Escuela Técnica Superior de Ingeniería
Universidad de Huelva

Degree in Energy Engineering

Final Degree Project

Design of a control system for stepper motors with micro-metric precision employed in the beam emittance measurement of the Linac4 at CERN

Domingo Gómez Domínguez

July 2017
\LaTeX{} has been employed to produce this manuscript, designed to be double-sided printed.
Abstract

A new linear accelerator (Linac4) is being designed to replace its predecessor (Linac2) at CERN. The new Linac4 will double the initial intensity giving an injection energy of up to 160 MeV. It will be an essential component of the LHC (Large Hadron Collider). To assess the quality of the beam, monitoring systems are placed along the beam pipe, being one of them devoted to measure its emittance, the so-called emittance scanner. The measurement of the emittance is important since it constitutes one of the two main parameters that limits the overall LHC performance, being the other parameter the energy of the beam. While the energy level of the beam can be modified during different phases at CERN, the beam emittance cannot; it is determined by the first source that produces the beam. The beam emittance directly influences the amount of particles colliding. For this purpose, the Linac4 emittance scanner will be placed on the very first step of the whole CERN accelerator complex right after the particles source.

This project consists in the development of a control system for the emittance scanner of the Linac4 using LabVIEW as programming environment. This task was assigned to the author as part of his student placement at CERN, which has been achieved during a 5 months period. It has to be said that the whole project is in a continuous process of update.

The main contribution of the author to the emittance scanner of the Linac4 has been the creation of a control system that allows the user to execute a scan of the beam by introducing a set of scanning parameters, and displays back on the user interface the acquired measurements. Part of the public information of this project has been assembled to give form to this student final degree project.
Acknowledgments

To each one of the members of MTA team at CERN for all the support and the high level formation they gave me, their excellent knowledge, capabilities, advice, and the incredible way they are, all I have become this year I owe to you.

To CERN, for granting students like me such an excellent opportunity that has settled down my first huge step in the best possible professional environment.

To my project supervisor at UHU, for his masterly guidance and contribution to achieve the best bachelor thesis possible.

To my family, for their unconditional support and encouragement, which made possible this new stage in my life.

With me deepest gratitude, thank you all.

Domingo
Contents

Abstract 2

Acknowledgments 3

1 Introduction 9
  1.1 CERN and Linac4 ................................. 9
  1.2 Beam emittance ................................. 11
  1.3 LabVIEW as Programming Environment .......... 13

2 Instruments and methods 15
  2.1 Emittance Scanner .............................. 15
     2.1.1 Slit .................................. 15
     2.1.2 Secondary Emission Monitor (SEM-grid) ...... 17
     2.1.3 Stepper Motors ............................ 18
     2.1.4 Acquisition system and associated electronics . 19
  2.2 Design of the control system .................... 20
     2.2.1 Selection of a Programming Style, Actor Framework for Linac4 ......................... 20
     2.2.2 Design of the architecture ................. 21
     2.2.3 Design of the modules ..................... 25

3 Tests and experimental results 33
  3.1 Test on a server simulator ..................... 33
  3.2 Test on the scanner without beam ............... 34
  3.3 Debugging and Handling errors ................. 36
  3.4 In-beam measurements ......................... 38

4 Conclusions and future work 41

Bibliography 43

List of Figures 45
Appendix A  Stepper motor data sheet  51
Appendix B  ADC card data sheet  57
Appendix C  PLC data sheet  61
Appendix D  The basic elements in LabVIEW code  65
Appendix E  User interface of Linac4 LEM  71
Chapter 1

Introduction

This chapter gives an overall view of the basic concepts needed for the making of this project. It begins with a brief introduction of one of the most important research institutions in the world, CERN. Then, a general notion of the beam emittance of a particle beam is introduced. Finally, the programming environment LabVIEW employed in this project is presented.

1.1 CERN and Linac4

Funded in 1954, the European Organization for Nuclear Research (CERN, acronym for the French Conseil Européen pour la Recherche Nucléaire) aims to probe the fundamental structure of the universe with the study of the basic constituents of matter, accelerating particles that are made to collide together at close to the speed of light. Nowadays CERN [1] counts with 22 member states and leads all kind research in the field of high-energy physics, with numerous experiments and companies involved, which results are useful not only for matter science but for all of advances in other fields such as medical, material, etc.

The CERN complex is placed on the French-Swiss border in the northwest of Geneva, and spreads both underground and above-ground throughout different French and Swiss sites along of which, it is found the different experiments and installations as shown in Fig. 1.1. One may say that the most famous part of CERN is the Large Hadron Collider (LHC), the world’s largest and most powerful particle collider. But the particles are required to be prepared by a series of systems that successively increase their energy. The first step on this process is the Linear accelerator Linac2, generating 50 MeV protons.
CHAPTER 1. INTRODUCTION

Figure 1.1: Top, CERN complex with an overview of the main components—experiments, type of beams and remarked dates. Bottom, CERN aerial view showing the main experiments and villages/towns of the proximity.

Then, the Proton Synchrotron Booster (PSB) increases the energy to 1.4 GeV, which feeds the Proton Synchrotron (PS) that increases it to 26 GeV. Finally, the Super Proton Synchrotron (SPS) further increases the energy to 450 GeV before injecting it to the LHC.

The Linac4 project will allow to reach a new energy level necessary for the future High Luminosity Large Hadron Collider (HL-LHC). The Linac4 project will replace the current Linac2 in the shutdown of 2019-20 starting with the same energy level as Linac2 (50 MeV), and will increases it up to 160 MeV at 2025 for the HL-LHC. More information about Linac4 can be found in the official site [2].
1.2 Beam emittance

A beam is made out of a number of particles moving together closely at a certain speed. Even though most of the velocity vectors of the particles are parallel to the beam’s direction, some of the particles have a small deviation. This introduces a small perpendicular component that makes the beam not ideal, and the property that defines the non-idealism of a beam is referred to by the term emittance. It is therefore expressed as the “quality” of a beam and it may be said that having a low emittance means that beam’s quality is high, i.e. most of the particles’ velocity vectors are parallel to the whole beam vector, see Fig. 1.2. It is also related to the area (or volume) it occupies as represented in a phase space consisting of position and momentum. There are many possible definitions for emittance according to how the “area” is represented.

![Figure 1.2: Beam velocity vector \( \vec{V} \) compared with a portion of the velocity vectors of the particles travelling inside the beam.](image)

Mathematically, the emittance can be obtained considering that (i) particles adopt the shape of an ellipse, (ii) the area remains constant along all the beam trajectory, (iii) while the aspect ratio and orientation of the ellipse changes in the plane \((x, x')\) (slit position, momentum). Then it can be represented by this area as:

\[
\epsilon = \frac{1}{\pi} \int \int_A dx dx'
\]  

(1.1)
CHAPTER 1. INTRODUCTION

The beam emittance is often described by using the so called Courant-Snyder or Twiss parameters, they define the ellipse of a beam as shown in Figure (1.3) where:

- $\sqrt{\beta \epsilon}$ is the beam half width
- $\sqrt{\epsilon\gamma}$ is the beam half divergence.
- $\alpha$ describes how strongly $x$ and $x'$ are correlated.
- for $\alpha > 0$, beam is converging.
- for $\alpha < 0$, beam is diverging.
- for $\alpha = 0$, beam size has minimum (waist) or maximum (anti-waist).

Finally, the beam’s ellipse and its orientation described by the four Courant-Snyder or Twiss parameters ($\epsilon, \beta, \alpha$, and $\gamma$) can be represented as a function of the emittance according to:

$$\epsilon = \gamma x^2 + 2\alpha xx' + \beta x'^2 \quad \text{with} \quad \gamma = \frac{1 + \alpha^2}{\beta}$$

Therefore, it is possible to know the characteristics of a beam and its emittance, by calculating only these three parameters $\beta, \alpha$ and $\gamma$. 

Figure 1.3: Ellipse of the beam as defined by the Twiss or Courant-Snyder parameters, where each of them represent the distance respect the origin [3].
1.3 LabVIEW as Programming Environment

LabVIEW stands for Laboratory Virtual Instrument Engineering Workbench which is a programming environment developed by National Instruments [4] that uses a dataflow programming language called G. The execution is determined by how the different elements or nodes are connected on a block diagram. It is commonly used for data acquisition, instrument control and industrial automation, it enables multi-processing and multi-threading and provides a build-in user interface where data can be displayed selectively from any part of the code, as well as, allowing the user interact with the code at run-time through controls. Figure 1.4 shows the LabVIEW front panel environment, although other main parts are: the block diagram, the functions palette and project explorer.

