NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

DIPLOMA THESIS

Design and Development of a Diagnostics Client for a Beam Loss Measurement System at CERN

Emmanouil I. Angelogiannopoulos

Supervisor: **Alexis Delis,** Professor UOA

ATHENS

MAY 2013

DIPLOMA THESIS

Design and Development of a Diagnostics Client for a Beam Loss Measurement System at CERN

Emmanouil I. Angelogiannopoulos
I.D.: 1115200700077

SUPERVISOR: **Alexis Delis,** Professor UOA

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και Υλοποίηση ενός Διαγνωστικού Πελάτη για ένα Σύστημα Παρακολούθησης Εκροής Σωματιδίων στο CERN

Εμμανουήλ Ι. Αγγελογιαννόπουλος

Επιβλέπων: **Αλέξης Δελής,** Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΜΑΙΟΣ 2013

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και Υλοποίηση ενός Διαγνωστικού Πελάτη για ένα Σύστημα Παρακολούθησης Εκροής Σωματιδίων στο CERN

Εμμανουήλ Ι. Αγγελογιαννόπουλος
Α.Μ.: 1115200700077

ΕΠΙΒΛΕΠΩΝ: Αλέξης Δελής, Καθηγητής ΕΚΠΑ

ΠΕΡΙΛΗΨΗ

Ο Ευρωπαικός Οργανισμός Πυρηνικών Ερευνών, γνωστός ως CERN, είναι ένα από τα μεγαλύτερα ερευνητικά κέντρα στον τομέα της σωματιδιακής φυσικής. Ο κύριος σκοπός του κέντρου είναι η παροχή επιταχυντών σωματιδίων και άλλων υποδομών, απαραίτητων για την ερευνητική διαδικασία στον τομέα της φυσικής υψηλών ενεργειών. Τα σωματίδια επιταχύνονται μέσα από ένα σύμπλεγμα επιταχυντών και έρχονται σε σύγκρουση, έτσι ώστε να μελετηθούν τα θεμελιώδη στοιχεία της ύλης και οι δυνάμεις μεταξύ αυτών. Φυσικά, τέτοιου είδους περίπλοκα και κοστοβόρα μηχανήματα χρειάζονται έλεγχο και προστασία. Για αυτό το σκοπό, μια ποικιλία διαφορετικών συστημάτων -υλικό και/ή λογισμικό- είναι απαραίτητα. Ένα τέτοιο σύστημα είναι το σύστημα Παρακολούθησης Εκροής Σωματιδίων (ΠΕΣ) ενός επιταχυντή. Ένα τέτοιου είδους σύστημα είναι σχεδιασμένο για την μέτρηση απωλειών σωματιδίων σε έναν επιταχυντή. Ο κατάλληλος σχεδιασμός του συστήματος ΠΕΣ και αντίστοιχα κατάλληλη τοποθέτηση των μηχανημάτων παρακολούθησης, επιτρέπουν ένα μεγάλο εύρος χρήσιμων δυνατοτήτων διαγνωστικών ακτίνας και μηχανισμών προστασίας των μηχανημάτων. Η συγκεκριμένη πτυχιακή εργασία επικεντρώνεται στον σχεδιασμό και την υλοποίηση μιας εφαρμογής πελάτη, η οποία πραγματοποιήθηκε με σκοπό τον έλεγχο, την συλλογή, την αποθήκευση και την προβολή δεδομένων τα οποία στέλνονται από ένα σύστημα παρακολούθησης εκροής σωματιδίων. Το υλικό του συστήματος περιγράφεται επίσης εν συντομία.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Πελάτης-Διακομιστής, Λογισμικό για Όργανα και Μετρήσεις

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: cern, blm, bledp, διαγνωστικός, ενσωματωμένος

ABSTRACT

The European Organization for Nuclear Research, known as CERN, is one of the biggest research centers in the field of particle physics. Its main function is to provide particle accelerators and other infrastructure needed for high energy physics research. Particles are accelerated through a complex of accelerators and are brought into collision, in order to study the fundamental elements of matter and the forces acting between them. Of course, such complex and expensive machines need control and protection. For that purpose, a variety of different systems -hardware and/or software- is needed. One such system is the Beam Loss Monitoring (BLM) system of an accelerator. This kind of system is designed for measuring beam losses around an accelerator. An appropriate design of the BLM system and an appropriate location of the monitors enable a wide field of very useful beam diagnostics and machine protection possibilities. This thesis focuses on the design and development of a client application, which is realized with the purpose of commanding, collecting, storing and viewing data sent by a beam loss measurement system. The hardware of the system is also described briefly.

SUBJECT AREA: Client-Server, Software for Instrumentation and Measurements

KEYWORDS: cern, blm, bledp, diagnostics, embedded

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor at CERN, Stephen Jackson, for his comments, support and professional guidance during the implementation of this project. I would also like to thank all the members of the BE-BI-BLM team and especially Maciej Kwiatkowski for their excellent collaboration and guidance. Lastly, I would like to thank my supervisor at the University of Athens, Prof. Alex Delis for his excellent cooperation, suggestions and corrections on this work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The Large Hadron Collider (LHC) Injectors Upgrade project was launched at CERN to provide higher intensity beam for the LHC, which will allow an increase of its luminosity. A new Beam Loss Monitoring (BLM) system is under design for the monitoring of the beam losses and the machine protection. The BLM Dual Polarity (BLEDP) module is the first stage of that system. The acquisition crate will be able to host up to 8 BLEDP modules each having 8 analogue inputs to attach various detectors. The BLEDP module should be able to digitise input current in a wide range from 10 pA up to 200 mA. The total range is split into two partially overlapping sub-ranges and for each of them a different measurement method is used. The current from 100 mA up to 200 mA should be measured directly by the ADC as a voltage drop on the input resistor. This method is called Direct Analogue Digital Conversion (DADC). The current in the lower range from 10 pA to 10 mA is measured by making use of a low noise Fully Differential Frequency Converter (FDFC). The BLEDP module is equipped with a Cyclone 4GX150 FPGA which is responsible for the processing of the FDFC data. In the standalone version of the module a custom made Ethernet server is implemented in the FPGA. The server is written in the C Programming Language. [10] Each BLEDP module in the acquisition crate can have a separate Ethernet link. The measurement data is sent to a dedicated JAVA client application which is the main subject of this thesis. The task of the client is to send commands to the server, collect, view and store the different types of acquisition data. The application is a Graphical User Interface (GUI) and makes use of the Java Swing Framework and many other Java features such as multi-threading and file I/O manipulation.

1. BACKGROUND

1.1 CERN

The European Organization for Nuclear Research, known as CERN, is one of the world's largest and most respected centers for scientific research. It was established in 1954 and is situated in the northwest suburbs of Geneva on the Franco-Swiss border. CERN's main function is to provide the particle accelerators and other infrastructure needed for fundamental physics research. The output of the organization is nominally pure research and trained professional staff experienced in research activity; a large proportion of whom subsequently contribute significantly to a wide variety of economic activity typically within Europe. Whilst there is little direct financial benefit to CERN from spinoffs, CERN generates significant economic support and effectively R+D funding to the European high-technology industry and numerous small company start-ups through the technological developments required to build the large experiments. Lastly, there are numerous by-products of the research which benefit society as a whole, the most obvious examples of which are the World Wide Web, GRID and cloud computing, and also large contribution in the medical physics arena. Numerous experiments have been constructed at CERN by international collaborations to make use of them. As an international facility, the CERN sites are officially under neither Swiss nor French jurisdiction. Member states' contributions to CERN for the year 2008 total CHF 1 billion which is approximately 664 million euros. CERN is also noted for being the birthplace of the World Wide Web. The main site at Meyrin has a large computer centre containing very powerful data processing facilities primarily for experimental data analysis, and because of the need to make them available to researchers elsewhere, has historically been (and continues to be) a major wide area networking hub. Grid computing is developed at CERN, which is the combination of computer resources from multiple administrative domains applied to a common task, usually to a scientific, technical or business problem that requires a great number of computer processing cycles or the need to process large amounts of data. Grid computing is distributed, large-scale cluster computing, as well as a form of network distributed parallel processing [1].

1.2 The CERN Accelerator Complex

The accelerator complex is a succession of particle accelerators that can reach increasingly higher energies. Each accelerator boosts the speed of a beam of particles, before injecting it into the next one in the sequence [5], which takes over to bring the beam to an even higher energy, and so on. In the LHC-the last element of this chain- each particle beam is accelerated up to the record energy of 7-8 TeV (April 2012) [6].

Protons are obtained by removing electrons from hydrogen atoms. They are injected from the linear accelerator (LINAC2) into the PS Booster, then the Proton Synchrotron (PS), followed by the Super Proton Synchrotron (SPS), before finally reaching the Large Hadron Collider (LHC). Protons will circulate in the LHC for 20 minutes before reaching

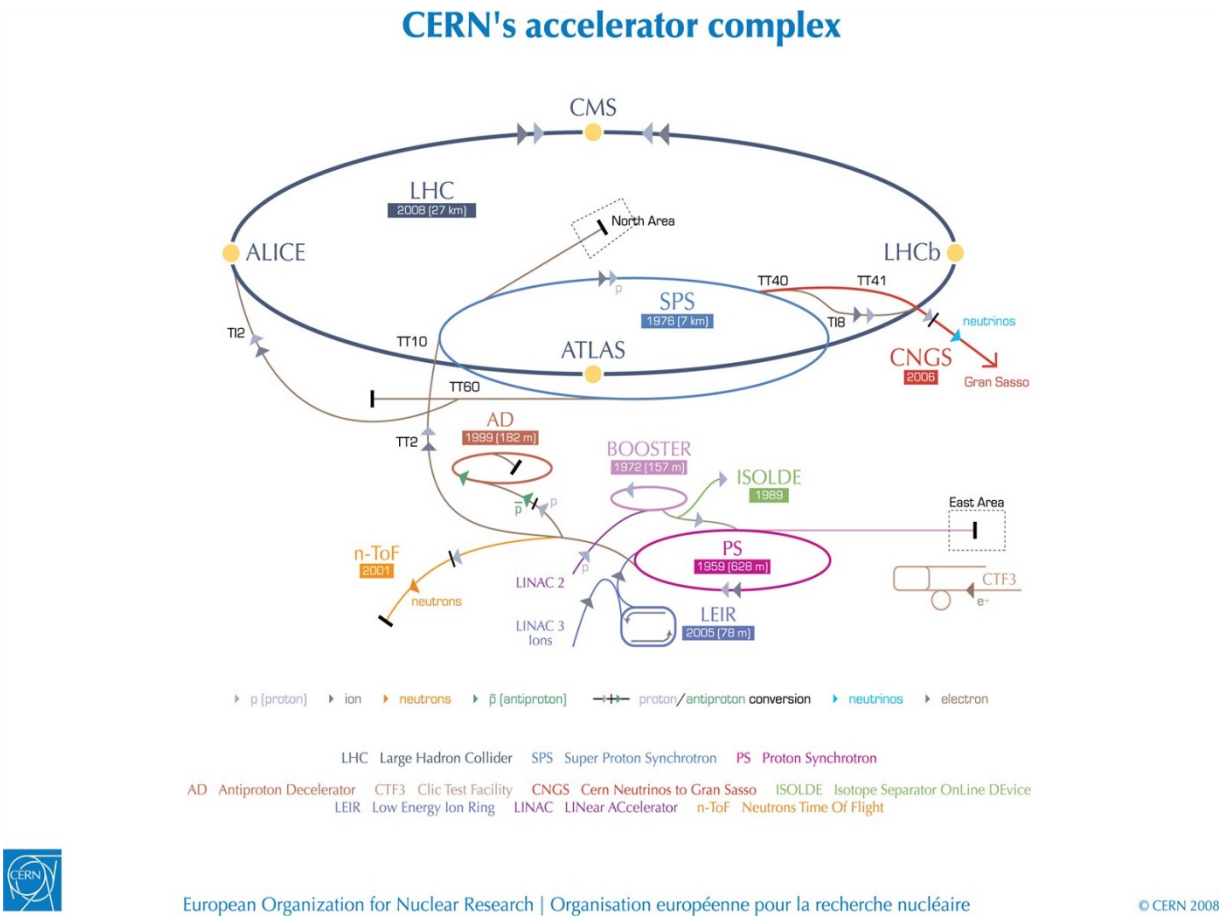

Figure 1: CERN's Accelerator Complex

the maximum speed and energy. The accelerators before the LHC are called injectors, because they are injecting the protons from one to the other to gain acceleration, until they finally reach the LHC. Lead ions for the LHC start from a source of vaporized lead and enter LINAC3 before being collected and accelerated in the Low Energy Ion Ring (LEIR). They then follow the same route to maximum acceleration as the protons [5].

The newly introduced BLM system is designed and developed only for the LHC injectors, i.e. every accelerator that boosts protons behind the LHC. We have to note here that a new linear accelerator, the Linac4 will replace the old Linac2 in the boosting process. The new BLM system is intended for use with this new accelerator as well. The upgrade will take place during Long-Shutdown 1 (LS1), which will start on February 2013 and will last approximately 18 months. This period is scheduled for the maintenance and upgrade of the accelerator complex and no beam will be commissioned until the end of it.

The client-server model discussed in this thesis is realised with the purpose of developing and testing the new BLM system, which will be used for the upgrade of the LHC injectors during the LS1 period. The current CERN accelerator complex as of April 2013 is shown in Figure 1.

2. SCOPE OF THE ASSIGNMENT

The scope of this assignment includes designing, developing, maintaining and supporting the client application for the BLM Dual Polarity (BLEDP) modules.

2.1 Necessary requirements upon the software architecture

The necessary requirements upon the software architecture include the following tasks:

- Communicating with the custom made server, implemented in the BLEDP FPGA, through the network. Each module has a separate Ethernet link and thus it is addressed individually by its unique address and port number. Each server is limited to accept only one client at a time. At this step it is necessary to decide which protocol (TCP/IP or UDP/IP) is suitable for the project's needs. The foreseen data rate should be taken into account.
- Commanding the server. After establishing connection with the server, the client should be able to send a command packet. There are two types of commands in the system. The first group of commands contains details of the acquisition request. There are also expert commands which can be sent to the server in order to set its internal registers. At this point a protocol for the commanding should be designed.
- Data sorting and processing. The server can send both mixed acquisition and status data. The protocol should be designed, so that the data type can be distinguished and processed accordingly.
- Displaying in real time two different types of data, acquisition and status, for diagnostics purposes. This is the main function of the application and the final goal of the development process.
- Storing the acquired data -in parallel with the real time view- into files for further offline analysis. The logic for the file storing follows a specific format, so it will be easier for the user to find a precise time frame from a previous acquisition.
- Designing the graphical user interface. The application must be user friendly. In order to achieve this goal it should group functionalities of a given category in a separate tab. It should also allow online real time data viewing and optional storage in the offline files. The online viewing of the acquisition and status data should be implemented by use of separate tabs. The file structure for the offline data should be proposed as well. The offline data analysis tool should be designed within the application. This stage of development makes use of the Java Swing framework. This framework provides a set of powerful and flexible components, which synthesize the look and feel of modern Java GUI applications.
- High data rate. The application will receive data stream at a high data rate (16 kbps in single-channel mode and 128 kbps in multi-channel mode). To manage online

data display and offline data storage in parallel, multi-threading technology must be engaged. Moreover, data reduction in the online view will be necessary.

2.2 Technical Tools

A variety of technical tools was used during the design and development process of the application:

- Programming Language: Java version 1.6 and 1.7.
- Linux Ubuntu version 12.04.
- MS Windows 7.
- Eclipse IDE environment, Indigo and Juno.
- Apache Subversion (SVN) as a revision control system.
- Apache Maven building tools.
- Communication Protocols: TCP, UDP.
- Java Swing Framework.
- JDataViewer, a Java based charting library developed at CERN.
- JIRA as an issue and bug tracking software.
- BDI Application Launcher, a software tool, developed inside the BI-SW section, which enables a user to start Java applications from different computers in the network. The application has to be deployed with the Ant building tool, before using it.

3. HARDWARE IMPLEMENTATION

After giving the general information about the BLM for the Injectors project, that is going to be needed in the next sessions, we are going to describe the hardware architecture and implementation of the new BLM system. This up-to-date BLM system will be included for the monitoring of the beam losses, machine protection and is still under development in terms of hardware (Electronics) and low-level software. The system is making use of reprogrammable devices, i.e. Field Programmable Gate Arrays (FPGA), to allow flexibility and target all injector's requirements. An overview of the new BLM system can be seen in Figure 2 [19].

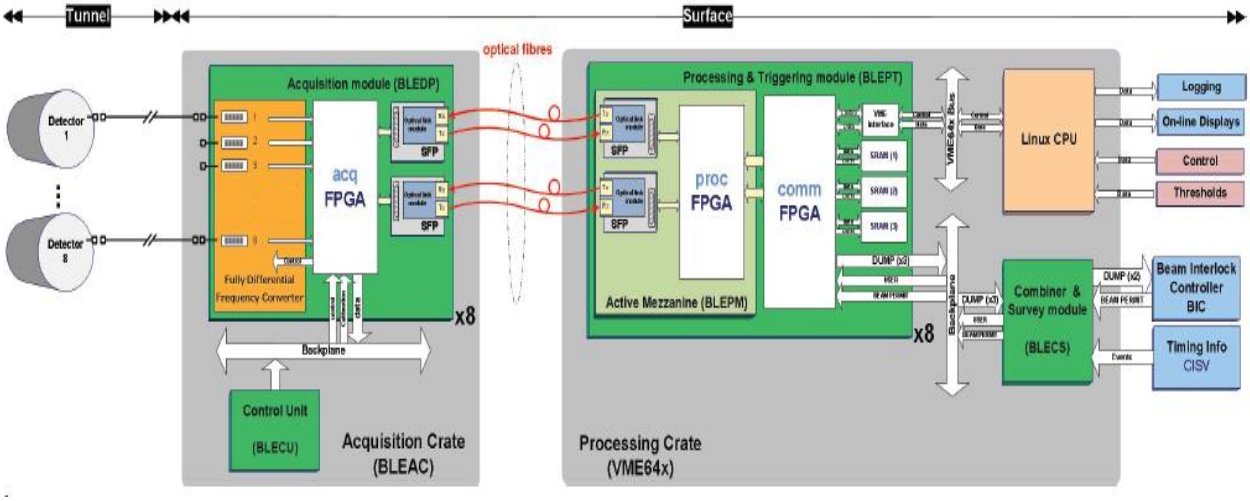

Figure 2: Overview of the new BLM System

3.1 Description of the new BLM System

The purpose of the new Beam Loss Monitoring System (BLM) is the beam setup and machine protection of the Injector accelerators at CERN. For their upgrade to higher beam energies and intensities, this new BLM system is under development. It will focus on providing faster measurement updates with higher dynamic range and the ability to accept more types of detectors as input compared to its predecessors. The detectors are connected to the acquisition part of the system. These detectors in the majority of the cases, is foreseen to use ionization chambers similar to those developed for the LHC. Nonetheless, several other types, such as secondary emission monitors, diamonds and Cherenkov detectors, may need to be used in some locations to cover particular cases. The detectors will be connected with the front-end using coaxial double-shielded cables and wherever possible, these cables will pass through enclosed cable trays, in order to make use of any possible means for noise reduction. The digitization of the detectors current input will make use of a new concept, which is currently under development. The input channel circuit should be able to measure current from 10 pA to 200 mA, a dynamic range of 10^{11}, making use of two measurement methods. These methods will be explained briefly in the next chapters. The connection between the acquisition and the processing part of the

system will be made with the use of optical fibres.

