Tartu University

Faculty of Science and Technology

Institute of Technology

Jüri Jõul

Software development for a high-speed digitizer to provide online access to longitudinal bunch profiles in the Large Hadron Collider

Master's thesis (30 EAP)
Robotics and Computer Engineering

Supervisors:

Veiko Vunder, PhD
Thomas Levens, MEng

Geneva 2023

Resümee/Abstract

Kõrge diskreetimissagedusega andmehõiveseadmele tarkvara arendamine Suure Hadronite Põrguti piki-kimbuprofiilidele kaugligipääsu tagamiseks

Osakestekiirendid nagu Suur Hadronite Põrguti (LHC) püüdlevad kõrgema heleduse poole, et koguda füüsikaeksperimentide jaoks võimalikult palju andmeid. Kiirendite parema soorituse saavutamiseks on oluline sooritust esmalt mõõta. Kiirendifüüsikud on pakkunud välja uusi diagnostikameetodeid, mis kasutavad seinavoolumonitoridelt pärinevaid piki-kimbuprofiile. Seni pole CERN-is olnud nendele profiilidele piisavalt hea jõudlusega kaugligipääsuvõimekust, et võimaldada uusi diagnostikameetodeid. Selle magistritöö tulemusena valmis kontseptsiooni tõendamiseks mõeldud tarkvaralahendus LHC seinavoolumonitoride andmete hõivamiseks ning kaugligipääsetavuse võimaldamiseks. Töö käigus arvestati olemasolevast riist- ja tarkvarast tulenevate piirangutega ning kasutajate vajadustega. Tarkvaralahendust koos tema osaks olevate algoritmidega valideeriti LHC-st pärinevate tõeliste seinavoolusignaalidega.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; P175 Informaatika, süsteemiteooria; T121 Signaalitöötlus

Märksõnad: tarkvaraarendus, andmehõive, kiireinstrumentatsioon, seinavoolumonitor, piki-kimbuprofiil, FESA

Software development for a high-speed digitizer to provide online access to longitudinal bunch profiles in the Large Hadron Collider

Particle accelerators such as the Large Hadron Collider (LHC) strive for high luminosity in order to produce as much data for physics experiments as possible. Measuring the performance of an accelerator is a crucial step in improving it. Accelerator physicists have proposed novel diagnostic methods utilising longitudinal bunch profiles from wall current monitors. So far at CERN, online access to these profiles with the required performance to enable new diagnostic methods did not exist. As a result of this thesis, a proof-of-concept software solution to acquire and provide online access to wall current monitor data from the LHC was created. In the process, the limitations stemming from the existing hardware and software resources, as well as the users' needs were considered. The software solution along with its component algorithms was validated with the acquisition of real wall current signals from the LHC.

CERCS: T120 Systems engineering, computer technology; P175 Informatics, systems theory; T121 Signal processing

Keywords: software development, data acquisition, beam instrumentation, wall current monitor, longitudinal bunch profile, FESA

Contents

List of Figures

Abbreviations, Constants, Generic Terms

ADC - Analogue to Digital Converter

API - Application Programming Interface

APWL - the specific type of Wall Current Monitor installed in the LHC

CERN - European Organization for Nuclear Research

CPU - Central Processing Unit

CMW - Controls MiddleWare

fBCT - fast Beam Current Transformer

FEC - Front End Computer

FESA - Front End Software Architecture

HDF5 - Hierarchical Data Format, version 5

IO - Input/Output

JSON - JavaScript Object Notation, a text-based file format

LHC - Large Hadron Collider

PC - Personal Computer

PCI - Peripheral Component Interconnect

PCIe - PCI Express

PS - Proton Synchrotron

PUB - ZeroMQ publisher socket

RAM - Random Access Memory

RDA - Remote Device Access

REP	- ZeroMQ reply socket
REQ	- ZeroMQ request socket
RF	- Radio Frequency
RPC	- Remote Procedure Call
SDK	- Software Development Kit
SPS	- Super Proton Synchrotron
SUB	- ZeroMQ subscriber socket
WCM	- Wall Current Monitor
ZMQ	- ZeroMQ, an open-source messaging library
XML	- eXtensible Markup Language, a text-based file format

1 Introduction

The European Organization for Nuclear Research (CERN) hosts the world's largest particle physics laboratory. Its accelerator complex accelerates particles to high energies and close to the speed of light, and then collides them in order to look for new physics beyond the Standard Model of particle physics. To capture these collisions, large particle detectors have been constructed and are operated at CERN as international collaborations between institutes. Due to the statistical nature of particle physics, the experiments need to analyse as many collision events as possible. To meet their demands, the accelerator complex needs to be run at peak performance.

At CERN, the Beam Instrumentation group is responsible for monitoring various parameters of the particle beams circulating in the accelerators, such as the beam's horizontal and vertical position inside the accelerator, effectively providing diagnostic information about the accelerator. Common causes for reduced accelerator performance are different types of oscillations of particle beams. Based on the readings of multiple instruments, these oscillations can be measured and the accelerator operators can use these measurements to optimise the performance, thereby increasing the number of collisions produced for the experiments.

One possible instrument to gather measurements is the wall current monitor (WCM), which is used to determine the distribution of particles along the circumference of an accelerator. Instead of a continuous beam, the particles in a particle accelerator are often clumped together into bunches. The length of these bunches is usually in the order of nanoseconds, with tens of nanoseconds of space between consecutive bunches. A WCM provides a longitudinal profile of these bunches, or how the particles are distributed inside a bunch. Monitoring how these profiles change provides a lot of information about the energies and impulses of the constituent particles. Novel diagnostic methods have been proposed by physicists to make use of these longitudinal bunch profiles from the WCM signals in CERN's largest accelerator – the Large Hadron Collider (LHC). As there is currently no way to access the WCM signals in the LHC with the required performance to enable new diagnostic methods, these signals need to be digitised and made available online for further analysis.

All Beam Instrumentation group's instruments need to be accessible and controllable in a standardised way, in order to save on development and operation costs. As such, to integrate any new instrument into CERN's existing software ecosystem, certain frameworks need to be used. One of them is the Front End Software Architecture (FESA) framework, which enables real-time control and online readout of measurement devices. In addition to providing standardised software resources, the group has provisioned hardware for digitising these instruments, such as high-speed digitizers.

1.1 Problem overview

Considering the motivation of enabling new diagnostic methods to improve accelerator performance, the main goal of this thesis is to develop software to provide online access via FESA to longitudinal bunch profiles from the WCMs in the LHC, acquired by existing Guzik Technical Enterprises (Guzik) ADC6044 digitizers. Three sub-goals are formulated as follows:

- Perform exploratory analysis and benchmarking to find out the capabilities and constraints of the digitizers.
- Gather and analyse the needs of the users, using them as a basis to formulate software requirements.
- Implement a proof-of-concept solution, maximally utilising the hardware while maintaining the flexibility needed for an iterative beam instrumentation development process.

1.2 Thesis organisation

This thesis is divided into six main parts. The current chapter introduces the problem. Chapter 2 gives a detailed overview of CERN, particle accelerators, and why longitudinal bunch profiles need to be acquired and analysed. In Chapter 3, benchmarking and experimentation with the high-speed digitizer are described, along with a brief introduction to software development. Chapter 4 analyses the users' needs, and finds trade-offs between them and the previously found limitations. Requirements for the software are also outlined. Chapter 5 gives an overview of the developed software, and Chapter 6 validates that the software meets the users' expectations. The final chapter concludes the work and gives a short overview of future developments.

2 Background

2.1 Overview of CERN

In 1952, an agreement between 11 countries was signed, founding the European Council for Nuclear Research (CERN) to investigate the possibility of creating a European atomic physics laboratory. Following the council's foundational work in Copenhagen, in 1954, 12 countries dissolved the provisional council and established the European Organization for Nuclear Research, keeping the acronym known today. When CERN started, the Standard Model of particle physics did not yet exist and thus the structure of matter was not fully known [1]. As more discoveries were made, it gradually shifted its aim from performing atomic physics to particle or high energy physics.

After confirming the correctness of the Standard Model with the observation of the Higgs boson in 2012, particle physicists at CERN are now looking to discover new physics beyond the Standard Model to answer fundamental questions about our universe [2]. To probe how the universe works, particles are smashed together at very high energies, thereby producing showers of other more exotic particles which are then carefully studied. Although CERN started small with the construction of its first particle accelerator, the Synchro-Cyclotron in 1957, over the years it has grown into the largest particle physics laboratory in the world [3]. Progressively larger accelerators to reach higher energies, as well as various experiments to better detect particles, have been constructed.

Fundamentally, high collision energies are required to produce collision events that offer new insights into physics. Since these events are very rare, a large number of collisions is needed to produce a significant amount of them. Therefore, the number of particles in a beam – the beam intensity – as well as how tightly they are packed when they collide with each other are also important. Both of these aspects are combined in a parameter named luminosity, which describes how many collisions take place in a given time period in unit area.

Its current accelerator complex consists of over ten particle accelerators and decelerators [4]. The largest and most powerful accelerator is the Large Hadron Collider (LHC) with its 27 km circumference and 7 TeV beam energy [5]. High energy proton-proton collisions produced in the LHC lead to showers of particles which are studied by physicists in hopes of finding new fundamental particles [6]. Sometimes, heavy ions are collided instead of protons to produce a state of matter called the quark-gluon plasma, mimicking the conditions found in the universe right after the Big Bang [7]. Due to having high enough energy and luminosity to produce a significant number of rare events, the LHC is often called "the discovery machine" [8].

2.1.1 The accelerator complex and experiments

Fig. 2.1 shows the accelerator complex at CERN at the time of writing. The main chain of accelerators aimed at producing proton-proton collisions starts with the hydrogen anion source and LINAC 4 (the linear accelerator number four), where initial acceleration is performed. After LINAC 4, the negative hydrogen ions are stripped of their electrons, leaving only protons, which are then further accelerated by the Proton Synchrotron Booster (PSB) [9]. Then, they are consecutively accelerated to even higher energies in the Proton Synchrotron (PS) and the Super Proton Synchrotron (SPS) before finally being injected into the LHC where the two particle beams are finally collided. During heavy ion runs, particles start from LINAC 3, and pass on to LEIR, the Low Energy Ion Ring, from which they are injected into the PS and the rest of the injection chain is the same, ending with the LHC [10]. Although particles are commonly accelerated, they are sometimes also decelerated in order to precisely study their properties. This is why CERN hosts the Antiproton Decelerator (AD) and the Extra Low ENergy Antiproton decelerator (ELENA): to produce and perform experiments on antimatter such as antihydrogen [11].

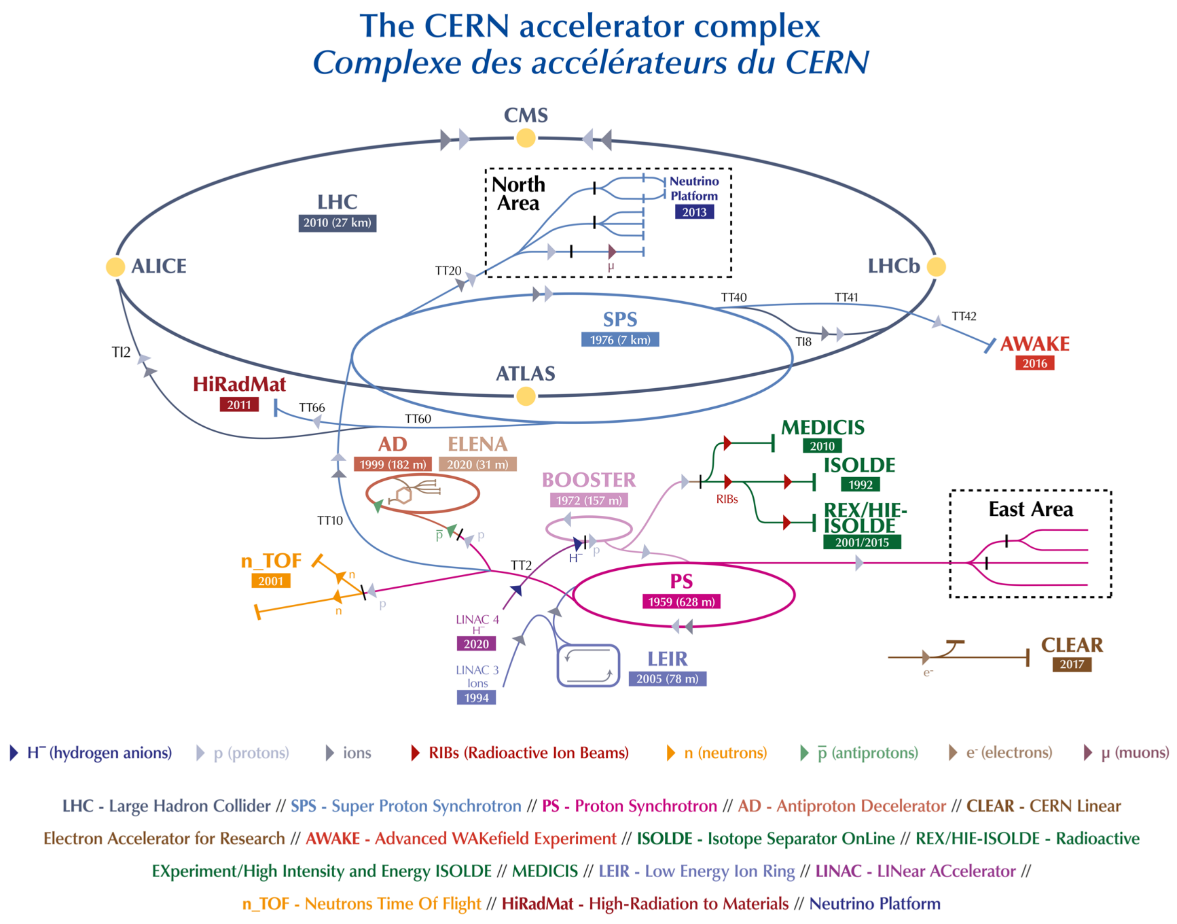

The CERN accelerator complex
Complexe des accélérateurs du CERN
CMS
North Area
Neutrino Platform 2013
LHC 2010 (27 km)
ALICE
LHCb
TT20
TT40
TT41
TT42
SPS 1976 (7 km)
TI8
AWAKE 2016
TI2
ATLAS
HiRadMat 2011
TT66
TT60
MEDICIS 2010
ISOLDE 1992
AD 1999 (182 m)
ELENA 2020 (31 m)
BOOSTER 1972 (157 m)
RIBs
REX/HIE-ISOLDE 2001/2015
East Area
TT10
n_TOF 2001
TT2
PS 1959 (628 m)
LINAC 4 H⁻ 2020
LINAC 3 Ions 1994
LEIR 2005 (78 m)
CLEAR 2017
H⁻ (hydrogen anions)
p (protons)
ions
RIBs (Radioactive Ion Beams)
n (neutrons)
p̄ (antiprotons)
e⁻ (electrons)
μ (muons)
LHC - Large Hadron Collider // SPS - Super Proton Synchrotron // PS - Proton Synchrotron // AD - Antiproton Decelerator // CLEAR - CERN Linear Electron Accelerator for Research // AWAKE - Advanced WAKefield Experiment // ISOLDE - Isotope Separator OnLine // REX/HIE-ISOLDE - Radioactive EXperiment/High Intensity and Energy ISOLDE // MEDICIS // LEIR - Low Energy Ion Ring // LINAC - LINear ACcelerator // n_TOF - Neutrons Time Of Flight // HiRadMat - High-Radiation to Materials // Neutrino Platform

Figure 2.1: CERN accelerator complex as of January 2022 [12]. According to the legend seen at the bottom of the image, little triangles on accelerator contours are colour-coded by the type of particles used, while the accelerators themselves and experiments are colour-coded arbitrarily

Producing collisions is the goal of the LHC but without detectors, the results of these collisions could not be quantified. The LHC has four intersection points where the two counter-rotating particle beams cross. Its four main experiments – ATLAS, CMS, ALICE, and LHCb – are located at these points. These experiments are large international collaborations between institutes, being effectively the clients of the accelerator complex at CERN [13]. Due to the nature of smaller accelerators injecting into larger ones, once the largest – the LHC – is completely filled up, previous accelerators in the chain would sit idle. To avoid that, accelerators besides the LHC are also used for fixed-target experiments, as can be seen from Fig. 2.1 – symbolised by T-sections at the end of lines.

2.1.2 Contributing to society

In addition to performing fundamental physics research, CERN also contributes to society in various other ways. In order to achieve its scientific objectives, CERN constantly advances the frontiers of technology, enabling not only knowledge transfer leading to high-tech industrial applications of particle accelerators and detectors but also spin-off technologies [14]. As examples of the former category, some applications related directly to the accelerators and detectors at CERN are positron emission tomography and computed tomography [15], oncological hadron therapy [16], and colour X-ray computed tomography [17], to name a few. Famous examples of the latter include the creation of the World Wide Web, invented at CERN in 1989 [18]. The World Wide Web has been estimated to have contributed to around 2.9% of the global GDP in 2011 [19].

CERN's contributions to society also stem from its procurement procedures, educating and training new generations of scientists and engineers, and public outreach. About half of CERN's yearly budget is invested in technical partnerships with industries from which it procures either final products, such as detectors, or components contributing directly to the economies of its member states [20]. With respect to education and training, CERN collaborates with various institutes and universities, hosting young researchers and engineers, as well as school students and teachers [20]. Finally, in the category of public outreach, visits and exhibitions are offered, along with its artists-in-residence program since 2012 [20, 21].