![Figure 1.4: Project Explorer of a LabVIEW project (left). Front Panel shown to the final user (right).](image)

The Front Panel (Figure 1.4) contains all the information that the final user will see (string indicators, numeric indicators, graphs and waveforms, 3D plots, etc.) and the elements that the user can interact with (booleans buttons, text control, knobs, slides controls, etc.).
CHAPTER 1. INTRODUCTION

Figure 1.5: LabVIEW’s block diagram showing two basic mathematical nodes (left) and Functions Palette window with programming functions provided by LabVIEW (right).

The Block Diagram is where the code (algorithms) that reflects the operation, logic and data-flow of the program is created. Here, programmers create all the necessary operations that they need for a concrete goal by placing elements (functions) on the “white canvas” (Fig. 1.5, left).

The functions that are placed on the block diagram can be developed by us, provided by third companies, made by users communities or the ones provided by National Instrument as default or complement palettes. By default, LabVIEW already includes palettes that cover a wide range of basic functions as examples of palettes: Mathematics, Signal Processing, Data Communication, Connectivity, File I/O Timing, etc.

As the project grows and involves a wider range of drivers, sub-programs, documentation, etc. It is necessary to organize all the content related to a project, this can be done on the “File Explorer” window. Here, programmers can organize all the files related to a project, edit them, change their functionalities and behaviors, etc.
Chapter 2

Instruments and methods

A general review of the system employed for the measurement of the beam emittance of Linac4 is given in this chapter. The main parts of the emittance scanner, the acquisition system and the associated electronics are described. Then, the steps taken for the design of the control system and its programming architecture are explained.

2.1 Emittance Scanner

The device in charge of measuring the emittance of the beam is the so called emittance scanner, for Linac4 we refer to it under the acronym “LEM” (Linac4 Emittance Meter). It consists of two wired-grids (horizontal and vertical) for the measurement of the beam profile, two slits that only allow a proportion of the beam pass to the grids and three stepper motors that move the mentioned components. Associated electronics will handle the current signal collected by the wires (more details are given in the next sections). A 3D computer model of the LEM is shown in Fig. 2.1.

2.1.1 Slit

The slit is a solid plate material placed in the beam path, it allows only a certain amount of the beam particles (beamlet) to pass through a narrow aperture. The purpose of the slit is to allow the study of a selected beam portion (slice) by a profile monitor (also called secondary emission monitor SEM-grid) that would tell us the beam distribution while at the same time, it protects the grids from the full intensity of the beam. By moving the slit across the full section of the beam, we can acquire the whole transversed
profile of a beam. As shown in Fig. 2.2 the slit introduced a scattering effect that could lead to errors in the emittance measurement, the geometry and material of the slit has been studied to be able to select the one that minimize the effect. Different slit materials and geometries have been computer simulated by using Monte-Carlo code (for more information about the simulation results please consult ref. [5]). Figure 2.3 shows how the slits are displaced allowing to scan the beam both horizontally and vertically.

Figure 2.1: 3D model representation of the emittance scanner.

Figure 2.2: Schematic of the beam passing trough the slit and the scattering effect it produces as view by the profile monitor (SEM-grid) [5].
2.1. EMITTANCE SCANNER

2.1.2 Secondary Emission Monitor (SEM-grid)

It consists of a grid of wires mounted together in a same frame, see Fig. 2.4. Each wire interacts individually with the beam and outputs a current signal that is sent to a multiplexer and a amplifier to finally be digitalized by an ADC module. Linac4 actually has two SEM-grid sets, the number of wires can vary depending on the resolution aimed to achieve. The choice of the material for these wires has required a wide research to select the one that proves to reduce well, and support, the thermal effects caused by the beam, and still provide an adequate output signal (usually made of carbon or tungsten), more details of his in-depth research can be found at [6]. Overall, the current measurement process could be considered non-destructive.

Figure 2.3: Linac4 two-slits installation on the same plate, for horizontal and vertical phase-space beam analysis.

Figure 2.4: Close-up of one of the two SEM-grids (left). Full view of the structure where grids are mounted (right) [6].
Since the grids are positioned after the beam passes the slit, the sampling correspond to only one slice of the total beam, thus requiring the slit to move to enough positions to cover the full range of the beam. There is some scattering of the beam due to the slit interaction that leads to an increase of the total emittance, which should be taken into account for post-processing or data interpretation.

SEM-grids are elements that determine the resolution limit of the system (together with the slit’s step size). Increasing the number of wires will not only provide a higher resolution, but also would increase the cost, due mainly, to the need of an electronic channel per individual wire, plus the fact that it would make the grid more complex and an increased chance to one-wire-failure, making this option difficult to consider. Therefore, the solution to this problem is to employ stepper motors of high precision. Having now a grid that moves will artificially fill the gap between two wires and therefore the resolution will relay on the wire diameter (typically 30 to 40 $\mu$m). The mayor drawback of this method is that the time needed for a full scan will take several minutes, since for every slit position, the grid must also move to align its center to match the slit, and wait for a new beam pulse to pass through its wires in order to acquire a new measurement before moving to the next slit position.

### 2.1.3 Stepper Motors

Three steppers motors are in charged of the movement of the slit and the SEM-grids. They will require a high reliability and resistance to fatigue with the minimum of precision loses while maintaining a good range of the smaller step size possible to provide the necessary resolution for the slit and grids. The motors (Fig. 2.5) are Phytron ZSH88/2 (data sheet in appendix A).

Figure 2.5: Stepper motor manufactured by Phytron, model ZSH88/2.
2.1. EMITTANCE SCANNER

2.1.4 Acquisition system and associated electronics

Figure 2.6 shows a schematic diagram of the complete hardware system. The modules are contained in a VME (Versa Module Europa) crate: computer (FEC), two ADCs modules, the power modules (VMOD, VHQ202M).

All the components of the system communicate via Ethernet to a server, “Front End Software Architecture” (FESA), were all the input and output data of a device is defined within a C++ class with a set of properties and values. A communication protocol “CERN Middleware” (CMW) is the middle layer between the program that will be designed for the control of the scanner, and the server that provides a communication through a secure and isolated network “Technical Network” (TN) at CERN. This allows the control of the scanner to be done from any console connected to the technical network.

The Bias devices are in charged of suppling a steady voltage level higher than zero, which is understood by the scanner components as an artificial zero level. This is necessary due to power requirements of some of the devices.
that cannot change from positive to negative voltage around the real zero, since the lacking of power at this level could not make them operate properly.

The 250 kS/s (kilo samples per second), 16 bits resolution, 36 channel, simultaneous sampling, ADC cards are intended to be used for the SEM-grids. The cards are equipped with both VME32 and serial interface, so it can be used outside the VME crate if needed. The cards are organized as 6 ADC units, 6 channels each. The total number of independent ADCs is therefore 36. The sampling clock frequency can be adjusted using one of the card registers. The cards has three trigger sources, two external hardware signals and one controlled by software. More details about these homemade ADC cards can be found in the appendix B.

The PLC employed is a Simatic CPU-315-2 from Siemens (see appendix C for manufacturer data sheet). It has a CPU with integrated 24 V DC power supply, 64 kB working RAM and expandable FEPROM, master/slave functionality and LabVIEW compatible.

Amplifiers are needed to boost the signals from the SEM-grids to the acquisition system, which is separated by a distance of about 200 m. There are two mother boards with 32 channels each in a chassis, a few meter away from the SEM-grids. The amplifier design is based on commercial linear amplifier (commercial components identification in naturally radiation hardened technologies) families: AD8610ARZ, AD8032ARZ and AD8397ARZ.

2.2 Design of the control system

In this section, it is explained how the Linac4 LEM control design was done. For this purpose the project will be break down into different parts or modules that build the so-called “programming architecture” and the most appropriated programming style for our requirements will be selected. For aiming a better understanding of the process, in appendix D is described the main programming elements and patterns for those readers who are not familiar with LabVIEW.

2.2.1 Selection of a Programming Style, Actor Framework for Linac4

It is paramount to know the requirements of the project that will lead us to the selection of our programming style and therefore to justify the deci-
2.2. DESIGN OF THE CONTROL SYSTEM

sion made. At the time of selecting one architecture with respect to others, some key features will lead throughout this process. The considerations and requirements are:

1. The project is intended to last long: meaning that, along many years, numerous features and modification can be requested, making the project grows in different directions.