Regarding the processing part, it will combine the information gathered by each channel and keep several moving integration windows between 2 μs and 1.2 s. The calculated values for each channel will be checked continuously against predefined threshold values both at the hardware and software level, as well as being forwarded to the control centre and databases for online observation and later analysis. The final system implementation will make use of a FESA server [20] running on a Linux CPU, which will provide data to clients to deal with settings and interlocks. The new BLM system, through its direct connection to the beam interlock system, will have the ability to block all upcoming injections, when the machine protection thresholds get exceeded [19].

3.2 Acquisition Electronics of the BLM System

The acquisition electronics refer to the part of the system, which contains the digitizer modules, the control unit and the crate that provides the hosting and interconnections.

3.2.1 Beam Loss Electronic Acquisition Crate (BLEAC)

The Beam Loss Electronic Acquisition Crate is the Electronic Crate, which hosts the BLEDP acquisition modules. Its design philosophy is to give the same attention to the measurement performance as well as the reliability of the entire system. The summary of the design philosophy of the crate is explained through the following points:

- Performance improving with respect to previous systems.
- Increase the Beam Loss Measurement system reliability.
- Preference to simple architectures.
- Implementation of the design architecture independent from the component manufacturers.
- Implementation of protection and current limiting.
- Guarantee a total separation between System Functionality and System Safety Functions.
- Allowance of remote diagnosis.
- Implementation of the full remote control and calibration. [14]

The crate is based on a custom designed backplane which provides connection for 64 channels (8x8) and for the BLEDP acquisition module, the connection with the power supply, voltages and control signals. In addition, the backplane has support for direct injection of a remotely adjustable reference current with two ways: (i) via a dedicated input or (ii) via an internal current source. For that purpose, each channel's input goes through a relay contact. As a result, the system will be able to guarantee the full dynamic

range and maximum linear response of each channel and check frequently the complete channel's connection. [19]

In Figure 3 below, the Acquisition Crate and some of its Technical elements are shown.

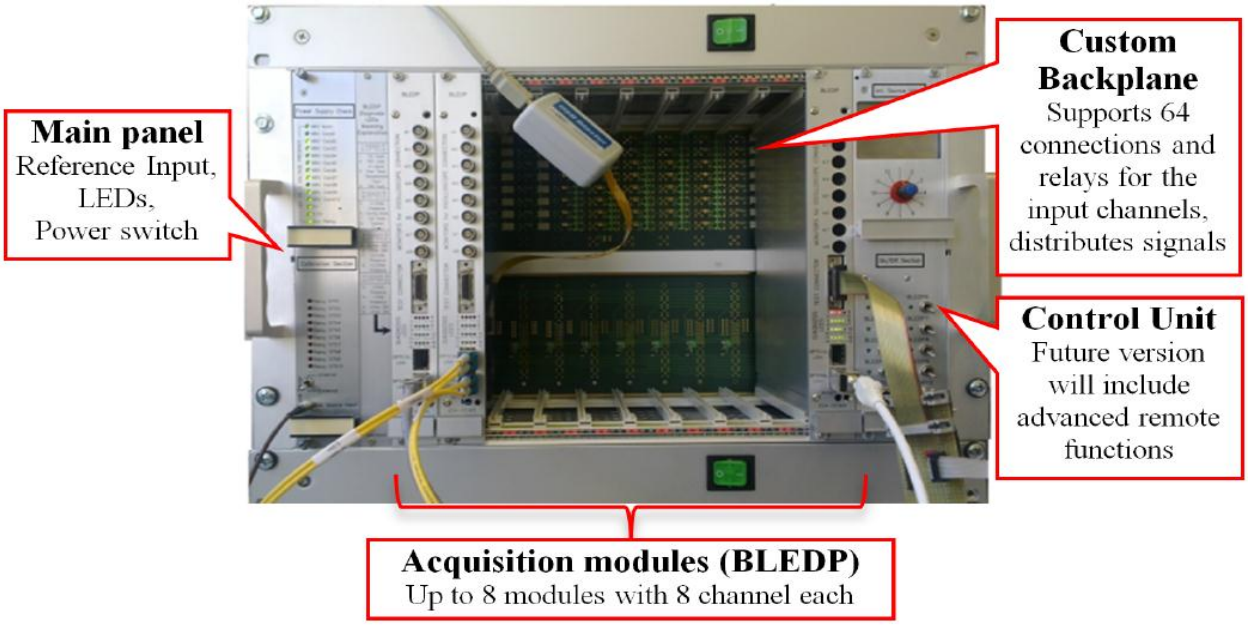

Figure 3: Acquisition Crate's Elements

The detailed description of the elements is not part of this thesis. Complete information about these technical elements of the Beam Loss Electronic Acquisition Crate can be found in the corresponding notes. [14]

3.2.2 Beam Loss Electronic Dual Polarity (BLEDP)

The BLEDP is the Beam Loss Electronic Dual Polarity Card. It is the most important part of the BLEAC and its aim is to measure with high precision the current produced by the Beam Loss Monitors. The crate can host up to 8 BLEDP cards. The BLEDP card contains an Altera Cyclone IV GX FPGA in which a Nios II soft processor is implemented. The server application implemented in this processor is written in C. Each card can acquire up to 8 input channels, so the total of input channels available on the crate can sum up to 64 (8x8). In addition, each card has an Ethernet Link, in order to be able to connect with the network for external diagnosis. The diagnostics client communicates with the cards through this link. The card implements two types of measurement: (i) The Fully Differential Frequency Converter (FDFC) and (ii) The Direct Acquisition Data Converter (DADC). Both methods produce raw digitized input at different rates:

- In the FDFC mode, the input current is integrated during 2μs period. The integrator produce count pulses which are combined with ADC values to provide precise digital values at 0.5 MSPS rate (i.e. one sample every 2μs).
- In the DADC mode the input current is converted into ADC values with the same conversion frequency of 0.5 MSPS (i.e. one sample every 2 μs).

More detailed description about these two methods will follow in the next chapter. Also, to be noted, that the raw ADC data is produced at conversion frequency of 10 MSPS (100 ns). In both methods it is then transformed by the BLEDP into processed data with a constant conversion frequency of 0.5 MSPS (2 μs). This means that one data sample is produced every 2μs and thus 500000 samples in one second. Each sample is a double word of 32 bits. A picture of the prototype BLEDP module is shown in Figure 4.

The main functions managed by the BLEDP card are the following:

- FPGA local or remote programming.
- 8 Input Analog Interfaces for Beam Loss Monitor.
- Bidirectional Optical Link.
- Power supplies with protection and diagnosis.
- Temperature and Humidity Measurement.
- Temperature and Humidity Measurement.
- ID Chip.
- Auxiliary Ethernet Link for diagnosis.

Figure 4: BLEDP Card [18]

3.2.2.1 Acquisition methods of the BLEDP: FDFC and DADC For the implementation of the electronic acquisition module (BLEDP) of the new Beam Loss Monitoring System in CERN Injector complex, a wide range digitizer card is needed. In order to reach a high dynamic measurement range of 10^{11} (10 pA to 200 mA), a mixed technique has been applied in the new BLEDP card. This technique is based on two matching principles:

- Fully Differential Frequency Converter (FDFC) circuit. In this method, the ADC con-

verter is attached to the differential output of the integrator.

- Direct Analogue Digital Conversion (DADC) circuit. In this method the ADC converter is attached to the input resistor on which the voltage drop is measured.

In this way the measurement range is split into two overlapping ranges. The FDFC is used between 10 pA – 30 mA and the DADC between 100μA – 200 mA. The analogue switch, that selects which of the two circuits is active at any given time, is automatic and controlled by the FPGA device. A module inside the FPGA is monitoring the data stream and depending on the measurements it receives, selects the most suitable measurement method for the next period. This switching between the two principles, depending on the input current, can be seen in Figure 5.

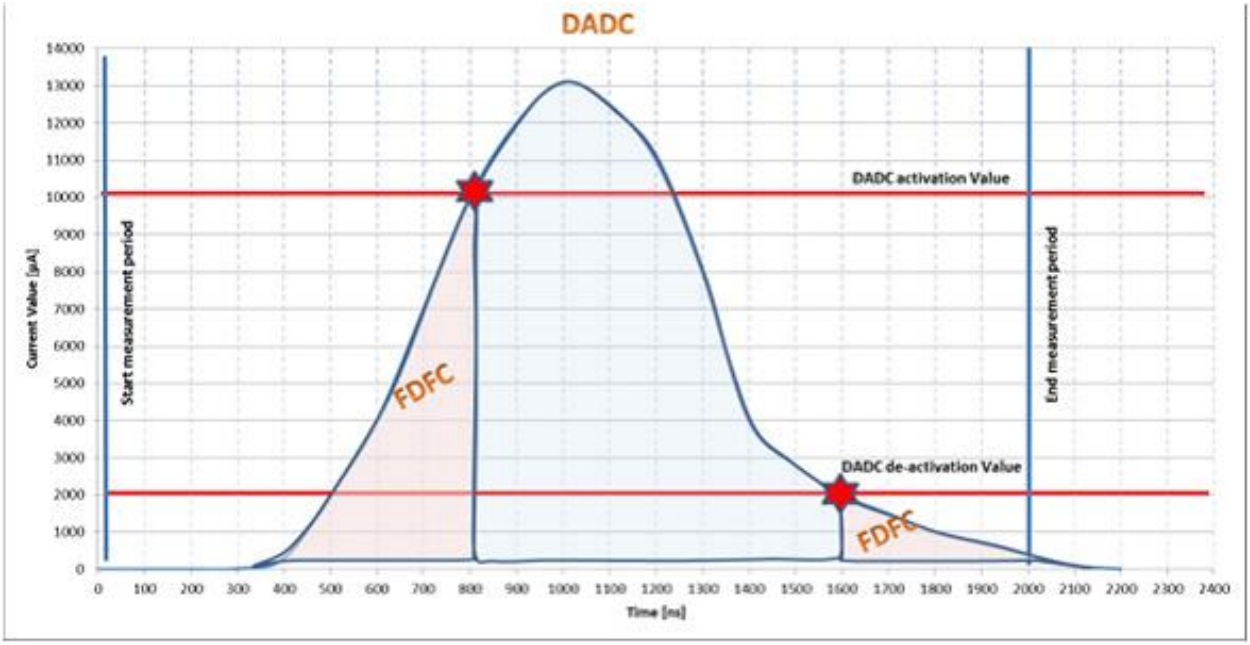

Figure 5: FDFC and DADC switch

The FDFC method is using an integrator and a status signal to select in which branch of the fully differential stage, the input current is integrated. This mechanism is shown in Figure 6.

The FDFC circuit's output is processed by the FPGA device. The outputs of both analogue comparators are complemented by the ADC samples to reach higher measurement precision. The calculation of the integrated loss is triggered by the FPGA over a 2 μs period. [18] This also marks the integration period, in which the measurement result is produced. In general, most of the operations are handled by the FPGA. So, the FPGA defines when the acquisition period starts or stops. It also keeps a count of the pulses occurred in the acquisition period and clocks the ADC circuitries in order to make differences of the recorded ADC values. Finally, it processes the data and provides this 2μs integral per acquisition channel.

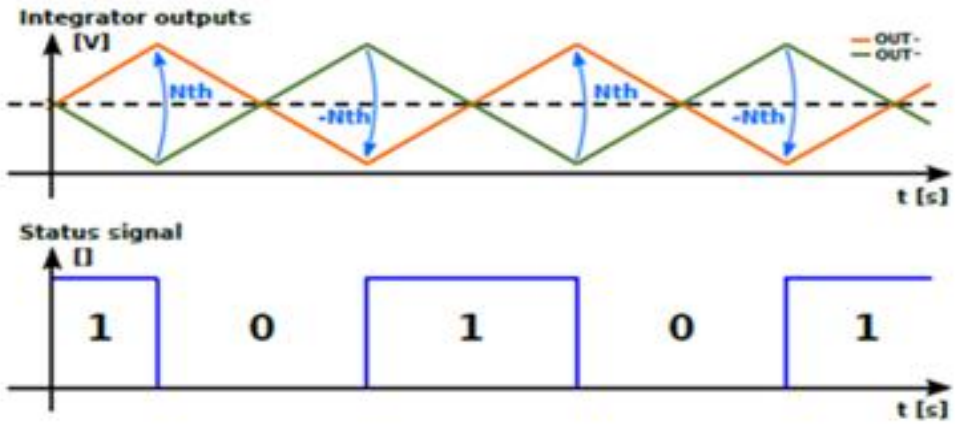

Figure 6: Fully Differential Integrator

On the other hand, the principle behind the DADC measurement method is much simpler. The input signal is sent directly to a digital converter, in order to produce the digital output. As already mentioned, the switching between the two methods is automatic and depends on the input current.

4. CLIENT IMPLEMENTATION

In the previous chapter we discussed briefly about the new Beam Loss Monitoring System for the LHC Injectors and more specifically about the acquisition part of the system, for which the communication client is designed. In this chapter, we outline the design and development of the client by first introducing the motivation for building this application. We then advance with a quick introduction of the FPGA firmware and embedded software. Afterwards, a full description of the client-server architecture protocol is analyzed. This section includes the definition of the commands sent by the client, the encapsulation of the frames sent by the server, the types of data sent by the server and the development steps of the architecture protocol. Finally, the last subchapter follows a detailed outline of the user interface and in general of the design and implementation of the client.

4.1 Motivation for this Client Development

The final implementation of the new Beam Loss Monitoring System is shown in Figure 2. The detectors are connected to the input of the new Beam Loss Acquisition Crate, which hosts up to 8 acquisition modules. Each module is equipped with two small form-factor pluggable (SFP) transceivers. The first one is a 1310 nm of type LX used for single mode fiber communication and the other one a 1000Base-T used for Gigabit Ethernet [21]. In the final system the optical SFPs will be used to communicate directly with the processing electronics (VME64x crate). Then, through the VME64x Bus, a module with a FESA server running on a Linux CPU, will be used to provide data to accompanying clients.The purpose of this communication is for the clients to deal with different settings and interlocks of the system. This is the standard way for most of the accelerator systems at CERN.

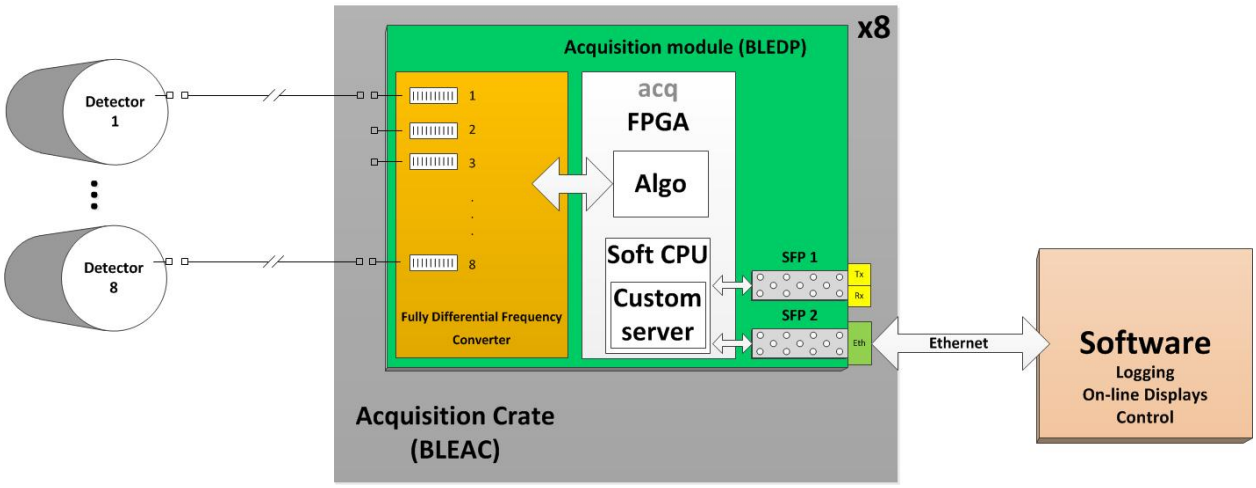

Figure 7: Test Acquisition System and Accompanying Diagnostics Client

At the time of development of the acquisition electronics, the processing electronics were not yet available. For that purpose, a test system implementation was introduced. This test system is based on a client-server model using Gigabit Ethernet readout, instead of fiber optics in the final system. A Nios II soft-core CPU is implemented inside the reconfigurable

FPGA device giving the possibility of a custom made server. This way a client application can communicate directly with the modules in order to collect and manipulate the different types of data provided by the server. The aim of this test system is the development, test and validation of the new acquisition system, as well as to later serve as a standalone measurement system. A graphical idea of how this new test system looks like is shown in Figure 7.

4.2 FPGA Firmware and Embedded Server

A block diagram of the Field Programmable Gate Array (FPGA) system architecture is shown in Figure 4.2. This architecture is used for the Gigabit Ethernet Readout in the BLEDP firmware. It consists of two parts. The first one is the custom User logic and the other one a System-On-a-Chip (SOC), generated by the Altera tools. [10]

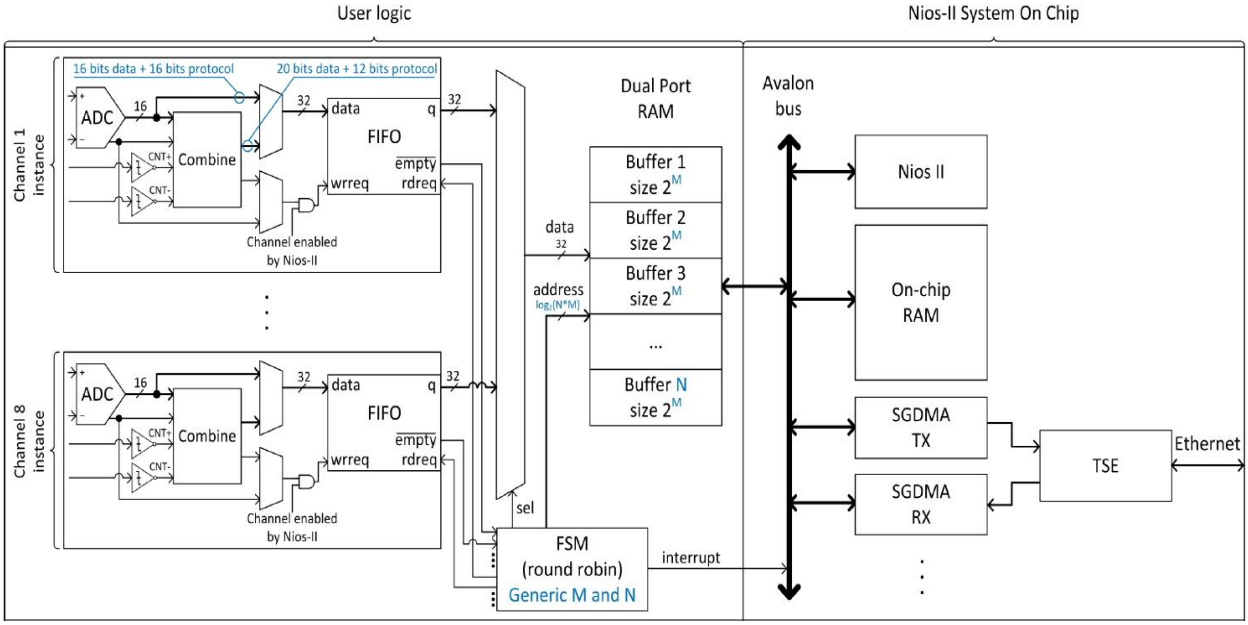

Figure 8: Architecture of the Nios-II system used for the Gigabit Ethernet Readout in the BLEDP firmware

More details regarding the block diagram, namely the user logic and the SOC, can be found on [10].

