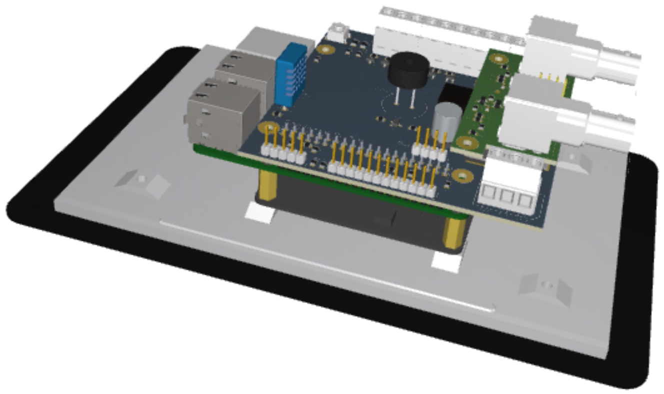
Andoni Pérez Segura is a Telecommunication engineer from San Sebastián, Spain. With this ambitious Master's Thesis he starts a fascinating new research line in the particle accelerators scope and finalizes his MEng.

Andrés María Roldán Aranda is the academic head of the present project, and the student's tutor. He is a professor in the Departament of Electronics and Computers Technologies.

MASTER'S THESIS

Andoni Pérez Segura

TELECOMMUNICATION ENGINEERING



175 MHz Cavity Electromagnetic Simulation & Design of a Particle Accelerator Control System

Andoni Pérez Segura

2022/2023

Tutor: Andrés María Roldán Aranda

“175 MHz Cavity Electromagnetic Simulation & Design of a Particle Accelerator Control System”

Printed in Granada, July 2023.

“175 MHz Cavity Electromagnetic Simulation & Design of a Particle Accelerator Control System”

MASTER IN
TELECOMMUNICATION ENGINEERING

Master's Thesis

"175 MHz Cavity Electromagnetic Simulation & Design of a Particle Accelerator Control System"

ACADEMIC COURSE: 2023

Andoni Pérez Segura

MASTER IN TELECOMMUNICATION ENGINEERING

"175 MHz Cavity Electromagnetic Simulation & Design of a Particle Accelerator Control System"

AUTHOR:

Andoni Pérez Segura

SUPERVISED BY:

Prof. Andrés Roldán Aranda

DEPARTMENT:

Electronics and Computers Technologies

“175 MHz Cavity Electromagnetic Simulation & Design of a Particle Accelerator Control System”

Andoni Pérez Segura

KEYWORDS:

RF, BLAS,Ansys HFSS, Pillbox cavity, EPICS, PCB, Altium Designer® 21, VNA, .

ABSTRACT:

This project aims to simulate the electromagnetic behaviour of an RF cavity and design a control system for a particle accelerator. The cavity to be simulated and the particle accelerator where the control system will be implemented is named BLAS. This accelerator was designed and built through a collaboration between BTESA and CIEMAT.

On one hand, the simulation will be performed using Ansys HFSS software. Different simulations will be conducted by modifying the cavity geometry to study its electromagnetic behaviour. On the other hand, EPICS software will be utilized for the control system design. EPICS is widely used in the field of particle accelerators as it enables interconnection and monitoring of multiple devices simultaneously. Additionally, a PCB dedicated to processing signals from BLAS will be designed for integration into the control system. This design will be carried out using Altium Designer® 21.

Moreover, the complexity and multidisciplinary nature of this Master’s thesis allows it to cover not only various specialities within the Telecommunication Engineering Master’s program but also acquire cross-cutting knowledge and specific skills from other fields such as particle physics. The culmination of this work will result in a detailed study of the electromagnetic behaviour of a pillbox cavity and a comprehensive laboratory monitoring control system.

“175 MHz Cavity Electromagnetic Simulation & Design of a Particle Accelerator Control System”

Andoni Pérez Segura

PALABRAS CLAVE:

RF, BLAS,Ansys HFSS, Pillbox cavity, EPICS, PCB, Altium Designer® 21, VNA, .

RESUMEN:

Este proyecto tiene como principales objetivos el de simular el comportamiento electromagnético de una cavidad de RF, y el de diseñar un sistema de control para un acelerador de partículas. La cavidad a simular y el acelerador de partículas donde implementar el sistema de control tiene el nombre de BLAS. Este acelerador fue diseñado y construido a raíz de una colaboración entre BTESA y CIEMAT.

Por un lado, la simulación se realizará con el software Ansys HFSS. Se realizarán diferentes simulaciones modificando la geometría de la cavidad para estudiar su comportamiento electromagnético. Por otro lado, para el diseño del sistema de control, se utilizará el software EPICS. Dicho software está muy extendido dentro del mundo de los aceleradores de partículas. EPICS permite la interconexión y monitorización de varios equipos simultaneamente. Además, se diseñará una PCB dedicada al procesamiento de señales provenientes del BLAS, para su incorporación en el sistema de control. Dicho diseño se realizará en Altium Designer® 21.

Asimismo, la complejidad y ámbito multidisciplinar de este Trabajo Fin de Máster le permite cubrir, no sólo las diferentes especialidades del Máster de Ingeniería de **Telecomunicación**, sino también adquirir conocimientos y habilidades transversales o específicos de otros campos como la física de partículas. El resultado de todo lo expuesto culmina con la obtención de un estudio detallado del comportamiento electromagnético de una cavidad tipo pillbox, y de un sistema de control capaz de monitorizar un laboratorio por completo.

‘Tough and competent’

Agradecimientos:

Este trabajo ha sido posible gracias a un número muy reducido de personas, pero a las cuales debo un gran agradecimiento; aunque el que pueda mostrarle en estas líneas será sin duda insuficiente, sirva este breve espacio como tal.

En primer lugar, quisiera dar las gracias a mi familia, mis padres, Antonio y Antonia, y mi hermana Michelle, por respaldarme durante toda mi vida y especialmente durante esta etapa. Gracias a mi padre, por impulsarme desde pequeño a estudiar ingeniería y a no rendirme en el camino; a mi madre, por animarme siempre que las cosas se empezaban a torcer; y a mi hermana, por su apoyo incondicional.

Gracias también a mis compañeros de GranaSAT, por hacerme compañía en esas tardes largas donde parece que nada sale y por brindarme su ayuda siempre que la he necesitado.

Este Trabajo Fin de Máster no habría podido llevarse a cabo sin la inestimable ayuda de mi tutor, Andrés Roldán Aranda, a quien estoy enormemente agradecido. También quiero darle las gracias por abrirme las puertas del laboratorio de GranaSAT, por impulsarme a trabajar en un campo diferente y a afrontar nuevos retos.

Y, por último, a Marta, por estar a mi lado, escucharme y apoyarme en todas mis decisiones, muchas gracias.

Contents

List of Figures

List of Tables

Glossary

Agilent Technologies provides electronic design and test solutions.

Altium Designer® 21 software used to design PCB from schematics. It allows 3D Design, as well as electronics simulation.

Anritsu is a Japanese company which provides electronic design and test solutions.

Ansys HFSS 3D electromagnetic (EM) simulation software for designing and simulating high-frequency electronic products such as antennas, antenna arrays, RF or microwave components, high-speed interconnects, filters, connectors, IC packages and printed circuit boards.

Datasheet Document that summarizes the performance and other characteristics of a product in sufficient detail that allows a buyer to understand what the product is and a design engineer to understand the role of the component in the overall system.

Diconex is a European supplier for power loads, resistors, and attenuators for RF and microwaves..

footprint is the arrangement of pads (in SMT) or through-holes (in THT) used to physically attach and electrically connect a component to a PCB.

GitHub is a collaborative development platform for hosting projects using the Git version control system.

GranaSAT GranaSAT is an academic project from the University of Granada originally consisting in designing and developing a picosatellite. Coordinated by the Professor Andrés María Roldán Aranda, GranaSAT is a multidisciplinary project with students from different degrees, where they can acquire and enlarge the knowledge necessary to face an actual aerospace project.

Infineon Technologies German semiconductor manufacturer founded in 1999, when the semiconductor operations of the former parent company Siemens AG were spun off.

Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience..

Keysight Technologies provides electronic design and test solutions.

Marconi Instruments was a company which produced electronic test and measurement equipment and systems, including automatic test equipment.

Metadata A set of data that describes and gives information about other data.

PT100 A resistance thermometer that consists of an element that uses resistance to measure temperature.

Python High-level general-purpose programming language.

Qt Designer Tool for designing and building graphical user interfaces with Qt Widgets.

Raspberry Pi 4 Low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse.

Siglent Global leader in research development, engineering, manufacturing, sales, and service support for electronic test and measurement equipment.

Texas Instruments TI is an American technology company headquartered in Dallas, Texas, that designs and manufactures semiconductors and various integrated circuits, which it sells to electronics designers and manufacturers globally.

Acronyms

AC	Alternating Current.
ADC	Analog to Digital Converter.
BLAS	Beam Loading Advanced Simulator.
BNC	Bayonet Neill–Concelman.
BTESA	Broad Telecom S.A..
CA	Channel Access.
CIEMAT	Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas.
CSS	Control System Studio.
CWS	Client WorkStation.
DC	Direct Current.
DUT	Device Under Test.
EPICS	Experimental Physics and Industrial Control System.
FFT	Fast Fourier Transform.
FM	Frequency Modulation.
GPIB	General Purpose Interface Bus.
GPIO	General Purpose Input Output.
GUI	Graphical User Interface.
HFSS	High-Frequency Structure Simulator.
HP	Hewlett Packard.
IFMIF	International Fusion Materials Irradiation Facility.
IFMIF/EVEDA	International Fusion Materials Irradiation Facility / Engineering Design and Engineering Validation Activities.
IOC	Input/Output Controller.

LAN	Local Area Network.
LCD	Liquid Crystal Display.
LED	Light Emitting Diode.
Linac	Linear accelerators.
LIPAc	Linear IFMIF Prototype Accelerator.
LLRF	Low Level RF.
PCB	Printed Circuit Board.
PV	Process Variable.
PVA	pvAccess.
RF	Radio Frequency.
SCH	Schematic.
SCPI	Standard Commands for Programmable Instruments.
SMA	SubMiniature version A.
SMD	Surface Mounted Device.
SMT	Surface Mount Technology.
SQL	Structured Query Language.
SSPA	Solid State Power Amplifier.
THT	Through-Hole Technology.
TI	Texas Instruments Incorporated.
VNA	Vector Network Analyzer.

Chapter 1

Introduction

The following master's thesis was written as a partial fulfilment of the Telecommunications Engineering Master's degree from the University of Granada (UGR). The goal of this work is to simulate the electromagnetic behaviour of an RF cavity and the design and implementation of a control algorithm for a particle accelerator. This project arises from the collaboration between BTESA, CIEMAT and the University of Granada and seeks to bring the technologies surrounding particle accelerators closer to the academic field.

The imperious need of discovering new ways to produce environmentally friendly energy has impulsed the development of nuclear fusion projects such as International Fusion Materials Irradiation Facility (IFMIF).

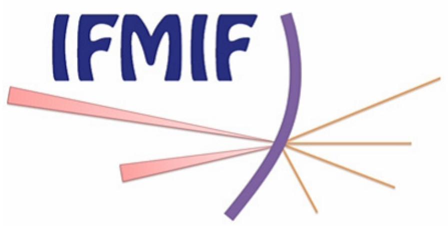


Figure 1.1 – *IFMIF logo*

This project was born to build a specific neutron source based on the nuclear reaction between deuterium and lithium nuclei. For the mitigation of technology risks, a first stage was launched as the IFMIF/EVEDA, aiming to produce an integrated engineering design of the IFMIF plant and the data necessary for future decisions on the construction, operation, exploitation and decommissioning of the future Fusion Neutron Source by completing the engineering validation activities of its three main facilities: the Test Facility, the Lithium Target Facility, and the LIPAc [16]. In the context of the Spanish contribution to IFMIF/EVEDA, BTESA and CIEMAT built the BLAS [5], Fig. 1.2.

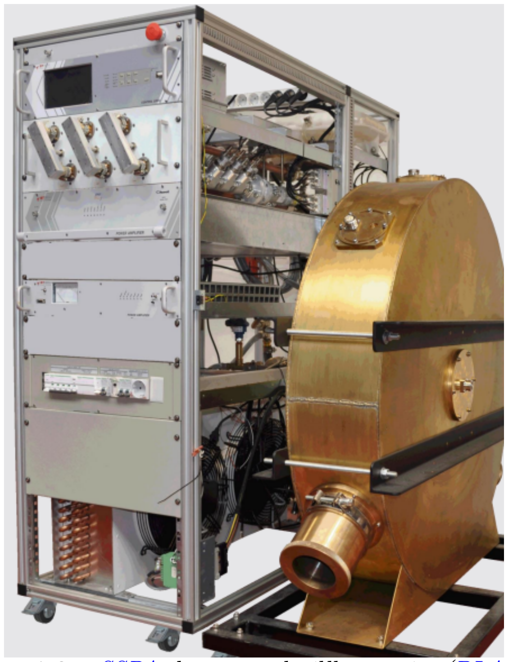


Figure 1.2 – *SSPA drawer and pillbox cavity (BLAS) [5]*

1.1 What is BLAS?

BLAS was designed for educational purposes and consists of an SSPA drawer and an RF cavity that together form the acceleration part from a particle accelerator. The SSPA drawer is in charge of generating the RF power that will make the particles accelerate inside the RF cavity. Both parts will be discussed in detail in Chapter 2.

Shortly, the SSPA drawer generates an RF signal of 175 MHz and 2 kW. The drawer is composed by two RF amplifiers, a circulator, a 50 Ω load, directional couplers and a refrigeration system, explained in Section 2.2. The cavity is a pillbox one with a radius of 0.5 m and that resonates at 175 MHz, analysed in Section 2.3. The electromagnetic simulations from this project will be based on the geometry of this cavity.

1.2 Control algorithm

One of the most important goals for this project is the design of an algorithm capable of controlling and monitoring the BLAS. The BLAS has different control signals that can be processed to know exactly the situation of the particle accelerator. These control signals refer from the output power from the 2 kW amplifier to the flow of the water pump used to refrigerate the system.

The control algorithm will be implemented in a Raspberry Pi 4 model B with touchscreen. This device is expected to receive the different control signals and process them to be able to monitor the particle accelerator. These signals have different tension, some incompatible with the Raspberry Pi, and can be analog or digital. For this reason, a PCB is needed to interpret the control signals and be able to manipulate them with the Raspberry Pi 4. Thus, the design of this PCB plays a crucial role in this work.

This project will have two different approaches to the control and monitor issue. On one hand, the monitoring will be done through a user interface designed with Qt Designer. The signals processed by the PCB will be plotted in the GUI made with the previous tool, using different scripts in Python.

On the other hand, an open source software tool, called EPICS, will be used to implement the algorithm.

EPICS provides a software infrastructure for use in building distributed control systems to operate devices such as particle accelerators, large experiments and major telescopes.

Figure 1.3 – *EPICS logo*

1.3 Project structure

This project has been structured as follows:

- Chapter 1 - Introduction: Brief explanation of what are the goals for this project.
- Chapter 2 - BLAS: Presentation of the main character of this project, a linear particle accelerator.
- Chapter 3 - Electromagnetic simulations: Simulations using Ansys HFSS of a pillbox cavity.
- Chapter 4 - Controller: Development of the different components that build up the system control to design. From PCB design to dedicated software implementation.

Chapter 2

BLAS

This chapter's main goal is to describe different parts that compose the BLAS. However, firstly is necessary to explain where the BLAS comes from. The term BLAS, comes from its name "Beam Loading Advanced Simulator" and was designed in a collaboration between BTESA and CIEMAT. As stated in Chapter 1, BLAS was created for educational purposes under the IFMIF/EVEDA project. This system acts like a linear particle accelerator on a small scale. In this way, the control mechanisms that a linear particle needs, can be tested without having a bigger scale accelerator near. The BLAS is mainly formed by two parts: the SSPA drawer and the RF cavity. These two will be explained in the following sections.

2.1 Particle linear accelerator basis

In this section, the theory behind the particle linear accelerators will be explained, also known as Linacs. A simplified block diagram of a Linac is shown in Figure 2.1. Every Linac needs an RF power system capable of creating the desired magnetic field inside the cavity, and an RF control system that manages the power and frequency of the input RF signal [22]. On the other hand, there is a DC particle injector that provides the beam to accelerate. This beam enters the cavity with a determined speed and exits with a much greater one.

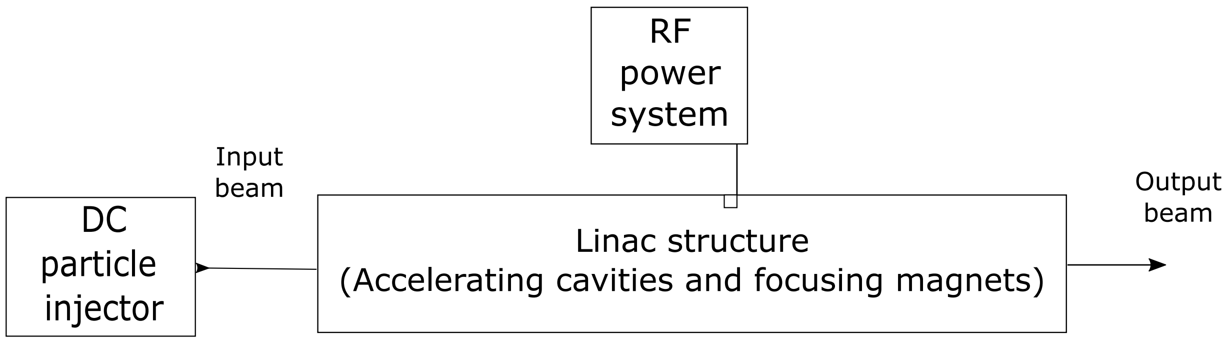

RF power system
Input beam
DC particle injector
Linac structure (Accelerating cavities and focusing magnets)
Output beam

Figure 2.1 – *Linac diagram block*

In order to accelerate the particles from the input beam, an electric field is used. Figure 2.2 indicates the electric field lines generated in an accelerating gap. Ideally, these lines should be horizontal to have a perfect linear acceleration. However, as shown in Figure 2.2, Rf transverse electric fields also act on the beam, so the particles experience transverse defocusing forces.

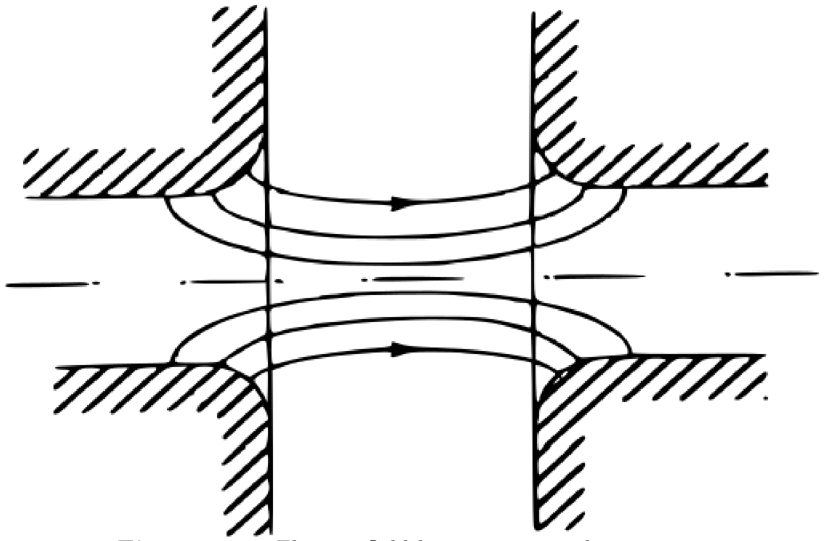


Figure 2.2 – *Electric-field lines in an accelerating gap*

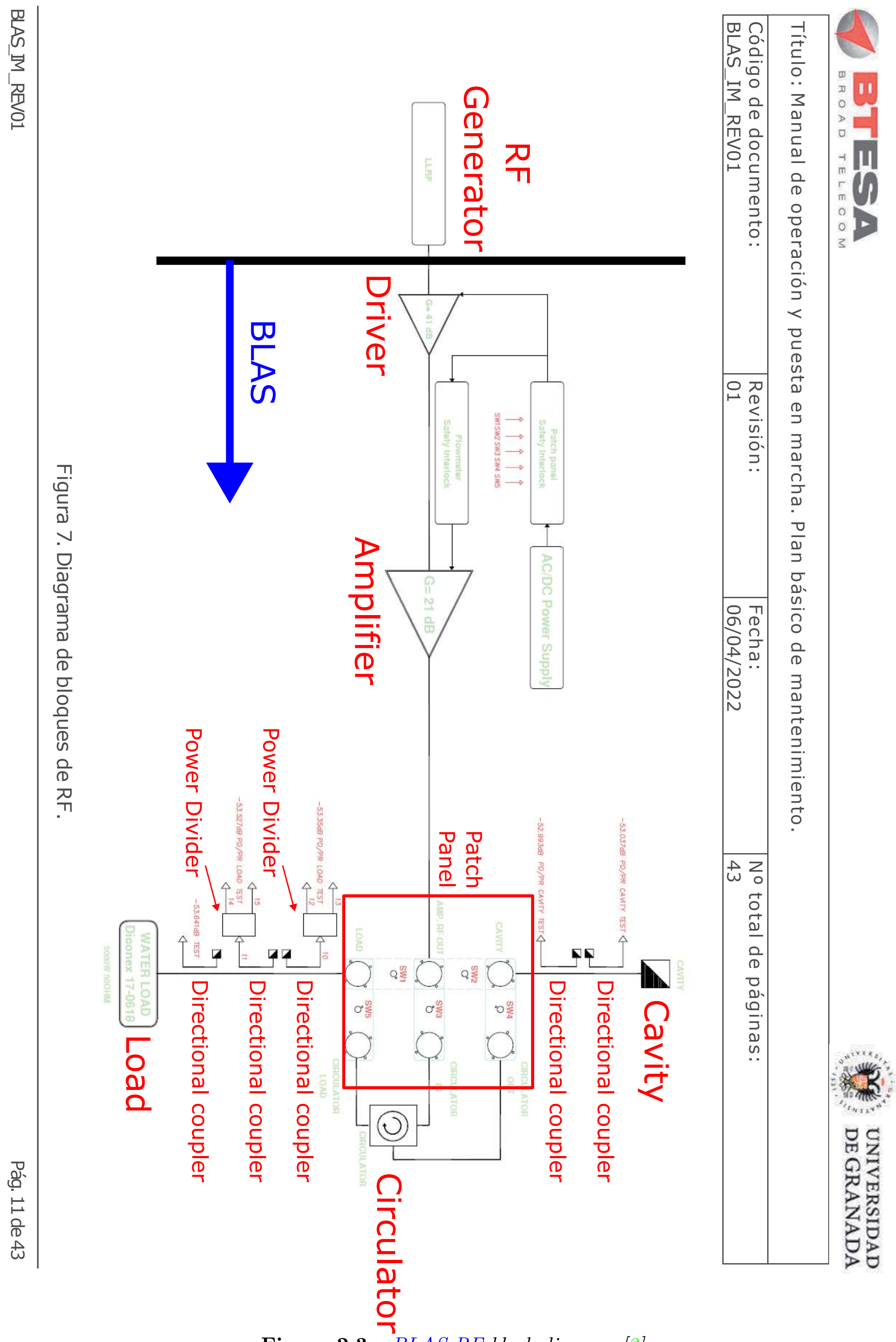
So, BLAS is considered a Linac because consists of an RF power system (SSPA drawer) and a Linac structure (RF cavity).

2.2 SSPA Drawer

The SSPA drawer block diagram is shown in Figure 2.3. The input RF signal is generated outside the BLAS, and introduced into the driver. This driver has a 41 dB gain, which output is connected to the input of the 21 dB gain amplifier. Considering BLAS was designed to provide 63 dBm, 2 kW, the driver must provide 42 dBm, 15.85 W, so the input RF signal from the generator must be of 1 dBm, 1.26 mW. The SSPA drawer was designed to amplify a 175 MHz and 1 mW signal up to 2 kW.

Once the high power signal is generated, it is inserted into a circulator. This circulator works as a safety device to avoid harming the previous RF components, with possible reflected power. The SSPA drawer's circulator has three ports connected to the amplifier output, the cavity, and the 5 kW load, respectively. So, the reflected power from the cavity does not enter the amplifier output and gets redirected to the load.

There are two ways to monitor the power available in the SSPA drawer. On one hand, the different directional couplers can be used to measure the power coupled in them. It is important to know the attenuation that each port has to then calculate the real power in the drawer. There are directional couplers in both circulator outputs, to measure the transmitted and reflected power from the cavity, and from the load. On the other hand, there is software, provided by BTESA, that monitors the input and output power from the driver and the amplifier. Each component that appears in the block diagram will be detailed in further subsections.



Título: Manual de operación y puesta en marcha. Plan básico de mantenimiento.			
Código de documento: BLAS_IM_REV01	Revisión: 01	Fecha: 06/04/2022	Nº total de páginas: 43

Figura 7. Diagrama de bloques de RF.

BLAS_IM_REV01 Pág. 11 de 43

Figure 2.3 – *BLAS RF block diagram* [2]

2.2.1 Driver

The driver was designed to be the first amplifying element, increasing the power value of the low level input RF signal coming from the generator. In Figure 2.4, the back panel of the driver is depicted. It can be seen, the input port is a N female type, and the output port is SMA female type.

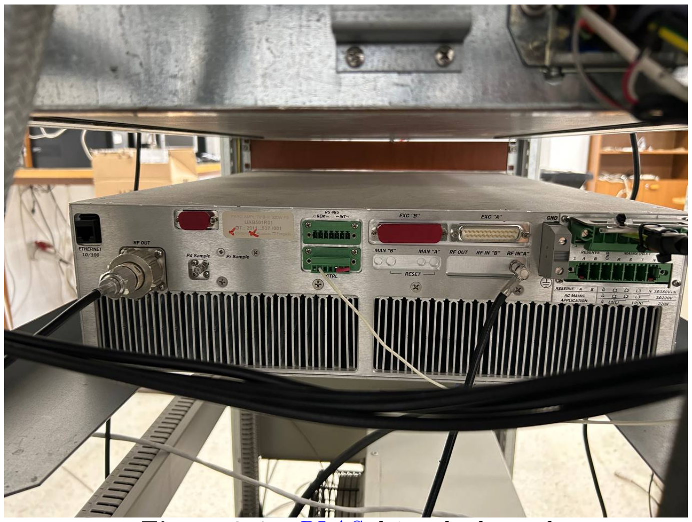


Figure 2.4 – *BLAS driver back panel*

The other connectors are for the serial connection with the software via RS-232. The DB9 connector for the RS-232 is in the front panel, Figure **??**. The mentioned cable connects the driver with a PC containing the monitoring software. Apart from the DB9 connector, there are LED indicators, for different applications. If the driver is working as expected the LEDs *Mains*, *DC supply*. If the other indicators turn orange or red, there is an issue with the driver functioning. For instance, if the RF is too low or high, the *RF OUT* indicators will turn red.

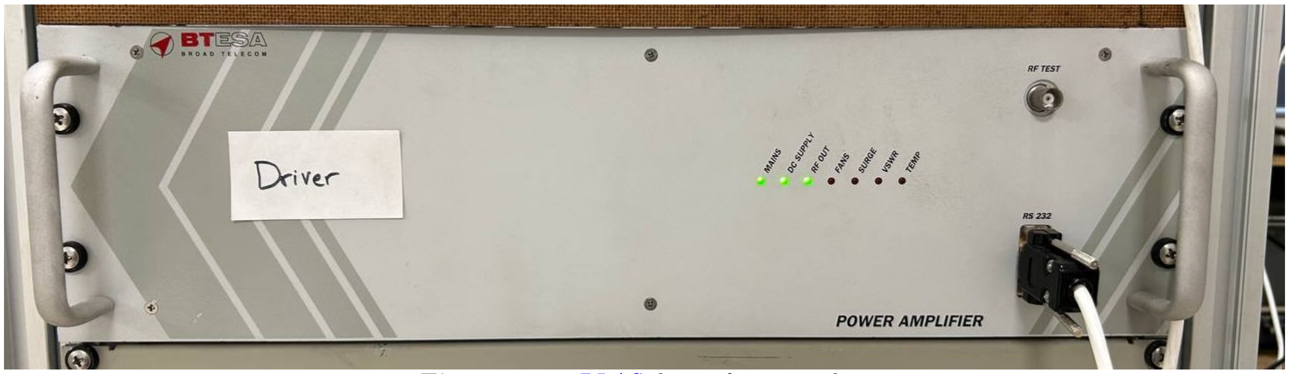


Figure 2.5 – *BLAS driver front panel*

2.2.2 Amplifier

The amplifier is the second amplifying device from the drawer. It is charged with providing the 2 kW signal, with the output signal from the driver. In Figure 2.6 the back panel is depicted. As the amplifier works with higher power signals, there is a need for a water cooling system. The input water port is on the left of the figure, and the output is on the right. Then, the input SMA port can be seen, coming from the driver output SMA port. Moreover, the rest of the connectors are for the serial connection with the software via RS-232.

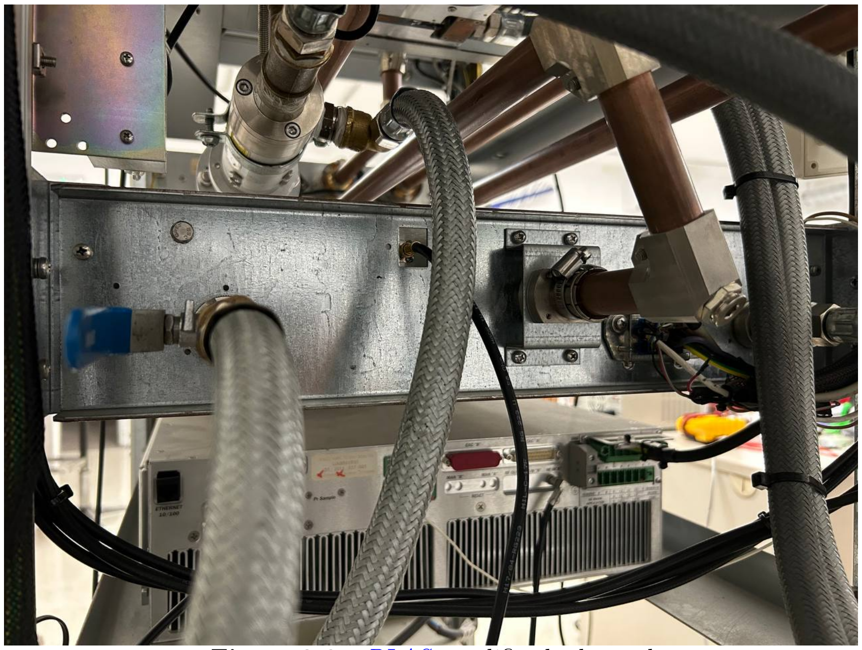


Figure 2.6 – *BLAS amplifier back panel*

The amplifier output is a connector for 7/8" tubing. The high power RF signal is transmitted via tubes and elbows of 7/8", towards the circulator, Figure 2.7.

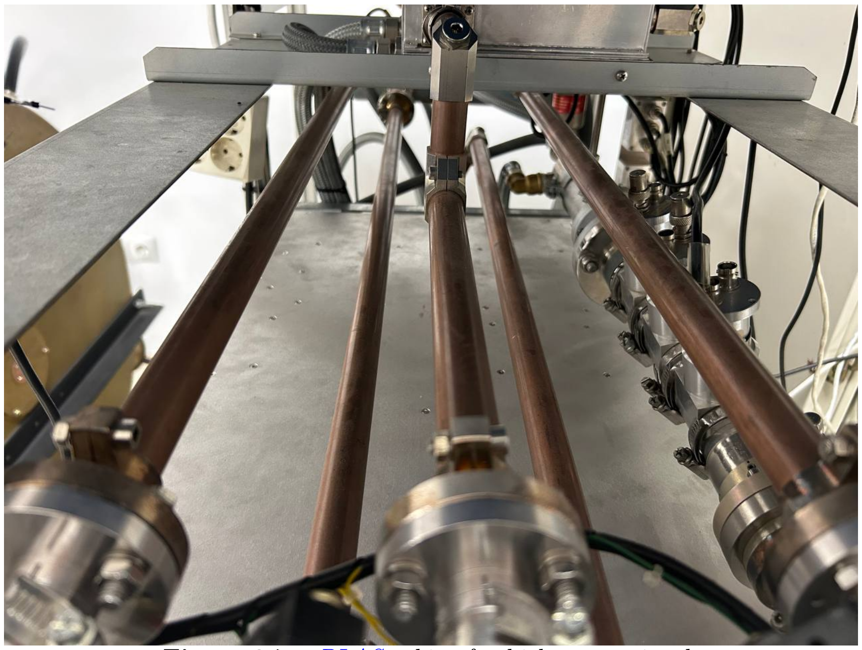


Figure 2.7 – *BLAS tubing for high power signals*

As in the driver's case, the front panel of the amplifier consists of a DB9 female connector, to communicate with the software, and of LED indicators, that have the same application as before, Figure 2.8.

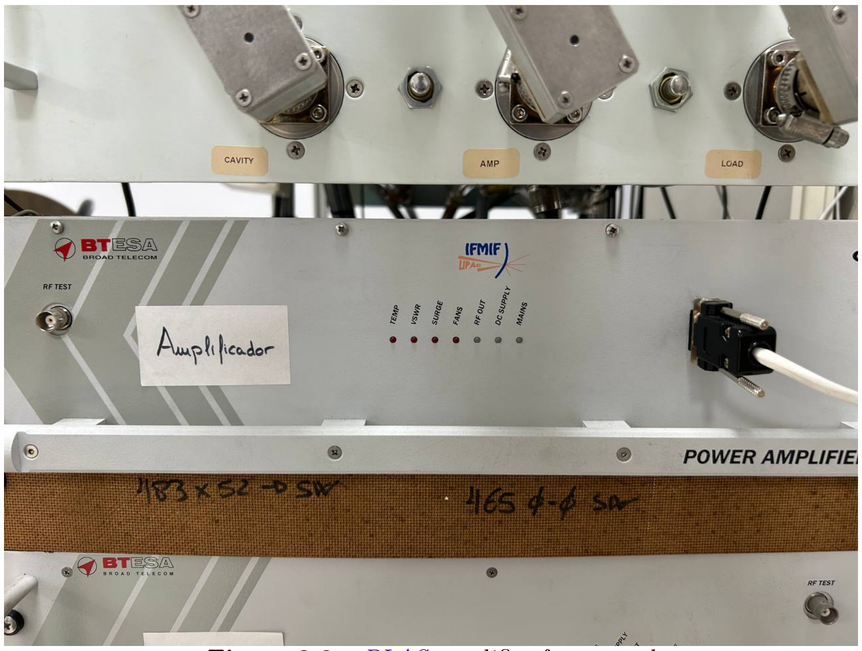


Figure 2.8 – *BLAS amplifier front panel*

2.2.3 Circulator

The circulator is used as a safety measure, because of the high power of the signals used. If the amplifier is directly connected to the cavity, and the cavity's port is not adapted, the 2 kW signal would be totally reflected into the amplifier, harming the RF components. In Figure 2.9, the circulator is shown.