2. It should facilitate the maintenance by anyone: at some point, the maintenance of the project could be relegate to another programmer.

3. Documented, easy to understand: to relegate, to learn and share experience with others.

4. Robust: the additions and modifications of the project should not affects its other parts nor its performance.

This characteristics match with Object-Oriented and Actor Framework design patterns. Both are modular and easy scalable, properly documented would make it easy to understand and relegate to other programmer. Even though both are robust pattern, it could be said that the tools that provides the Actor Framework facilitate the creation of the messaging system between modules, ensures their concurrency execution, and settle down a set of general functions that all modules will have (initialization, stop, threat errors, etc.). Thus, and taking advantage of the newly introduced official training in Actor Framework at CERN, the selected design for this project will make use of the Actor Framework.

2.2.2 Design of the architecture

Once we have decided our programming style as Actor Framework, we will build the concept of our software architecture by splitting the project into different modules (actors) and define how they communicate. Later, we will enter in the design of the modules and their overall logic.

A project in Actor framework implies having two architectures: actor architecture (that we call modules), and class architecture (inheritance, as any other class-oriented language).

- The actor architecture allows us to split the project at will and have as many actors (modules) as we wish, having them handling different tasks. The communication between actors is done by messages. Usually
the communication is defined by the terms “caller” and “nested”, and it is always done between callers and nesteds, it is up to the programmer to define who is the caller of whom, and who is the nested of whom. In Actor Framework, one actor can launch one or more actors, and those actors can as well launch more actors. So, in the point of view of a given actor, we refer to the previous actor who launch him as his “caller” (only one caller per actor), and the actor(s) that it launches are its nested(s) actor(s) (can be any number). The term “Launch an actor” means that the actor is not initially in execution until another actor launches it. Once an actor is launched, he will be permanently in execution doing any programmed routine of functions and waiting for messages of his caller actor or nested(s) actor(s), to which it responds by doing a action (executing a certain method, answering with any internal data, passing the message to another actor to which it is related, etc.).

- The class architecture simply defines the inheritance of our actors, the things that a given actor can inherit from other that have similar functionalities and allow us to use the same functionalities (methods) and type of data (class private data) without the need of programming a same block of code again.

By default Actor Framework already gives all the actors a set of inherited methods, they are:

- Actor Core: is the heart of one actor. When the actor is running, it stays idle on this status waiting a message from any other actor (usually its caller). Once the message is received, it will execute a corresponding method, usually one that we created individually for the actor, but could be as well one of the already defined ones by the framework such as Stop Core.

- Pre Launch Init: this method is by agreement the proper place to do the pre-configurations, is the first thing that one actors does when launched by its caller before it goes to the Actor Core and start responding to messages or sending its owns.

- Handle Error: if during execution there is an error in an actor, it will execute this method. So it is the recommended place to thread errors.

- Stop Core: when an actors receives the message to stop, from any other actors or from itself, he will executes this method. It is also the recommended place to clean all opened references and send stop to its nested actors.
2.2. DESIGN OF THE CONTROL SYSTEM

- Other actor methods: to these already defined methods automatically inherited by the actor framework, we will create a set of individual methods for every actor, and with it, the messages that points to these methods, so that others actors can send them the message to request the execution of a code. As an example, an actor that manage different pieces of hardware, receive a message from an actor that measures water levels “water level too low”, the hardware manager actor will then send a message “start pump” to the pump actor who will start pumping water by executing the neccesary method for that.

To create the architecture, decide how many actors we will need, in which task we will focus them, first we will need to know which functionalities must the control system have, the list of request given by the clients are:

- The use must be able to select two different type of scans, one where they enter all the scan parameters manually (Manual Inputs mode), and other where they select a previously scan log file which includes the settings used for that scan and we execute the scan with the same parameters (File Inputs mode). The user must be able to see on the interface two buttons to access both scan modes.

- In the Manual Inputs scan mode, we need to display to the user a menu with the list of inputs for a manual scan and the user must be able to edit them: initial position of the slit, final position of slit, step size, k off-set factor between slit-grid’s center, number of measurement in every acquisition. Also the clients wish to see and modify some advance settings as the ADC sampling settings, or ADC and motor timeouts.

- In the File Inputs mode, we need to prompt the use with a file browser window so they can search on the disk previously realized scan logs, and when they select one, we do the scan with the same parameters used in that previous scan.

- For the outputs and visualization, the clients wish to be presented three different graphs: phase-space-surface graph, AverageIntensity-Wires Graph and Intensity-Time Graph.

- The clients also wish to log the measurements and parameters used for the scan into a log file on the disk, following their format indications (a .csv file following a naming convention and determined rows-column structure necessary for other tools that will use our generate log files as source at CERN).
Would be also necessary then, to add to the Manual Inputs mode, three parameters that the clients wish to modify to make the wires graphs display selected information, two of the parameters will be the start and finish time to average the intensity value on a wire to avoid considering background noise and secondary emission effects produce in the last moments of the intensity, as well a the selection of which wire information they wish to show on the graph.

To determine our actors, we thought that a good approach would be to split them, in part, by the hardware, we assign one actor per hardware module. This will give us two actors as a starting point: the “Motor Actor” and the “Acquisition Actor”. So far, we will not enter in the design of the logic, just think about them in general and abstract concepts.

Now, we need to translate the inputs given by the user on the interface of our program since we would need to generate a full list of position for the motor according to our start position, final position and step size. We also need to know when to move the motor, and when to do a measurement. It cannot be unsynchronized, must be done when the hardware is ready, the motor are place in the correct position, etc. So we create an actor on top of the Motor Actor and Acquisition Actor, that will receive the user inputs, generate a list of tasks with the necessary steps to do and acquisitions to realize, and this actor will send the necessary commands (messages) to the motor and acquisition when necessary, we will call this actor Hardware Manager Actor.

An interface or GUI (Graphical User Interface), would be necessary, to interact with the user to receive the inputs, and show the acquired data on the graphs. We will define an actor to only be in charge of the interface control named “GUI Actor”.

Another requirement is to log the data in spreadsheet file (.csv). We will probably need some more extra data logs apart of the spreadsheet requested by the clients as error loggers, temporal data, etc. For this we define an actor to deal only with hard disk operations, “Logger Actor”.

For the class architecture, in addition to the inherited default Actor Framework methods, we will find convenient to define new behaviors (new set of methods) that several actors of our project will benefit from. Since all our actors use a protocol to communicate to the hardware via CERN’s FESA server, we could define a common set of methods so that all actors could in-
herit a set of FESA communication capabilities (denominated GETs, SETs, Subscribe).

We will create another set of inherited methods by creating a “Base Actor”. It will let us load our actor’s front panels into the GUI Actor (i.e. on the user interface), and will also let all actors to have the capability to send data directly to the GUI Actor and the Logger Actor. This is a jump over the actor architecture’s tree, but helps in performance and makes the messages much easier to be sent to the GUI and Logger Actors since most of the other actors will need to message them, and without it, a message would require to pass trough several actors until it arrives to the destination.

2.2.3 Design of the modules

Once the main architecture and actors are define, it is created the logic of every actor, their methods and messages. Not entering now in proper code description due to the wide range of LabVIEW knowledge that would be required, but rather it will be given a description based on the flowcharts

![Motor actor flowchart](image)

Figure 2.7: Motor actor flowchart.
that represent their execution.

**Motor Actor.** Its flowchart is shown in Fig. 2.7. Motor Actor will wait for an order from its caller (Hardware Manager Actor). As a starting point, it needs to receive the scan inputs to calculate the number of steps and their lengths. After this, it can receive either “Do Next Step” to move to the next position, “Do Previous Step” in case that its caller has detected some issue, or “Reset”, that can help in cases the motor reaches and unknown status e.g. get out of synchronization. Internally the motor differentiate 4 types of steps. It is necessary to be programmed in this way because the settings for a normal step are different that for the first step of a sequence, which requires special first settings. As an example, the init position internally moves to motor to an ‘end position’ to solve stress issues of the motor components.