Most of the components included in the embedded software are provided by Altera and other vendors. The advantage of that solution is the very short time from a specification to a first working prototype. Another advantage of the ready-made software libraries is the availability of numerous services like ICMP, DHCP, etc.

The only custom software implemented for the BLEDP module is the server application. It is written in the C programming language and is a single-threaded application. The server creates a standard TCP/IP socket which listens for incoming client connections. It is limited to listen for only one client at a time and where there are requests for more clients the connection is refused. The server is parsing incoming commands and disables or en-

ables requested channels of the BLEDP module. The detailed logic of how the embedded BLEDP server transmits data from the input channels can be found on [10].

4.3 Client Server Architecture Protocol

A specific architecture was discussed, designed and then implemented for the communication of the Java client and the embedded server inside the FPGA. As already mentioned, the server application is limited to serve only one client at a time and when other requests come they are refused. The connection is established via a standard TCP/IP handshake between the two applications, after a connection request from the client is received by the server. The first thing sent by the server is the MAC address of the module on which it is running. This MAC address is used by the client in the logging protocol specified. The top folder which contains data from a specific module is named after this address. This logging-to binary files protocol will be discussed later on. The next thing done by the client is to send a command to the server in order to specify several parameters like the amount of expected data, the data format and the channel number it would like to receive. In response to that command, the server will either start the data transmission or it will stop all active transmissions. The corresponding options concerning readout and triggering for this command are listed in the following chapter.

4.3.1 Commands Sent by the Client

There are three groups of commands sent by the client to the server. All types use a TCP socket for communication. The first group is called Acquisition commands and refer to an online acquisition. After the establishment of a connection with the server, a start command is sent. It contains all the critical information for the acquisition, like the mode, channel number and time. On the other hand, a stop command can be sent at any time, before the end of a requested acquisition time. This command is triggered by the stop button in the graphical interface. The server simply terminates the data flow after receiving this command and the client reinitializes. The last acquisition command contains the destination UDP port in the server. The UDP stream sent by the server will be addressed to that port. This command is sent only if multi-channel mode is selected. The user is responsible to open this port, so that incoming UDP data won't be blocked by a firewall or other computer protection. The second group of commands is called Expert or Static commands. The purpose of these commands is to set the values of different internal registers of the BLEDP hardware, such as relays and potentiometers. The third group is called Advanced or Dynamic commands. Its purpose is to add a dynamic set of commands to control different hardware elements in the card. All the commands sent by the client share a common characteristic. Every command has a fixed size of 7 bytes. The first two bytes constitute the header, which the server uses to distinguish them. Furthermore, the next 4 bytes represent the payload of the command. This payload has different meaning depending on the type. The last byte is a CRC byte and is used to protect against

accidental/incorrect commands reception. Every command from the client side is fired by user actions. More specifically, two different panels in the user interface implement the Expert and Advanced commands. These panels implementation will be discussed later on. The Acquisition commands are produced automatically inside the client's logic, when a connection is initiated.

4.3.2 BLEDP Frames Encapsulation

The data frames produced by the BLEDP acquisition module and sent over the network by the embedded server are encapsulated into TCP/IP or UDP/IP frames depending on the mode of acquisition. Each data frame has an integer point of 32 bit size. The TCP protocol was selected only for the slower single-channel transmission due to its high reliability and automatic data reordering. The expected data rate for the single-channel transmission is 16 Mbit/s and the TCP protocol was considered the best choice for this mode. This rate comes from the fact that a new data frame is produced and transmitted every 2 μs. Respectively, for the multi-channel mode the transmission rate is expected to be 128 Mbit/s, so it is considered necessary the use of the UDP protocol for this mode. The UDP protocol is less reliable and data frames can arrive in the wrong order, but in comparison with the TCP it is much faster, because it does not make use of any error-checking and requires less computing resources. In respect to these principles, the UDP protocol may be more reasonable choice for a small embedded system like the Nios II.

The acquisition data frames encapsulated into one TCP/IP frame are presented in Figure 9. Each acquisition data frame sent by the server has 32 bits consisting of a 6 bits long header and 26 bits payload. The header and data format are described in more details in the next chapter. The "Maximum Segment Size" of the TCP frame has a size of 1460 bytes and can contain up to 365 acquisition frames. [16]

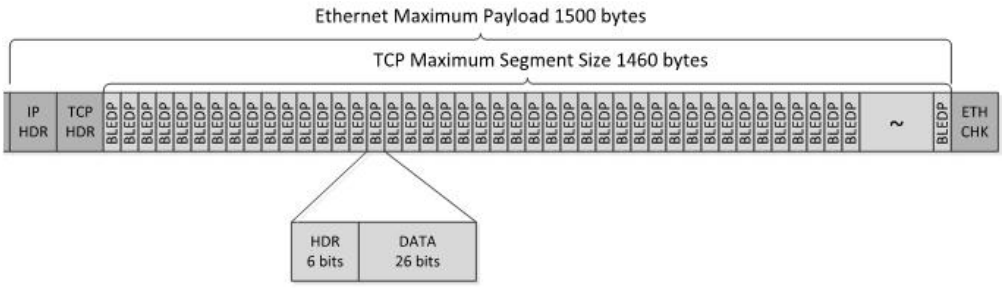

Figure 9: BLEDP data frames encapsulated into one TCP/IP frame

It is vital for the Nios II CPU performance to avoid the IP fragmentation at all costs, because of the lack of memory resources inside the embedded system. As a result the allowed payload was reduced and fixed to 1400 bytes. So, the BLEDP server always transmits TCP frames with a fixed TCP payload of 350 BLEDP acquisition frames. The package of 350 BLEDP frames was agreed to be referred to as BLEDP data bundle. The request from the client contains the data count number of BLEDP bundles. Regarding the case where the UDP protocol is used, the same principle is applied, which means that the data

count number request refers also to data bundles, even though UDP has no restriction for the payload size of the datagrams. This is for simplicity reasons. In Figure 10, a BLEDP bundle of size 350 and one of its acquisition frames is shown.

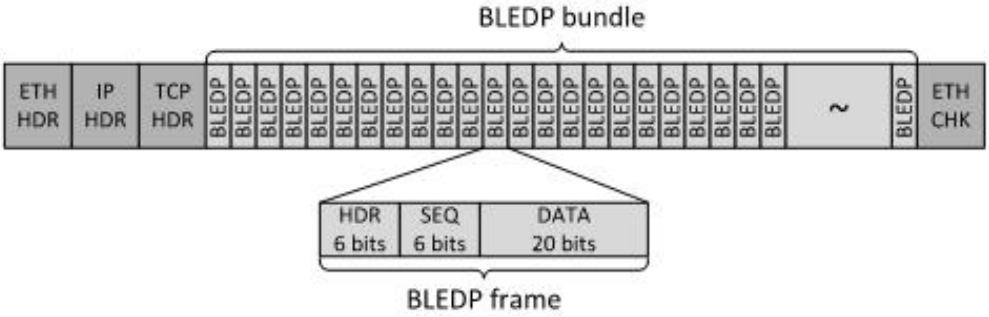

Figure 10: BLEDP bundle and Acquisition frame

4.3.3 Data Sent by the Server

There are two types of data sent by the embedded server inside the BLEDP cards. The first type is the acquisition data frames stored inside fixed-size buffers (bundles) as already mentioned in the previous chapter. The second type is the status data frames. This type is also transmitted inside fixed-size buffers and is interleaving in parallel with the acquisition data from the established TCP socket.

4.3.3.1 BLEDP Acquisition Frame

The BLEDP acquisition frame is the main packet produced by the FPGA and sent by the embedded server. The monitoring of these packets is the main goal of the design and development process of the client application.

In the final operating version of the system, several beam loss detectors like secondary emission monitors, diamonds, Cherenkov detectors and ionization chambers are going to be connected to the input of the acquisition system. The system makes use of two measurement acquisition methods, described in the corresponding chapters, and produces digital processed data out of the analog input signal. One processed data frame is produced every 2 μs for each of the 8 acquisition channels. The embedded server “talks” directly to the hardware enabling the transmission of these acquired processed data points.

Each processed data frame is acquired and manipulated by the embedded server into a 32-bit integer point with header, sequence number and payload. There are 6 bits composing the header with 3 types of information. Subsequently, there are 6 bits of sequence after the header for the processed data. These bits indicate an incremental counter value for debugging purposes. The range of this number goes up to its maximum value and then rolls back to 0. Lastly, the next bits indicate the payload value. The detailed information about the BLEDP acquisition packet structure is shown in table 1.

Bits	Name	Description
31	Acquisition/status	0 - acquisition data 1 - status data
30:29	Acquisition Data type	00 - processed data FDFC 01 - processed data DADC
28:26	Acquisition channel	channel number 0 to 7
25:20	Debug Counter	Increasing counter value 0-63 (for debug purposes). When counter riches max value it will roll back to 0.
19:0	Payload	Processed data value (unsigned integer number)

Table 1: BLEDP Acquisition Frame

4.3.3.2 BLEDP Status Frame

The second type of data sent by the embedded server is the status data frames. Status data refer to a number of sensors on the card and also several other parameters to be monitored (e.g. Temperature). This status data is interleaving in parallel with the acquisition data frames at 1 Hz frequency. Status data are transmitted exclusively through the TCP socket regardless of the protocol used for the transmission of acquisition data (TCP or UDP).

As already discussed, acquisition buffers (bundles) are sent through the network with a constant rate. The same principle is applied to the status data, so they are collected by the hardware and then packed by the server into fixed-size buffers and afterwards sent through the network. This fixed-size buffer transmission takes place every 1 second of acquisition time. The size of the buffer may differ throughout the development process. In Figure 11 an example of how a fixed-size status buffer interleaves between the acquisition bundles is shown. This example is from a single-channel transmission where in 1 second of acquisition, 1428 to 1429 acquisition data bundles are transmitted.

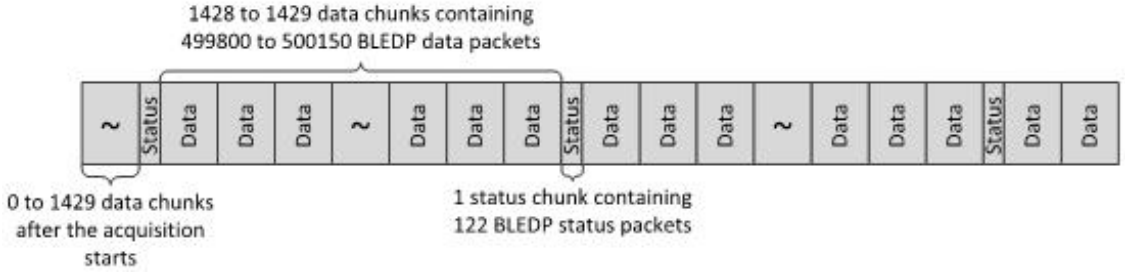

Figure 11: Status buffer interleaving between acquisition bundles.

The status packet has also a size of 32 bits and can be distinguished from the acquisition packet from the top bit (see Table 1). Furthermore, the first 16 bits of each status packet are the header and have a hexadecimal value of “0x8002”. In order to decode the desired payload values of the status packets (bottom 16 bits) the use of a complex memory map

as shown in Figure 12 is necessary. This process is done inside an appropriate class of the client application.

Category	Subdata	Interface Number [hex]	Tag Number[hex]	Real Size [bits]
Power	none	0x00	0x00	16
BLEDP FW	none	0x01	0x00	32
Chip ID	none	0x02	0x00	64
Global Address	none	0x03	0x00	16
SHT15	Temp	0x04	0x03	14
SHT15	Hum	0x04	0x05	12
Relay	none	0x05	0x00	9
Saturation Chan1	STOPOUT	0x06	0x00	32
Saturation Chan2	STOPOUT	0x07	0x00	32
Saturation Chan3	STOPOUT	0x08	0x00	32
Saturation Chan4	STOPOUT	0x09	0x00	32
Saturation Chan5	STOPOUT	0x0A	0x00	32
Saturation Chan6	STOPOUT	0x0B	0x00	32
Saturation Chan7	STOPOUT	0x0C	0x00	32
Saturation Chan8	STOPOUT	0x0D	0x00	32
Service ADC	PLUS_5V_CB1	0x0E	0x00	12
Service ADC	MINUS_5V_CB1	0x0E	0x01	12
Service ADC	PLUS_5V_CB2	0x0E	0x02	12
Service ADC	MINUS_5V_CB2	0x0E	0x03	12
Service ADC	HV_VALUE	0x0E	0x04	12
Digital Potentiometer Chan 1	Analog TH	0x0F	0x01	8
Digital Potentiometer Chan 1	Offset Current	0x0F	0x03	8
Digital Potentiometer Chan 2	Analog TH	0x10	0x01	8
Digital Potentiometer Chan 2	Offset Current	0x10	0x03	8
Digital Potentiometer Chan 3	Analog TH	0x11	0x01	8
Digital Potentiometer Chan 3	Offset Current	0x11	0x03	8
Digital Potentiometer Chan 4	Analog TH	0x12	0x01	8
Digital Potentiometer Chan 4	Offset Current	0x12	0x03	8
Digital Potentiometer Chan 5	Analog TH	0x13	0x01	8
Digital Potentiometer Chan 5	Offset Current	0x13	0x03	8
Digital Potentiometer Chan 6	Analog TH	0x14	0x01	8
Digital Potentiometer Chan 6	Offset Current	0x14	0x03	8
Digital Potentiometer Chan 7	Analog TH	0x15	0x01	8
Digital Potentiometer Chan 7	Offset Current	0x15	0x03	8
Digital Potentiometer Chan 8	Analog TH	0x16	0x01	8
Digital Potentiometer Chan 8	Offset Current	0x16	0x03	8
SFP1 MEM	Serial[15]	0x17	0x44	8
SFP1 MEM	Serial[14]	0x17	0x45	8
SFP1 MEM	Serial[13]	0x17	0x46	8
SFP1 MEM	Serial[12]	0x17	0x47	8
SFP1 MEM	Serial[11]	0x17	0x48	8

Figure 12: Status Memory Map.

The above Figure shows only a part of the actual size of the real status memory map. Nonetheless, the logic remains the same for the rest of the elements. It is referring to the bottom 16 bits of the packets in a status buffer. For each status element a decode process is described. In this regard, each element has an interface and a tag number translated as a header of 16 bits and a given payload size. For example, the temperature element has an interface number of 0x04 and a tag number of 0x03 as well as a payload size of 14 bits following this header (see Figure 12).

Now, we can analyze an example of how status elements are decoded from the integer packets inside the status buffers. Therefore, we have an arriving status fixed-size integer buffer and we are looking for the BLEDP FW element. The first few integer points of the buffer have the values in the following table:

Name	Status Prefix	Interface Number	Tag Number
		Payload	
Power Header	x8002	x00	x00
Power Payload	x8002	x003f	
BLEDP FW Header	x8002	x01	x00
BLEDP FW Payload [31...16]	x8002	x4e49	
BLEDP FW Payload [15...0]	x8002	x4f53	
Chip ID Header	x8002	x02	x00
Chip ID Payload [63...48]	x8002	xf700	
Chip ID Payload [47...32]	x8002	x000e	
Chip ID Payload [31...16]	x8002	x323c	
Chip ID Payload [15...00]	x8002	x4801	

Table 2: Portion of Status Buffer Example

We know from the memory map that the BLEDP FW record has a header with hex value 0x0100 and a payload size of 32 bits. Firstly, we track the third integer point in the buffer and we notice that the bottom 16 bits match the BLEDP FW header. Following this observation, we know that the next two points in the buffer contain the 32-bit payload of this element. It is shared between the bottom 16-bits of them. The same principles apply for all the status elements.

There is a specific class in the client side, which is designed and developed with the purpose of decoding the status buffers and getting the payload using several bitwise operations, in order to update the corresponding panel of the graphical user interface.

4.3.4 Development Phases of the Communication Protocol

The communication protocol between the embedded server and the diagnostics client was designed and developed in three different stages. These stages depend on whether single or multi-channel transmission is chosen and on the two acquisition methods (see chapter 3.2.2.1). The order of the development is the following:

- ***Processed Data Transmission From One Channel.***
  This mode is using the TCP/IP protocol for transmission of both the acquisition and the status data. As already mentioned (see chapter 4.3.2), the foreseen data rate for this mode is around 16 Mbit/s. The BLEDP processed packet is produced every 2 µs. Therefore, a TCP packet with its agreed payload of 1400 bytes will contain 350 BLEDP acquisition frames and will be filled within 700 µs.
- ***Processed Data Transmission From All 8 Channels.***
  This mode is using the UDP/IP protocol for the transmission of the acquisition data and the TCP/IP protocol for the transmission of the status data. The foreseen data rate of the acquisition data is 128 Mbit/s. Each channel produces one packet every 2

μs. Accordingly, each UDP datagram with a payload of 1400 bytes will contain 350 BLEDP acquisition bundles with multiple channels frames inside and will be filled within a time of approximately 87 μs. The associated channel number will be stored inside each frame header.

4.4 Client Development and Requirements

The BLEDP diagnostics client is developed using the JAVA programming language. It is a graphical user interface application, which was built using the Swing framework. The general purpose of this application is to test and validate the new beam loss acquisition system, before the final implementation. In this aspect, commanding and collecting data from the embedded server is required. The collected data –acquisition and/or status- should be displayed in a real-time view and also stored in binary files for further offline analysis. For the online view data reduction and processing is necessary due to the very high data rate. This high data rate is the reason and purpose of the offline view as well. The logic behind the offline analysis is that a user can navigate into previous acquisition sessions and analyze the data with a desired accuracy. A graphical user interface is needed to achieve the aforementioned functionalities. A user friendly environment, which separates the 3 different categories into tabs, was introduced. The online interface of the acquisition and status data and the offline interface is designed into 3 different tabs with multiple panels inside. There are several java classes within the project for handling the GUI tabs, the overall processing and the different modes of communication with the server. In addition, the integrity of the incoming data is of high importance and the relatively high data acquisition rate as well as other exogenous factors like network load, complicate this fact. The client should by any means cope with the server in terms of speed. In order to manage data reduction, online viewing, GUI updating and offline data storage in parallel, java multithreading technology is engaged.

4.4.1 Initial Resources

The most important component of the online and offline visualization interfaces is the JDataViewer package [13] developed at CERN.

JDataViewer (JDVE) is a plotting package that creates a simple representation of data, with powerful, extensible and easy to use function editing capabilities. The main reason why the JDVE package was chosen instead of other technologies, is the fact that it is developed at CERN, which meant instant support and feedback at the time of development and also it has the implementation of all necessary components for data monitoring, which includes data modeling (DataSource and DataSet objects), DataViewer class (the simple display object and its basis part), Chart class (simple chart object), different plotting renderers (Bar, Scatters, Polyline, etc), chart interactors (Zoom interactor, Pick interactor) , scalers, as well as data point annotations and data indicators. Also, a clear API is

provided to configure and customize all chart elements (e.g. colors, fonts, data ranges…) programmatically. The package has a nice user interface and also the ability to develop its functionality in general. Last but not least, the package offers class-leading performance, a very important feature for our data-intensive application. A visual example of a Chart component and different renderers and scales, produced by the JDVE package is shown in Figure 13.

