in any way. It is known that each SAMPA chip has a certain and fixed amount of input channels. Based on this knowledge the iteration of the SAMPA chips and the channels was very easy to achieve in the code. The principle of addressing input channels is as follows:

If it is given by the configuration file, that the model will consist of 12 FECs, each FEC of 5 SAMPA chips and each SAMPA chip will have 32 input channels, so there must be 12 * 5 = 60 SAMPAs and 60 * 32 = 1920 input channels. Using the mathematical modulo operation with index of certain input channel as a dividend, and divisor equal to the number of channels for each SAMPA, it is possible to calculate from which SAMPA chip and channel each sample is coming from. For example, if a given sample is addressed with number 75:

75 / 32 gives SAMPA[2]

75 mod 32 gives channel[11]

It results in SAMPA[2] and channel[11]

It means that the sample was read by SAMPA chip indexed with number 2 and by its channels number 11. Because the indexing starts from [0] the sample comes from the twelfth channel of the third SAMPA chip installed on the first FEC. If FEC would be

needed to be implemented in the computer model for some reason, it could be calculated in an analogous manner.

The mechanism described above is simple but powerful. In spite of its simplified implementation, it allows to send data to the exactly specified input port, belonging to the specified SAMPA chip, which are placed on the particular FEC. The implementation is shown in the listing 5.3

```
1 //Produce samples for given number of time windows
2   while(currentTimeWindow < constants::NUMBER_TIME_WINDOWS_TO_SIMULATE)
3   {
4     //foreach channel for every SAMPA chip
5     for(int i = 0; i < (constants::NUMBER_OF_SAMPA_CHIPS * constants::
      SAMPA_NUMBER_INPUT_PORTS); i++)
6     {
7       [...]
8     }
9   }
```

Listing 5.3: Part of the implementation of main loop inside the data generator.

Samples are created and sent to the channels inside the for-loop. Each sample is an instance of the Sample class, which is a custom defined object type. The Sample class is used by all implementations of Data Generator. The table 5.1 shows structure of the Sample class.

Class Element	Description
Sample ID	Each sample get its unique id. The id is an integer generated by a counter inside the Data Generator. The id makes it easier to debug the simulation and it allows to track precisely the path of each sample.
Time Window	Stores the information about during which time window the sample was generated.
Signal Strength	This variable is used by the Black Event Data Generator to store data about charge value for each generated sample. The charge value is used by zero-suppression algorithm, implemented in SAMPA, to filter out noise.
Constructor	There are two constructors implemented in Sample class. The standard constructor, initializing all variables to its standard values, and the special constructor, allowing to set the desired values of the class variables.
Output operator <<	Overloading of output operator is required by SystemC library for objects to be sent through ports implemented using SystemC.
Assignment operator =	Like described above, assignment operator is also required because of the SystemC framework specification.

TABLE 5.1: Structure of the Sample class.

The Sample class is used as shown in the listing 5.4. An instance of the Sample class is created by using the special constructor with specified time window, sample id and the value of charge. After being created, each sample is sent to the appropriate output port of the Data Generator by the "nb_write" method.

```
1  Sample sample(currentTimeWindow, packetCounter, 0);
2  //Send a sample to appropriate SAMPA and SAMPAs channel
3  porter_DG_to_SAMPA[i]->nb_write(sample);
4  //Save event to the logfile
5  write_log_to_file_sink(packetCounter, i, currentTimeWindow);
```

LISTING 5.4: Creating and sending samples by data generator.

The last set of instructions to be executed in the while-loop, shown in the listing 5.5, consists of an if-test which checks number of samples to be sent in the current time window. If the number of samples sent for each channel is equal to the number of samples set in the configuration file, then the condition of the if-test is positive and the contents of if block is executed. The Time Window Counter is incremented and the number of samples sent during the time window is reset to zero. The information about the progress of simulation is written to the output.

Irrespective of the result of the if-test condition, after each series of sent samples to the output ports, the Data Generator waits a certain period of time. This time is set by the configuration file and it is based on the sampling ratio of SAMPA chips, as described in point 3 of the parameters list for Data Generator module.

When the while-loop has been executed for the last time window, the work of the Data Generator module is done. The method generating samples will not be executed any more, and no more samples will be created. However, the simulation will still be running. Buffers on particular modules will be drained out from the data and SAMPA chips will generate only empty packets, without any samples.

```
1  //If this time window is done, go to next time window
2  if(currentSample == constants::NUMBER_OF_SAMPLES_IN_EACH_TIME_WINDOW )
     //1021 samples
3  {
4    currentTimeWindow++;
5    currentSample = 0;
6    cout << "currrent time window: " << currentTimeWindow << ", TimeStamp
     : " <<  sc_time_stamp() << endl;
7  }
8    //Each sample get its own unique id (currentSample)
9    //Can be used to identify samples and to track path of the sample in
     logfile
10   //or in the code
11   currentSample++;
12   //SAMPA receives 10-bit data on 10 MHz
13   wait(constants::DG_WAIT_TIME, SC_NS);
14 }
```

LISTING 5.5: Part of the loop implemented in the data generator, responsible for creating and sending samples to SAMPA chips.

5.2.2 Various Implementations of Data Generator

Together with the simulation follow three different versions of the Data Generator which serve three alternative kinds of input data to the model. All three variants are based

on the same principle of sending samples to the SAMPA chips, but the implementation of the part responsible for generating input varies. The three Data Generator modules used to test CRU are:

1. Static Data Generator
2. Gauss Data Generator
3. Black Event Data Generator

5.2.3 Static Data Generator

The Static Data Generator is the first and the simplest implementation of Data Generator module. It uses a random generator to introduce the concept of probability of occurrence of a signal on a certain reading pad. As mentioned already, one input channel is connected to one pad on the end-plates of the detector.

The basic principle of the Static Data Generator is as follows:

The main while-loop is iterating time windows and samples during each time window. Inside the while-loop a for-loop which iterates input channels is implemented. Finally, the if-test decides if there will be a signal, which is used to create a sample, for each channel or not. The implementation of this solution is shown in the listing 5.6.

The if-test uses DG_OCCUPANCY, mentioned in point 4 of the parameters list for Data Generator module, to set the probability for creating a sample. The method "generate(...)" from the custom class RandomGenerator is called with two parameters. The parameters provide the range of numbers generated by the random generator. In this case, the parameters are 0 and 100, which means that an integer between those two numbers inclusive will be returned by the method. Then, the if-test compares the generated integer to the value of occupancy provided by the configuration file. If the integer is lower or equal to the occupancy, the sample will be created and sent to the SAMPA. If the test condition is not fulfilled, then no sample will be generated and the data generator will jump over to the next channel and perform the test again.

The two input parameters set to 0 and 100 makes a range of integers corresponding to the 0 - 100% scale of probability. If the occupancy parameter in the configuration file is set to be for instance 30%, it means that the integers from 1 to 30 fullfills the test condition, which results in chance of 30% to generate a sample.

This solution generates flat occupancy over channels and over time. However, on account of using a random generator, small variations of data distribution over channels can occur.

```
1       //Decide whether sending a sample to the particular channel or not
2       if(randomGenerator.generate(0, 100) < constants::DG_OCCUPANCY)
3       {
4         //Create a new sample
5         Sample sample(currentTimeWindow, packetCounter, 0);
6         //Send a sample to appropriate SAMPA and SAMPAs channel
7         porter_DG_to_SAMPA[i]->nb_write(sample);
8         //Save event to the logfile
9         write_log_to_file_sink(packetCounter, i, currentTimeWindow);
10      }
11      //Bypass Data Generator with for instance:
12      //else if((currentTimeWindow == 0 || currentTimeWindow == 1) && (i
    < 30))
13      else if (false)
14      {
15        [...]
16      }
```

LISTING 5.6: Part of Static Data Generator.

As shown in the listing 5.6, on the end of the if-test section is made place to create a next test condition by else if(...) structure. In this place the test based on random generator can be bypassed by another condition. The comments in the code listing show an example usage of secondary if-test which ensures that the first 30 input channels will get 100% occupancy during the first two time windows. Using this solution, many additional tests can be performed using only the Static Data Generator. The example of usage of bypassing the first if-test can be for instance testing behaviour of one SAMPA chip which can get a much higher amount of data than the other SAMPA chips.

5.2.4 Gaussian Data Generator

The main purpose of developing a Gauss Data Generator was to create a more realistic input scenario to the computer simulation. The Gauss Data Generator is an extension of the Static Random Generator, and it is meant to generate input based on Gaussian, called also normal, distribution of samples.

The Gaussian distribution is described by the form 5.1.

$$f(x) = a \exp\left(-\frac{(x-b)^2}{2c^2}\right)$$

FIGURE 5.1: The form of the Gaussian function.

The parameter a is the height of the curve's peak, b is the position of the center of the peak and c is the standard deviation which controls the width of the "bell" curve. The example of the curve generated by the Gaussian distribution is shown in the figure 5.2.

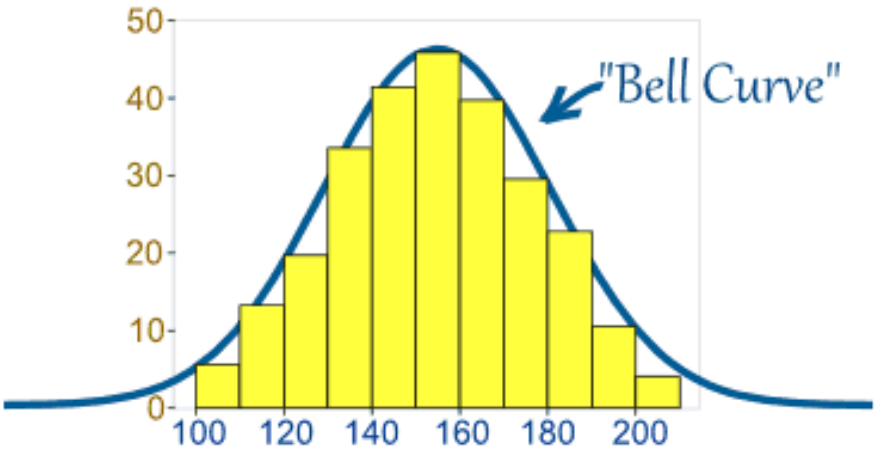


FIGURE 5.2: Example of normal distribution Bell Curve. [36]

The idea behind the Gauss Data Generator is to distribute data over input channels and over time. In the first scenario the data is distributed over channels like in the figure 5.2, where the horizontal axis represents channels and the vertical axis amount of data. In the second scenario the horizontal axis represents time, and amount of data varies from time window to time window but it is equal for each channel during a particular time window.

The occupancy based on a Gaussian distribution was implemented as an array storing the occupancy for each port or each time window. The Gauss Data Generator instead of using occupancy parameter from the configuration file, uses occupancy based on Gaussian distribution. This value is put into if-test which determines creation of samples.

5.2.5 Black Event Data Generator

The third implementation of the Data Generator, called Black Event Data Generator, uses black events as input data to the simulation. The Data Generator reads first file containing input data, looking for the events with recorded samples. When some event containing data is found, then the channel address, value of detected charge and time is stored in the memory. When reading the file, the data generator decodes the address of input channels to make it possible to inject the sample to the appropriate SAMPA chip and input port.

Black Events

The black events are real data collected during the RUN 1 of the Large Hadron Collider by the ALICE Experiment. Some of black events based also on simulated data. The black events data are delivered as a text files which were converted from the files supported by the computer program AliRoot. The AliRoot is a software used in ALICE experiment for off-line data analysis, reconstruction and as data simulation tool [37].

The approximate size of one file with black events is 3.5 GB. The file contains information about the signal charge detected on the pads, time when the charge was detected and address for readout channel connected to the pad.

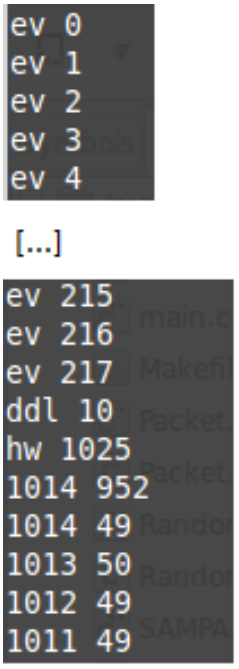


FIGURE 5.3: Part of the text file containing black event.

Figure 5.3 shows an example of a part of the file with black events. The file lists events as "ev x", where x is ordinal number of an event. Most of events do not contain any information. The figure above shows that all events until event number 217 do not have any data. Then, the event 217 stored data for 952 timebins, started from timebin 1014 to 62. The data consist of readout branch and address of the pad, timebin - when data was recorded for the listed address and the value of charge on the pad. The example in the figure 5.3 can be read as follows:

The input data were detected for the event number 217 and contain charge values for 952 timebins read from branch and channel with address 1025. The charge values are:
during timebin 1014, charge value is equal to 49,
during timebin 1013, charge value is equal to 50, ...

When all values of charge for the given address are listed, then the file goes to the next addresses, and list charge values for the next pads.

Address Decoding

The implemented logic for reading the file with black events tries first to find an event containing data. It is realized by searching the "hw" string in the line. If the string is found then it is known that the event has stored data and the reading procedure for samples for this address is started. When reading the data for the particular address is done, then the logic goes to the next address and repeats the reading procedure for this address.

To address an input channel in the computer model correctly, it is required to know the channel and the SAMPA chip it belongs to. The address in the files with black events is given as a decimal number like for example "hw 2358".