The overall impact of CERN and its large-scale infrastructure projects on society has been the subject of several studies and reports. According to Florio, Forte, and Sirtori, the lifetime expected benefit/cost ratio of CERN's largest project – the LHC – was determined to be around 1.2, meaning the total sum of social benefits and technical spillovers, not including the benefits of fundamental physics research, is 20% higher than its operating and capital expenditure [22]. Furthermore, the High Luminosity upgrade to the LHC, a presently ongoing project, is estimated to pay back the society 1.7 times its investments [23]. A study by the Organisation for Economic Co-operation and Development (OECD) also reaffirms the socioeconomic benefits that large-scale research infrastructures such as CERN bring with them, highlighting additionally its role in international collaboration, as well as its status as a European world-class research facility [24].

2.2 Working principles of particle accelerators

Particle accelerators, as their name implies, are built to accelerate particles to high energies. According to Newton's second law of motion, in order to accelerate a body, e.g. a particle, that is to change its momentum, some force needs to be acted upon it [25]. In the case of charged particles, an electric field can be used for this purpose. The Lorentz force or electromagnetic force relates a charge in an electric field to the force acting upon it [26]. From Equation 2.1, it can be seen that both the direction and magnitude of the electric field $\mathbf{E}$, as well as the charge q play a role in determining the vector of the resulting force $\mathbf{F}$. Acceleration, in terms of increasing the energy, is not possible with the magnetic field $\mathbf{B}$, as this acts perpendicular to the particle motion $\mathbf{v}$.

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{2.1}$$

There are two categories of particle accelerators distinguished by the type of electric field used: electrostatic and electrodynamic [27]. As can be understood from their names, static and dynamic electric fields, respectively, are used in these machines to give energy to particles. Popular examples of the former are the Cockroft-Walton and Van de Graaff generators [28]. Because particle energy is proportional to the voltage generated by the electrostatic accelerator, very high voltages required will lead to a dielectric breakdown of the insulator, even theoretically of vacuum. The resulting short circuit reduces the voltage and therefore particles receive less energy. Therefore, due to the lower energies achieved, such accelerators are not widely used for particle physics today [29]. To work around the limitation of dielectric breakdowns, electrodynamic accelerators use oscillating electric fields to repeatedly impart energy on particles, allowing the voltage to be kept lower but accumulating higher final energy [30].

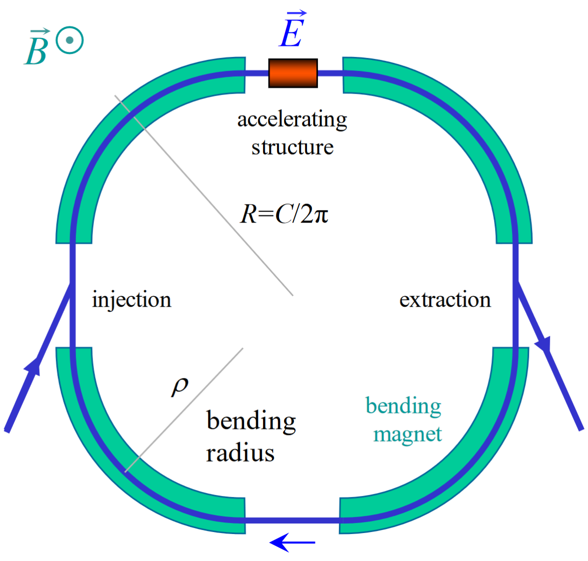


Figure 2.2: A schematic overview of a circular accelerator [26]. The electrodynamic accelerating structure (brown tube), as well as bending dipole magnets (in green), can be seen. In addition, the injection and extraction lines of the accelerator (blue, with arrows) are shown

In the case of an electrodynamic linear particle accelerator, often called a linac, a particle can therefore be accelerated in a straight line if a suitable oscillating electric field is generated [30]. However, the amount of energy linear accelerators can impart to particles depends on their length, and with their length being finite, they are often not sufficient to give enough energy to particles to generate interesting collisions from a physics perspective. This gives rise to circular accelerators whose electrodynamic accelerating principle remains the same, however, particles are now steered such that they form a closed loop. This allows the machine to continuously accelerate particles over multiple turns, creating an effectively infinitely long accelerator, thus giving the accelerated particles much higher energy. As an example of the difference in beam energy, CERN's LINAC 4, accelerates particles to 160 MeV, while the LHC accelerates them to 7 TeV – over four orders of magnitude higher. A simplified schematic of such a circular electrodynamic accelerator can be seen in Fig. 2.2, where an accelerating structure is imparting energy on particles every accelerator turn.

To build such a closed loop, the trajectories of particles need to be bent. To do this, the second part of the Lorentz force equation, concerning the presence of a magnetic field, is used. As can be deduced from Equation 2.1, charged particles experience a force orthogonal to the magnetic field and the direction of their movement. As a consequence, ordinary or electromagnets can be used for bending the trajectories. In Fig. 2.2, there are four such bending magnets displayed.

Naturally, there is a limit on how strong a magnetic field can be artificially generated to bend particles' trajectories while having the magnets still fit in the accelerator structure. Although faster particles experience a stronger centripetal force due to the Lorentz force, in the case of ultra-relativistic particles, the increase in mass far outweighs the increase in the almost constant speed near the speed of light. From Newton's first law of motion, the more mass a particle has the more force it takes to alter its trajectory, meaning that it gets more difficult to steer the particles as the energies increase [25]. At some point, the magnetic fields required to maintain a closed-loop trajectory exceed the amount that is possible to generate. Even superconducting magnets such as those used in the LHC are not able to achieve more than 8.3 T [6]. Therefore, the higher the energy, the larger the radius of a circular accelerator has to be, too.

2.3 Synchrotron motion and longitudinal phase space

There are some consequences to using oscillating electric fields, fundamentally determining the particle beam dynamics in an accelerator. Phenomena inherent to circular particle accelerators, also known as synchrotrons, are presented, along with methods to describe the behaviour of particles in such machines. To better discuss these phenomena, first, a coordinate system is defined in Fig. 2.3.

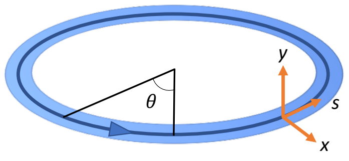


Figure 2.3: The coordinate system of a circular accelerator. Based on a schematic by Baird [31]

Coordinates x and y signify the horizontal and vertical transverse axes, while s is the longitudinal axis or phase θ, varying between 0 and 2π radians. Particles move around the circumference of an accelerator along the s-axis, meaning their phase is constantly increasing. Currently, no transverse motions are considered. In the case of highly relativistic particles travelling in a circular accelerator, such as in the LHC, the higher the energy of a particle, the lower its revolution frequency around the accelerator. This is due to having higher mass and therefore higher inertia which in turn leads to longer orbits, as is illustrated in Fig. 2.4 [26]. Conversely, particles with lower energy have higher revolution frequencies. The effects of this phenomenon will now be considered in the context of oscillating electric fields.

An assumption is made that in some place or phase in the accelerator, there is, in the S-axis, a sinusoidal oscillating electric field of some frequency being generated. Most often, radio frequency (RF) electric fields would be used for this. Again due to Lorentz force, this field can apply some force to the particles in it, changing their energies. By being periodic, this sinusoidal

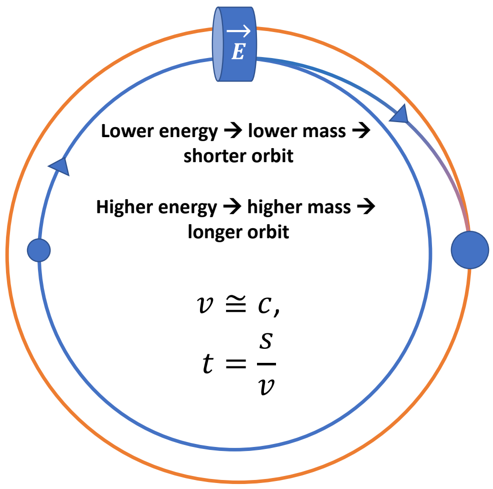

$\vec{E}$
Lower energy → lower mass → shorter orbit
Higher energy → higher mass → longer orbit
$v \cong c,$
$t = \frac{s}{v}$

Figure 2.4: Particles with higher energies have more mass and inertia. Therefore, magnets are not able to bend their trajectories as much and they end up on longer orbits

voltage constantly varies in time or phase, so the particles passing by the RF voltage generator would see a voltage of different magnitude and sign based on their time of arrival, or their phase in the accelerator in relation to the generator. Let there be a particle whose revolution frequency is just right so that it always arrives at the generator when its output is zero volts, crossing from positive voltage to negative. This particle, being synchronous to the RF frequency, would get no additional energy, neither would it lose any, meaning its revolution frequency would also stay constant, assuming no other energy loss mechanisms [31]. If some other particle is a bit slower and arrives later than the synchronous particle, it would see a negative voltage due to the phase of the voltage having advanced in this additional time. Therefore, this particle would lose some energy, leading to a higher revolution frequency, so its next arrival time would be earlier. If the arrival time happens to be too early, the particle would see a positive voltage and gain energy, shifting its arrival time later [32]. Fig. 2.5 illustrates this phenomenon. As a corollary, every particle that is not the synchronous particle would orbit around it, clumping together in what is known as a bunch. A period of this sinusoidal voltage which effectively forms such a potential well is called an RF bucket [31]. This is shown in Fig. 2.6.

What was just described as the relation between the arrival time or phase and the energy of a particle constitutes a longitudinal phase space, where traditionally the horizontal axis symbolises (relative) time of arrival (or phase) in seconds, and the vertical axis shows energy (deviation) in electronvolts [31]. An example of a plot in phase space is Fig. 2.5, showing the motion of one particle performing its oscillation around the synchronous particle. Such oscillations performed by particles can collectively be called synchrotron motion [33].

In the previous example, a collection of particles occupying the length of a single bucket were considered. However, provided that the RF generator is not turned off, all other particles that happen to circulate in the machine are affected by it as well. Therefore, the whole beam present forms into bunches. To make sure that synchronous particles always coincide with the zero-crossing of the oscillating voltage, the frequency of this oscillation has to be an integer multiple of the revolution frequency of the particle. This integer ratio is called the harmonic number of the accelerator or the number of buckets [31].

In a real accelerator, not every bucket has to be populated. In the LHC, for example, only every

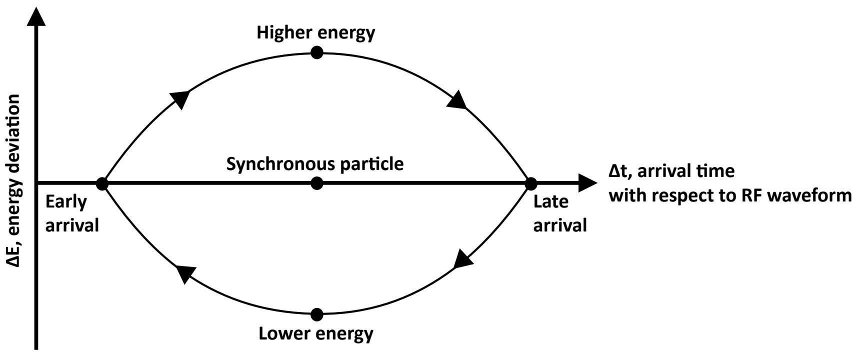


Figure 2.5: Synchrotron oscillation of a single particle around the synchronous particle. As can be seen, the difference in energy leads to a difference in arrival times with respect to the synchronous particle, and vice versa. Based on a schematic by Baird [31]

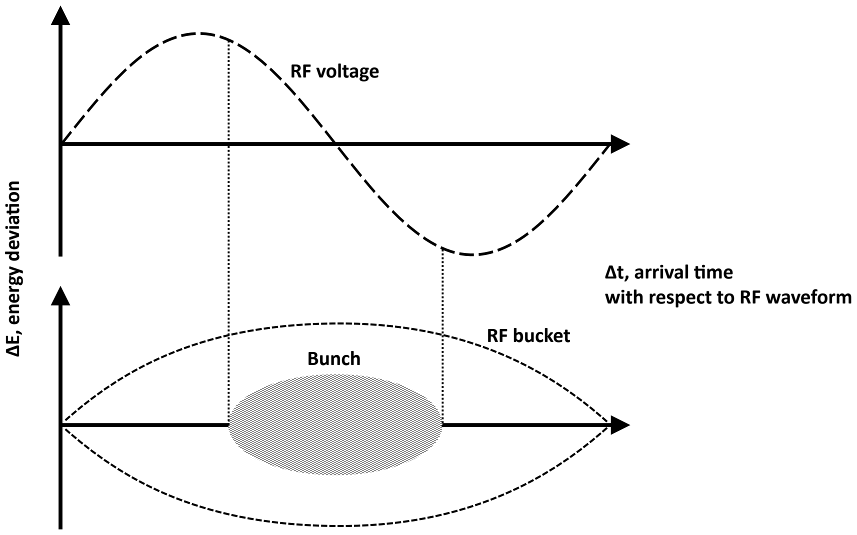


Figure 2.6: An RF bucket corresponding to a single period of the RF voltage, with a bunch of particles inside it. Based on a schematic by Baird [31]

10th bucket contains a bunch. This ratio of total buckets to populated buckets would be called the bunch harmonic number or the number of bunch slots. As a real-world example, the harmonic number of the LHC is 35640, with the bunch harmonic number being 10 times lower – 3564 [34]. The LHC's revolution period is about 89 μs. Knowing this, it can be deduced that the RF frequency must be around 400 MHz. It should also be mentioned that not all of these 3564 bunch slots are actually used: only 2808 are filled. This allows for small gaps between trains of bunches to ramp up the injection magnets. Lastly, there are no bunches in the last 3 μs of an accelerator turn, constituting what is called an abort gap [35]. This gap is used to give time for magnets to react to dump the beam in case of problems.

2.4 Importance of longitudinal bunch profiles

Considering again the energy *vs.* time relation in longitudinal phase space, it can be realised that its projection to the horizontal axis results in a bunch profile. Longitudinal phase space is therefore inherently related to the time domain representation of particle bunches going past an instrument along the beam pipe that produces a voltage related to the beam current [36]. Such a profile, for typical LHC bunch parameters, can be seen in Fig. 2.7. Due to the longitudinal phase space carrying information about the energy distribution of particles, as well as the distribution of synchrotron oscillation amplitudes [36, 33], there are multiple use cases for longitudinal bunch profiles in determining the operational parameters of an accelerator. In turn, this information can be used to optimise the machine performance to experience fewer instabilities and gain higher luminosity [32, 34].

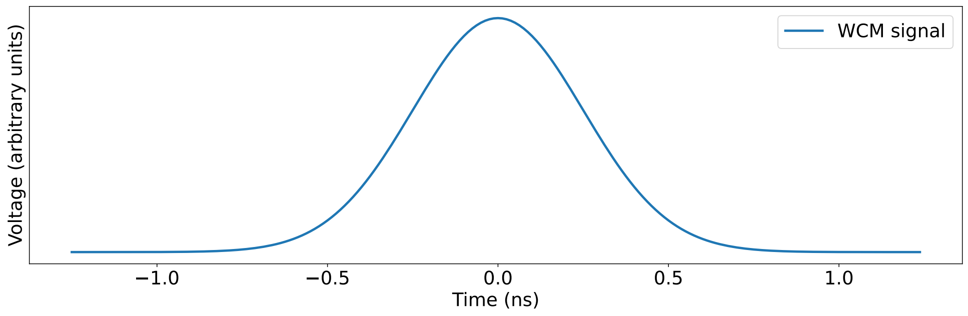


Figure 2.7: Time-domain plot of a single bunch profile, with the vertical axis in arbitrary units

There are two new use cases for longitudinal bunch profiles in the LHC. The first of two is the study of Schottky signals, for which longitudinal bunch profiles are needed to find the distribution of synchrotron motion amplitudes in bunches [36]. This, combined with other data from different instruments such as the Schottky monitor, leads to estimates of a parameter called chromaticity – describing how the transverse oscillations caused by magnets in an accelerator are related to particle momenta [37]. The main motivation is that the resulting measurement procedure is the only non-invasive way to measure chromaticity in the LHC [38].

The second use case is performing longitudinal phase-space reconstruction which can be used to study how well the bunches were injected into the machine, and bunch stability, among other things [33]. Considering the synchrotron oscillations described before, the bunch is effectively rotating around its synchronous particle. This enables tomography: reconstructing the 2D phase space from 1D bunch profiles effectively captured from different angles [39].

2.5 Longitudinal bunch profile measurements

It is clear that acquiring longitudinal bunch profiles is essential for multiple different use cases. However, there are also many ways of acquiring them. In this section, a brief description of possible instruments is given, as well as the general principles of longitudinal bunch profile measurements.

In the LHC, the bunched beam of charged particles, such as protons, moves in a metallic beam pipe. Every beam particle attracts opposite charges in the walls of the beam pipe, as illustrated in Fig. 2.8 [40]. When beam particles move inside the beam pipe, the opposite charges move along with them in the metallic walls, generating a wall image current, proportional to the beam current. It should be noted that there is effectively no electric or magnetic field outside the beam pipe [40]. Therefore, to measure the beam current by proxy of wall current, coupling inside the beam pipe has to be performed, or alternatively, the field could be extended outside the pipe. Although both capacitive and inductive coupling are possible, most often, capacitive coupling is preferred in order to avoid issues arising from stray magnetic fields [40].