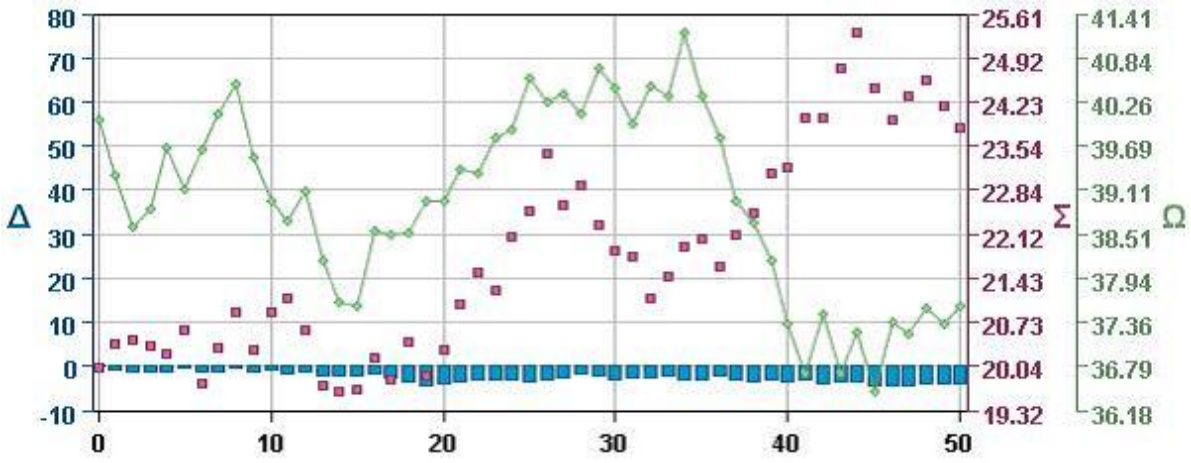

Figure 13: Chart component with three rendering types associated with three different Y scales. [13]

4.4.2 Online Acquisition Data Display

The most important tab and starting point of the diagnostics GUI client application is the online display. Data transmitted by the embedded server and received by the client are plotted on the online display after the desired data reduction. The online acquisition user interface is shown in Figure 14.

The online user interface is the first and default tab, which a user who starts the diagnostics client observes. A number of different settings have to be set before starting an online session.

First of all, a user has to select the data type of the acquisition, i.e. single or multi-channel processed data. This setting is represented by a group of two radio buttons and is located in the north-west of the GUI. It is referring to the 3 different modes of acquisition of the BLEDP cards. The next important setting is referring to the channel selection and is represented also by a group of 8 radio buttons. This setting has to be set only when we have single-channel transmission. Hereupon, the next group of settings refers to the connection with the BLEDP server. It is a group of 3 text fields, in which the user must enter information about the remote address of the embedded server and the TCP remote port. The third field is the UDP port and is referring to the local port, where acquisition data are coming, when the UDP protocol is engaged for the data transmission. To be noted here, that if the application is running on a PC, this port number has to be opened and secured in order to receive data, because in many cases, an installed firewall blocks

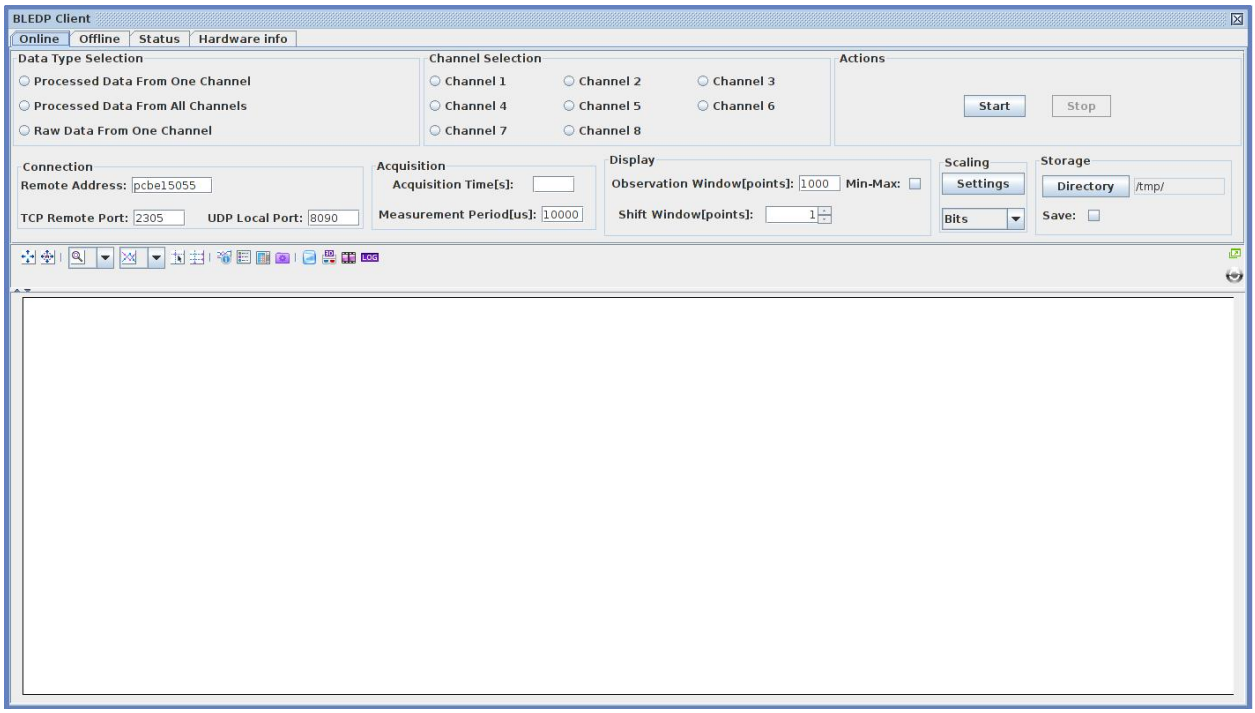

Figure 14: Online tab without data acquisition.

incoming UDP datagrams. Figure 15 shows the connection parameters. Next to them, another small but important group of settings is located. It is the acquisition group of settings and consists of two text fields. The first one is the acquisition time to be requested from the BLEDP server. The input value must be in seconds. The application transforms the value from seconds into the amount of data bundles which are equal to the requested time and sets the appropriate bits of the acquisition command to be sent to the server (see chapter 4.3.1). The second text field represents the desired measurement period in the online graph. The value should be put in µs. The measurement period refers to how many acquisition points will be averaged in the online display. The actual averaging number is the measurement period divided by 2, because one point is acquired every 2 µs by the card. For example if 10000 µs is put as measurement period, then one displayed point will be the average of 5000 acquisition points. Figure 16 shows the acquisition parameters as they appear in the interface.

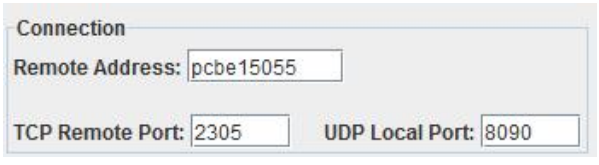

Figure 15: Connection settings

Figure 16: Acquisition settings

The next 3 groups are the display, scaling and storage settings and parameters. Regarding the display group, there are three components inside. The first one is a text field about the observation window in points and refers to the amount of points to be displayed in the online graph. The second one is spinner and refers to the shift window, again in points.

The shift window is strictly related to the observation window. Its value cannot be lower than 1 and higher than the value of the observation window. These restrictions justify the use of a spinner instead of a text field. More specifically, the shift window value states the limit where the display buffer filled with averaged points is refreshed with new values. The last component of this group is the Minimum-Maximum checkbox. It states whether or not the minimum and maximum plots should be displayed along with the average plot. This checkbox is also editable while an online session is running. The display parameters are shown in Figure 17.

Figure 17: Display settings

Figure 18: Scaling

The next group of parameters is the scaling settings. It consists of a settings button and a combo box. The Settings button fires a pop-up window, which is shown in Figure 19. The name of the units in this window indicates the name of the Y-Axis in the online graph. Furthermore, the A and B parameters are used in the scaling equation along with the payload value of the received BLEDP acquisition frames, in order to calculate the actual displayed value. The user can save the desired A and B values for each unit. This pop-up window and the panels and operations of it are handled by a class named ScaleSettings. This class is also used later in the offline and status tabs.

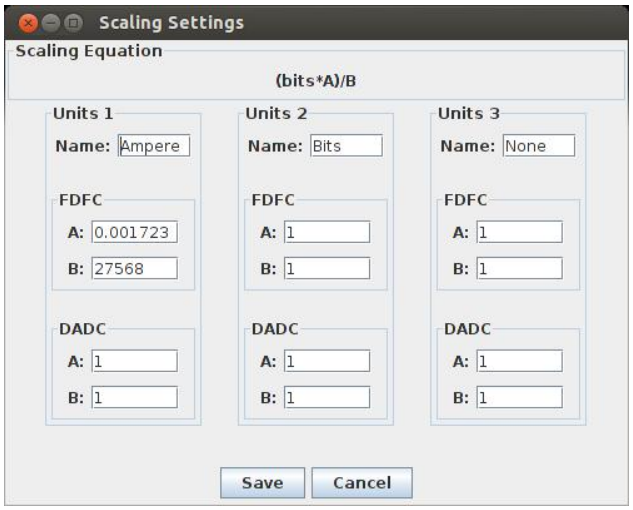

Figure 19: Scaling pop-up window

Figure 20: Storage

The combo box is used to select the desired display unit for the upcoming online session. You can spot that the A and B parameters are different depending on the acquisition method (DADC or FDFC). The purpose of the scaling units is mainly to provide better development and feedback.

The last and very important group of settings is the storage group. It consists of a button, a small text field and a checkbox. It is shown in Figure 20. A user should utilize the check box to select whether or not to save the incoming data into binary files. The path used for the logging process is displayed in the text field. This path is specified by the user, who must press the directory button, so that a pop-up window with a Folder-Chooser appears for path selection.

All of the aforementioned parameters should be set before starting a new acquisition session. In the actions group of the GUI the Start and Stop buttons exist. A control check of the values of the parameters should be initiated before establishing a connection with the BLEDP server. If a parameter is wrongfully or not set and the user presses the Start button, an error pop-up window appears.

Most of the important parameters have default values, in case it is the first time that the client executes on a computer. If it has been executed more than once on the same computer, then the application “remembers” the settings of the last time. This feature is achieved with the help of a class named PersonalSettingsManager. This class makes use of a HashMap and a binary file, where the latest values of many Swing components are stored. The directory where this binary file is stored, depends on the operating system on which the client is running. Furthermore, it is obvious that different binary files are stored for different users of the client.

4.4.2.1 Design implementation of the online interface

In general, the online panel is handled by a class named OnlineDataViewerGUI. This class creates and handles all the panels and the components inside them using several swing layout managers, i.e. FlowLayout, GridLayout, GridBagLayout and BorderLayout. It also contains several action listeners and event-handling methods for the different components and manages every control check of the parameter values before initiating the communication with the embedded server. Also, the main method of the application is part of this class.

The Swing event handling code runs on a special thread known as the event dispatch thread (EDT). Most code that invokes Swing methods also runs on this thread. This is necessary because most Swing objects methods are not “thread safe”, so invoking them from multiple threads risks thread interference or memory consistency errors. It’s useful to think of the code running on the event dispatch thread as a series of short tasks. Most tasks are invocations of event-handling methods, such as ActionListener.actionPerformed. Other tasks can be scheduled by application code, using invokeLater or invokeAndWait. Tasks on the event dispatch thread must finish quickly; if they don’t, unhandled events back up and the user interface becomes unresponsive. [12] In our case, we need to execute a long-running task, which is the communication with the server and then data acquisition, processing, logging and plotting. We have to use a background worker thread for that

purpose. For our background task we will use an instance of the SwingWorker abstract class and two simple custom-implemented methods:

- public void doInBackground(), which executes our time-consuming task by calling the start method of Communication class. This class will be discussed later on.
- public void done(), which in our case handles gui operations like activating or deactivating critical components or reinitializes other variables, after the background task is done. We have to note here that the done() method is executed on the EDT.

Our SwingWorker has the ability to terminate at any given time before the original end of the background task. This is the purpose of the Stop Button in the Actions group of parameters. In case of early cancellation, the SwingWorker.cancel is invoked. Our background task cooperates with its own cancellation by invoking SwingWorker.isCancelled at short intervals and terminates depending on the return value; true if cancel has been invoked for this SwingWorker.

4.4.3 Communication and data processing

A user is asked to set an ensemble of different parameters and then initiate the beginning of an online session with the BLEDP server by pressing the Start button on the interface. As already discussed, a Swingworker is spawned in order to handle the background processing.The Communication class is responsible for the establishment of the link with the server, as well as the acquisition, distribution and processing of the data.

It is required that the client will be able to cope in terms of speed with the server, so a simple threading mechanism is used to achieve this feature. The current Swingworker thread spawns another two threads; the first is responsible for the processing and plotting in the online graph and the second for the logging of the data to binary files. These three threads have different execution priorities. The current thread that acquires data from the socket has the maximum priority, the plotting thread has medium priority and finally the logging thread has minimum priority. The three threads are sharing two ConcurrentLinkedQueues. This type of queue is thread-safe and based on linked-nodes. Furthermore, it orders elements in a classical first-in-first-out (FIFO) mode. The head of the queue is the element that has been on the queue the longest time and the tail is the element with the shortest time on the queue. The elements of the queue are fixed size integer buffers as they are acquired by the socket; TCP or UDP depending on the mode. Additionally, there is no alteration of the data by the plotting thread, in terms that the buffer polled by the first queue is the buffer offered to the second queue with no corruption. Figure 21 demonstrates the function of the 3 threads and the shared queues.

The queues contain only acquisition data. The distribution of the acquisition and status data packets takes place inside the Swingworker socket-acquisition thread. Status data buffers are first plotted online with the help of another class in the status panel. The

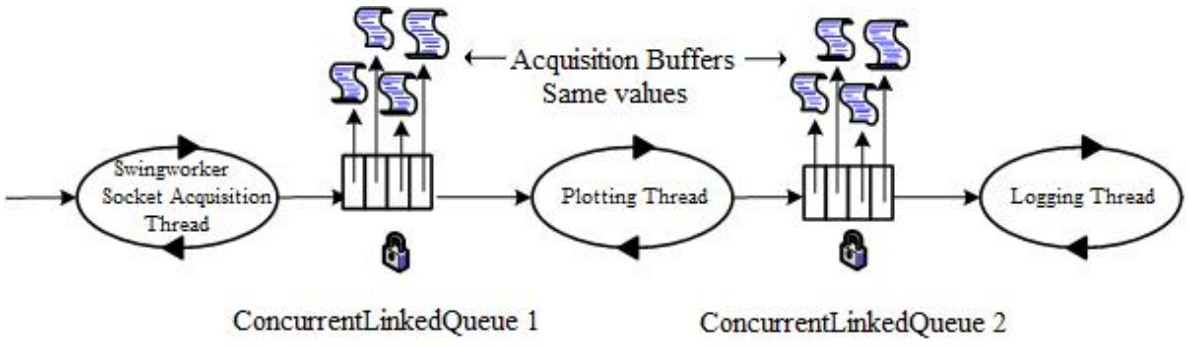

Figure 21: Socket, plot and log threads.

status panel will be discussed later on. Thereafter, status buffers are offered to another ConcurrentLinkedQueue which is polled only by the logging thread. The update of the status panels, which takes place inside the Swingworker socket-acquisition thread, was not considered harmful for the performance of the thread due to the low data rate of the status buffers (1 Hz).

The algorithms used by the three threads for sharing and processing the data is described below. We have to emphasize here again that the queues are thread-safe, which means

that a safe execution with no conflicts is guaranteed.

1. Read Data from the socket(s) –TCP always, UDP if applicable;
2. Distribute acquisition/status data;
2.1 Update status panels if enough data;
2.2 Offer Status_Log_Queue filled buffer;
2.3 Condition-signal Status_Log_Queue;
3. Offer Plotting_Queue buffer;
4. Condition-Signal Plotting_Queue;
5. Goto 1;

Algorithm 1: Swingworker-Socket Thread

1. **while** *Plotting_Queue.isEmpty* **do**
   | wait_for_condition_signal
end
2. Poll buffer;
3. Offer Logging_Queue buffer;
4. Condition_Signal Logging_Queue;
5. Process buffer and plot data;
6. Goto 1;

Algorithm 2: Plotting Thread

1. **while** *Logging_Queue.isEmpty* **do**
   | wait_for_condition_signal
end
2.Poll buffer;
3.Process buffer and log acquisition data into files;
3.1 Poll status data from the Status_Log_Queue and log them into files;
4. Goto 1;

Algorithm 3: Plotting Thread

The plotting thread uses the settings given by the user to calculate the average, maximum and minimum values of incoming data and then plots them in the graph of the online panel. This data reduction depends on the display width of the graph and on the width of the observation window.

On the other hand, the logging thread is responsible for storing all the data into files (acquisition and status) without data reduction. The data stored in these files will be used for the offline data display panel, which will be discussed later. Only processed data are plotted both online and offline. A special format for the data storage was discussed and agreed. The number of the BLEDP packets in the storage file is configurable with default value 500000 for the single-channel and 4000000 for the multi-channel transmission. In the default setting the client application creates one file per each second of acquisition. The BLEDP packets are written into files with their header. Each file name is composed

of the UTC time (in ms) and of the data type (0 – processed data from one channel, 1 – processed data from all channels). Along with the acquisition file, a stats and a status file is created every second. The stats files include three values; the average, the maximum and the minimum value of the payload of the data in the corresponding acquisition file. The status file includes the status buffer for the given second. Each of these three files share the same filename with a different extension for each type. Every hour of the acquisition (3600 files) is stored in a separate folder named using time in a human readable format: YYYYMMD_hh. Every new acquisition is represented by a folder which is created as the parent of the hour folders. This folder is also named in a human readable format: YYYYMMDD_hhmmss, more accurate than the hour folders. All these folders are created in the top folder, created inside the configurable input path and named using the MAC address of the BLEDP card. We mentioned already that this MAC address is the first status element sent by the embedded server after the establishment of connection. A graphical representation of the storing format is shown in Figure 22.

The incoming data are stored in binary files, only when a user selects the corresponding checkbox in the online panel. Otherwise, only the plotting thread is active. It is also quite important the amount of free space on the target hard drive, due to the high data rate of the incoming data. For example, a 2-hour single-channel acquisition would require approximately 13.42 gigabytes of free space. For the same period of multi-channel acquisition almost 107.3 gigabytes should be available.

4.4.4 Offline Acquisition Data Display

The second and equally important tab of the diagnostics client is the offline panel. This panel is designed and implemented for the offline analysis of past online sessions with the BLEDP server. The logic behind this feature of the application is that a user is able to load data files stored from previous online sessions and using the designed interface, make a very accurate analysis of past acquisition data.

We can proceed with explaining the user interface of the offline panel. In Figure 23 a screenshot of the offline panel with the Open button marked is shown. In general, the interface of the offline panel is quite simple. It contains two buttons, some labels indicating names of settings, a combo-box, a checkbox, a spinner and one or two graphs depending on the state.