**Acquisition Actor.** Its flowchart its shown in Fig. 2.8, the logic of is behavior is as follow: it will need a pre-configuration based on the parameters of the scan, so it will require a first mandatory message with this information. After that, it will wait the “Acquire order” message to start acquiring data, it will set up the ADC modules and subscribe to the server property where the measurements coming out of the scanner are sent. Every time a new data is put in the server, the actor will increment its counter of the number of measurements, process the data and send it to the other actors (to plot it, and to log it on the hard disk) and if the number of measurements is equal to the desired one entered by the user on interface, it will send its caller a message telling the task is finished, and will it wait again for the next “Acquire order”.

**Hardware Manager Actor.** Its flowchart is shown in Fig. 2.9, the hardware manager actor will wait for messages from its caller (GUI Actor) sent when the user click on the Star Scan, Pause/Restart or Abort button on the front panel. For every of these buttons he will react according to its flowchart. Moreover, it will also receive messages from its nested actors (Motor Actor and Acquisition Actor). So when the users click Start Scan button on the interface, it will receive the message “New Scan” and generate a list of tasks with every order it needs to send to the motor and to the acquisition to do a full scan. Every time an message (order) is sent to the motor or acquisition, the hardware manager will wait for it to finish the task before he send the next order to the correspondent actor, and also will check the status of the feedback, if it was not successful, he can apply logic to retry the last task, to restart the scan, to warn the user, etc. Additionally, in any moment the user will be able to click on the front panel the Pause/Restart and Abort buttons, the GUI actor will be in charge of notify about this to the hardware manager.
2.2. DESIGN OF THE CONTROL SYSTEM

Figure 2.8: Acquisition actor flowchart.
in the form of a message, so that the hardware manager can do the necessary actions: Pause/restart is done internally in the HW Manager, while the “Abort” will require him to notify the Motor and Acquisition Actors so they can start their abort routines.

**GUI Actor.** This actor will be in charge of the user interaction, where all controls, indicators and graphs are displayed. Is the one that includes the graphical design, that requires effort to improve the user experience aiming for a satisfactory layout of the elements, meaningful elements and labels, adapted to the user requirements and to the constant changes as the project develops or the users provide feedback and new features to add. Its flowchart is shown in Fig. 2.10, for the design of its logic, two main routines are differentiated: one waiting for user actions, and other waiting for actors messages. The one waiting for user actions (Fig. 2.10-top) will act as a communication channel “user to actors”. When the user click one of the triggering actions buttons, the GUI Actor will grab the necessary data from the front panel controls and send them to the corresponding actors via messages. The other routine (Fig. 2.10-bottom) is the one that waits for messages from the others modules (actors) that came with data to be displayed to the user on the user interface, acting as a communication channel “actors to user”. As an example of this routine, the Acquisition actor will send the message “Acquired data”
to the GUI, this message also carries the processed data that the Acquisition actor has measured and processes ready to be displayed on the graph, GUI Actor will receive this message and executes its method to included the data on the graph that is shown to the user on the interface.

Figure 2.10: GUI actor flowcharts, one reacts to user actions on the front panel (top) while the other does for another actors messages (bottom).
Logger Actor. Its flowchart is shown in Fig. 2.11, it is in charge of managing all hard disk and files operations. When the program is opened, it loads the local path specified in the project requirements where clients wish the files to be placed, and creates a temporally text file to log any possible error message between the opening of the program and the start of a scan. Right after, it will load the last logged scan parameters and send it to the GUI actor via message, so that these parameters can be displayed on the front panel to the user as default values in the controls, instead of having empty-values controls.

![Logger Actor Flowchart](image)

Figure 2.11: Logger Actor Flowchart.

Once this is done, and it could take less than a second, the logger actor is ready to starts its routine, waiting to receive a message from any other actor. It can receives three different messages that we can explain without too much extension. “Start new scan” will makes the Logger actor to create three files
for the current scan with the required name format specified by the clients; the first file contains the scan parameters (.ini); the second file is the one used to save the acquired data during the scan as spreadsheet (.csv); Finally, the third file is to log text messages (.txt). It will also generate a header for the .csv file with the scan parameters information. The other two messages, when received, it saves the internal message’s data in the corresponding file: in the .csv if the message was “Log acquired data” that is mean to be sent by Acquisition actor after processing the data, and in the .txt file if it was the “Log message” that is sent by any actor that gives meaningful information about their states and also, the message is re-sent to GUI actor so the user can be inform on the interface about internal status of the process going on.
Chapter 3
Tests and experimental results

In this chapter some of the software tests and also experimen-
tal results performed to assess the functionality of the emittance
scanner are presented. In-beam test has been performed without
off-set between the slits and grids positions.

3.1 Test on a server simulator

The purpose of this test is to run the first versions of the control system made
on LabVIEW for this project with a simulated server that mimic the scanner
hardware components without needing to use the real one. With this, any
possible mechanical damage of first test’s failures could be avoided, plus it
is a method to save time, since obtaining beam time is difficult. In order to
run a code of this characteristics, based on server communication, it is neces-
sary a method to simulate the data in the direction program-server-scanner
and scanner-server-program. It is needed then, to create a simulation of the
server that will intermediate by receiving our program inputs, and at the
same time, it will simulate signals (outputs) coming front the scanner that
will be read by our program. Without being connected to the server (real or
simulated), the testing of the LabVIEW code would be impossible, since it
requires server responses from the very beginning of the code.

For the simulation of the server (FESA server) it will be used the tool
CMWserver by MTA Team, and create a simulation of the parameters that
are needed for the communication in both directions. The first step is to
match every one of the parameters with the same name and data-type as the
one used to communicate with the server. Second, it is needed to create for
every property a mechanism in the simulator to reacts to the LabVIEW pro-
CHAPTER 3. TESTS AND EXPERIMENTAL RESULTS

Figure 3.1: Example of the simulation mechanism where every time the LabVIEW program command the motor to move, it simulates the changing status and time response that the motor requires to send back the information through the server.

program requests as it would be the real scanner. Sometimes the program will set a request in the server to which, the server automatically responds (as could be getting the status of one property); Some other times, the LabVIEW program will set parameters on the server, but this time the hardware will require a certain time to produce a response back through the server, as is the case when we turn on the ADC module. So every property has been designed with different trigger mechanism and time of response according to the time required by the real scanner to produce those same responses. Figure 3.1 shows an example of the simulator when LabVIEW commands the motor to move.

With this simulator it was able to run the program, and thus to test every one of the actors working and communicating each other, accomplishing with the flowchart structure designed for this project. Furthermore, this non-realistic measurements randomly generated, sometimes helped to find errors that were not expected to happen when all signals and outputs are supposedly ideal.

3.2 Test on the scanner without beam

Once the project has been proven to run successfully in the simulated server, i.e. all the actors work accordingly to the designed flowcharts, the next step is to make a dry run (no beam involved) on the emittance scanner. Even though the current program can be running flawlessly, working with real devices and real server will introduce potential situations of errors. Hence, the way to start testing with the real scanner, and it is a recommendable practice to do so, is by doing an unit test in every possible module. Figure 3.2 shows the different testing phases needed to assure the full functionality of the scanner.
First unit test was performed on the Motor Actor. Since the program did not run through any other actor, no even GUI Actor, in the Actor Core method of the Motor Actor was implemented a simple user interface to pass some step parameters following the Even Handler pattern. Also, it was helpful for the test to constantly show on the interface all the internal data of the Motor Actor. This did speed up the test phase considerably, it allowed to see the state of the motor and its parameters during operation. During this first testing phase, numerous problems and errors were identified, these errors found during the testing phases and how to handle them, will be discussed later in the next section.

The second unit test was performed on the Acquisition Actor. Nonetheless, due to the controller of the ADC modules, it was not possible to set any of their control parameters because it detected that the grids were out of position, not even background noise or communication could be obtained.

Next test, the third one, focused in the integration of the Hardware Manager Actor with the Motor Actor by performing a full scan without acquisition, just the corresponding steps being commanded by the hardware manager.

The fourth test reintroduced the Acquisition Actor. This time being able to do several full scans where the Hardware Manager Actor was commanding both the Motor Actor and the Acquisition Actor via messages. After this test was successfully performed, several test with beam were made.

The fifth test does not correspond to a properly test with the scanner, but
rather with the execution of the program inside the network were the final project is going to be executed. It was a simple test to ensure that the local path where files are logged correspond to the correct ones.

The sixth test aim to do a full scan and log it on the hard disk.

Finally, the seventh test was one were the GUI was implemented according to the needs of the clients.

3.3 Debugging and Handling errors

A continuation, it is describe the treatment of the problems found during the above mentioned tests as well as their solutions.