To decode the address, the decimal number is first converted to the binary format. The address is decoded from the four groups of bits:

- 0 - 3 Channel
- 4 - 6 SAMPA chip
- 7 - 10 Front-End Card
- 11 - 15 Branch

For instance, the given address "hw 2358" is decoded to be the channel number 6 belonging to the SAMPA number 3 installed on FEC number 2. It is important to remember

that ordering of channels, SAMPA chips and FECs starts from zero. The computer model simulates only one branch of the readout electronics so it is not needed to decode the branch number. However, the branches with low numbers belong to IROCs - Inner ReadOut Chambers, which have higher occupancy, which results in the worse input scenario based on black events. Therefore, the branch with index number 0 was used in the simulation. Figure 5.4 shows the example of address encoding.

Decimal Address	Binary Address
2358	00001 0010 011 0110

Hardware Address		Branch	FEC	SAMPA	Channel
2358	Binary	00001	0010	011	0110
	Decimal	1	2	3	6

FIGURE 5.4: Encoding of the hardware address from file containing black events.

A careful reader could notice a quite significant problem related to this method of hardware addressing. The hardware address in binary form uses 4 groups of bits to addressing. A four bits address allows to store 16 values what is sufficient for FECs and SAMPAs but not for 32 input channels for each SAMPA.

The black events based on data read out by the electronics used during the RUN 1 which results in different setup of addressing. The predecessor of SAMPA - ALTRO chip could read data only from 16 instead of 32 channels. To solve this problem, empty channels were mapped to the channels which get data to reuse signals twice.

The address encoding is implemented by using bitshift operators and binary mask. The binary masks used for address encoding are shown in the figure 5.5.

Address	Decimal Mask	Binary Mask
Channel	**15**	00000 0000 000 1111
SAMPA	**112**	00000 0000 111 0000
FEC	**1920**	00000 1111 000 0000
Branch	**63488**	11111 0000 000 0000

FIGURE 5.5: Binary masks used to decode hardware address.

The binary address is stored in the code as a bitset containing 16 bits. The implementation of the method encoding channel from the whole hardware address is shown in the listing 5.7.

```
int DataGenerator::decodeChannelAddress(unsigned int _hw)
{
  unsigned int channelMask  = 15;

  std::bitset<16> channelAdd{_hw & channelMask};
  unsigned int channelNo = channelAdd.to_ulong();

  return channelNo;
}
```

LISTING 5.7: Implementation of the method returning channel in decimal format decoded from the hardware address.

The other methods for encoding addresses for SAMPA, FEC and branch based on the same principle as the method for channel encoding but they used a proper binary mask and executed the bitshift operation. The example of using bitshift operation for address encoding of SAMPA chip is shown in the listing 5.8.

```
unsigned short hwAdd2 = (hwAdd & sampaMask) >> 4;
```

LISTING 5.8: Code example of applying binary mask together with bitshift operation.

Injecting Black Event to the Computer Model

The data retrieved from the file with black events are stored in the custom created object type, called Signal. The table 5.2 shows the structure of the Signal class with its class members.

Signals are put into a two-dimensional vector data structure where the first dimension is a timebin and the second a channel number:

```
std::vector< std::vector<Signal> > signalArray;
signalArray.resize(1021, std::vector<Signal>(1920));
//To make the code example more readable, parameters from configuration
    file were replaced with the real values.
```

LISTING 5.9: Creating a two-dimensional vector to store data from input file.

Class Element	Description
Timebin	Stores the information about timebin the signal was detected according to the data from input file containing black events.
Address	The original hardware address read from input file.
Signal Strength	The value of charge.
Channel Address	Decoded channel address.
SAMPA Address	Decoded SAMPA address.
FEC Address	Decoded FEC address.
Branch Address	Decoded branch address.
Constructor	Standard and special constructors.

TABLE 5.2: Structure of the Signal class.

The Data Generator is reading the right location from the vector while iterating the channels to retrieve the proper signal object. The signal object is the basis for creating the sample object which is sent to the SAMPA.

5.3 SAMPA

There are totally 60 instances of the SAMPA chip in the standard setup of the computer model with 12 FECs. SAMPA chips are configurable by the configuration file and all the parameters related to the SAMPA are shared between each SAMPA instance.

According to the standard configuration of the computer model, each SAMPA chip has 32 input channels, called readout channels. Each channel can sample 10 bits of data with a frequency of 10 MHz. The readout channels read data in parallel and asynchronous. After collecting 1021 samples per readout channel, SAMPA creates one data packet per channel with the samples read by this channel. The time it takes to collect 1021 samples is called a time window. The data packets are stored in the output buffer where they wait for departure. The data packets are sent through four serial links. The data are sent out in parallel but the time of sending each data packet can vary depending on the size of the particular packet. The size of the packet depends on the number of samples collected by the particular readout channel.

The description given above of how SAMPA works is mentioned to help to understand the implementation of the SAMPA chip in the computer model. More detailed description of the SAMPA chip is given in section 3.3.5.

5.3.1 Reading Input Data

SAMPA reads data, generated by the Data Generator module, using "t_source" thread. The main part of the thread is an infinite while-loop which is executed during the whole simulation time. To present the algorithm implemented inside the while-loop, in clear way, the instructions are described in the list below, in the order of execution:

1. Wait a certain amount of time

   The first thing SAMPA does, is to wait for the Data Generator which needs time to send samples for the first timebin. The waiting time is based on the sampling frequency and allows to synchronize SAMPA with the Data Generator module.

2. Read samples from the input channels

   A for-loop is used to iterate all 32 input channels. A sample of data is read from each channel. If data come from the Black Event Data Generator then, zero-suppression is performed to discard samples containing only noise. The implementation of the zero-suppression algorithm is described in subsection 5.3.3. If samples were generated by the Static Data Generator or the Gauss Data Generator, there is no need to perform zero-suppression, because the occupancy is already taken into account by the data generator module.

3. Save sample in data buffer

   Samples are saved in the data buffer. There is one data buffer per readout channel.

4. If all sampling for the current time window was performed, start sampling for the new time window

   After reading samples 1021 times from each readout channel, SAMPA creates data packets which are ready to be sent and starts sampling for the next time window. The procedure starts again from the first point.

The reading of the samples from input channels is performed in parallel so there is no waiting time while SAMPA jumps from one channel to another. After reading data from all input channels, SAMPA waits for the next sampling. The waiting time is set by the configuration file and it is based on the frequency of sampling. The waiting time is equal

to the waiting time used by the Data Generator module and it is one shared parameter in the configuration file.

The data buffer is implemented as an array of lists. There is one data buffer per readout channel and the array makes it easy to navigate to the proper buffer.

5.3.2 Creating Data Packets

When all samples within one time window are read, the "makeHeader" method is called. The method iterates data buffers for each port to find the number of stored samples for the given time window. This number is used to create a data packet.

The data packet used in the computer model was simplified compared to the real hardware model. The data packet in the simulation consists only of a header, the payload is omitted. There is actually no need in the simulation to take care about content of the samples. To estimate buffer usage and the time it takes to send data between modules, it is enough to know the size of the data.

The method creating the data packet - header, counts the number of samples retrieved during one time window and puts the number of samples into the header. Knowing the number of samples per data packet and that one sample has a size of 10 bits, the size of the whole data packet can easily be calculated on each step of the simulation. The structure of the Packet class is shown in the table 5.3.

Class Element	Description
Packet ID	Each packet get its unique id. The id is an integer generated by a counter inside the SAMPA. The id makes it easier to debug the simulation and allows to track precisely path of each packet.
Time Window	Stores the information about during which time window the data packet was generated.
Channel Address	Decoded channel address.
SAMPA Address	Decoded SAMPA address.
Number of Samples	Decoded branch address.
Buffer Overflow	Some samples were discarded due to buffer size limitation.
Constructor	Standard and special constructors.
Output operator <<	Overloading of output operator is required by the SystemC library for objects to be sent through ports implemented using SystemC.
Assignment operator =	Like described above, assignment operator is also required because of the SystemC framework specification.

TABLE 5.3: Structure of the Packet class

5.3.3 Implementation of Zero-Suppression

In the real, physical detector, lots of samples contain only noise. The samples containing data from collisions make about 15 - 26% of all samples [24]. The noise is removed to limit the amount of data sent through the readout electronics. Removal of noise is realized by use of the zero-suppression algorithm implemented in SAMPA chips.

The simplified algorithm of zero-suppression was implemented in the computer model. The algorithm is looking for two consecutive samples with signal strength above the given threshold. The threshold for the input data based on black events is set to 50. The figure 5.6 illustrates how the implemented algorithm works.

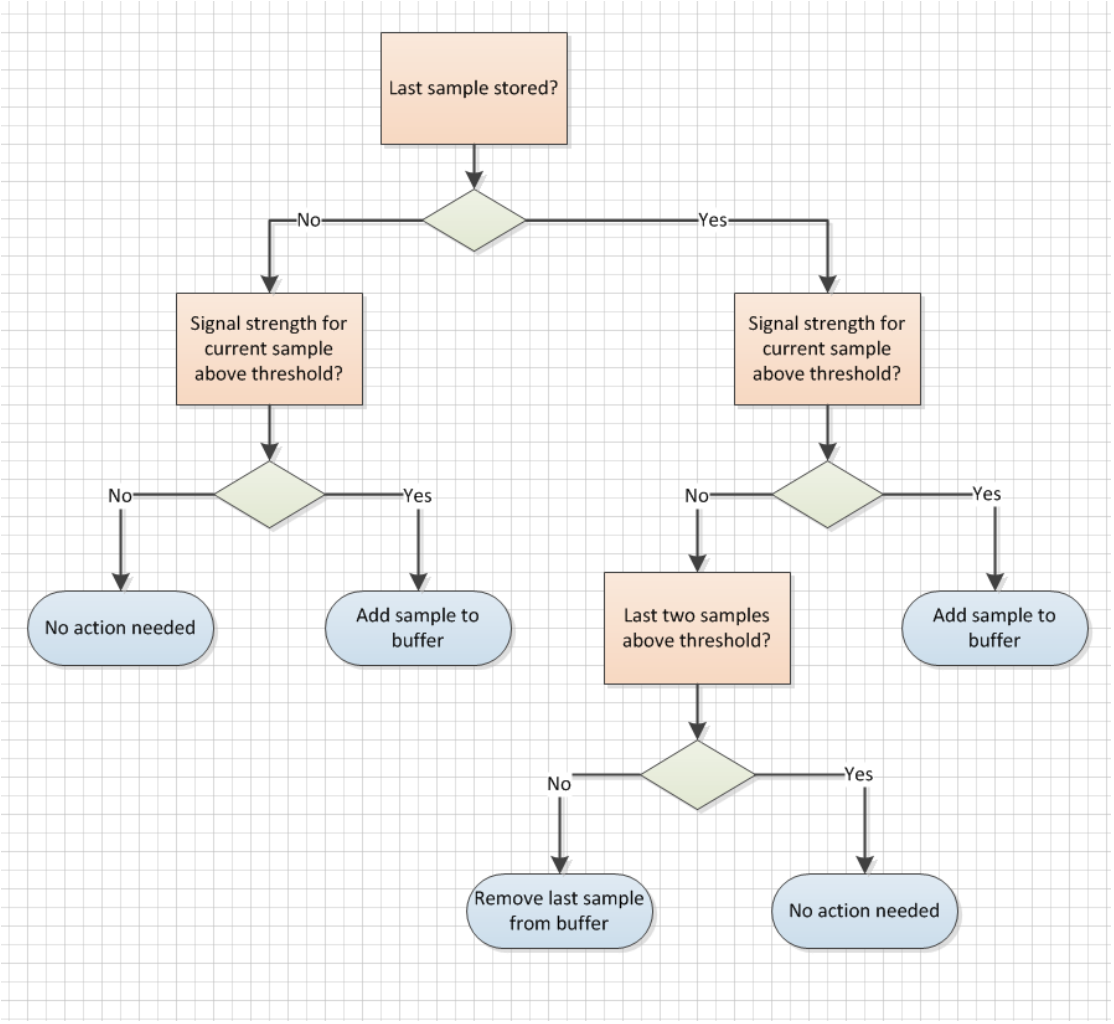


FIGURE 5.6: Decision tree illustrating the implemented zero-suppression algorithm.

The implemented zero-suppression algorithm is based on a worst case scenario. Before any sample is discarded, it is first saved in the buffer. It ensures that the usage of the buffer implemented in the SAMPA will always be as realistic as possible. The implemented algorithm was tested on different input scenarios shown in figure 5.7. The same figure illustrates also the result of applying the implemented zero-suppression algorithm.