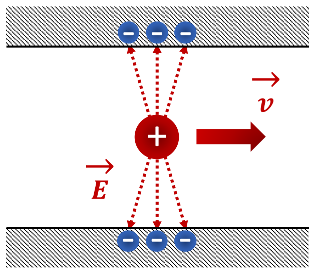


Figure 2.8: A charged particle attracts opposite charges in the beam pipe wall. As the particle is moving, an image wall current is generated. Figure based on Kube [40]

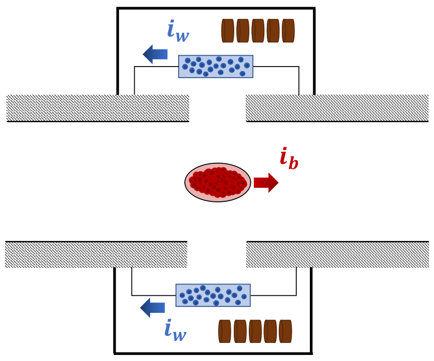


Figure 2.9: A symbolic representation of a WCM. A bunch of protons moves inside the metallic beam pipe, causing wall current. Due to a gap in the beam pipe wall, electrons have to move through a resistor instead. The voltage across the resistor can then be measured. Ferrites in the housing can also be seen. Figure based on Kube [40]

Considering the contrasting outlined possibilities, electrodes or pickups inside the beam pipe could be used, as well as transformers outside the vacuum chamber, when a dielectric insert is installed to the beam pipe, letting fields propagate outside. Such a transformer is called a fast beam current transformer (fBCT). Another option would be to put a loading resistor across the gap in the beam pipe and measure the voltage across the resistor. The working principle of such an instrument, called a wall current monitor (WCM), is depicted in Fig. 2.9. To make most current flow through the resistor, not through the housing, ferrites are often used to increase the impedance of the housing. The advantage of WCMs is that they are not as limited in bandwidth as fBCTs, reaching even into tens of gigahertz [41]. For digitising bunch profiles, this additional performance is needed to capture the shape of the profile as accurately as possible. WCMs are also preferred over pickups inside the beam pipe due to their frequency response being flatter, especially in the low frequency range [42], leading to more accurate bunch profiles.

2.6 Available resources and equipment

The work described in this thesis is performed in the Intensity and Tune Measurement (IQ) section in collaboration with the Software (SW) section, both under the Beam Instrumentation (BI) group at CERN. The previously described two use cases of longitudinal bunch profiles are in the interest of physicists in the IQ section with the SW section providing support in building required software to enable their acquisition.

Various resources and equipment are available for the project, starting with physical instruments located in the tunnel, and ending with software frameworks to be utilised in order to be consistent with the existing CERN control systems ecosystem. First, an overview of the available WCMs will be given, followed by the high-speed digitizers and their host PCs. Finally, available software solutions will be discussed. A schematic overview of the different hardware, as well as their connections and data flow can be seen in Fig. 2.10.

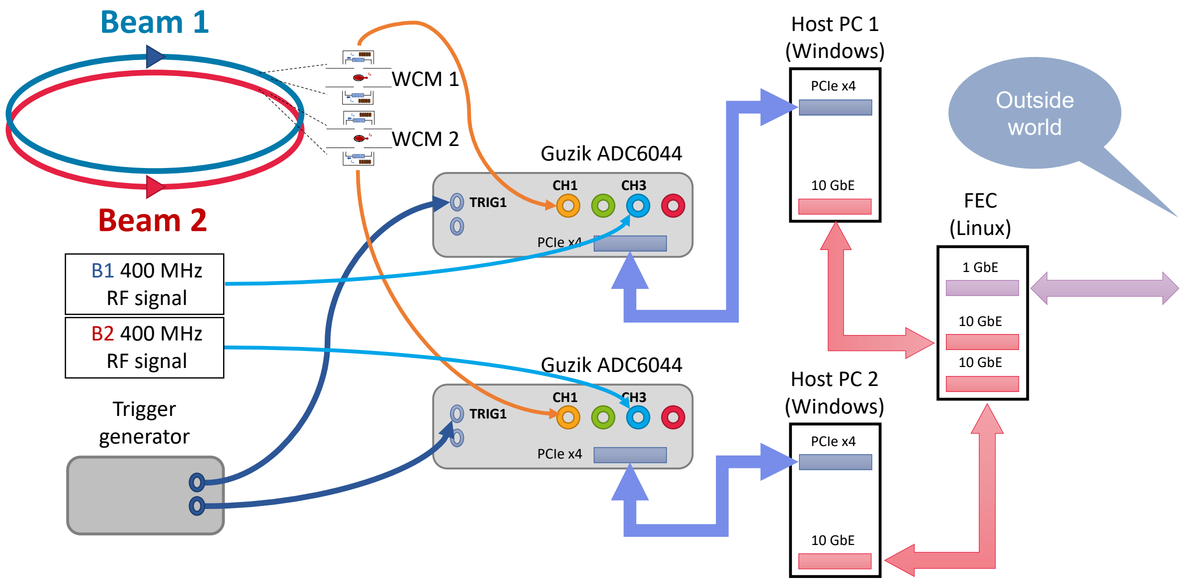


Figure 2.10: A schematic overview of the hardware used in the thesis. Curved lines represent analogue signals, while the thicker angular lines represent digital communication. Signals coming from the LHC's WCMs, as well as the RF reference signal, are digitised. Host computers interface with the digitizers and forward the data to a Front End Computer, from which it can be accessed by the users. For generating trigger signals for acquisitions, a custom trigger generator is available [40]

Installed to the LHC are two APWL series WCMs which were developed in-house specifically for the LHC, based on a pick-up designed for the SPS in the 1970s [43]. APWLs are longitudinal wide-band coaxial-type WCMs with a bandwidth of 2.2 GHz [43]. In Fig. 2.10, they are symbolised as WCM 1 and 2. Connected to the WCMs by approximately 25 m long 50 Ω coaxial cables are two Guzik ADC6044 high-speed digitizers which are located in a radiation-free service gallery next to the accelerator tunnel. These digitizers were initially procured in 2014. One of them was used for the SPS Head-Tail Monitor [44]. The Head-Tail Monitor digitised bunch profiles from long stripline pickups in order to monitor the head-tail oscillations resulting from beam instabilities [44]. Some software was also developed to integrate these digitizers into CERN's control systems. The existing software is discussed in detail in a later paragraph. At some point, the Guzik digitizer in the SPS was replaced by an oscilloscope with higher vertical resolution, and the original system based on the Guzik digitizer was disused. The other Guzik digitizer was meant for digitising a WCM in the PS but was never actually used.

Each of the two Guzik digitizers has four available input channels when digitising at 10 GS/s or two channels when the sampling rate is 20 GS/s. These particular units have a total internal memory of 64 GB, meaning 16 GB or 32 GB per input channel, depending on whether four or two channels are used. This differs from the ones with larger storage generally sold by Guzik [45]. Sampling two channels with the highest frequency, it can store at most 1.6 s of data. Although the input channels themselves can be used for triggering, there are an additional two external trigger inputs. In the 10 GS/s mode, the bandwidth of the digitizers is 4 GHz and with the 20 GS/s mode, it is increased to 6.5 GHz [45]. As can be seen, both are sufficiently wider than the 2.2 GHz bandwidth of the WCM to be digitised. By default, the vertical resolution of its ADCs (Analogue to Digital Converters) is 8 bits.

The two digitizers each have a host PC which is an HP Z420 Workstation with one Intel E5-1620 v2 CPU and 64 GB of RAM. To interface with host computers, the ADC6044 digitizers leverage four lanes of PCI Express (PCIe) Gen2 5 GT/s connection, both for control and data transfer [45]. The data transfer rate is limited to 1.6 GB/s, assuming an 8-bit vertical resolution. Physical connections are made via PCIe cables. These are all devices procured for the initial use case and taken as initial conditions for this project. The operating system for host PCs has to be Windows, as this is the only platform for which Guzik Enterprises provides digitizer drivers. In addition, some utility applications as well as an SDK are provided which enables the development of this project. The SDK limits the possible programming languages to C, C++, MATLAB, and Visual Basic.

In order to integrate the new application to be developed into the existing CERN control systems ecosystem, the Front End Software Architecture (FESA) real-time framework needs to be used. FESA enables the creation of abstractions of various devices and collections of devices, called device classes, so they could be controlled in a standardised manner [46]. After a FESA class has been designed, its code is generated. FESA uses the inversion of control design pattern, so the developer can insert custom code into the generated code if needed, for example, to interface with hardware. Then, all the code is compiled into an executable, called a deploy-unit, which can finally be run.

To store values, FESA uses device fields which are similar to private member variables of a class in an object-oriented paradigm. There are three different types of fields available in FESA: configuration, setting, and acquisition. Configuration is read-only and can only be set before launching the program. Settings can generally be thought of as settings of the underlying hardware. They are used to store the values input by the client. Finally, there are acquisition fields, which can only be written by real-time actions. Device fields are exposed publicly through properties, which are similar to object-oriented getters and setters, with the difference being that one property can have multiple members, called value-items, that can be linked with multiple device fields.

Real-time actions are another fundamental concept of FESA. Often containing custom code, these actions can be triggered by various events and scheduled in dedicated threads in order to meet possible real-time constraints. Setting priorities is supported. Furthermore, real-time actions can notify properties, so the client would know that for example new data has been written to an acquisition field. In a FESA class, all real-time actions are directly triggered by logical events. This implies that the actions themselves have no knowledge of how they are triggered, allowing for more flexibility, as any real-time action could be configured to be triggered by any real event. To generate logical events, event sources are used, such on-demand, timer, timing user, or custom event sources. Which event source triggers which event is specified by the instantiation unit in an event configuration. Tying this all together are scheduling units under concurrency layers. A concurrency layer is a separate thread, and scheduling units link actions and logical events.

FESA deploy-units typically run on Front-End Computers (FECs). Therefore, in addition to the host PCs, another PC is needed as the FEC. For this, a Siemens IPC847E with one Intel i5-8500 CPU and 64 GB RAM was used. An additional dual 10 Gb/s Ethernet network adapter was installed along with the standard 1 Gb/s card included with this model. The increased RAM compared to the standard model of this PC was needed to store the large amounts of data, and the high-speed Ethernet card was required to allow for fast communication between the two host PCs and the FEC. It is important to note that in the FEC, there is also a custom trigger generator expansion card which allows the users to generate trigger signals at arbitrary times [47]. This can be used to trigger acquisitions. Although technically embedded in the FEC, in Fig. 2.10, it is depicted as a separate abstraction for the sake of clarity.

3 Exploratory work

Considering the general goal, it was clear that there was some uncertainty as to what could feasibly be achieved with the given resources. As an example, would it be possible to acquire and store data continuously? Furthermore, what are the different ways of operating the digitizer, and could the existing software be used? Therefore, first, the existing code was explored and revived by the author, which in turn led to the creation of an easier-to-use minimal abstraction layer on top of the digitizer's low-level SDK. The new code was used to benchmark various aspects of the acquisition system's performance. Although a later chapter is dedicated to software development, some initial implementation details are already described here.

3.1 Overview of existing code

Over the years, multiple attempts at using these digitizers have been made. For the SPS Head-Tail Monitor, a prototype solution was developed in MATLAB by Thomas Levens to capture data for the head-tail bunch instability monitor and store it to HDF5 (Hierarchical Data Format version 5) files. As a completely custom one-off solution, there was no integration with FESA. In addition, although the performance of the code was not measured because it was only written as a prototype, MATLAB performance is generally orders of magnitude worse than C++ [48]. Finally, MATLAB requires a commercial licence of which there are only a limited number available at CERN. Due to these reasons, the possibility of using it as a base for further development was not considered in detail. Even though the program could end up being IO-bound, starting the development of the current project in MATLAB would still have carried a potential risk of redevelopment in case its performance turned out to be a bottleneck.

After the completion of the prototype in MATLAB, an operational version for the SPS Head-Tail Monitor was created by Gennady Sivatsk. The software, written in modern object-oriented C++, was logically organised into classes with the methods being relatively easy to understand. It also featured a set of unit tests, although no additional documentation. The solution was originally built in Visual Studio and used the MSVC compiler, with the exception of unit tests, which were meant to be compiled by GCC to be run under Linux. Its use case was to receive digitizer configurations from FESA and then perform a single acquisition for each received configuration. Even though in principle the software was well-organised, at the time the author started work on the project, it did not run or even compile due to missing dependencies.

The existing software was always meant to work together with FESA, meaning it utilised CERN Controls MiddleWare (CMW) libraries, such as CMW-RDA3, CMW-LOG, and CMW-UTIL. The first was used to communicate with FESA, as it is the standard communication protocol meant for interfacing with various devices related to FESA. CMW-LOG was used for logging and CMW-UTIL contained some utilities used by both. Although all of these libraries are available for Linux at the time of writing, no official Windows support is offered. The author found another team at CERN that was building Windows binaries of these libraries for their internal use, however, no guarantees of future support were made. As such, it was decided that these libraries should not be used in the proof-of-concept application.

To initially make the software compile, it was decided that CMW-RDA3 could be removed, as for experimentation, no communication with FESA was necessary. The author found the current versions of CMW-UTIL and CMW-LOG, cloned them, and made various changes to make them compile under Windows. This process allowed the author to be familiarised with both the existing code and how it uses its dependencies. Due to this, it was easier to later remove the CMW dependencies altogether. For example, the whole CMW-LOG library was later replaced by a very small subset of its methods that the author adapted and used in the application. The software also had other dependencies, namely the digitizer's SDK (called GSA SDK), and C++ support for HDF5. The SDK came with the installation of the digitizer drivers and related software, while for HDF5, pre-built binaries were found and used. Both libraries were dynamically linked to the project.

After the successful compilation of the project, a set of hard-coded configurations was tested and the digitizer was indeed able to be configured and made to acquire data. As it was now possible to experiment with different configurations and acquisition modes, such as continuous or segmented acquisition with different triggers, the next step was to analyse how long different steps, such as configuring or acquiring data can take.

3.2 Testing the performance of the digitizer

In this section, performance measurements are described along with the reasoning behind creating a new application from scratch to be used for this purpose. Finally, the limitations of the digitizer as well as considerations for further software development are given.

3.2.1 Abstraction of the digitizer driver

Trying to measure the performance of the digitizer, it was quickly found that the existing software had quite a large overhead because it was designed with some modularity and multiple layers of abstraction. As an example, to configure the digitizer, a configuration object had to be queued, then dequeued, and different other objects constructed to finally arrive at the configuration stage itself. To get a better idea of the raw performance, it was decided to create a new application from scratch that only abstracts away the lowest-level SDK functions and does not add much additional complexity. C++ was chosen as the optimal language for various reasons. First, a working application written in C++ now existed, meaning parts of it could theoretically be reused to save development time. However, even more crucially, the anticipated timing needs and future integration into CERN's existing ecosystem, as well as the competencies of CERN personnel, eliminated the use of both MATLAB and Visual Basic. Finally, C was not found to have any advantages over C++, whereas C++ offers built-in safer memory management and exception handling [49].

The SDK provided by Guzik has a very specific low-level interface: almost all of the functionality is provided via the "informational" and "lifecycle" functions. Informational functions allow the user to poll the digitizer for its various capabilities, such as what is the range of allowed gain values of its input variable-gain amplifier. A lifecycle function, on the other hand, is used to command the digitizer to perform various operations. Its usage is complex: the same function is called with different sets of arguments, depending on what operation is to be performed and how. One of these arguments in the set is the desired lifecycle or state to be reached after calling the function. For example, setting the lifecycle argument to “configure” will command the digitizer to perform a configuration operation. Other arguments, such as the configuration parameters for the digitizer, can also be set.

Since using these functions requires the user to be intimately familiar with the SDK, an object-oriented *Digitizer* class was created as an abstraction. The class has methods such as *configure*, and *acquire*, making it easier for humans to operate the digitizer. However, as already mentioned, the initial goal was to have the code be as lean as possible, in order not to potentially reduce performance, so all complex or time-consuming functionality, such as input validation, was skipped at first.

To make it easy to access the acquired data during development, it was decided that data should be saved to files. For each acquired channel, the results were packaged in a C++ *struct* (data structure) which stores both the pointer to the data array as well as some metadata about the acquisition. When double-buffering was later added to enable high availability of the digitizer, two *structs* were used per channel. File saving routines, or later additions to the software, could then access these *structs*, and through them, the data and metadata.

The existing code had already implemented saving data to files. For this, the Hierarchical Data Format version 5 (HDF5) had been used. HDF5, as the name implies, enables creating arbitrarily complex hierarchies in files, as well as packaging metadata with data [50]. There are other applications at CERN already using this file format, and even custom viewers have been developed. As such, there was strong motivation to use it also for this project. As an HDF5 writer was already implemented in the existing code, it was reused in the newly developed application with only minor modifications.

3.2.2 Initial theoretical considerations

To get an overview of the digitizer’s capabilities, it was first required to know what possible operation modes it supported. Based on the provided manual, the digitizer can either acquire continuously or in segments. In both cases, the user can set the duration of an acquisition, with the difference being that this acquisition can quickly be repeated in the case of a segmented acquisition. The number of repeats can be freely set, and a trigger provided to start each acquisition segment. An assumption was that consecutive segments could be acquired more frequently than performing separate acquisitions, which was later confirmed to be true. One special limitation of segmented acquisition is the post-trigger hold-off time of 300 ns, meaning there has to be a minimum gap of 300 ns between consecutive segments [51]. This has to be taken into account by the users, as acquiring one complete accelerator turn’s worth of data would mean that the consecutive turn could not be acquired. However, the acquisition period could also be set such that the hold-off time coincides with the abort gap, so no bunches would be missed.