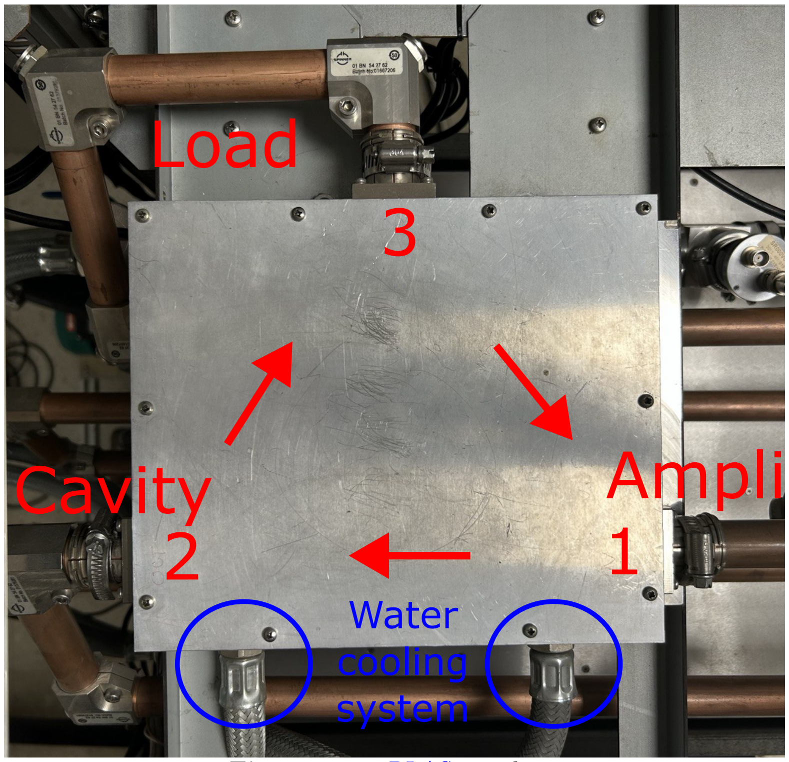


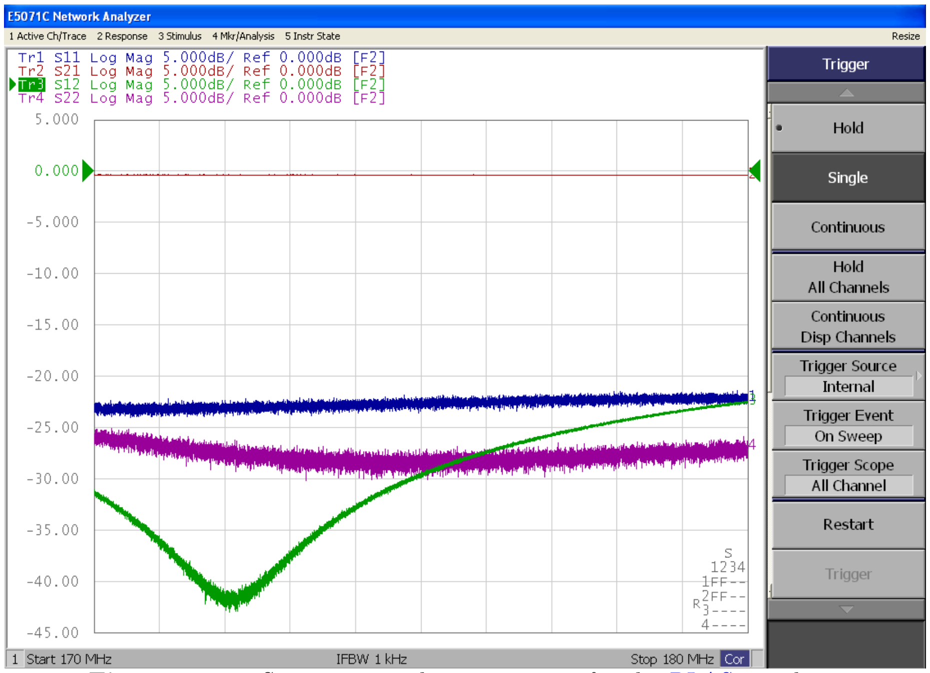
Figure 2.9 – *BLAS circulator*

2.2.3.1 Measurements

As the circulator is a crucial part of the Radio Frequency (RF) system, it has been characterized with a VNA, E5071C from Agilent Technologies. A guide to calibrate with this VNA is in Appendix A.

The circulator has three ports, but as one is connected to the 5 kW load, the characterization will be only with two ports. Port 1 will be the port connected to the amplifier, and port 2 the one connected to the cavity. It is expected that both reflected S parameters (S_{11} and S_{22}) are low, around -25 dB, so the port is adapted. Whereas for transmission S parameters, S_{21} must be near zero, so everything from the amplifier is transmitted to the cavity, and S_{12} must be really low so the reflected power from the cavity dissipates in the load.

The results from the characterization are shown in Figure 2.10. The frequency sweep was between 170 MHz and 180 MHz, as is the range of frequencies BLAS works.



```
Tr1 S11 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr2 S21 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr3 S12 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr4 S22 Log Mag 5.000dB/ Ref 0.000dB [F2]
```

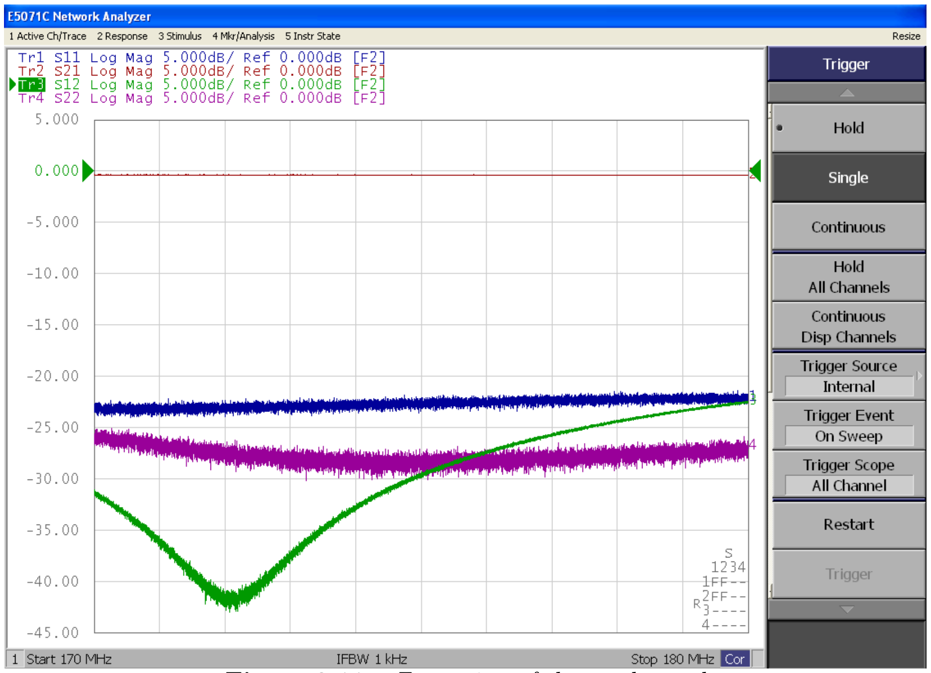
```
1  Start 170 MHz          IFBW 1 kHz          Stop 180 MHz
```

Figure 2.10 – *S parameters characterization for the BLAS circulator*

As expected, the reflected S parameters are below -20 dB, which indicates both ports are adapted. Moreover, S_{21} is nearly zero, so everything is being transmitted, and S_{12} is below -30 dB for most relevant frequencies. The circulator will work as desired.

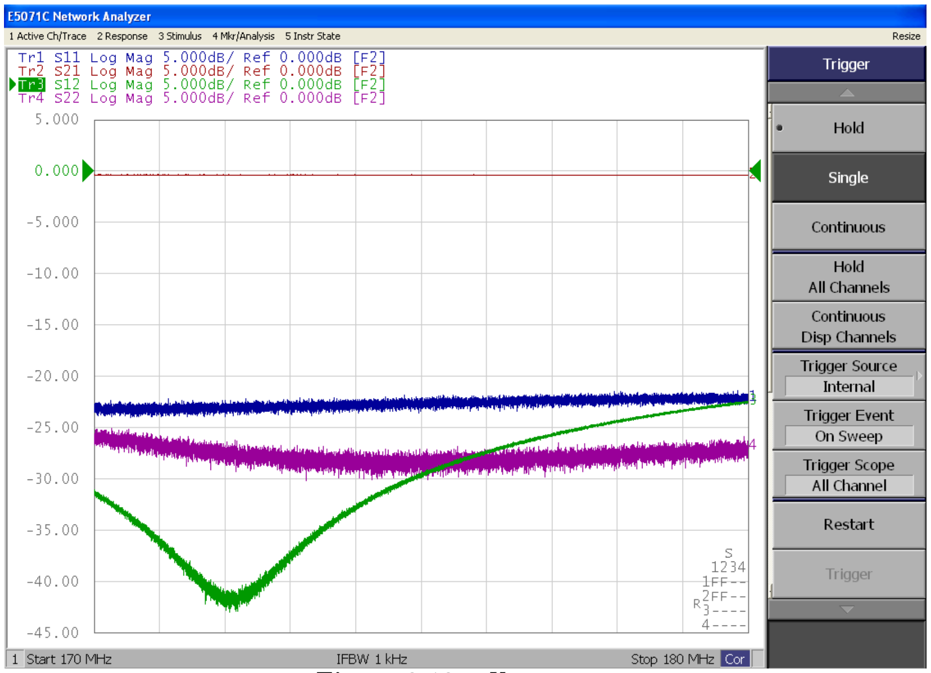
2.2.4 Patch panel

The patch panel is located on the front side of the SSPA drawer, Figure 2.11. The tubing coming from the amplifier output, three circulator ports, the cavity, and the load is located in this panel. These tubes are interconnected using "U" connectors, Figure2.12, commonly known as "telephones". In this way, various configurations can be chosen to operate the BLAS.



```
Tr1 S11 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr2 S21 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr3 S12 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr4 S22 Log Mag 5.000dB/ Ref 0.000dB [F2]
1 Start 170 MHz          IFBW 1 kHz          Stop 180 MHz
```

Figure 2.11 – *Front view of the patch panel*



```
Tr1 S11 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr2 S21 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr3 S12 Log Mag 5.000dB/ Ref 0.000dB [F2]
Tr4 S22 Log Mag 5.000dB/ Ref 0.000dB [F2]
1 Start 170 MHz          IFBW 1 kHz          Stop 180 MHz
```

Figure 2.12 – *U connector*

2.2.4.1 Patch panel configurations

In this section, the various configurations will be explained. The manner to know if the configuration is correct is by using the switches located between each port in the patch panel. Depending on the switches pressed, by the telephones, it is possible to know which tubes are being used. Any configuration out of the list below, will not let the system start. The patch panel diagram is depicted in Figure 2.13.

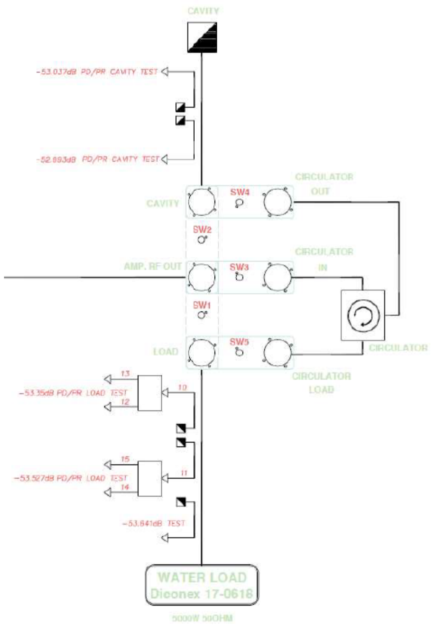


Figure 2.13 – *Patch panel block diagram*

- Amplifier → Cavity: In this configuration, the amplifier output is directly connected to cavity, so there is no circulator protection. SW2 pressed. This configuration is not recommended to avoid issues with reflected power, Figure 2.14.

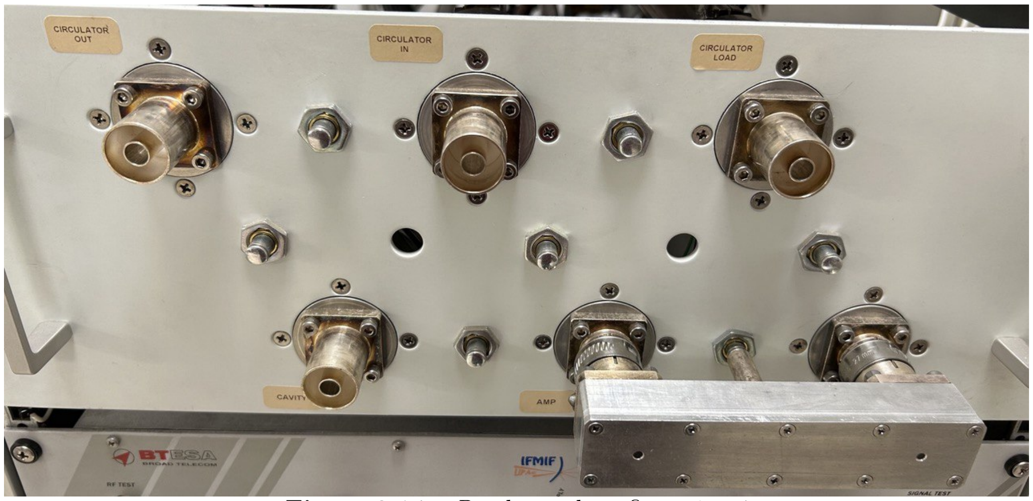


Figure 2.14 – *Patch panel configuration 1*

- Amplifier → Load: In this configuration, the amplifier output is directly connected to the load, SW1 pressed. This configuration is used when no cavity is used, to make measurements in the amplifier's output power, Figure 2.15.

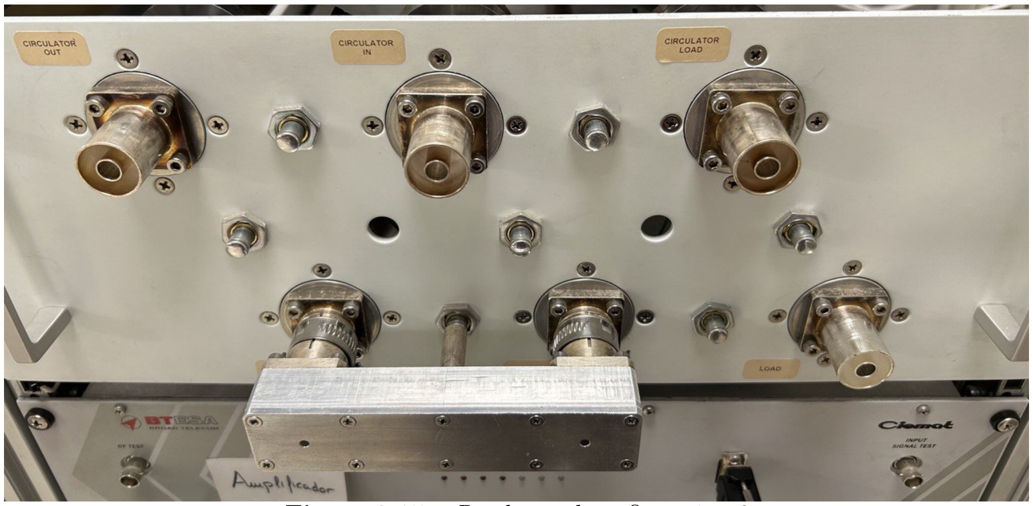


Figure 2.15 – *Patch panel configuration 2*

- Amplifier → Circulator → Cavity → Circulator → Load: In this configuration, the circulator is used. SW3, SW4 and SW5 pressed. This configuration is used when the cavity is used and RF protection is needed for the amplifier, Figure 2.16.

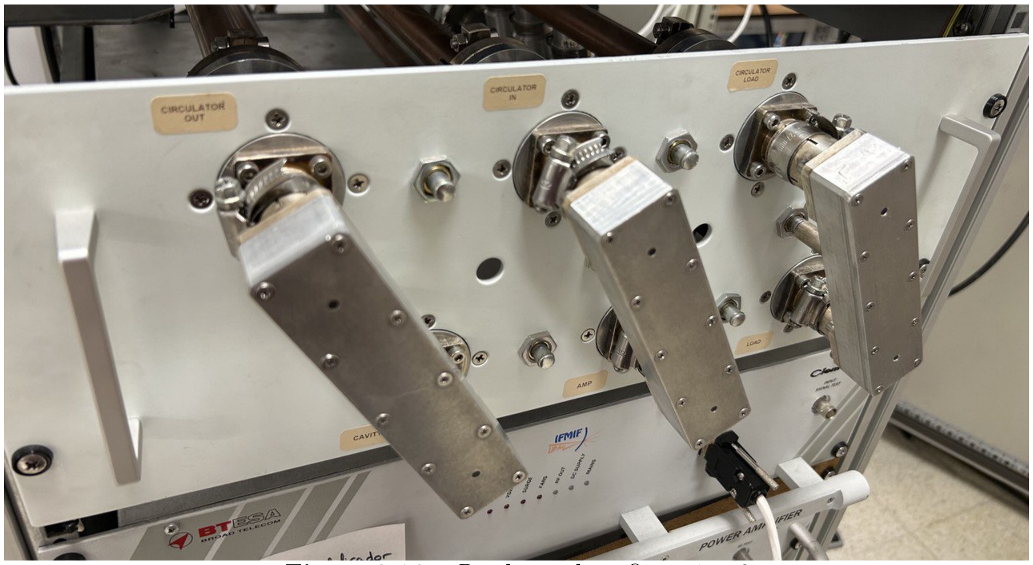
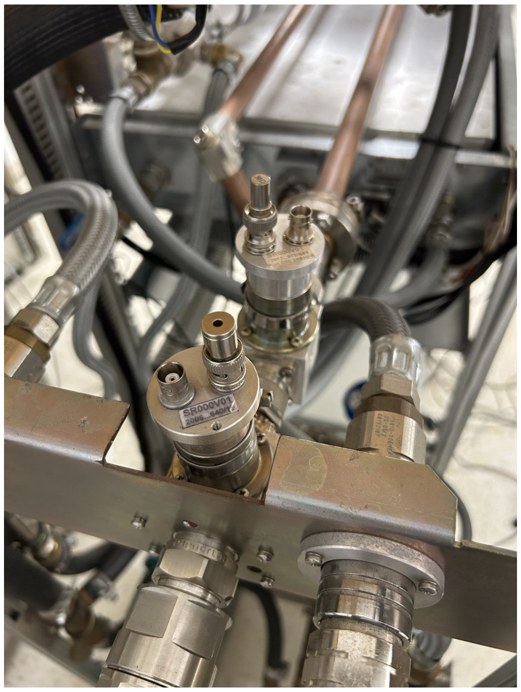


Figure 2.16 – *Patch panel configuration 3*

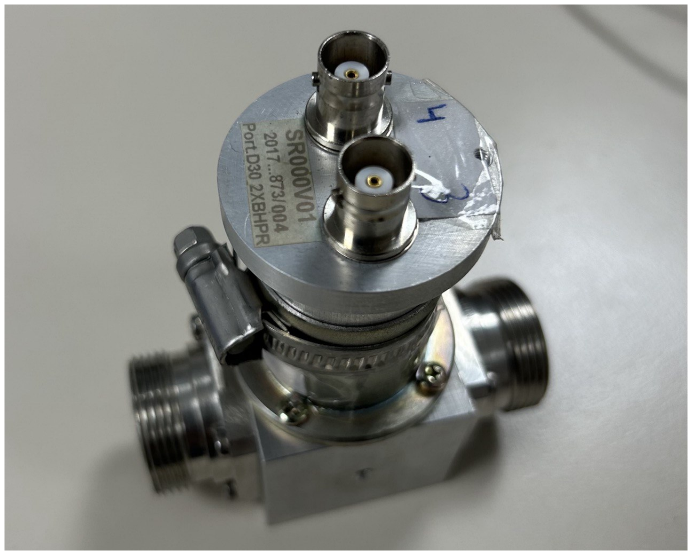
2.2.5 Directional Couplers

In the SSPA drawer there are five directional couplers, three just before the 5 kW load, and two before the cavity. Each directional coupler has two ports to measure the transmitted and reflected power, Figure 2.17. The RF block diagram provided by BTESA indicates the attenuation of each directional coupler. However, the directional couplers are very sensitive to the orientation and position, and they could have moved in the transport from Madrid to Granada. So, in the next subsection, the directional couplers will be characterized.

(a) *Before load directional couplers*



(b) *Before cavity directional couplers*



(c) *Directional coupler*

Figure 2.17 – *SSPA drawer's directional couplers*

As seen in Figure 2.3, the directional couplers power is measured using BNC pigtails that are enumerated. In the case of the couplers before the load, pigtail number 10 will be used for the transmitted power, and probe 11 for the reflected power. Probe 10 will be connected to the directional coupler that is farther from the load, and probe 11 to the nearest one. Probes 10 and 11 will be connected to a power divider with two outputs. So, the power from probe 10 will be divided into probes 12 and 13, while probe 11 power will be divided into probes 14 and 15. Both power splitters are depicted in Figure 2.18.

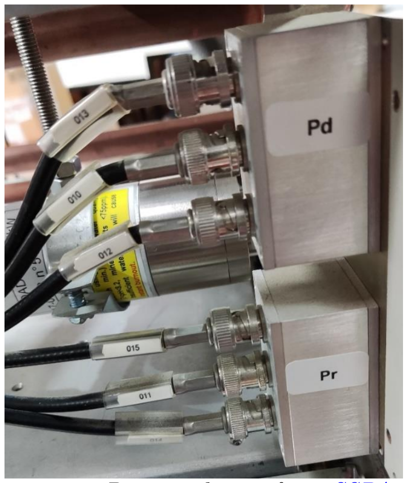
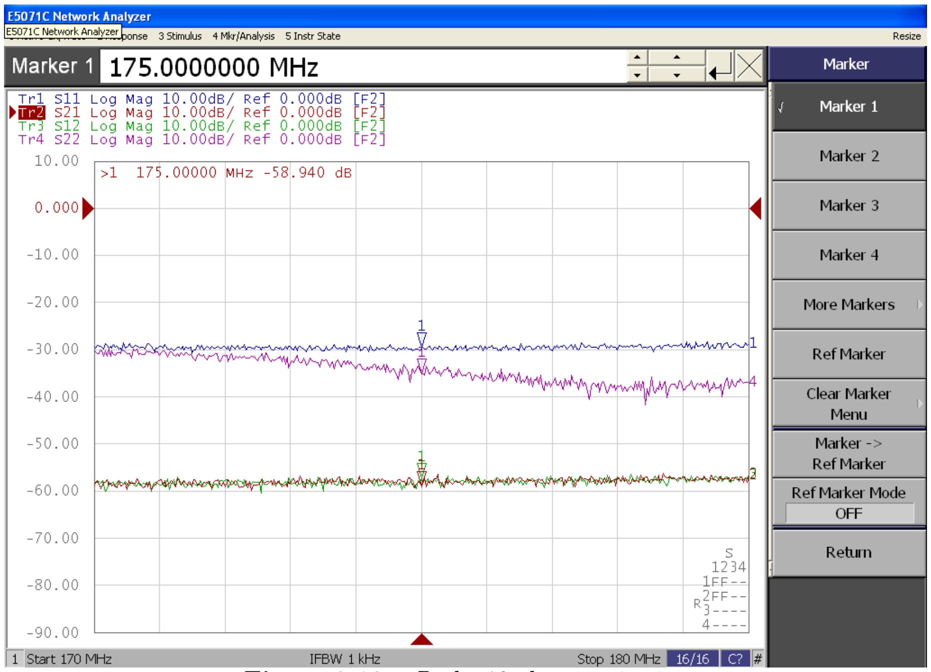


Figure 2.18 – *Power splitters from SSPA drawer*

2.2.5.1 Measurements

To be able to rely on the attenuation values given in Figure 2.3, the attenuation from probe 12 will be measured using the E5071C VNA. The characterization will be a two-port one, with port 1 in the load port from the patch panel, and port 2 at the end of probe 12, Figure 2.19.



```
Marker 1  175.0000000 MHz
Tr1 S11 Log Mag 10.00dB/ Ref 0.000dB [F2]
Tr2 S21 Log Mag 10.00dB/ Ref 0.000dB [F2]
Tr3 S12 Log Mag 10.00dB/ Ref 0.000dB [F2]
Tr4 S22 Log Mag 10.00dB/ Ref 0.000dB [F2]
>1  175.00000 MHz -58.940 dB
1  Start 170 MHz          IFBW 1 kHz          Stop 180 MHz  16/16
```

Figure 2.19 – *Probe 12 characterization*

In Figure 2.3, the power divider was not included in the attenuation values. That's for the difference between the attenuation of probe 12 in the block diagram, -53.35 dB, and the one calculated with the VNA, -58.94 dB. The power divider used in the SSPA drawer is the ZSC-2-4+, which attenuation is 3.2 dB for 175 MHz [17]. So, the attenuations from the block diagram are reliable.

2.2.6 Load

The load is a Diconex 17-0618. This load has an impedance of 50 $\Omega \pm 5\%$, a range frequency of DC – 1 GHz, and a maximum power of 5 kW. The mentioned load is depicted in Figure 2.20.

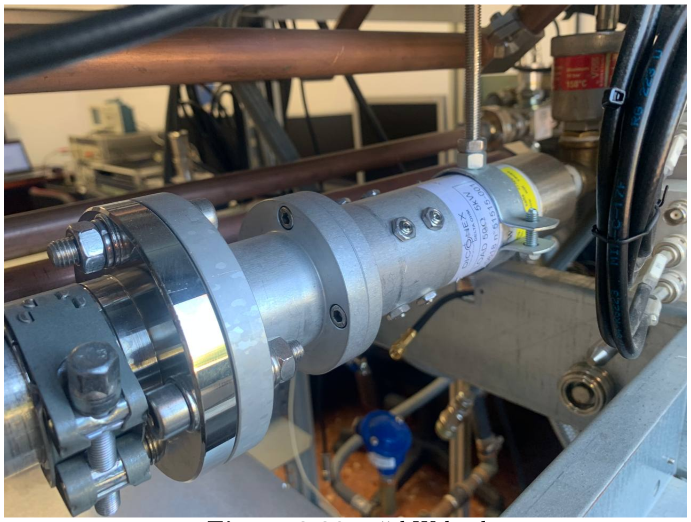


Figure 2.20 – *5 kW load*

2.2.7 BLAS Control Signals

The SSPA drawer includes the possibility to monitor some characteristics and functionalities of the driver and the amplifier, using what is called "BLAS Control Signals". These signals can be manipulated on the left side of the drawer, Figure 2.21.

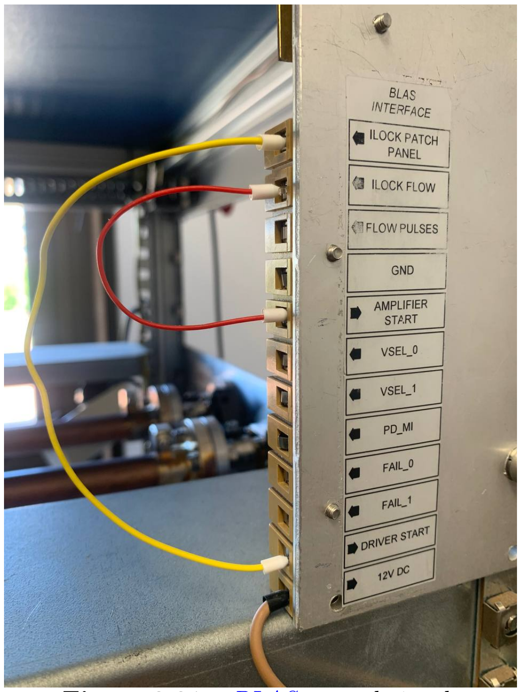


Figure 2.21 – *BLAS control signals*

They can be divided into input or output signals from the BLAS.

2.2.7.1 Output signals

- ILock Patch Panel: This signal controls the patch panel configuration is the correct one. In this case, the signal value is 12 V, if not, zero, Figure 2.22

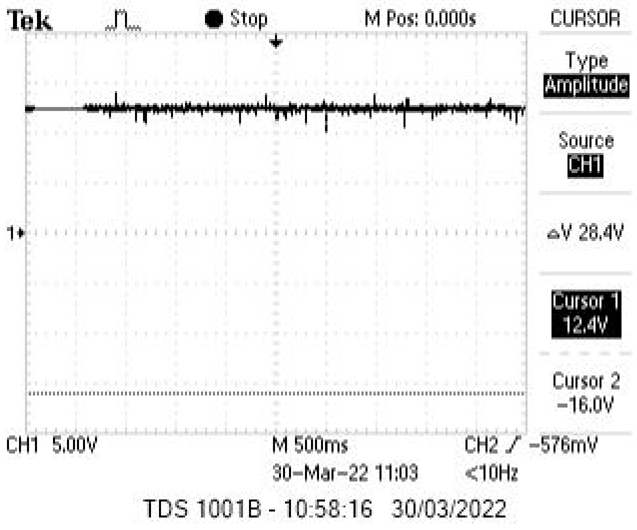


Figure 2.22 – *ILock Patch Panel signal*

- ILock Flow: Controls if the water flow is in the operating range for the system to work properly, 12 V. If not, zero. Figure 2.23.

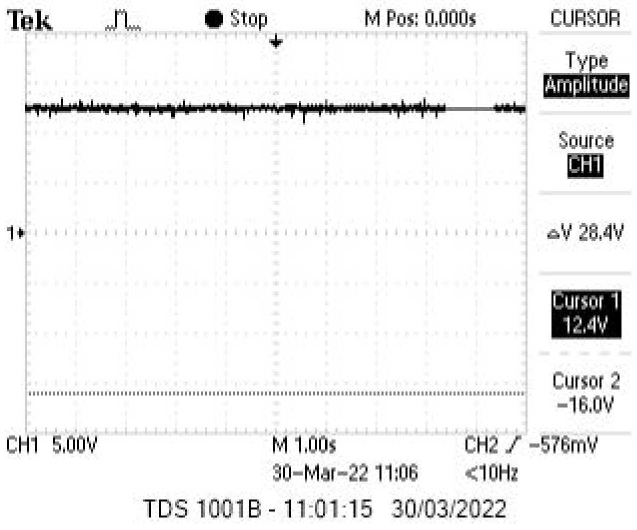


Figure 2.23 – *ILock Flow signal*

- Flow Pulses: Pulsed signal, the frequency of the signal determines de water flow of the system. Conversion: 50 pulses/l. In Figure 2.24, the frequency is of 5.8 pulses/s, so the flow is 7 l/min.

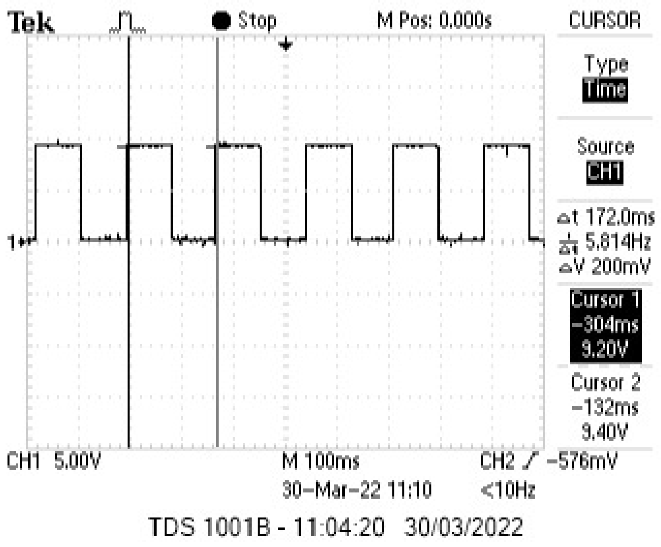


Figure 2.24 – *Flow Pulses signal*

- VSEL 0: Indicates the second control bit to select the source voltage for the amplifying module. Refer to Table 2.1.

- VSEL 1: Indicates the first control bit to select the source voltage for the amplifying module. Refer to Table 2.1.

VSEL 0 (V)	VSEL 1 (V)	Source voltage (V)
0	0	41
0	5	43
5	0	48
5	5	50

Table 2.1 – *VSEL signal voltages*

- PD_MI: Corresponds to the direct power transmitted by the amplifier. A voltage of 7.6 V means 2 kW on the output of the amplifier.

- Fail 0: Indicates if there is an issue with pallet 0 from the amplifier. 0 V if it is working properly, and 5 V if there is a problem. Figure 2.25.

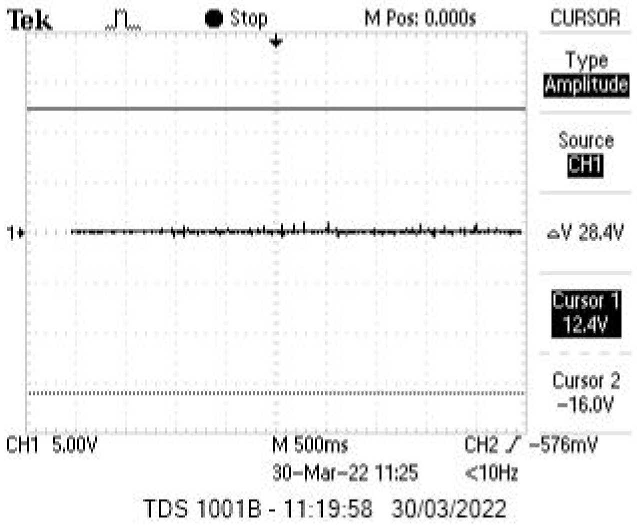


Figure 2.25 – *Fail 0 signal*

- Fail 1: Indicates if there is an issue with pallet 1 from the amplifier. 0 V if it is working properly, and 5 V if there is a problem. Figure 2.26.

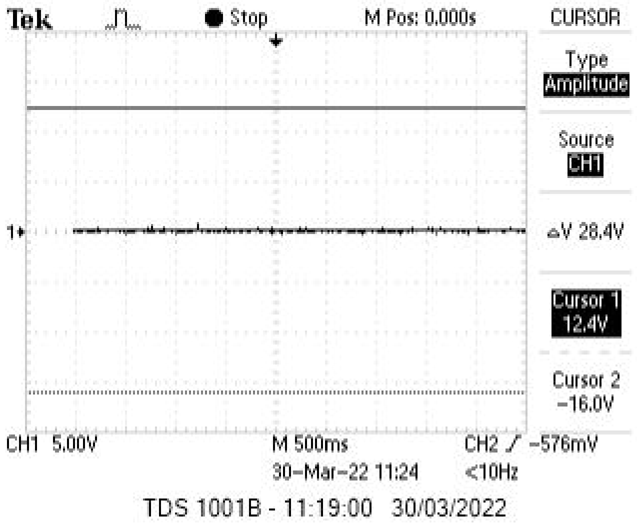


Figure 2.26 – *Fail 1 signal*

2.2.7.2 Input signals

– Amplifier Start: Start signal for the amplifier. In this way, if there is an input of 12 V, the amplifier will start. As seen in Figure 2.21, it is connected to ILock Flow. This means the amplifier will only start if the water flow in the system is correct.

– Driver Start: Start signal for the driver. As in the previous case, the driver will only start if there is a 12 V signal as input. Connected to ILock Patch Panel signal, so the driver will only start if the patch panel configuration, mentioned in Section 2.2.4, is correct.

2.3 RF Cavity

In this section, the RF cavity that makes the BLAS among the SSPA drawer will be presented. Firstly, the electromagnetic fields' mathematical analysis will be explained, and then the Radio Frequency (RF) cavity will be presented.