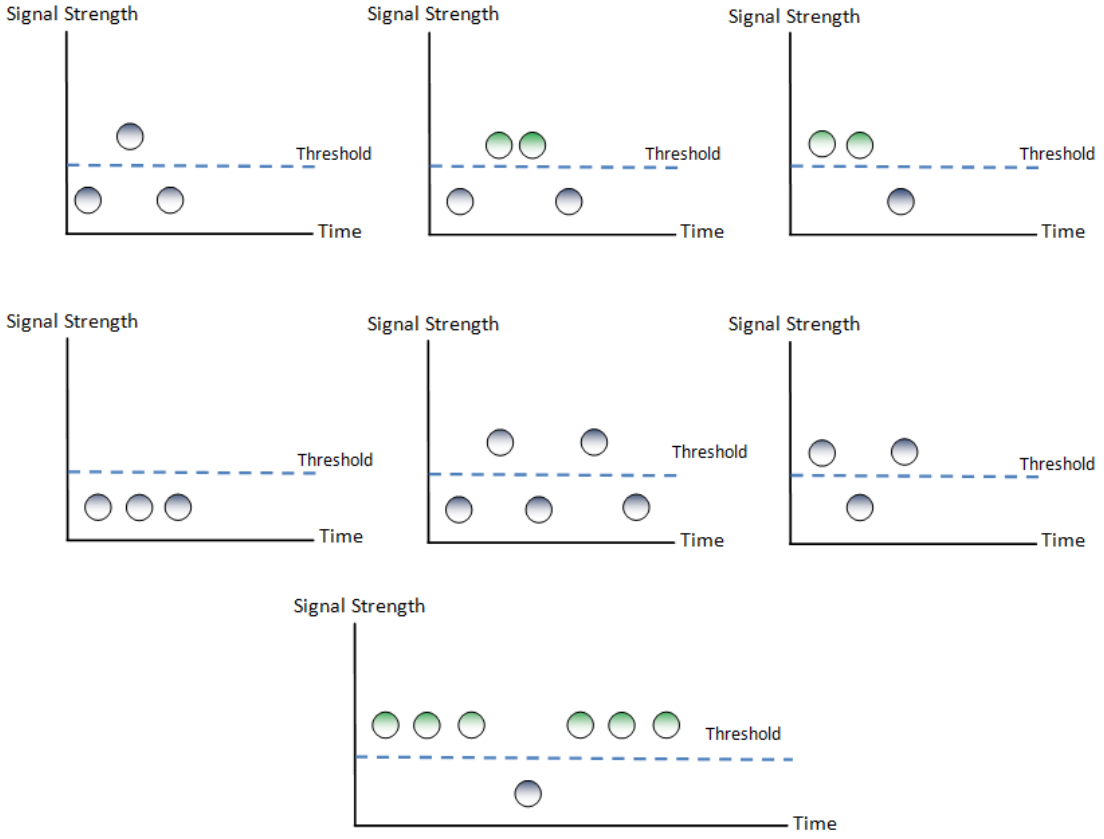


FIGURE 5.7: Illustration of results after applying the implemented zero-suppression algorithm to the different input scenarios. Only green samples were saved in the buffer after end of a time window.

5.3.4 Sending Data Out from SAMPA

The four output links of SAMPA are implemented as four independent threads. Every output thread sends data concurrently and asynchronously. To send data out of the SAMPA, every thread calls the "sendDataThroughSerialLink" method with different parameters. Every thread using this method specifies a range of input channels to read data from and index of output link to send data through.

```
1       Packet temp = headerBuffers[i].front();
2       //(x number of samples * 10 bit + 50 bit header) / 320 * 10^6 b/s *
      10^9 ns
3       wait(((0.0 + temp.numberOfSamples * 10 + 50) / 320) * 1000000,
    SC_PS);
4       //Send data to GBT
5       ports_SAMPA_TO_GBT[_outputPort]->nb_write(temp);
```

```
6       //delete header from header buffer
7       headerBuffers[i].pop();
8
9       while(!dataBuffers_queue[i].empty() && dataBuffers_queue[i].front()
    .timeWindow ==  temp.timeWindow)
10        {
11          //delete data from data buffer
12          dataBuffers_queue[i].pop_front();
13        }
14    }
```

LISTING 5.10: Part of the implementation of the SAMPA chip responsible for sending data packets to the GBT.

The "sendDataThroughSerialLink" method iterates header buffer for each channel specified within the given range. If there is a header in the header buffer, it means that a data packet is ready to send. The method calculates the time it takes to send a particular data packet and using the "wait(...)" method, it is pausing the execution of one of the four threads as shown in listing 5.10. After the calculated waiting time, the data packet is sent and removed from the data and header buffer. The execution of thread is resumed.

5.4 Implementation of GBT

The GBT module is implemented in a very basic way in the simulation. It has only two threads. One thread reads data from SAMPAs and saves them in a queue buffer. The second thread, removes data packets from the buffer and sends them to the CRU. The GBT's clock frequency is set by the configuration file. In addition, GBT introduces a latency between reading in and sending out data. GBT can count number of samples received in data packets, and write information to the log file.

The functionality of GBT can be simply extended by implementing more details to the GBT if needed.

5.5 Implementation of CRU

There are three different implementations of the CRU. The most relevant design of the CRU is the model which reads data from 12 FECs and sends data out by one PCIe link. This design of CRU is most supported by ALICE collaboration by now. However, before this design was chosen as a favourite implementation of the CRU, a few other

implementations were developed and tested. Other implementations are described briefly at the end of this section.

```
1  // Constructor
2  SC_CTOR(CRU)
3  {
4    SC_METHOD(prepareMappingTable);
5    SC_THREAD(readInput);
6    SC_THREAD(sendOutput);
7    SC_THREAD(t_sink_0);
8    numberSentPackets = 0;
9  }
```

LISTING 5.11: Constructor initializing all threads and methods used by the CRU.

The four methods, shown in the listing 5.11, control and steer all work of the CRU. The method "prepareMappingTable" is called only once, when the CRU module is being initialized. This method reads mapping table from the file which is used to map input channels to input fifos implemented in the CRU. It enables to change order of reading data from channels. To make this change as easy as possible, the source file with the mapping table is an Excel sheet. By using MS Excel or freeware alternatives like Libre Office Calc, it is possible to map all 1920 channels to 1920 fifos in an efficiently way.

```
1  while(true)
2  {
3    for(int i = 0; i < NUMBER_OF_CHANNELS_BETWEEN_GBT_AND_CRU *
     CRU_NUMBER_INPUT_PORTS; i++)
4    {
5      while (porter[i]->nb_read(val))
6      {
7        numberOfSamplesReceived += val.numberOfSamples;
8        //Put data packet to the appropriate fifo
9        input_fifos[((val.sampaChipId * SAMPA_NUMBER_INPUT_PORTS) + val.
     channelId) % (SAMPA_NUMBER_INPUT_PORTS * NUMBER_OF_SAMPA_CHIPS)].push(
     val);
10       //Update monitoring of buffer usage
11       cruMonitor.addPacketToBuffer(val, (val.sampaChipId *
     SAMPA_NUMBER_INPUT_PORTS) + val.channelId, sc_time_stamp().value());
12       //Write event to log file
13       write_log_to_file_source(val, i, numberOfSamplesReceived);
14     }
15   }
16   //Wait one clock cycle
17   wait(CRU_WAIT_TIME, SC_PS);
18 }
```

LISTING 5.12: Part of implementation of thread reading input data in the CRU. Some namespaces of constants were omitted to make code listing easier to read.

The "readInput" method is implemented as a thread which is scanning all 1920 input ports continuously. If there is a data packet which arrived to the port, it is read out from the channel and it is added to one of the input fifos. The "addPacketToBuffer" method of the CRU monitor is called to inform the monitor that some data were written to the buffer. The monitoring of buffer usage for the CRU is described more in section 5.6. The thread is also logging the event by calling dedicated method to update log file.

If there is no data packets to read from particular port, the thread is going to check next port and repeats the whole procedure described above.

The source code of "readInput" method is shown in listing 5.12. To make the code more readable for the purpose of this rapport, the namespaces of parameters were omitted.

```
1   if(!output_fifo.empty())
2   {
3     Packet temp = output_fifo.front();
4
5     //The real time it takes to send the packet through one DDL3 link
    or PCIe link
6     //Packet size / Throughput * 10^9 ns
7     //(x number of samples * 10 bit + 50 bit header) / 10 * 10^9 b/s *
    10^9 ns
8     wait(((0.0 + temp.numberOfSamples * 10 + 50) / 128) * 1000000,
    SC_FS);
9
10    //The worst case, first packet must be sent in 100% after that it
    can be deleted from buffer
11    output_fifo.pop();
12    write_log_to_file_sink(temp, _link);
13    cruMonitor.deletePacketFromBuffer(temp, temp.sampaChipId *
    SAMPA_NUMBER_INPUT_PORTS + temp.channelId, sc_time_stamp().value());
14  }
15  else
16  {
17    //Time it takes to jump to the next fifo
18    wait(constants::CRU_WAIT_TIME, SC_PS); //320MHz
19  }
```

LISTING 5.13: Part of the implementation of the thread which sends data out from the CRU.

The thread "sendOutput" is responsible for preparing data before sending them out from the CRU. Depending on the design of the CRU, the first instruction executed by this thread is to check if all needed data packets arrived for entire time window. There is one proposed design of the CRU which always waits for all data packets from all 1920 input channels, to sort them and after that send them. Another proposed design does

not sort data pad-by-pad but padrow-by-padrow. In this case, data can be sent out from the CRU, once some padrow has received data from each readout channel.

If there are some data packets ready to be sent, the thread copies the pointer to the data packet to the output_fifo. It is a signal for the CRU to send those data out.

The thread "t_sink_0", shown in listing 5.13, is monitoring the output_fifo, mentioned above, continuously. If the fifo is not empty, the thread calculates the time it takes to send data from the fifo and using the "wait(...)" method from the SystemC library, pause its execution. When the waiting time is over, the data packet is sent out from the CRU module. The CRU removes the data packet from the fifo buffer, updates the buffer monitor and writes the event to the log file. A data packet is never removed from the buffer before it is sent out from the CRU. It guarantees that the monitoring of buffer usage is made in a proper way.

For the implementation using PCIe as output link, there is only one "t_sink_0" thread implemented. For eight DDL3 links, CRU uses 8 such a threads to send data out from output fifos concurrently. There is also a difference in calculation of time it takes to send data via the link. The time calculation is based on bandwidth for each type of link.

The only difference between CRUs reading data from 12 FECs and 16 FECs is number of input fifos. In the first case, there are implemented 1920 input fifos and for 16 FECs - 2560 fifos. Such values as number of fifos in the CRU are not hardcoded. The fifo-buffers are created dynamically based on calculation of input channels defined by the configuration file.

5.6 Monitoring of CRU

A CRU Monitor class is attached to each of the CRU instances. Every time the CRU writes or removes some data packet from the buffer, the monitor is informed by calls to the appropriate method.

The CRU Monitor stores maximal buffer usage of each fifo-buffer for each time window. These data are stored in a two-dimensional vector. One dimension determines the time window, and other the index of the fifo.

When the simulation is done, the CRU Monitor saves data into a MS Excel file. The Excel sheet makes it very easy to analyse raw data generated by the CRU Monitor, by using various formulas supported by Excel.

5.7 Implementation of Test Bench

All of the code responsible for setting up the computer model of the readout electronics and initializing test bench used to run simulation, is implemented in the main method. The main method reads the configuration file and uses the parameters provided by the file to create and to initialize all modules.

```
1  //Module creation
2  DataGenerator dg("DataGenerator");
3  SAMPA *sampas[constants::NUMBER_OF_SAMPA_CHIPS];
4  GBT *gbts[constants::NUMBER_OF_GBT_CHIPS];
5  CRU *crus[constants::NUMBER_OF_CRU_CHIPS];
```

Listing 5.14: Creation of modules by the test bench.

Listing 5.14 shows creation of all module types used in the computer model. The data generator has only one instance in the simulation and therefore, it is created and initialized statically by a call to the constructor.

Other modules like SAMPA, GBT and CRU are created dynamically. The number of instances of a particular module can be manipulated by changing parameters in the configuration file. For instance, just by changing configuration file, it is possible to change the computer model of the readout electronics to consist of for example 1 FEC (5 SAMPA chips and 2 GBTs) connected to one CRU or, for example 24 FECs connected to 2 CRUs. Number of ports and channels between module is configurable as well. All these changes do not require any modifications of the source code.

```
1  //Module initialization
2    //CRU
3    for(int i = 0; i < constants::NUMBER_OF_CRU_CHIPS; i++)
4    {
5      module_name_stream << "CRU_" << i;
6      module_name = module_name_stream.str();
7      crus[i] = new CRU(module_name.c_str());
8      module_name_stream.str(string());
9      module_name_stream.clear();
10   }
```

Listing 5.15: Initialization of the CRU by the test bench.

The three dynamic arrays are created to store pointers - addresses to the memory, to each instance of modules like SAMPA, GBT and CRU. The size of these arrays are determined in the runtime and it is based on the number of instances of particular modules supplied by the configuration file.

Each module must be initialized after creation. The listing 5.15 shows initialisation of all CRU instances. The for-loop iterates each instance of the CRU chip. Inside the loop, each instance is assigned a unique name which is used as a parameter while calling a constructor for the object.

Channels are objects which are used to connect ports of modules together. Channels are delivered by the SystemC library and there is no need to implement them manually. There are different types of channels, for purpose of this model, channels of type fifo were used. They are provided as template objects which make it possible to send a custom object type, like for instance Packet, via them.

The listing 5.16 shows creation of channels which are used to connect GBT chips with the CRU. Channels are stored in dynamically created "fifo_GBT_CRU" array and then, for-loop is used to initialize them.

```
1 //Channel initialization
2   //GBT-CRU
3   sc_fifo<Packet>* fifo_GBT_CRU[constants::
      NUMBER_OF_CHANNELS_BETWEEN_GBT_AND_CRU * constants::
      NUMBER_OF_GBT_CHIPS];
4   for(int i = 0; i < constants::NUMBER_OF_CHANNELS_BETWEEN_GBT_AND_CRU *
      constants::NUMBER_OF_GBT_CHIPS; i++)
5   {
6     fifo_GBT_CRU[i] = new sc_fifo<Packet>(constants::
      BUFFER_SIZE_BETWEEN_GBT_AND_CRU * constants::NUMBER_OF_CRU_CHIPS);
7   }
```

Listing 5.16: Creation of channels used to connect GBTs with the CRU together.