Another consideration was the available trigger modes. Immediate, internal, and external trigger modes are supported. Immediate or manual triggering by a software start signal can be ruled out, except for testing the performance of the digitizer, as to acquire for example a specific bunch profile, more precise timing would be needed. The digitizer’s user guide also supports this view, noting that the software start signal is “not a real-time signal” [51]. With the highest

20 GS/s sampling frequency, internal triggering has the lowest jitter of 50 ps out of all triggers, due to utilising the input channel's sampling clock, compared to the external trigger's 4 ns, corresponding to the trigger's dedicated 250 MHz sampling clock [51, 45]. Thus, provided there are unused input channels, internal triggering would be the ideal option. As a limitation, only input channels being acquired can be used for triggering. This has some implications: even if the acquired trigger data is not used later, dedicating an analogue input channel just for the trigger is still required. Furthermore, this data, even if not needed, will still have to be transferred to the host computer, unnecessarily reducing the transfer rate of useful data. Therefore, if there is a limited number of inputs available, external triggering might have to be used, instead.

Based on the datasheet, the digitizer is connected to its host PC via a PCIe interface. The x4 PCIe limits the transmission speed to 1.6 GB/s [45]. Acquiring two channels at 20 GS/s with an 8-bit vertical resolution, 40 GB of data is produced every second that needs to be transferred to the host PC using Direct Memory Access (DMA) [51]. Clearly, the bottleneck of this pipeline is the communication channel, being 25 times slower than the produced data rate. This was later verified experimentally, with different periods of data being acquired. In addition, ways of directly mitigating this issue, for example by disallowing one of the channels to be sent, do not exist to the best of the author's knowledge.

3.2.3 Continuous acquisition

For performance testing, including CPU and memory profiling, tools included with Visual Studio 2022 were used [52]. Acquisitions with durations ranging from tens of microseconds to a hundred milliseconds were tested and measured durations were compared with theoretical durations which were calculated based on the expected number of samples and known sampling rate. Immediate or software triggering was used to start the acquisitions. As measured, actual acquisitions indeed took around 25 times longer than expected or even more, confirming the bottleneck. It should be noted that the digitizer almost never acquires exactly the number of samples as can be calculated from the acquisition duration parameter. Instead, based on another configuration parameter, it acquires either fewer or more, somewhat affecting the actually measured duration. Based on these results, it was understood that the amount of data acquired needs to be reduced by at least 25 times in order to facilitate quasi-continuous acquisition. An interesting anomaly was also found: the first acquisition always took a lot longer, in the order of seconds, regardless of the specified acquisition duration. This could be due to the digitizer initialising itself.

The measurements described in the previous paragraph were all done in a blocking manner: if an acquisition was requested, the function would block until all data had been acquired and transferred to the host PC. The SDK offers an option of non-blocking function calls, called dynamic acquisition. The idea is that the program could initiate an acquisition and then check whether some or all of the data has already been moved to the host PC's memory. While advantageous due to enabling other operations to be performed while the digitizer is busy acquiring data, effectively the same functionality can also be achieved using threads. Threads offer the advantage of not complicating the measurement process further by having to constantly check for new data, while still enabling the program to meanwhile, for example, post-process the already acquired data or send it forward. It should be noted that dynamic acquisition was tested and the overhead was not found to be any lower compared to regular blocking acquisition. This is expected, as the bottleneck was determined to be the PCIe interface.

To verify that any measured overhead was not due to poorly performing code, a profiler was used.

In the case of both regular and dynamic acquisitions, over 99.8% of the time was spent on SDK calls, with the rest spent mostly on logging statements. As the code was specifically written to be as lean as possible, this was not unexpected, but it also confirmed a fundamental limitation of the system which cannot be solved by better software implementation. Even eliminating all overhead caused by custom code, no noticeable speedup could be foreseen.

3.2.4 Memory limitations

In the case of constantly acquiring data, sending out or processing the already acquired data while the next acquisition is in progress would split the total time required in half compared to doing one after another, even when assuming no bottleneck between the host PC and the digitizer. However, as such, double the amount of memory is needed to reserve a buffer for both the ongoing and the previous acquisition. With the digitizer's effective available internal memory of 32 GB per channel, using two channels and a maximum acquisition duration, 64 GB of data would need to be transferred to the host PC. Since the memory needs to be allocated up front, the computer needs to have at least 128 GB of free random access memory available due to double buffering, in addition to the amount used by the operating system. As existing PCs with 64 GB RAM each are available for this project, the maximum acquisition time is reduced even further. Not considering the amount needed by the operating system and other processes, less than 0.8 s of continuously acquired double-buffered data could be fit into memory.

Despite the amount of memory needed, double-buffering could be required to meet timing requirements in cases where it is necessary to start a new acquisition almost immediately after the previous one has ended. In the code, double buffering was implemented by swapping the pointer to the data array passed to the digitizer SDK after each acquisition. For safe heap memory management, smart pointers were used. To maintain flexibility, a flag can be set to enable or disable double buffering, in the end letting the users make the trade-off between the amount of data acquired and the time it takes to send it forward, process it, or save it.

3.2.5 Data rate reduction

Considering the data transfer rate bottleneck, as well as memory limitations, a data reduction technique would be beneficial. Different ways of achieving this were considered. First, it would be possible to capture only a part of each accelerator turn, provided that the whole turn is not needed by the users. Second, every nth, for example, every 25th full turn could be acquired instead, making the effective data transfer rate equal to the data channel bandwidth, provided that the users do not care about the other 24 turns of lost data. Finally, the amount of data that is sent out from the host computer can be reduced even further by filtering out the gaps between consecutive bunches, due to there being nine empty RF buckets for every one bucket containing a bunch. Data reduction will be explored in detail in a later paragraph.

Since a custom trigger generator would likely be needed in any case to start the acquisition at the start of a turn or even at some specific bunch, it would be feasible to also use it to reduce the amount of data transferred to the host computer's memory by masking the output such that it would only pulse every 25th turn, as mentioned before. However, with each acquisition request, there is some overhead due to the SDK and possibly configuring the digitizer. A turn in the LHC lasts roughly 89 μs. Therefore, starting a new acquisition every 25th turn would mean an acquisition interval of 2.2 ms which might be infeasible, considering possible overheads. Indeed, as will be analysed next, the overhead is actually considerably longer.

3.2.6 Segmented acquisition

To avoid the overhead of starting a completely new acquisition for every trigger pulse, the digitizer provides a segmented acquisition mode. This means that instead of a single continuous acquisition, multiple segments would be acquired automatically, following trigger pulses. As an example, the digitizer could be set up to acquire 89 μs of data for 10 trigger pulses. Segmented acquisition therefore easily enables capturing every nth turn, provided there is access to a suitable trigger generator. Thus, its capabilities needed to be tested further to find out its overhead and whether it would be possible, in practice, to acquire data continuously.

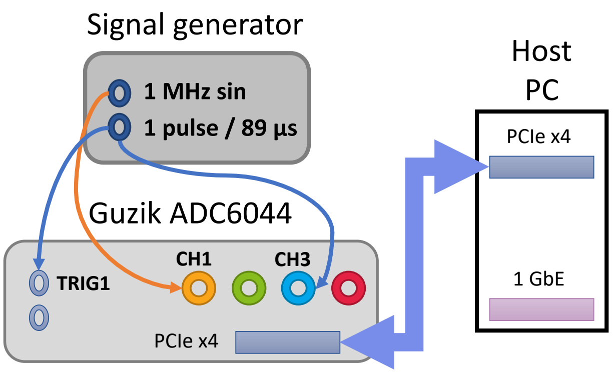


Figure 3.1: A simple measurement setup to test segmented acquisition

To enable testing segmented acquisition, as well as both of the triggering modes, a signal generator was connected to the digitizer, as shown in Fig. 3.1. A 1 MHz sine test signal was generated to one of the digitizer inputs. The digitizer itself was connected to its host PC. In addition, a short pulse with an amplitude of 1 V was generated every 25 * 89 μs to happen every 25th turn. With this trigger, there should no more be 25 times overhead as before, as the data channel bottleneck would effectively be mitigated. In Fig. 3.2, measurement results using internal triggering are shown (the lines "Internal triggering"). As can be seen, the relative difference between the time spent in acquisition mode vs. the expected acquisition time decreases with longer durations. However, the absolute time difference was never below 37.75 ms and remained around 40 ms on average. To ensure this was not due to the acquisition mode, dynamic acquisition was also tested and the results were not found to be any better.

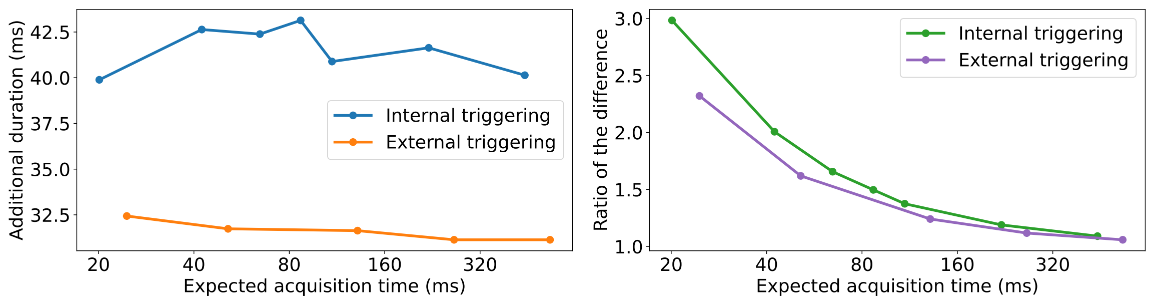


Figure 3.2: The measured acquisition duration was always longer than the expected duration. The additional measured duration was plotted in the left plot, while the ratio of the longer duration to the expected duration is shown on the right. As can be seen, since the additional duration in milliseconds remains relatively constant, the ratio of the two durations decreases

Curiously, when performing the same measurements using external triggering and therefore eliminating one input channel's worth of data, on average the absolute time difference turned out to be about 30 ms, as can be seen in Fig. 3.2 (the lines "External triggering"). Even so, the difference was always there, no matter how long the actual acquisition lasted. What could also be seen from the data was the bottleneck now being 12.5 times, rather than 25 times, which could be expected, as the data rate was halved. It should be noted that for this measurement, the trigger pulse was generated every 15th turn, although this should not have had any effect on the overhead. Taking into account the almost constant overhead which cannot be eliminated, it follows that for as continuous acquisition as possible, relative overhead can be minimised by choosing maximal acquisition periods.

3.2.7 Found limitations and considerations

To recap the results from analysing the digitizer's specifications and experimentation, some hard limitations were found. First, the theoretical bottleneck of the PCIe interface was confirmed. Thus, the data rate needs to be reduced by 25 times when acquiring two channels. Furthermore, both the digitizer's and host PC's available memory has to be considered, especially when using double buffering. Configuring the digitizer was also timed, however, trying to change different configuration parameters did not lead to any clear conclusions. A further limitation concerns triggering: the availability of input channels has to be weighed against trigger jitter. Maybe most importantly, there was found to be a constant overhead of about 40 ms on top of all acquisitions, effectively eliminating performing truly continuous acquisition. In addition to the limitations, it was found that a threaded program would be the simplest way to enable sending out data while performing acquisitions.

4 Analysis of user needs

The following is a preliminary study detailing the needs of the users. In the first and second sections, based on discussions with the users, their potential needs and expectations on the system under development are analysed. Finally, in the third section, a summary of the most important requirements based on the users' needs is presented.

4.1 Use cases

The users of the software to be developed are accelerator physicists belonging to the Intensity and Tune Measurement section at CERN. There is interest in analysing Schottky signals and performing longitudinal phase space reconstruction. In a previous overview of these two topics, it was explained why longitudinal bunch profiles are needed for both. With the main requirement of making these profiles available for the users, more detailed needs can now be outlined.

For the analysis of the Schottky signals, a profile of a pre-selected bunch is required to be measured every *n* seconds. The time interval between consecutive profiles is not exactly defined, however, the users have found a balance between sensitivity to changes *vs.* precision to be in the range of one profile every 50 seconds, assuming no noise. In reality, the acquisition system has some noise and therefore an average Schottky spectrum has to be found instead. For this reason, based on an initial estimate by the users, a bunch profile should be captured every 10 seconds, so they can later perform averaging. These acquisitions would ideally happen constantly and never stop.

On the other hand, for longitudinal phase space reconstruction, only the data acquired while an injection is happening is required, in order to capture the oscillations taking place after bunches have been injected into the LHC. An injection can happen as often as 40 seconds, stemming from its injector's – the SPS's – supercycle [53]. According to the users, these oscillations can last for about 30 seconds. Ideally, for this duration, the acquisition should be performed continuously. Furthermore, the system cannot miss any injections. Finally, when there are no injections taking place, no data is required.

Comparing the two use cases, it can be seen that they are partially in conflict. When there is an injection to the LHC, profiles for the Schottky signal use case can not be acquired. In the worst case, multiple injections might happen in a row and the Schottky use case would effectively be starved for data. In theory, some of the data acquired for phase space reconstruction could be also used for Schottky measurements, provided that the digitizer configuration is the same. However, based on the information gathered from the users, it is not critical for the Schottky use case if an acquisition is missed. Considering that multiple injections happening at such a schedule where many acquisitions would be missed in a row is not common, the users accept this trade-off. The opposite scenario of Schottky starving the phase space reconstruction use case should not happen, as will be discussed in the following section.

Although two use cases had been identified, the users did not further know what their needs were, as first, some actual data needed to be analysed to determine how to proceed. Therefore, in the first version of the software, a more general solution was desired, such that users could freely trigger the acquisitions at any given time, independent of any external timing such as injection events for the phase space acquisition or a pre-programmed timer event for the Schottky. Furthermore, having multiple configuration parameters that could be changed on the fly was found to be necessary for exploring the functionality of the digitizer and the software from a physicist's perspective.

4.2 Detailed analysis of user needs

Once the use cases had been analysed, three common steps were identified: configuring the digitizers, triggering the acquisition, and accessing the data. To integrate the software into the existing CERN controls ecosystem, the users have to be able to perform all of these steps via FESA. In light of known hardware limitations, some data reduction and post-processing possibilities were also considered. Furthermore, there was a general need to digitise the 400 MHz RF reference signal which forms the RF buckets. This is needed by the users in order to match the bunch profiles exactly to the centres of their respective buckets, due to the fact that the free-running sampling clock is not related to the RF frequency. Without this information, further analysis of the acquired bunch profiles would be difficult. The three common steps are now described in more detail.

4.2.1 Configuration

The digitizers enable various configuration parameters to be set, such as acquisition duration, number of acquisitions, trigger specification, and many others. Similarly, some parameters to control the data reduction would be required. It was determined that most low-level digitizer controls would be unnecessary for the users to know and to change on a regular basis. Therefore, an analysis was performed to find out which configuration parameters would the users want to frequently change, and thus be able to be easily set from FESA. Furthermore, sending only a subset of parameters would mean less data to transfer between the applications.

A table, shown in Appendix A, was compiled by the author to offer the users a choice of possible configuration parameters to be set from FESA, as well as their usefulness according to the author. The required number of bytes for each parameter was also added to emphasise the trade-off between the number and size of parameters *vs.* the time it takes to send them. Then, based on a discussion and feedback from the users, a final selection of parameters was made, as can again be seen in Appendix A. These were, for example, the number of triggers, and gains and offsets of the digitizer's input preamplifiers.

4.2.2 Acquisition

Due to finite digitizer memory and limited data transfer rate, as detailed in the previous chapter, it was known that an acquisition could not run continuously forever but rather needed to be triggered by external events. In the case of Schottky measurements, an acquisition has to be performed about every 10 seconds. Triggering precision is critical to ensure the correct bunch profile can be acquired. In the case of the longitudinal phase space reconstruction, the injection warning message in the LHC is broadcast only 1 s in advance and therefore, the system has to be ready to acquire data in that time. Again, the users want to be able to start the acquisition from a specific bunch in the accelerator, demanding high trigger precision. Therefore, manual triggering can be ruled out and it should instead be done either based on the signals that are being digitised or some externally generated pulses. This is the case even in the proof-of-concept solution, as the users would rather configure an external trigger generator to give precisely timed pulses corresponding to whichever start time they prefer.

As was described previously, phase space reconstruction should be able to preempt any other use of the digitizer, such as Schottky measurements, in order not to miss any injections. However, once an acquisition has been started, it cannot be stopped. Therefore, when acquiring profiles for Schottky measurements, the total duration of the procedure should not last more than one second minus the time it takes to perform another configuration and start an acquisition, due to the injection warning being issued one second in advance. Thus, in general, run-time performance should be a priority in developing the software. As a final common acquisition-related need for both use cases, the 20 GS/s sampling rate is preferred by the users to achieve higher time resolution.

4.2.3 Data reduction and processing

Although the data rate between the digitizers and their host PCs can easily be reduced by not acquiring every turn, to further reduce the amount of data transferred between the host PCs and the FEC, filtering out unneeded data in software is required.

Bunch filtering

First, bunches that the users do not need at a given moment can be filtered out. For example, in the Schottky use case, only one bunch profile may be required at a time. However, to meet the requirements, the users still have to be able to request the profiles of any number of arbitrary bunches in the accelerator. As a further need, to precisely filter the bunches, the RF signal needs to be acquired and the centres of RF buckets found. This is due to having no other precise way of knowing where a bunch centre is other than to take the centre of the corresponding bucket as a reference. Theoretically, even if all bunches are wanted by the users, removing all the other samples would greatly reduce the amount of data. Since there is one bunch for every 10 RF buckets, removing all of the empty buckets would result in only 1/10 of the data remaining. Implementing these techniques is thus necessary.