2.3.1 RF Cavity Theory

2.3.1.1 Maxwell's equations

In order to figure out the electric and magnetic field inside a pillbox cavity, it is necessary to solve the four fundamental equations that describe the behaviour of these fields in a medium and their propagation together as a wave. These are called the Maxwell's equations and are given as in the following [Pozar, 2012 - Balanis, 2012]:

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{2.3.1}$$

$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \tag{2.3.2}$$

$$\nabla \cdot \vec{D} = \rho \tag{2.3.3}$$

$$\nabla \cdot \vec{B} = 0 \tag{2.3.4}$$

where $\vec{E}$, $\vec{H}$, $\vec{D}$, $\vec{B}$ stand for electric field intensity, magnetic field intensity, electric flux density and magnetic flux density, respectively. Moreover, $\vec{J}$ is the electric current density and ρ indicates the charge density. To obtain the above fields, we need to indicate the relations between them, shown in Equations 2.3.5 and 2.3.6.

$$\vec{D} = \epsilon \vec{E} \tag{2.3.5}$$

$$\vec{B} = \mu \vec{H} \tag{2.3.6}$$

Here is where the propagation material is introduced. The electric permittivity ϵ and the magnetic permeability μ depend on the medium, and they are considered as material properties in response to the electromagnetic fields that interact with it. If the propagation medium is vacuum, they are defined as:

$$\mu_0 = 4\pi \times 10^{-7} Hm^{-1} \tag{2.3.7}$$

$$\epsilon_0 = \frac{1}{\mu_0 c^2} Fm^{-1} \tag{2.3.8}$$

where c is the speed of light in vacuum. When we are working in another medium apart from vacuum, the relative permittivity and permeability are used. These can be calculated by using the following relations $\epsilon = \epsilon_r \epsilon_0$ and $\mu = \mu_r \mu_0$. Additionally, to solve Equations 2.3.1,2.3.3, 2.3.2 and 2.3.4 we need an extra relation concerning the electric current density, $\vec{J}$. As referred in [Transmision de Ondas], $\vec{J}$ is defined as:

$$\vec{J} = \vec{J_i} + \vec{J_s} = \sigma \vec{E} + \rho_v \vec{u} \tag{2.3.9}$$

where σ is the conductivity, $\vec{J_i}$ and $\vec{J_s}$ are current and source densities, ρ_v is the volume charge density and $\vec{u}$ the charge velocity. Applying the previous relations and the conditions for electromagnetic fields for source open region, $\sigma = 0$ and $\rho_v = 0$, Maxwell's equations become:

$$\nabla \times \vec{E} = -\mu \frac{\partial \vec{H}}{\partial t} \tag{2.3.10}$$

$$\nabla \times \vec{H} = \epsilon \frac{\partial \vec{E}}{\partial t} \tag{2.3.11}$$

$$\nabla \cdot \vec{E} = 0 \tag{2.3.12}$$

$$\nabla \cdot \vec{H} = 0 \tag{2.3.13}$$

At this point, we can obtain the homogeneous wave equation for the electric field by using the vector identity $\Delta\vec{A} = \nabla \cdot (\nabla \cdot \vec{A}) - \nabla \times \nabla \times \vec{A}$ following these steps:

$$\nabla \times \vec{E} = -\mu \frac{\partial \vec{H}}{\partial t} \tag{2.3.14}$$

$$\nabla \times \nabla \times \vec{E} = -\mu \frac{\partial (\nabla \times \vec{H})}{\partial t} \tag{2.3.15}$$

$$\nabla \cdot (\nabla \cdot \vec{E}) - \Delta\vec{E} = -\mu \frac{\partial (\epsilon \frac{\partial \vec{E}}{\partial t})}{\partial t} \tag{2.3.16}$$

$$\Delta\vec{E} = \mu\epsilon \frac{\partial^2 \vec{E}}{\partial t^2} \tag{2.3.17}$$

On the other hand, the homogeneous wave for the magnetic field can be obtained by employing the duality theorem, as seen in Equation 2.3.18.

$$\Delta\vec{H} = \mu\epsilon \frac{\partial^2 \vec{H}}{\partial t^2} \tag{2.3.18}$$

Until this point, Maxwell's equations were time dependant and three-dimensional. In order to omit this time dependency, phasor representation can be used, supposing harmonic time variation. The phasor form of electric and magnetic fields are obtained by using the following steps:

$$\nabla \times \vec{E}(x,y,z,t) + \mu \frac{\partial \vec{H}(x,y,z,t)}{\partial t} = 0 \tag{2.3.19}$$

$$\nabla \times Re\left\{\vec{E}(x,y,z)e^{j\omega t}\right\} + \frac{\partial}{\partial t} Re\left\{\mu\vec{H}(x,y,z)e^{j\omega t}\right\} = 0 \tag{2.3.20}$$

$$Re\left\{\nabla \times \vec{E}(x,y,z)e^{j\omega t}\right\} + Re\left\{\frac{\partial}{\partial t}\mu\vec{H}(x,y,z)e^{j\omega t}\right\} = 0 \tag{2.3.21}$$

$$Re\left\{\left[\nabla \times \vec{E}(x,y,z) + j\omega\mu\vec{H}(x,y,z)\right] e^{j\omega t}\right\} = 0 \tag{2.3.22}$$

$$\nabla \times \vec{E}(x,y,z) + j\omega\mu\vec{H}(x,y,z) = 0 \tag{2.3.23}$$

Considering source in free region, the homogeneous Helmholtz equations can be written as:

$$\Delta\vec{E} + k^2\vec{E} = 0 \tag{2.3.24}$$

$$\Delta\vec{H} + k^2\vec{H} = 0 \tag{2.3.25}$$

where $k = \omega\sqrt{\epsilon\mu}$ is the wave number for the lossless medium. These Helmholtz equations can be solved by using the separation of variables in different coordinate systems such as cartesian, cylindrical and spherical for the corresponding boundary conditions.

2.3.1.2 Cylindrical waveguide

A pillbox cavity can be seen as a cylindrical waveguide with the two ends covered. In order to obtain the EM fields inside our cavity, it is necessary to calculate them for a cylindrical waveguide. We contemplate an infinitely long cylindrical waveguide in the z-direction with a radius a. The walls are made of a perfect conductor ($\sigma \to \infty$) and the waveguide is filled by a material of permeability μ and permittivity ϵ, as shown in Figure 2.27.

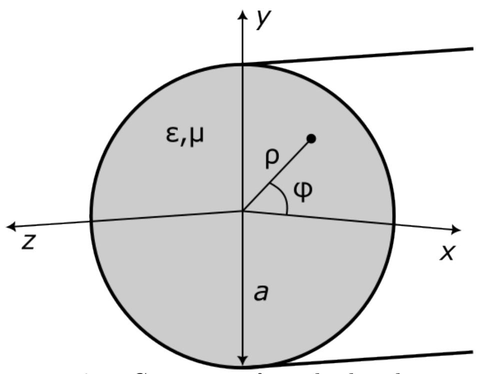


Figure 2.27 – *Geometry of a cylindrical waveguide.*

The analysis of the electric and magnetic fields for cylindrical waveguides is achieved using cylindrical coordinates. The three set of solutions for plane waves are transverse electric (TE), transverse magnetic (TM) and transverse electromagnetic (TEM). Nevertheless, TEM modes cannot propagate inside a waveguide [Gerigk, 2013a]. Because of this, only TE and TM modes can be found in the cylindrical waveguides of interest. The EM fields are defined in cylindrical coordinates as:

$$\vec{E}(\rho,\phi,z) = E_\rho(\rho,\phi)e^{-j\beta z}\hat{\rho} + E_\phi(\rho,\phi)e^{-j\beta z}\hat{\phi} + E_z(\rho,\phi)e^{-j\beta z}\hat{z} \tag{2.3.26}$$

$$\vec{H}(\rho,\phi,z) = H_\rho(\rho,\phi)e^{-j\beta z}\hat{\rho} + H_\phi(\rho,\phi)e^{-j\beta z}\hat{\phi} + H_z(\rho,\phi)e^{-j\beta z}\hat{z} \tag{2.3.27}$$

From now on, Maxwell's equations will be written in the frequency domain as:

$$\nabla \times \vec{E} = -j\omega\mu\vec{H} \tag{2.3.28}$$

$$\nabla \times \vec{H} = j\omega\epsilon\vec{E} \tag{2.3.29}$$

$$\nabla \cdot \vec{D} = \rho \tag{2.3.30}$$

$$\nabla \cdot \vec{B} = 0 \tag{2.3.31}$$

To decompose each component of the electric and magnetic field, we need to solve Maxwell's equations, for that we apply the cylindrical rotational to the electric field as following:

$$\nabla \times \vec{E} = \frac{1}{\rho} \begin{pmatrix} \hat{\rho} & \rho\hat{\phi} & \hat{z} \\ \frac{\partial}{\partial\rho} & \frac{\partial}{\partial\phi} & \frac{\partial}{\partial z} \\ E_\rho & \rho E_\phi & Ez \end{pmatrix} = -j\omega\mu\vec{H} \tag{2.3.32}$$

$$= \frac{1}{\rho}\left[\frac{\partial E_z}{\partial\phi} - \frac{\partial\rho E_\phi}{\partial z}\right]\hat{\rho} - \frac{1}{\rho}\left[\frac{\partial E_z}{\partial\rho} - \frac{\partial E_\rho}{\partial z}\right]\rho\hat{\phi} + \frac{1}{\rho}\left[\frac{\partial(\rho E_\phi)}{\partial\rho} - \frac{\partial E_\rho}{\partial z}\right]\hat{z} = -j\omega\mu\vec{H} \tag{2.3.33}$$

$$H_\rho = -\frac{1}{j\omega\mu}\left[\frac{1}{\rho}\frac{\partial E_z}{\partial\phi} - j\beta E_\phi\right] \tag{2.3.34}$$

$$H_\phi = -\frac{1}{j\omega\mu}\left[j\beta E_\rho - \frac{\partial E_z}{\partial\rho}\right] \tag{2.3.35}$$

$$H_z = -\frac{1}{j\omega\mu}\frac{1}{\rho}\left[\frac{\partial\rho E_\phi}{\partial\rho} - \frac{\partial E_\rho}{\partial\phi}\right] \tag{2.3.36}$$

Applying the same method with Equation 2.3.29 we obtain:

$$E_\rho = -\frac{1}{j\omega\epsilon}\left[\frac{1}{\rho}\frac{\partial H_z}{\partial\phi} - j\beta H_\phi\right] \tag{2.3.37}$$

$$E_\phi = -\frac{1}{j\omega\epsilon}\left[j\beta H_\rho - \frac{\partial H_z}{\partial\rho}\right] \tag{2.3.38}$$

$$E_z = -\frac{1}{j\omega\epsilon}\frac{1}{\rho}\left[\frac{\partial\rho H_\phi}{\partial\rho} - \frac{\partial H_\rho}{\partial\phi}\right] \tag{2.3.39}$$

Now, we arrange Equation 2.3.35 and Equation 2.3.37:

$$H_\phi = -\frac{\beta}{\omega\mu}E_\rho + \frac{1}{j\omega\mu}\frac{\partial E_z}{\partial\rho} \tag{2.3.40}$$

$$E_\rho = -\frac{1}{j\omega\epsilon\rho}\frac{\partial H_z}{\partial\phi} - \frac{\beta}{\omega\epsilon}H_\phi \tag{2.3.41}$$

Combining both of the above equations, we obtain E_ρ following these steps:

$$E_\rho = -\frac{1}{j\omega\epsilon\rho}\frac{\partial H_z}{\partial\phi} - \frac{\beta}{\omega\epsilon}\left[-\frac{\beta}{\omega\mu}E_\rho + \frac{1}{j\omega\mu}\frac{\partial E_z}{\partial\rho}\right] \tag{2.3.42}$$

$$E_\rho = -\frac{1}{j\omega\epsilon\rho}\frac{\partial H_z}{\partial\phi} + \frac{\beta^2}{\omega^2\epsilon\mu} - \frac{\beta}{\omega\mu}E_\rho - \frac{\beta}{j\omega^2\mu\epsilon}\frac{\partial E_z}{\partial\rho} \tag{2.3.43}$$

$$E_\rho(\frac{\omega^2\mu\epsilon - \beta^2}{\omega^2\mu\epsilon}) = \frac{1}{j\omega\epsilon\rho}\frac{\partial H_z}{\partial\phi} - \frac{\beta}{j\omega^2\epsilon\mu}\frac{\partial E_z}{\partial\rho} \tag{2.3.44}$$

where β is the propagation constant and $k = \omega\sqrt{\mu\epsilon}$ is the wave number. Looking at Equation 2.3.44, using β and k, the cut-off wave number can be defined as $k_c^2 = k^2 - \beta^2$, in order to have wave propagation $k_c > 0$. E_ρ is rewritten as:

$$E_\rho(\frac{k_c^2}{k^2}) = \frac{1}{j\omega\epsilon\rho}\frac{\partial H_z}{\partial\phi} - \frac{\beta}{jk^2}\frac{\partial E_z}{\partial\rho} \tag{2.3.45}$$

For the rest of the components of the EM fields, the same steps are followed and are given as:

$$E_\rho = -\frac{j}{k_c^2}(\beta\frac{\partial E_z}{\partial\rho} + \frac{\omega\mu}{\rho}\frac{\partial H_z}{\partial\phi}) \tag{2.3.46}$$

$$E_\phi = -\frac{j}{k_c^2}(\frac{\beta}{\rho}\frac{\partial E_z}{\partial\phi} - \omega\mu\frac{\partial H_z}{\partial\rho} \tag{2.3.47}$$

$$H_\rho = \frac{j}{k_c^2}(\frac{\omega\epsilon}{\rho}\frac{\partial E_z}{\partial\phi} - \beta\frac{\partial H_z}{\partial\rho} \tag{2.3.48}$$

$$H_\phi = -\frac{j}{k_c^2}(\omega\epsilon\frac{\partial E_z}{\partial\rho} + \frac{\beta}{\rho}\frac{\partial H_z}{\partial\phi} \tag{2.3.49}$$

E_z and H_z will be discussed in the next subchapter.

2.3.1.3 TE mode

As mentioned before, in cylindrical waveguides, the only modes of propagation are TE and TM. In this subchapter, TE modes will be discussed, the ones where the electric field component in the propagation direction is equal to zero. In our case, we are studying the propagation in the z direction, so $E_z = 0$. Because of this, we only need to obtain H_z to have all the EM field components defined. This can be extracted from the following homogeneous Helmholtz equation:

$$\Delta\vec{H}_z + k^2\vec{H}_z = 0 \tag{2.3.50}$$

where $\Delta = \frac{\partial^2}{\partial\rho^2} + \frac{1}{\rho}\frac{\partial}{\partial\rho} + \frac{1}{\rho^2}\frac{\partial^2}{\partial\phi^2} + \frac{\partial^2}{\partial z^2}$ is Laplacian in cylindrical basis and $\vec{H}_z(\rho,\phi,z) = h_z(\rho,\phi)e^{-j\beta z}$. Adding these two expressions to the Helmholtz equation, we obtain

$$\left(\frac{\partial^2}{\partial\rho^2} + \frac{1}{\rho}\frac{\partial}{\partial\rho} + \frac{1}{\rho^2}\frac{\partial^2}{\partial\phi^2} + k_c^2\right)h_z(\rho,\phi) = 0 \tag{2.3.51}$$

Following the separation of variables method, $h_z(\rho, phi) = R(\rho)P(\phi)$, the next expression must satisfy Equation 2.3.51.

$$\left(\frac{\partial^2 R(\rho)}{\partial\rho^2}\right)P(\phi) + \left(\frac{1}{\rho}\frac{\partial R(\rho)}{\partial\rho}\right)P(\phi) + \left(\frac{1}{\rho^2}\frac{\partial^2 P(\phi)}{\partial\phi^2}\right)R(\rho) + k_c^2 R(\rho)P(\phi) = 0 \tag{2.3.52}$$

To distinct Equation 2.3.52 into ρ and ϕ components, both sides are divided by $R(\rho)P(\phi)$ and multiplied with ρ^2 which leads:

$$\frac{1}{R\rho}\rho\frac{\partial}{\partial\rho}\left[\rho\frac{R(\rho)}{\partial\rho}\right]+\rho^2k_c^2=-\frac{1}{P(\phi)}\frac{\partial^2P(\phi)}{\partial\phi^2} \tag{2.3.53}$$

According to the separation of variables method, each side can be solved by equating the zero. For example, the "ϕ" dependent function is obtained from the solution of the following equation.

$$\frac{\partial^2P(\phi)}{\partial\phi^2}+k_\phi^2P(\phi)=0 \tag{2.3.54}$$

Because of the perfect conductor materials of the surface of the cylindrical waveguide, the electric and magnetic fields $(\vec{E},\vec{B})$ must satisfy $\hat{n}\times\vec{E}$ and $\hat{n}\cdot\vec{B}$, where $\hat{n}$ is the surface's normal vector. In our case, the field $h_z(\rho,\phi)$ is a continuous function, periodic with 2π in ϕ and k_ϕ is an integer. The general solution for the second order differential Equation 2.3.54 is

$$P(\phi)=Asin(k_\phi\phi)+Bcos(k_\phi\phi)) \tag{2.3.55}$$

Applying the boundary conditions, $k_\phi=n=0,1,2\ldots$ then:

$$P(\phi)=Asin(n\phi)+Bcos(n\phi)) \tag{2.3.56}$$

where A and B are arbitrary constants. Once $P(\rho)$ is defined, it is inserted into Equation 2.3.54, both sides are multiplied by $R(\rho)$ and added to Equation 2.3.53:

$$\rho\frac{\partial}{\partial\rho}\left[\rho\frac{R(\rho)}{\partial\rho}\right]+\left[(k_c\rho)^2-n^2\right]R(\rho)=0 \tag{2.3.57}$$

This equation is called Bessel differential equation. The two linearly independent solutions to this equation are

$$R(\rho)=CJ_n(k_c\rho)+DY_n(k_c\rho) \tag{2.3.58}$$

$$H_z(\rho)=R(\rho)\left[Acos(n\phi)+Bsin(n\phi))\right]e^{-j\beta z} \tag{2.3.59}$$

where J_n is the Bessel function of first kind, Y_n is the Bessel function of second kind, and C and D are arbitrary constants. Nevertheless, $Y_n(\rho=0)$ goes to infinite and causes singularity at field, so D has to be equal to zero. This let us with h_z defined as:

$$h_z(\rho,\phi)=\left[Asin(n\phi)+Bcos(n\phi)\right]J_n(k_c\rho) \tag{2.3.60}$$

Now that all the components from both fields are defined, it is necessary to calculate the cut-off wave number, k_c, to be able to know the bandwidth from each TE mode. To do so, boundary conditions $E_\phi(\rho=a,\phi,z)=0$ are applied to E_ϕ, Equation 2.3.47.

$$E_\phi(\rho=a,\phi,z)=\frac{j\omega\mu}{k_c}\left[Asin(n\phi)+Bcos(n\phi)\right]J_n(k_ca)=0 \tag{2.3.61}$$

From Equation 2.3.61 is observed that:

$$J_n'(k_c a) = 0 \rightarrow k_c a = P_{nm}' \rightarrow k_c = \frac{P_{nm}'}{a} \tag{2.3.62}$$

where P_{nm}' is the m^{th} root of $J_n'(P_{nm}')$ and their values are listed in Table 2.2.

P_{nm}'	**m=1**	**m=2**
n=0	3.832	7.016
n=1	1.841	5.331
n=2	3.054	6.706

Table 2.2 – *Root values for J_n'*

Equation 2.3.62 indicates the way to calculate the bandwidth for each TE mode. If the cut-off wave number, k_c and the roots of the derivative of the Bessel function of first kind, P_{nm}', relate as:

$$k_{c,nm} = \frac{P_{nm}'}{a} \tag{2.3.63}$$

where $k_{c,nm}^2 = k^2 - \beta_{nm}^2$, then the cut-off frequency can be defined as follows:

$$\beta_{nm} = \sqrt{k^2 + k_{c,nm}^2} = \sqrt{k^2 + \left(\frac{P_{nm}'}{a}\right)^2} \tag{2.3.64}$$

$$f_{c,nm} = \frac{k_{c,nm}}{2\pi\sqrt{\mu\epsilon}} \tag{2.3.65}$$

So if $k > k_c$, β is real and the wave propagates. However, if $k < k_c$ the wave is attenuated, being k the wave number in free space. Finally, the electromagnetic field in a cylindrical waveguide in TE mode are summarized as:

$$E_\rho = -\frac{j\omega\mu}{k_c^2\rho}\left[Acos(n\phi) - Bsin(n\phi)\right] J_n(k_c\rho)e^{-j\beta z} \tag{2.3.66}$$

$$E_\phi = \frac{j\omega\mu}{k_c}\left[Asin(n\phi) + Bcos(n\phi)\right] J_n'(k_c\rho)e^{-j\beta z} \tag{2.3.67}$$

$$E_z = 0 \tag{2.3.68}$$

$$H_\rho = -\frac{j\beta}{k_c}\left[Asin(n\phi) + Bcos(n\phi)\right] J_n'(k_c\rho)e^{-j\beta z} \tag{2.3.69}$$

$$H_\phi = -\frac{j\beta n}{k_c^2\rho}\left[Acos(n\phi) - Bsin(n\phi)\right] J_n(k_c\rho)e^{-j\beta z} \tag{2.3.70}$$

$$H_z = \left[Asin(n\phi) + Bcos(n\phi)\right] J_n(k_c\rho)e^{-j\beta z} \tag{2.3.71}$$

2.3.1.4 TM mode

TM mode refers to transverse magnetic field to the propagation direction, in our case, +z direction with $H_z = 0$. To obtain the electric and magnetic fields for TM modes, the steps from the previous subchapter can be followed. Instead of having $E_z = 0$, we have $E_z(\rho, \phi, z) = e_z(\rho, \phi)e^{-j\beta z}$. Using the method of separable variables, a similar solution is derived:

$$e_z(\rho, \phi) = [Asin(n\phi) + Bcos(n\phi))]\, J_n(k_c\rho) \tag{2.3.72}$$

with A and B as arbitrary constants, and J'_n is the first kind of Bessel function. The boundary condition $E_z(\rho = a, \phi, z) = 0$ is applied to E_z:

$$E_z(\rho = a, \phi, z) = [Asin(n\phi) + Bcos(n\phi))]\, J_n(k_ca)e^{-j\beta z} = 0 \tag{2.3.73}$$

As was done before, to satisfy the previous Equation, J_n must be equal to zero:

$$J_n(k_ca) = 0 \rightarrow k_ca = P_{nm} \rightarrow k_c = \frac{P_{nm}}{a} \tag{2.3.74}$$

where P_{nm} is the m^{th} root of $J_n(P_{nm})$ and their values are listed in Table 2.3.

P_{nm}	**m=1**	**m=2**
n=0	2.405	5.520
n=1	3.832	7.016
n=2	5.135	8.417

Table 2.3 – *Root values for J_n*

Equation 2.3.74 indicates the way to calculate the bandwidth for each TM mode. If the cut-off wave number, k_c and the roots of the derivative of the Bessel function of first kind, P_{nm}, relate as:

$$\beta_{nm} = \sqrt{k^2 - k_{c,nm}^2} = \sqrt{k^2 - \left(\frac{P'_{nm}}{a}\right)^2} \tag{2.3.75}$$

$$f_{c,nm} = \frac{k_{c,nm}}{2\pi\sqrt{\mu\epsilon}} = \frac{P_{nm}}{2\pi\sqrt{\mu\epsilon}} \tag{2.3.76}$$

Between TE and TM modes, the mode with smallest cut-off frequency is defined as the "dominant mode". For circular waveguides, TE_{11} is the dominant one, as can be seen in Tables 2.2 and 2.3. The smaller the root of the Bessel function, the smaller the cut-off frequency. Finally, the electric and magnetic fields for TM modes are defines as:

$$E_z = [Asin(n\phi) + Bcos(n\phi)]\, J_n(k_c\rho)e^{-j\beta z} \tag{2.3.77}$$

$$E_\rho = -\frac{j\beta}{k_c}\, [Asin(n\phi) + Bcos(n\phi)]\, J'_n(k_c\rho)e^{-j\beta z} \tag{2.3.78}$$

$$E_\phi = -\frac{j\beta n}{k_c\rho}\, [Asin(n\phi) - Bsin(n\phi)]\, J_n(k_c\rho)e^{-j\beta z} \tag{2.3.79}$$

$$H_z = 0 \tag{2.3.80}$$

$$H_\rho = -\frac{j\omega\epsilon n}{k_c^2\rho}\, [Acos(n\phi) - Bsin(n\phi)]\, J_n(k_c\rho)e^{-j\beta z} \tag{2.3.81}$$

$$H_\phi = -\frac{j\omega\epsilon}{k_c^2\rho}\, [Asin(n\phi) - Bsin(n\phi)]\, J'_n(k_c\rho)e^{-j\beta z} \tag{2.3.82}$$

2.3.1.5 Cylindrical pillbox cavity

As was mentioned before, a pillbox cavity can be seen as a cylindrical waveguide with two end walls on the sides. This type of accelerating cavity has an infinite number of resonant modes. The goal of this subchapter is to present the most important modes using the electric and magnetic field equations defined in previous chapters. For this, it is assumed that the pillbox cavity to study has a length "d" and a radius "a", as seen in Figure 2.28.

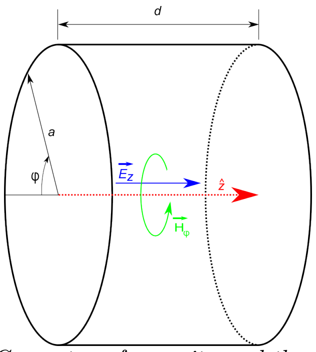


Figure 2.28 – *Geometry of a cavity and the EM fields inside.*

The main difference between a cavity and a conventional waveguide is the idea of having standing waveforms rather than travelling waves. As before, the only modes that can propagate inside this structure are the TE and TM modes. In this case, we consider the electric field as:

$$\vec{E}(\rho, \phi, z) = \vec{e}(\rho, \phi)(A^{+}e^{-j\beta_{nm}z} + A^{-}e^{-j\beta_{nm}z}) \tag{2.3.83}$$

where $\vec{e}(\rho, \phi)$ refers to the transverse mode variations, the ones that are not in the z propagation direction, A^{+} and A^{-} are the amplitudes of the forward and backward waves, respectively. For TE modes, the propagation constant was defined as:

$$\beta_{nm} = \sqrt{k^2 + k^2_{c,nm}} = \sqrt{k^2 + \left(\frac{P\prime_{nm}}{a}\right)^2} \tag{2.3.84}$$

and for the TM modes as:

$$\beta_{nm} = \sqrt{k^2 + k^2_{c,nm}} = \sqrt{k^2 + \left(\frac{P_{nm}}{a}\right)^2} \tag{2.3.85}$$

The first boundary condition to apply is $\vec{E}(\rho, \phi, z = 0) = 0$, in the wall we have no electric field as it is a perfect conductor.

$$\vec{E}(\rho, \phi, z = 0) = \vec{e}(\rho, \phi)(A^{+} + A^{-}) \tag{2.3.86}$$

Equation 2.3.86 is only true if $A^{+} = -A^{-}$. The second boundary condition, $\vec{E}(\rho, \phi, z = d) = 0$, states that the electric field in the second wall is zero, because of the condition of perfect conductor the wall has.

$$\vec{E}(\rho, \phi, z) = \vec{e}(\rho, \phi)(A^{+}e^{-j\beta_{nm}d} + A^{-}e^{-j\beta_{nm}d}) = 0 \tag{2.3.87}$$

$$A^{+}(cos(\beta_{nm}d) - jsin(\beta_{nm}d)) + A^{-}(cos(\beta_{nm}d) - jsin(\beta_{nm}d)) = 0 \tag{2.3.88}$$

$$cos(\beta_{nm}d)(A^{+} + A^{-}) + jsin(\beta_{nm}d)(A^{+} + A^{-}) = 0 \tag{2.3.89}$$

From the first condition, we defined $A^{+} = -A^{-}$. However, if this assumption is applied to Equation 2.3.89, the result is not equal to zero so,

$$sin(\beta_{nm}d) = 0 \rightarrow \beta_{nm}d = \frac{l\pi}{d} \tag{2.3.90}$$

where $l = 0,\ 1,\ 2....$ Hence, the cut-off resonant frequencies for the TE and TM modes can be calculated as:

$$TE_{nml}: \quad f_{c,nml} = \frac{c}{2\pi\sqrt{\mu_r\epsilon_r}}\sqrt{\left(\frac{l\pi}{d}\right)^2 + \left(\frac{P'_{nm}}{a}\right)^2} \tag{2.3.91}$$

$$TM_{nml}: \quad f_{c,nml} = \frac{c}{2\pi\sqrt{\mu_r\epsilon_r}}\sqrt{\left(\frac{l\pi}{d}\right)^2 + \left(\frac{P_{nm}}{a}\right)^2} \tag{2.3.92}$$

Looking at the equations, we can determine that if the dominant mode for the cylindrical waveguides was TE_{11}, for the pillbox cavity will be TE_{110}. Nevertheless, not all the electromagnetic fields are valid to accelerate particles, so we need to consider Lorentz Force equation:

$$\vec{F} = q\left(\vec{E} + \vec{v} \times \vec{B}\right) \tag{2.3.93}$$

where $\vec{v}$ and q are the speed and charge of the particles, respectively. From the Lorentz equation, two different are assumed. On the one hand, $\vec{F_e} = q\vec{E}$ is the electric force applied to a particle with the same direction as the electric field. On the other hand, $\vec{F_m} = q(\vec{v} \times \vec{B})$ is the magnetic force perpendicular to the magnetic field and the speed of the charge. Work-energy relation is used to calculate energy changes of a particle. The kinetic energy variation can be expressed as:

$$\Delta E_k = \int \vec{F}d\vec{l} \tag{2.3.94}$$

Inserting the Lorentz Force equation, Equation 2.3.93 into the kinetic energy variation one, Equation 2.3.94[Jensen, 2016], and keeping in mind that $\vec{v} = \frac{d\vec{l}}{dt}$, we obtain:

$$\Delta E_k = q\int \vec{E}d\vec{l} + q\int(\vec{v} \times \vec{B})d\vec{l} = q\int \vec{E}d\vec{l} \tag{2.3.95}$$

This demonstrates that is the electric field the one that can actually accelerate particles by adding kinetic energy to them. The magnetic field is used to guide particles. Consequently, the particle will be accelerated in the direction of $\vec{E}$. We are looking for an acceleration in the z direction, so the TE modes are directly discarded for acceleration purposes, as $E_z = 0$. Only TM modes can be used, specifically, there is a mode called "acceleration mode" that is the dominant one inside the TM ones, the TM_{010} mode. For TM_{010}, the electric field only has E_z and the magnetic field H_ϕ. The other components are equal to zero.

$$E_z = E_0 J_0 \left(\frac{2.405\rho}{a}\right) e^{-j\omega t} \tag{2.3.96}$$

$$H_\phi = -j\sqrt{\frac{\epsilon_0}{\mu_0}} E_0 J_1 \left(\frac{2.405\rho}{a}\right) e^{-j\omega t} \tag{2.3.97}$$

Finally, we can rewrite Equation 2.3.92 to define the resonant cut-off frequency for the TM_{010}, which becomes independent of the cavity length d. We can conclude that the bigger our frequency is, the smaller the cavity we are going to need.

$$TM_{010}: \quad f_{c,010} = \frac{2.405c}{2\pi a\sqrt{\mu_r \epsilon_r}} \tag{2.3.98}$$

2.3.1.6 Quality factor

One of the most important parameters that describe a cavity is the quality factor, which defines how much energy is lost, or RF power stored in the cavity. Also, it characterizes the bandwidth of the cavity relative to its resonance frequency. As mentioned in [Gerigk,2013b], the quality factor is defined as:

$$Q = \omega_{wres}\frac{Stored\,energy\,in\,cavity}{Energy\,dissipated\,in\,cavity\,walls} = 2\pi f_{res}\frac{W}{P_c} \tag{2.3.99}$$

On one hand, the stored energy (W) in a certain volume can be calculated by using the electric or magnetic fields based on the Ampere equation. The formulation for stored energy is the following:

$$W = \frac{\epsilon_0}{2}\iiint_V |\vec{E}|^2 dV = \frac{\mu_0}{2}\iiint_V |\vec{H}|^2 dV \tag{2.3.100}$$

On the other hand, the dissipated power, $P_c = \frac{R_s}{2}\iint_S |\vec{H}|^2 ds$, is caused by surface resistance, $R_s = \frac{1}{\sigma\delta}$, where δ and σ are skin depth and conductivity of the material, respectively. Because of skin depth, not all the wall's thickness can be used for the current flow. This limitation of the conductor available section, causes an increase in the impedance and increases the losses. Looking at surface resistance definition, a higher conductivity implies higher quality factors.

There are three different types of quality factors depending on the connections the cavity has to external components.

- Unloaded quality factor, $Q_0 = \omega_{res}\frac{W}{P_c}$, depends on the losses due to the walls or conductivity without any connections.
- External quality factor, $Q_e = \omega_{res}\frac{W}{P_ext}$, measures the losses caused by connecting the cavity to external circuits, as generators, power couplers, etc.
- Loaded quality factor, $Q_l = \omega_{res}\frac{W}{P_ext+P_c}$, defines whole system losses.

The relation between the three types is:

$$\frac{1}{Q_l} = \frac{1}{Q_0} + \frac{1}{Q_e} \tag{2.3.101}$$

An RF cavity can be modeled as the following parallel RLC circuit:

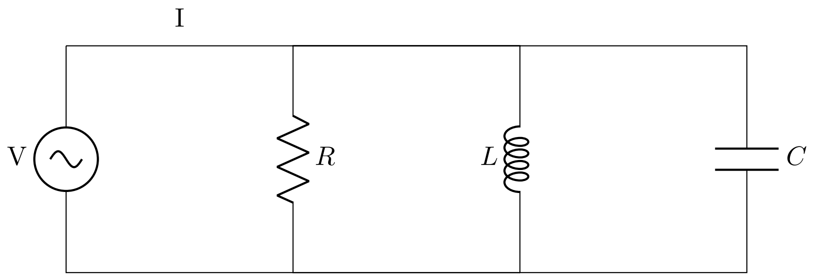


Figure 2.29 – *Circuital model of an RF cavity*

So, the input impedance seen from the voltage source can be denoted as:

$$Z_{in} = \left(\frac{1}{R} + \frac{1}{j\omega L} + j\omega C\right)^{-1} = \frac{R}{1 + jR(\omega C - \frac{1}{\omega L})} \tag{2.3.102}$$

The definition of the resonance state of a parallel RLC circuit states that voltage and current must be in the same phase, the complex term of Z_{in} is zero and the resonance angular frequency is $\omega_{res} = \frac{1}{\sqrt{LC}}$. To be able to determine Q_0, it is necessary to found the energy dissipated in the walls and the stored energy, Equation 2.3.99. The only losses in Figure 2.29 are the ohmic ones, defined as:

$$Pc = \frac{|V|^2}{2R} \tag{2.3.103}$$

Also, C and L generate an electric and magnetic energy that is stored and expressed as:

$$W_{electric} = \frac{|V|^2 C}{4} \tag{2.3.104}$$

$$W_{magnetic} = \frac{|V|^2}{4\omega^2 L} = \frac{|i_L|^2 L}{4} \tag{2.3.105}$$

Introducing Equations 2.3.103, 2.3.104 and 2.3.105 into the Q_0 definition, we achieve the quality factor in terms of passive components and the ω_{res}.

$$Q_0 = \omega_{res}\frac{W}{P_c} = \frac{R}{\omega_{res} L} \tag{2.3.106}$$

To obtain the external quality factor, we add another R_L resistor in parallel with the other passive components, and $Q_e = \frac{R_L}{\omega_{res} L}$ is defined.

To calculate the equivalent impedance of the circuital model, $\omega_{res} = \frac{1}{\sqrt{LC}}$ can be rewritten as $C = \frac{1}{\omega_{res}^2 L}$ and inserted into Equation 2.3.102.

$$Z_{in} = \frac{R}{1 + j\frac{R}{L}(\frac{\omega}{\omega_{res}^2} - \frac{1}{\omega})} \tag{2.3.107}$$

where $\frac{\omega^2-\omega_{res}^2}{\omega_{res}^2\omega} = \frac{(\omega-\omega_{res})(\omega+\omega_{res})}{\omega_{res}^2\omega}$ and we assume $\omega \approx \omega_{res} \rightarrow \omega + \omega_{res} \cong 2\omega_{res}$. To associate with Q_0, this equation becomes $\frac{\omega^2-\omega_{res}^2}{\omega_{res}^2\omega} = \frac{2}{\omega_{res}}\left(\frac{(\omega-\omega_{res})}{\omega_{res}}\right)$ so the input impedance seen from the generator is:

$$Z_{in} = \frac{R}{1 + jQ_0 2\left(\frac{\omega-\omega_{res}}{\omega_{res}}\right)} \tag{2.3.108}$$

To ensure maximum power transmission, the transmission line that connects the voltage source and the cavity have to be coupled, Figure 2.30. This relation is defined by the coupling coefficient (β) and it is related to the external quality factor. The relation between β and Q_e, using Equation 2.3.108 in resonance condition, $Z_{in} = R$, where $\omega = \omega_{res}$ and $Q_0 = \frac{R}{\omega_{res}L}0$. To obtain as much power transmission as possible, $Z_{in} = Z_0$, so $Q_e = \frac{Z_0}{\omega_{res}L=Q_0}$. Equation 2.3.101 states the Q_l is equal to half of Q_0. Coupling coefficient is defined as $\beta = \frac{Q_0}{Q_e} = \frac{R}{Z_0}$. Then, if $\beta < 1$ we have an under coupling state, if $\beta = 1$ the coupling is called critical and if $\beta > 1$ we have over coupling.