The last step to fulfil building of the computer model of the readout electronics is to use channels to connect modules with each other. The main principle of connecting modules is as follows:

The output port number 1 of module A is connected to channel X. Then, the input port number 1 of some other module B, is connected to the same channel X. It allows data flow from the port number 1 of module A to the port number 1 of module B.

```
1 //Connecting Port-Channel-Port
2   //GBT-CRU
3   for (int i = 0; i < constants::NUMBER_OF_GBT_CHIPS * constants::
      NUMBER_OF_CHANNELS_BETWEEN_GBT_AND_CRU; i++)
4   {
5     if (i != 0 && i % constants::NUMBER_OF_CHANNELS_BETWEEN_GBT_AND_CRU
      == 0)
6     {
7       gbt_number++;
```

```
8       gbt_port = 0;
9     }
10
11    if (i != 0 && i % constants::CRU_NUMBER_INPUT_PORTS == 0)//24 gbt per
      1 cru
12    {
13      cru_number++;
14      cru_port = 0;
15    }
16    gbts[gbt_number]->porter_GBT_to_CRU[gbt_port++](*fifo_GBT_CRU[i]);
17    crus[cru_number]->porter[cru_port++](*fifo_GBT_CRU[i]);
18  }
```

LISTING 5.17: Using channels to connect GBTs to the CRU.

The for-loop is used to iterate each channel. The implementation uses modulo operation to connect the proper port of the right GBT to the correct CRU module and its appropriate port. The same approach is used to connect SAMPA's port to the data generator and to GBTs.

```
1 //start simulation
2   sc_start(constants::SIMULATION_TOTAL_TIME, SC_US);
```

LISTING 5.18: Using sc_start method to start running of the simulation.

At the end, the simulation is started by calling the method "sc_start(...)" from the SystemC library. The time and unit is specified as parameters. It is possible to call the method without any parameters, then SystemC will stop the simulation automatically. SystemC is monitoring all channels and if there is no data sent via them for a certain amount of time, the simulation is stopped. However, this solution was used in the earlier phases of the development and it was a source of many problems which were difficult to discover. Because of SAMPA's sampling rate, SystemC sometimes terminated running of the simulation before the simulation was done.

Chapter 6

Results Generated by the Simulation

This chapter presents results generated by running the simulation with different implementations of the hardware model and using various scenarios of input data.

Different implementations of the Data Generator module were used to run the simulation and to measure buffer usage for the CRU in diverse conditions. Different implementations of the CRU were also tested and compared to each other. Generating results needed to run simulation many times and each running was a time consuming process. The results are presented below by ordering them by different input scenarios to make them more intelligible and clear.

6.1 Flat Occupancy

This section presents results generated by the simulation using flat occupancy of data as input. Flat occupancy in this case, means that every readout channel receives the same amount of data. The input data is generated by the Static Data Generator described in section 5.2.3. Production of data in the Static Data Generator is based on a random generator. For this reason some fluctuation between channels can occur.

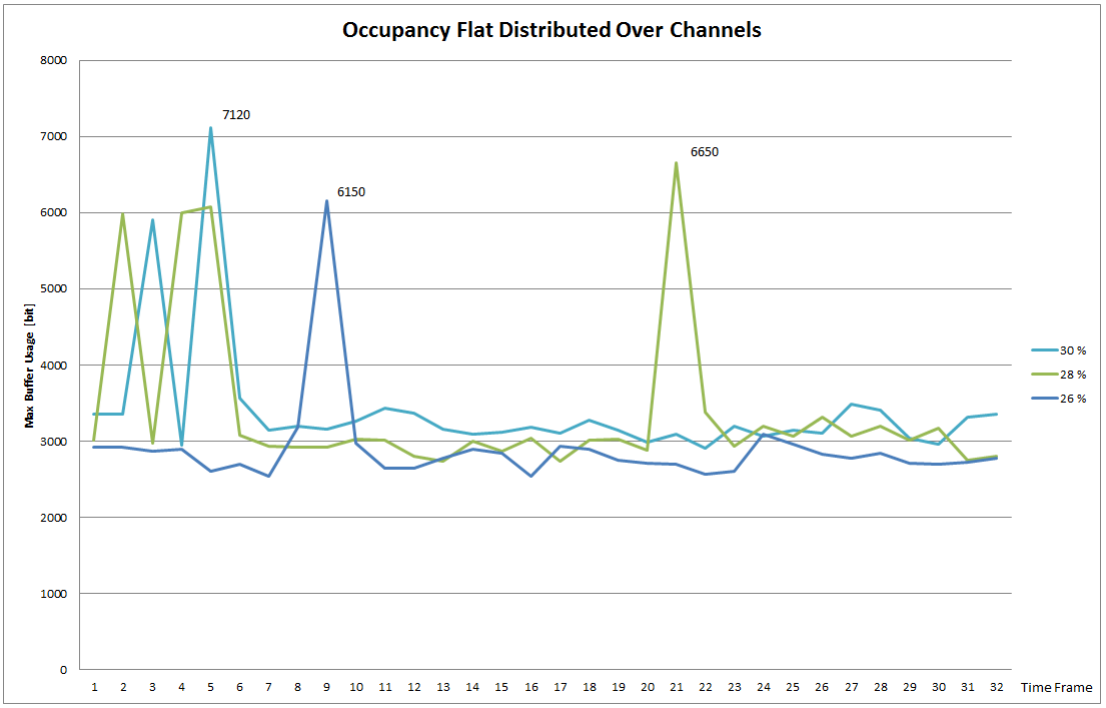


FIGURE 6.1: The maximal buffer usage for three different values of static occupancy.

The input data, which are generated based on flat occupancy, are not the most realistic scenario which can be observed in the TPC detector. However this scenario was used to test the simulation and to ensure that everything works fine before testing the computer model with more advanced input data scenarios. In addition, this scenario illustrates well how the simulation works and how the data flow through each module looks like.

The Static Data Generator was set to generate input data of 26 to 30 % occupancy. One simulation was run for each value of occupancy, six simulations totally. Each simulation was running independently and the computer model was supplied with new data during the first 30 time windows.

For each scenario, one channel with the highest peak usage of input buffer in the CRU, was picked and presented on the plot. All six scenarios are quite similar to each other, and therefore only three of them (26-28-30%) were presented on the plot to make the plot clearer. The plot is illustrated in figure 6.1. The horizontal axis of the plot is a time axis, and it shows time windows. The vertical axis shows buffer usage in bits.

The max buffer usage oscillates around a level of 3000 bits for each value of occupancy. The occupancy of 26%, the dark blue line, stays a little bit under 3 kb with the peak at 6150 bits. A little bit higher occupancy - 28%, generates higher buffer usage but still around 3 kb, and the highest - 30% occupancy - has the average buffer usage over 3 kb.

All of the three graph lines make one or more peaks. These peaks have values about two times higher than the median. It means that during these time windows, the CRU was not able to send a packet from the input buffer, before a next packet arrived. Therefore it needed to store two data packets in the buffer in the same time.

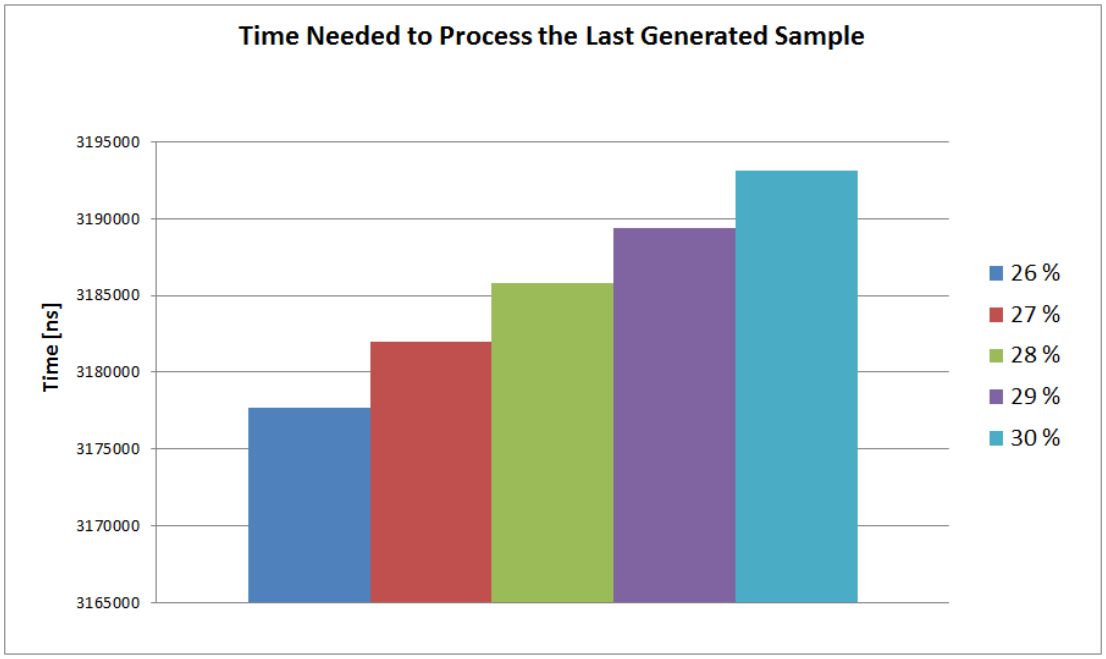


FIGURE 6.2: Illustration of time needed to process all generated samples by computer model.

The simulation can measure the time it takes to process all samples. The start point for time measurement is when the first sample is generated by the data generator. The time measurement is finished when the CRU sends out the last one sample generated by the data generator. The plot in figure 6.2 shows the time measurement for all six input scenarios. As it is shown in the plot, the higher occupancy demands more time for processing all samples.

The simulation stores also number of samples received by each GBT and by the CRU. These values were very useful under development and for evaluating the simulation. Using this information and log files generated by the simulation, it is possible to track each sample, which was quite helpful for examining data flow and for debugging. Table 6.1 shows how samples were distributed over 24 GBT for 30% occupancy.

GBTx	Number of Samples Received
GBT 0	752074
GBT 1	742781
GBT 2	741248
GBT 3	740527
GBT 4	742725
GBT 5	741938
GBT 6	742794
GBT 7	742350
GBT 8	742725
GBT 9	741786
GBT 10	741185
GBT 11	741457
GBT 12	742328
GBT 13	740965
GBT 14	741508
GBT 15	743737
GBT 16	742387
GBT 17	741936
GBT 18	742251
GBT 19	742060
GBT 20	743611
GBT 21	743303
GBT 22	742440
GBT 23	741969
CRU 0	17812085

TABLE 6.1: Number of samples received by each GBT for 30% occupancy.

The number of samples received by the CRU should be always equal to the sum of received samples by all GBTs.

One GBT forwards data from 2.5 SAMPA, which gives 80 readout channels:

2.5 SAMPA chips * 32 readout channels per SAMPA gives 80 readout channels

Each channel can read maximum 1021 samples per time window. For the simulation described above, input data were supplied during the first 30 time window. It means

that every channel can detect 30630 samples during 30 time window if the occupancy would be 100%:

1021 samples per time window * 30 time window gives 30630 samples per channel

Then, 80 readout channels can read 2450400 (80 * 30630) samples at 100% occupancy. For 30% occupancy it should give 30% * 2450400 = 735120 samples per GBT. Compared to the table 6.1, the number of received samples per GBT is around the calculated value. The small difference can be caused by the implementation of the random generator. In addition, because the creation of samples is based on probability, sometimes the occupancy can be a little bit higher and sometimes a little bit lower. Therefore, it is important to repeat the simulation a few times and compare the results. It can be extremely relevant if the buffer usage is very close to the maximum size of the buffer which is decided to be implemented in the real hardware model.

According to the simulation for 30% occupancy, the 17812085 samples were generated and received by the CRU. For 100% occupancy, the total number of samples should be equal to:

1920 readout channels * 1021 samples per time window * 30 time frames window 58809600 at 100% occupancy

The real occupancy for the simulation was:

17812085 generated samples / 58809600 maximum number of samples, gives 0.30287

It means that occupancy generated by the data generator during this run of the simulation was about 30.3%.

The simulation measures also the maximal total buffer usage for the CRU. The maximal total buffer usage describes total amount of memory used by all input buffers implemented in the CRU. The measurements are shown in table 6.2.

Occupancy	Total Memory Usage by the CRU [bit]
26%	5346220
27%	5538040
28%	5739820
29%	5929640
30%	6123580

TABLE 6.2: The maximal buffer usage for all input fifos implemented in the CRU.

6.2 Gaussian Distribution of Occupancy

According to the Technical Design Report for the Upgrade of the ALICE Time Projection Chamber [24], the average number of interactions within a time window of 100 μs is equal to 5 for 50 kHz interaction rate. It results in expected average occupancy of 15%, increasing up to 27% for the innermost pad row. Occasionally, for central collisions, occupancy can increase up to 42% during one event. For the most extreme scenario, occupancies of up to 80% may be reached. It was estimated that the probability to reach such a high occupancy is less than 0.3% [24]. The expected occupancies were estimated based on measurements made under RUN 1.

6.2.1 Gaussian Distribution of Samples over Pads

The first attempt to use a Gaussian distribution as input to the simulation, was to distribute samples over pads - channels. The average occupancy of 30% was used as the first input scenario. The same input pattern was repeated during the first 30 time windows. Figure 6.3 shows how the samples with data were distributed over input channels. Some of the channels have got flat occupancy of 50% while some other channels were empty.