Phase offsets

As described in the section about the LHC, RF voltage generators are used to accelerate particles, forming bunches in the process. In the case of the LHC, the RF frequency is around 400.8 MHz, although varying slightly due to the acceleration of the particles. The free-running sampling clock frequency of the digitizer is 20 GHz. If the clocks were exactly integer multipliers of each other, then for every RF period, an integer number of samples would be acquired at exactly the same phases of the RF signal. However, these two clocks are not in any way synchronised with each other and therefore, they are, in fact, not integer multiples of each other. This means that from bucket to bucket, or from accelerator turn to accelerator turn, the samples are acquired at slightly different times. The outcome of this, as well as a possible correction, are shown in Fig.

4.1. It should be noted that this is simply due to digitising the analogue signal. With an infinitely high sampling rate, this problem would not exist. However, digitizers with a sufficiently high sampling rate to effectively mitigate this issue are not available.

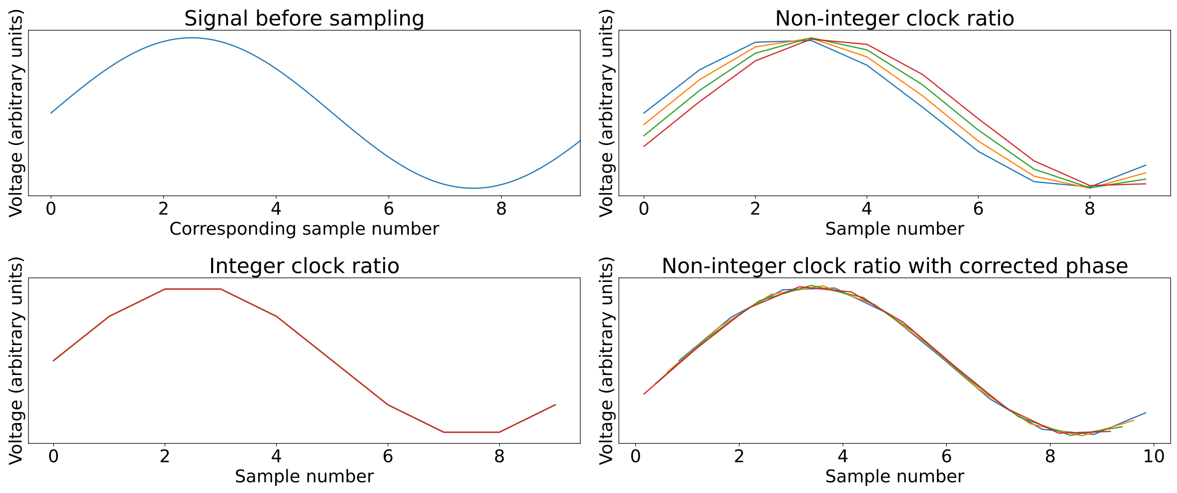


Figure 4.1: Demonstration of the effects of an asynchronous sampling clock. The ideal signal before sampling, as well as the signal samples with a synchronous clock, can be seen in the left column. Four periods of a sinusoidal signal acquired with a 10 times higher sampling frequency are overlapping perfectly on top of each other. This is in contrast to the top-right plot, where a sampling frequency of 10.25 was used. As can be seen, when assuming exactly 10 samples per signal period, the phase changes with every signal period. However, as seen from the bottom-right plot, this can be corrected

The implication of sampling the RF signal at slightly different times was already explained. However, considering that both the wall current signal and the RF signal are sampled synchronously to each other, slightly different longitudinal bunch profiles would also be acquired. This is problematic for the users, as the profiles from turn-to-turn would be shifted in relation to each other. To correct the shift, that is to align the bunch profiles, the sub-sample phase offset of the RF signal with respect to the sampling clock needs to be found.

4.2.4 Data access

The users only strictly required that the acquired data be saved to files, with online access being a nice-to-have feature in the proof-of-concept software. However, as it will become a requirement in the future, it was still considered a requirement by the author. An important consideration was whether receiving data could be done in parallel to the acquisition, as this would considerably increase the availability of the digitizer to be used for acquiring data. As was found out in the previous chapter, there are no hardware limitations preventing this, although the acquisition time would be more limited due to the amount of memory required for double-buffering. In order not to miss any acquisition in the phase space reconstruction use case, it was decided that acquisition should be asynchronous to data transfer.

In addition to receiving the raw data, users also wanted to capture some metadata about the acquisition. Similarly to configuration parameters, a table with a selection of metadata was compiled, as shown in Appendix B. Examples of the metadata include the amplitude resolution and offset of the digitizer, as well as an acquisition timestamp. This metadata was determined to be important to first convert the raw ADC readings to voltages but also to have some identifying values available, such as the device name and timestamp, and to receive the phase offsets between bunches and their buckets.

As mentioned, saving data to files was deemed to be a necessity. In the previous chapter, the implementation of saving files both in the HDF5 as well as a custom format was described. Based on previous experience with various file formats, the users preferred HDF5, as it packages metadata with the data. There is also existing software both publically available and under development at CERN to view HDF5 files.

4.3 Requirements

Considering the detailed needs outlined in the previous sections, formal requirements were generated. Appendix C shows the full set of requirements for the proof-of-concept solution. The following is a short summary of the requirements.

The users have to be able to use FESA to interface with the digitizers. The users need to be able to configure the digitizer with some often-changing configuration parameters. They need to be able to command the digitizer to start the acquisition and they need to receive the acquisition data, as well as metadata. Which configuration parameters can be sent and what metadata can be received was specified in cooperation with the users. Consecutive configuration and acquisition need to be fast enough to meet the one-second deadline specified for the phase space reconstruction use case. The acquisition data has to be saved to HDF5 files if so requested, but preferably also made available online via FESA. The users have to be able to determine whether the digitizer has completed an operation, as well as be reported any errors that have arisen. Finally, while not strictly a user need, the use of third-party libraries is generally discouraged at CERN due to the potentially long life spans of the applications in contrast with the limited availability of support from third parties.

The scope of this thesis covers only the development of proof-of-concept software. Therefore, the automation of the acquisition process based on various timing events is not in scope. In addition, the algorithms to perform post-processing of the acquired data are not yet implemented by physicists, so in the current stage, automating everything would lead to needlessly accumulating data into files. Nevertheless, the timing requirements have been considered, so that for future development, it is known that the software is able to meet these requirements.

5 Software development

One of the main goals of the thesis was to develop a proof-of-concept software solution. In this chapter, a general overview of the software architecture is given, as well as implementation details. All source code of the developed software is available on request from CERN (SY.BI.secretariat@cern.ch). At the time of writing, it is not open-source due to licencing considerations.

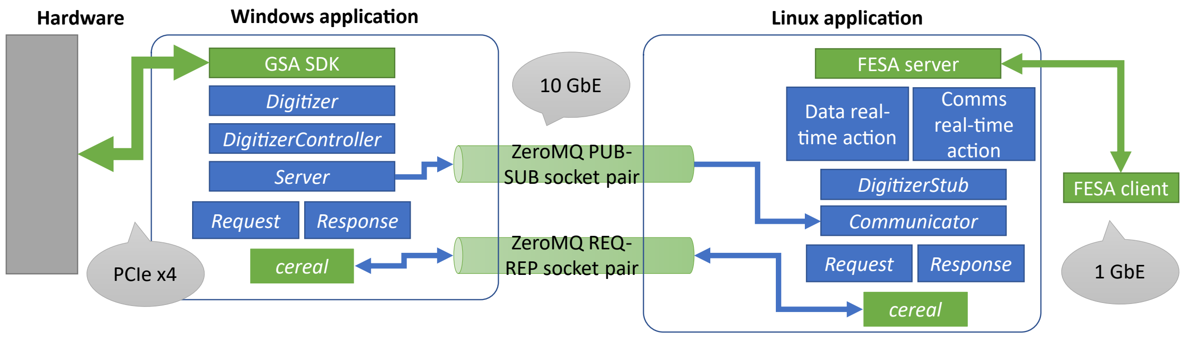


Figure 5.1: The architecture of the proof-of-concept software. Components in green symbolise already existing technologies, while everything in blue is the product of this thesis. The two applications are outlined, while the components *Request*, *Response*, *cereal*, *ZeroMQ*, *Communicator*, and *DigitizerStub* are also part of the communication protocol

The software is composed of two separate applications, one for the Windows host PCs and one for the Linux FEC. The Linux application, more specifically the FESA server, enables the users to set digitizer configurations, request acquisitions, and receive data along with metadata. Via a communication protocol, the FESA server interfaces with the Windows application, which in turn controls the physical hardware. An overview of the whole system is shown in Fig. 5.1.

5.1 Communication protocol

Regardless of the exact architecture, the two applications running on different operating systems need to communicate. From the start, there was no obvious communication protocol to be chosen. Therefore, in this section, CERN internal, third-party, and custom options are discussed to determine the most suitable protocol.

Based on the software requirements outlined earlier, requirements for the communication protocol could be generated. From the perspective of the Linux application, it needs to enable sending and retrieving digitizer settings, issuing commands, and receiving the acquisition data along with metadata. The protocol should also not be the bottleneck compared to a 10 Gb/s Ethernet connection. In the context of the protocol, the Windows application is the server and the Linux application is the client.

From the client's perspective, the requirements for the communication protocol were established. The protocol shall

- enable sending commands,
- enable sending and receiving digitizer configurations,
- enable sending acquisition data,
- enable sending metadata,
- enable polling the completion of digitizer operations,
- propagate errors to the client,
- work on the operating systems and platforms of the existing hardware,
- enable data transfer at least as fast as 10 Gb/s,
- include versioning information.

5.1.1 Existing communication protocols

Three existing options were first considered: RDA3, IPbus, and gRPC. Remote Device Access 3 or RDA3 is a product of CERN's Controls MiddleWare team [54]. It enables interfacing with various equipment using a device-property model. Devices, which are abstractions of hardware, have properties which can be set, get, and subscribed to. An API for both Java and C++ is provided. As was discussed earlier, there is no support for Windows. Certainly, RDA3 for Windows could ideally be revived, however, there will be no resources dedicated to this for the foreseeable future. Thus, using it was not considered further.

The second option, IPbus, is an Ethernet-based communication protocol used at CERN to interface with Field Programmable Gate Array (FPGA) devices [55]. From the perspective of the client, memory-mapped resources can be read or written. Currently, IPbus offers a server implementation only in the form of firmware for FPGAs and a client-side library for either C++ or Python. The fact that there is no server implementation in C++ is the main reason to exclude IPbus from consideration. However, its throughput of 0.5 Gb/s (0.0625 GB/s) could also be a limiting factor compared to 10 Gb/s (1.25 GB/s) Ethernet [55].

Developed by Google, gRPC is a high-performance remote procedure call (RPC) framework widely used by microservices [56]. It provides both a transport layer and a serialisation protocol – Google's Protocol Buffers [57]. The main advantages of gRPC are its high performance and large community. The remote procedure call model also suits the current use case. However, the open-source version of the framework does not have any guarantee of support. Furthermore, even though the API is relatively easy to understand and use, it is still a large and complex framework. Combined with being a third-party framework, fixing potential issues and maintaining it might be difficult, especially considering that there are no active projects at CERN using it. In fact, some applications at CERN have been migrated away from gRPC for exactly this reason.

Nevertheless, as a proof-of-concept, gRPC was tested. A minimal server layer was written on top of the *DigitizerController* class, with gRPC being effectively a wrapper. The Protocol Buffer language was used to write the specifications for all the required structures as well as remote functions. Using an included compiler, code to be included in the existing C++ application was then generated [57]. To control the application, a minimal client was written in Python. Python was chosen over C++ because of its simplicity, which enabled rapid prototyping.

The framework was generally found to work well, however, some potential issues were noticed.

The primary issue was how byte arrays were handled by Protocol Buffers. In the C++ port, they are represented as standard strings. This is a problem because the data is kept in *uint8_t* buffers and a string cannot be constructed from such a buffer without any copy operations. In the case of possibly tens of gigabytes of acquisition data, copying would require additional memory and delay the data transfer. The performance of the prototype was tested and its data transfer rate was found to be 3.65 Mb/s (0.457 MB/s). The very low speed could be due to having a badly optimised message size or due to the client being written in Python. Even with the possibility of improving the performance, possibly to match the Ethernet bottleneck, the fact remained that gRPC is a third-party framework and based on input from the users, it was decided to abandon it.

5.1.2 Custom communication protocol

Having not found a suitable protocol to be used, and based on the recommendations of the users, the direction to develop a simple lightweight custom protocol was taken. The protocol integrates a transport layer, serialisation, and custom data structures, as well as additional abstraction layers to make it easier to use. The protocol fulfils all the requirements set earlier. The architecture in Fig. 5.1 includes the components of the communication protocol and in Fig. 5.2, the structure of the messages can be seen.

In designing the protocol, inspiration was drawn from gRPC, thus effectively providing the user with a remote digitizer API. The communication protocol is fundamentally an exchange of requests and responses, both to send commands and exchange data. A data structure is associated with each request and response, mimicking respectively the input parameters and returned values of a remote procedure call. These structures can contain any number of C++ built-in or standard template library types, or other *structs*, thereby maintaining flexibility in case of any later changes in user needs. It was decided that a separate data channel would be allocated exclusively for the acquisition data transfer in raw bytes, in order to have as low overhead as possible.

As the transport layer, raw network sockets were first considered due to their performance. However, from the implementation of RDA3 [54], a messaging library called ZeroMQ (abbreviated as ZMQ) [58] was known to the author and the users. Even more crucially, ZeroMQ is integrated into FESA, requiring no additional dependencies in the Linux application. ZMQ offers various advantages over raw sockets, such as platform independency, background I/O, messaging patterns, and multiple endpoints [58]. An advantage of ZMQ that made it suitable to be used in the communication protocol was its multiple messaging patterns [58]. The request-reply pattern (REQ-REP) suited ideally the back-and-forth communication controlling the digitizer. On the other hand, its publisher-subscriber (PUB-SUB) pattern lent itself to sending the data.

For data to be sent over ZMQ sockets, it needs to be in a binary format. ZeroMQ's creators recommend for example Protocol Buffers to serialise the data. However, like gRPC, this technology is third-party and also relatively large and complex. Custom serialisation was considered but it was ultimately found to be too impractical to implement. After some further research, a header-only library called *cereal* was found to handle serialisation [59]. *cereal* supports both native and portable binary serialisation, as well as serialisation to JSON and XML. For the current use case, portable binary serialisation was chosen as a trade-off between stability and performance. *cereal* is small and very portable, requiring no dynamic linkage, and working with multiple C++ versions and compilers [59]. It is thus suitable for use in both the Windows application and the FESA class. Another essential feature is that it supports most C++ standard template library types out of the box [59], which enables the definition of almost arbitrarily complex request and response structures for the custom protocol. As a note, even though the requests and replies are serialised, the acquisition data is not, as it is already in binary format, and can be sent by ZeroMQ.

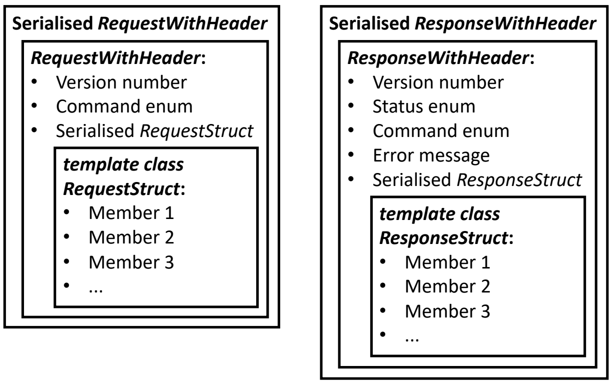


Figure 5.2: Generic request and response structures wrapped in parent structures with additional information. To enable different data structures to be sent, C++ templates are used

In Fig. 5.2, generic request and reply structures can be seen. Starting with the requests, any given request structure, containing arbitrary fields, even other structures, is serialised. For example, this could be a structure containing digitizer configuration parameters. It is then embedded into another more generic request structure, along with a supplied command enumeration for the receiving side to know how to deserialise the request. A version number of the protocol is added to ensure that identical messages are used by the two applications. Finally, the generic request structure itself is serialised and can be sent using the request-reply ZMQ socket.

After a request has been received and processed, a response should be sent back as an acknowledgement to the client that the server has received the message. Similarly to requests, additional data can be sent in a response, such as metadata about an acquisition or the completion status of an operation. While otherwise very similar to requests, responses also package an error code and an error message in order to relay all the errors that have occurred on the server to the client.

Functions in namespaces *Request* and *Response* were implemented to make it easy to create and parse requests and responses. In order to fully parse a request or response, first, the generic request string is decoded to find the issued command. It is then possible to select the correct instantiation of a deserialisation method to parse the underlying structure. It should be noted that the received version number is also checked and an exception is thrown when it does not match the local version. In addition, in the case of responses, any received exceptions are also thrown.

Further abstractions were added to create a remote procedure call style interface for the client. First, functions from *Request* and *Response* were packaged in a *Communicator* class to integrate them with ZeroMQ. A stub class (*DigitizerStub*) was created to mimic the functionalities offered by the server. This way, the calling code does not have to know the implementation details but can act as if it simply calls local methods such as *setConfiguration* and *startAcquisition*. Some additional functionality is also built into this layer, such as an option to wait until an acquisition has been completed, utilising the polling of the completion status operation underneath, as well as methods to convert the ADC codes to voltages, based on received metadata.