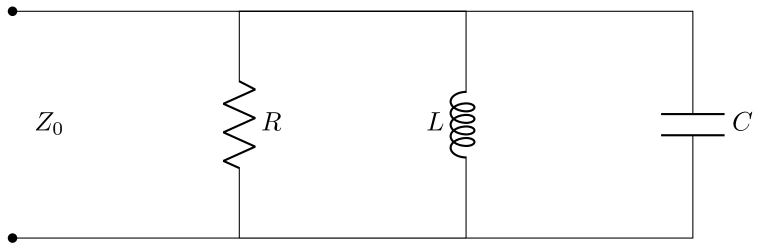


Figure 2.30 – *Transmission line connected to an RF cavity*

2.3.2 BLAS Cavity

In this section, the BLAS RF cavity will be presented, Figure 2.31. Firstly, the physical characteristics are going to be presented. Then, an S parameter characterization of the cavity will be performed.

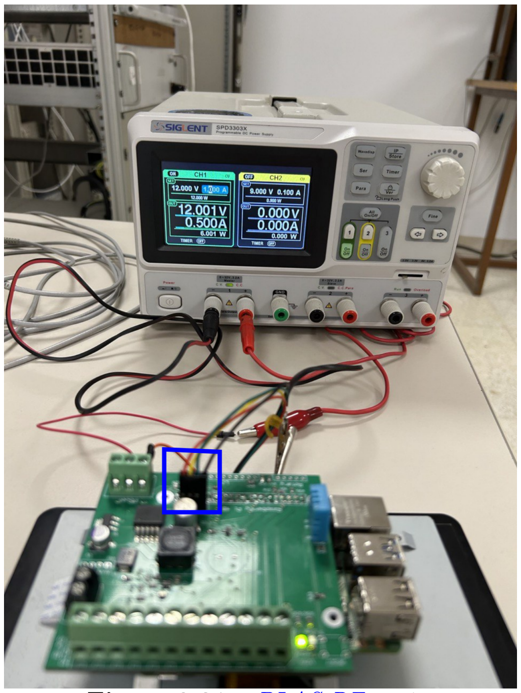


Figure 2.31 – *BLAS RF cavity*

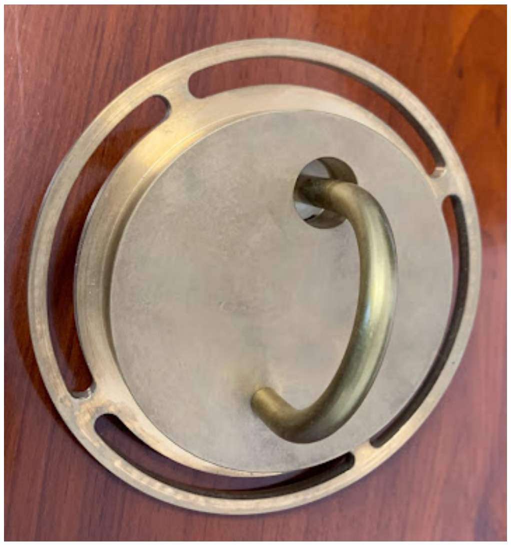
2.3.2.1 Physical characteristics

The BLAS cavity is a pillbox type one made from brass. Being a pillbox cavity means it has a cylindrical shape in which the resonant frequency will be given by the radius of the cylinder, independent of the width, as explained in 2.3.1. The radius of this pillbox cavity is 506 mm, the width is 20 mm and the thickness is 6 mm. This thickness needs to be bigger than the skin effect, which determines the depth of conductor that is transmitting an AC signal. The skin effect depth comes from Equation 2.3.109, [1].

$$\delta\ (mm) = \sqrt{\frac{\rho}{\pi f_0 \mu_r \mu_0}} = \sqrt{\frac{7.5}{\pi \cdot 175 \cdot 10^6 \cdot 1 \cdot 4 \cdot \pi \cdot 10^{-7}}} = 0.1042\ mm \qquad (2.3.109)$$

Where $\rho = 7.5\ \mu\Omega \cdot cm$ is the resistivity of brass [23], $f_0 = 175\ MHz$ the frequency used, $\mu_r = 1$ the relative permeability as brass is a non-ferrous metal [18], $\mu_0 = 4 \cdot \pi \cdot 10^{-7}$ is the permeability in free space. So, in our case, the cavity must have more than 10.42 μm.

In order to introduce the 175 MHz signal into the cavity, there are 3 ports. These ports are pick up antennas that are used to introduce the signal into the cavity, or used as feedback from the cavity's fields, Figure 2.32. The connector used is a 7/16". The assembly of the 7/16" pigtail is in Appendix B.


(a) *Inside of a cavity port*

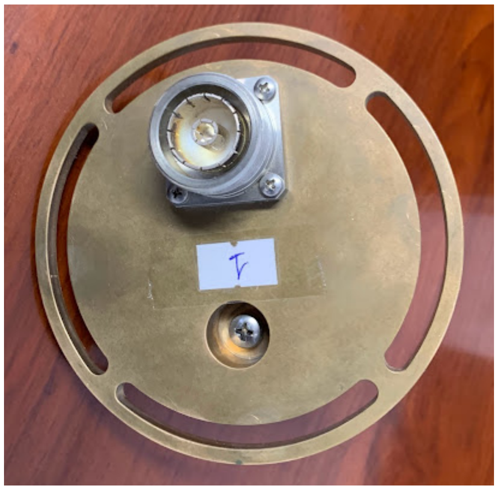
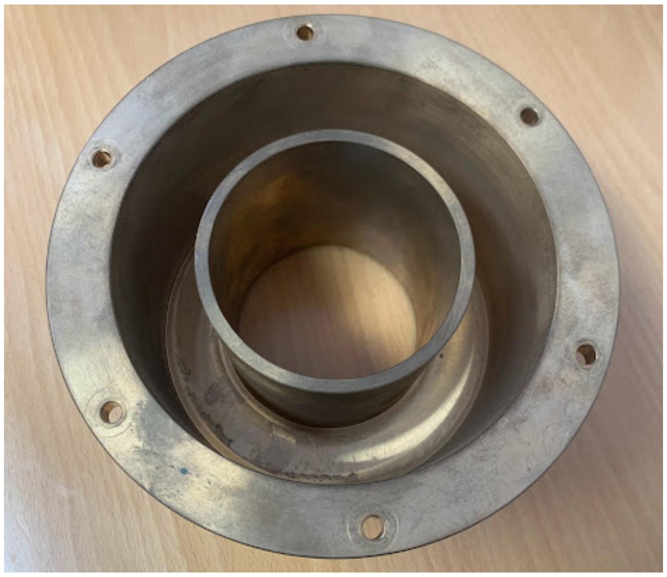

(b) *Outside of a cavity port*

Figure 2.32 – *Cavity port*

Apart from the three ports, the cavity has a nose cone in its centre. As it will be seen in Chapter 3, this is used to focalize the electric field in the centre, where the particle beam is passing through. The nose cone consists of two pieces, Figure 2.33, bolted to each side of the cavity.


(a) *Nose cone bolted part*

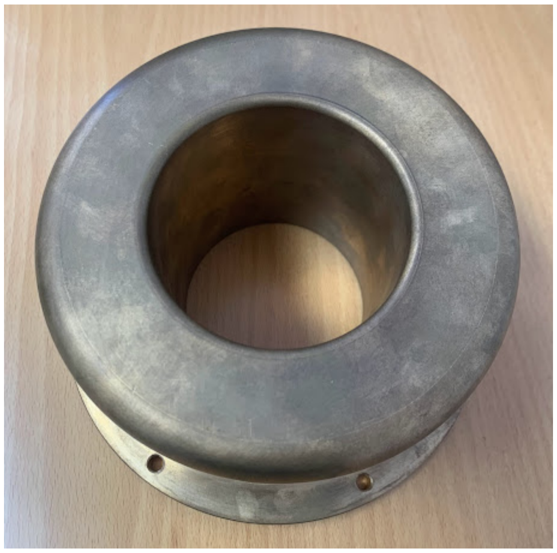
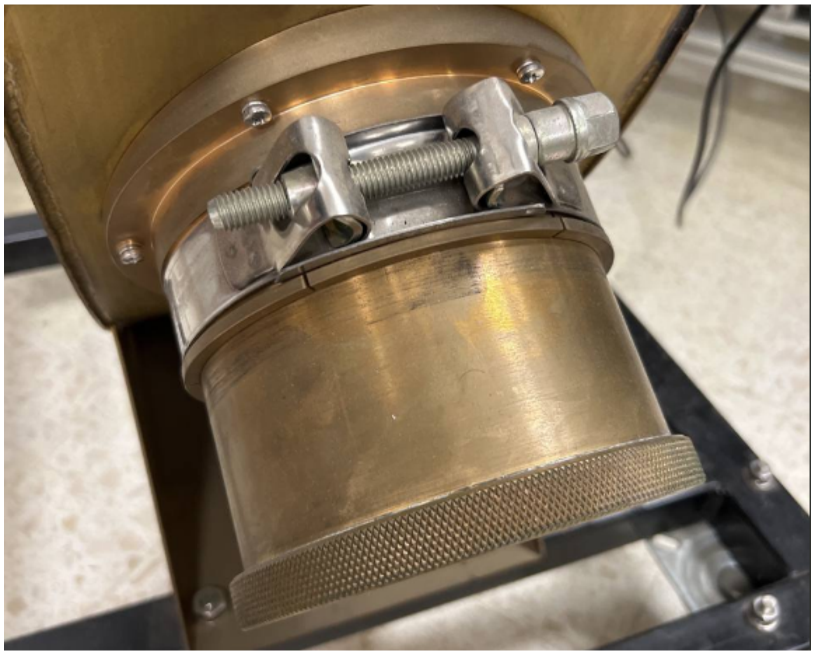
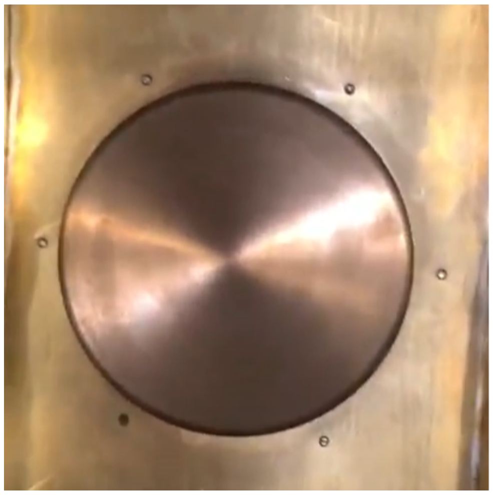

(b) *Inside part of nose cone*

Figure 2.33 – *Nose cone*

The resonant frequency of the pillbox cavity is 175 MHz. However, when a particle beam is introduced, this frequency changes as the particles are conductive material. In this case, on both sides of the cavity, there is a cylindrical tuner, that can be moved inside the cavity to decide how much of the brass cylinder to introduce inside it, Figure 2.34. As this changes the geometry of the cavity, it changes the resonant frequency. So, when the particles are accelerated, using these tuners, the cavity can return to the resonant frequency of interest.



(a) *Tuner from outside*



(b) *Tuner from the inside of the cavity*

Figure 2.34 –

2.3.2.2 Cavity characterization

Finally, the RF cavity will be characterized by its S parameters. The maximum adaptation will be achieved if only port is coupled with the inside fields. A port to be coupled, the antenna needs to be parallel to the electric field, Figure 2.35, port 3.

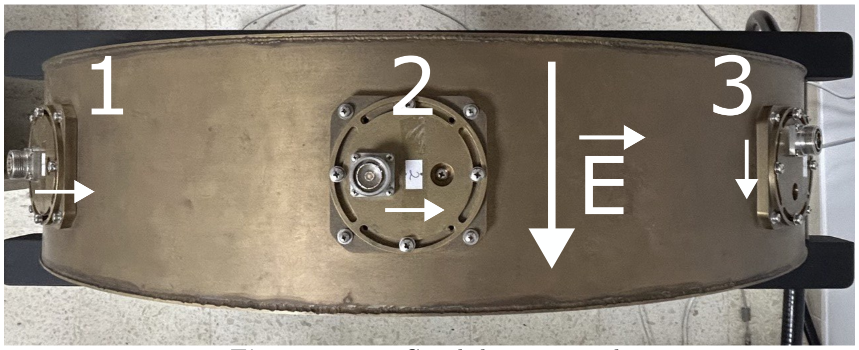


Figure 2.35 – *Coupled port example*

Proceeding with 3 port characterization, the results are very clear, Figure 2.36. The only port to present a resonant frequency is port 3 because it is the only one to have its reflected S parameter (S_{33}), below zero. In this case, the adaptation is around -14 dB.

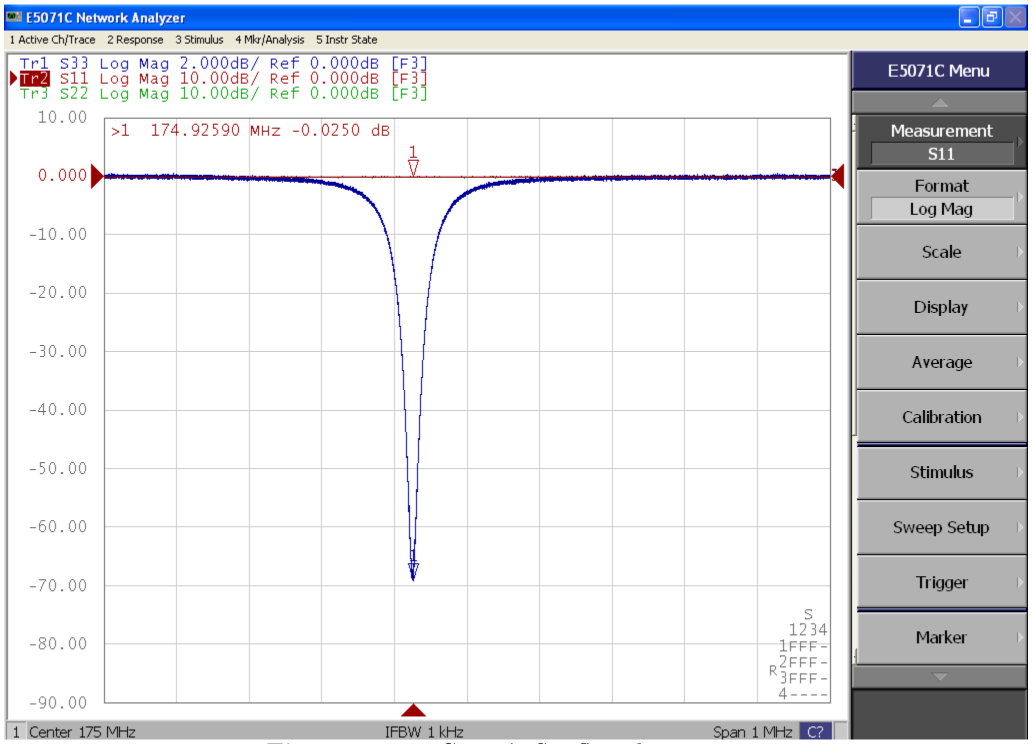
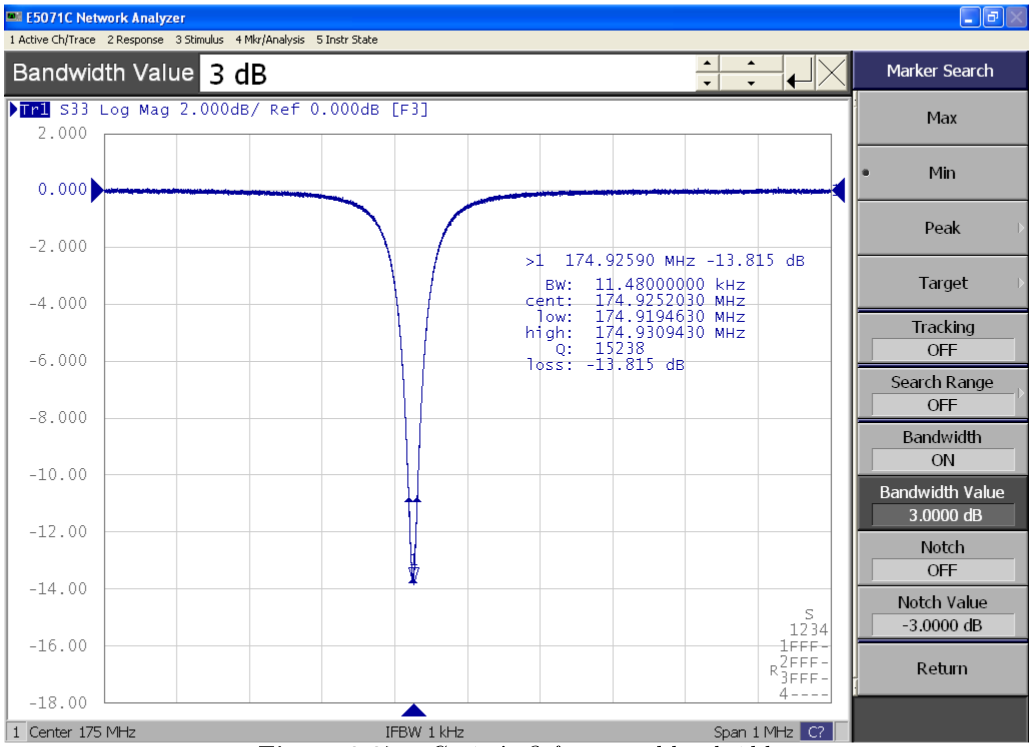

```
Tr1 S33 Log Mag 2.000dB/ Ref 0.000dB [F3]
Tr2 S11 Log Mag 10.00dB/ Ref 0.000dB [F3]
Tr3 S22 Log Mag 10.00dB/ Ref 0.000dB [F3]
>1  174.92590 MHz -0.0250 dB
1 Center 175 MHz          IFBW 1 kHz          Span 1 MHz
```

Figure 2.36 – *Cavity's S reflected parameters*

Moreover, the E5071C VNA, can calculate the bandwidth and the Q factor of the cavity, Figure 2.37. The Q factor of the cavity is 15238 and the bandwidth is 11.48 kHz.



```
Bandwidth Value 3 dB
Tr1 S33 Log Mag 2.000dB/ Ref 0.000dB [F3]

>1  174.92590 MHz -13.815 dB
  BW:   11.48000000 kHz
cent:   174.9252030 MHz
 low:   174.9194630 MHz
high:   174.9309430 MHz
   Q:   15238
loss:  -13.815 dB

1  Center 175 MHz          IFBW 1 kHz          Span 1 MHz
```

Tracking OFF
Search Range OFF
Bandwidth ON
Bandwidth Value 3.0000 dB
Notch OFF
Notch Value -3.0000 dB

Figure 2.37 – *Cavity's Q factor and bandwidth*

Chapter 3

Electromagnetic cavity simulations

3.1 Introduction

The main goal of this chapter is to simulate the BLAS' electromagnetically RF pillbox cavity to understand its behaviour and compare it with the experimental results obtained in Section 2.3.2. In order to simulate electromagnetic fields, the software Ansys HFSS will be used. This software gives the possibility to choose between different types of solutions, for this project, two will be used. The first one, Eigenmode, provides results in terms of eigenmodes (or resonant frequencies) of a given structure. The solver provides the resonant frequencies as well as the fields at a particular resonant frequency. On the other hand, HFSS Modal, This option yields S-matrix solutions that are expressed in terms of the incident and reflected powers of transmission line modes [3]. Moreover, the steps to replicate the simulations performed will be explained in the following sections. The simulated geometries will be designed in and imported into HFSS.

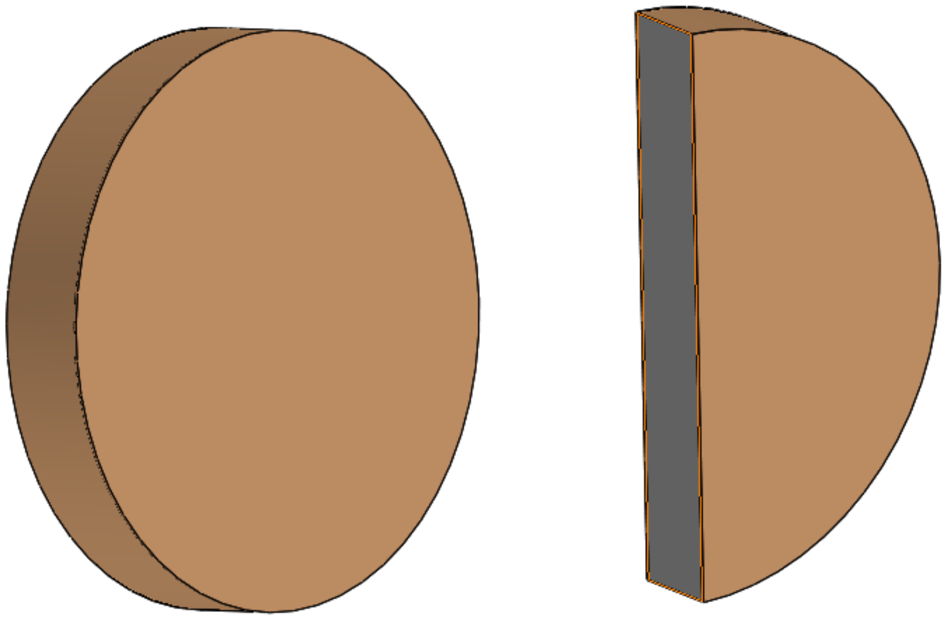
3.2 Simple cavity

In this section, the simplest cavity geometry is going to be simulated. It consists of a hollow cylinder, the other components from the cavity as the tuners, pickups, and nosecone will be added progressively to be able to understand the behaviour of each. The goal of this simulation is to find the resonant frequency of the cavity.

3.2.1 Theoretical cavity

Following Equation 2.3.98, the diameter needed to design a 175 MHz cavity is 1.31 meters, which does not coincide with the BLAS' RF cavity diameter of 1 meters. The reasoning behind this is the geometry of each cavity, in further sections the dependency between the resonant frequency and the extra components of the BLAS' will be discussed.

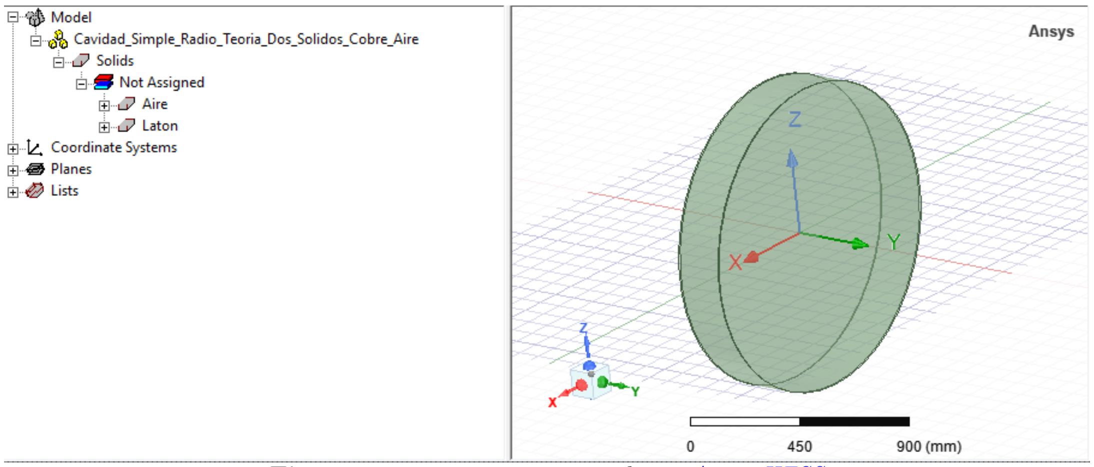
Firstly, the eigenmode simulation will be done with the 1.31 meter diameter cavity, to check Equation 2.3.98. As stated in Section 2.3.1.5, the cavity's width does not affect the resonant frequency's results. The real BLAS' cavity's length will be used, 200 mm. In this simulation, the steps followed to manage Ansys HFSS and the geometry will be explained. The first step is to open an HFSS file and import the geometry model from , *Modeler* → *Import* → *Select file*. In Figure 3.1, the model is depicted, as mentioned before, it consists of two solids, the cavity itself (brass) and the filling (air).



(a) *Whole view* **(b)** *Section view*

Figure 3.1 – *1.31 meter diameter cavity's geometry*

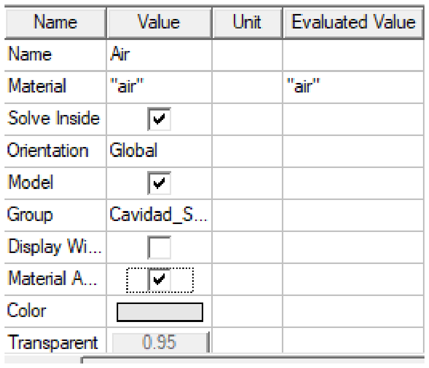
On the other hand, in Figure 3.2, a screenshot of the HFSS screen is shown, after importing the geometry.



Model
Cavidad_Simple_Radio_Teoria_Dos_Solidos_Cobre_Aire
Solids
Not Assigned
Aire
Laton
Coordinate Systems
Planes
Lists

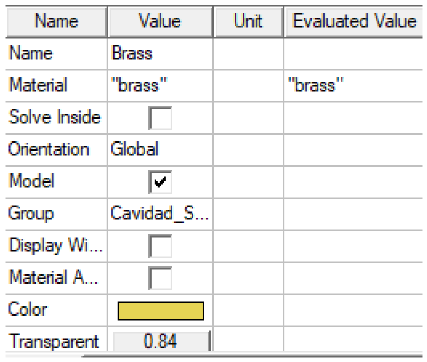
Figure 3.2 – *geometry imported into Ansys HFSS*

Then, the material of each body must be assigned. In this case, the outer body will be brass, and the inner body, air. In order to assign a material, *right-click on the solid → assign material and select the desired material*, to load its characteristics. The reason why two solids are needed is that Ansys HFSS requires a body capable of resolving electromagnetic fields inside it, a dielectric material. Ansys HFSS is not able to solve these fields inside conductors. In Figure 3.3, a screenshot of both material properties is shown.



Name	Value	Unit	Evaluated Value
Name	Air		
Material	"air"		"air"
Solve Inside	✓		
Orientation	Global		
Model	✓		
Group	Cavidad_S...		
Display Wi...	☐		
Material A...	✓		
Color	⬜		
Transparent	0.95		

(a) *Air properties*

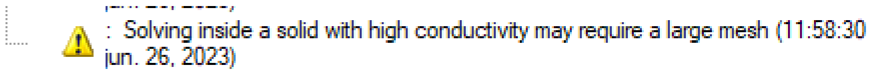


Name	Value	Unit	Evaluated Value
Name	Brass		
Material	"brass"		"brass"
Solve Inside	☐		
Orientation	Global		
Model	✓		
Group	Cavidad_S...		
Display Wi...	☐		
Material A...	☐		
Color	🟨		
Transparent	0.84		

(b) *Brass properties*

Figure 3.3 – *Materials properties in Ansys HFSS*

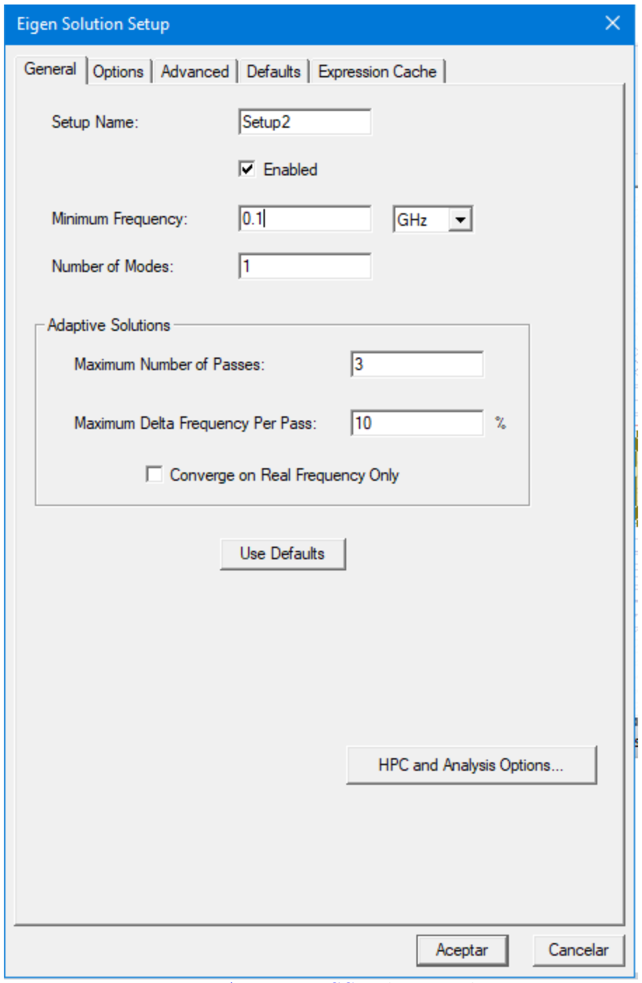
The solve inside option is only available on air (dielectric), and not in the brass (outer conductor). If this option is activated in a conductor, Figure 3.4 warning is shown. As it is only a warning, the simulation can be run, but it becomes much slower.



: Solving inside a solid with high conductivity may require a large mesh (11:58:30 jun. 26, 2023)

Figure 3.4 – *Ansys HFSS solve inside warning*

Once the materials are assigned, it is time to set the simulation wanted. Firstly, the type of simulation is selected, eigenmode in this occasion, *HFSS* → *Solution Type...* → *Eigenmode*. Then, the simulation parameters are fixed in *Simulation* → *Setup*, Figure 3.5. The number of modes to simulate is only the dominant one, the minimum frequency has to be set to a value below the expected resonant frequency, and the maximum number of passes and delta frequency are not critical but can make the simulation as precise as needed.


Eigen Solution Setup

General | Options | Advanced | Defaults | Expression Cache

Setup Name: Setup2

☑ Enabled

Minimum Frequency: 0.1 GHz

Number of Modes: 1

Adaptive Solutions

Maximum Number of Passes: 3

Maximum Delta Frequency Per Pass: 10 %

☐ Converge on Real Frequency Only

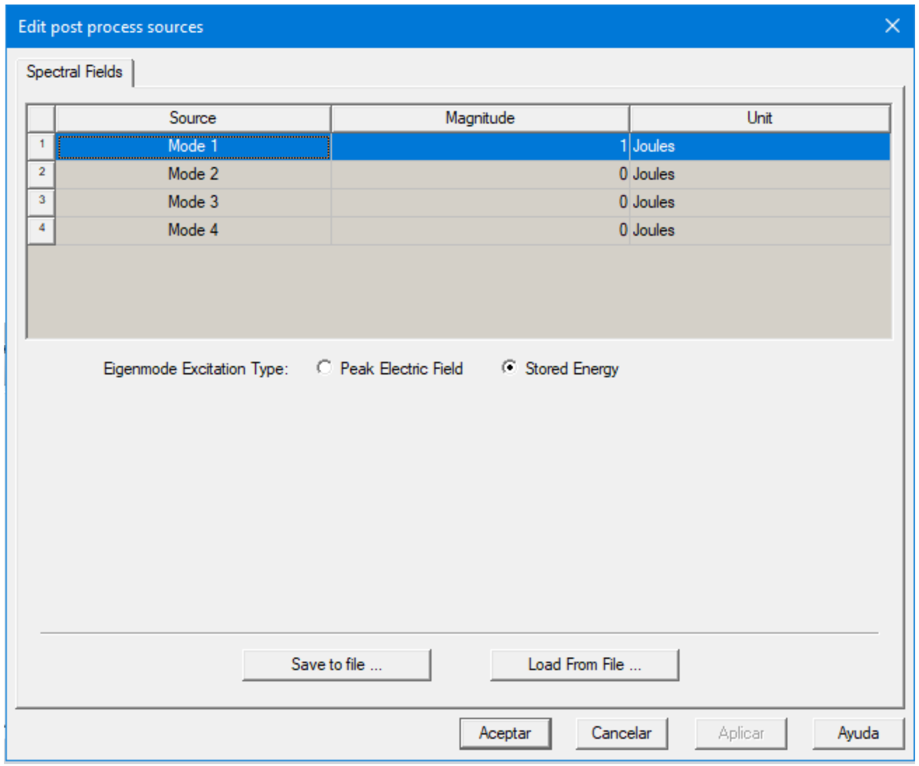
Use Defaults

HPC and Analysis Options...

Aceptar Cancelar

Figure 3.5 – *Ansys HFSS solve inside warning*

Before running the simulation selected, it is important to fix the energy of each mode of transmission. By default, mode 1 is given 1 J, and no energy to the rest of the modes, Figure 3.6. As the main goal is the resonant frequency, the modes energies will stay as default. This can be changed in *HFSS* → *Fields* → *Edit Sources*.


Edit post process sources

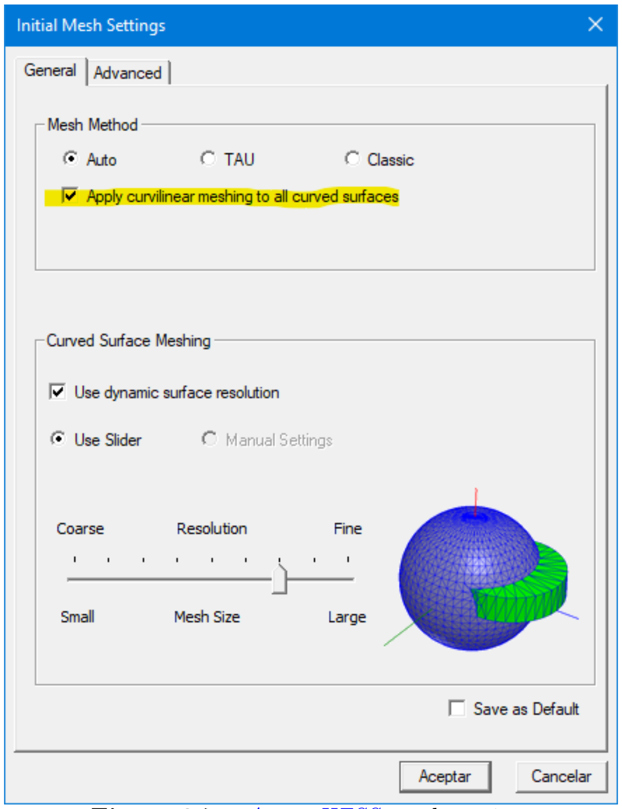
Spectral Fields

	Source	Magnitude	Unit
1	Mode 1	1	Joules
2	Mode 2	0	Joules
3	Mode 3	0	Joules
4	Mode 4	0	Joules

Eigenmode Excitation Type: Peak Electric Field [✓ Stored Energy]

Figure 3.6 – *Ansys HFSS eigenmode sources*

As in every geometry simulation software, a mesh has to be set, depending on the mentioned simulated geometry. By clicking on *Simulation* → *Mesh Settings*, the mesh can be modified, Figure 3.7. For a cylindrical cavity, it is recommended to click the *Apply curvilinear meshing to all curved surfaces* option, as the default mesh is planar. Moreover, as the geometry is still not complex, a fin mesh can be selected. The finer and larger the mesh is, the more time it will take the simulation to finish, but it will be more precise.