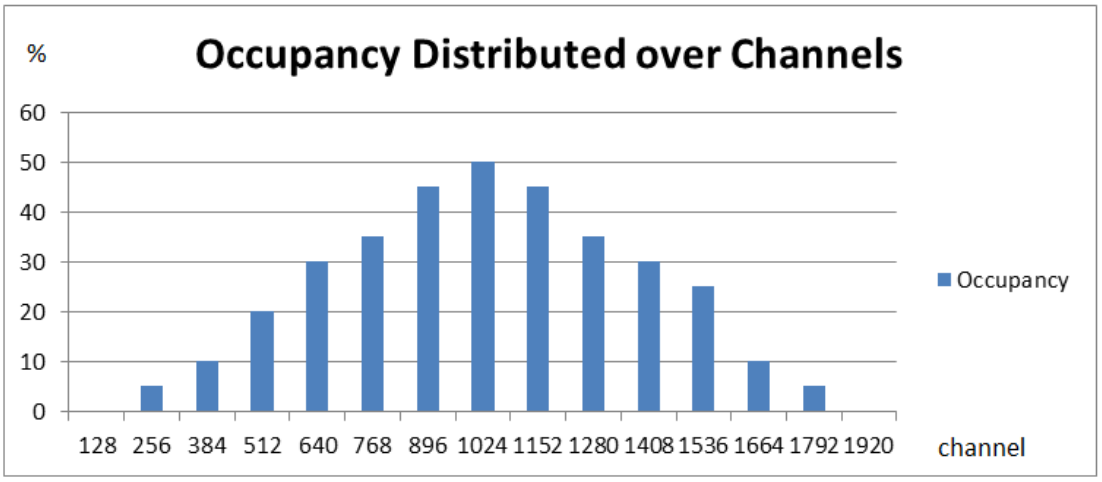


FIGURE 6.3: The distribution of samples over readout channels.

After applying this input pattern, it turned out that the buffer usage has not reached any stable level. During each time, when input data were delivered to the computer model, buffer usage was increasing until it reached the level of 36270 bits during 32. time window. Analysing the Excel sheet with the generated results, it is possible to

see that the channels which had occupancy of 0% of all time, had 500 bits saved in the buffer during 32. time window.

When the time window is done and there is no sample for a certain channel, then SAMPA creates an empty packet, which consists only of a header and no data. This procedure is described more detailed in section 3.3.5, it is important here to mention that the header has a fixed size of 50 bits.

When the input buffer in the CRU had 500 bits and it is known that this channel has never received any samples, it is clear that the buffer must contain 10 headers.

It means that during the time when buffer usage was highest - during 32. time window, the CRU had to store data from 10 time windows at the same time.

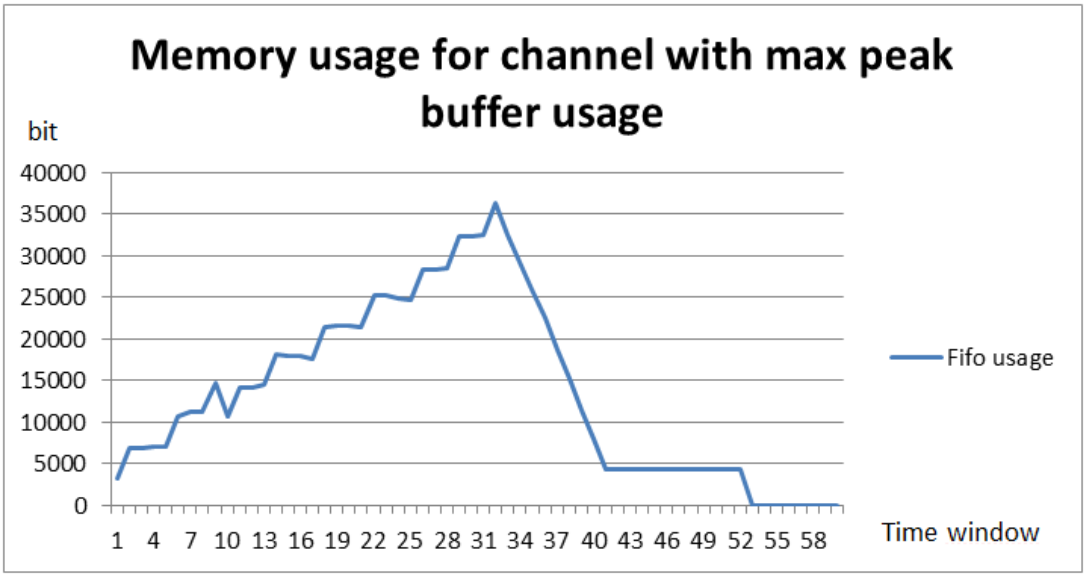


FIGURE 6.4: Maximal buffer usage for channel which had the highest peak usage of memory during run of the simulation.

Another input scenario based on Gaussian distribution was applied as well. The average occupancy was set to 22% and it was distributed over channels from 5% of occupancy to 35% with some fluctuations. The input pattern is shown in the figure 6.5.

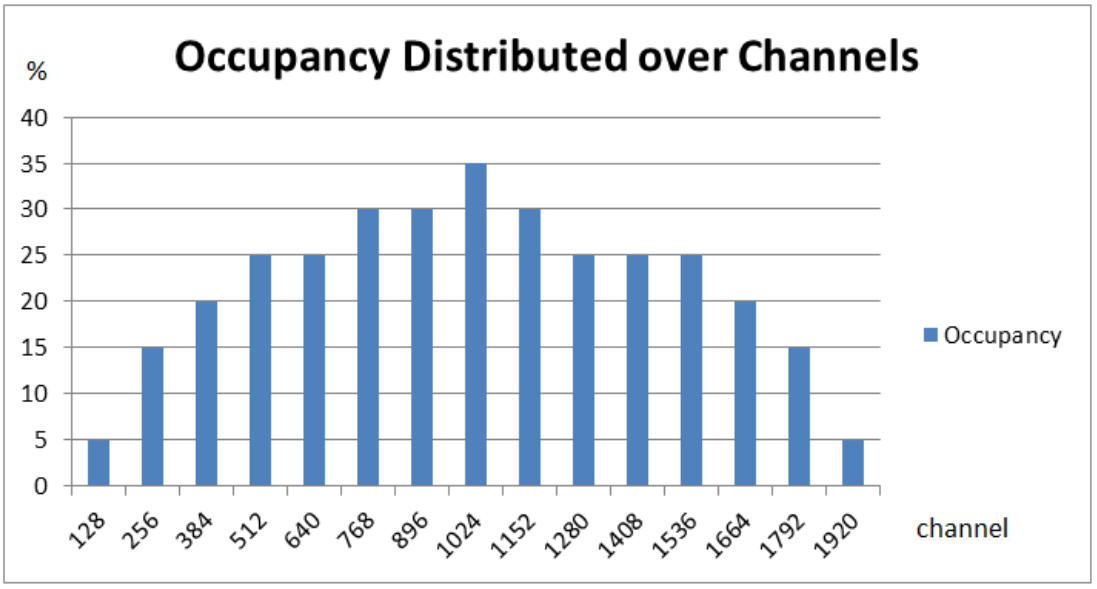


FIGURE 6.5: Input pattern for the next run of the simulation.

The maximal buffer usage was oscillating around 4000 bits. Only once during 40. time window, the CRU did not manage to send out all samples during one time window and it resulted in a peak usage of memory on the level of about 7000 bits, as shown in the figure 6.6.

Not only amount of data but also the distribution of those data over channels has a crucial meaning for memory usage in the CRU for one buffer fifo. When a flat occupancy of 30% required buffer size of about 3000 bits, shown in figure 6.1, 22% occupancy distributed over channels demands fifo size of at least 4000 bits.

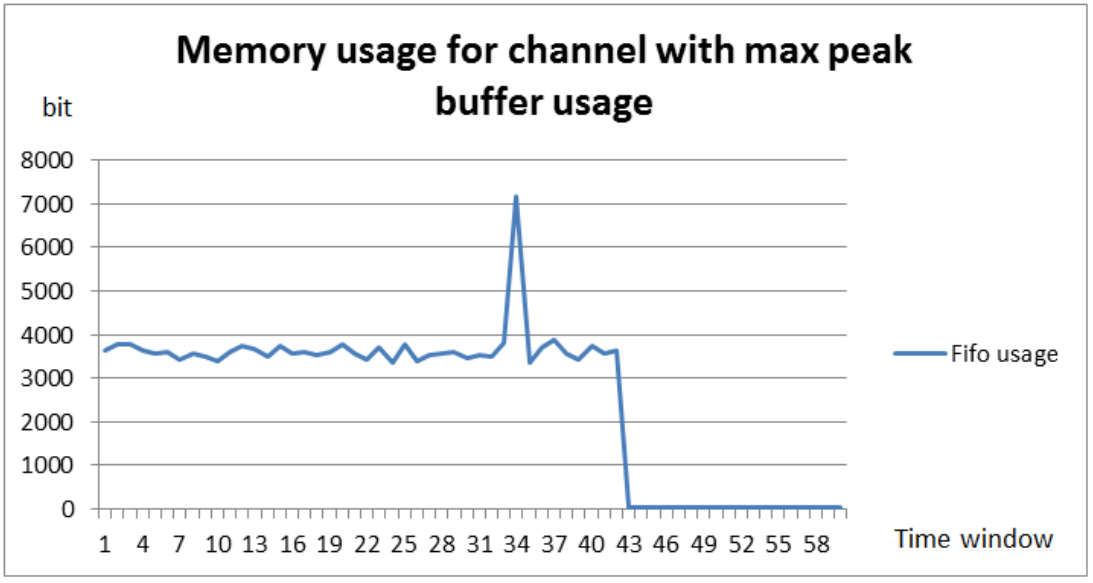


FIGURE 6.6: Buffer usage for Gaussian based input data with average occupancy of 22%.

6.2.2 Gaussian Distribution of Samples over Time

In this input scenario, input data were distributed over time. It means that every channel has got approximately the same amount of data during the time window. The amount of data was changing over the time using a Gaussian distribution. The distribution followed a Gaussian curve with 5% occupancy for the first time window, peak occupancy equal to 35% during seventh time window, and at the end 5% occupancy for the fifteenth time window. This pattern was repeated over 100 time windows.

The results of the buffer usage are shown in figure 6.7. The peak buffer usage was 7370 bits and was recorded for input channel 1905. Time needed to process all data was 10.3 ms (10316687866 ps). The average occupancy of channels was equal to 22%.

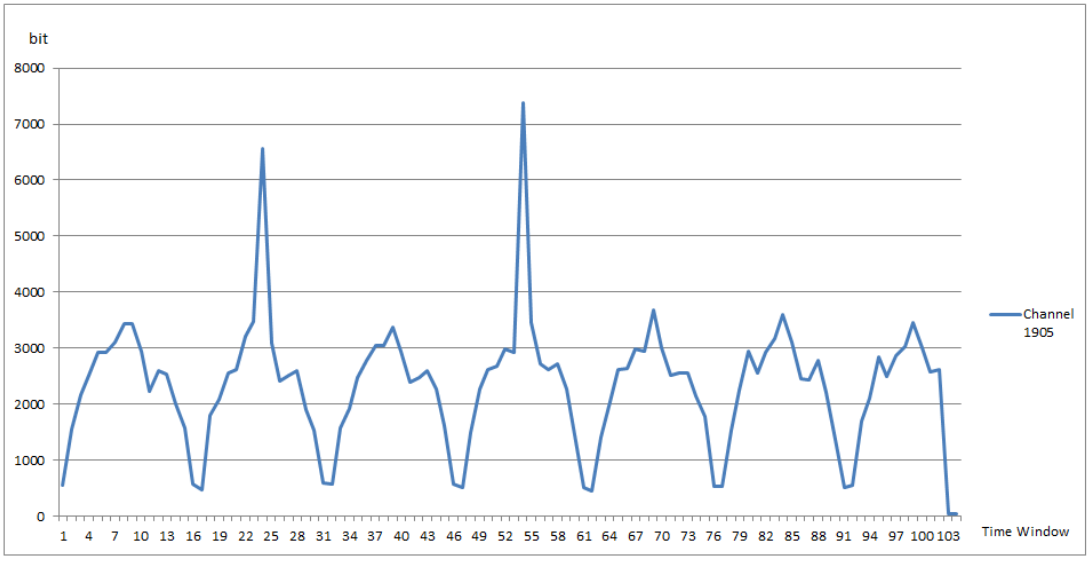


FIGURE 6.7: Buffer usage for Gaussian based input distributed over the time with average occupancy of 22%.

Using the Gauss Data Generator it is possible to generate many different input scenarios. Only steering the parameters used to make a Gauss curve, like for example a width of curve, and the top of curve, it is possible to change the results generated by simulation dramatically. Therefore, it is important to find an input scenario which is as close as possible to the real condition in the TPC detector. This part of work is out of scope for this master thesis. However, the results above show features and possibilities which the simulation gives and show how the computer model can be tested with different custom input scenarios.

6.3 Real Data as Input to the Simulation

The files with black events were supplied as input data to the simulation of the readout electronics. The concept of black events and the way how they were used as input data, is described in section 5.2.5.

6.3.1 Running Simulation for 30 Time Windows with Direct Addressing of Channels

The first simulation with real data was run over 30 time windows. In this time, input data were supplied to the model and repeated for each time window. In the first experiment of running simulation on real data, all readout channels were mapped exactly to the given

addresses determined by the input files. As mentioned already in section 5.2.5, real data were based on the readout electronics which used the predecessor of SAMPA chip which had 16 readout channels - exactly half the amount the SAMPA chip has. It means that many channels never receive any data and it resulted in low average occupancy.