**Motor position bouncing**, status not constant. One of the first problems noticed, was that the Motor Actor after setting the parameters to move to one determined position, it continuously checked the change of position over time waiting to reach the position of destination. When it detected that the motor stopped at the destination position, it was programmed to automatically tell the Hardware Manager Actor that its task was accomplished; which made the Hardware Manager Actor sent the next order to the needed actor (Motor or Acquisition) to continue with the scan. This situation started to produce errors occasionally, because the motor was not completely stopped when it reached the destination position, it was microscopically bouncing around the destination value, and because of this, the driver of the hardware refused our next commands (the status was flagged as busy instead of ready). The problem was easy to resolve, the solution was to check several times the position of the motor after reaching its destination. But this has the drawback of making the system more slow.

**Missing steps.** Another problem was that the steppers motor’s driver could fail if a sequence of steps was maintained for a long time at a fast frequency, making a step command to not to be executed read nor executed. The problem was resolved by self-checking current position when setting the destination position parameters and defining a timeout. If the time of the timeout occurs before the motor reach the position, it is understood that the step fails or is drop, and thus, it is retried once more. If it fails again the Motor actor will notify the Hardware Manager actor, where a new logic of decision-making can applied, as could be to decide to send a reset to the motor and retry the operation.
3.3. DEBUGGING AND HANDLING ERRORS

Motor unknown status, not responding and not resetting. After some support from the team in knowledge of the information related to the motor, we were informed that if this happens, there exist another hard-reset function. So it could be implemented it.

Missing acquisition measurements (shots). When the Acquisition Actor was sent the message to start acquiring, it is supposed to measure the beam a certain number of times defined by the user on the user interface as a parameter of the scanning session. When the Acquisition gets the number of measures indicated by the user, it informs back the HW manager to do the next action. The problem was that the Acquisition actor was never contacting back the HW Manager because it never was getting the number of required measurements. This happened because the gap of time between the setting of the parameters to start the acquisition and the start of the measurement, was enough to lose some measures. It was resolved by going to the waiting for measure status before even setting the acquisition on.

Minor errors during development phase. During the programming phase, it is common the take small part of code that does not require neither hardware nor server communication to execute and test them. Here it was identified numerous errors, and some of them even produced with one idea on mind to trigger another action or see how the code behave to them: to notify the user, to log the errors, to propagate the abort message to the other actors, etc.

Handling future and unknown errors. Apart of the already known errors, it is necessary to prepare our program for when new errors occurs that were not found before. For this purpose, Actor Framework includes one method in every actor called “Handle Error” that will execute every time that and error is produced. So we can filter the already known errors by the error number associated with them, and executed the appropriate actions to resolve them. But for the ones we do not know the number and thus we are not filtering, we will create a generic error logged via server and in file, display the error to the user, and for every actor, apply a general mechanism to try resolve it in general way (as could be a restart or repeat again after a delay of time) while in the mean time we study the error in deep and apply an individual focused solution for this error, making our program more and more robust.
3.4 In-beam measurements

Once the program reached a stabled version, several tests were performed with beam. The beam is given by the team that works with the particle source. They set up the source in different configurations that generates different beam’s characteristics every time, and with it, different emittances. The LabVIEW program perform the measurement that will let them know the quality (emittance) of a beam for a certain set of source’s settings.

A continuation it is shown the measurement obtained with a beam for a scan starting at 35 mm and finishing at $-35$ mm (spatial coordinates), with step size of 7 mm (slit displacement), 10 measurements on every step (i.e. per acquisition), and fixed position of the SEM-grids, yielding the following results once the data has been processed. It has to be mentioned that the data analysis of the emittance measurement (how good is the emittance, distribution of beam intensity, etc) is out of the score of this project and it will be included in the next phase of this continuous growing project.

Figure 3.3 shows the phase-space-surface graph, a 2D representation of a system $(x, x')$ where the intensity values are shown in color scale (left-hand side). It usually adopts the shape of a drifted line (or narrow ellipse) with maximum intensity in its center. The horizontal axis matches the grid wires position.

Figure 3.3: Phase-space-surface graph of the beam intensity. The horizontal axis matches the grid wire position.
3.4. IN-BEAM MEASUREMENTS

Figure 3.4: Average intensity of the beam (vertical axis) in every one of the wires (horizontal axis) after executing a full scan.

Figure 3.4 shows the average intensity per wire graph, where it can be observed how the intensity of a beam passing through the grids is represented by a plot. The wires from 0 to 19 and from 21 to 47 have a low value of the intensity while center wires (i.e. 20 to 30) show the highest values. It can be noticed, that the peaks of highest intensity on this graph are in the same position as the phase-space-surface graph. In this measurement concretely, we see that the five peaks of the average intensity per wire graph (Fig. 3.3) match the five strongest intensity points on the phase-space-surface graph.

The intensity over time graph is shown in Fig. 3.5, where the user can select a wire and get a plot with the intensity over time for every one of the measurements that the wire does in a scan task. As an example, let's imagine that we select the wire 25, which is close to the middle of the grid and will always receive a high intensity for every slit position. Now, every time a acquisition is taken, we will generate a plot where it can been seen how the intensity has grown over time: starting at the moment that the particles hits the wire, then the intensity keeps growing as the beam's particle travels on the wire's internal structure, and finally the intensity decreases when the particles stops (or escapes the wire).
Figure 3.5: Intensity (vertical axis) over time (horizontal axis) graph for a selected wire.
Chapter 4
Conclusions and future work

The decision of choosing LabVIEW as the development environment for the control system of the Linac4 Emittance Scanner has been proved to be a successful one. It has been achieved a modular and flexible architecture that for a project of such magnitude will suit perfectly the expected lifetime, while being capable of to adapt to future’s requirements and needs.

In-beam tests have shown the functionality of the Linac4 emittance scanner, proving not only an excellent performance but also has reaffirm itself as a indispensable tool for the beam emittance analysis.

During the creation of the project, feedback about the functionalities of the system have been received once the first versions of the control system were released. Besides, new features and request have been made by the clients for the future development of the application:

1. Documentation of the project: as any other project, it will include documentation (for developers and users of the application, a wiki with main aspects of the project, contacts, etc.) as well as, an user manual integrated on the user interface, where every aspect of the execution of the software, scan parameters, elements of the interface, and so on, will be detailed and explained to the user.

2. Scan sequencer: a new module that will allow the user to create a list of different scans to perform in sequence, without needing to wait for one scan to finish before adding new parameters.

3. Plot to show motor positions and status over time: a view of how the positions of the motors changes over time, the status of different parts of the system, etc.
4. Method to exclude non-interest particles from measurements: different approaches to this have been thought by the clients, they wish to have a tool where they can set a point of reference from which values of intensity below of it could be automatically exclude but at the same time should tell if this value was real background noise, or just a low intensity region of the beam. More details will be discussed to the clients when they decide the final approach to deal with this goal.

5. Re-programming of the server class: The server class used to communicate with the hardware have some limitations and it is not longer maintained, some changes would be necessary for future developments.

A final conclusion of this project is that, within the CERN department that I work, which is exclusively dedicated to LabVIEW applications, a person with initially not deep knowledge of LabVIEW can achieve, in not too much time, the necessary level required to design a control system using one of the most new and advance programming styles in LabVIEW as it is the Actor Framework.
Bibliography


List of Figures

1.1 Top, CERN complex with an overview of the main components-experiments, type of beams and remarked dates. Bottom, CERN aerial view showing the main experiments and villages/towns of the proximity. .......................... 10
1.2 Beam velocity vector \( \vec{V} \) compared with a portion of the velocity vectors of the particles travelling inside the beam. .............. 11
1.3 Ellipse of the beam as defined by the Twiss or Courant-Snyder parameters, where each of them represent the distance respect the origin \([3]\). ................................................................. 12
1.4 Project Explorer of a LabVIEW project (left). Front Panel shown to the final user (right). ................................. 13
1.5 LabVIEW’s block diagram showing two basic mathematical nodes (left) and Functions Palette window with programming functions provides by LabVIEW (right). ....................... 14