Mesh Method: [✓ Auto] TAU Classic

[✓] Apply curvilinear meshing to all curved surfaces

Curved Surface Meshing

[✓] Use dynamic surface resolution

[✓ Use Slider] Manual Settings

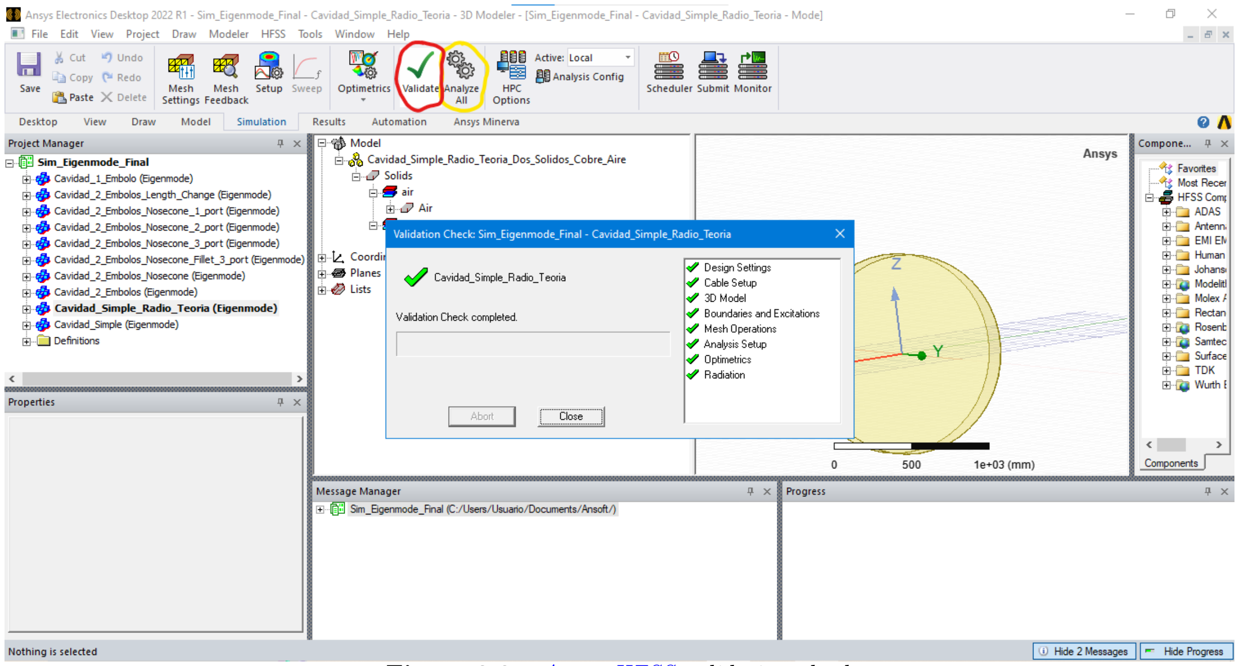
Coarse — Resolution — Fine

Small — Mesh Size — Large

[ ] Save as Default

Figure 3.7 – *Ansys HFSS mesh settings*

Finally, to run the simulation, HFSS needs to validate the different components that define the simulation as the design settings, 3D model, and setup, among others, click on *Validate*, red in Figure 3.8. After validating, the simulation starts after clicking on *Analyze All*, yellow in Figure 3.8.



Sim_Eigenmode_Final
- Cavidad_1_Embolo (Eigenmode)
- Cavidad_2_Embolos_Length_Change (Eigenmode)
- Cavidad_2_Embolos_Nosecone_1_port (Eigenmode)
- Cavidad_2_Embolos_Nosecone_2_port (Eigenmode)
- Cavidad_2_Embolos_Nosecone_3_port (Eigenmode)
- Cavidad_2_Embolos_Nosecone_Fillet_3_port (Eigenmode)
- Cavidad_2_Embolos_Nosecone (Eigenmode)
- Cavidad_2_Embolos (Eigenmode)
- **Cavidad_Simple_Radio_Teoria (Eigenmode)**
- Cavidad_Simple (Eigenmode)
- Definitions

Model
- Cavidad_Simple_Radio_Teoria_Dos_Solidos_Cobre_Aire
  - Solids
    - air
      - Air
- Coordi
- Planes
- Lists

Validation Check: Sim_Eigenmode_Final - Cavidad_Simple_Radio_Teoria

✓ Cavidad_Simple_Radio_Teoria

Validation Check completed.

- ✓ Design Settings
- ✓ Cable Setup
- ✓ 3D Model
- ✓ Boundaries and Excitations
- ✓ Mesh Operations
- ✓ Analysis Setup
- ✓ Optimetrics
- ✓ Radiation

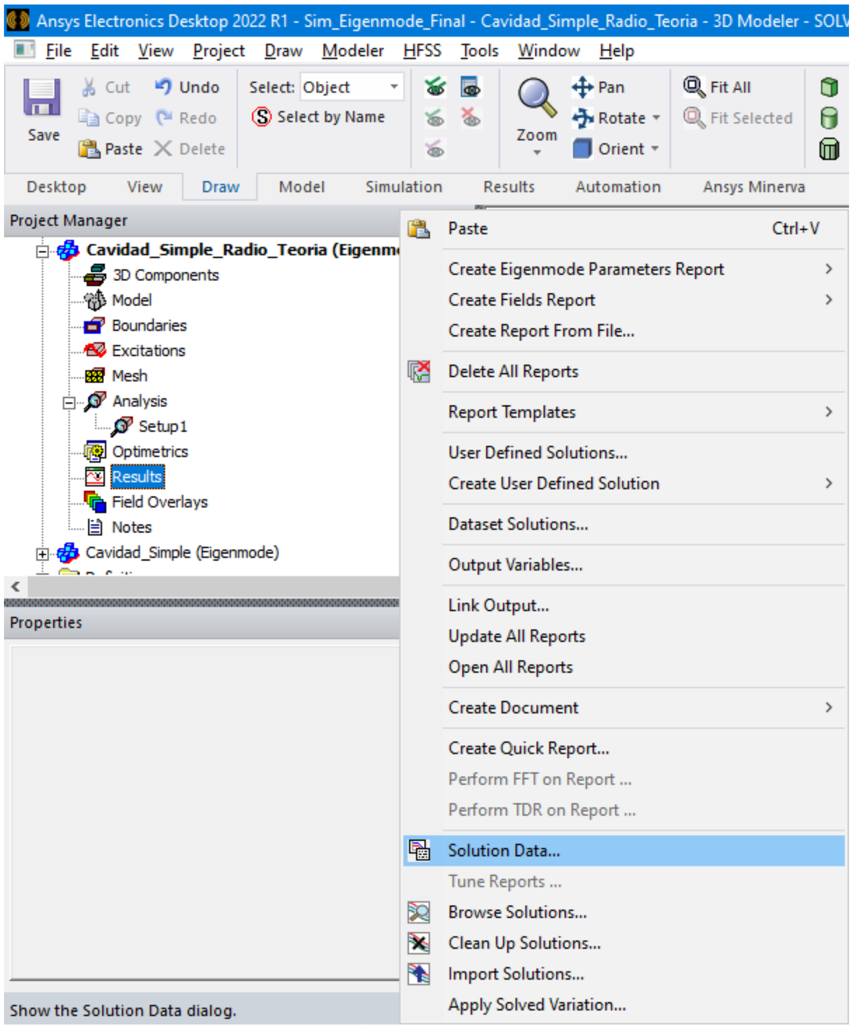
Abort Close

Sim_Eigenmode_Final (C:/Users/Usuario/Documents/Ansoft/)

Figure 3.8 – *Ansys HFSS validation check*

After the simulation is finished, results can be consulted by right-clicking on *Results* → *Solution Data...*,

Figure 3.9.



Show the Solution Data dialog.

Figure 3.9 – *Ansys HFSS validation check*

The final results are shown in Table 3.1. The resonant frequency of a cylindrical cavity of 1.312 meters diameter is 175.134 MHz. This frequency differs from the theoretical one by a little more than a hundred kHz, which could be due to different factors such as the mesh, the number of passes, or the delta frequency per pass. Moreover, the Q factor, explained in Section 2.3.1.6, is of 15895.2. The Q factor defines the relationship between the resonant frequency and the bandwidth of the cavity, Equation 3.2.1.

$$Q = \frac{f_{res}}{BW} \tag{3.2.1}$$

Eigenmode	**Frequency (MHz)**	**Q Factor**	**Bandwidth (kHz)**
Mode 1	175.134	15895.2	11.018

Table 3.1 – *Eigenmode results for theoretical cavity*

Furthermore, HFSS is able to plot the electromagnetic fields from inside the cavity. In Figure 3.10, the electric field is depicted. As expected, Figure 2.28, the electric field is parallel to the axis of the cavity and is more intense in the centre of it.

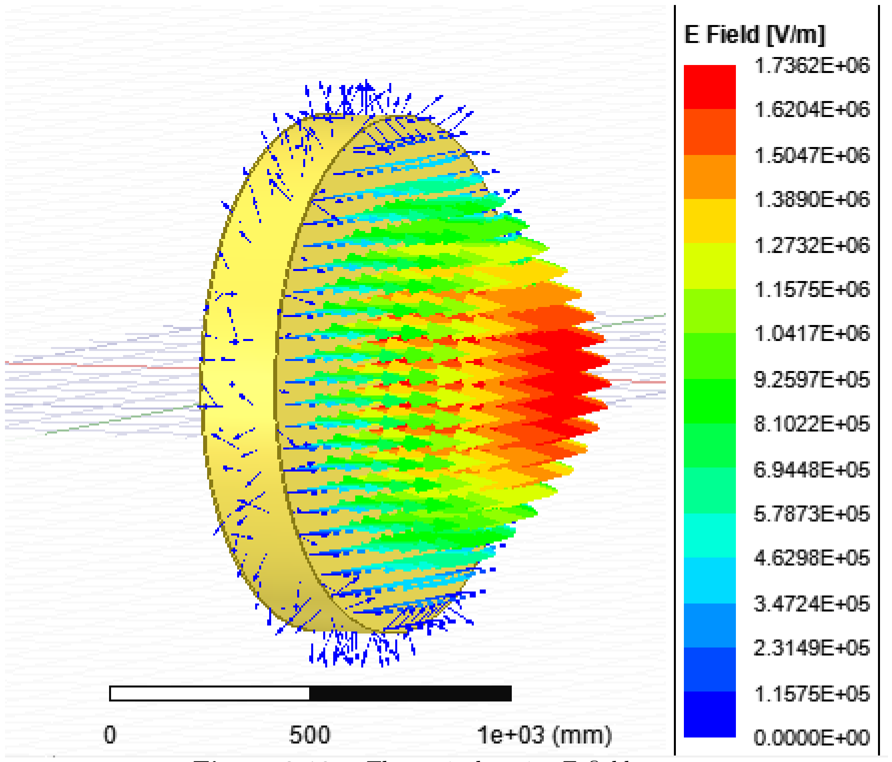


Figure 3.10 – *Theoretical cavity E field vector*

Furthermore, the magnetic field behaves as in Figure 2.28. This field is depicted in Figure 3.11.

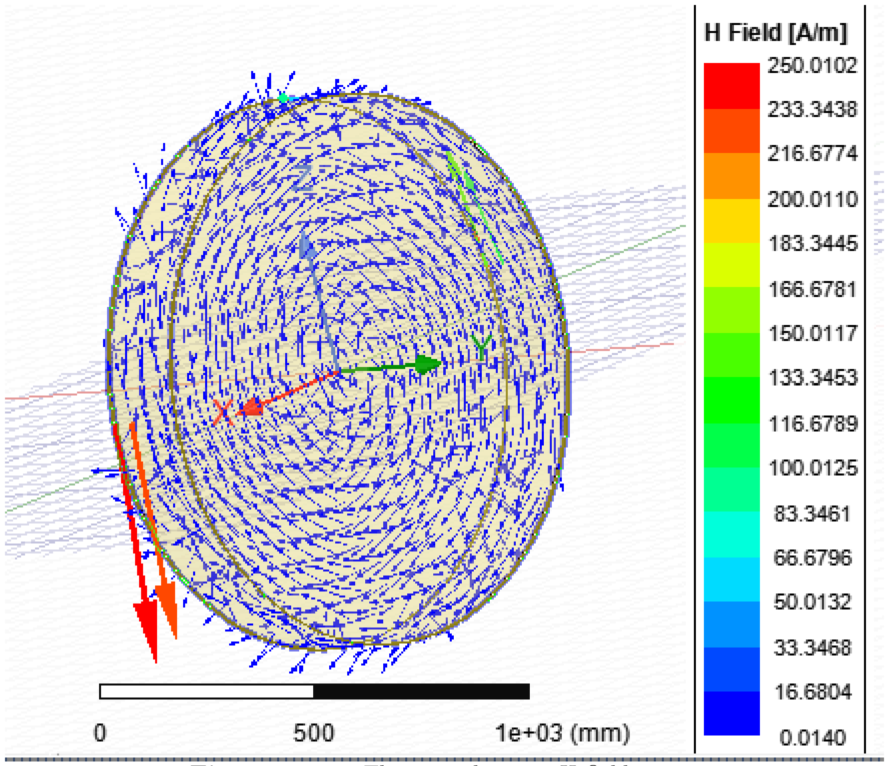


Figure 3.11 – *Theoretical cavity H field vector*

3.2.2 BLAS simple geometry cavity

In this subsection, the same simulation will be run as in the previous section, but changing the geometry's diameter to the one of the BLAS, 1 meter. As the previous steps to running the simulation have already been explained, from now, only the results will be shown.

Following Equation 2.3.98, a cavity of this diameter should have a resonant frequency of 229.66 MHz. The simulation's results are shown in Table 3.2. As happened before, the theoretical and the simulated frequencies are only separated by a hundred kHz. Moreover, the Q factor has increased with respect to the cavity with a bigger diameter, as the bandwidth did also.

Eigenmode	Frequency (MHz)	Q Factor	Bandwidth (kHz)
Mode 1	229.56	16578.1	20.834

Table 3.2 – *Eigenmode results for simple geometry BLAS cavity*

Finally, the E field vector and H field vector share the same direction as their counterparts in the theoretical cavity. It is relevant to note that the electric field intensity has increased, with respect to the previous geometry, Figure 3.12. For now, the main goal is to detect the behaviour of cavities depending on their geometry, not the exact field values. So, it can be concluded the smaller the cavity, the bigger the resonant frequency, Q factor, bandwidth, and the fields inside, having the same field source.

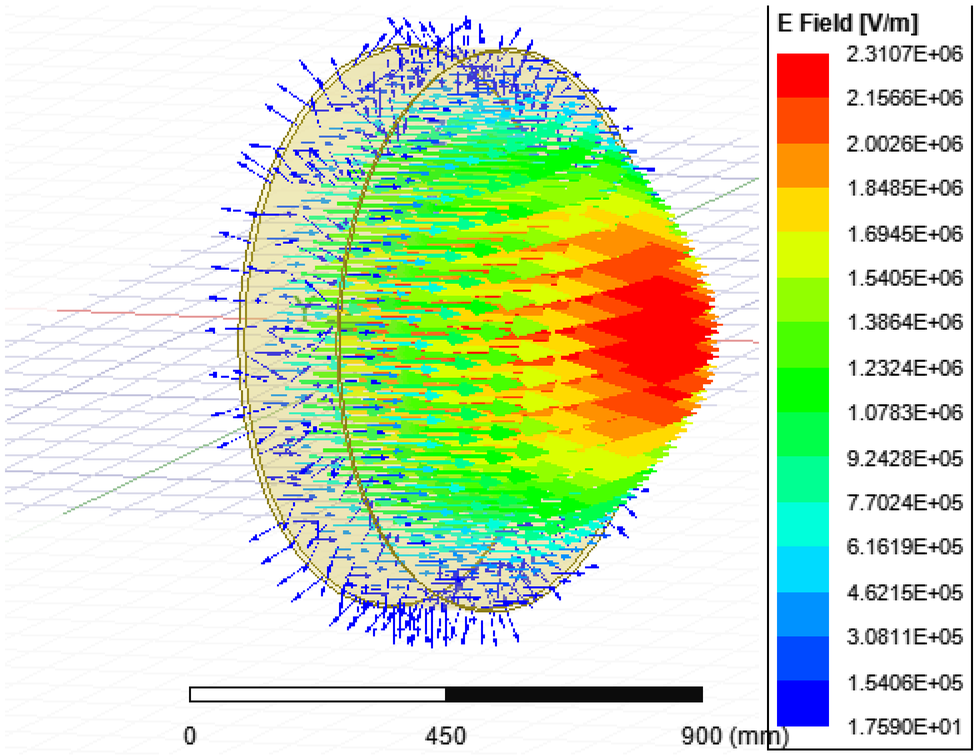
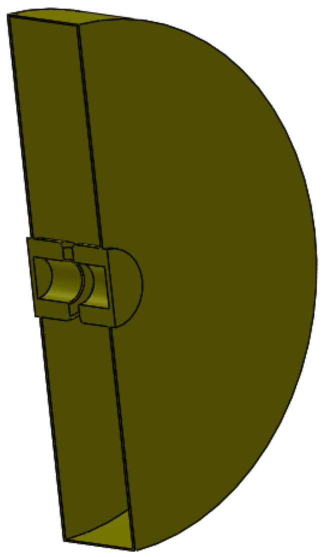


Figure 3.12 – *BLAS' simple geometry cavity E field vector*

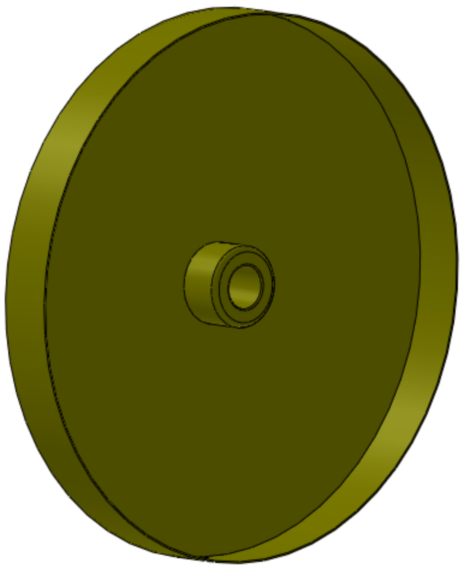
3.3 Nose cone

From this section and onwards, components will be added to the previous geometry, 1 metre diameter cavity, that will affect their resonant frequency, as geometry changes. The first component to add will be the nose cone. As mentioned in Section 2.3, the nose cone is used to focalize the E field in a certain part of the cavity, where the particle beam will enter, to maximize the acceleration.

The inside of the geometry to simulate is shown in Figure 3.13 and 3D Figure 3.14.



(a) *Nose cone cavity section view*



(b) *Nose cone cavity section view*

Figure 3.13 – *Two different nose cone cavity section views*

Figure 3.14 – *3D section view of the nose cone cavity*

A detailed view of the nose cone is depicted in Figure 3.15 and 3.16.

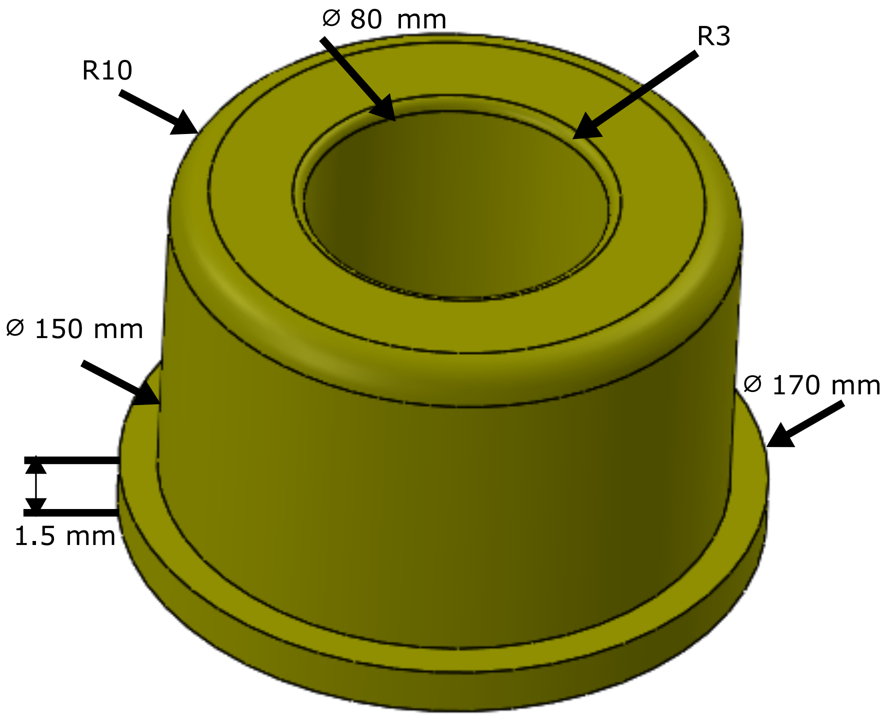


Figure 3.15 – *Nose cone detailed view*

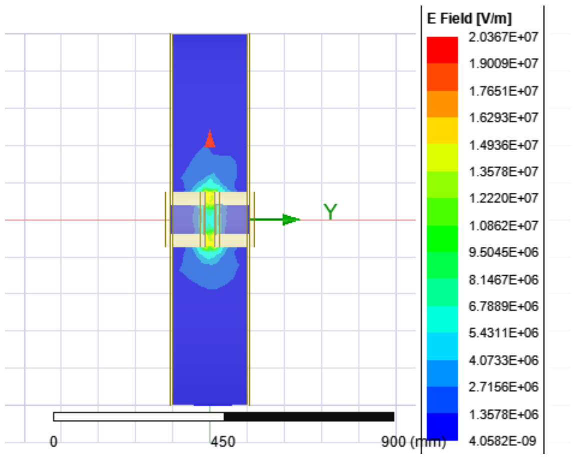
Figure 3.16 – *3D nose cone geometry*

Furthermore, the eigenmode results for the dominant mode are shown in Table 3.3

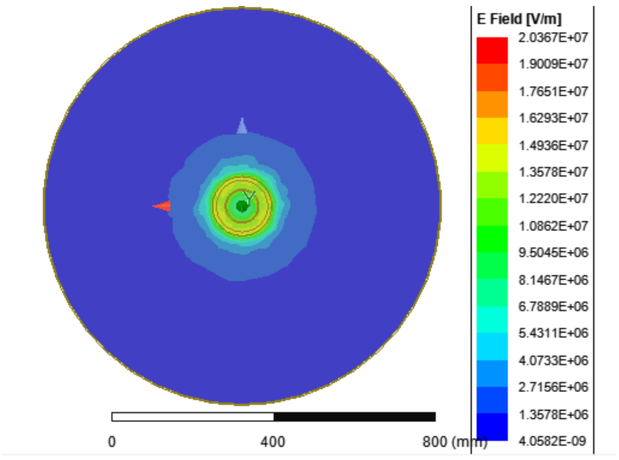
Eigenmode	**Frequency (MHz)**	**Q Factor**	**Bandwidth (kHz)**
Mode 1	172.828	13424.7	12.874

Table 3.3 – *Eigenmode results for nose cone cavity*

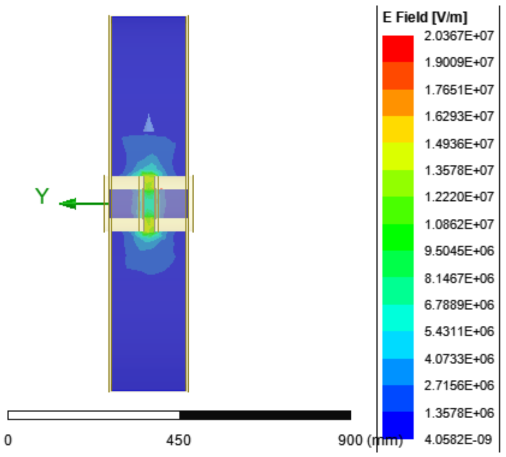
Moreover, to appreciate the behaviour of the electromagnetic fields in the cavity, and to check if the nose cone idea is correct, the electric field will be plotted in the XY, XZ, and YZ planes, Figure 3.17.



(a) *Nose cone cavity E field in XY*



(b) *Nose cone cavity E field in XZ*



(c) *Nose cone cavity E field in YZ*

Figure 3.17 – *Nose cone cavity E field in planes XY, XZ, YZ*

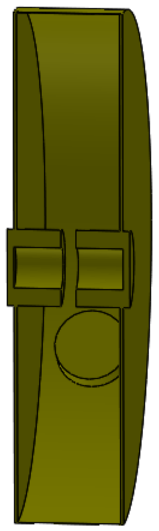
Compared with the E field in the nose cone less cavity, Figure 3.12, the E field is much bigger with the nose cone, nearly 10 times bigger. The zones with higher field are concentrated towards the beam particle zone. This will translate into faster accelerations when a particle enters the cavity.

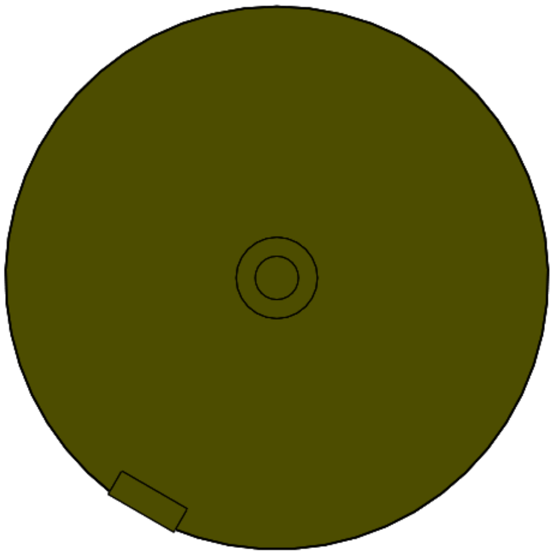
3.4 Tuner geometry cavity

As mentioned in Section 2.3, the cavity has two cylinders used to tune the cavity by changing its geometry, depending on the length of cylinder introduced.

3.4.1 1 tuner geometry

Firstly, the behaviour of only one tuner will be studied, by measuring the resonant frequency in a tuner length sweep. The frequency for the dominant mode will be noted for different tuner lengths. In Figure 3.18, a section view of the geometry is depicted for a tuner of 50 mm and 100 mm long. The diameter of the cylindrical tuner is 150 mm, as in the real cavity.


(a) *Section view of cavity with 1 tuner*


(b) *Section view of cavity with 1 tuner of 50 mm long*

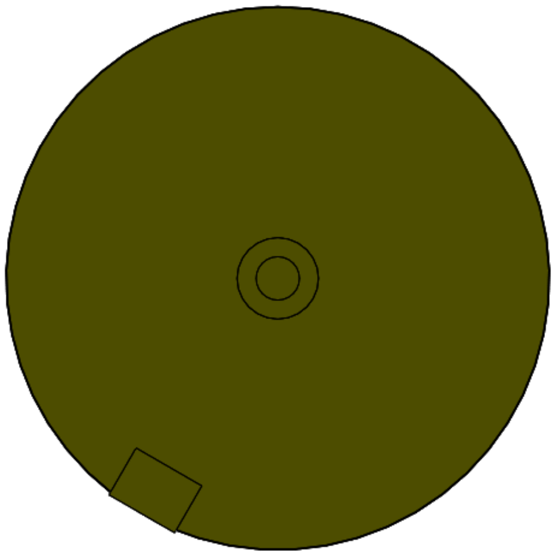

(c) *Section view of cavity with 1 tuner of 100 mm long*

Figure 3.18 – *Cavity geometry with 1 tuner*

The sweep was made from 50 to 200 mm, in steps of 50 mm. Results are shown in Table 3.4. The simulation indicates, the bigger the tuner, the higher the resonant frequency will be. So, if in the application the cavity is used, there is a need to tune towards a higher frequency, it can be achieved by introducing more of the cylindrical tuner inside the cavity.

Eigenmode	Tuner Length (mm)	Frequency (MHz)	Q Factor	Bandwidth (kHz)
Mode 1	0	172.828	13424.7	12.8743
	50	173.05	13146.4	13.163
	100	173.579	12747.3	13.617
	150	174.443	12233.8	14.259
	200	175.23	11573.7	15.140

Table 3.4 – *Eigenmode results for nose cone and one tuner cavity*

3.4.2 2 tuner geometry

In this subsection, a second cylindrical tuner will be added to the geometry, as in the real cavity. The geometry is depicted in Figure 3.19. Both tuners are separated by 120º.

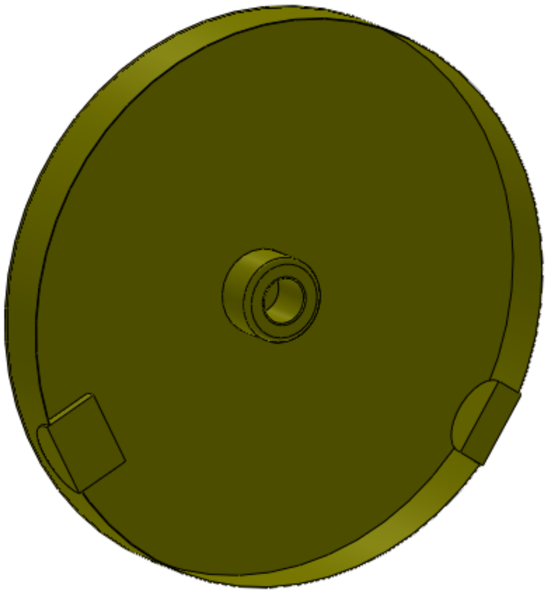


Figure 3.19 – *Cavity geometry with 2 tuners, 100 mm and 50 mm long, respectively*

The simulation results are shown in Table 3.5. This time, the sweep was done for the newly added tuner, while the previous one was fixed to 50 mm long. As happened with the other tuner, the longer the tuner, the higher the resonant frequency.

Eigenmode	Tuner Length (mm)	Tuner Length (mm)	Frequency (MHz)	Q Factor	Bandwidth (kHz)
Mode 1	0	0	172.828	13424.7	12.874
	50	50	173.396	12880.6	13,462
	50	100	174.239	12904.1	13.503
	50	150	174.847	12653.2	13.818
	50	200	175.668	12330.3	14.247

Table 3.5 – *Eigenmode results for nose cone and two tuner cavity*

3.5 Pick up ports

In this section, the last component will be added, the pick up ports. In this case, both simulation types will be realised, the eigenmode and modal types. For the eigenmode type, the pick up ports will be introduced progessively, to study the impact on the resonant frequency. For the modal simulation, the geometry to simulate is going to be the one with the three ports, because is the real geometry and the one to have the 2 kW inserted.

3.5.1 1 port geometry - Eigenmode

In this geometry, only one port will be added. The mentioned geometry is depicted in Figure 3.20. The size of the pick up is the same as the one the real cavity uses and there are no tuners inside. The results for the resonant frequency are shown in Table 3.6.

Figure 3.20 – *1 port cavity geometry*

Eigenmode	Tuner Length (mm)	Tuner Length (mm)	Frequency (MHz)	Q Factor	Bandwidth (kHz
Mode 1	0	0	172.935	13383.4	12.921

Table 3.6 – *Eigenmode results for 1 port cavity*

Adding a port to the cavity, makes the resonant frequency to go up a hundred kHz. Easily tuneable using the tuners.

3.5.2 2 port geometry - Eigenmode

The analysis for this geometry will be the same as the previous one. In Figure 3.21, the two port cavity is depicted.

Figure 3.21 – *2 port cavity geometry*

The results for the eigenmode simulation are shown in Table 3.7. The addition of another port, kind of counters the increase in the frequency of resonant value, caused by the addition of the first pick up port.

Eigenmode	**Tuner Length (mm)**	**Tuner Length (mm)**	**Frequency (MHz)**	**Q Factor**	**Bandwidth (kHz)**
Mode 1	0	0	172.875	13352.2	12.947

Table 3.7 – *Eigenmode results for 2 port cavity*

3.5.3 3 port geometry - Eigenmode

Finally, the last geometry will be simulated. This 3 port geometry is the one that resembles better the geometry of the real cavity form BLAS, Figure 3.22.

Figure 3.22 – *3 port cavity geometry*

Results are shown in Table 3.8. As happened in the 1 port cavity geometry, the resonant frequency increases in around hundred of kHz. It seems like each new port counters the behaviour of the previous one. This coulb be interesting for future works, as it is not going to be developed in this project.

Eigenmode	**Tuner Length (mm)**	**Tuner Length (mm)**	**Frequency (MHz)**	**Q Factor**	**Bandwidth (kHz**
Mode 1	0	0	172.949	13341.1	12.963

Table 3.8 – *Eigenmode results for 3 port cavity*

3.5.4 3 port geometry - Modal

The geometry to simulate is the same as the previous subsection, but not the type of simulation. Implementing a modal simulation means some modifications are needed to run it. As mentioned at the beginning of this chaper, a modal simulation is the one that calculates the S parameters of a passive component. So, if S parameters are calculated, the geometry needs RF ports, and, minimum, an input signal. The steps to run a modal simulation are the following.

Firstly, the simulation must be changed, *HFSS* → *Solution Type...* → *Modal*. Then, the excitations must be set. HFSS referes as excitations to the ports of the device. An excitation is set by selecting a face from RF connectors, and this face must be of a material with the solve inisde option selected, not he proper conductor that will transmit the signal. In Figure 3.23 and example is depicted. The face of the connector's dielectric is selected (pink).

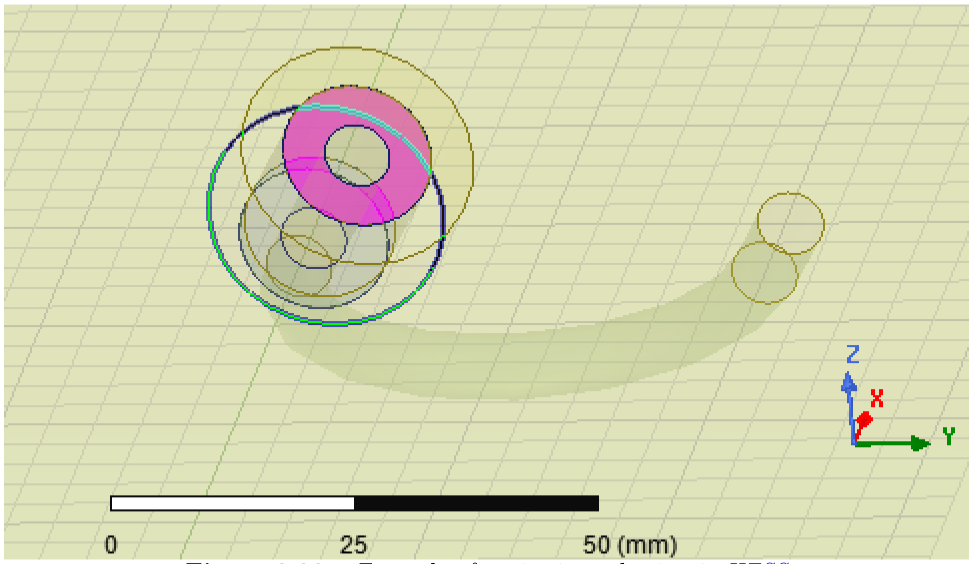
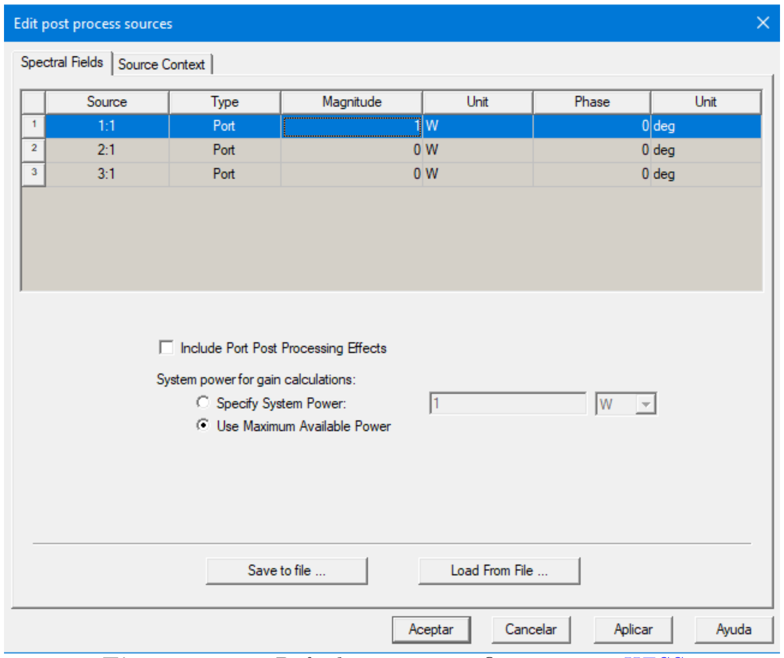


Figure 3.23 – *Example of excitation selection in HFSS*

While the face is selected, right click on it *Assign Excitation* → *Port* → *Wave*. Then, it is only necessary to give it a name. Finally, the input signals have to be set, by *HFSS* → *Fields* → *Edit Sources...*, Figure 3.24.