The protocol's performance was tested. Due to the limitations of the testing setup, the server and client were both run on the same Windows machine. Nevertheless, they were running as different processes connected via sockets, so only the network interface card was actually left out of the loop. Otherwise, the software layers were the same. Different data chunk sizes were tested and it was found that as a rule of thumb, higher chunks seemed to yield better performance. Establishing a benchmark, with a 1 MB chunk size, a data transfer rate of about 1.6 GB/s was achieved. Considering that this is higher than the bandwidth of 10 Gb/s (1.25 GB/s) Ethernet, the requirement regarding the transfer rate was met.

5.2 Windows application

As part of exploratory work, an abstraction of the digitizer hardware had already been created. While this class had all the necessary methods for performing the required operations, some of its methods were blocking. For instance, the *acquire* method waits for the acquisition to be completed. Therefore, an asynchronous *DigitizerController* layer was created to allow multiple operations to be performed simultaneously. Finally, a server layer was also implemented to integrate the communication protocol. A simplified class diagram of the Windows application can be seen in Fig. 5.3. As a note, since the developed *Digitizer* class turned out to be much less complex than the original software, it was decided to continue developing the rest of the software on top of it, using only small segments of the original software.

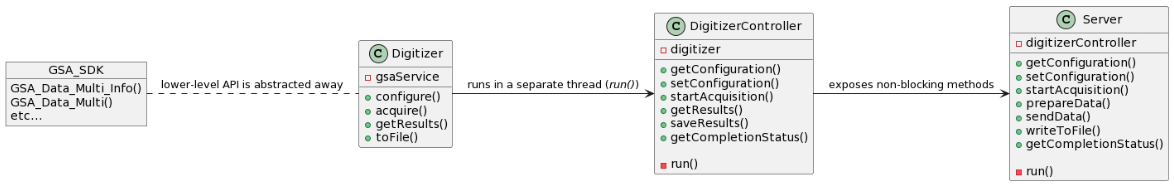


Figure 5.3: Simplified class diagram of the Windows application. The three main classes – *Digitizer*, *DigitizerController*, and *Server* – can be seen, as well as the low-level Guzik SDK

5.2.1 Asynchronous controller layer

In principle, the *DigitizerController* exposes a uniformly asynchronous API for all of its methods, while managing concurrency in the background. For methods which have blocking calls, such as acquisition, asynchronous public methods are used to signal that the operation has been requested. Additional data, such as configuration values, can be passed as arguments to these methods. In a separate thread, a loop is constantly running and performing the requested operations. For synchronisation of data and signalling flags, mutexes and atomic booleans are used. On the other hand, for non-blocking methods, such as retrieving the current configuration *struct*, the public method simply performs the operation, in this case returning the structure.

An exception to the previously mentioned system of a threaded loop and various non-threaded methods is the *saveResults* method, which commands the *Digitizer* instance to write the last acquisition data to a file. Even though the digitizer hardware is not utilised for this, saving data to a file can be a long operation when dealing with gigabytes of data. Thus, it was decided to have this method spawn a separate thread to not interfere with any other operations the user might want to perform, for example, the next acquisition. Additionally, the spawned thread can have shared ownership of the data array, guaranteeing that it will not be deleted before the entire file has been successfully saved.

A crucial component to making the asynchronous API work is having a way to poll the completion of operations. Otherwise, due to asynchrony, the user would not have any indication of the completion and outcome of a requested operation. Therefore, completion status flags are set and cleared by the public methods and the main loop. The user can then use a separate method to poll the state of any blocking operation.

5.2.2 Server layer and exceptions

To integrate the communication protocol into the Windows application, a server layer was created. Effectively a wrapper of *DigitizerController*, the server receives and parses requests from the client in a separate thread, decides which asynchronous *DigitizerController* methods to call, and send back a response, as well as acquisition data if needed. It also performs some additional processing which falls out of the scope of the layers below it, such as how to divide the data into chunks to be sent more efficiently, or how to filter data.

With all the Windows software layers in place, an overview of error handling can now also be given. Modern C++ exceptions were favoured, as they allow the propagation of exceptions through multiple function calls without the middle layers having to know how to handle them. This suits the current use case, where issues can arise in any of the layers, but should, in the end, be propagated to the client.

Since the digitizer SDK only provides C-style error codes, the *Digitizer* class was programmed to parse them and throw C++ exceptions when needed. All the layers above then transport exceptions towards their parent threads until they are finally processed and sent to the client in the form of an error code with an optional error message. As mentioned, all the exceptions will be rethrown on the client's side.

As an example of the usefulness of exception handling, the *Digitizer* class implements configuration parameter validation and throws exceptions in case of incorrect parameters. However, neither the controller nor the server layer know what to do with them, so they are forwarded to the client, instead. The users can then decide how to proceed.

5.2.3 Bunch filtering

To reduce the amount of data sent from the server to the client, the users are given the option of selecting which bunches' profiles they want to receive. The users would have to provide the trigger pulse and a list of bunch indices to be acquired. Furthermore, due to fixed cabling delays, the users can specify the offset of the bunch centres from their bucket centres to be corrected by the software. Finally, the users can specify how much data of each bunch they would like to receive.

Based on the bunch indices sent by the users, the program first calculates the acquisition segment duration. This is used as a configuration parameter for the acquisition, as is the number of segments or turns worth of data to be acquired which is also sent by the users.

The algorithm uses the fact that, as shown in Fig. 5.4, the bunches should be centred in their enclosing RF buckets and spaced by 10 RF periods. Therefore, if the locations of RF periods defining the buckets are known in the data, the expected locations of bunches are also known. For each 10th bucket centre in the RF signal data, thus corresponding to each bunch, it is checked whether its index is in the list sent from the client. If yes, the specified length of WCM data is

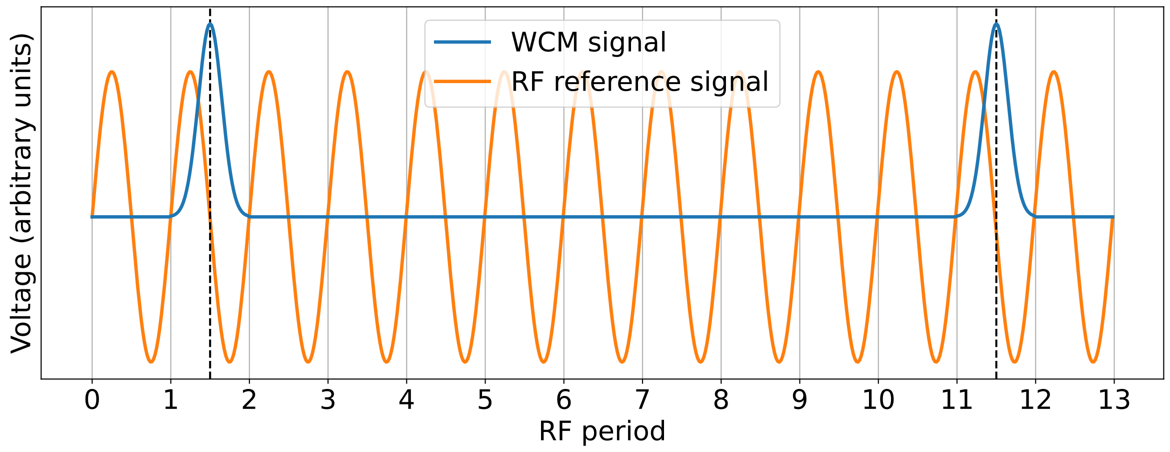


Figure 5.4: For every 10 RF periods defining the buckets, there is one bunch. As can be seen and indicated by the dashed vertical lines, the bunches are centred in their respective buckets

copied to an output buffer, adjusted for the specified offset. If not, the next 9 RF periods are skipped to arrive at the next bunch. This process is repeated until the last wanted bunch has been found or the data has been exhausted. Finally, the output buffer with the concatenated bunch profiles is ready to be sent to the client.

To locate the buckets themselves, the program starts going through the sinusoidal RF reference signal data and detecting zero-crossings, which are the bucket centres. To detect a zero-crossing with sample-level precision, a simple algorithm can be used that iterates over all samples, and when the current sample is negative and the previous sample positive, a zero-crossing is deemed to have been found. A helper class was created for this. Moreover, the digitizer's SDK can be used to find what is the current zero level in ADC codes. To maintain loose coupling, this value is saved as metadata by the *Digitizer* class when an acquisition is performed and can be accessed via the *DigitizerController* API in a standard way.

Considering the relatively large 4 ns jitter of the external trigger input channel in comparison to the 2.5 ns bucket length, the generated trigger pulse could move around enough to end up in the wrong bucket. Since the bunches are filtered based on the underlying buckets, this would cause incorrect data to be kept and correct data to be disposed of. To avoid this, a helper function was created to find the first actual bunch in the wall current data. Simple in principle, a threshold, specified by the users, is set. Then, the first upward and downward crossings are recorded and the middle sample between the two is taken to be the first bunch centre. It is from this index that the actual algorithm begins filtering.

5.2.4 Phase offset calculation

When filtering the bunch data, phase offsets due to the asynchronous sampling clock are also calculated, as requested by the users. The simplest possible solution is to linearly interpolate between the last positive and the first negative sample to find the true zero-crossing. While being efficient, only using two data points makes this method sensitive to noise and the resulting offset could therefore be relatively imprecise. Traditionally, especially in analogue electronics, low-pass filtering can be used to reduce noise, leading to a more precise zero-crossing detection [60]. Digitally, filters such as Savitzky-Golay are often used for smoothing, in this case by local polynomial fitting [61]. It is robust to noise while having zero phase shift. Also, it maintains the absolute signal amplitudes and widths of the peaks [61]. Although effectively eliminating noise, such filters can be computationally complex and therefore might cause issues in meeting real-time deadlines, such as the one-second injection warning.

Another possibility arising from the acquired signal itself being an elementary function would be to fit a sine. The phase of the sine could then be the phase offset returned to the user. This solution was implemented and found to work. However, even with an iterative algorithm, the number of elementary operations was quite high, even with a relatively small number of samples used. No closed-form solution to the problem exists either to the best of the author's knowledge, especially if the frequency of the sine is drifting [62]. Therefore, additional solutions needed to be considered. Finally, it was decided that a simple linear regression model would work well to estimate the true zero-crossing, as well as be computationally less complex than other methods.

$$y = b_0 + b_1 x \tag{5.1}$$

Equation 5.1 shows the formula of a line. Coefficient b_0 signifies the y-intercept, while b_1 is the slope. The goal of linear regression is to find these two coefficients, such that the residuals, or how far the samples are from the best-fit line, are as small as possible. The solution to linear regression is given in Equation 5.2 [63]. To find the x-intercept or the phase offset, the additive inverse of the estimated intercept has to be divided by the estimated slope. Even though matrix-form solutions exist that are traditionally suitable for computers to solve, in this use case, iteratively finding the two averages, sample covariance, and sample variance would prove to be more efficient, based on a quick evaluation of the amount of required elementary operations.

$$\begin{aligned} b_0 &= \bar{y} - b_1\,\bar{x}, \\ b_1 &= \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2} \end{aligned} \tag{5.2}$$

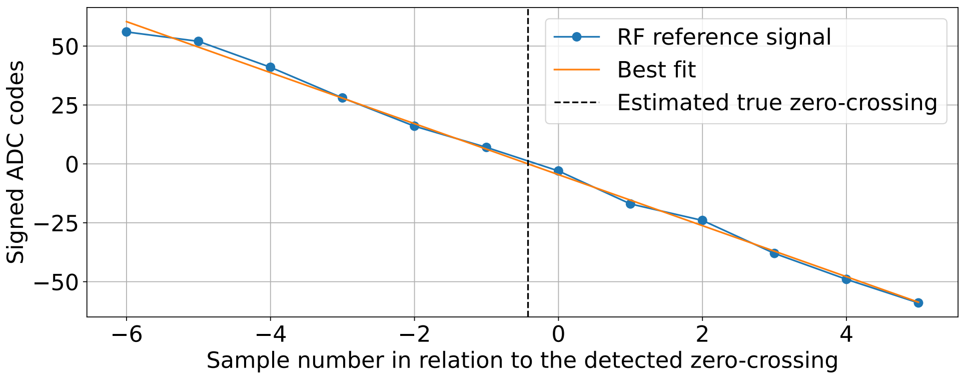


Figure 5.5: The phase offset calculation run on 12 samples of the real sinusoidal RF reference signal near its zero-crossing, producing an estimate of the true zero-crossing which is less susceptible to noise

The linear regression was implemented and found to work well in finding the offsets. An example of the algorithm run on real data is shown in Fig. 5.5. As the last step, even though the number of elementary operations performed by the algorithm was already relatively small, the author further reworked the solution to further reduce the run time. First, it is known that the y-values are the acquired samples, and the x-values are indices relative to the zero-crossing. As such, the mean of the x-values is constant, as with more samples, both the positive and negative x-values are added symmetrically. Finally, the variance can also be calculated analytically, since all the x-values and their mean are known.

5.3 FESA server

As a part of this thesis, a FESA class to interface with the developed Windows application was created. For developing the FESA class, CERN's internal FESA training course and various examples of existing classes were consulted. After the class definition was complete, previously created custom code, such as the *DigitizerStub* and *Communicator* classes could be embedded in the generated code to be called in real-time actions.

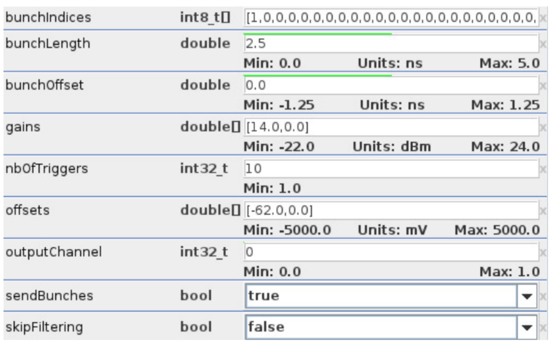
As the first step, device fields were defined. These included all the settings that are to be sent using the communication protocol, as well as some configuration, such as the hostname and port number of the Windows application. Acquisition fields included both acquisition data, and metadata, but also a duplication of the configuration parameters of the digitizer, as these can be queried from it for verification. Next, some properties were created. One setting property is responsible for managing all the digitizer configuration parameters and also additional parameters passed as arguments with some commands. Another setting property was created for setting the operation enumeration to be used for polling the server whether an operation has been completed. Multiple acquisition properties were added: acquisition data, metadata, configuration parameters returned by the digitizer, and the completion status of a specified operation. In addition to setting and acquisition properties, two command properties were also created. The first enables the client to manually perform enumerated operations in any order and time. The second allows the client to perform a frequently used sequence of operations: set the configuration, perform an acquisition, and read the data.

Three real-time actions were created to fulfil all functionality. Firstly, an action was needed for receiving the data. Inserted into it is custom code listening for incoming messages on the data socket and writing them to a device field, after converting the data in the form of ADC codes to voltages. Another real-time action is responsible for performing operations based on the command input by the client. The third action performs the predetermined sequence of operations. It is also triggered by input from the client. While the first action only receives data, the second and third actions exchange requests and responses with the Windows application.

The fact that input validation is supported by FESA was used. All value-items in properties were assigned minimum and maximum values. Also, units were specified, in order to document the class and make it easier to operate. The minimum-maximum values are enforced by FESA by default, however, custom validation can also be provided. For example, in the case of arrays, the default validation does not function properly, so in that case, custom code was written to check all the indices in an array. Even though input validation was already implemented in the Windows application, the advantage of also doing this in FESA is that the incorrect inputs would be caught sooner and the user could be notified immediately, instead of having to communicate with the Windows application and receive back an error message.

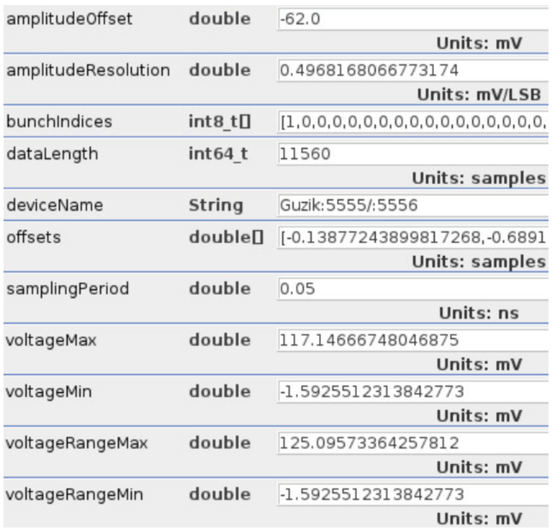
6 Validation

After the proof-of-concept software had been developed, it needed to be validated that it had met the users' needs and fulfilled the requirements. As the users will operate the digitizer via FESA, an application called FESA Navigator was used to validate program behaviour. FESA Navigator is a GUI client to the FESA server running on the FEC. It allows the users to modify settings, send commands, and read out the data. It also has some plotting capabilities for the data. In this section, example usage is described to demonstrate the created solution. After the development, the digitizers were installed in the LHC, and therefore real signals were used for these examples.



bunchIndices	int8_t[]	[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,			
bunchLength	double	2.5	Min: 0.0	Units: ns	Max: 5.0
bunchOffset	double	0.0	Min: -1.25	Units: ns	Max: 1.25
gains	double[]	[14.0,0.0]	Min: -22.0	Units: dBm	Max: 24.0
nbOfTriggers	int32_t	10	Min: 1.0		
offsets	double[]	[-62.0,0.0]	Min: -5000.0	Units: mV	Max: 5000.0
outputChannel	int32_t	0	Min: 0.0		Max: 1.0
sendBunches	bool	true			
skipFiltering	bool	false			

(a) FESA settings corresponding to the digitizer's configuration parameters and other parameters related to, for example, data processing. Limited input ranges can also be seen as a part of input validation



amplitudeOffset	double	-62.0	Units: mV
amplitudeResolution	double	0.4968168066773174	Units: mV/LSB
bunchIndices	int8_t[]	[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,	
dataLength	int64_t	11560	Units: samples
deviceName	String	Guzik:5555/:5556	
offsets	double[]	[-0.13877243899817268,-0.6891	Units: samples
samplingPeriod	double	0.05	Units: ns
voltageMax	double	117.14666748046875	Units: mV
voltageMin	double	-1.5925512313842773	Units: mV
voltageRangeMax	double	125.09573364257812	Units: mV
voltageRangeMin	double	-1.5925512313842773	Units: mV

(b) FESA acquisition data corresponding to the metadata received from the digitizer, as well as some values calculated based on the metadata, such as the ADC input voltage range

Figure 6.1: Screenshots of the FESA Navigator showing FESA settings and acquisition data

As can be seen in Fig. 6.1a, the users have the capability of changing the configuration parameters determined in the user needs analysis. Based on the depicted configuration, the first bunch after a trigger signal is to be acquired for 10 triggers or turns. The WCM's digitizer input gains and offsets have been modified to maximally use the input range of the ADC. Some other parameters have also been set, such as the first bunch detection threshold in millivolts, the bunch length in nanoseconds, and the last two flags which control whether filtering for data reduction is performed and whether for the digitizer to send back the bunch indices it used as input.