The first step from the user perspective is to press the Open button. After this action a file search window will appear like the one in Figure 24. Following the storing format discussed in the previous chapter, a user should navigate through the selected path and then find the folder(s) named after the BLEDP card MAC address(es). Inside these folders, an acquisition or hour folder should be picked for the offline analysis. A visual representation of the pop-up window used for the path selection is shown in Figures 24 and 25.

After this action a very important component should appear in the offline user interface,

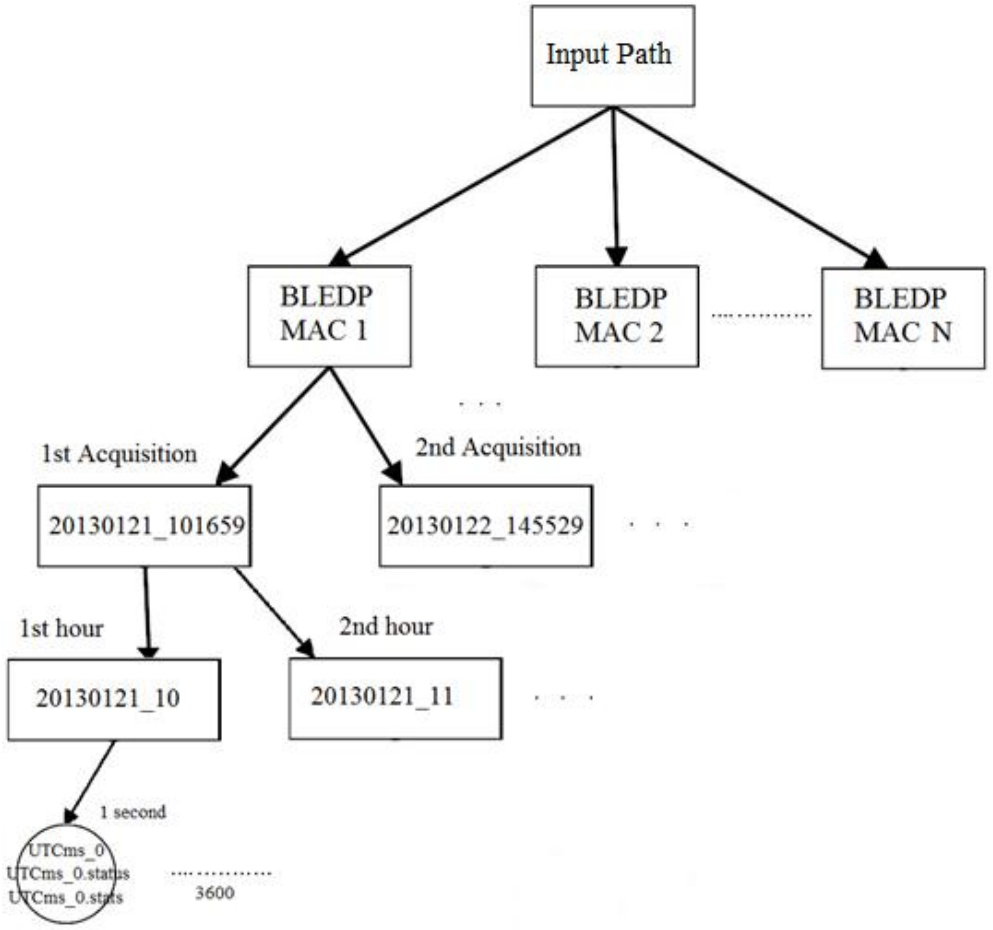

Figure 22: Storing format of acquisition and status files.

the data-picker graph. The plots in this graph are created by the .stats files inside the acquisition folders, which are created for that purpose by the logging thread during a past online session. Accordingly, there is a plot for the minimum, maximum and average values for every second of each channel in the selected acquisition period. This means, that each point of the average plot represents the average value of 500000 acquisition points as they were calculated and stored from the logging thread. Respectively, each point of the minimum and maximum plots represents the minimum and maximum values for the 500000 points of a second. Depending on the mode, the maximum amount of plots in the data-picker graph is from 3 (single-channel mode) to 24 (multi-channel mode). There is also an option for switching On/Off the maximum and minimum plots. Furthermore, on multi-channel mode there is a possibility to disable/hide different channels average values for comparison purposes.

For example, the figure below shows the created data-picker with an average plot of a past single-channel acquisition. The minimum and maximum plots are disabled in this case. Additionally, in parallel with the data-picker, several other labels appear on either sides of the Open button, after the selection action. These inform the user about the selected path, as well as the starting and ending point of the past period of the displayed acquisition. The example below shows a 100-second single-channel acquisition. The data-picker graph is

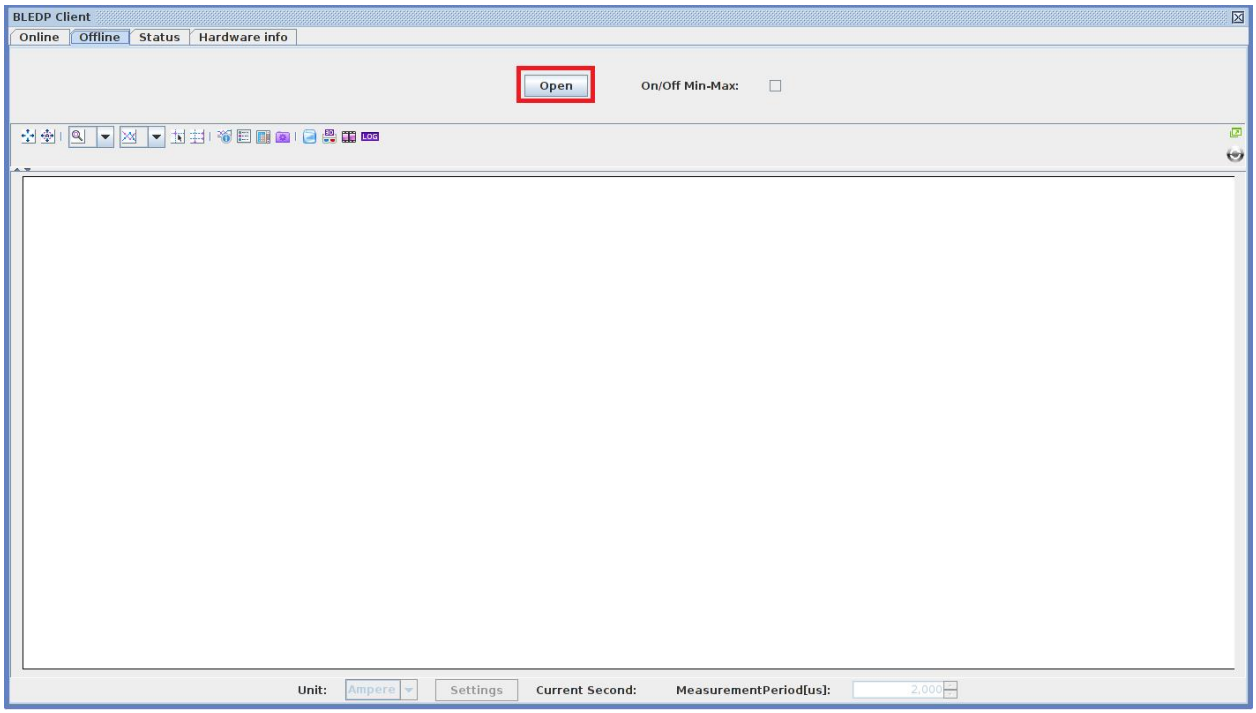

Figure 23: Offline tab without data.

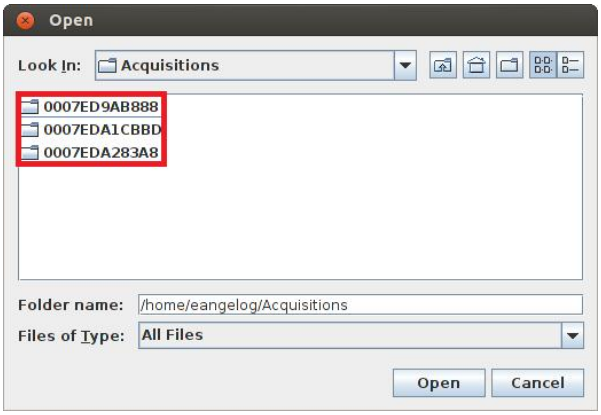

Figure 24: Different Mac Addresses (BLEDP Cards)

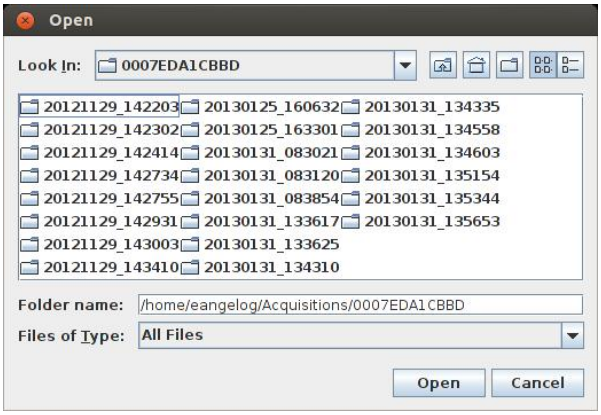

Figure 25: Different Acquisition sessions to pick for analysis

equipped with two mouse and one keyboard interactors. The first mouse interactor is the zoom interactor, provided by the JDataViewer API, and the second is a custom-made data-pick interactor, which we will discuss about in the next few lines. The keyboard interactor is about moving around the data using the arrow keys, if a zoom interaction was performed. The most important custom-made data-pick interactor gives the possibility to pick a point using the left mouse button. After this action, the data of the picked point, which represents a second, are read from the corresponding data file and plotted in the lower graph using the average value from the measurement period as reference. Furthermore, all the settings in the lower part of the interface get enabled. The measurement period and the combo-box have the same meaning as in the online interface. There is also a label informing the user about the current second picked. Additionally, a set of black markers appear in the upper graph, one on every point of every plot visible, indicating which second was last picked.

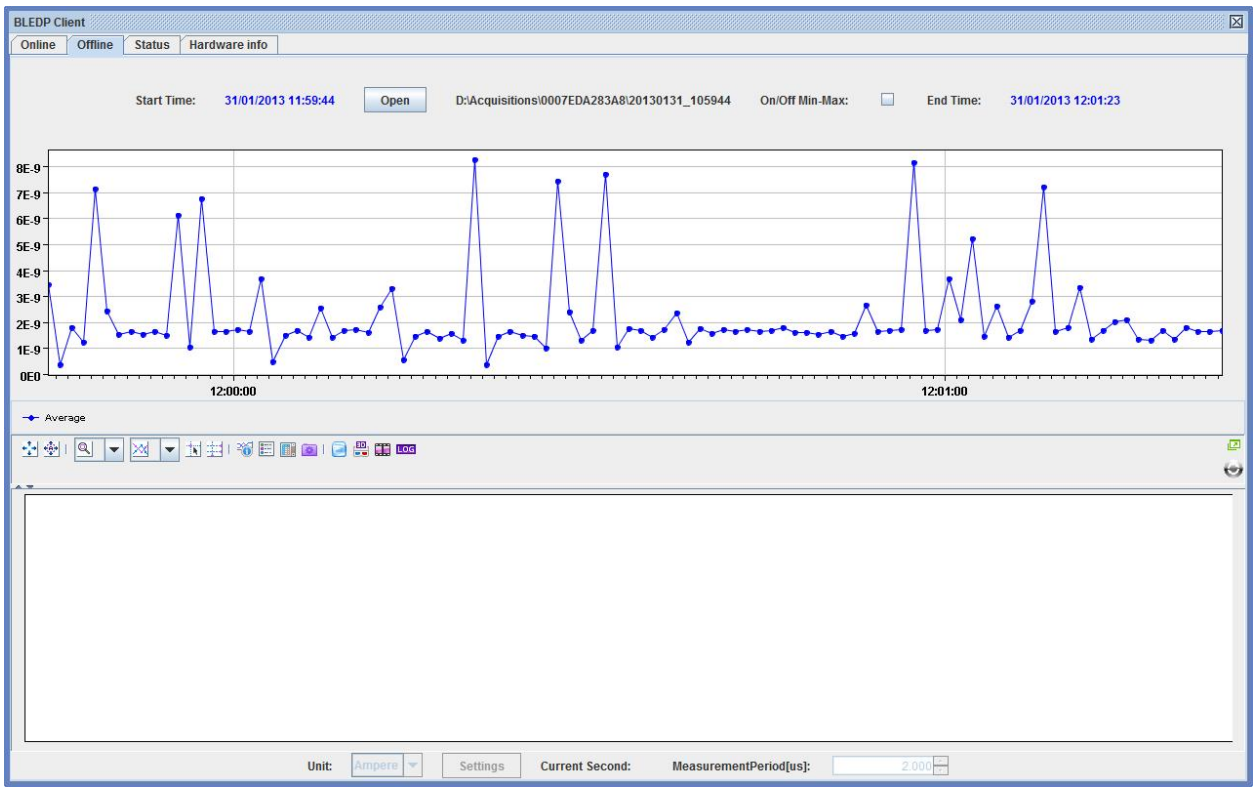

Figure 26: Offline Data-Picker.

The number of plots in the lower graph is the same as the number of plots in the upper data-picker graph, i.e. from 1 to 3 for single-channel acquisitions and from 1 to 24 for multi-channel acquisition. The only exception in this rule is when the measurement period equals 2 μs and the minimum and maximum plots are enabled. In this scenario there will be no maximum and minimum plots in the lower graph, because every single point of the 500000 points in the picked second will be plotted. In Figure 27 an example of the above description is shown.

Another important fact about the offline logic is that except from the current second picked, another 5% of the previous and the next, if they exist, is plotted on the lower graph. This feature was proposed for more flexibility regarding the navigation between the acquisition seconds. In the above figure, the edge borders of the X scale indicate the start and the end of the data stream of the picked second respectively. The remaining data make up the 5% of each neighboring second.

Furthermore, there is a small difference in the interface depending on the mode of the acquisition session that was opened. If a multi-channel acquisition was opened, a small panel will appear south of the Open button, containing 8 labels with channel numbering and 8 corresponding checkboxes. This panel gives the option to enable/disable one or more channels from the graphs dynamically. This feature is very useful for comparison of the data between the channels, especially when different detectors are connected to them. The following figure demonstrates this fact with 3 enabled channels.

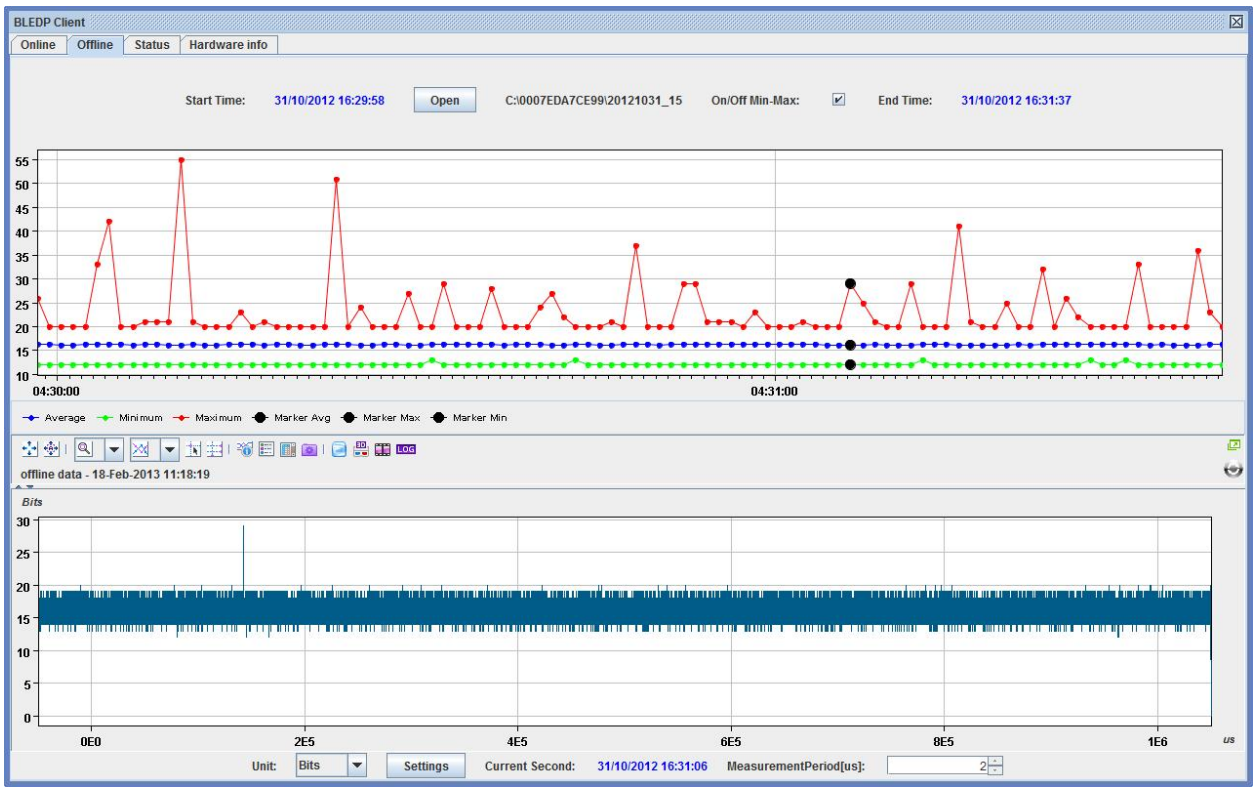

Figure 27: Offline pick with minimum-maximum enabled and measurement period 2 μs.