Edit post process sources

Spectral Fields | Source Context

	Source	Type	Magnitude	Unit	Phase	Unit
1	1:1	Port	1	W	0	deg
2	2:1	Port	0	W	0	deg
3	3:1	Port	0	W	0	deg

☐ Include Port Post Processing Effects

System power for gain calculations:

○ Specify System Power: 1 W

● Use Maximum Available Power

Figure 3.24 – *Default sources configuration in HFSS*

Once the steps to prepare the geometry for a modal simulation are clear, the 3 port geometry will be simulated. In this geometry, only one of the three pick up ports will be coupled, one will be parallel to the electric field, or perpendicular to the magnetic field, and the other two will be perpendicular to the electric field, or parallel to the magnetic field, Figure 3.25.

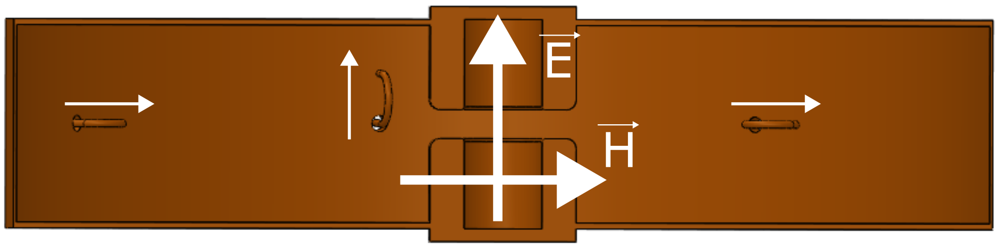


Figure 3.25 – *Pick up orientation, one coupled (centre)*

Next, the three ports are set as wave ports. The coupled port will be number 1. Moreover, the input power will be of 2 kW in port 1, as is the one coupled, the port that will introduce the RF signal into the cavity to create the electromagnetic fields. In order to calculate the S parameters a frequency sweep must be defined. In this case, as we know from the eigenmode result, the resonant frequency must be around 172.949 MHz, so the sweep set is between 171 and 175 MHz.

To be able to prove, only port 1 is coupled, the three S reflection parameters (S_{11}, S_{22}, S_{33}) have been depicted, Figure 3.26. The results are the expected ones, as the only port to present a resonant frequency is port 1, the coupled one. This means, if the 2 kW is connected to port 2 or 3 the signal will be totally reflected, as their reflection S parameters are null. Moreover, a resonance of -12 dB means the 25 % of the signal will be reflected. Considering the cavity will work with high power, the ports could be redesigned to maximize the transmitted power, and not damage the previous RF components with the reflected power.

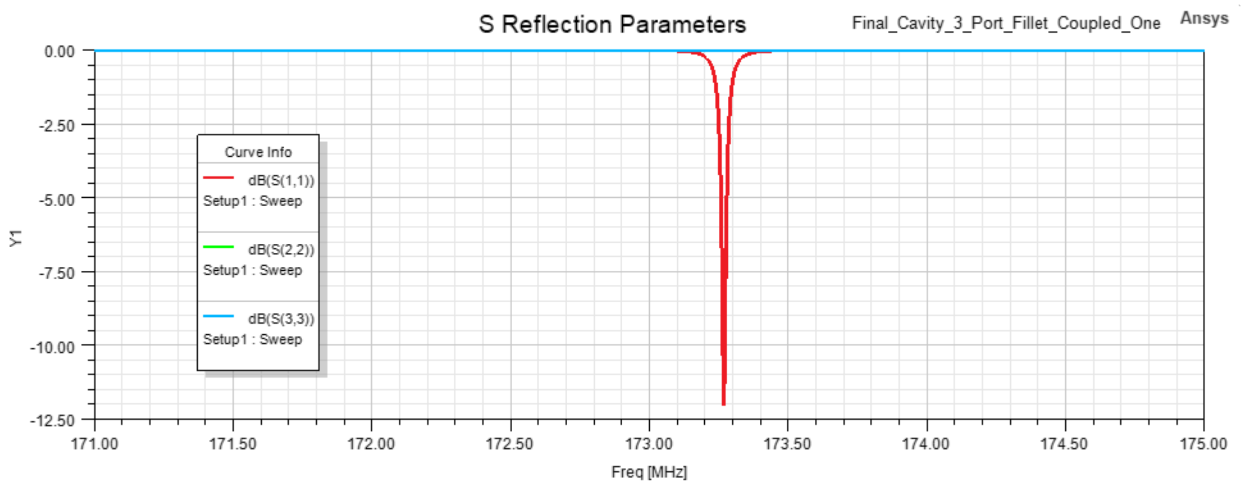
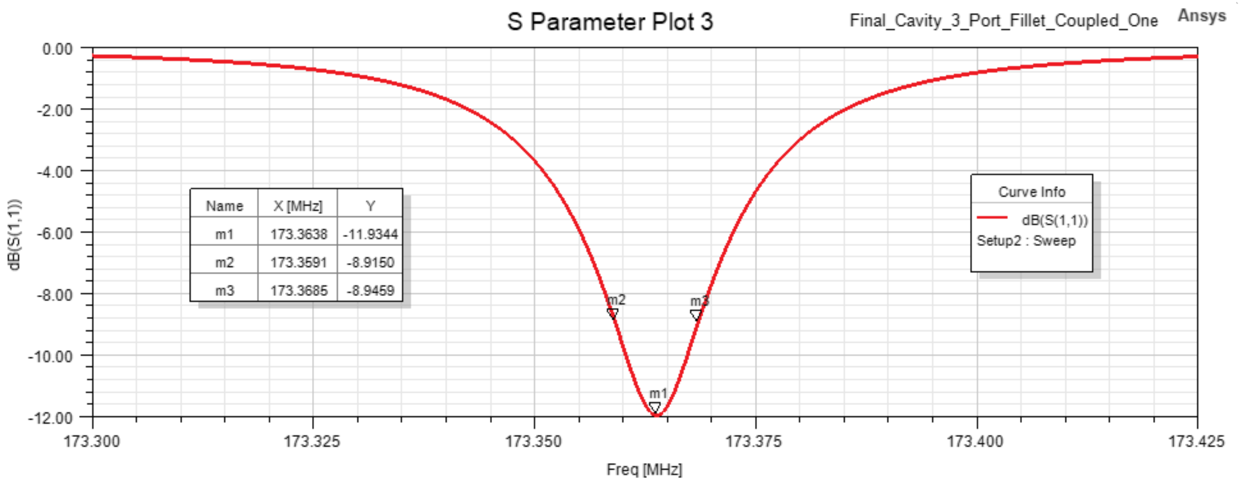


Figure 3.26 – *3 port cavity reflection S parameters*

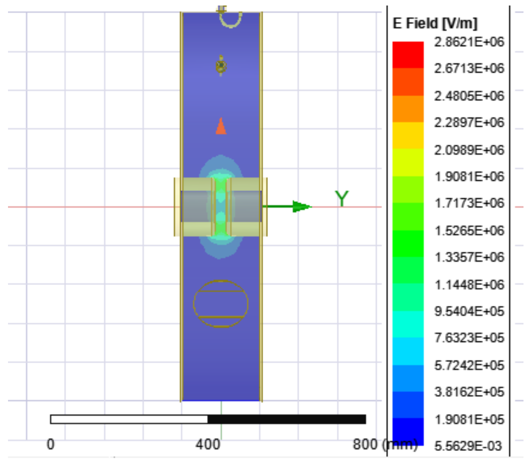
Moreover, the cavity's bandwidth is depicted in Figure 3.27. The resonant frequency of the cavity in modal analysis is of 173.3638 MHz, that differs with the one obtained in the eigenmode results, 172.949 MHz 3.8. By being both geometries the same one, it was expected for both resonant frequencies to be the same. Despite not having any explanation in the HFSS of why this can occur, the possible answer is the way the signal is entering the cavity. As for the eigenmode simulation there is no clear input port, the software introduces the RF signal "magically" inside the cavity. Whereas, in the modal simulation, an excitation has to be defined and is where the RF signal is transmitted. Furthermore, the -3 dB bandwidth is of 9.4 kHz, calculated with markers m2 and m3.



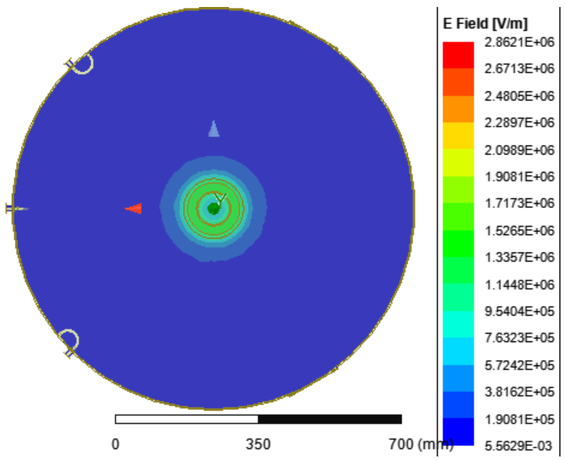
Name	X [MHz]	Y
m1	173.3638	-11.9344
m2	173.3591	-8.9150
m3	173.3685	-8.9459

Figure 3.27 – *3 port cavity* S_{11}

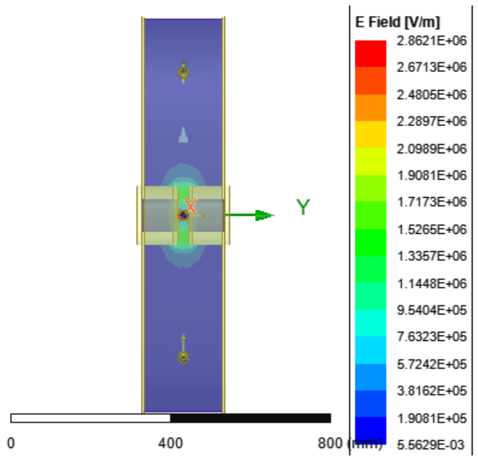
Finally, the simulated electric field is depicted in Figure 3.28. As in previous occasions, the electric field will be plotted for planes XY, XZ and YZ. By the colour chart, the electric field expected in the particle beam path with 2 kW of RF signal is around **6 MV/m**. As stated in section 3.3, the nose cone help to intensify the electric field towards the centre of the cavity, but does not focalize exactly in the particle beam. So, the electric field could be higher with a better design.



(a) *Nose cone cavity E field in XY*



(b) *Nose cone cavity E field in XZ*



(c) *Nose cone cavity E field in YZ*

Figure 3.28 – *Final cavity E field in planes XY, XZ, YZ*

3.6 Conclusions

After all the simulations and results provided, this section will summarize the studied cavity behaviour. Foremost, the BLAS' cavity diameter does not coincide with the diameter resultant from Equation 2.3.98 for 175 MHz cavities. It has been proved that the addition of different components inside the cavity alters the resonant frequency of it.

Moreover, the introduction of a nose cone, is helpful to focalize and intense the electric field around the centre of the cavity. However, this nose cone design could be upgraded to have the maximum field values in the particle beam path. Furthermore, to counter the conductive behaviour of the charged particles from the beam, and the change in the cavity's resonant frequency that causes, the cylindrical tuners are used. By introducing only 50 mm of tuner, the frequency changes in around half MHz.

Finally, the introduction of pick up ports to insert the RF signal and use them as feedback from the cavity, also varies the resonant frequency. This variation is not as relevant as the previous components, but has to controlled. The simulated cavity has an adaptation on its coupled port of -12 dB and a bandwidth of 9.4 kHz. This information is essential in order to design the RF components that follow or precede the cavity, as the expected reflected power is known, and the bandwidth for the input signal. In order to know the acceleration the cavity provides, the electric field has been calculated inside the cavity. With an RF signal of 2 kW, the particles will experiment an electric field of around **6 MV/m**. In Section 2.3.2.2, the characterization measured a bandwidth of around 12 kHz and a Q factor of somewhere near 15000. These results are quite similar to the ones obtained in Table 3.8. So, it can be concluded the simulations done are pretty accurate with the practical results.

Chapter 4

Controller

4.1 Introduction

In this chapter, the system control design will be approached. The main goal is to interconnect and monitor all the equipment available in the GranaSAT laboratory. In Figure 4.62, the desired final system control diagram is depicted. The measurement equipment is formed by the Agilent Technologies N9020A signal analyzer and the Anritsu MS2830A signal analyzer that can also act like a signal generator. Both pieces of equipment will be in charge of analysing the signals coming from the SSPA drawer's directional couplers. Moreover, in Figure 4.62, an LLRF appears. An LLRF, is a device in charge of generating a low-level RF signal, like and . This LLRF was going to be designed as part of another different parallel project from the GranaSAT laboratory. However, the mentioned project was delayed and as the LLRF the Anritsu MS2830A signal analyzer will be used. As seen in section 2.3.2.2, a small frequency drift, and the behaviour of the system could drastically change. To avoid these drifts, a synchronization network has been implemented, to have all the measurement equipment of the laboratory using the same reference signal for their local oscillator. The implementation is explained in Appendix C.

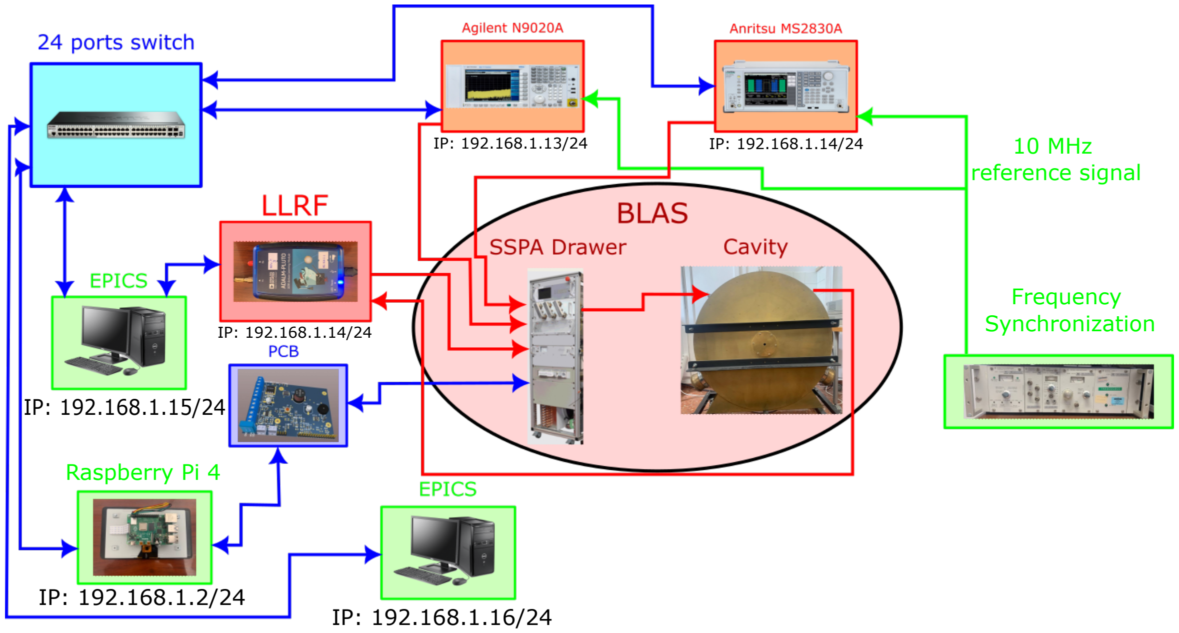


Figure 4.1 – *System control block diagram*

Moreover, in order to manipulate the BLAS control signals, presented in Section 2.2.7, and the RF signals coming from the directional couplers, two PCBs will be designed. Both PCBs will be controlled by a Raspberry Pi 4 with a 7" touchscreen. Furthermore, there are two PCs with EPICS running, Input/Output Controller (IOC)s or Client WorkStation (CWS), as will be explained in Section 4.4.2. Apart from these two PCs, other PCs or laptops can be added to the laboratory's LAN. As a resume, the idea is to monitor both signal analysers, the BLAS control signals and the power from the directional couplers (via the Raspberry Pi 4), using an EPICS IOC and CWS tool (Control System Studio (CSS)).

4.2 Electronic Instrumentation

One of the requirements for this project was the implementation of libraries to remotely control the various pieces of measurement equipment from the GranaSAT laboratory, and the documentation behind these libraries. The documentation will be crucial for further students to be able to get this code and use it without issues. To develop these libraries and test them, Jupyter Notebook has been used. Jupyter Notebook is a great tool when it comes to documentation, as in the same document it is possible to mix Python code and Markdown text. These libraries are apart from the EPICS implementation and can be run in any Python environment. The pieces of equipment to be remotely controlled using Jupyter Notebook are:

- Agilent Technologies N9020A Signal Analyzer (Appendix E.1)
- Anritsu MS2830A Signal Analyzer (Appendix E.2)
- Agilent Technologies MSO-X-4140A Oscilloscope (Appendix E.3)
- Agilent Technologies E5071C VNA (Appendix E.4)
- Marconi Instruments 2022D Signal Generator (Appendix E.5)

The common language to communicate with these types of pieces of equipment is Standard Commands for Programmable Instruments (SCPI). Using Python libraries, these SCPI commands can be sent to the devices. For instance, the *print("Hello world")* equivalent to SCPI would be **IDN?*. This command asks the device to identify itself. The Python libraries depend on the communicating protocol to use, Ethernet or GPIB. The documentation for the code implemented is in the appendixes referenced in the list above.

4.3 PCB

In this section, the PCBs designed are going to be presented. Two PCB are going to be assembled: the Main PCB and an RF Voltmeter. The first one will be controlled by the Raspberry Pi, and its main goal is to process the various signals coming from the SSPA drawer. On the other hand, the RF voltmeter will be used to be able to measure the signal power coming from the directional couplers, without needing a signal analyzer.

4.3.1 Design

This section will include all the thinking and decisions behind the design process for the PCBs from this project. Once we know the different functionalities our control algorithms need, it is possible to start organizing the different parts that will form both PCBs. A PCB design process has essentially two parts: **circuit design** and **modules placement**. Firstly, all the circuits are planned and then, are placed and routed into a PCB. Both process will be done in Altium Designer® 21.

The circuit design consists of implementing the circuit needed for the modules to work on SCH files. In these files, the main goal is to describe the circuit with the components we are going to implement. However, not only the circuits are shown, also information about the different modules from their Datasheets and the reasons behind the circuit planning.

Once the circuits have been implemented, we need to place them on our PCBs. This is not a banal process, since we have to distribute the available space to have the most functional and efficient design. The files used in this process are PCB ones, and are linked to the SCH files. This link is done via the footprint of each component. A footprint is the arrangement of pads or through holes used to physically attach and electrically connect a component to a PCB. The land patterns on a circuit board matches the arrangement of leads on a component. The way Altium Designer® 21 associates the SCH symbols to the footprint is via pin numbers and pad numbers, respectively. Consequently, it is especially important to maintain the pin enumeration on both the SCH symbols and on the footprint.

Furthermore, as the footprint is the physical representation of the element, the size must be similar to the real element. For implementing each module footprint, we should follow the datasheet recommended PCB land pattern. As an example of all this process, we will refer to Figure 4.2 for the SCH symbol of the ADC ADS115 from Texas Instruments. The footprint associated to this symbol can be observed in Figure 4.3.

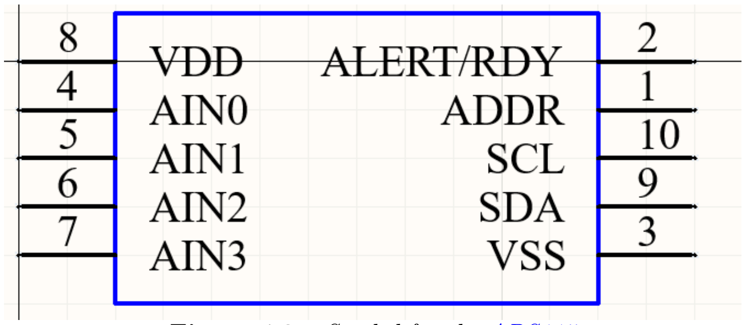


Figure 4.2 – *Symbol for the ADS115*

Note that for implementing the footprint we have followed the PCB land pattern on the Datasheet (Figure 4.4) [13]. The measurements unit from Figure 4.3 is mills, and the one from Figure 4.4 is mm. Doing the conversion from one unit to the other one, we can ensure the footprint is correct. In most of the cases, instead of having to create a new footprint from scratch, it is possible to find them on the manufacturer's or distributor's website.

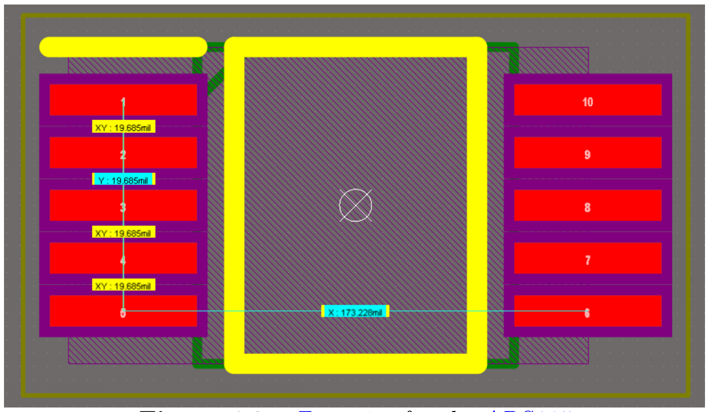


Figure 4.3 – *Footprint for the ADS115*

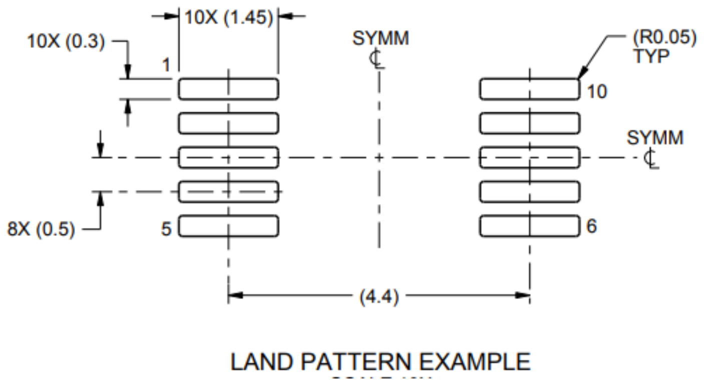


Figure 4.4 – *Land pattern for the ADS115 from its Datasheet*

The last thing we have to mention it that Altium Designer® 21 allow us to observe the 3D model of our design, therefore for each module we need to also include the 3D model (Figure 4.5). This is very useful to be able to have an idea of how the final PCB will look like, and the space the component will occupy.

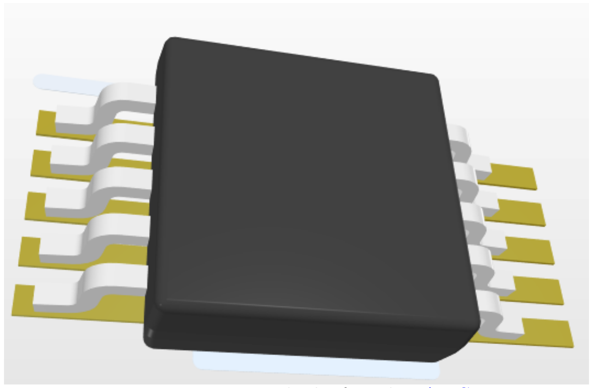


Figure 4.5 – *3D body for the ADS115*

4.3.1.1 Steps for a good PCB design

Just before starting the module placement phase, it is necessary to have clear and take into account different aspects that could modify the way of designing your PCB.

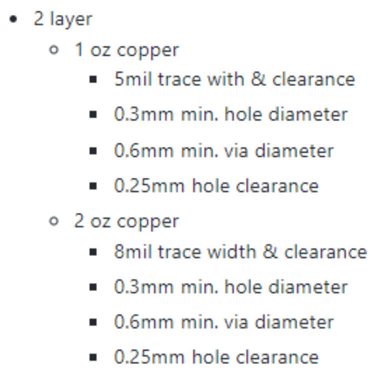
- **PCB production technology:** The main item we need to consider before starting to design our PCB design process is the technology we are going to be using to do such a thing. In our case, we can choose between manufacturing in the GranaSAT laboratory or having it produced in China. Since the elements we are going to use are quite restrictive in terms of size and routing, we will proceed to send it to be produced in China. This decision has been made taking into account the fact that in the laboratory we do not have the possibility of finishing the PCB with a solder mask layer, therefore the space between elements for manual soldering should be large enough not to risk making any short circuit. To sum up, our production is going to be made by JLPCB manufacturer [15].

- **Capabilities:** Once we have decided the manufacturer of our PCBs, it is important to know exactly how their production work and what of kind of restrictions they have, this is called capabilities. In our case, JLPCB's capabilities can be checked in this link. Manufacturing our PCBs in China gives us the opportunity to be more flexible as their machines can produce PCBs of different thicknesses with better tolerance, dimensions and other characteristics that the machine from our laboratory cannot. The final features chosen are the following, stated in Table 4.1.

Layer Count	2
PCB Qty	5
Material	FR-4 Standard Tg 130-140C
Dielectric Constant	4.5 (double-side PCB)
Dimension	Rounded 60x60 Rectangular: 100x100
Board Thickness	1.6 mm
Thickness Tolerance	10% 0.1%
Finished Outer Layer Copper	1 oz (35 μm)
Finished Inner Layer Copper	0.5 oz (17.5 μm)
PCB Colour	Blue
Silkscreen	White

Table 4.1 – *PCB capabilities chosen for both PCBs in JLCPCB [15]*

- **PCB design rules:** Another item that depends on the production technology and that we have to take into account whenever designing a PCB are the designing rules. These rules will give us the space needed between each element, the maximum and minimum hole size, the minimum and maximum track size, etc. In our case, this rules can be directly downloaded from their site and imported to . The rules can be found in this GitHub link. These rules were written by an engineer, external to JLCPCB to ease the designing process. Some rules are shown in Figure 4.6 as an example of the type of characteristics that are restricted when manufacturing a PCB, depending on the number of layers and the layer's copper weight.



- 2 layer
  - 1 oz copper
    - 5mil trace with & clearance
    - 0.3mm min. hole diameter
    - 0.6mm min. via diameter
    - 0.25mm hole clearance
  - 2 oz copper
    - 8mil trace width & clearance
    - 0.3mm min. hole diameter
    - 0.6mm min. via diameter
    - 0.25mm hole clearance

Figure 4.6 – *Rules for a 2 layer PCB produced in JLCPCB*

- **Mounting technology:** Once we know the PCB production features and their designing rules, we will need to decide which mounting technology is best for our device. In PCB design, as in many engineering aspects, the less space you need, the better. In this case, the best technology is using SMD components. Even though this is the best technology, some elements are not available on this technology (such as the buzzer or the temperature sensor) or are required to be THT (such as the pin header for connecting both boards and the Raspberry Pi), therefore we will be using also THT although in less proportion.

- **Package types:** Depending on the device we are going to use, different packages may be available. Therefore, we will need to be choosing which one we want to use. For instance, the SMD resistor and capacitor can be found in different sizes such as 1005, 0201, 0402, 0603, 0805, 1008, etc. For choosing which is best, we will be considering whether we plan to be soldering them manually or not. As we intend to solder them manually we will choose the smallest size we could solder easily by hand (usually we will use 0805 capacitors and resistors which are the smallest size available on the laboratory stock).

Once we have explained all the previous steps that have to be made before starting the PCB design process, it is time to detail the steps followed for both of the PCBs produced for this work. Below, for each PCB, all the SCH files are explained, the components selected, with justification if necessary, and the posterior routing and placement in the PCB.

4.3.1.2 Main PCB

As mentioned before, the denominated *Main PCB*, is the one in charge of processing the signals that come from the BLAS, so is the one that is directly connected to the Raspberry Pi. Now, the two phases of a PCB design process, stated at the beginning of Section 4.3.1, will be detailed.

4.3.1.3 Circuit design - Main PCB

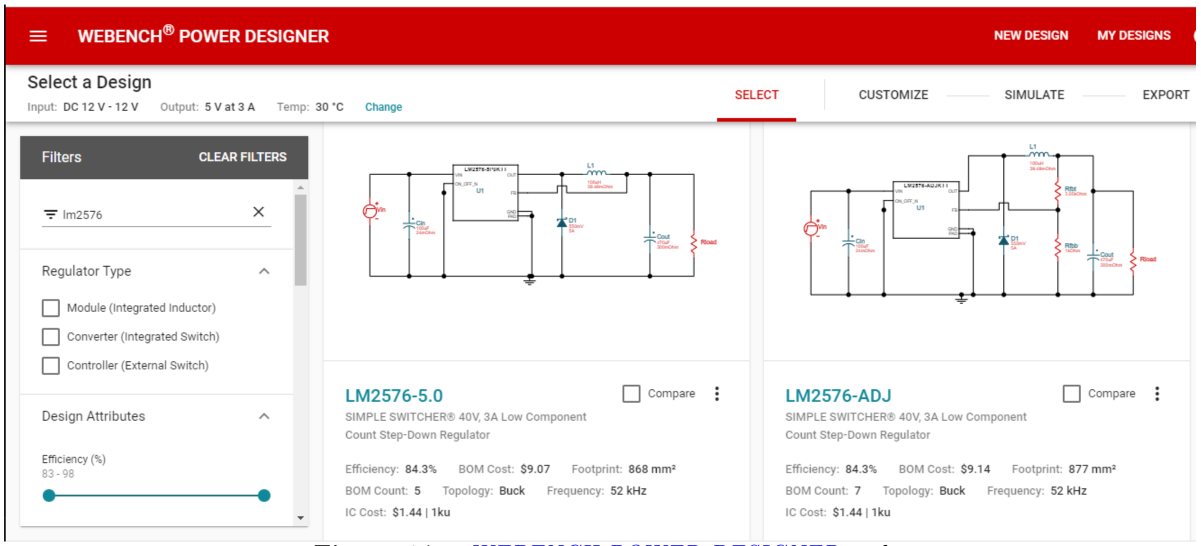
As stated earlier, the circuit design process is based on the design of the different circuits that are needed in the PCB. For this, a list, with the essential functionalities of the PCB was made. By writing this list, it is possible to visualize the different circuits to think about. The list is the following:

- **BLAS' control signals process:** Circuitry is needed to adapt the control signals' tension coming from the BLAS or vice versa for the output ones. However, not all the signals need the same treatment. For instance, the water flow measurement signal only needs to adapt its tension range, although, the signal that indicates the output power from the amplifier (PD_MI), needs to be processed by an ADC before going to the Raspberry Pi GPIO pins.
- **Power supply:** This PCB needs to be able to supply the power needed for the Raspberry Pi and the touchscreen.
- **Sensor management:** Will have to design the circuits capable of reading and processing the information coming from the PT100 in the BLAS and the temperature and humidity sensor will add to this PCB. For instance, for the PT100, will need an ADC.
- **Indicators:** There will be different indicators such as LEDs, reset button and a buzzer.
- **LCD and rotary:** An LCD and a rotary will be connected to the Raspberry Pi via this PCB, so it is necessary to design the proper circuits for these connections.

Once it is clear the different functions our *Main PCB* needs to achieve, it is possible to divide this circuit design process into several modules, each one will have an SCH file associated. Now, the SCH files for our *Main PCB* are going to be explained, including the components used.

Power Supply:

As stated in the previous list, we need to supply power to the Raspberry Pi and the touchscreen. As mentioned in [20], the Raspberry Pi 4 model B needs a power supply capable of delivering 5 V at 3 A. Figure 4.10 shows the SCH file for the power supply circuit. To obtain 5 V at 3 A, we will use the 12 V coming from the BLAS and a voltage regulator. The company Texas Instruments has a free tool called WEBENCH POWER DESIGNER, that given different parameters designs a circuit suitable to the specifications needed. So, we indicate that the input voltage will be 12 V, the output one will be 5 V, and the maximum current we need is 3 A. The results given are shown in Figure 4.7



Select a Design

Input: DC 12 V - 12 V Output: 5 V at 3 A Temp: 30 °C

lm2576

Module (Integrated Inductor)
Converter (Integrated Switch)
Controller (External Switch)

Efficiency (%)
83 - 98

LM2576-5.0
SIMPLE SWITCHER® 40V, 3A Low Component Count Step-Down Regulator
Efficiency: 84.3% BOM Cost: $9.07 Footprint: 868 mm²
BOM Count: 5 Topology: Buck Frequency: 52 kHz
IC Cost: $1.44 | 1ku

LM2576-ADJ
SIMPLE SWITCHER® 40V, 3A Low Component Count Step-Down Regulator
Efficiency: 84.3% BOM Cost: $9.14 Footprint: 877 mm²
BOM Count: 7 Topology: Buck Frequency: 52 kHz
IC Cost: $1.44 | 1ku

Figure 4.7 – *WEBENCH POWER DESIGNER tool*

This tool suggests the different voltage regulators Texas Instruments offers to use. In this case, we will use the 3 A voltage regulator called LM2576SX-5.0/NOPB. This voltage regulator is designed to be used as follows:

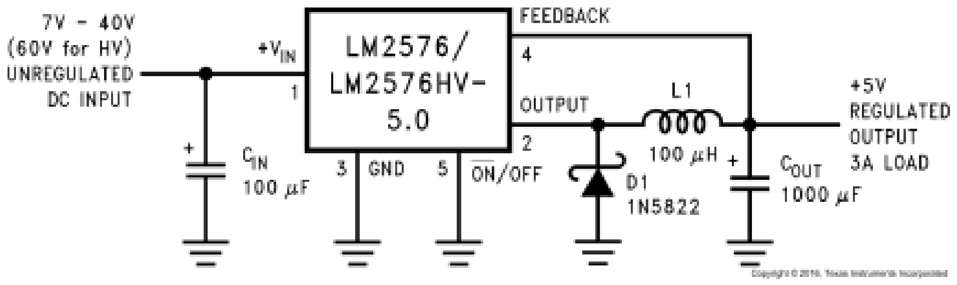


Figure 4.8 – *LM2576 implementation [14]*

The circuit from Figure 4.8 is the implementation philosophy suggested in the component's datasheet [14], that coincides with the one given by the tool WEBENCH POWER DESIGNER. Furthermore, this tool, not only suggests the voltage regulator and the circuit's topology, but also the other components. The components used in Figure 4.10 are the ones proposed by WEBENCH POWER DESIGNER.

However, we need to protect our circuit from possible tension changes that can occur in the 12 V source. To do so, we will use the smart high side switch called ITS4200 from Infineon Technologies. The block diagram of the ITS4200 is shown in Figure 4.9. High-side switches are used to turn electrical loads ON and OFF by switching the positive (high-) side of the load supply. Additionally, smart high-side switches are designed with the ability to protect themselves and diagnose possible unintended system behaviour. Hence, we now have our Raspberry Pi protected against possible over tensions coming from the source.