2.1 3D model representation of the emittance scanner. ............. 16
2.2 Schematic of the beam passing through the slit and the scattering effect it produces as view by the profile monitor (SEM-grid) \([5]\). ................................................................. 16
2.3 Linac4 two-slits installation on the same plate, for horizontal and vertical phase-space beam analysis. ......................... 17
2.4 Close-up of one of the two SEM-grids (left). Full view of the structure where grids are mounted (right) \([6]\). ................. 17
2.5 Stepper motor manufactured by Phytron, model ZSH88/2. ... 18
2.6 Schematic diagram of the hardware installation at Linac4 emittance scanner. The Ethernet connexion provides the link between the scanner and the software infrastructure. ........... 19
2.7 Motor actor flowchart. ............................................. 25
2.8 Acquisition actor flowchart. ..................................... 27
2.9 Hardware manager actor flowchart. .............................. 28
2.10 GUI actor flowcharts, one reacts to user actions on the front panel (top) while the other does for another actors messages (bottom) ........................................... 29

2.11 Logger Actor Flowchart ........................................... 30

3.1 Example of the simulation mechanism where every time the LabVIEW program command the motor to move, it simulates the changing status and time response that the motor requires to send back the information trough the server .................. 34

3.2 Evolution of the testing phase of the Linac4 LEM project .... 35

3.3 Phase-space-surface graph of the beam intensity. The horizontal axis matches the grid wire position .......................... 38

3.4 Average intensity of the beam (vertical axis) in every one of the wires (horizontal axis) after executing a full scan ............ 39

3.5 Intensity (vertical axis) over time (horizontal axis) graph for a selected wire ...................................................... 40

D.1 A For Loop structure in LabVIEW (left), a flowchart of its mechanism (middle) and its equivalence in test-based programming languages (right) ........................................... 65

D.2 (1) Block Diagram of a VI calling another VI, which received the term SubVI. (2) Conector pane and icon of the SubVI. (3) Block Diagram of the SubVI ........................................... 66

D.3 State machine in LabVIEW, its different states grants the dataflow of a certain degree of flexibility and modularity ........ 67

D.4 Event Handler pattern or event-driven programming style in LabVIEW. The basic pattern to interact with user actions .... 68

D.5 Producer-Consumer Pattern in LabVIEW, where two part of the code executes in parallel at different rates sharing data ... 69

D.6 (1) Project files of a OOP LabVIEW program. (2) Internal private data of the three classes in the project. (3) Class Hierarchy architecture, based in one parent and two children .......... 70

D.7 Example of a code using OOP in LabVIEW, that executes the method “Fee Salary stats” of the class “Worker” that given a certain number of inputs, will calculate the the amount of salary hold by the state ........................................... 70
E.1 User Interface that is displayed to the user for the Linac4 Linear Emittance Meter. At the left are placed two menus, one static with the main options of the control program, and the one in the right is a dynamic one which content changes according the option selected on the left-most one. Immediately at the right of both menus, is it placed the graph to visualize the measurements of the beam.
Appendices
Appendix A

Stepper motor data sheet
ZSH Stepper Motor
Robust. Powerful. Reliable.

Phytron’s HARSH-Environment motors are particularly suitable for challenging applications in mechanical engineering and industry. Challenging conditions are solved with the precise running performance, high torque and the motor’s robust design for environments up to IP68. Inside climate chambers, adjusting the paper thickness in paper machines, adjusting rotor blades in aviation or directly in a fuel tank. In those and other environments Phytron’s stepper motors and controllers for HARSH Environments provide precise and reliable work.

Highlights

IP68

The ZSH stepper motor convinces with its robust housing with high-strength cable gland. The motor is waterproof up to 10 m with the IP68 option.

Extended Temperature Range
The ZSH stepper motor not only convinces with a very balanced, smooth and low resonance running performance with maximum positioning accuracy, but also with the optional extended ambient temperature range of -30 to +80 °C (briefly up to 100 °C).

Overview: Extensions
- Stepper motor with brake: Permanent magnet brake for 24 V DC supply
- Stepper motor with encoder: Resolution 50, 200 or 500 lines, 2- or 3-channels
- Stepper motor with encoder and motor brake
- Stepper motor with low-backlash planetary gear: 1-, 2- or 3-stages, Reduction ratios from 3:1 to 512:1

Options

In Focus

- 2-phase hybrid stepper motor
- Number of steps: 200 / step angle: 1.8°
- Standard version: 4-lead, parallel windings, with terminal box
- Holding torques from 0.45 to 17 Nm
- Protection class: IP 54, optional: IP 68
- Perm. ambient temperature: -30 to +50 °C (optional: +80 °C) (up to 100 °C for short time)
- Design voltage: 250 V AC acc. to EN 60034
- Insulation class F acc. to VDE 0530
- Test voltage: 1800 V AC (1 sec)
- High permissible axial and radial bearing loads
- Step accuracy: ±3 % [ref. to 1.8° step angle, not cumulative]
- Optional:
  - 2nd shaft (IP 41)
  - Free wire ends (IP 41)
  - Different types of flange and shaft (mm or inch)
  - Motor brake
  - Encoder
  - Low-backlash planetary gear

Edition 2016 July

www.phytron.eu/ZSH ENG
# Dimensions Stepper motor ZSH 57 to ZSH 107 / Key / Flange / Shaft

### Harsh ZSH – data sheet

**Option:** ZSH Stepper motor with free wire ends

<table>
<thead>
<tr>
<th>Stepper motor</th>
<th>Dimensions</th>
<th>Key</th>
<th>Flange and shaft:</th>
<th>Flange and shaft:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mm</td>
<td>mm</td>
<td>mm (inch)</td>
<td>mm (inch)</td>
</tr>
<tr>
<td><strong>ZSH 57/1</strong></td>
<td>56.5</td>
<td>108</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 57/2</strong></td>
<td>56.5</td>
<td>108</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 57/3</strong></td>
<td>56.5</td>
<td>108</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 67/1</strong></td>
<td>68.5</td>
<td>145</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 67/2</strong></td>
<td>68.5</td>
<td>145</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 67/3</strong></td>
<td>68.5</td>
<td>145</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 87/1</strong></td>
<td>86.5</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 87/2</strong></td>
<td>86.5</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 87/3</strong></td>
<td>86.5</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 88/1</strong></td>
<td>86.5</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 88/2</strong></td>
<td>86.5</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 88/3</strong></td>
<td>86.5</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 107/1</strong></td>
<td>108</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 107/2</strong></td>
<td>108</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 107/3</strong></td>
<td>108</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td><strong>ZSH 107/4</strong></td>
<td>108</td>
<td>180</td>
<td>1.5</td>
<td>70</td>
</tr>
</tbody>
</table>