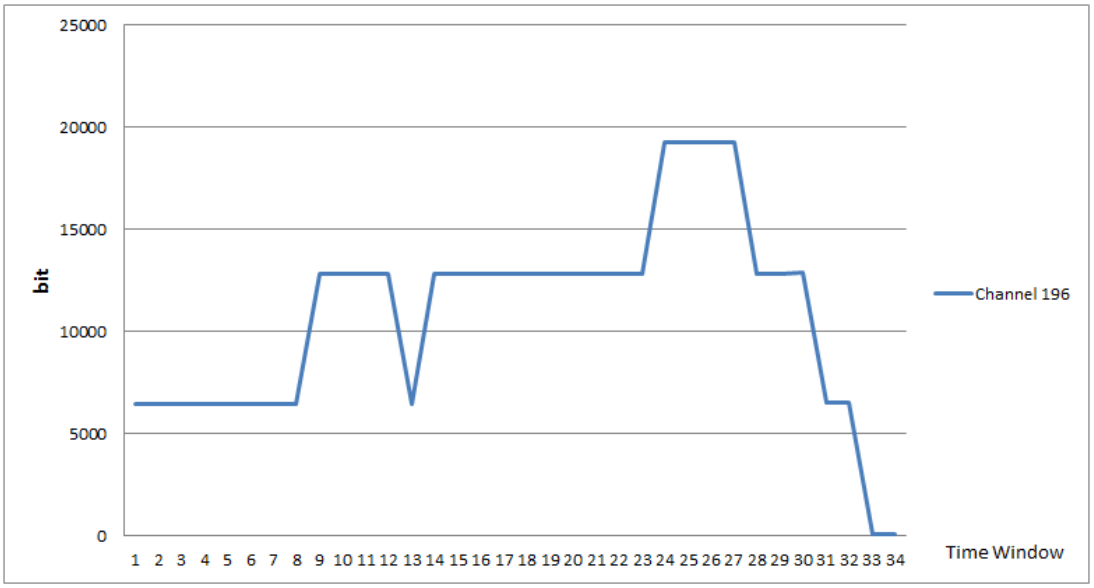


FIGURE 6.8: Buffer usage for one channel with highest peak usage.

Figure 6.8 shows the memory usage for the input fifo which had the maximal peak buffer usage. This fifo was responsible for buffering data coming from the readout channel number 196. The buffer usage for the fifo increased gradually during supplying the input data and reached 19260 bits at the highest point. When new input data were not supplied any longer, the buffer usage decreased to the minimum.

The input data for the channel number 196 is shown in the figure 6.9. The threshold for zero-suppression was set to 50.

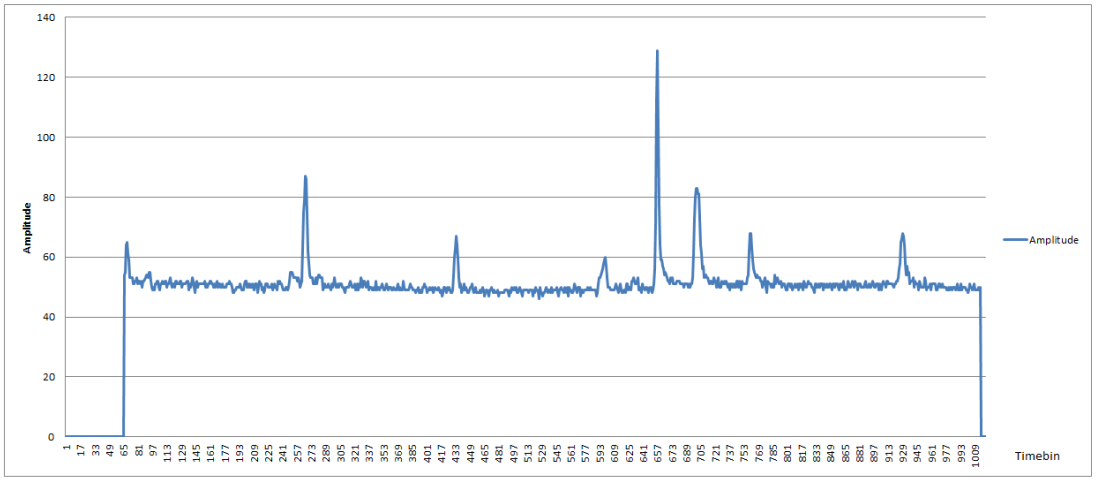


FIGURE 6.9: Amplitude for channel 196 supplied as input data based on black events.

Analysing results based on the black events as input data, without changing mapping of channels, shows that only 7 of 24 GBTs had received any data. The average occupancy was only 3.54% and the CRU had received totally about 20860 kb of data. The maximal total memory usage by the CRU, sum of memory used by all 1920 fifos, was 2154 kb. The model of readout electronics needed exactly 3.35 ms (3349032410 ps) to process the last generated data packet.

Examining the generated plot, it is difficult to recognize if the buffer usage was stabilized on some level or not. Therefore it was decided to run the simulation again, but over longer period of time.

6.3.2 Running Simulation for 100 Time Windows with Direct Addressing of Channels

The number of time windows, when input data were supplied to the model, was set to 100. The plot presented in figure 6.10 shows the memory usage for the one input fifo which had the maximal peak usage. In this scenario, the input fifo with the highest memory usage was again fifo number 196.

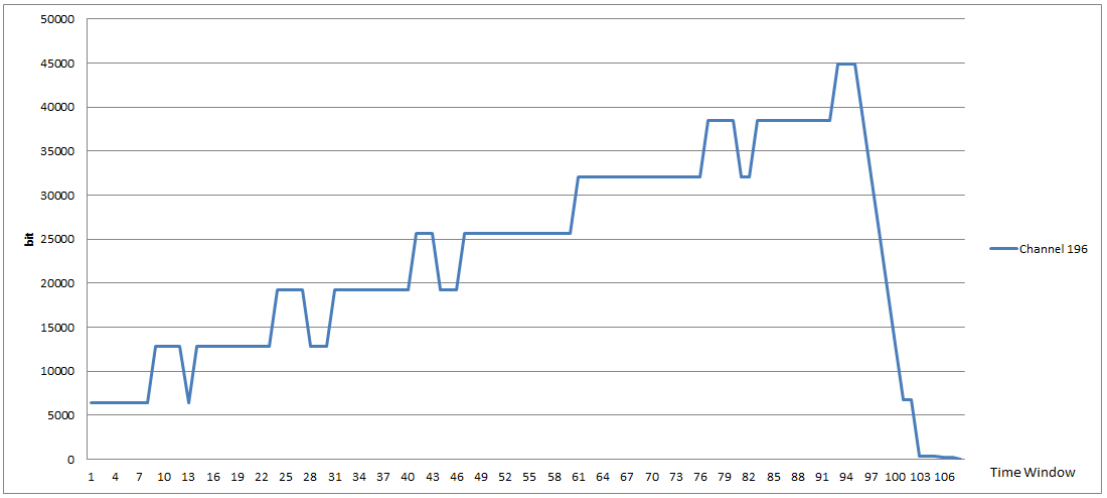


FIGURE 6.10: Buffer usage for fifo-buffer number 196 over time. Input data were supplied during the first 100 time windows.

The plot shows clearly that buffer usage has not been stable. It was increasing all the time while the input data were supplied to the model.

6.3.3 Running Simulation for 30 Time Windows with Readdressed Channels

A creative mapping of the readout channels, allowed to reuse signals, and apply them to the channels which were empty previously. The simulation was run with black events two times: the first running, where input data were supplied over 30 time windows, and the second time - when input data were supplied over 100 time windows.

For this scenario, where data were generated during first 30 time windows, the CRU received approximately 80980 kilobit totally. The remapping of the readout channels allowed to achieve occupancy of 13.7%. The occupancy is almost four times higher, but it is still relatively low compared to the expected occupancy described in the Technical Design Report, namely 15 - 27%.

All GBTs have received a certain amount of data. The total time needed to proceed all samples by the readout electronics was 3.4 ms (3364660580 ps). The maximal total memory usage by the CRU was 7908 kb.

There were three input fifos which had the same peak usage of memory. One of them was channel number 196 and the two other channels were the channels which were mapped to the same address as channel 196, and have got the same input data.

The highest needed buffer size for one fifo was 19260 bits, exactly the same value as for scenario before remapping channels. It means that even the average occupancy increased, it did not affect the buffer usage for the channel which received the highest amount of data.

The maximal buffer usage for the fifo for readout channel 196 is exactly the same as shown in figure 6.8 for the previous input scenario.

6.3.4 Running Simulation for 100 Time Windows with Readdressed Channels

The simulation with remapped channels was run over 100 time windows as well. There was no difference in maximal buffer usage compared to the simulation without remapped channels. However, the average buffer usage for the input fifo number 196 was higher in this case. The figure 6.11 presents plot with the maximal buffer usage for one channel in each time window.

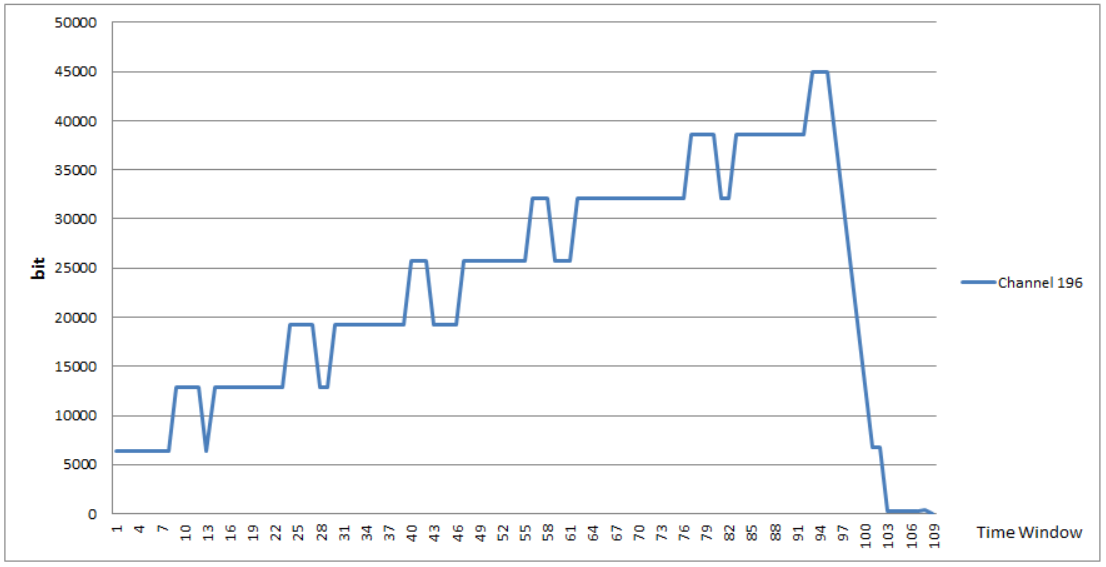


FIGURE 6.11: Buffer usage for channel number 196. Input data were supplied during first 100 time windows and they were reused by delivering them to the empty channels.

This input scenario needed 10.9 ms (10926409225 ps) to process all data packets. The CRU received 26949800 samples which corresponds to about 269938 kb of received data. The maximal total memory usage by the CRU was approximately 18294 kb.

6.3.5 Channels Mapped Dynamical over Time

Results based on the input scenarios described above, show that it is impossible to find a fixed buffer size for some of the input fifos implemented in the CRU, which could store all needed data.

Disadvantage of this testing approach was that the same input data, the same black event, was repeated over each time window. The result of that was that the same amount of data were delivered to the same channels all the time. Despite the fact that input data were based on real data, it turned out that it was not realistic to repeat those data during each time window to the same channels.

To solve this problem and to test the model with more realistic input data, it was decided to remap channels dynamically during the simulation. For each time window, the address for each channel was moved upwards by one, for instance, after first time window, channel with index 0 was mapped to be a channel 1, and channel indexed 1919 was mapped to index 0. All addresses were moved in a circle.

Figure 6.12 shows a plot illustrating maximal memory usage over 30 time windows for channels with the highest peak buffer usage. The maximal buffer usage was observed to be equal to 12140 bits. There were three channels with the same values of maximal buffer usage and they were presented on the plot.

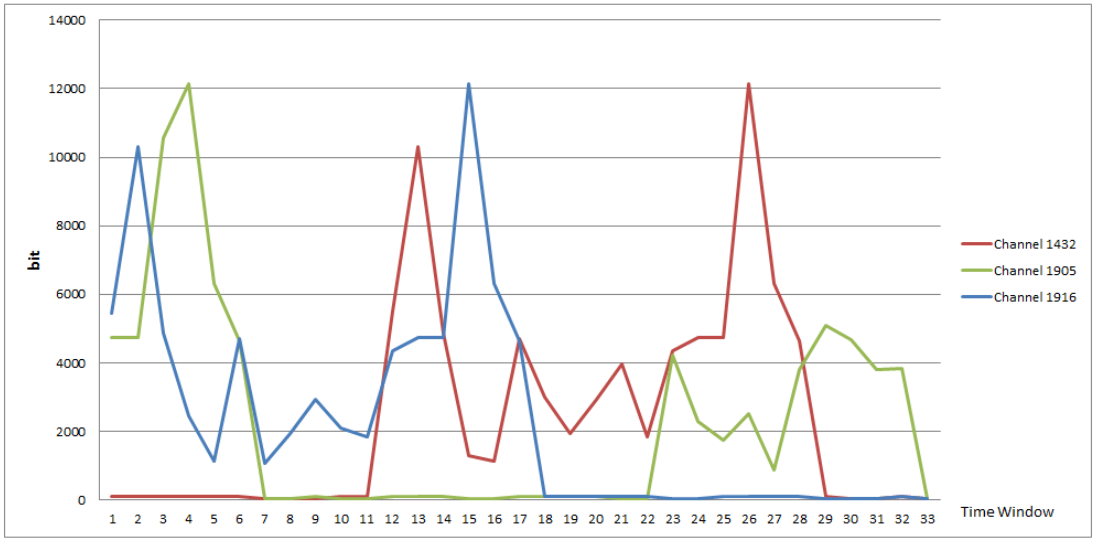


FIGURE 6.12: Maximal buffer usage for three channels with the highest peak memory usage. Addressing of channels was dynamically remapped during running the simulation.

The model of the readout electronics needed 3.2 ms (3186798080 ps) to process all data. The maximal total buffer usage for the CRU was 2923980 bits. During this run, there were generated 8084940 samples, which resulted in occupancy of 13.7%.