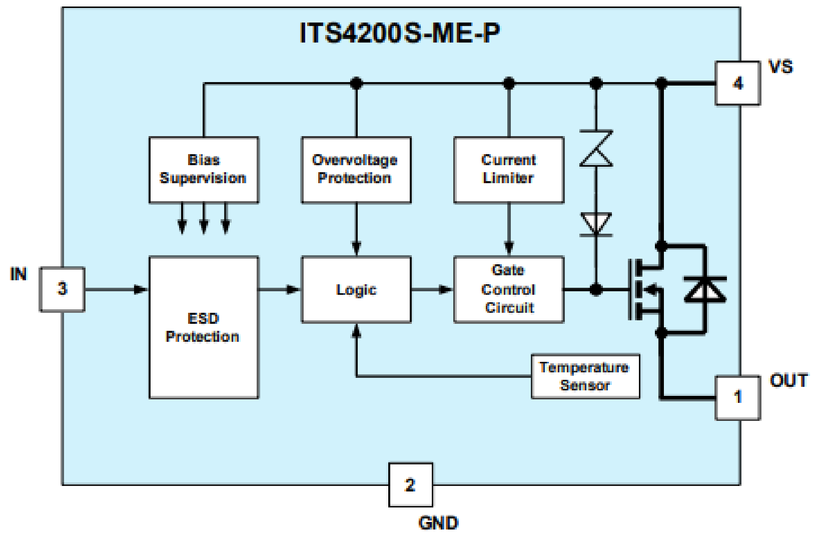


Figure 4.9 – *ITS4200's Block Diagram* [10]

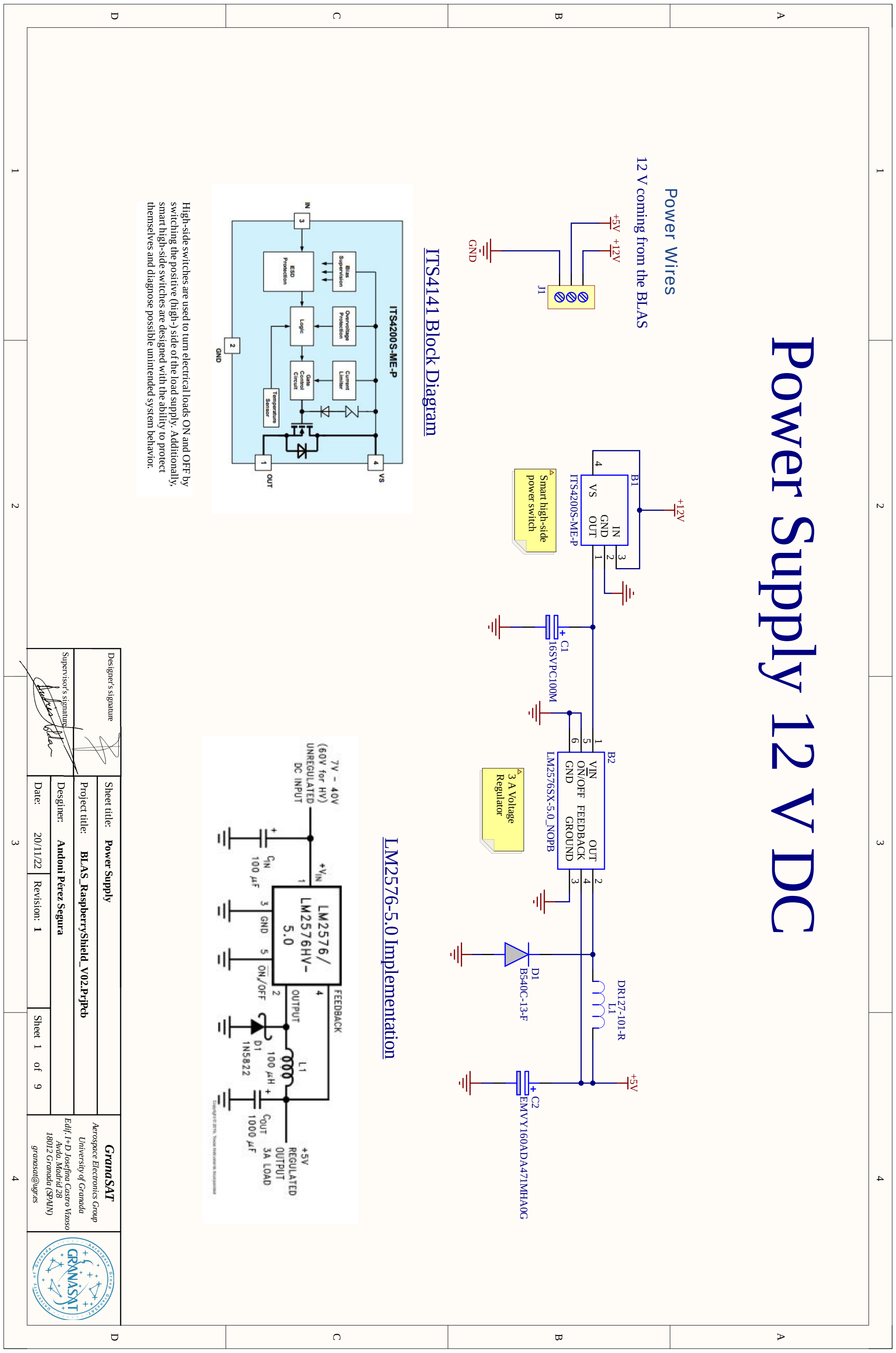

Power Supply 12 V DC
Power Wires
12 V coming from the BLAS
+5V
+12V
GND
J1
B1
ITS4200S-ME-P
VS
IN
GND
OUT
Smart high-side power switch
C1
16SVPC100M
B2
LM2576SX-5.0_NOPB
VIN
ON/OFF
GND
OUT
FEEDBACK
GROUND
3 A Voltage Regulator
D1
B540C-13-F
L1
DR127-101-R
C2
EMVY160ADA471MHA0G
ITS4141 Block Diagram
ITS4200S-ME-P
High-side switches are used to turn electrical loads ON and OFF by switching the positive (high-) side of the load supply. Additionally, smart high-side switches are designed with the ability to protect themselves and diagnose possible unintended system behavior.
LM2576-5.0 Implementation
7V - 40V (60V for HV) UNREGULATED DC INPUT
LM2576/ LM2576HV-5.0
+5V REGULATED OUTPUT 3A LOAD
Designer's signature
Supervisor's signature
Sheet title: Power Supply
Project title: BLAS_RaspberryShield_V02.PrjPcb
Designer: Andoni Pérez Segura
Date: 20/11/22
Revision: 1
Sheet 1 of 9
GranaSAT
Aerospace Electronics Group
University of Granada
GRANASAT

Figure 4.10 – *Power supply SCH file*

PT100 signal process:

As explained before, a PT100 is a resistance temperature detector, made of material Platinum (PT), and its resistance value at 0 °C temperature is 100 Ω. Hence, the name is PT100. The impedance of the PT100 is dependant of the temperature, as Equations 4.3.1 and 4.3.2 state:

$$\frac{R_t}{R_0} = 1 + AT + BT^2 \quad (for \quad T > 0\,^{o}\mathrm{C}) \tag{4.3.1}$$

$$\frac{R_t}{R_0} = 1 + AT + BT^2 + CT^3(T - 100) \quad (for\, T < 0\,^{o}\mathrm{C}) \tag{4.3.2}$$

where R_T is the impedance of the PT100 with a temperature of T, R_0 is the impedance for 0 °C, in our case, 100 Ω, A, B and C are constant variables whose values are fixed by the ICE 60751 normative as:

- $A = 3.9083 \times 10^{-3}\,^{o}\mathrm{C}$
- $B = -5.775 \times 10^{-7}\,^{o}\mathrm{C}$
- $C = -4.183 \times 10^{-12}\,^{o}\mathrm{C}$

So, the way to know the temperature via the PT100 is by measuring the voltage drop in the resistance. For that, it is necessary to design a constant current source, so that the voltage drop only depends on the resistance value. Furthermore, this voltage drop needs to be between 0 and 3.3 V, so the ADC can convert it into a digital value. The circuit that accomplishes all these requirements is shown in Figure 4.12. This design was already done in the GranaSAT laboratory.

In Figure 4.11, the circuit simulation in LTSpice is depicted. The circuit has been designed so the maximum voltage value of 3.3 V comes with a temperature of 80ºC (x-axis).

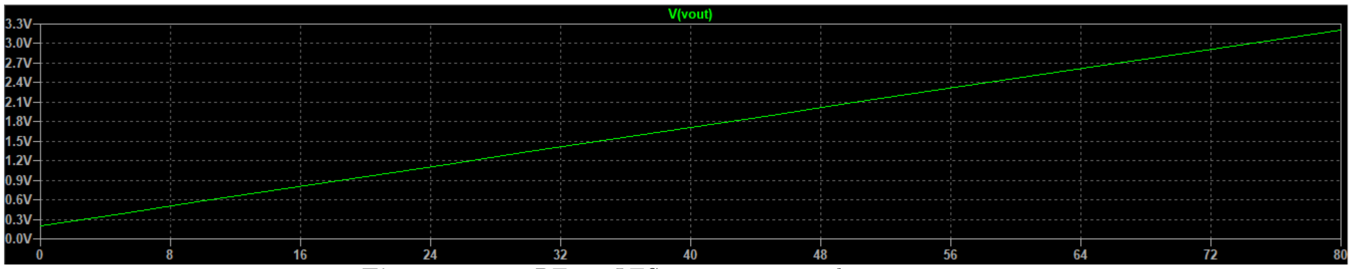


Figure 4.11 – *PT100 LTSpice circuit simulation*

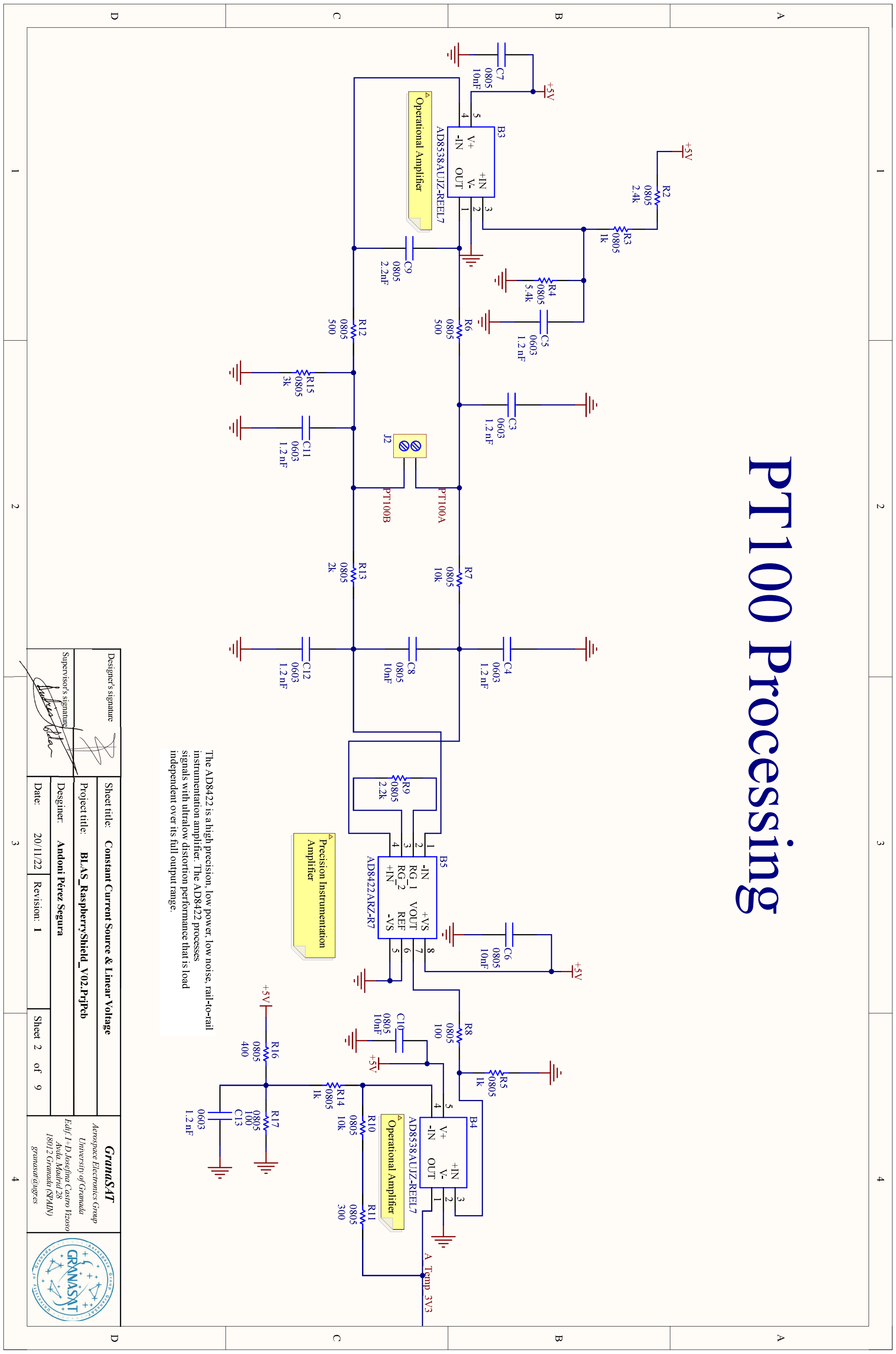
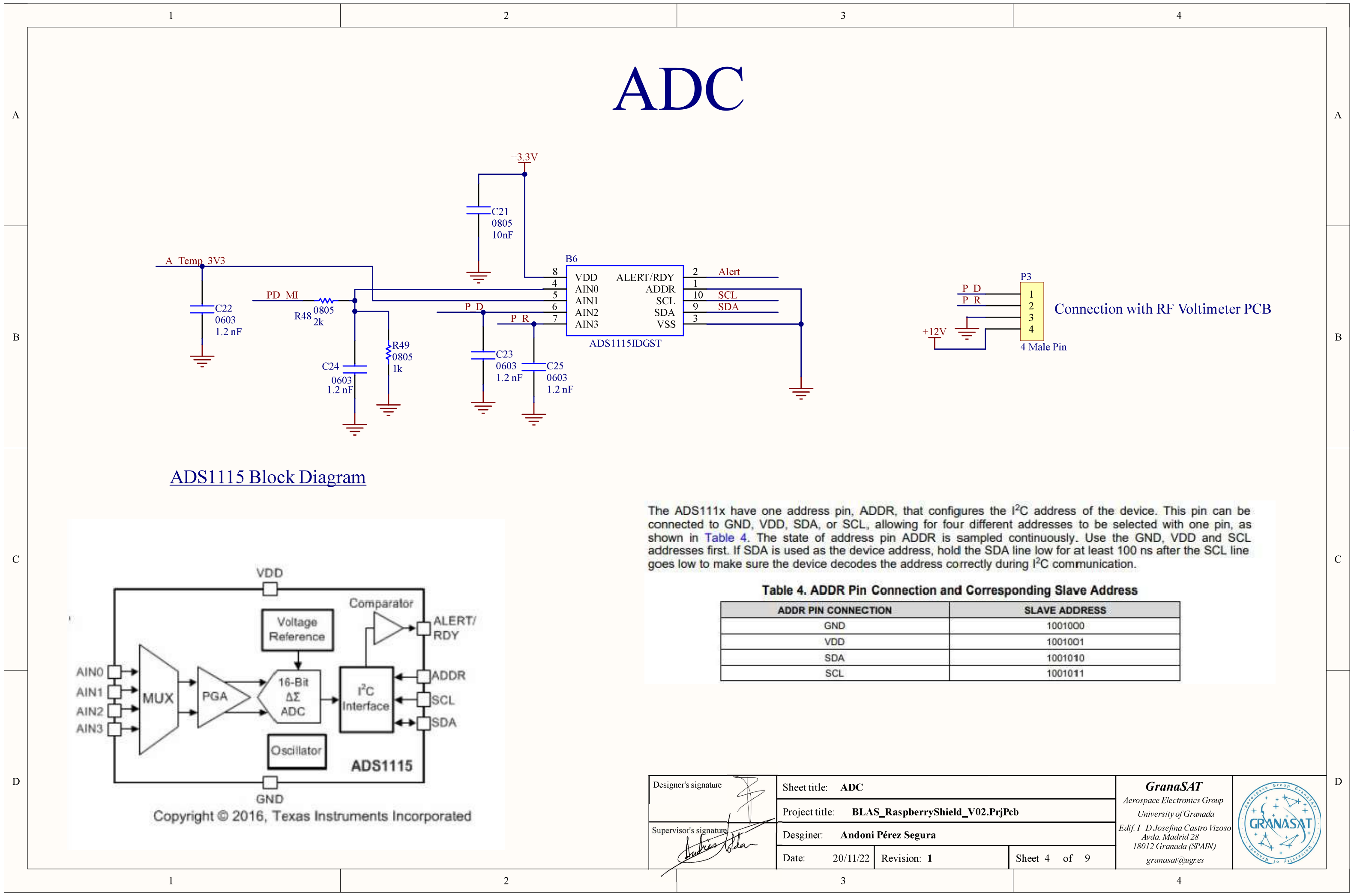


Figure 4.12 – *PT100 SCH file*

ADC:

An ADC will be used to convert the analogic values available: the PT100 signal (A_Temp_3V3), two signals from the directional couplers (P_D and P_R), and the signal PD_MI from the BLAS. The circuit, Figure 4.13, has been designed following the specifications from the ADC datasheet [13].

As the PD_MI signal will have tension values higher than the maximum input voltage (5.5 V), a resistor divider has been introduced. Moreover, a 4 via connector is necessary to get the P_D and P_R signals from the RF Voltmeter PCB that will be explained in the next section. Moreover, this connector is used to power the mentioned PCB.



The ADS111x have one address pin, ADDR, that configures the I^{2}C address of the device. This pin can be connected to GND, VDD, SDA, or SCL, allowing for four different addresses to be selected with one pin, as shown in Table 4. The state of address pin ADDR is sampled continuously. Use the GND, VDD and SCL addresses first. If SDA is used as the device address, hold the SDA line low for at least 100 ns after the SCL line goes low to make sure the device decodes the address correctly during I^{2}C communication.

Table 4. ADDR Pin Connection and Corresponding Slave Address

ADDR PIN CONNECTION	SLAVE ADDRESS
GND	1001000
VDD	1001001
SDA	1001010
SCL	1001011

Designer's signature	Sheet title: ADC		GranaSAT Aerospace Electronics Group University of Granada Edif. I+D Josefina Castro Vizoso Avda. Madrid 28 18012 Granada (SPAIN) granasat@ugr.es
	Project title: BLAS_RaspberryShield_V02.PrjPcb		
Supervisor's signature	Desginer: Andoni Pérez Segura		
	Date: 20/11/22 Revision: 1	Sheet 4 of 9	

Figure 4.13 – *ADC SCH file*

BLAS Control Signals:

To process the output and input signals from the BLAS, 2N7002 MOSFET will be used. As the GPIO pins from the Raspberry Pi 4, cannot have voltages bigger than 3.3 V [20], a MOSFET inverter has been designed using the 3.3 V from the Raspberry Pi power pins. The same is for the input signals to the BLAS that need to be of 12 V. A 12 via connector has been chosen to connect the PCB with the BLAS Control Signals. The circuit design is shown in Figure 4.16

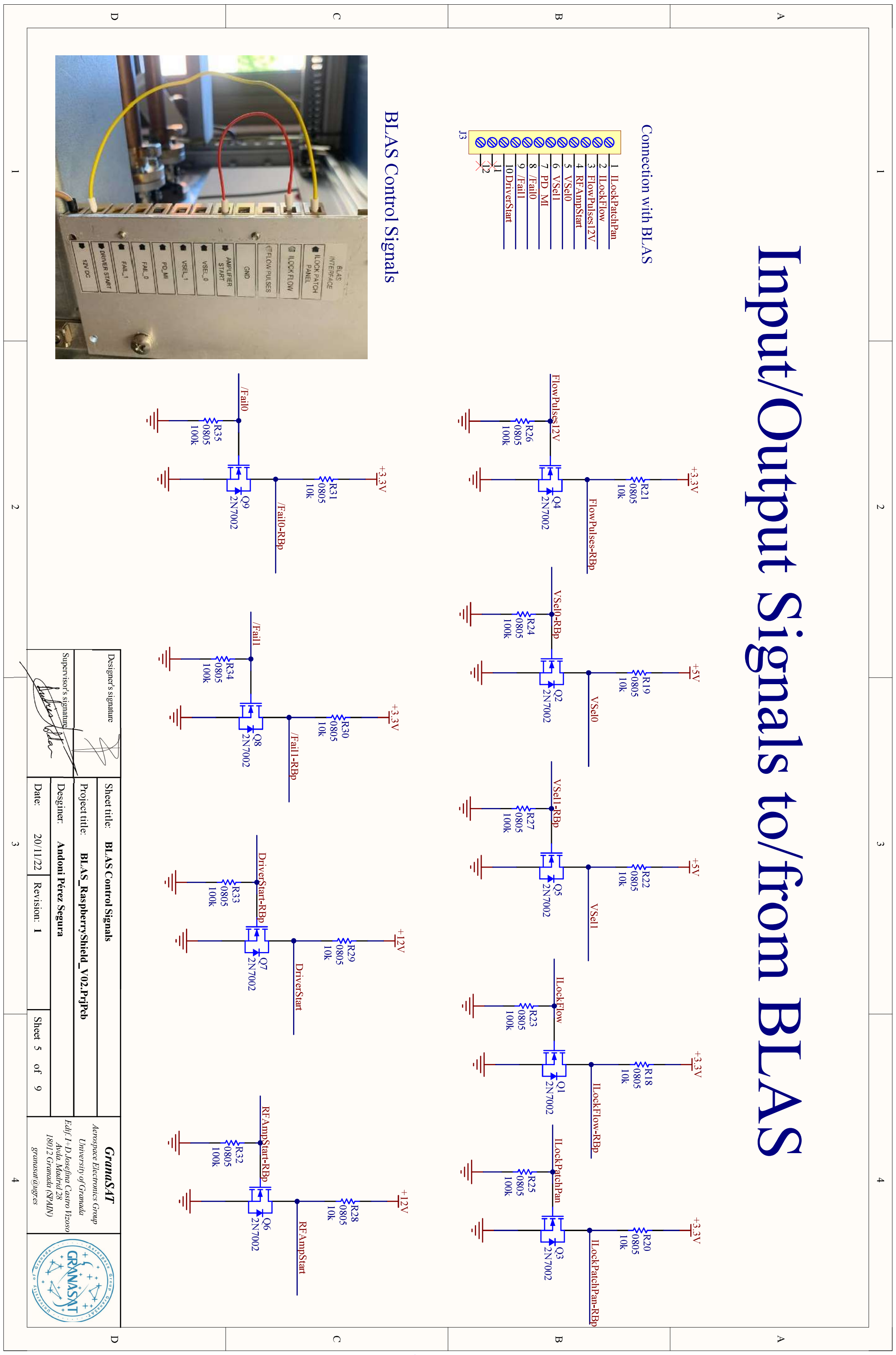
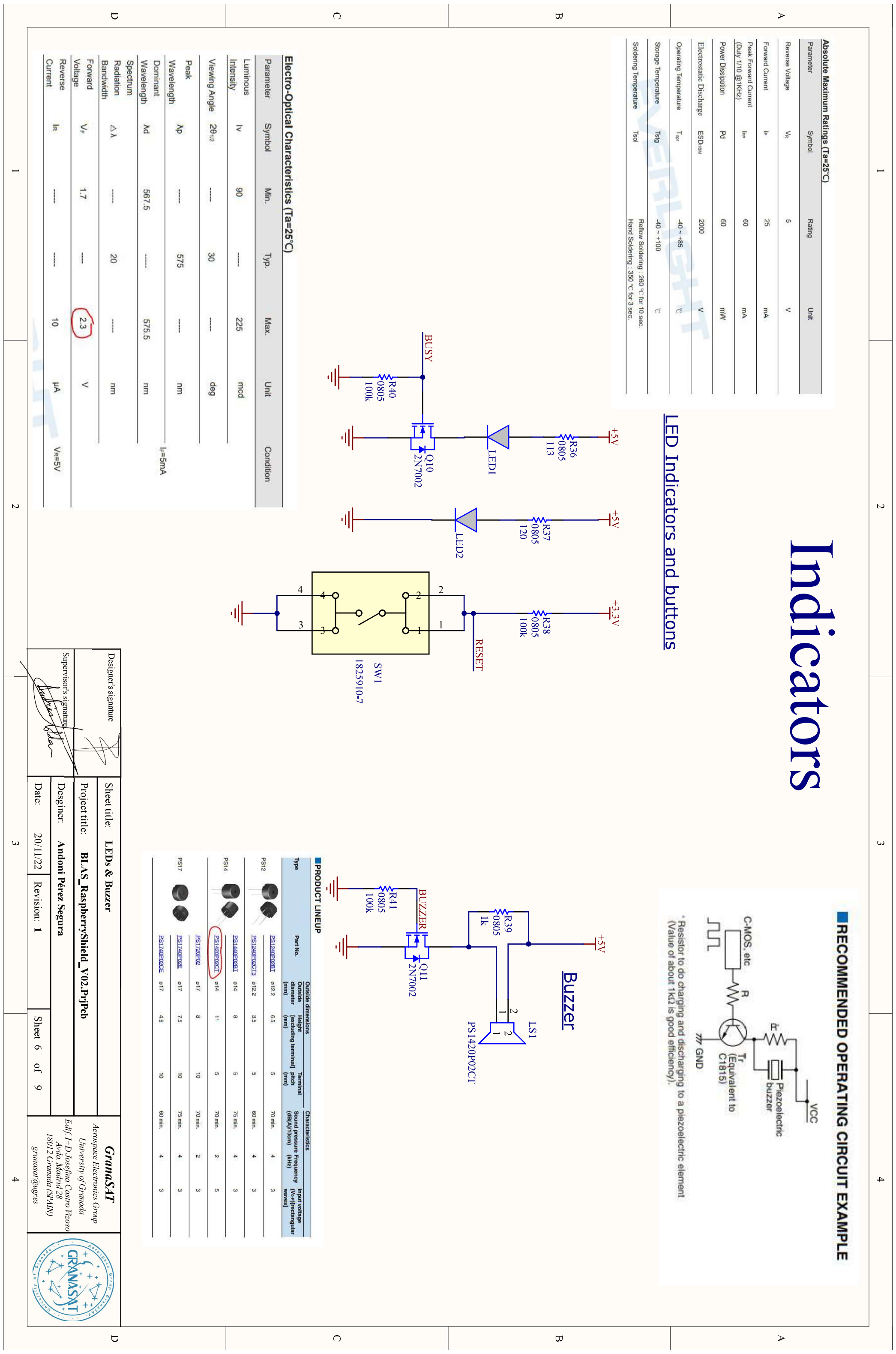


Figure 4.14 – *BLAS Control Signals SCH file*

Indicators:



Absolute Maximum Ratings (Ta=25°C)

Parameter	Symbol	Rating	Unit
Reverse Voltage	V_R	5	V
Forward Current	I_F	25	mA
Peak Forward Current (Duty 1/10 @1KHz)	I_{FP}	60	mA
Power Dissipation	Pd	60	mW
Electrostatic Discharge	ESD_{HBM}	2000	V
Operating Temperature	T_{opr}	-40 ~ +85	℃
Storage Temperature	T_{stg}	-40 ~ +100	℃
Soldering Temperature	Tsol	Reflow Soldering : 260 ℃ for 10 sec. Hand Soldering : 350 ℃ for 3 sec.	

Electro-Optical Characteristics (Ta=25°C)

Parameter	Symbol	Min.	Typ.	Max.	Unit	Condition
Luminous Intensity	Iv	90	-----	225	mcd	
Viewing Angle	$2\theta_{1/2}$	-----	30	-----	deg	
Peak Wavelength	λp	-----	575	-----	nm	I_F=5mA
Dominant Wavelength	λd	567.5	-----	575.5	nm	
Spectrum Radiation Bandwidth	Δλ	-----	20	-----	nm	
Forward Voltage	V_F	1.7	-----	2.3	V	
Reverse Current	I_R	-----	-----	10	μA	V_R=5V

Figure 4.15 – *Indicators SCH file*

Temperature sensor (DHT11):

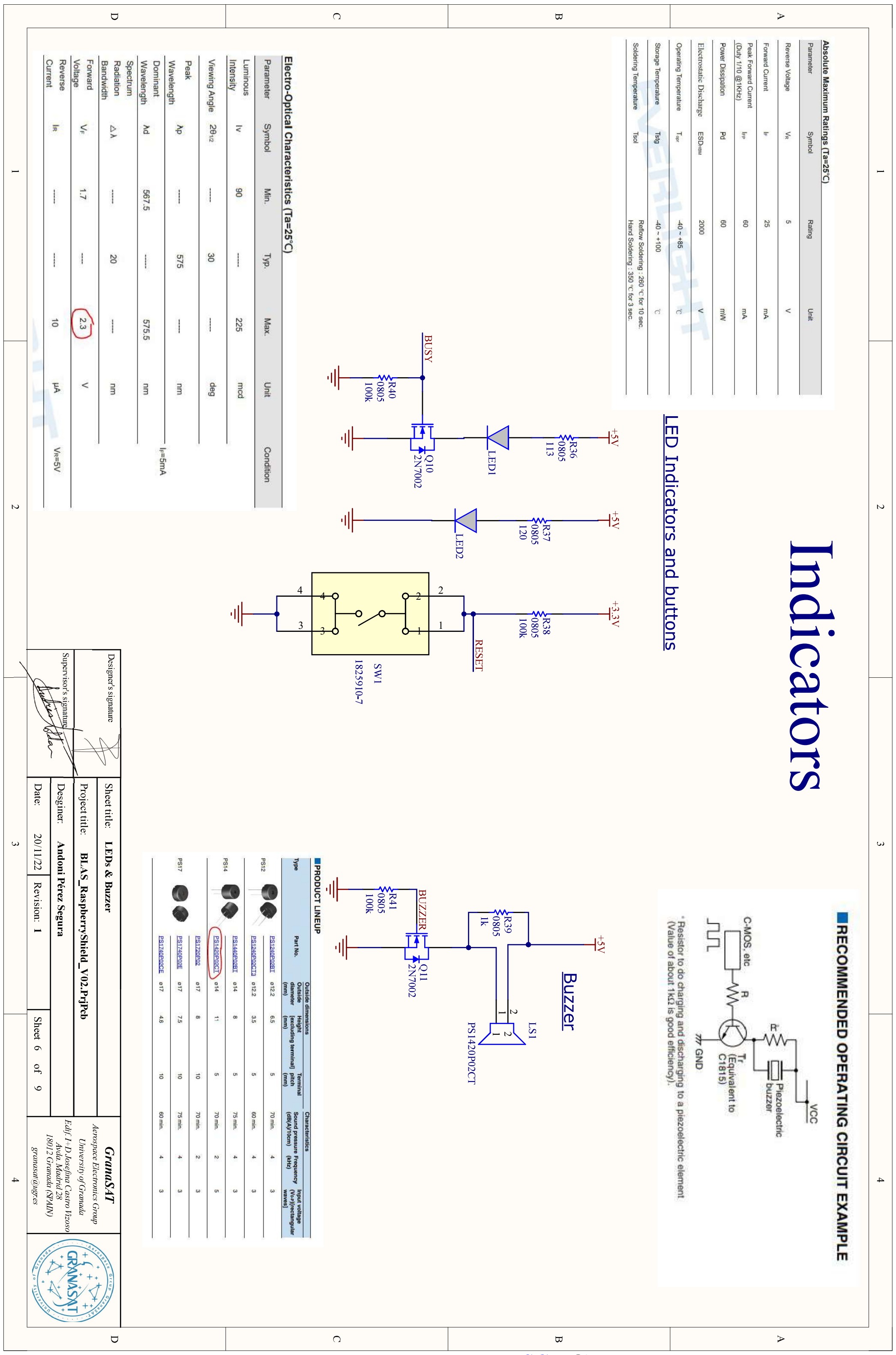


Figure 4.16 – *DHT11 SCH file*

LCD:

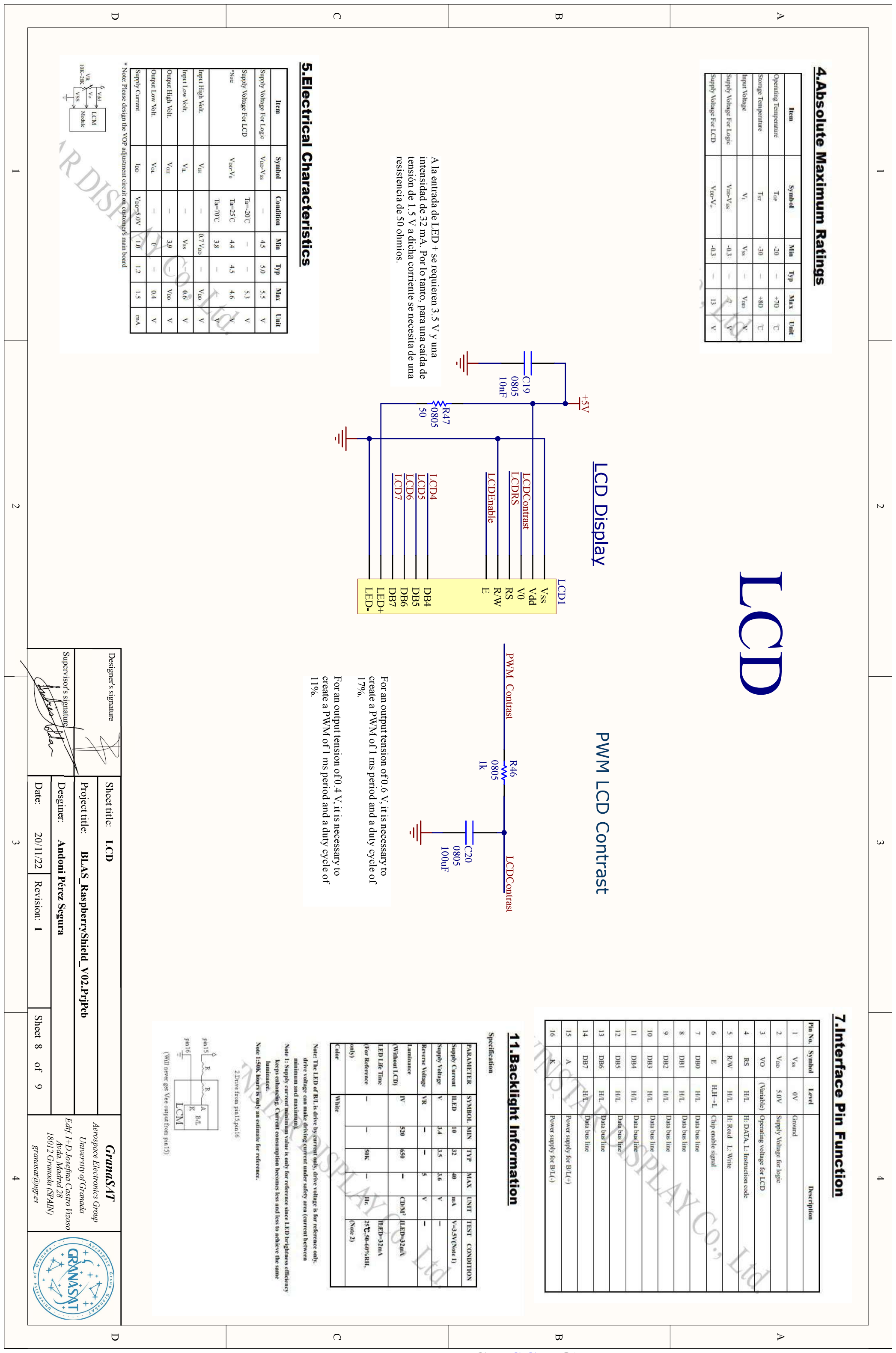

LCD
LCD Display
PWM LCD Contrast
LCD1
Vss
Vdd
V0
RS
R/W
E
DB4
DB5
DB6
DB7
LED+
LED-
+5V
C19
0805
10nF
R47
0805
50
LCDContrast
LCDRS
LCDEnable
LCD4
LCD5
LCD6
LCD7
PWM_Contrast
R46
0805
1k
C20
0805
100uF
A la entrada de LED + se requieren 3,5 V y una intensidad de 32 mA. Por lo tanto, para una caída de tensión de 1,5 V a dicha corriente se necesita de una resistencia de 50 ohmios.
For an output tension of 0,6 V, it is necessary to create a PWM of 1 ms period and a duty cycle of 17%.
For an output tension of 0,4 V, it is necessary to create a PWM of 1 ms period and a duty cycle of 11%.
4.Absolute Maximum Ratings
5.Electrical Characteristics
7.Interface Pin Function
11.Backlight Information
Designer's signature
Supervisor's signature
Sheet title: LCD
Project title: BLAS_RaspberryShield_V02.PrjPcb
Designer: Andoni Pérez Segura
Date: 20/11/22
Revision: 1
Sheet 8 of 9
GranaSAT
Aerospace Electronics Group
University of Granada
Edif. I+D Josefina Castro Vizoso
Avda. Madrid 28
18012 Granada (SPAIN)
granasat@ugr.es
A
B
C
D
1
2
3
4