**Note:** Required space for terminal box cover fixing screws: up to 2 mm

---

www.phytron.eu

Edition 2016 July / DS-094-A003 EN / 2

---

**Blue** = standard version

1 Optional for size 57: Woodruff key 2x2.6 DIN 6888
### Mechanical and Electrical Characteristics ZSH 57 to ZSH 107

<table>
<thead>
<tr>
<th>Stepper motor type</th>
<th>Phase current bipolar</th>
<th>Phase current unipolar</th>
<th>Resistance per winding</th>
<th>Inductivity per winding</th>
<th>Holding torque</th>
<th>Detent torque</th>
<th>Rotor mass inertia</th>
<th>Permissible bearing load</th>
<th>axial</th>
<th>radial</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZSH 57/1</td>
<td>1.4 / 4.2 / 5.5</td>
<td>2)3)</td>
<td>1.3 / 3.9</td>
<td>0.5 / 0.7 / 0.5</td>
<td>0.45</td>
<td>0.01</td>
<td>0.125</td>
<td>80</td>
<td>150</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>ZSH 57/2</td>
<td>2.1 / 2.8 / 4.2</td>
<td>2)3)</td>
<td>1.5 / 2 / 2.6</td>
<td>9 / 5 / 4.6</td>
<td>0.85</td>
<td>0.017</td>
<td>0.25</td>
<td>180</td>
<td>280</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>ZSH 57/3</td>
<td>2.1 / 4.2 / 5.5</td>
<td>2)3)</td>
<td>1.5 / 3 / 4.4</td>
<td>9 / 3 / 1.2</td>
<td>1.25</td>
<td>0.025</td>
<td>0.375</td>
<td>80</td>
<td>150</td>
<td>1.35</td>
<td></td>
</tr>
<tr>
<td>ZSH 87/1</td>
<td>2.3 / 4.2 / 7</td>
<td>2)3)</td>
<td>1.6 / 3 / 5</td>
<td>0 / 0.8 / 0.3</td>
<td>0.026</td>
<td>0.026</td>
<td>0.65</td>
<td>180</td>
<td>280</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>ZSH 87/2</td>
<td>5 / 6.5 / 8.4</td>
<td>2)3)</td>
<td>3.5 / 6 / 8</td>
<td>1.1 / 0.5 / 0.4</td>
<td>3.6</td>
<td>0.05</td>
<td>1.3</td>
<td>180</td>
<td>280</td>
<td>2.65</td>
<td></td>
</tr>
<tr>
<td>ZSH 87/3</td>
<td>5 / 8.4 / 10</td>
<td>2)3)</td>
<td>3.5 / 6 / 7</td>
<td>1.1 / 0.5 / 0.4</td>
<td>5.4</td>
<td>0.08</td>
<td>1.95</td>
<td>180</td>
<td>280</td>
<td>3.65</td>
<td></td>
</tr>
<tr>
<td>ZSH 88/1(1)</td>
<td>2 / 4 / 8</td>
<td></td>
<td></td>
<td>1.8 / 0.5 / 0.13</td>
<td>0.042</td>
<td>0.042</td>
<td>1.35</td>
<td>180</td>
<td>280</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>ZSH 88/2</td>
<td>2 / 4 / 8</td>
<td></td>
<td></td>
<td>3.61 / 0.74 / 0.25</td>
<td>3.6</td>
<td>0.08</td>
<td>2.7</td>
<td>180</td>
<td>280</td>
<td>2.65</td>
<td></td>
</tr>
<tr>
<td>ZSS 88/3</td>
<td>4 / 8 / 12</td>
<td></td>
<td></td>
<td>1.14 / 0.25 / 0.14</td>
<td>10.9 / 2.6 / 1</td>
<td>9.13</td>
<td>4.05</td>
<td>180</td>
<td>280</td>
<td>3.65</td>
<td></td>
</tr>
<tr>
<td>ZSH 107/1</td>
<td>7 / 8 / 12.5</td>
<td></td>
<td></td>
<td>11.1 / 2.5 / 0.75</td>
<td>3</td>
<td>0.08</td>
<td>1.35</td>
<td>180</td>
<td>280</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>ZSH 107/2</td>
<td>8 / 10 / 12.5</td>
<td></td>
<td></td>
<td>26 / 5.5 / 1.15</td>
<td>6</td>
<td>0.08</td>
<td>2.7</td>
<td>180</td>
<td>280</td>
<td>2.65</td>
<td></td>
</tr>
<tr>
<td>ZSH 107/3</td>
<td>10 / 12.5</td>
<td></td>
<td></td>
<td>10 / 2.6 / 1.15</td>
<td>9</td>
<td>0.21</td>
<td>4.05</td>
<td>180</td>
<td>280</td>
<td>3.65</td>
<td></td>
</tr>
<tr>
<td>ZSH 107/4</td>
<td>12.5</td>
<td></td>
<td></td>
<td>19.9 / 3.1 / 1.15</td>
<td>13</td>
<td>0.3</td>
<td>12</td>
<td>180</td>
<td>280</td>
<td>9.8</td>
<td></td>
</tr>
<tr>
<td>ZSH 107/5</td>
<td>15</td>
<td></td>
<td></td>
<td>39.8 / 5.5 / 1.15</td>
<td>17</td>
<td>0.4</td>
<td>16</td>
<td>180</td>
<td>280</td>
<td>12.5</td>
<td></td>
</tr>
</tbody>
</table>

**Blue – popular types**

1) Size 88 for bipolar operation only
2) Standardwicklung 1/2/3
3) The current value given in the ordering data (e.g. ZSH 107/2.200.8) refers to the bipolar mode (parallel windings).
4) Current in unipolar mode = 0.7 x current in bipolar mode
5) Resistance per phase in bipolar mode = 0.5 x resistance per winding
6) The inductivity values apply for each single winding as well as for two parallel windings.
7) For series mounted windings, the inductivity is multiplied by 4.
The phytron stepper motors type ZSH are built in 4-lead parallel windings (standard).

Alternative windings like 8-lead are available on request:
The motors can be used with unipolar or bipolar control mode, as the windings can be differently connected.
5-lead connection is applicable for the unipolar control mode.
In the bipolar control mode, 4-lead motor wiring is required, windings connected in parallel or in series.
The information in the ZSS motor connection leaflet (delivered with each motor) must be regarded when wiring the motor in order to provide for EMC compliant wiring. The motor connection leaflets are also available by download from the phytron homepage.
Appendix B

ADC card data sheet
frequency can be adjusted using one of the card registers. The card has three trigger sources – 2 external hardware signals and one controlled by software.

**The ADC Card Specification**

| Number of channels     | 36 with flat cable input
|                        | 18 with LEMO/SMA inputs
| Resolution             | 16 bits
| Max sampling rate      | 250kS/s per Channel
|                        | **Simultaneous** sampling of all 36 channels
| Sampling range         | 25MHz/R, where R <100,65535>
| Inputs                 | Differential, Instrumentation amplifier at each input
| Input range            | Selectable (+/-5V, +/-10V), adjustable gain (resistor)
|                        | Input protection up to +/-100V
| Interface              | VME 6U, 32bit DATA, 24 bit ADDR, RS232, RS485
| Programmable logic     | FPGA, 8,000 Logic Cells, CPLD 570LC
| Memory                 | SDRAM 64MB, max 128MB (option)
| Digital I/O            | 2x trigger input (TTL)
|                        | 3x general I/O (TTL)
|                        | RS232C and RS485
| Indicators             | 9 LEDs at the front panel
| Card identification    | 48 bit unique serial number, 8 bit Address switch (address bits 23:16)
|                        | FPGA version
|                        | CPLD version
| FPGA remote programming/configuration | Yes/Yes
| Internal voltage monitoring | Yes
| Embedded operating system / firmware | Yes (optional)
| Input signal connectors | IDC, compatible with MPV901
Appendix C

PLC data sheet
Data sheet

*** SPARE PART*** SIMATIC S7-300, CPU 315-2 DP CPU WITH INTEGRATED 24 V DC POWER SUPPLY, 64 KBYTE WORKING MEMORY 2ND INTERFACE DP-MASTER/SLAVE

<table>
<thead>
<tr>
<th>Supply voltage</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rated value (DC)</td>
<td>24 V</td>
</tr>
<tr>
<td>permissible range, lower limit (DC)</td>
<td>20.4 V</td>
</tr>
<tr>
<td>permissible range, upper limit (DC)</td>
<td>28.8 V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input current</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Current consumption (rated value)</td>
<td>1 000 mA</td>
</tr>
<tr>
<td>Inrush current, typ.</td>
<td>8 A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power loss</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Power loss, typ.</td>
<td>8 W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Work memory</td>
<td></td>
</tr>
<tr>
<td>• integrated</td>
<td>64 kbyte; 64 KB / 21K instructions RAM (integrated)</td>
</tr>
<tr>
<td>Load memory</td>
<td></td>
</tr>
<tr>
<td>• expandable FEPROM</td>
<td>Yes; Flash-EPROM</td>
</tr>
<tr>
<td>• expandable FEPROM, max.</td>
<td>4 Mbyte</td>
</tr>
<tr>
<td>• integrated RAM, max.</td>
<td>96 kbyte</td>
</tr>
<tr>
<td>Backup</td>
<td></td>
</tr>
<tr>
<td>• with battery</td>
<td>Yes; all blocks</td>
</tr>
<tr>
<td>• without battery</td>
<td>Yes; 4 KB: bit memory, counter, times and data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU processing times</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>for bit operations, typ.</td>
<td>0.3 µs</td>
</tr>
<tr>
<td>for bit operations, max.</td>
<td>0.6 µs</td>
</tr>
<tr>
<td>for word operations, typ.</td>
<td>1 µs</td>
</tr>
<tr>
<td>for fixed point arithmetic, typ.</td>
<td>2 µs</td>
</tr>
<tr>
<td>for floating point arithmetic, typ.</td>
<td>50 µs</td>
</tr>
<tr>
<td>for timer/counter operations, typ.</td>
<td>12 µs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU-blocks</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td></td>
</tr>
<tr>
<td>• Number, max.</td>
<td>255</td>
</tr>
<tr>
<td>• Size, max.</td>
<td>16 kbyte</td>
</tr>
<tr>
<td>FB</td>
<td></td>
</tr>
<tr>
<td>• Number, max.</td>
<td>192</td>
</tr>
<tr>
<td>• Size, max.</td>
<td>16 kbyte</td>
</tr>
</tbody>
</table>
### Programming