The plot shows one highest peak for each channel. Away from the peaks, the buffer usage oscillated between 3000 and 4000 bits.

It was decided to run the simulation with the same input scenario for longer period of time - 100 time windows, to see if the peak usage of one of the input buffers will tend to increase.

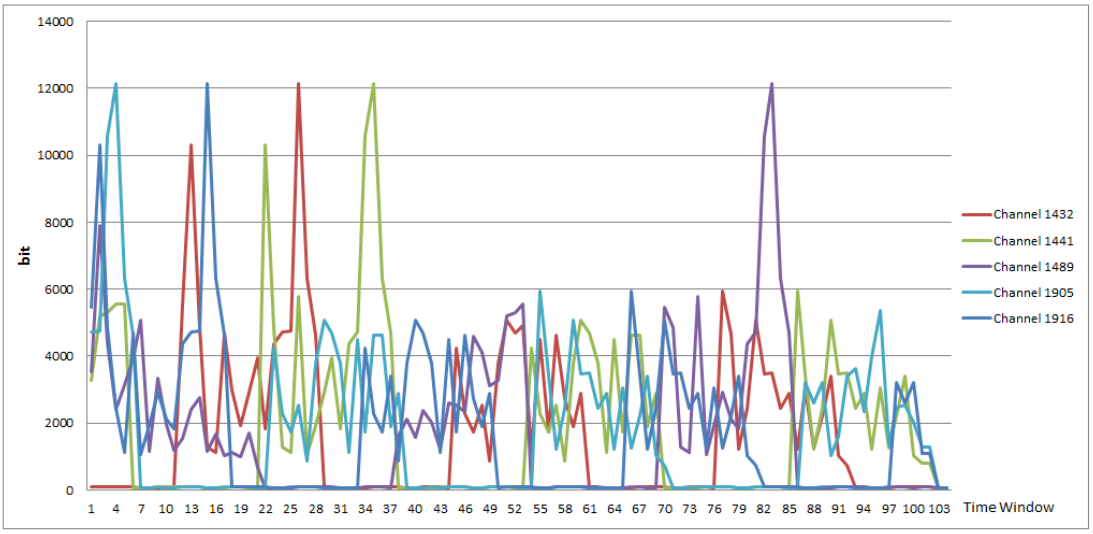


FIGURE 6.13: Buffer usage for six channels. Data were supplied over 100 time windows.

The figure 6.13 illustrates a simulation run with the same input scenario described above but for longer period of time.

The plot shows clearly that the highest buffer usage did not increase. The average buffer usage is still oscillating at the same level.

For comparison and for clearer overview all key results were put into table 6.3. The two last presented input scenarios are the two most realistic scenarios. The data used during those tests are based on the real data from previous experiments. The injection of the input data were dynamically changed over time, like during real collisions of the particles.

Input Scenario	Avg. Occupancy [%]	Total Time [ns]	Max Total Memory Usage [bit]	Max Memory Usage per Fifo
30 TW	3,54	3349032	2154160	19260
100 TW	3,54	10910780	4970500	44940
30 TW, statically remapped	13,7	3364660	7908210	19260
100 TW, statically remapped	13,7	10926409	18293950	44940
30 TW, dynamically remapped	13,7	3186798	2923980	12140
100 TW, dynamically remapped	13,7	10340415	2923980	12140

TABLE 6.3: Comparison of results based on real data.

Many files containing black events were delivered during this master thesis, and because of limited time of this project, only several of those files were used to deliver real data to the simulation. Therefore, it is strongly recommended to run the simulation using more black events as input data. It is possible that the highest buffer usage would increase for some events which could generate higher data density.

6.4 Comparison of Different Designs of CRU

On request of scientists from the ALICE Collaboration, two additional simulations were performed. The aim was to compare different design propositions for the CRU. The four CRU designs considered in this section are:

- CRU supporting 12 FECs vs CRU supporting 16 FECs
- CRU sending data out via 8 DDL3 links vs CRU sending data out via 1 PCIe link

The first test was executed to compare the two versions of the CRU which could read data from 12 or 16 FECs. It is obvious that the CRU serving 16 FECs will need more memory. However, it is not easy to predict if the change will be proportional to the number of the readout channels or maybe needed memory will increase dramatically. In addition, the simulation should show how the time needed to process all samples will change.

Figure 6.14 presents plots which compare the two mentioned CRU designs. The Statical Data Generator was used to supply input data to the computer model. The highest noticed buffer usage for CRU reading data from 12 FECs is approximately of 6000 kbit, and for 16 FECs, buffer usage reached about 8000 kbit.

The results show that buffer usage for this input scenario is proportional to the number of FECs - and thereby to the number of input channels. Time needed to process all data was longer for 16 FECs but not significantly. Because the CRU can read data parallel from its ports, the total time was affected only by additional time it takes to send more data out from the CRU.

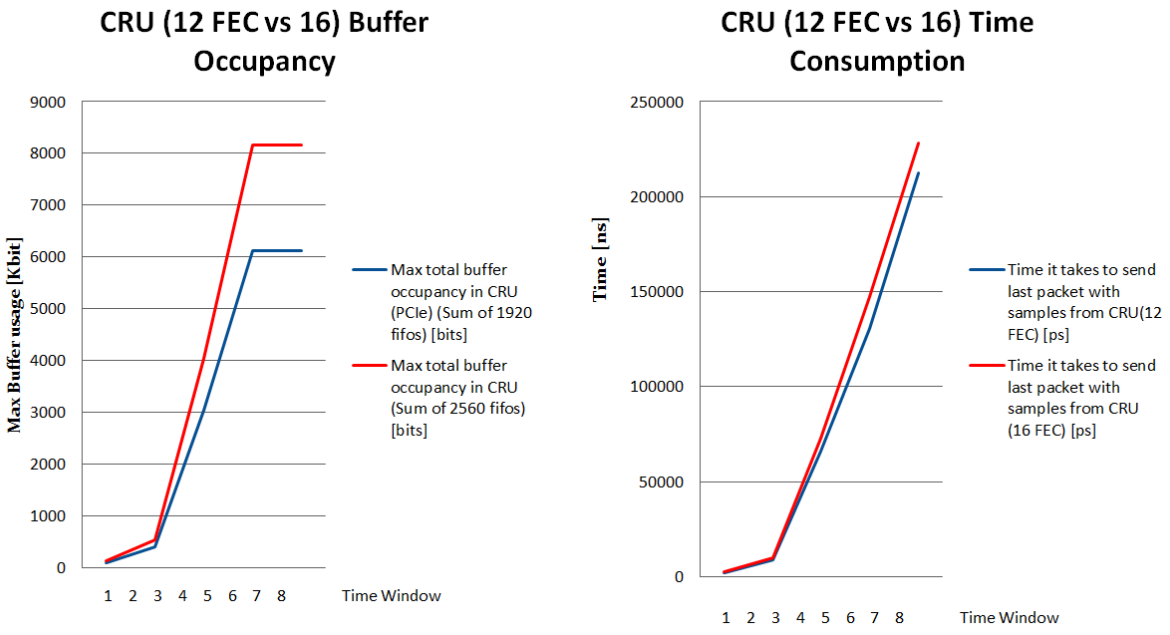


FIGURE 6.14: Comparison of two implementations of the CRU. One CRU which reads data from 12 FECs and the second one which reads data from 16 FECs.

To compare the two implementations of the CRU which use 8 DDL3 links or 1 PCIe link, the Static Data Generator was used. Figure 6.15 shows total memory usage for all input fifos of the CRU for 30% occupancy of channels. The differences in buffer usage are not huge. Memory usage for the CRU with PCIe is lower in average but around 12. time window it reached a little bit higher level than the CRU with DDL3 links. Running simulation for longer period of time shows that buffer usage for both implementations oscillates around the same values.

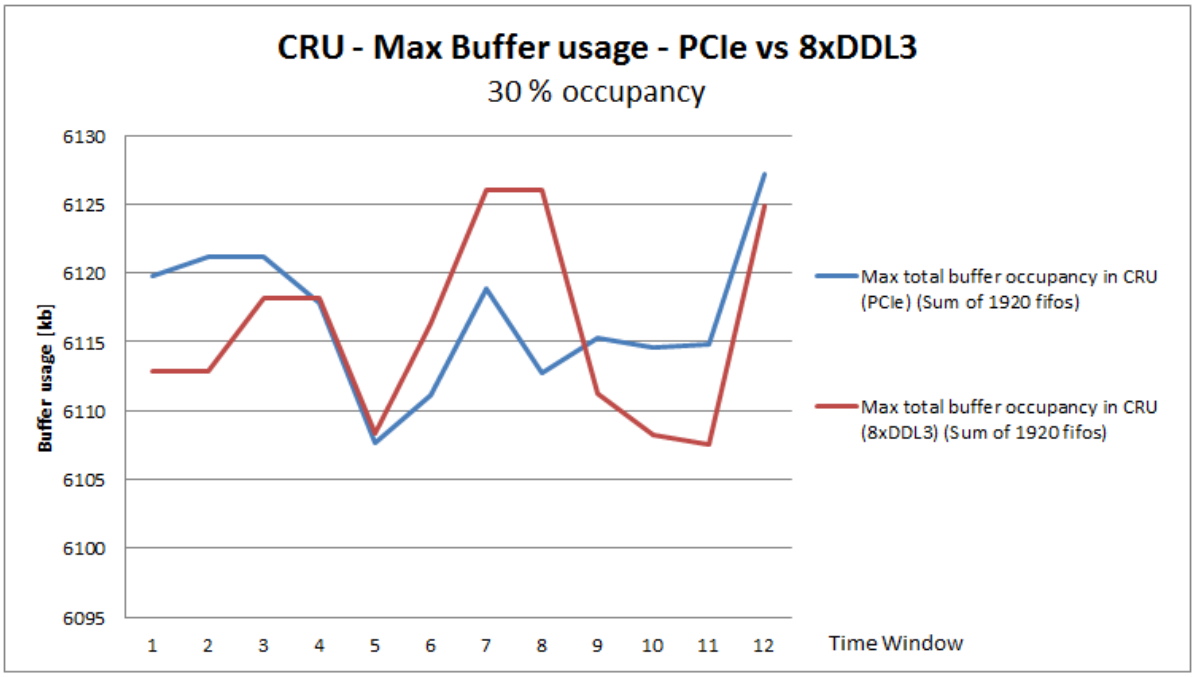


FIGURE 6.15: Comparison of buffer usage for CRUs which send data out by 8 DDL3 links vs one PCIe link.

The custom input scenario was developed to test behaviour of those two implementations in more extreme conditions. The Static Data Generator was modified to deliver 100% occupancy for the first 30 readout channels during the two first time windows. All other channels has got 30% occupancy. The plot in figure 6.16 shows that maximal buffer usage for the CRU in both cases is almost the same. No significant difference in memory usage for any of implementations of the CRU was observed.

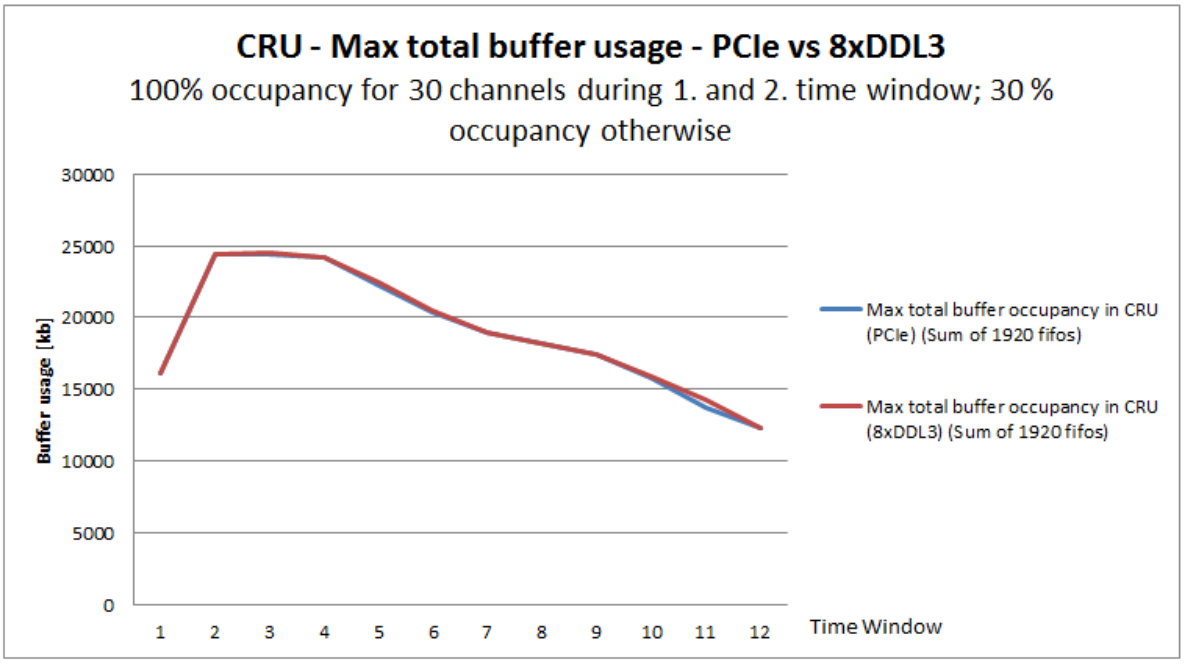


FIGURE 6.16: Comparison of buffer usage for two implementations of CRU tested during custom input scenario.

Chapter 7

Conclusion and Outlook

7.1 Summary

CERN – the European Organization for Nuclear Research is the world's largest physics laboratory which is dedicated to study the fundamental structure of the Universe. Investigating the smallest building blocks of the fundamental laws of nature demands large and sophisticated scientific instruments like particle accelerators. Scientists from CERN have developed the Large Hadron Collider, the largest and most powerful particle accelerator ever built by humans, to explore the world in very small scales.