4.4.4.1 Design implementation of the offline interface

The offline user interface and the logic behind it, is created and handled by 3 different sub-classes which extend a JPanel class. The first one named OfflineDataViewerGUI is responsible for the creation of the interface. Similar to the online interface, this class uses different layout managers to create small inner panels with all the required components. An object of this class is created inside the constructor of the OnlineDataViewerGUI. The second class of the offline concept is named UpperOfflinePanel. Likewise, an object of that class is created inside the constructor of the OfflineDataViewerGUI and is added to the north of the interface as a JPanel. Basically, it implements the logic behind the Open button and all the different components that lie in the upper panel of the interface. It creates an action listener for the button which handles requests for displaying past online sessions by the user. Given the selected online session path, all the data are collected by the .stats files of the acquisition and stored into buffers. These buffers, along with other variables such as file descriptors and lengths, constitute the arguments of the constructor of the third class; the DataPicker. This class creates the data-picker graph with the stats plots from the requested acquisition session. A DataPicker instance is constructed inside the action listener after a selection of an acquisition session path by the user. This is the reason why the graph appears in the interface after a valid path selection. This class handles most of the picking and processing logic of the displayed data in both graphs. It also contains an action and an item listener for the scaling settings and the measurement period spinner respectively. Finally, it provides a method to dereference every inner object field. This is vital to avoid memory leaks in the heap space after a sequence of opening different past

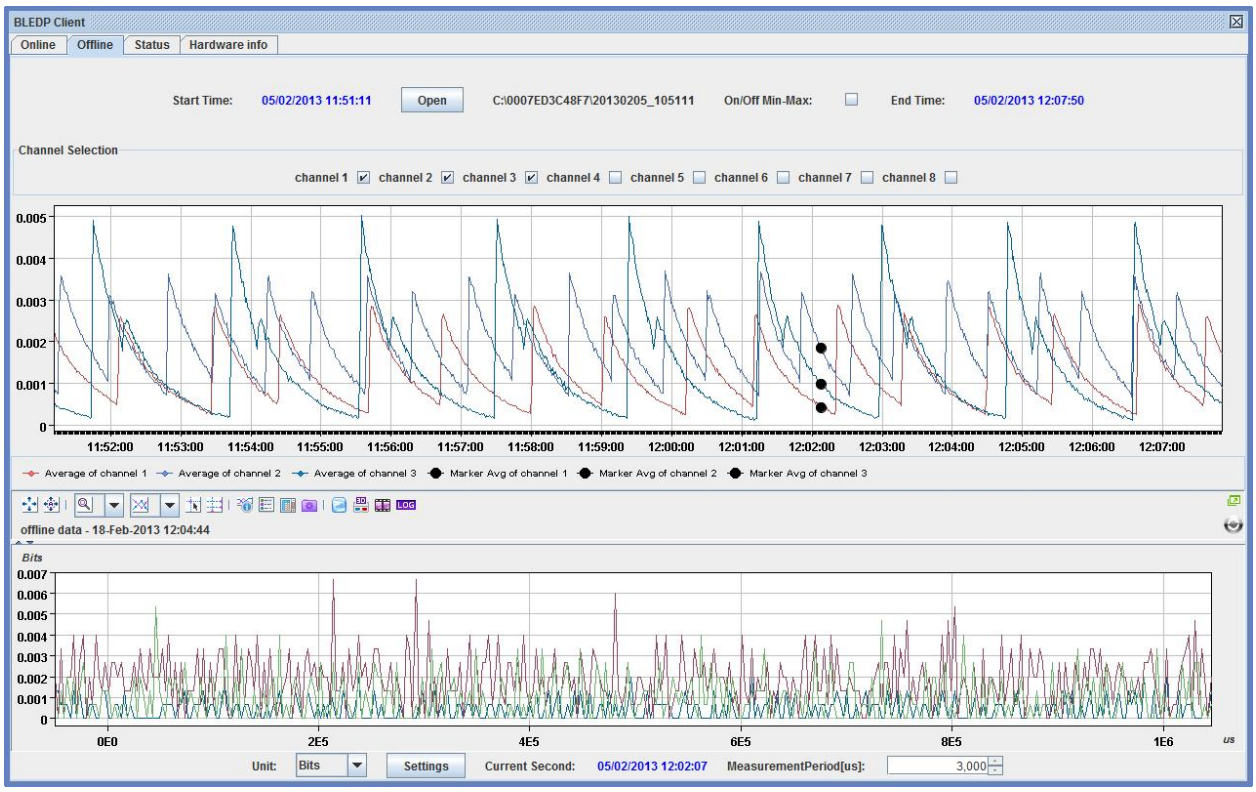

Figure 28: Offline pick of a multi-channel acquisition.

online sessions. Lastly, everything regarding the offline logic runs on the Event Dispatch Thread (EDT), as most of the offline implementation is regarded as Swing event handling code. Due to the fact that all of the tasks executed in the offline panel take a rather small amount of time to execute, the user interface does not become unresponsive and thus there is no need to spawn a Swingworker thread to execute the offline code.

4.4.5 Status Data Displays

The two status tabs of the user interface are in charge of displaying in real-time, the status data which are transmitted in parallel with the acquisition data as already discussed in chapter 4.3.3.2. The two tabs are activated only when an online session with the server is in progress. The concept of displaying in real time the status data, is split into two tabs, because of the big amount of the necessary components and the limited amount of space. The first tab was agreed to be called “Status” and the second tab “Hardware Info”, because most of the status elements of this tab provide information about the BLEDP hardware. The update rate of these tabs is 1 Hz, so the elements are updated every one second. We can proceed with explaining the user interfaces. In Figure 29 the default outlook of the Status tab is shown.

This tab contains several components, such as textboxes, LEDs, labels and graphs, used for the monitoring of the status elements. The lower three graphs contain information about the current consumption, the high voltage monitoring, the temperature and humidity of the connected card. The upper graph is a micrography of the graph in the online tab,

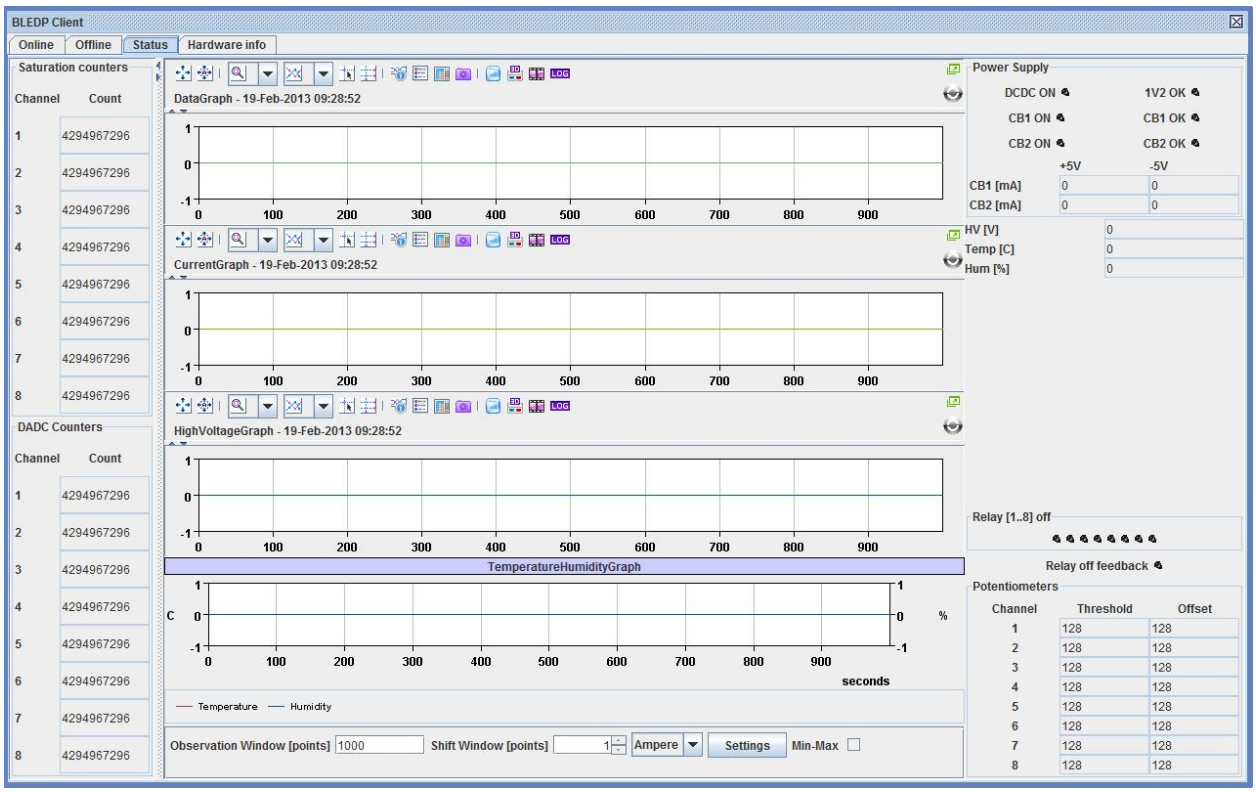

Figure 29: Default outlook of the status tab.

with the difference that the measurement period in this one is stable at 1000000 µs. This way an update rate of 1 point per second for the acquisition data is achieved, matching the rate of the status data. We anticipate that the plots of the four graphs will be aligned in the X axis during an online session. In the south point of the status tab, several settings related to the graphs exist. These settings are similar to the ones in the online tab and refer to the scaling, observation window and the shift window of the graphs. In the west of the tab, several other components exist. They are in charge of plotting the values that appear in the textboxes, the status of the power supplies as well as the status of the calibration relays using state changing LEDs. Lastly, in the southeast and the west of the tab, other status values like potentiometers and event counters subsist.

The second Hardware Information tab displays the remaining status elements. The default view of this tab is shown in Figure 30.

The Hardware info tab contains several components like textboxes, LEDs, labels and graphs. Most of the elements are mainly information about the status of the two small form-factor pluggable transceivers (SFPs) on the BLEDP card used for the communication of the card with the network, as well as other hardware information such as firmware version, chip ID and global address. The same update rate of 1 Hz for the plots and the other information applies here as well

4.4.5.1 Design implementation of the offline interface

The two status tabs are developed around two main classes, but also include a number

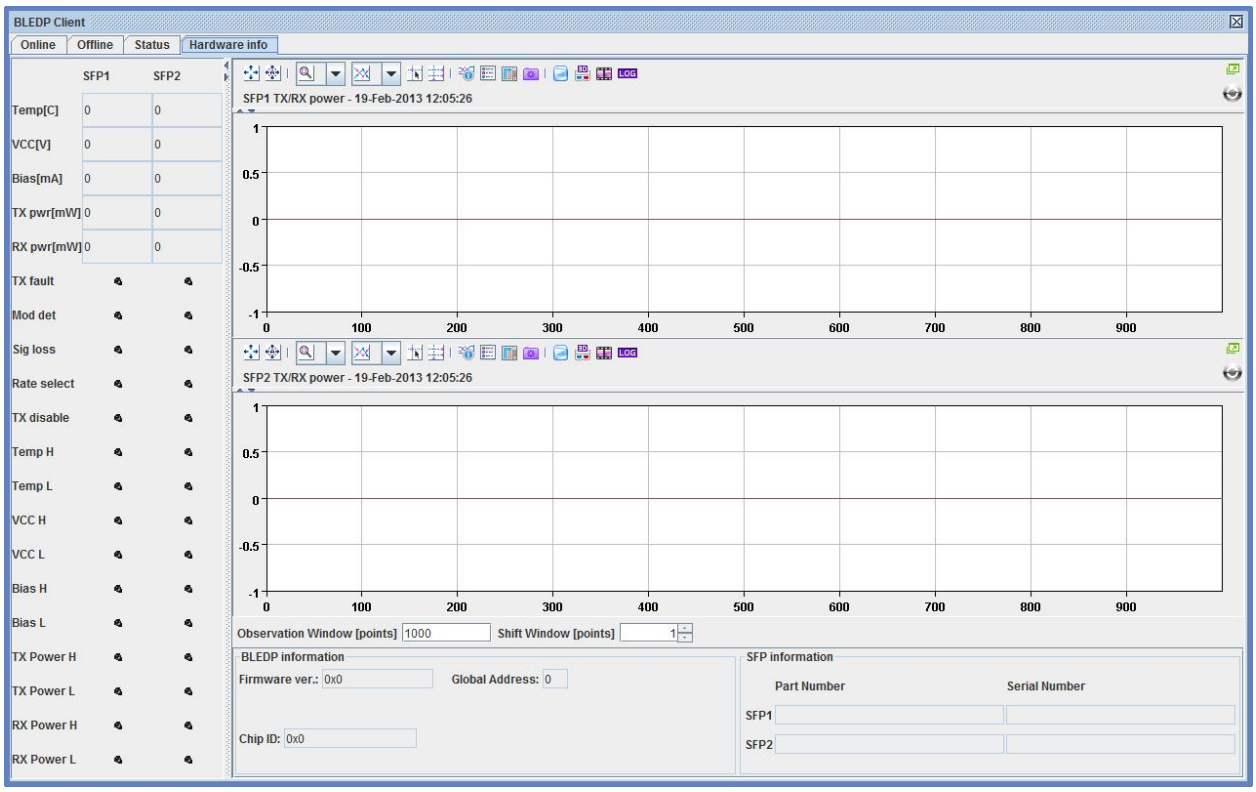

Figure 30: Default outlook of the hardware info tab.

of auxiliary ones for the inner panels.

StatusGUI is the entry point class. An object of this class is created inside the constructor of the OnlineDataViewerGUI class, where the base frame of the application is created as well. Similarly to the other tabs of the GUI, the constructor of this class builds the interface of the first status tab using different swing layout managers and utilities. The other main class is the HwGUI. The main principles of the StatusGUI apply to this class as well, i.e. an object is constructed in the same place and the interface is built with the same techniques. These two tabs are updated every one second via a dedicated method in the StatusGUI, which takes as an argument the status buffer (see chapter 4.3.3.2). This method distributes the status elements in the buffer and makes all the necessary processing for the final result. Afterwards, it calls other methods from the auxiliary classes in order to update both interfaces with the newly calculated status values. The creation occurs in the EDT, but the update of the status panels is executed in the Swingworker socket thread (see the algorithm in chapter 4.4.3)

4.4.6 Commands Displays

The two remaining tabs of the GUI implement the Static and Advanced commands discussed in Chapter 4.3.1. These commands are sent to the server via the TCP socket. Their purpose is to set different internal registers of the FGPA and thus affect the measurements. Regarding the static commands tab, the payload of every unique command can be set either with the use of a textbox or a toggle button. A textbox is used in case

the required command payload is greater than one bit. Similarly, a toggle button is used in case the payload of the command is one bit (On-Off state). In Figure 31, the static commands tab is illustrated. In this Figure every component is disabled, because a connection with a BLEDP server is not established. The tab groups similar commands inside titled borders. For every group there is a “Get All” button. Since all these commands refer to status elements, the purpose of the “Get” buttons is to update the textfields and togglebuttons with the very last values of the status elements they refer to. These values lie inside a dedicated buffer, which is updated every one second with the new incoming status values. The buffer is protected against concurrent access when a “Get” button is pressed. Furthermore, the “Set” and “Set All” buttons, inside the Potentiometers group, fire a corresponding block of code, which is responsible for the structuring of the command using the value in the textfield and then sending it to the server. In a similar way, a set action in the other two groups is initiated by the dedicated togglebuttons.

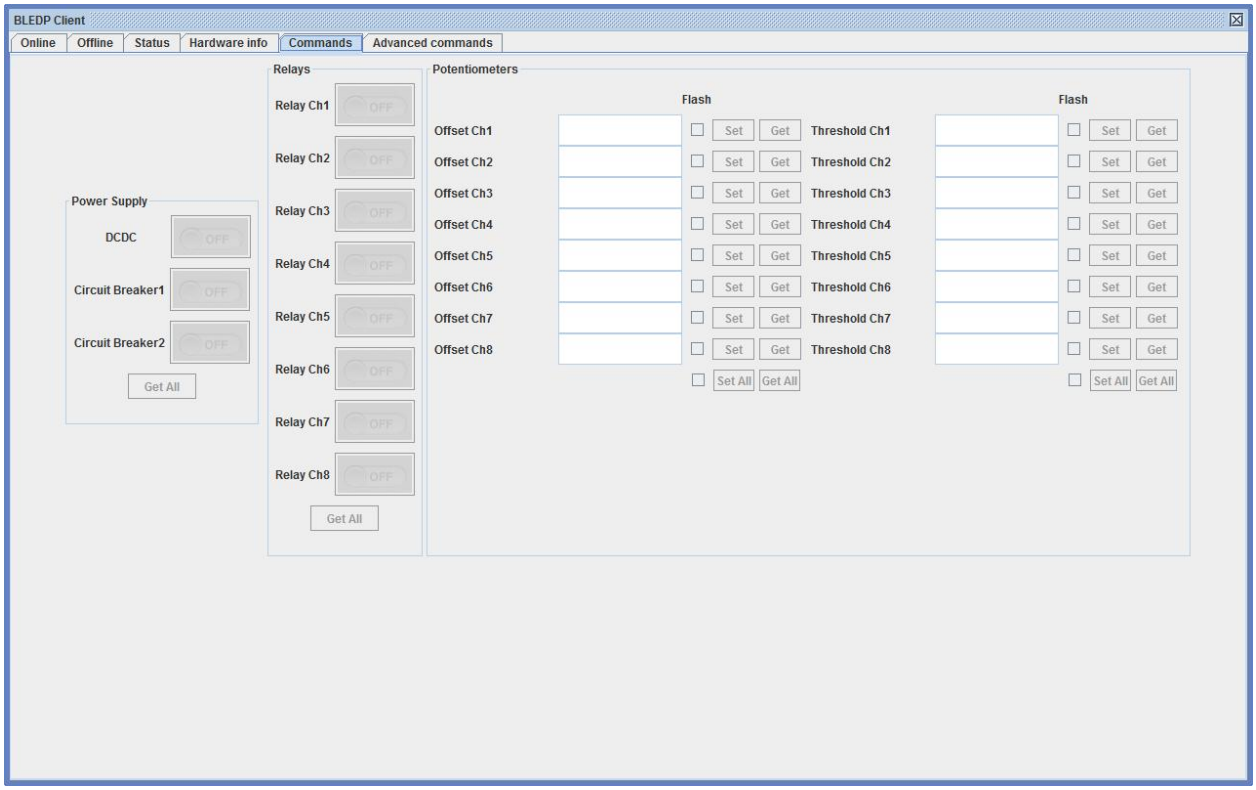

Figure 31: Default outlook of the static commands tab.

Finally, inside the potentiometers group several checkboxes exist. If checked, a special bit is activated inside the command byte buffer, which tells the server to write the incoming command in the flash memory of the FPGA instead of the RAM.

Regarding the Advanced commands tab, it is created only in case a dedicated CSV file is included in the project’s resources. This file contains all the information needed to construct this tab. Figure 32 illustrates the outlook of the Advanced commands tab, when a connection with a server is not initiated.

Each row of this panel implements an advanced command to the server. The payload

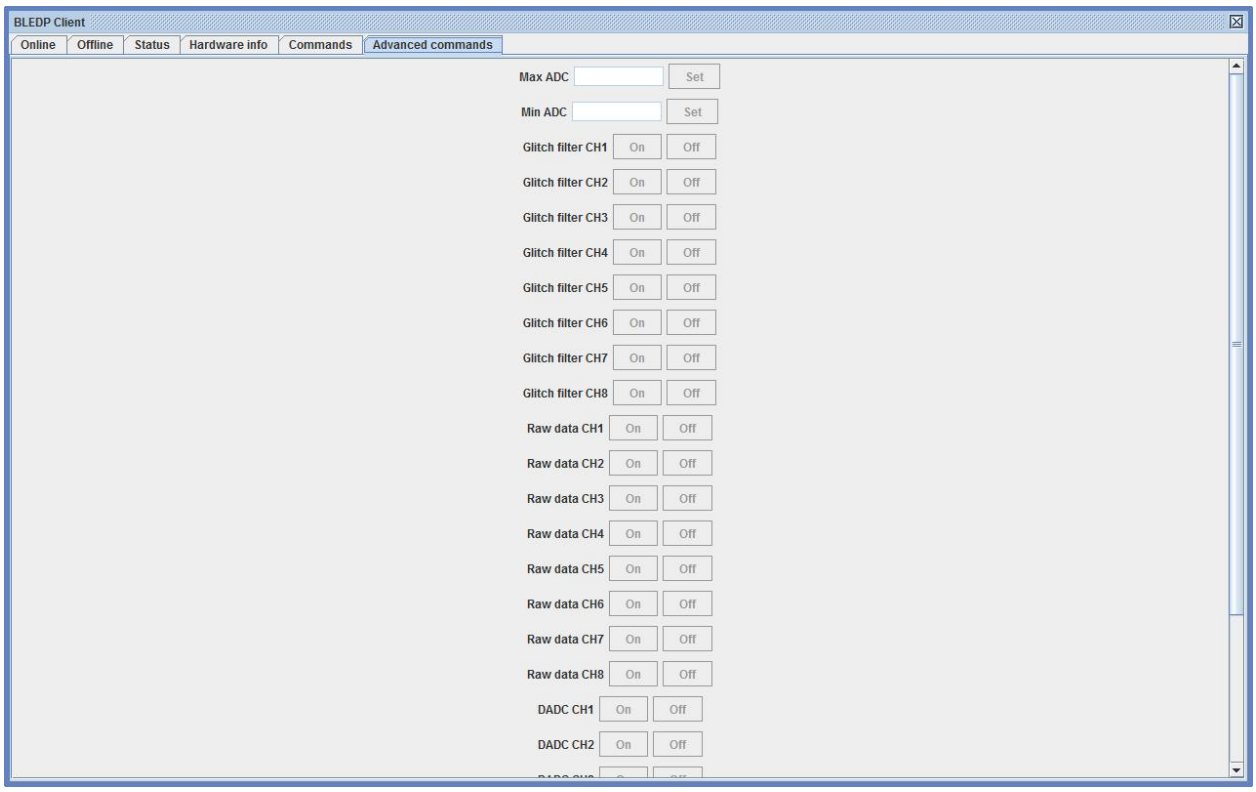

Figure 32: Default outlook of an advanced commands tab.

size of each advanced command is variable, so it can be either greater or equal to 1. In the first case, the payload value is parsed from a textfield as in the first two rows. The Set button is used to send the command in this respect. In the second case, the payload value (On/Off states) is set and the command is sent, by a “Press” action of the corresponding On/Off buttons. The total number of the advanced commands in this tab is based on the CSV file and is completely dynamic. Using this feature, the developers of the FPGA can add/remove new commands in the file and reload the client updated.