- **Command set**
  - Binary logic operations, bracketed operations, result allocation, saving, counting, loading, transferring, comparing, shifting, rotating, complementation, calling blocks, fixed point arithmetic, floating point arithmetic, jump functions

- **Nesting levels**
  - 8

- **Program organization**
  - Linear, structured

- **System functions (SFC)**
  - Interrupt and error processing, copy data, clock functions, diagnostic functions, module parameterization, operating mode transitions

### Programming language

- LAD
- FBD
- STL
- SCL
- CFC
- GRAPH
- HiGraph®

### Software libraries

- Process diagnostics
- Software controller

### Know-how protection

- User program protection/password protection
  - Yes

### Cycle time monitoring

- **lower limit**: 1 ms
- **upper limit**: 6 000 ms
- **adjustable**: Yes
- **preset**: 150 ms

### Dimensions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td>80 mm</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>125 mm</td>
</tr>
<tr>
<td><strong>Depth</strong></td>
<td>130 mm</td>
</tr>
</tbody>
</table>

### Weights

- **Weight, approx.**: 530 g; Memory card 16 g

---

**last modified:** 06/28/2017

---

6ES7315-2AF03-0AB0

Page 5/5

© Copyright Siemens
Appendix D

The basic elements in LabVIEW code

Structures: as any other programming languages, there are pieces of code that should execute conditionally, for a certain number of times, until one condition occurs or as a consequence of another event (ex. user action). They are also commonly referred to as “loops” in other programming languages. An example of a ”For Loop” is shown in figure D.1, for more details and others main LabVIEW structures, readers are invited to visit the LabView’s web site [4].

Virtual Instrument (VI): a VI is the resulting file with extension .vi that contains the block diagram and front panel of a LabVIEW program. When this VI is executed, the front panel is shown to the users and they can run and interact with the code. As a note, when compiled, the extension changes to an executable.
SubVIs: a SubVI is nothing more than a VI being called in the form of a node in the Block diagram of another VI. This behavior correspond to a subroutine call in text-based programming languages [7]. Often, when the code in the block diagram grows and get more complex, we can identify parts of the code that executes several times at different places. When this happens, it is common to identify these blocks of code and create SubVIs, this way, allowing re-usability of an algorithm in different places. When a VI is created, and it is going to be called in other VI’s block diagram, i.e. as a SubVI, it will not display its front panel, so the programmer must define how the input and output data is transferred to, and from, the SubVI. This is done by the “connector pane”, where programmers define the number and types of inputs and outputs of a SubVI, allowing them to be connected to the require data (wires) in the block diagram (see Fig. D.2). It is also possible to define and icon to easily identify the function in the block diagram (Fig. D.2 (2)).

Design Patterns and Programming Styles

They are not properly an element, but a way to organize functions and structures in different styles that will determine how the performance and efficiency of a code is, how flexible, the amount of parallelism or serialization, etc. There are infinite ways to accomplish the same goal, but even using the same functions, the pattern or programming style a person follows will end up giving different results that are more appropriate for one project or other. For example, some design patterns benefit from a short time of development, but a more difficult maintenance or flexibility, some others would sacrifice simplicity in favor of a more robust and efficiency code, others
will take the most of the multi-cores CPUs, etc. Some design styles are well-known due to its reliability, easy understanding and probed efficiency after many years being used, and they form one of the basic keys for professional LabVIEW applications [8]. As projects grows in complexity and scale, it is common to use architectures that make use of more than one basic design pattern. We will discuss the main ones that will be necessary to understand our architecture:

**State Machine:** State Machines constitute one of the most fundamental design patterns; they allow decision-making algorithms to be represented in the block diagram as they could be represented too, by a state diagrams or flowchart (see Fig. D.3). State machines can be implemented where different states are distinguishable and a mechanism of decision can determine the next required state or sequence of states to be executed.

![State Machine in LabVIEW](image)

**Figure D.3:** State machine in LabVIEW, its different states grants the dataflow of a certain degree of flexibility and modularity.

**Queue driven state machine:** A more complex state machine that waits for data to be added to a 'queue' to which it reacts by executing one or several states. The queue acts as a buffer of data so if the data is added to the queue faster than the state machine can process it, it will be buffered waiting for the machine to processes it.

**Message handled state machine:** A similar mechanism to the Queue driven state machine, but here the data use to be flagged with an order or command (the message) that reflects how the one that added it to the queue wants it to be processed.

**Event Handler:** Event Handler design pattern is the most common and recommendable way to interact with user actions, being able to execute one
APPENDIX D. THE BASIC ELEMENTS IN LABVIEW CODE

piece of code per event that are wished to react to: click on control buttons, keys pressed on the keyboard, mouse clicks, moving slides, gauges, etc. It is also possible to react before events triggered programmatically (see Fig. D.4). For each event required to react, one case is create on the event list of the event structure, thus, placing a while loop in its outside, allows to re-use the structure and waits for the next event to happen.

![Event Handler pattern in LabVIEW](image)

Figure D.4: Event Handler pattern or event-driven programming style in LabVIEW. The basic pattern to interact with user actions.

**Consumer-Producer and Master-slave Patterns:** The main feature of these patterns is the advantage of parallel execution at different rates that require data sharing between them. Consumer-Produces pattern is commonly used when acquiring multiple sets of data that needs to be processed; as the acquisition of data should not be delayed by the process time, decoupling the two process in two different loops, allows the acquisition (producer) to buffer data that the consumer will process in order when he finish the previous pack of data D.5. It could be said that the producer-consumer pattern is based in Master-Slave pattern [9], but the main difference is that in Master-Slave, the data is not buffered and thus, some packets can be lost and not processed, which also bring advantages for other kinds or purposes such as rendering applications. Master-Slave pattern could also count with a relation 1 to many, where produces-consumer is recommended for 1 to 1 [10].

**Object Oriented Programming in LabVIEW:** Object-oriented programming (OOP) has demonstrated its superiority over procedural programming as an architecture choice in several programming languages as could be C++, Java, Smalltalk, and the list goes on. For this reason, LabVIEW also has include the option to allow OOP design into their users developments. It covers and adapt the OOP techniques and concepts in the same programming environment, where objects are instances defined by classes, that
encapsulate their attributes ("class private data" in LabVIEW) and their methods (as VIs), including all kind of OOP functionalities or features, from inheritance/composition, to polymorphism and dynamic dispatch, etc. [11]. Main advantages of OOP are modularity, robust design, easy maintenance (add rest), etc. Figures D.6 and D.7 show an example of OOP in LabVIEW where an instance of the Child class "Workers" access to its own attributes (class private data) and also, the ones inherited from its Parent in order to execute the method "salary stats.vi".

**Actor Framework:** Actor model is a concurrent computational model based on the idea of "actors" as primitives entities that can receive messages to which they: make local decisions, create more actors, send other messages, and change theirs states. It is inspired by physics, including general relativity and quantum mechanics, and "motivated by the prospect of highly parallel computing machines, consisting of dozens, hundreds, or even thousands of independent microprocessors, each with its own local memory and communications processor, and communicating via a high-performance communications network." (William Clinger, June 1981, ‘Foundations of Actor Semantics’, Mathematics Doctoral Dissertation MIT).

LabVIEW implements this model under a framework that ‘raise’ the Object-Oriented Programming style to the Actor Model. Enabling the creation of actors as classes that run independently with a predefined set of default methods: Initialization, Core (where received messages are processed), stop, error handling, etc. Includes the mechanism to create the necessary messages they will use to communicate with each other, and a number of functions to
Figure D.6: (1) Project files of a OOP LabVIEW program. (2) Internal private data of the three classes in the project. (3) Class Hierarchy architecture, based in one parent and two children.

Figure D.7: Example of a code using OOP in LabVIEW, that executes the method “Fee Salary stats” of the class “Worker” that given a certain number of inputs, will calculate the amount of salary hold by the state.

grant them with the actors principles as: launching other actors, sending messages and executed custom method’s as a message response that could change their own state. The biggest advantage of Actor Framework it is modular, scalable and extensible, allowing to build software architectures for LabVIEW applications requiring a high level of concurrency.
Appendix E

User interface of Linac4 LEM
Figure E.1: User Interface that is displayed to the user for the Linac4 Linear Emittance Meter. At the left are placed two menus, one static with the main options of the control program, and the one in the right is a dynamic one which content changes according the the option selected on the left-most one. Immediately at the right of both menus, is it placed the graph to visualize the measurements of the beam.