Various properties of collisions are measured by seven detectors installed on the LHC ring. One of them is ALICE – A Large Ion Collider Experiment, which is specialized in analysing heavy-ion collisions. ALICE uses a Time Projection Chamber detector as its main detector responsible for tracking and identification of charged particles.

Under the Long Shutdown 1, the TPC detector has been prepared to the RUN 2. The readout electronics installed on the detector have been upgraded to meet new conditions introduced by higher energy and frequency of collisions. After the Run 2, the LHC will be shut down again. During this time, the accelerator and all detectors installed on it will be prepared to even higher energy of collisions. The expected data rate for TPC detector for RUN 3 is about 1 Tbyte/s. To handle the increased data rate, the readout electronics for TPC will be completely rebuilt.

The purpose of this master thesis is to contribute to designing a new readout electronics, which will be used during RUN 3, by developing computer simulation of the part of the readout system. Developing a new hardware is a complicated and time consuming process. It is almost impossible to produce hardware first and test it after that. This

approach would be a recurrent process which would consume enormous economical and time resources.

As an answer to the research question, a computer simulation of the readout electronics for the TPC detector has been developed. The computer model of the hardware and test bench allow to study data flow between implemented modules and it is a powerful tool for estimating many parameters of the designed electronics.

The computer simulation was developed using C++ as programming language, and the SystemC framework which extends the capabilities of C++ to enable hardware description by adding important concepts as concurrency, events and data types. The computer model is essentially a C++ program that exhibits the same behaviour as the designed system when executed.

The SAMPA, GBT, and CRU were implemented in the computer model as three main modules. Additional modules like Data Generator and CRU Monitor were implemented as well to enable the computer model to be simulated and to generate results based on the behaviour of the model. Each module of the simulated model has been implemented in detail and described in this report.

SAMPA is implemented as a module which samples data from Data Generator and uses them to create data packets which are sent through GBT. The GBT is implemented as a basic module which forwards data further to the CRU.

The CRU module is the main focus of this master thesis. The design of the CRU has been evolving all the time during the work on this project. Many different design solutions have been proposed by the scientists from CERN, and to help to choose the best one, many of design propositions were implemented and tested using the simulation.

Required buffer size and time needed to process data by readout electronics were estimated for four different implementations of the CRU. The CRU reading data from 12 FECs was compared to the CRU which reads data from 16 FECs. Two different interfaces for sending data out from the CRU were compared as well. The CRU with implemented output link as PCIe was compared to the CRU which uses eight DDL3 links. The estimation of needed memory by CRU is an important factor in choosing proper FPGA board for hosting the CRU.

The three different implementations of Data Generator, and some variations of them, were used to study the behaviour of CRU under diverse conditions. The computer model was supplied with input data based on: Static Data Generator, Gaussian Data Generator and Black Events Data Generator. Results generated based on different input scenarios give an indication of required memory size by CRU. The most realistic results

achieved during this master thesis are results based on black events. In this scenario the real data from RUN 1 were injected into the computer model. In addition, the input data were distributed dynamically over time to change the occupancy of channels to extend one event, supplied by an input file, over many time windows. The results were presented, described and analysed in this report.

7.2 Outlook

Many files with black events, both including real data and simulated data, have been delivered to test on different stages of this master thesis. Many of them were used as input data to the simulation. However, because of limited time of this master thesis, not all available files were used to test the computer model of the readout electronics. Therefore, it is strongly recommended to run the simulation with different black events, both those which were available during this master thesis, and those which will be created in the future.

The computer model which has been developed during this master thesis is a powerful tool for simulating and studying the behaviour of the readout electronics for the ALICE TPC detector. The simulation can be used to make the final design better but also for planning future upgrades of the electronics. The simulation is very flexible which makes it possible to have different implementations of modules and to compare them with each other. Changing of the model setup is realized by manipulating the configuration file. Just by changing parameters in the configuration file, it is possible to change number of module, like for example number of FECs, and CRUs. Many additional parameters can be changed by using the configuration file. It is also possible to extend the computer model by adding additional modules.

7.3 Reflections

The major outcome of this master thesis is a contribution to the ALICE Experiment. The work on this project was very challenging but also rewarding. The complexity of the readout systems for TPC detector, and need to learn new SystemC framework turned out to be a steep learning curve. The characterization of this project required to gain knowledge in fields of physics and electronics. Implementing the CRU was challenging due to the constantly evolving design and limited available human resources that had the required knowledge about its design.

All things considered, it was privilege to contribute to such a sophisticated experiment as ALICE is, and it turned out to be a unique experience that has been greatly appreciated.

Abbreviations

ADC	**A**nalog-**t**o-**D**igital **C**onverter
ALICE	**A** **L**arge **I**on **C**ollider **E**xperiment
ALTRO	**AL**ICE **T**PC **R**ead-**O**ut
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
ATLAS	**A** **T**oroidal **L**HC **A**pparatu**S**
CERN	The European Laboratory for Nuclear Research
CMOS	**C**omplementary **M**etal-**O**xide-**S**emiconductor
CMS	**C**ompact **M**uon **S**olenoid
CRU	**C**ommon **R**ead-out **U**nit
CSA	**C**harge **S**ensitive **A**mplifier
EMCal	**E**lectromagnetic **Cal**orimeter
FEC	**F**ront-**E**nd **C**ard
FPGA	**F**ield-**P**rogrammable **G**ate **A**rray
GBT	**G**iga**B**it **T**ransceiver
HMPID	The **H**igh **M**omentum **P**article **I**dentification **D**etector
IROC	**I**nner **R**ead**O**ut **C**hamber
ITS	**I**nner **T**racking **S**ystem
LEP	**L**arge **E**lectron-**P**ositron Collider
LHC	**L**arge **H**adron **C**ollider
LHCb	**L**arge **H**adron **C**ollider **b**eauty
LHCf	**L**arge **H**adron **C**ollider **f**orward
MoEDAL	**Mo**nopole and **E**exotics **D**etector **A**t **L**HC
MWPC	**M**ulti-**W**ire **P**roportional **C**hamber
PHOS	**PHO**ton **S**pectometer
PS	**P**roton **S**ynchrotron

QGP	**Q**uark-**G**luon **P**lasma
RCU	**R**eadout **C**ontrol **U**nit
SSW	**S**ervice **S**upport **W**heel
TOTEM	**TOT**al **E**lastic and Diffractive Cross Section **M**easurement
TOF	**T**ime-**o**f-**F**light detector
TPC	**T**ime **P**rojection **C**hamber
TRD	**T**ransition **R**adiation **D**etector

Bibliography

[1] Cern official website - the history of CERN, . URL http://timeline.web.cern.ch/timelines/The-history-of-CERN. Last visited 2015-01-15.

[2] National Centre for Nuclear Research in Swierk. URL http://www.ncbj.gov.pl/node/1700. Last visited 2015-02-16.

[3] LHC DESIGN REPORT, VOL III THE LHC INJECTOR CHAIN, 15 December 2004. URL https://cds.cern.ch/record/823808/files/CERN-2004-003-V3.pdf?version=2. Last visited 2015-01-20.

[4] CERN, CERN Brochure, Februar 2009, . URL http://cds.cern.ch/record/1165534/files/CERN-Brochure-2009-003-Eng.pdf. Last visited 2015-01-20.

[5] Cern official website - the Large Hadron Collider, . URL http://home.web.cern.ch/topics/large-hadron-collider. Last visited 2015-02-16.

[6] NASA, Goddard Space Flight Center, Cryogenics and Fluid Branch - Introduction to Liquid Helium, 9 September 2004. URL http://istd.gsfc.nasa.gov/cryo/introduction/liquid_helium.html. Last visited 2015-01-20.

[7] CERN, Cryogenics: Low temperatures, high performance, . URL http://home.web.cern.ch/about/engineering/cryogenics-low-temperatures-high-performance. Last visited 2015-01-20.

[8] Jean-Luc Caron. CERN Document Server - Cross section of LHC dipole. URL http://cds.cern.ch/record/841539. May 1998.

[9] CERN, CERN's Large Hadron Collider gears up for run 2, 12 December 2014, . URL http://home.web.cern.ch/about/updates/2014/12/cerns-large-hadron-collider-gears-run-2. Last visited 2015-01-20.

[10] ALICE Collaboration, ALICE Official Webpage. URL http://aliceinfo.cern.ch/. Last visited 2015-01-24.

[11] Cern official website - heavy ions and quark-gluon plasma. URL http://home.web.cern.ch/about/physics/heavy-ions-and-quark-gluon-plasma. Last visited 2015-01-14.

[12] The ALICE Collaboration et al. Performance of the ALICE Experiment at the CERN LHC. *Int. J. Mod. Phys. A 29 (2014) 1430044*, 2014. URL http://arxiv.org/abs/1402.4476.

[13] Wikimedia Commons - Schematics of the ALICE subdetectors, . URL http://commons.wikimedia.org/wiki/File:2012-Aug-02-ALICE_3D_v0_with_Text_(1)_2.jpg. Last visited 2015-03-18.

[14] The ALICE Collaboration et al. The alice experiment at the cern lhc. *Journal of Instrumentation*, 3(08):S08002, 2008. URL http://iopscience.iop.org/1748-0221/3/08/S08002.

[15] ALICE Collaboration. ALICE Info - ALICE Time Projection Chamber. URL http://aliceinfo.cern.ch/TPC/node/7. Last visited 2015-03-18.

[16] CERN, ALICE Time Projection Chamber, Official Webpage. URL http://aliceinfo.cern.ch/TPC/node/10. Last visited 2015-02-12.

[17] K.Ackermann et al. Results from a TPC Prototype for the Linear Collider Tracker with the MWPC and GEM Endplate Technologies. *LC-DET-2012-066*, 2012. URL http://iopscience.iop.org/1748-0221/3/08/S08007.

[18] The ALICE Collaboration et al. Upgrade of the ALICE Experiment Letter Of Intent. *CERN-LHCC-2012-012 / LHCC-I-022*, September 8, 2012.

[19] F Riggi P La Rocca. The upgrade programme of the major experiments at the Large Hadron Collider. *J. Phys.: Conf. Ser. 515 012012*, 2014. URL http://iopscience.iop.org/1742-6596/515/1/012012.

[20] Agnes Szeberenyi (CERN). Accelerating News. URL http://acceleratingnews.web.cern.ch/content/recent-progress-hilumi-project-0. Last visited 2014-04-15.

[21] P. Gasik for the ALICE Collaboration. Development of GEM-based Read-Out Chambers for the upgrade of the ALICE TPC. *10.1088/1748-0221/9/04/C04035*, April, 2014.

[22] M.C. Altunbas et al. Aging Measurements with the Gas Electron Multiplier (GEM), . URL http://www.desy.de/~agingw/trans/ps/kappler.pdf. International Workshop on Aging Phenomena in Gaseous Detectors DESY Hamburg - October 2-5, 2001.

[23] The ALICE Collaboration et al. Technical Design Report for the Upgrade of the ALICE Readout & Trigger System. *CERN-LHCC-2013-019*, July 2014.

[24] The ALICE Collaboration et al. Technical Design Report for the Upgrade of the ALICE Time Projection Chamber. To be published.

[25] C. Lippmann and D. Vranic. Alice tpc numbering conventions. November 18th, 2011. URL http://lippmann.web.cern.ch/lippmann/TPC/num.pdf.

[26] Heitor Neves Bruno Sanches. SAMPA - Digital Specifications - Presentation. Escola Politecnica Da Usp, 2014-01-29.

[27] Arild Velure. Upgrades of the ALICE TPC Front-End Electronics for Long Shutdown 1 and 2. *IEEE TRANSACTIONS ON NUCLEAR SCIENCE, 0018-9499*, 2014.

[28] Altera. Specification of Arria 10 GX FPGA, . URL http://www.bittware.com/products-services-fpga-cots-hardware/a10pl4. Last visited 2015-02-12.

[29] Altera. Documantation of Arria 10 FPGA, . URL https://documentation.altera.com/#/00035112-NT$NT00068575. Last visited 2015-02-12.

[30] J.P. Cachemiche et al. Recent developments for the upgrade of the LHCb readout system. *TOPICAL WORKSHOP ON ELECTRONICS FOR PARTICLE PHYSICS 2012*, CPPM, Aix-Marseille Universite, CNRS/IN2P3, 17 SEPTEMBER 2012. URL http://iopscience.iop.org/1748-0221/8/02/C02014/pdf/1748-0221_8_02_C02014.pdf.

[31] Filippo COSTA et al. CERN intern resources - CRU Development Status and Plans, . URL https://indico.cern.ch/event/380993/material/slides/0.pdf. 18 March, 2015.

[32] Accellera Systems Initiative. Accellera SystemC - Official Webpage. URL http://accellera.org/downloads/standards/systemc. Last visited 2015-04-19.

[33] J. Bhasker. A SystemC Primer. *Star Galaxy Publishing*, ISBN 0-9650391-8-8, 2002.

[34] David C. Black, Jack Donovan, Bill Bunton, Anna Keist. SystemC: From the Ground Up, Second Edition. *Springer US*, ISBN 978-0-387-69957-8, 2010.

[35] Deepak Kumar Tala. ASIC WORLD - SystemC Tutorial. URL http://www.asic-world.com/. Last visited 2014-05-17.

[36] Math is Fun - Webpage. URL https://www.mathsisfun.com/data/standard-normal-distribution.html. Last visited 2014-04-20.

[37] Homepage of ALICE Off-line Project, . URL http://aliweb.cern.ch/Offline/. Last visited 2014-04-28.