4.4.6.1 Design implementation of the commands interfaces

The two commands displays are developed around two main classes and an auxiliary one.

The static commands tab is implemented by a class named StaticCommandsGUI. This class responsibility is to create the static commands tab interface, as well as implement action listeners for the different elements inside the interface. The auxiliary class named Commands, implements the separate construction of each command. The commands are sent via the TCP socket to the server after construction. A “Set” action by the user is required to fire the code for constructing and sending the corresponding command inside an action listener. The other main class, used for the creation of the Advanced commands tab, is called GUIScalarElement. This class is used only in case a suitable CSV file is present in the project’s resources. Firstly, it parses the CSV file and sets its fields with the required information. Finally, an ArrayList, which contains all the Advanced commands rows as elements, is returned by dedicated methods and manipulated properly.

5. EVALUATION

For the evaluation of the diagnostics application we used two methods of communication. The first is remote and the second is point-to-point. The remote method requires two computer systems and of course a BLEDP card. The first computer system resides next to the Beam Loss Electronic Acquisition Crate (BLEAC) and is connected directly with it using the Ethernet SFP transceiver at 1 GBps. The diagnostics client is executed in the second computer, which initiates a connection with the BLEDP server after a request from the user. The request is sent through the CERN network to the first computer, which redirects to the corresponding BLEDP card. The point-to-point method is used when the client is executed in the computer system next to the BLEAC.

The final implementation of the client runs under both Linux and Windows 7 Operating Systems. For our tests we used a Linux system, featuring an Intel Core i7 2670QM processor running at 2.8 GHz, 4 GB of RAM and an Intel SSD Drive for the remote connection. For the point-to-point connection, a Windows 7 system was engaged, residing next to the Beam Loss Electronic Acquisition Crate (BLEAC) inside the BLM Laboratory or again a Windows 7 system next to the BLEAC in the Proton Synchrotron (PS) accelerator facility. Using the latter system, we can examine real data from the accelerator detectors, which are connected to the input channels of the BLEDP cards.

5.1 Deliverables and their validity

The final deliverable can be defined as a successfully implemented diagnostics system that meets the requirements of the Beam Loss Monitoring team. Thereby, the quality of the end product can be evaluated by the detailed analysis of the system characteristics against the list of requirements in Chapter 2.

- Communicating with the custom made server, implemented in the BLEDP FPGA, through the network. Each module has a separate Ethernet link and thus it is addressed individually by its unique address and port number. Each server is limited to accept only one client at a time. At this step it is necessary to decide which protocol (TCP/IP or UDP/IP) is suitable for the project's needs. The foreseen data rate should be taken into account.

  A user is able to establish a connection with the BLEDP server, through the Online Panel Interface of the client. All the necessary parameters, depending on the mode of acquisition, should be set before a Start Communication action. Otherwise, an error message should appear. It was decided, that the TCP protocol should be used for a single channel transmission and the UDP protocol for the multi-channel. The data rates of the two modes were the most important factor for this decision. Of course, the connection is always established through the TCP socket, regardless of the acquisition mode, as the UDP protocol is connectionless.

- Commanding the server. After establishing a connection with the server, the client should be able to send a command packet. There are two types of commands in the system. The first group of commands contains details of the acquisition request. There are also expert commands which can be sent to the server in order to set its internal registers. At this point a protocol for the commanding should be designed.

  The client's logic implements this feature. The protocol and the description of all the types of commands (Acquisition, Expert and Advanced) is presented in Chapter 4.3.1. Furthermore, the description and implementation of the Expert and Advanced commands is explained thoroughly in Chapters 4.4.6 and 4.4.6.1.

- Data sorting and processing. The server can send both mixed acquisition and status/diagnostic data. The protocol should be designed so the data type can be distinguished and processed accordingly.

  In Chapter 4.4.3, we explain the internal distribution process of the two data types. The acquisition, distribution and processing of the data is done by three different threads for better performance.

- Displaying in real time two different types of data, acquisition and/or status, for diagnostics purposes. This is the main function of the application and the final goal of the development process.

  The online acquisition tab and more precisely the graph component inside it is used for displaying the acquisition data in real-time. The status data tabs are used for displaying the status elements in parallel with the acquisition data. The refresh rate of the data in the graphs inside the status tabs is always 1 Hz. For the online graph, the refresh rate depends always on the measurement period field, but the user should be careful with this value, because a small measurement period could cause performance issues.

  We can examine in the following figures, some real-time data coming from a BLEDP server in the Proton Synchrotron (PS) accelerator facility. Different detectors are connected in the input of some channels of that system. All the following figures were captured during an online data transmission.

  Figures 33, 34 and 35 illustrate the real-time data-viewing of the two different types of data, distributed in three tabs, from a single-channel acquisition with a measurement period of 10000 μs. The type of detector in the acquisition channel of this example is an ionization chamber.

  In Figure 33, the averaged acquisition data plot in the online display is shown. The observable peaks indicate beam losses produced by the connected detector in channel 5. Obviously, the different components used for settings manipulation are disabled during an online session. Only the stop button and the Min-Max checkbox are enabled. Similarly, from the same acquisition, Figures 34 and 35 illustrate the status

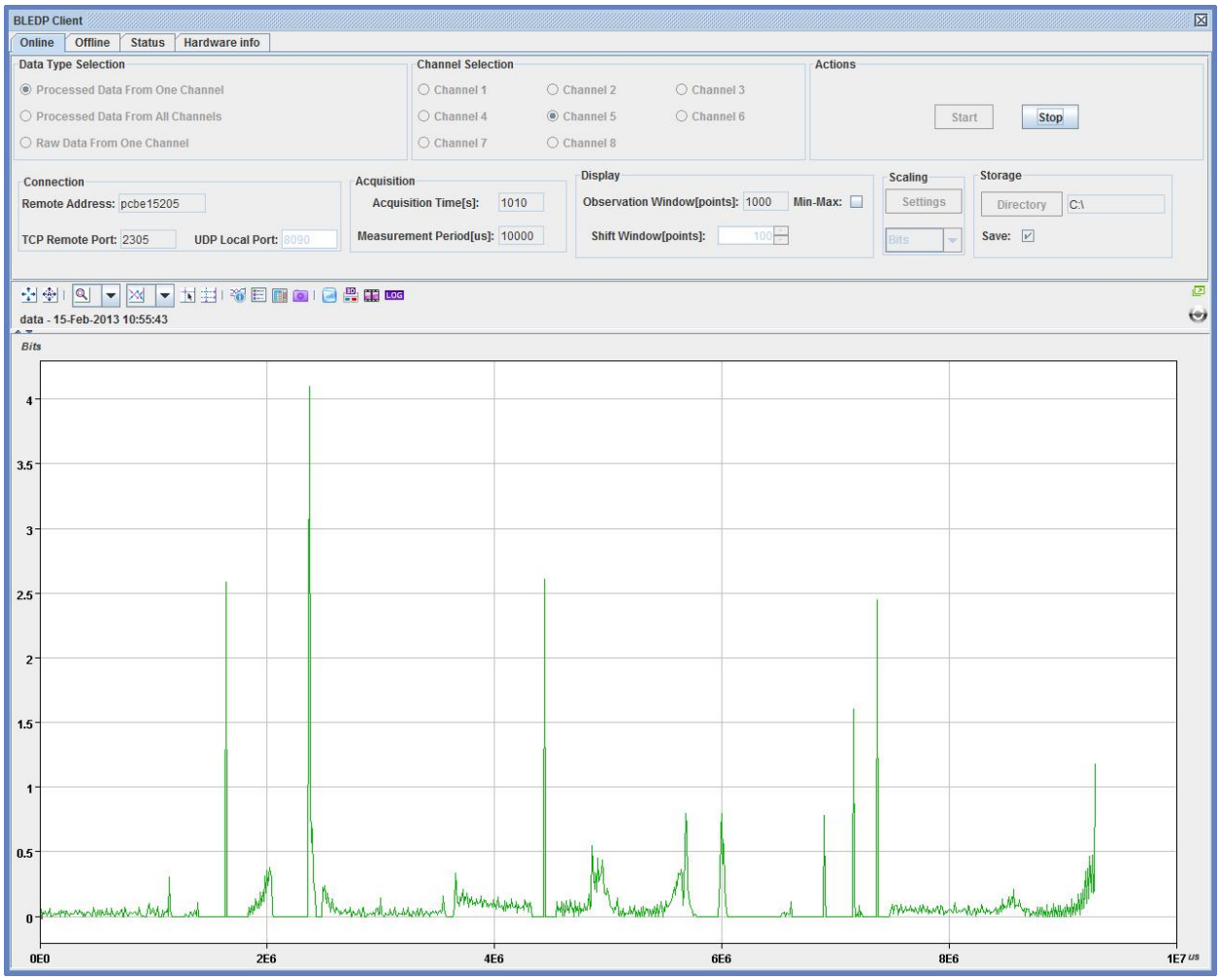

Figure 33: PS Single-Channel Acquisition. Online Tab.

data handling and viewing. We have to note here that the SFP2 graph and the corresponding components in Figure 35 remain inactive, because the connection in this case was achieved through the Ethernet SFP (SFP1). The optical fiber SFP (SFP2) was not even connected to the system during this acquisition.

Accordingly, we can examine Figures 36 and 37 which represent a multi-channel acquisition from the same system in the PS facility. We exclude the figure with the Hardware Info status tab, because it is identical to the single-channel transmission.

Figure 36 shows the online display during a multi-channel acquisition. Every averaged data plot corresponds to a specific channel. In this example, only three detectors are connected with the input of every channel, so the other channels produce simply a stable trace or noise. In this mode, a user is able to hide or reveal channel plots at will. The channel checkboxes are used for that purpose.

Likewise, Figure 37 indicates the first status tab in the multi-channel acquisition. The only difference from the single-channel mode is that the first data graph contains the data plots from all 8 channels forming a micrography of the online graph.

- Storing the acquired data -in parallel with the real time view- into files for further

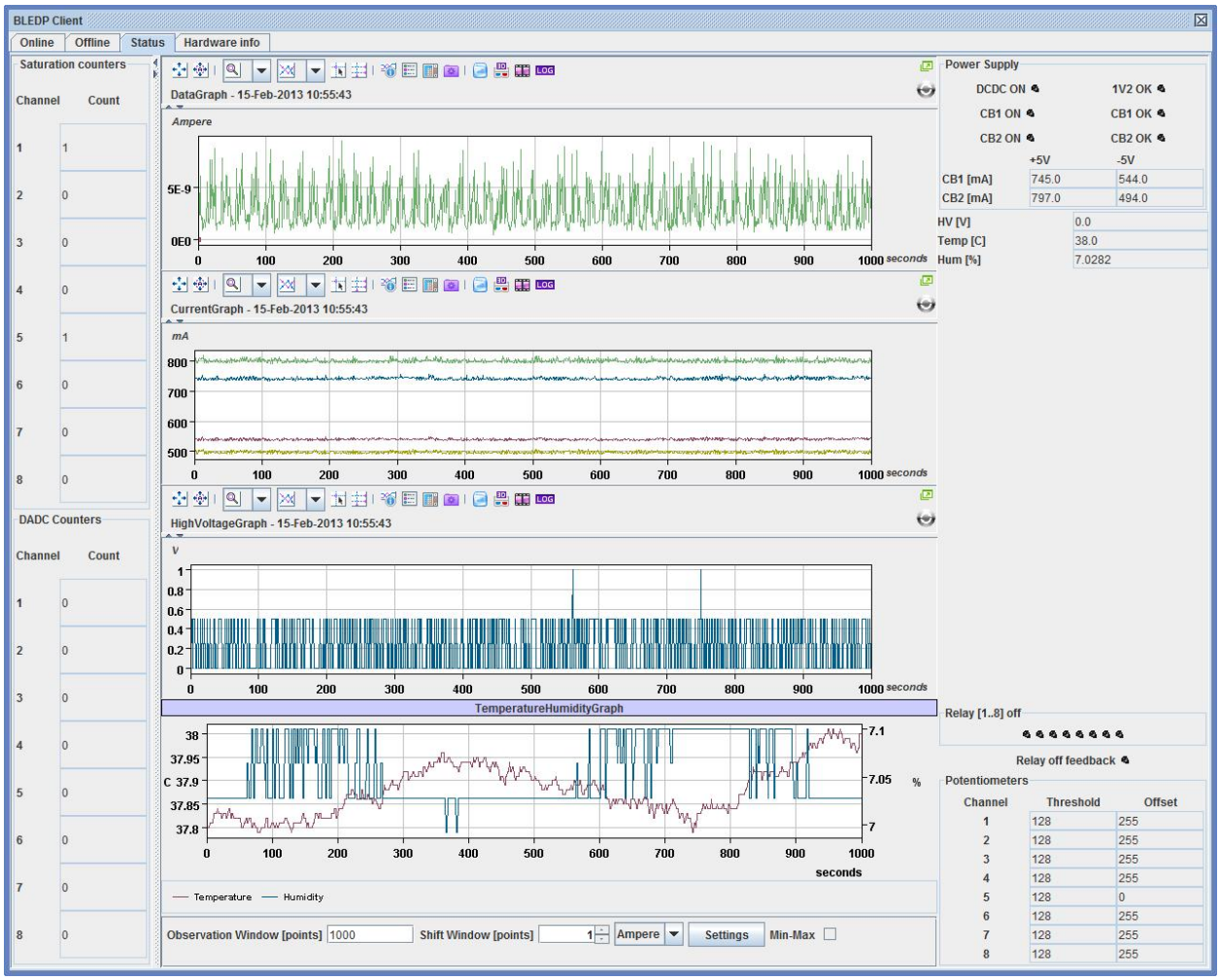

Figure 34: PS Single Channel Acquisition. Status Tab.

offline analysis. The logic for the file storing follows a specific format, so it will be easier for the user to find a precise time frame from a previous acquisition.

The parallel data storage is achieved with a process that is discussed in Chapter 4.4.3. The offline tab is used for further offline analysis of the acquired data. A full description of this interface is given in Chapter 4.4.4. In Figure 38, we can examine some real accelerator data, received and stored using the BLEDP client's online interface, from the BLM system inside the PS accelerator facility.

Figure 38 represents a glimpse of the offline tool capabilities. A user started by choosing the stored acquisition session. Afterwards, they picked the desired second from the picker graph (see Chapter 4.4.4). Inside the picked second, a beam loss occurred. At the time of acquisition, three different detectors were connected in channels 1, 2 and 5 of the BLEDP card. The lower graph of the offline analysis tool shows the different decays in the 3 channel traces, when the beam loss occurred. Each detector behaves differently to the loss and thus the observable data decay differs between the traces.

- Designing the graphical user interface. The application must be user friendly. To

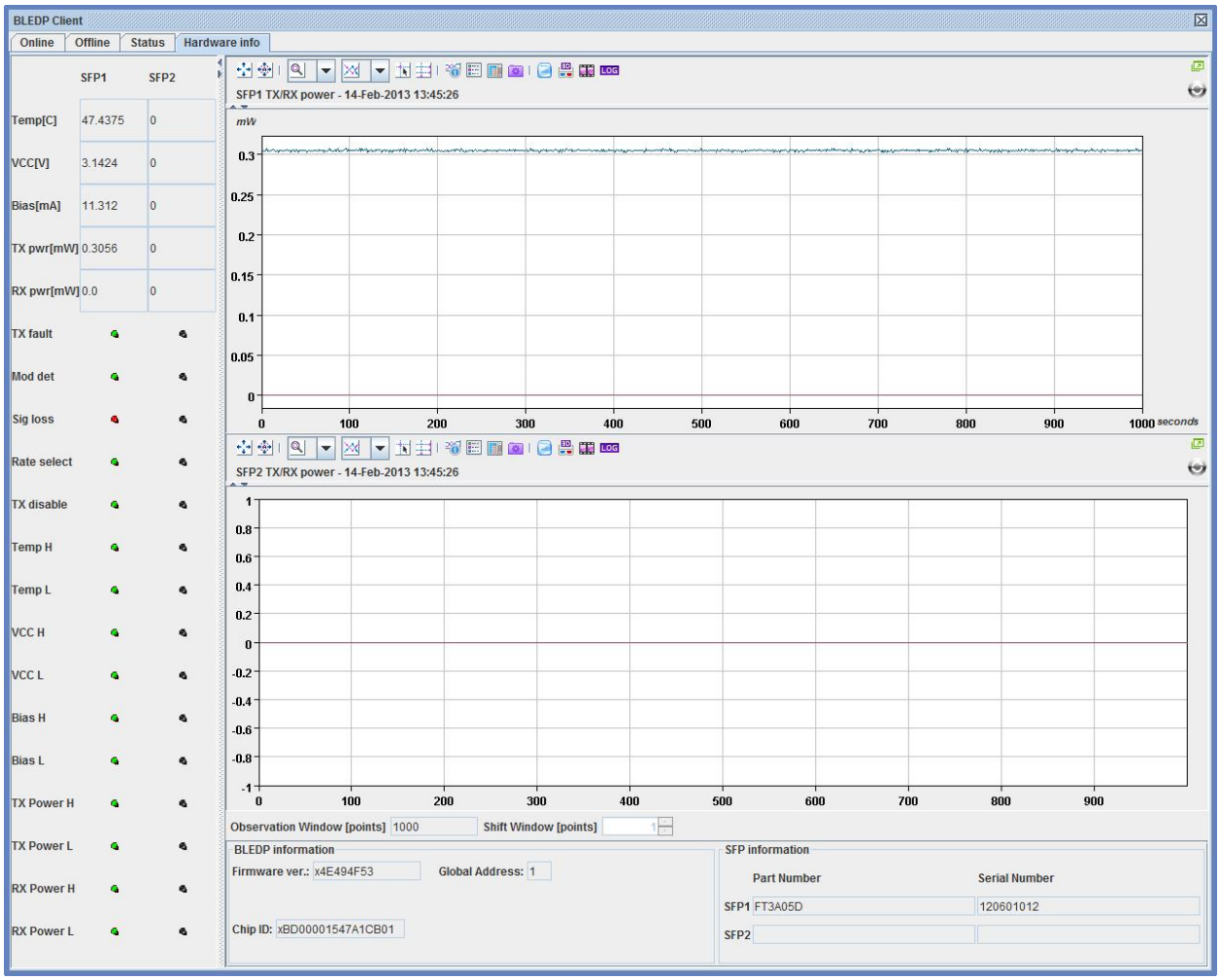

Figure 35: PS Single Channel Acquisition. Hardware Info Tab.

achieve this goal it should group functionalities of given category in a separate tab. It should also allow online real time data viewing and optional storage in the offline files. The online viewing of the acquisition and status data should be implemented by use of separate tabs. The file structure for the offline data should be proposed as well. The offline data analysis tool should be designed within the application. This stage of development makes use of the Java Swing framework. This framework provides a set of powerful and flexible components, which synthesize the look and feel of modern Java GUI applications.