After setting the configuration parameters, an acquisition was performed and the data was sent back to FESA. In Fig. 6.1b, the received metadata can be seen. All the values required by the users are shown, such as the acquisition timestamp, sampling rate, and calculated phase offsets for each requested bunch per each turn. Fig. 6.2 shows the received data, already converted to voltages, in two plots: the first shows the concatenated profiles of a single bunch over 10 turns, while the second shows them overlaid on top of each other. Not seen in the figures, the functionality to save data to files was also tested and found to work.

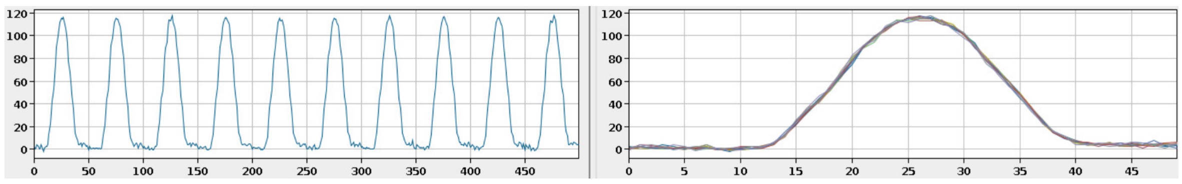


Figure 6.2: Screenshot of FESA Navigator plotting concatenated as well as overlaid profiles of a single bunch over 10 turns. The x-axes is in samples and the y-axes in millivolts

In order to meet the timing requirements, the program needed to be fast enough to configure the digitizer and acquire one turn of data in under one second minus the time it takes to configure the digitizer again and start another acquisition. The time it took to configure the digitizer, acquire a whole turn, and to send back the data, was measured based on FESA log statements. Based on 10 measurements the average time was 279.665 ms with a standard deviation of 14.964 ms. This is well below the one-second limit and even three consecutive configuration-acquisition-readout procedures could be performed in that time period. Therefore, the timing requirement was considered to be fulfilled.

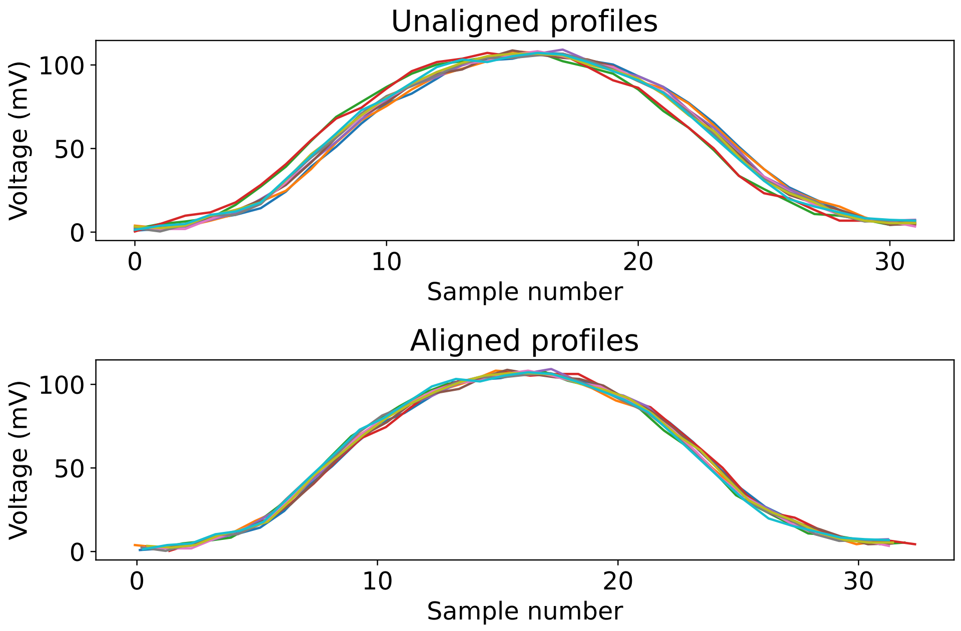


Figure 6.3: Bunches profiles from multiple turns overlaid, both without the sub-sample phase correction and with

Finally, to validate whether the calculated phase offsets are correct and of potential value to the users, profiles of a single bunch were acquired over multiple turns and plotted both unmodified, as well as aligned by the received phase offsets. As can be seen from Fig. 6.3, the alignment works well, centring the bunch profiles.

7 Conclusion

This thesis focused on the development of a proof-of-concept software solution to enable novel beam instrumentation techniques using a high-speed digitizer. As the first step, the digitizer's performance was evaluated. Limitations were found, such as the bottleneck of the interface between the digitizer and its host PC. Secondly, the users of the developed software were consulted and their needs were analysed, as well as the requirements for the software generated. Based on the user needs and previously found limitations, software for the digitizer host PCs was developed and a FESA class was created to enable online access to the acquired bunch profiles. As a validation, all the required hardware was installed in the LHC and the proof-of-concept software was tried out with real signals. It was found to fulfil the users' needs.

Immediate future endeavours are geared towards automating the acquisition for the two beam instrumentation use cases. To this end, the FESA class is currently being developed further. In addition, a continuous integration and delivery pipeline is being developed for the Windows application to allow for quick and automatic software updates, as well as automated testing. The final goal is to make the developed system fully operational and able to completely satisfy the two use cases.

Acknowledgements

I want to thank

my supervisors Veiko and Tom for supporting me throughout this process,

colleagues from SY-BI-IQ and SY-BI-SW for their help and encouragement

and finally, my friends and family who are always there.

Bibliography

[1] *Our Mission — CERN*. URL: https://home.cern/about/who-we-are/our-mission (visited on 12/12/2022).

[2] *The Standard Model — CERN*. URL: https://home.cern/science/physics/standard-model (visited on 05/08/2023).

[3] *The History of CERN — Timeline.Web.Cern.Ch*. URL: https://timeline.web.cern.ch/timeline-header/89 (visited on 04/10/2023).

[4] *The Accelerator Complex — CERN*. URL: https://www.home.cern/science/accelerators/accelerator-complex (visited on 12/12/2022).

[5] *The Large Hadron Collider — CERN*. URL: https://www.home.cern/science/accelerators/large-hadron-collider (visited on 12/12/2022).

[6] *LHC the Guide FAQ — CERN*. URL: https://home.cern/resources/brochure/knowledge-sharing/lhc-facts-and-figures (visited on 04/10/2023).

[7] Wit Busza, Krishna Rajagopal, and Wilke van der Schee. "Heavy Ion Collisions: The Big Picture and the Big Questions". In: *Annual Review of Nuclear and Particle Science* 68.1 (2018), pp. 339–376. DOI: 10.1146/annurev-nucl-101917-020852. URL: https://doi.org/10.1146/annurev-nucl-101917-020852.

[8] *Large Hadron Collider: The Discovery Machine - Scientific American*. URL: https://www.scientificamerican.com/article/the-discovery-machine-hadron-collider/ (visited on 05/08/2023).

[9] *Linear Accelerator 4 — CERN*. URL: https://home.cern/science/accelerators/linear-accelerator-4 (visited on 05/08/2023).

[10] *The Low Energy Ion Ring — CERN*. URL: https://home.cern/science/accelerators/low-energy-ion-ring (visited on 05/08/2023).

[11] *ELENA - Home*. URL: https://espace.cern.ch/elena-project/SitePages/Home.aspx (visited on 05/08/2023).

[12] Fabienne Landua. "The CERN Accelerator Complex Layout in 2022. Complexe Des Accélérateurs Du CERN En Janvier 2022". In: (2022). URL: https://cds.cern.ch/record/2813716.

[13] *Experiments — CERN*. URL: https://home.cern/science/experiments (visited on 05/09/2023).

[14] *Contribute to Society — CERN*. URL: https://home.cern/about/what-we-do/our-impact (visited on 05/08/2023).

[15] *Forty Years since the First PET Image at CERN*. CERN. Mar. 23, 2023. URL: https://home.cern/news/news/knowledge-sharing/forty-years-first-pet-image-cern (visited on 05/08/2023).

[16] *CNAO: Treating Cancer with Ions since 2011 — Knowledge Transfer*. URL: https://kt.cern/success-stories/cnao-treating-cancer-ions-2011 (visited on 05/08/2023).

[17] "Medipix. Medipix". 2015. URL: http://cds.cern.ch/record/2730889.

[18] *The Birth of the Web — CERN*. URL: https://home.cern/science/computing/birth-web (visited on 05/08/2023).

[19] James Manyika and Charles Roxburgh. *The Great Transformer: The Impact of the Internet on Economic Growth and Prosperity*. McKinsey Global Institute, 2011, p. 10. URL: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-great-transformer.

[20] *The Impact of CERN*. Dec. 2016. URL: https://home.cern/sites/default/files/2018-07/CERN-Brochure-2016-005-Eng.pdf (visited on 05/08/2023).

[21] *Collide — Arts · at · CERN*. URL: https://arts.cern/programme/collide (visited on 05/08/2023).

[22] Massimo Florio, Stefano Forte, and Emanuela Sirtori. "Forecasting the Socio-Economic Impact of the Large Hadron Collider: A Cost–Benefit Analysis to 2025 and Beyond". In: *Technological Forecasting and Social Change* 112 (Nov. 2016), pp. 38–53. DOI: 10.1016/j.techfore.2016.03.007. URL: https://doi.org/10.1016%2Fj.techfore.2016.03.007.

[23] Andrea Bastianin and Massimo Florio. *Social Cost Benefit Analysis of HL-LHC*. CERN-ACC-2018-0014. Geneva: CERN, 2018. URL: https://cds.cern.ch/record/2319300.

[24] *Report on the Impacts of Large Research Infrastructures on Economic Innovation and on Society: Case Studies at CERN*. Paris, 2014. URL: https://cds.cern.ch/record/1708387.

[25] S. Humphries. *Principles of Charged Particle Acceleration*. Dover Books on Physics Series. Dover Publications, Incorporated, 2012. ISBN: 978-0-486-49818-8. URL: https://books.google.ch/books?id=9ELnk0jZw70C.

[26] Frank Tecker. "Longitudinal Beam Dynamics in Circular Accelerators". Version 1. In: *Proceedings of the CERN–Accelerator–School course: Introduction to Particle Accelerators* (2020). DOI: 10.48550/ARXIV.2011.02932. URL: https://arxiv.org/abs/2011.02932 (visited on 05/08/2023).

[27] Simone Di Mitri. "Low Energy Accelerators". In: *Fundamentals of Particle Accelerator Physics*. Cham: Springer International Publishing, 2022, pp. 25–36. ISBN: 978-3-031-07662-6. URL: https://doi.org/10.1007/978-3-031-07662-6_2.

[28] F Hinterberger. "Electrostatic Accelerators". In: (2006). DOI: 10.5170/CERN-2006-012.95. URL: https://cds.cern.ch/record/1005042.

[29] Wolfgang K. H. Panofsky. "Evolution of Particle Accelerators and Colliders". In: *SLAC Beam Line* 27N1.SLAC-REPRINT-1997-101 (1999), pp. 36–44.

[30] David Alesini. "Linac". Version 1. In: *Proceedings of the CERN–Accelerator–School course: Introduction to Particle Accelerators* (2021). DOI: 10.48550/ARXIV.2103.16500. URL: https://arxiv.org/abs/2103.16500 (visited on 05/08/2023).

[31] Simon Baird. *Accelerators for Pedestrians*. Geneva, Switzerland, Feb. 7, 1997. 147 pp.

[32] Robert C. Webber. "Longitudinal Emittance: An Introduction to the Concept and Survey of Measurement Techniques Including Design of a Wall Current Monitor". In: *AIP Conference Proceedings* 212.1 (Oct. 2, 1990), pp. 85–126. ISSN: 0094-243X. DOI: 10.1063/1.39712. URL: https://doi.org/10.1063/1.39712 (visited on 05/09/2023).

[33] Diogo Alves and Kacper Lasocha. "Kalman Filter-Based Longitudinal Phase-Space Reconstruction Method for Hadron Machines". In: *Physical Review Accelerators and Beams* 24.072801 (July 2, 2021). ISSN: 2469-9888. DOI: 10.1103/PhysRevAccelBeams.24.072801.

[34] Lyndon R Evans and Philip Bryant. "LHC Machine". In: *JINST* 3.S08001 (2008), p. 164. DOI: 10.1088/1748-0221/3/08/S08001. URL: https://cds.cern.ch/record/1129806.

[35] R Bailey and Paul Collier. *Standard Filling Schemes for Various LHC Operation Modes*. LHC-PROJECT-NOTE-323. Geneva: CERN, 2003. URL: https://cds.cern.ch/record/691782.

[36] Kacper Lasocha and Diogo Alves. "Estimation of Longitudinal Bunch Characteristics in the LHC Using Schottky-based Diagnostics". In: *Physical Review Accelerators and Beams* 23.6 (June 2020), p. 062803. DOI: 10.1103/PhysRevAccelBeams.23.062803. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.23.062803.

[37] Kacper Lasocha and Diogo Alves. "Estimation of Transverse Bunch Characteristics in the LHC Using Schottky-based Diagnostics". In: *Physical Review Accelerators and Beams* 25.6 (June 2022), p. 062801. DOI: 10.1103/PhysRevAccelBeams.25.062801. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.25.062801.

[38] Kacper Lasocha. "Non-Invasive Beam Diagnostics with Schottky Signals and Cherenkov Diffraction Radiation". Kraków, Poland: Jagiellonian University in Kraków, 2022. URL: https://cds.cern.ch/record/2846538.

[39] S. Hancock et al. "Tomographic Measurements of Longitudinal Phase Space Density". In: *Computer Physics Communications* 118.1 (1999), pp. 61–70. ISSN: 0010-4655. DOI: 10.1016/S0010-4655(99)00194-0. URL: https://www.sciencedirect.com/science/article/pii/S0010465599001940.

[40] Gero Kube. "Beam Diagnostic Requirements: An Overview". Version 1. In: *Proceedings of the 2018 CERN–Accelerator–School course on Beam Instrumentation* (2020). DOI: 10.48550/ARXIV.2005.08389. URL: https://arxiv.org/abs/2005.08389 (visited on 05/06/2023).

[41] Patrick Odier. "A New Wide Band Wall Current Monitor". In: (May 2023).

[42] M. Wendt. "BPM Systems: A Brief Introduction to Beam Position Monitoring". Version 1. In: *Proceedings of the 2018 CERN–Accelerator–School course on Beam Instrumentation* (2020). DOI: 10.48550/ARXIV.2005.14081. URL: https://arxiv.org/abs/2005.14081 (visited on 05/15/2023).

[43] T Bohl and J F Malo. *The APWL Wideband Wall Current Monitor*. CERN-BE-2009-006. Geneva: CERN, 2009. URL: https://cds.cern.ch/record/1164165.

[44] T. E. Levens et al. "Automatic Detection of Transverse Beam Instabilities in the Large Hadron Collider". In: *Physical Review Accelerators and Beams* 22.11 (Nov. 20, 2019), p. 112803. ISSN: 2469-9888. DOI: 10.1103/PhysRevAccelBeams.22.112803. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.22.112803 (visited on 05/14/2023).

[45] *ADC6000 Series 8-Bit Digitizers - Guzik Technical Enterprises*. URL: https://www.guzik.com/product/adc6000-series-8-bit-digitizers/#Specifications (visited on 04/27/2023).

[46] L Fernandez et al. "Front-End Software Architecture". In: ICALEPS. Knoxville, Tennessee, USA, 2007. URL: https://accelconf.web.cern.ch/ica07/papers/WOPA04.pdf.

[47] *Home · Wiki · Projects / FMC DEL 1ns 2cha · Open Hardware Repository*. URL: https://ohwr.org/project/fmc-del-1ns-2cha/wikis/home (visited on 05/20/2023).

[48] Tyler Andrews. “Computation Time Comparison between Matlab and C++ Using Launch Windows”. In: 2012.

[49] *Welcome Back to C++ - Modern C++ — Microsoft Learn*. URL: https://learn.microsoft.com/en-us/cpp/cpp/welcome-back-to-cpp-modern-cpp?view=msvc-170 (visited on 05/09/2023).

[50] *The HDF5® Library & File Format - The HDF Group*. URL: https://www.hdfgroup.org/solutions/hdf5/ (visited on 05/14/2023).

[51] *User’s Guide for the M9731/2/4G AXIe Waveform Digitizers — Keysight*. URL: https://www.keysight.com/us/en/assets/9018-07530/user-manuals/9018-07530.pdf (visited on 04/27/2023).

[52] *First Look at Profiling Tools - Visual Studio (Windows) — Microsoft Learn*. URL: https://learn.microsoft.com/en-us/visualstudio/profiling/profiling-feature-tour?view=vs-2022#measure-performance-while-debugging (visited on 05/14/2023).

[53] Gianluigi Arduini et al. “The SPS Beam Parameters, the Operational Cycle, and Proton Sharing with the SHiP Facility”. In: (2015). URL: https://cds.cern.ch/record/2063023.

[54] J. Lauener and W. Sliwinski. “How to Design & Implement a Modern Communication Middleware Based on ZeroMQ”. In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems*. International Conference on Accelerator and Large Experimental Control Systems (16th). International Conference on Accelerator and Large Experimental Control Systems. Geneva, Switzerland: JACoW, Jan. 2018, pp. 45–51. ISBN: 978-3-95450-193-9. DOI: 10.18429/JACoW-ICALEPCS2017-MOBPL05. URL: http://jacow.org/icalepcs2017/papers/mobpl05.pdf.

[55] C. Ghabrous Larrea et al. “IPbus: A Flexible Ethernet-based Control System for xTCA Hardware”. In: *Proceedings, Topical Workshop on Electronics for Particle Physics (TWEPP14): Aix En Provence, France, September 22-26, 2014*. Vol. 10. JINST. 2015, p. C02019. DOI: 10.1088/1748-0221/10/02/C02019.

[56] *About gRPC — gRPC*. URL: https://grpc.io/about/ (visited on 05/20/2023).

[57] *Introduction to gRPC — gRPC*. URL: https://grpc.io/docs/what-is-grpc/introduction/ (visited on 05/20/2023).

[58] Pieter Hintjens. *ZeroMQ*. O’Reilly Media, Inc. URL: https://zguide.zeromq.org/ (visited on 05/06/2023).

[59] Shane W. Grant and Randolph Voorhies. *Cereal - A C++11 Library for Serialization*. 2017. URL: https://uscilab.github.io/cereal/ (visited on 05/19/2023).

[60] R.W. Wall. “Simple Methods for Detecting Zero Crossing”. In: *IECON’03. 29th Annual Conference of the IEEE Industrial Electronics Society (IEEE Cat. No.03CH37468)*. Vol. 3. 2003, 2477–2481 Vol.3. DOI: 10.1109/IECON.2003.1280634.

[61] Ronald W. Schafer. “What Is a Savitzky-Golay Filter? [Lecture Notes]”. In: *IEEE Signal Processing Magazine* 28.4 (July 2011), pp. 111–117. ISSN: 1558-0792. DOI: 10.1109/MSP.2011.941097.

[62] Balázs Renczes, István Kollár, and Tamás Dabóczi. “Efficient Implementation of Least Squares Sine Fitting Algorithms”. In: *IEEE Transactions on Instrumentation and Measurement* 65.12 (2016), pp. 2717–2724. DOI: 10.1109/TIM.2016.2600998.

[63] *1.2 - What Is the ”Best Fitting Line”? — STAT 501*. URL: https://online.stat.psu.edu/stat501/lesson/1/1.2 (visited on 05/19/2023).

A Appendix: Configuration parameters

Table A.1: A proposal of digitizer configuration parameters and arguments for data processing to be set from FESA and sent via the communication protocol. The final decision on whether to include a parameter, arrived at in cooperation with the users, is in the last column

Configuration parameter	**Description**	**Required? (proposal)**	**Size**	**Comment**	**Required? (final decision)**
Gain (global)	Applies to channels for which individual gains have not been set	-	8 B	Maybe easier just to set 2 invidual gains?	-
Gain (individual)	Preamplifier gain by channel	+	16 B (2 CH)	Definitely needed.	+
Use only calibrated gains	If true, then the specified gain values are coerced to the ones for which the ADC frequency characteristics are calibrated.	-	1 B	To be set in advance / predefined.	-
Offset (global)	Meaning of ”global” the same as with gain. Offset itself means the offset (in mV) of the built-in variable amplifier.	-	8 B	Maybe easier just to set 2 invidual offsets?	+
Offset (individual)	Preamplifier offset by channel	+	16 B (2 CH)	Definitely needed.	+

Amplitude factor (global)	Meaning of ”global” the same as with gain. Amplitude factor is basically a gain multiplier. A more detailed explanation can be found from the SDK manual, p 148.	-	8 B	Not going to use this parameter at all, as gains and offsets are sufficient.	-
Amplitude factor (individual)	Amplitude factor by channel	-	16 B (2 CH)	Same as above.	-
Adaptive configuration	Whether to configure adaptively or not.	?	1 B	Adaptive configuration takes less time but only works for changing gains and offsets. Can leave the option in for testing.	+
Acquisition time	In ns. If <= 0, then the acquisition will be long enough to cover the specified amount and duration of sectors.	-	8 B	Should be determined by the bunch indices and other parameters.	-
Bunch indices	Only the profiles of specified bunches should be sent back. 3564 values.	+	446 B	Size = ceil(3564 / 8) using a bitmask. If 16-bit bucket numbers/indices were sent instead, 223 buckets could be designated using the same amount of data.	+

Send bunch indices flag	Whether to send bunch indices back after an acquisition or not.	?	1 B	If the receiver of the data should not be the one to set the configuration parameters, this would be needed, otherwise not. For testing and verification, can be left in.	+
Skip filtering flag	Send all the acquired data back, do not filter.	+	1 B	To be used for debugging or when a whole turn's worth of data is needed, for example.	+
Output channel	Related to the skip filtering flag, when all the data is sent back, selects which channel to send.	+	4 B	Can be used if the RF reference signal is also needed.	+
Bunch length	Length of a single bunch.	+	8 B	Can be used for filtering data, to only copy the length specified to the output buffer.	+
Bunch-bucket offset	Difference between the bunch and bucket centres.	+	8 B	Used in filtering data to correct for the offset in order to copy the correct samples. Empirically determined from the reference signal zero-crossings. Due to cabling delays.	+
Bunch detection threshold	From which signal level can something be determined to be a bunch.	+	8 B	For filtering the bunches.	+

Number of segments	Number of segments to acquire.	+	8 B	Corresponds to the number of turns.	+
Sector time	Acquisition time per sector	-	8 B	Should be determined by the bunch indices and other parameters.	-
Acquisition timeout	In ms. After this time the measurement function will return.	-	4 B	To be set in advance, can be -1, meaning infinitely blocking.	-
Adjust up	Whether the amount of samples acquired should always be at least as high as specified by the acquisition duration, or the opposite of always being smaller.	?	1 B	If needed at all, rather set in advance.	-
List of input channels	List of digitizer's input channels.	-	8 B	It would be the easiest if this was set at the start of the program and never changed.	-
Trigger mode	Internal vs external	-	1 B	It would never realistically change during the program's lifetime, so to be set in advance.	-
Trigger input	The input channel / external trigger port	-	4 B	To be predefined.	-
Trigger condition	Rising or falling slope	-	1 B	To be predefined.	-
Trigger threshold	In V. Default is 0.9 V.	-	8 B	To be predefined, as we can change the trigger pulse itself if needed.	-

Trigger delay	In μs. Can be used also for pre-triggered acquisitions.	-	8 B	Probably will not be needed because we would already use a controlled external trigger.	-
Command	An enum to specify commands to the digitizer	+	4 B	To command the digitizer.	+

B Appendix: Metadata

Table B.1: A proposal of digitizer metadata parameters as well as the data itself to be received by FESA and sent via the communication protocol. The final decision on whether to include a parameter, arrived at in cooperation with the users, is in the last column

Configuration parameter	**Description**	**Required? (proposal)**	**Size**	**Comment**	**Required? (final decision)**
Timestamp	Timestamp of the acquisition in ns from the epoch.	+	8 B	Timestamp or some other unique identifier	+
Device name	To identify the beam/device.	+	<= 32 B?	To be determined what exactly, maybe hostname + port.	+
Sampling period	Sampling period of the digitizer in ns.	+	8 B	For verification and conversions between ns and samples.	+
Bunch indices	Same as the input parameter.	+	446 B, if sent	For verification and/or parsing the data.	+
Amplitude offset	Amplitude offset in signed LSB.	+	8 B	Can be used to remove the data offset.	+
Amplitude resolution	The final amplitude resolution in mV/LSB.	+	8 B	Needed because it cannot be exactly calculated based on the input parameters, as the digitizer has final control over it.	+
Data type	Whether it is 8-bit or 16-bit.	-	1 B	Since we will only use 8-bit data, it is not needed.	-

Data	A byte array containing the acquisition data.	+	Max 32 GB per channel	Definitely needed. Due to large size, should possibly be sent separately from others.	+
Data length	Length of the data in samples or bytes.	+		For verification and/or parsing the data.	+

C Appendix: Requirements

Table C.1: General description of the two main use cases, as well as their commonalities

UID	Description
GUC0	**Common for all use cases** The user specifies the configuration parameters, performs the acquisition, and receives the acquired data via FESA. The data should also be saved to HDF5 files if needed. On the software side, since data reduction is likely to be required, the users should be able to select which bunch profiles they would like to digitise. It is also assumed that the user has control over a custom trigger generator to generate trigger pulses for the digitizer at desired times. Therefore, the digitizer has to use external triggering. In addition to the wall current monitor signal, the 400 MHz RF reference signal also has to be digitised, in order to perform the aforementioned data reduction and to align the data afterwards. For the alignment, sub-sample precision phase offsets of the RF signal have to be found for each selected bunch and reported back to the users. For maximum time resolution of both the bunch profiles and the RF reference signal, sampling with the maximum frequency is desired. As a constraint, the digitizer drivers only exist for Windows and FESA only for Linux. Therefore, there has to be separate software for Windows which communicated with the FESA software. The communication should be as fast as possible and the overall acquisition system should have as low overhead as possible. The acquisition system should ideally never crash, as part of it is located in the LHC service gallery and serious issues could be difficult to fix. Moreover, external dependencies should only be used when support is guaranteed or when no other option is available.
GUC1	**Schottky signal analysis** The user should receive one bunch profile every $\sim$10 seconds. However, priority is given to GUC2, and as such, some missed acquisitions are acceptable.
GUC2	**Longitudinal phase space reconstruction** The users should be able to receive data gathered in $\sim$30 after an injection into the LHC. In the proof-of-concept software, automating the start of the acquisition is not required, however, the software should be fast enough to react to the 1 s injection warning signal.

Table C.2: Use cases

UID	Description
UC1	Goal: configure the acquisition system Preconditions: the user has opened the FESA Navigator and gained access to the right instance. Steps: The user opens the settings property. The user sets the desired settings – the digitizer's configuration parameters. If any incorrect values are set, the system notifies the user. The user opens the commands property. The user commands the system to perform the configuration. The system lets the user know whether the operation was successful.
UC2	Goal: arm the digitizer Preconditions: the user has opened the FESA Navigator and gained access to the right instance. In addition, the digitizer has been configured as needed. Steps: The user opens the commands property. The user commands the system to arm the digitizer. The system lets the user know whether the operation was successful.
UC3	Goal: receive data Preconditions: the user has opened the FESA Navigator and gained access to the right instance. In addition, the digitizer has been configured as needed. Steps: The user opens the commands property. The user commands the system to prepare and send the data. The system lets the user know whether the operation was successful. The user opens the data property. The user can get the data and metadata.
UC4	Goal: save data to file Preconditions: the user has opened the FESA Navigator and gained access to the right instance. In addition, the digitizer has been configured as needed. Steps: The user opens the commands property. The user commands the system to save the data. The system lets the user know whether the operation was successful. The user can access the data from the filesystem.

Table C.3: User requirements

UID	Type	Description	Source
UR1	Non-functional	The users shall be able to interact with the acquisition system via FESA	UC1–4
UR2	Functional	The user shall be able to configure the acquisition system	UC1

UR3	Functional	The user shall be able to modify the specified configuration parameters	UC1
UR4	Functional	The user shall be able to receive longitudinal bunch profiles	UC3
UR5	Functional	The user shall be able to select which bunches' profiles are to be sent	GUC0, GUC1
UR6	Non-functional	The bunch profile data should be in volts	GUC0
UR7	Functional	The user shall be able to save acquisition data	UC4
UR8	Non-functional	The HDF5 data format should be used to save files	GUC0
UR9	Functional	The user shall receive metadata about acquisition data	UC3
UR10	Functional	The received metadata shall correspond to specifications of the users	UC3
UR11	Functional	The user shall receive phase offsets of bunches of an acquisition, if so specified	UC3
UR12	Non-functional	The system shall react fast enough to meet the 1 s injection warning real-time deadline.	GUC2
UR13	Non-functional	The digitizer shall sample with its highest sampling rate	GUC0
UR14	Non-functional	The acquisition system should not crash if an error occurs	GUC0
UR15	Non-functional	The system should be able to acquire as long as needed and possible, even if it requires acquiring less frequently	GUC2
UR16	Functional	The system shall report the statuses of operations to the user	UC1–4
UR17	Functional	The system shall report any errors to the user	UC1–4
UR18	Non-functional	The system should avoid using third-party dependencies when possible	GUC0

Table C.4: System requirements

UID	**Type**	**Description**	**Source**
SR1	Functional	The beam signals shall be digitised	UR4
SR2	Functional	The RF reference signals shall be digitised	UR11
SR3	Non-functional	The acquisition system shall use a Windows application to interface with a digitizer	UR3
SR4	Non-functional	The acquisition system shall use Linux to host the FESA server.	UR1
SR5	Functional	The two applications on different operating systems shall be able to communicate with each other	UR1, UR3
SR6	Non-functional	The communication between the two applications should be as fast as possible given existing hardware	UR12

SR7	Functional	There shall be a communication protocol to communicate between the two applications	SR3, SR4
SR8	Functional	The communication protocol shall enable sending configuration parameters	UR3
SR9	Functional	The communication protocol shall enable sending metadata	UR9
SR10	Functional	The communication protocol shall enable sending acquisition data	UR4
SR11	Functional	The communication protocol shall enable sending commands to the digitizer	UR1
SR12	Functional	The communication protocol shall enable propagating errors from the Windows application to the FESA server	UR14, UR17
SR13	Functional	The communication protocol shall enable propagating operation completion statuses from the Windows application to the FESA server	UR16
SR14	Non-functional	The digitizer shall use external triggering	GUC0
SR15	Non-functional	The digitizer shall use segmented acquisition	UR15
SR16	Functional	The Windows software shall minimise the data rate when needed	UR5, UR15
SR17	Functional	The Windows software shall filter bunches	UR5
SR18	Functional	The Windows software shall calculate phase offsets	UR11
SR19	Functional	The acquisition system shall validate user input	UR14

Table C.5: Hardware requirements

UID	**Type**	**Description**	**Source**
HWR1	Non-functional	20 GS/s sampling rate shall be used	UR13
HWR2	Non-functional	External triggering shall be used	SR14
HWR3	Non-functional	There shall be one host PC for each digitizer to run the Windows application	SR3
HWR4	Non-functional	There shall be one Front End Computer (FEC) to host the FESA server	SR4
HWR5	Non-functional	The host PCs shall have a 10 GbE connections	SR6
HWR6	Non-functional	The FEC shall have two 10 GbE connections	SR6
HWR7	Non-functional	The FEC shall have a 1 GbE connection	UR1

Table C.6: Software requirements

UID	**Type**	**Description**	**Source**
SWR1	Functional	The Windows application shall use the provided digitizer SDK to control the digitizer	SR3
SWR2	Functional	The Windows application shall expose non-blocking methods to control the digitizer	SR6, UR15

SWR3	Functional	The Windows application shall double-buffer the acquisition data if requested	SWR2
SWR4	Functional	The Windows application shall be able to receive requests from the FESA server using the communication protocol	SR5, SR8
SWR5	Functional	The Windows application shall be able to send responses to the FESA server using the communication protocol	SR5, SR9
SWR6	Functional	The Windows application shall be able to send data to the FESA server using the communication protocol	SR5, SR10
SWR7	Functional	The Windows application shall be able to filter out unnecessary RF buckets	SR16
SWR8	Functional	The Windows application shall be able to detect zero-crossings as bucket centres in the RF signal	SWR7
SWR9	Functional	The Windows application shall calculate sub-sample phase offsets between bunches and their respective buckets	SR18
SWR10	Functional	The Windows application should be able to detect the beginning of the first bunch in the data array	SWR7
SWR11	Functional	The Windows application shall save data to HDF5 files	UR8
SWR12	Functional	The Windows application shall validate input parameters whenever possible	SR19
SWR13	Functional	The FESA class design shall have a property with value-items corresponding to the configuration parameters specified by the users	UR3
SWR14	Functional	The FESA class design shall have a property with value-items corresponding to the metadata specified by the users	UR9, UR10, SR4
SWR15	Functional	The FESA class design shall have a property for acquisition data	UR4, SR4
SWR16	Non-functional	The FESA class design shall use real-time actions to communicate with the Windows application	UR12, SR6
SWR17	Functional	The FESA class shall validate input parameters whenever possible	SR19
SWR18	Functional	The FESA class should convert received data to voltages	UR6