The final outlook of the GUI was discussed and agreed between the members of the development team and of course the end users. It groups the different functionalities of the application into separate tabs, i.e. Online, Offline, Status, Hardware Info, Static and Advanced Commands. A user-friendly environment was achieved with the use of the Java Swing framework with several built-in components, as well as the JDataViewer[13] for the graphical representations.

- High data rate. The application will receive data streamed at a high rate (16 kbps in single-channel mode and 128 kbps in multi-channel mode). To manage online data display and offline data storage in parallel, multi-threading technology must be

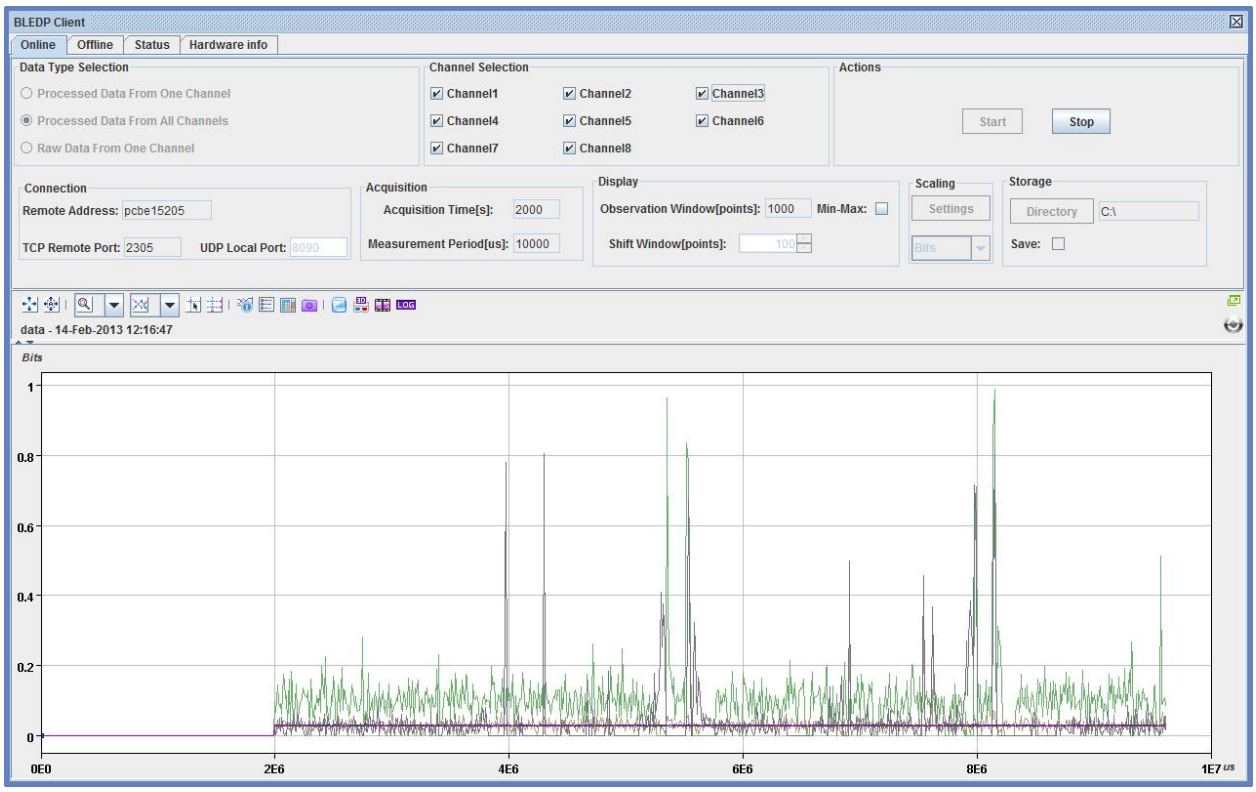

Figure 36: PS Multi Channel Acquisition. Online Tab.

engaged. Moreover, data reduction in the online view will be necessary.

The high data rate in both modes from the server's side was known from the beginning and considered as a very important fact during the development phases. A simple multi-threading process, described in Chapter 4.4.3, was employed to achieve parallel data socket acquisition, online plotting and storing. This parallel manipulation of incoming data streams is expected to boost the performance of the client, especially in multi-core systems. Furthermore, the data reduction in the online view is necessary and vital for the performance. The measurement period field should be set with a proper value for averaging, that won't cause any stability issues. Prior to any online session, a recommended minimum value of 400μs, ascertained after many tests, for measurement period should be used for stable performance.

5.1.1 Reliability Analysis

The reliability of the client application can be assessed by several methods. First of all, the main factor that impacts the system reliability is how well the end product meets the predefined requirements. This factor is indicated by the quality of the design and final implementation of the system discussed in Chapter 4. The final characteristics of the BLEDP client are compared against the list of the predefined requirements in the previous section. Summarizing the points presented there, the general conclusion that can be drawn is that the final system successfully fulfills the list of specifications at the required level. During the BLEDP cards development, the client application proved to be stable

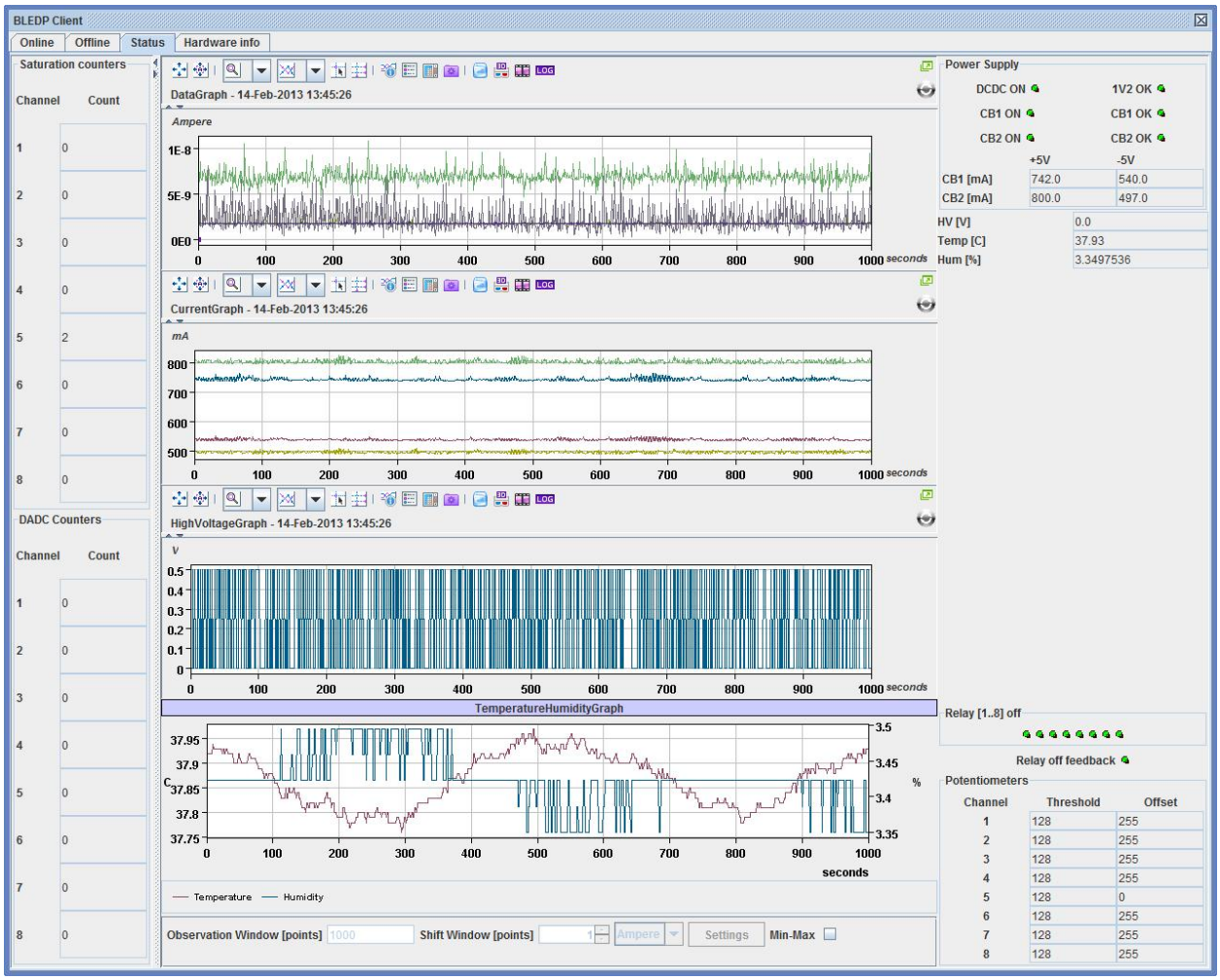

Figure 37: Multi Channel Acquisition. Status Tab.

enough to successfully perform all the demanded functions.

Another important factor that affects the reliability and effectiveness of the client application is its efficiency. This includes the system crash rate and unexpected behavior. Many problems of this nature arose during the development process, but the end product, after all the required testing and bug fixing, proved to be stable enough for operational usage.

A last important factor that reflects the success of the end product is the user satisfaction rate. This criteria is closely dependent on the system performance and behavior. The main users of this application are the developers of the BLEDP FPGAs, members of the BLM team and also sometimes the operators' crew in the Cern Control Center (CCC). During the development period, the application received many positive responses from its users, whereby it could be concluded that it satisfies the main function it is employed to fulfill.

5.1.2 Performance

It could be very interesting to measure the performance of the application in a typical computer system, such as the one mentioned for the remote connection in the beginning

Figure 38: Multi Channel Offline tool. BLM Detectors Decay.

of this chapter. For that purpose, we ran 4 indicative 100 seconds online acquisitions in single and multi-channel modes with storing and not storing data to the hard drive. We used the time command to measure the performance. The quoted results per case follow:

☛ **Single-Channel 100s, no save.**

Command being timed: ”java -jar BLEDPClient.jar”
User time (seconds): 52.12
System time (seconds): 7.83
Percent of CPU this job got: 53%
Elapsed (wall clock) time (h:mm:ss or m:ss): 1:52.46
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1364400
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 139854
Voluntary context switches: 1479949
Involuntary context switches: 9336
Swaps: 0
File system inputs: 168
File system outputs: 240

Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

☛ **Single-Channel 100s, save.**

Command being timed: ”java -jar BLEDPClient.jar”
User time (seconds): 57.44
System time (seconds): 9.06
Percent of CPU this job got: 56%
Elapsed (wall clock) time (h:mm:ss or m:ss): 1:56.93
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1414992
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 140258
Voluntary context switches: 1492263
Involuntary context switches: 11040
Swaps: 0
File system inputs: 1104
File system outputs: 393296
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

☛ **Multi-Channel 100s, no save.**

Command being timed: ”java -jar BLEDPClient.jar”
User time (seconds): 101.30
System time (seconds): 10.01
Percent of CPU this job got: 93%
Elapsed (wall clock) time (h:mm:ss or m:ss): 1:59.04
Average shared text size (kbytes): 0

Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 2026208
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 174890
Voluntary context switches: 2271883
Involuntary context switches: 22096
Swaps: 0
File system inputs: 0
File system outputs: 208
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

☛ **Multi-Channel 100s, save.**

Command being timed: ”java -jar BLEDPClient.jar”
User time (seconds): 133.31
System time (seconds): 14.67
Percent of CPU this job got: 124%
Elapsed (wall clock) time (h:mm:ss or m:ss): 1:58.40
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 2229312
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 183921
Voluntary context switches: 2478495
Involuntary context switches: 29019
Swaps: 0
File system inputs: 16
File system outputs: 3127672
Socket messages sent: 0
Socket messages received: 0

Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

A lot of useful information can be drawn after observing the above results. For example, the CPU usage varies from case to case with multi-channel mode consuming more CPU time. Furthermore, the file system outputs rises up to a high value when data storage is on. In general, the final deduction after studying these indicative results is that this application is stable and medium resource consuming for an average computer system.

6. CONCLUSION

The BLEDP diagnostics client application was designed and developed as a non-commercial CERN internal tool to satisfy the needs of the Beam Loss Monitoring team. The development of the system was fully implemented inside the Software section in collaboration with the BLM team, which set certain specific characteristics to the working process. It is very important the fact that in the confined working environment the developer and the end users of the system constantly interact with each other, which speeds up the development process and provides the necessary feedback instantly.

As for the technical details, it is very important to mention that the concept of developing the diagnostics client in the first place, is based on the need to debug and verify the final hardware, as well as further develop the analogue circuits and the final firmware. Hence, this test system implementation with a direct Ethernet communication facilitated a standalone implementation, before the final system, using this client. This GUI allowed the development of both firmware and hardware along with the detailed analysis of the Real-Time algorithm implemented inside the FPGA.

The future work regarding the whole BLM for the Injectors project, includes developing software for controlling the new BLM electronics after installation in the injectors, i.e. usage of processing electronics and FESA servers to provide data to clients to deal with settings and interlocks (see Chapter 3). This is going to be the final system implementation following the standard way at CERN, after the full development and validation of the new BLM electronics. Our proposed test system implementation with the direct-communication diagnostics client, detours the aforementioned procedure and allows the development and validation of the hardware, before the final operational usage. Eventually, it can serve as a future standalone diagnostics software implementation for these instruments.

TABLE OF TERMINOLOGY

Below is a sample of a table of terminology.

επιταχυντής	accelerator
κλάσση	class
εντολή	command
γραφικό περιβάλλον	graphical interface
πελάτης	client
ενσωματωμένος διακομιστής	embedded server
νήμα	thread
επικοινωνία	communication
εκροή σωματιδίων	beam loss

ABBREVIATIONS AND ACRONYMS

Below is a sample of abbreviations and acronyms .

CERN	European Organization for Nuclear Research
LHC	Large Hadron Collider
FPGA	Field Programmable Gate Array
BLEAC	Beam Loss Electronic Acquisition Crate
BLEDP	Beam Loss Electronic Dual Polarity
GUI	Graphical User Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
MAC address	media access control address
JDVE	JDataViewer
CSV	Comma-seperated values
SFP	Small form-factor pluggable transceiver
DADC	Direct Analogue Digital Conversion
FDFC	Fully Differential Frequency Converter
FESA	Front End Software Architecture

REFERENCES

[1] CERN in a nutshell, Available: http://public.web.cern.ch/public/en/About/About-en.html.

[2] CERN's Structure, Available: http://public.web.cern.ch/public/en/About/Structure-en.html.

[3] BI-SW Section page, Availabe: http://project-beam-instr-sw.web.cern.ch/project-beam-instr-sw/Welcome.php.

[4] BI-BL Section page, Availabe: http://sl-div-bi-pb.web.cern.ch/sl-div-bi-pb/.

[5] CERN's Accelerator Complex, Availabe: http://public.web.cern.ch/public/en/research/AccelComplex-en.html.

[6] LHC the guide, Availabe: http://cdsweb.cern.ch/record/1092437/files/CERN-Brochure-2008-001-Eng.pdf.

[7] C. Zamantzas, *"BLM for the Injectors Project, Linac4 and PSB".*

[8] Linac4 Project, Availabe: http://linac4-project.web.cern.ch/linac4-project/.

[9] Kay Wittenburg, *"Beam Loss Monitoring and Control"*, Proceedings of EPAC 2002, Paris, France.

[10] M. Kwiatkowski, M. Alsdorf, E. Angelogiannopoulos, B. Dehning, S. Jackson, W. Vigano, C. Zamantzas, *"DEVELOPMENT OF A BEAM LOSS MEASUREMENT SYSTEM WITH GIGABIT ETHERNET READOUT AT CERN",* Tsukuba, Japan, Oct 1-4, 2012.

[11] Herbert Schildt, *"Java the Complete Reference 8th Edition",* Oracle Press.

[12] Oracle, http://docs.oracle.com/javase/tutorial, November 2012.

[13] CERN, JDataViewer Charting Library, https://espace.cern.ch/be-dep/CO/ICALEPCS%202009/1398%20%20JDataViewer%20-%20Java-based%20Charting%20Library/THP093.pdf

[14] CERN, Christos Zamantzas, Marcel Alsdorf, William Vigano, *"NOTES ON THE BEAM LOSS ACQUISITION CRATE"*

[15] CERN, Christos Zamantzas, *"BLM FOR THE INJECTORS PROJECT",* Linac4 Coordination Committee, May 2012.

[16] CERN, Maciej Kwiatkowski, *"BLEDP CARD COMMUNICATION PROTOCOL OVER GIGABIT ETHERNET",* May 2012.

[17] W. Vigano, B. Dehning, E. Effinger, G. G. Venturini, C. Zamantzas, *"COMPARISON OF THREE DIFFERENT CONCEPTS OF HIGH DYNAMIC RANGE AND DEPENDABILITY OPTIMISED CURRENT MEASUREMENT DIGITISERS FOR BEAM LOSS SYSTEMS",* Tsukuba, Japan, Oct 1-4, 2012.

[18] M. Kwiatkowski, C. Zamantzas, M. Alsdorf, B. Dehning, W. Vigano, *"A REAL-TIME FPGA BASED ALGORITHM FOR THE COMBINATION OF BEAM LOSS ACQUISITION METHODS USED FOR MEASUREMENT DYNAMIC RANGE EXPANSION",* Tsukuba, Japan, Oct 1-4, 2012.

[19] M. Kwiatkowski, C. Zamantzas, M. Alsdorf, B. Dehning, W. Vigano, S. Jackson, *"SYSTEM ARCHITECTURE FOR MEASURING AND MONITORING BEAM LOSSES IN THE INJECTOR COMPLEX AT CERN",* Tsukuba, Japan, Oct 1-4, 2012.

[20] Alexander Schwinn, Solveigh Matthies, Dorothea Pfeiffer(GSI, Darmstadt) , Michel Arruat, Leandro Fernandez, Frank Locci, David Gomez Saavedra (CERN, Geneva), *"FESA 3, THE NEW FRONT-END SOFTWARE FRAMEWORK AT CERN AND THE FAIR FACILITY",* Proceedings of PCaPAC 2010, Saskatoon, Saskatchewan.

[21] AFCT-57J5APZ Data Sheet, http://www.avagotech.com/docs/AV02-0680EN