Figure 4.17 – *LCD SCH file*

Rotary:

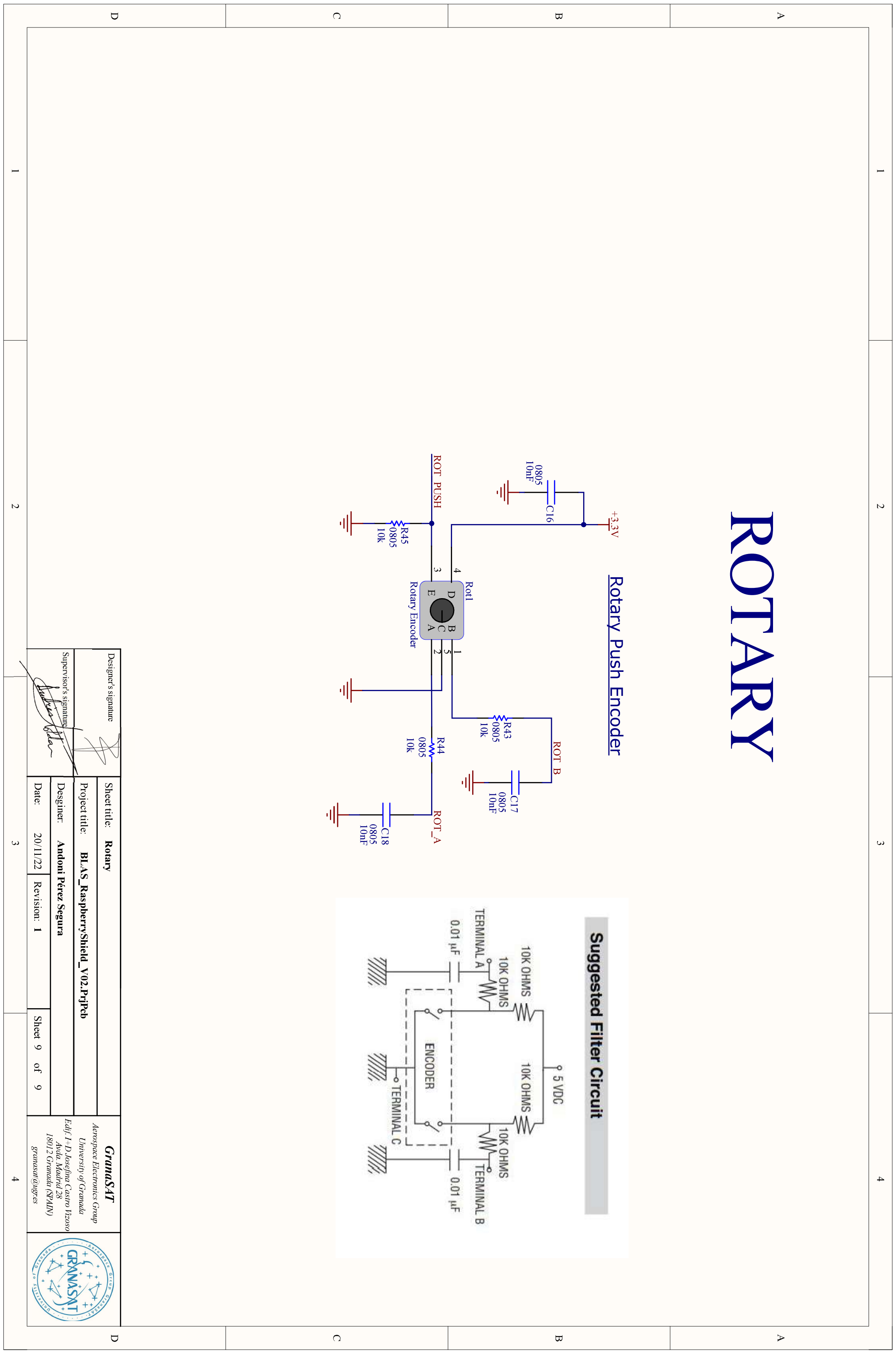


Figure 4.18 – *Rotary SCH file*

Raspberry Pi pin connections:

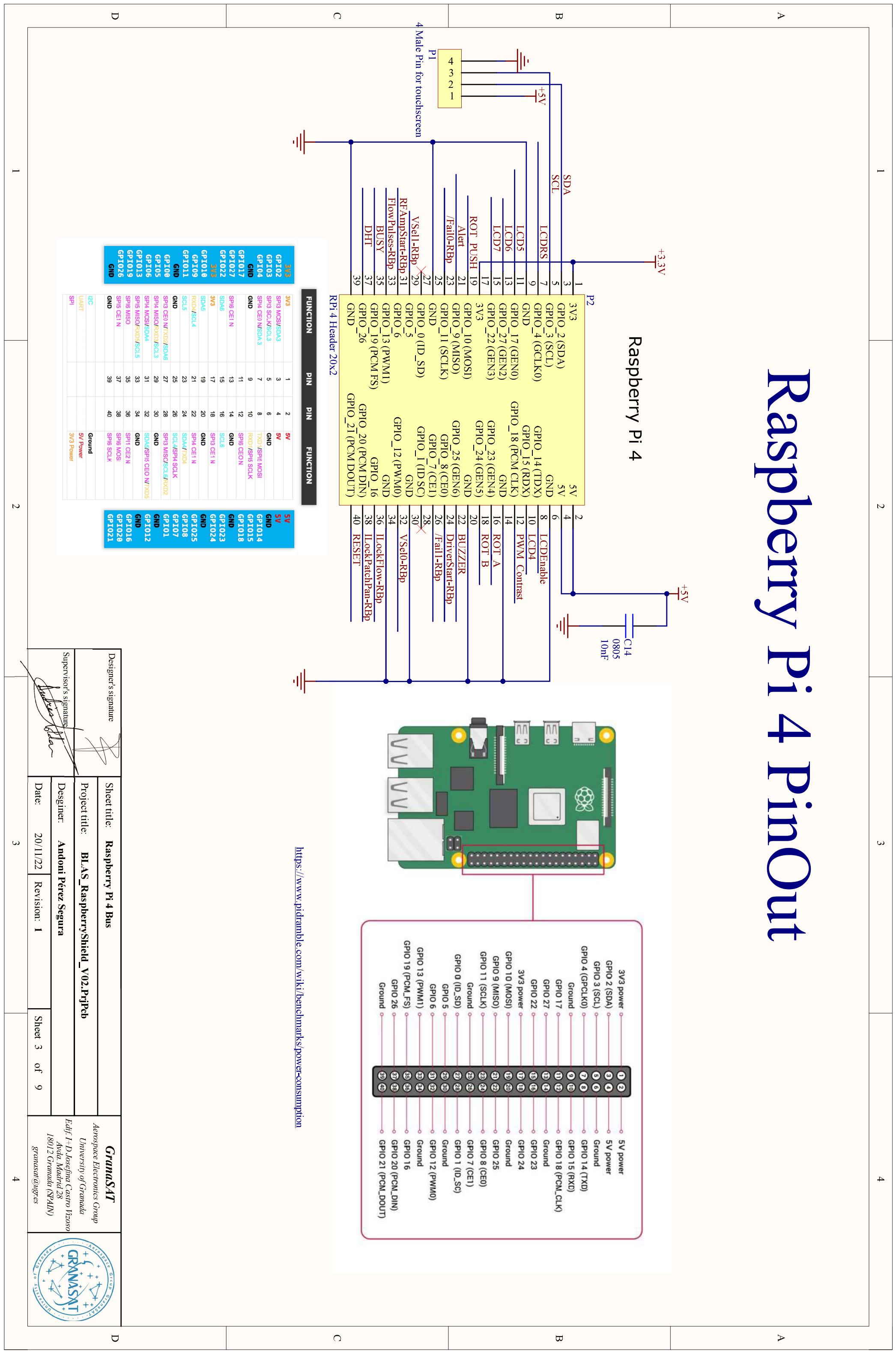


Figure 4.19 – *Raspberry Pi pinout SCH file*

4.3.1.4 Modules placement & route - Main PCB

Once all the necessary circuits are designed, it is time to place all the components on the PCB board and route the track between them, Figure 4.20. Firstly, the size of the board has to be set, in this case, 86 x 79 mm. Then, the different components are placed on the board. The criteria followed for the placing was leaving the power supply components in the top layer but in the inferior part of the board, as the 12 V coming from the BLAS are located in the inferior part of the BLAS interface (Figure 2.21). On the other hand, as the cables from the PT100 are coming from the SSPA drawer, the 2 via connector needs to be on the right. Moreover, the 12 via connector for the BLAS control signals are located in the superior part of the board.

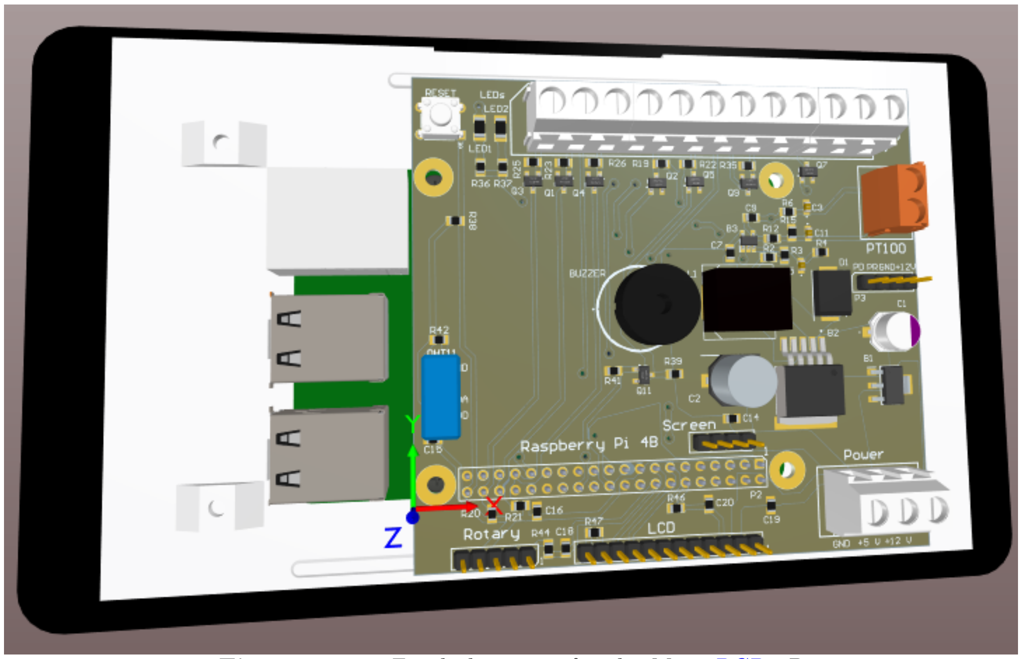


Figure 4.20 – *Final placement for the Main PCB 3D*

Most of the components are located in the top layer, Figure 4.21, so it is easier to test the components. However, as there is a finite space on the board, and it was the first PCB designed, some components were placed in the bottom layer, Figure 4.22. For instance, the ADC circuit was mainly placed in the bottom layer, which would be a problem when testing comes.

4

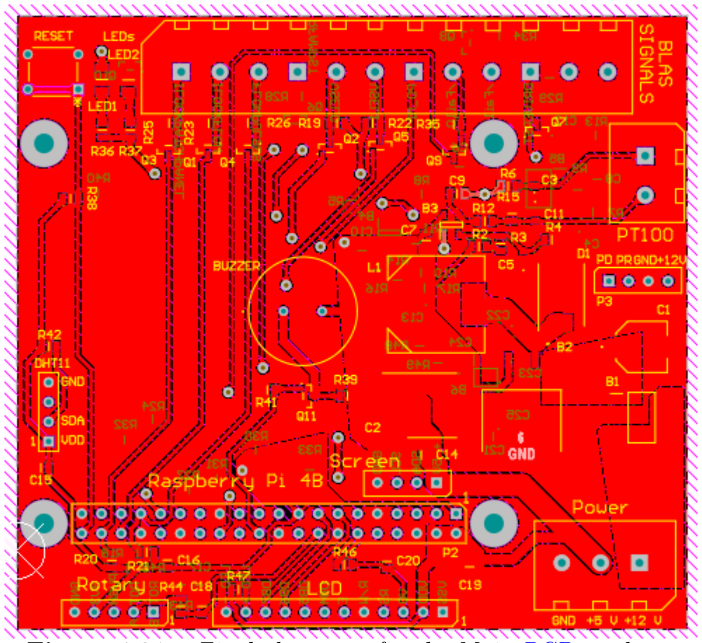


Figure 4.21 – *Final placement for the Main PCB top layer*

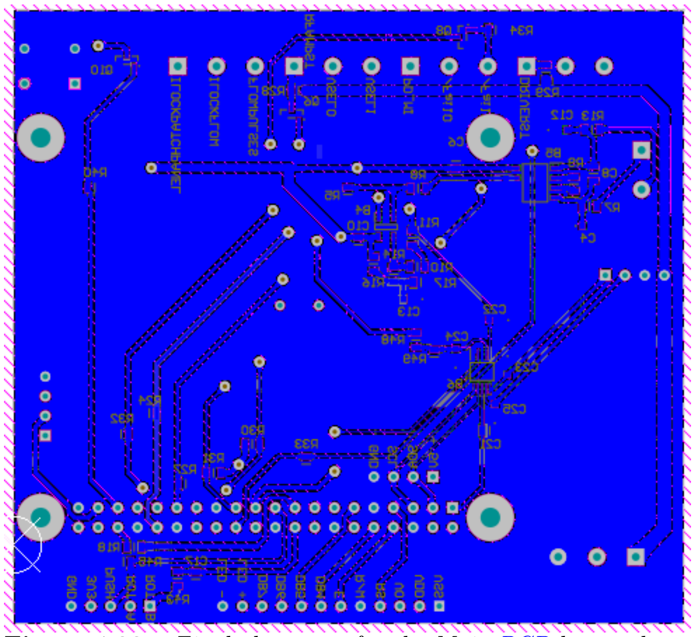


Figure 4.22 – *Final placement for the Main PCB bottom layer*

4.3.2 Assembly and testing - Main PCB

In this section, the assembly of the Main PCB will be explained. First, solder paste will be placed using the stencil. To not have the PCB moving while the solder paste is being applied, the extra PCBs will be used, Figure 4.23

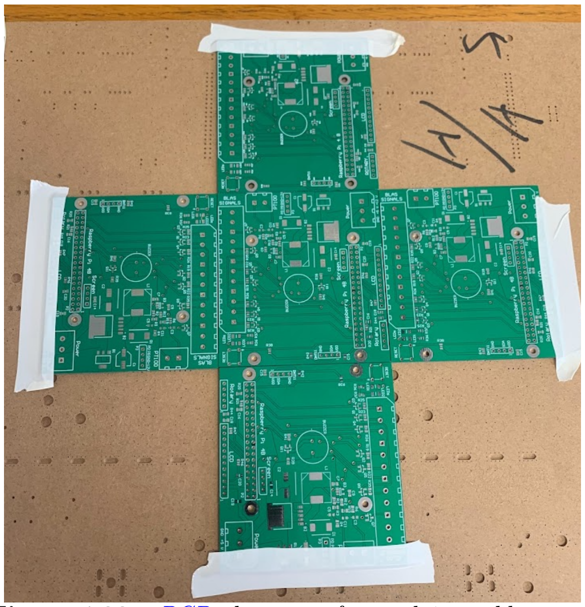


Figure 4.23 – *PCB placement for applying solder paste*

Then, using a spatula (Figure 4.24), the solder paste is applied above the stencil. Once the components' footprints are full of solder paste (Figure 4.25), the components are placed and introduced into an oven, so the components get soldered to the PCB.

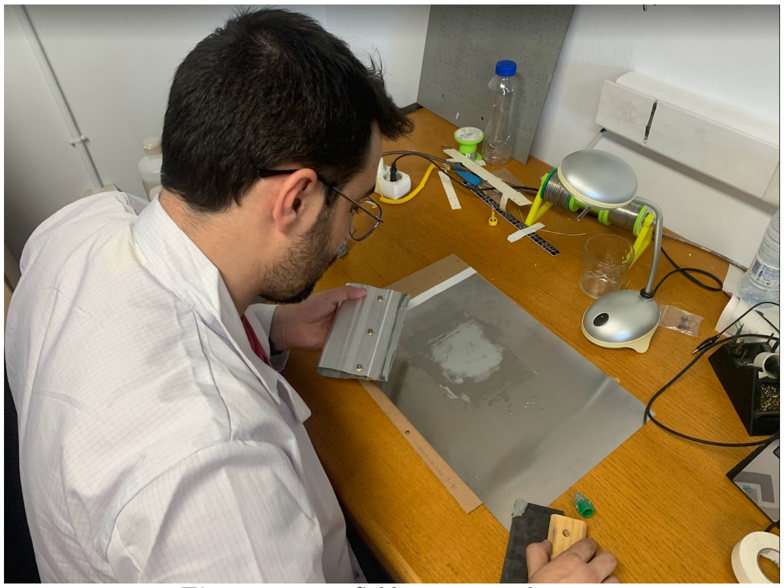


Figure 4.24 – *Solder paste application*

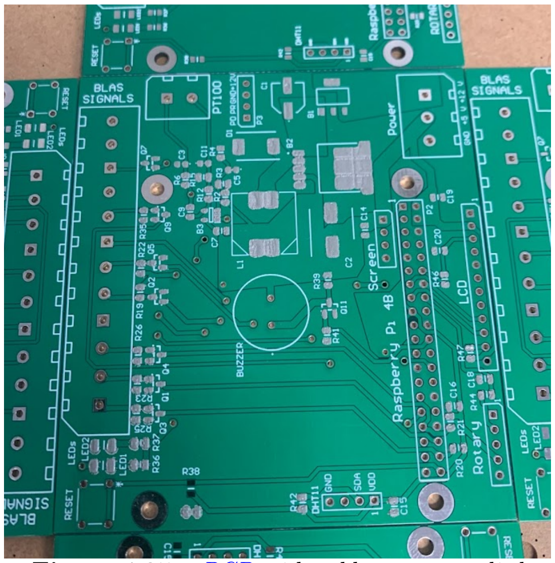


Figure 4.25 – *PCB with solder paste applied*

Finally, the components from the bottom layer and the connectors are soldered manually, Figure 4.26.

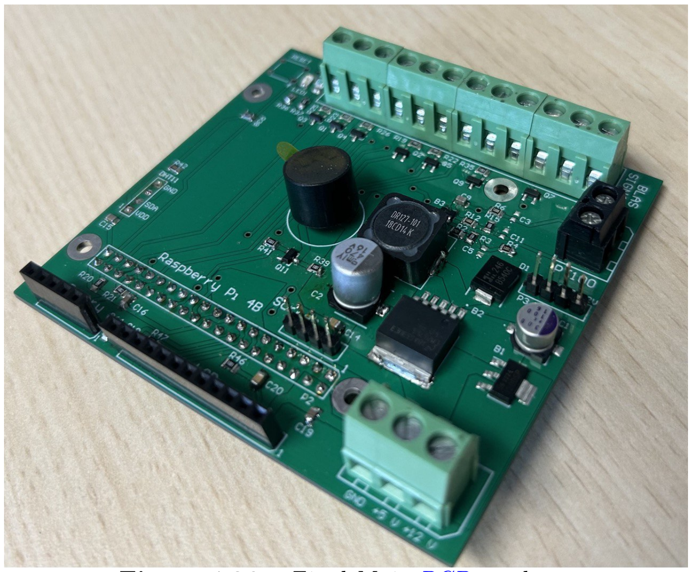


Figure 4.26 – *Final Main PCB top layer*

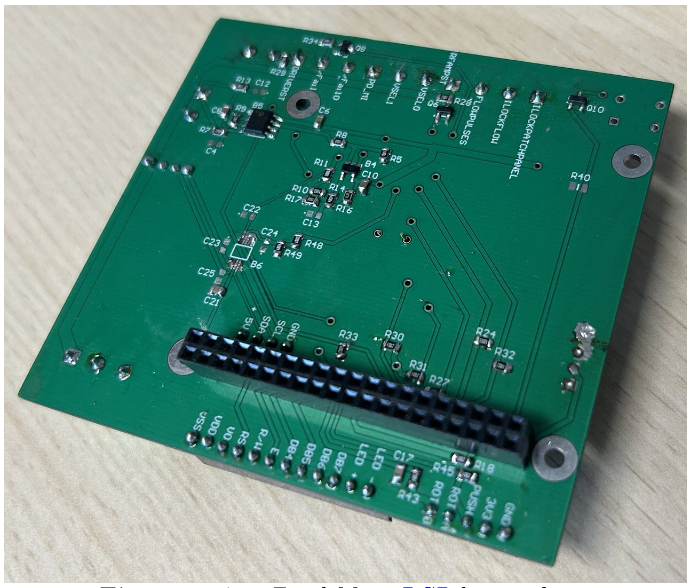


Figure 4.27 – *Final Main PCB bottom layer*

Once the PCB is finished tests to check the different functionalities, the main PCB was designed for, work as desired, will be done. The mentioned functionalities are listed below:

- Power supply for the Raspberry Pi 4 and touchscreen
- Indicator LEDs
- Process BLAS control signals
- LCD

4.3.3 Power supply

The Raspberry Pi 4 requires a voltage supply of 5 V to switch on, as well as its touchscreen, in certain pins. So, to get the Raspberry Pi 4 running, the circuit explained in Figure 4.10 is used. The mentioned circuit can be seen in Figure 4.28, after being soldered to the PCB, inside the red lines. It is important to point out, this circuit switches a 12 V DC signal into a 5 V one, to take advantage of the BLAS 12 V signal supplied by the BLAS.

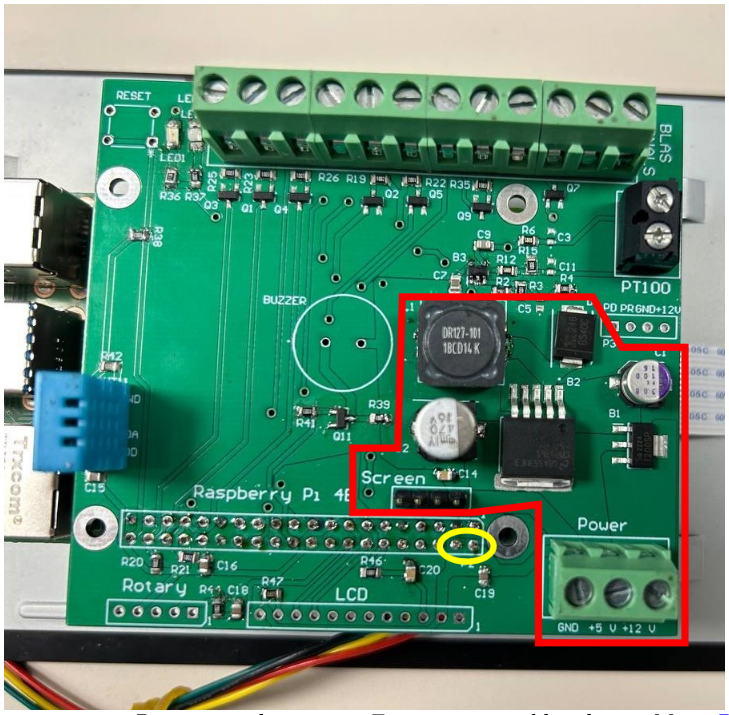


Figure 4.28 – *Power supply circuit, Figure 4.10, soldered into Main PCB*

To examine the proper working of the circuit designed, before connecting it to the BLAS, it will be tested using the DC power supply SPD3303X from Siglent. This device offers the possibility to limit the current supplied to the circuit, so in case of a short circuit, damages could be avoided. Firstly, the PCB will be supplied through the terminals, without the Raspberry Pi 4 connected to it. The current will be limited to 100 mA to avoid damage in case of a short circuit, as seen in Figure 4.29.

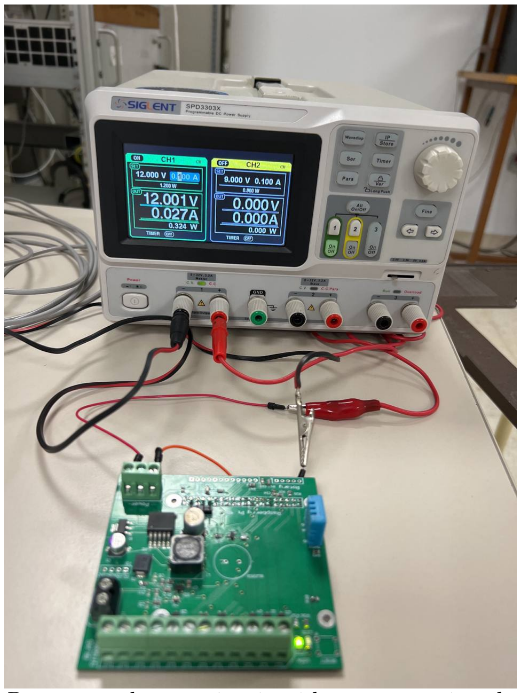


Figure 4.29 – *Power supply test circuit without connecting the Raspberry Pi 4*

If everything goes to plan, there will be a 5 V signal in the terminal prepared for it and in pins 2 and 4, as stated in the Raspberry Pi 4 pinout scheme, Figure **??**, found inside the yellow ellipse in Figure 4.28. To measure this signal, the MSO-X-4104A oscilloscope will be used. In figure 4.30, there is a screen capture of the oscilloscope measuring in pin 2, obtained with the Python library elaborated to do so.

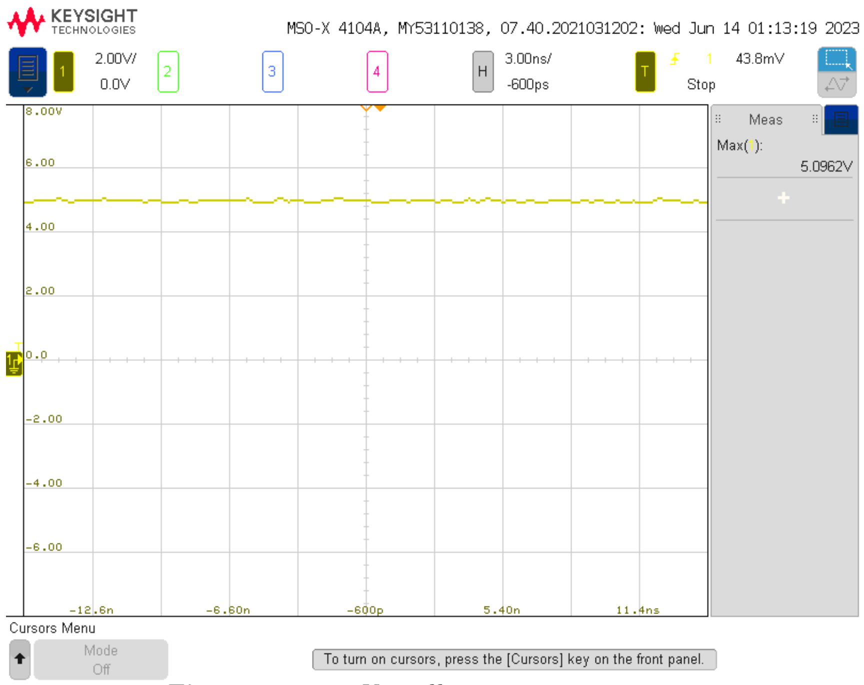


Figure 4.30 – *5 V oscilloscope measure in pin 2*

It is crucial to measure all the other pins, to identify if there is only 5 V in the pins desired, as an overvoltage in the GPIO pins would cause damage to the circuitry. The main PCB current consumption only with the devices using 5 V and 12 V, without the Raspberry Pi 4 and the touchscreen, is of 27 mA. Once the 5 V signal is available, the Raspberry Pi 4 will be connected but not the touchscreen, to check if it switches on and the current consumption. In this occasion, the power supply will be limited to 400 mA, Figure 4.31. Apparently, the Raspberry Pi 4 switches on as its LED switched on. The current consumption of the Raspberry Pi 4 in an idle state is around 230 mA.

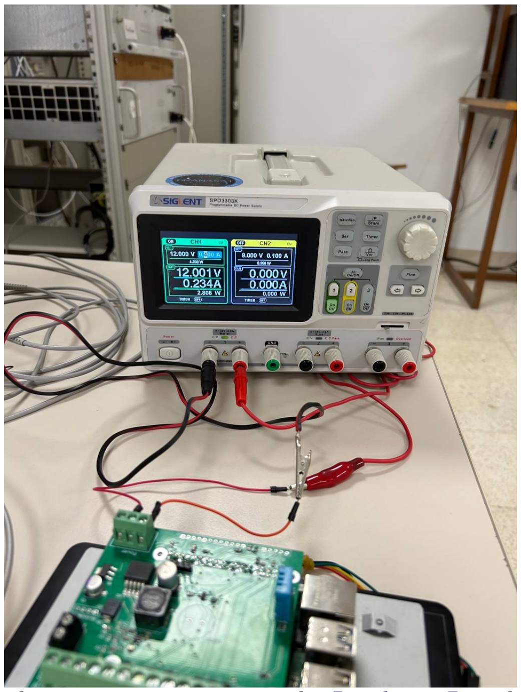


Figure 4.31 – *Power supply test circuit connecting the Raspberry Pi 4, but without the touchscreen*

Finally, the touchscreen will be connected to obtain the final current consumption and to check if the power supply is capable of switching the touchscreen on. The touchscreen cables are connected to the pins prepared, blue box in Figure 4.32. The current consumption for this case is around 500 mA, being the Raspberry Pi 4 in an idle state.

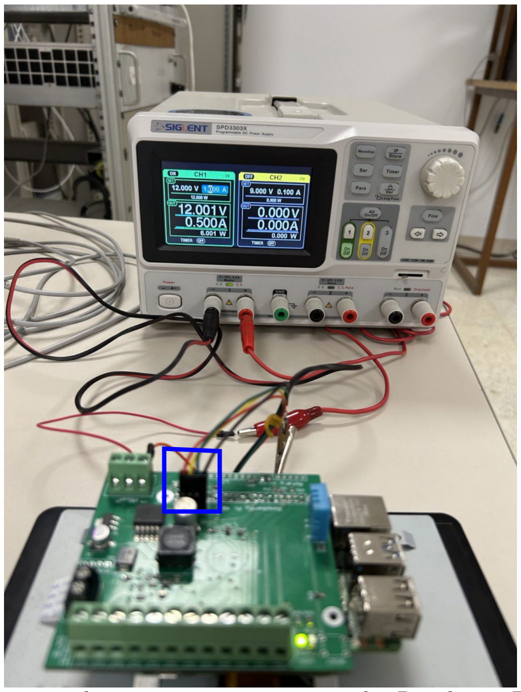


Figure 4.32 – *Power supply test circuit connecting the Raspberry Pi 4 and touchscreen*

By looking at Figure 4.33, the proper functioning of the power supply circuit is demonstrated, as the touchscreen appears switched on.

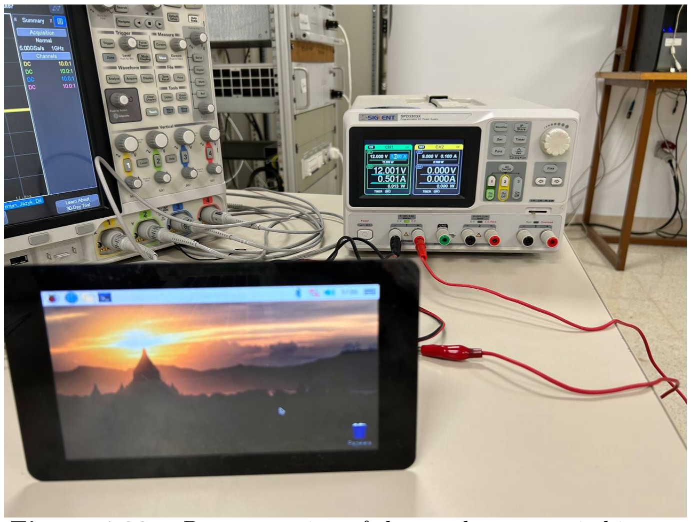


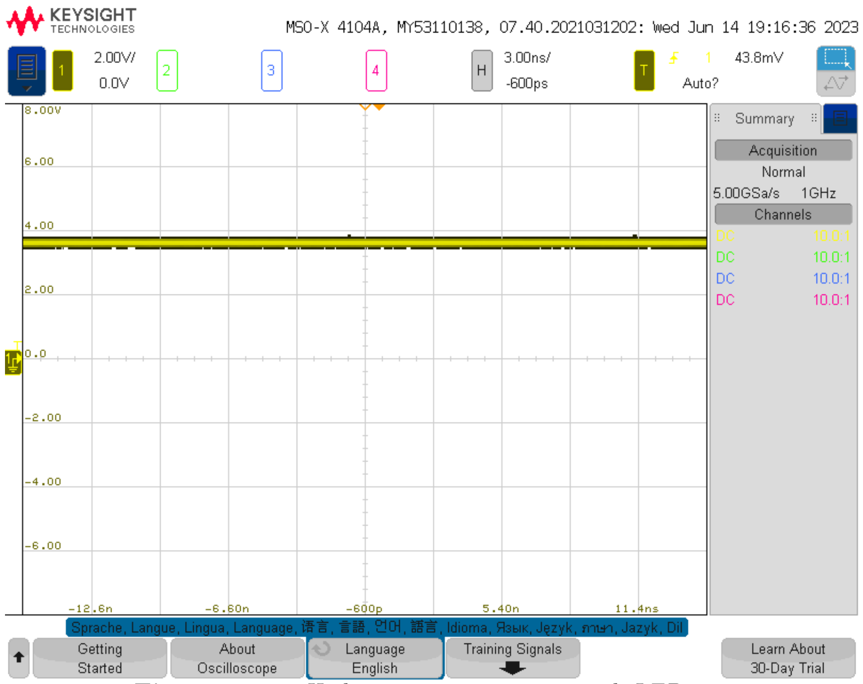
Figure 4.33 – *Demonstration of the touchscreen switching on*

4.3.4 Indicator LEDs

The main PCB contains two LEDs, LED1 (red) and LED2 (green). LED1's purpose is to signal when the Raspberry Pi 4 is busy. The red LED must switch on when pin 35 is in a high state. The code to test the mentioned action is the following:

```python

# Code to test LED1

import RPi.GPIO as GPIO  # Library to control GPIO pins

GPIO.setmode(GPIO.BCM)  # To respect the BCM numeration given by the
  GPIO number names

GPIO.setup(19,GPIO.OUT,initial=GPIO.HIGH) # Need to set as output
 GPIO 19 (pin 35) associated to the Busy net in the main PCB to a
 HIGH state

GPIO.setwarnings(False) # To disable warnings
```
Listado 4.1 – *LED1 Test*

By running this code, there should be a 3.3 V signal or similar (high state) in pin 35, as depicted in Figure 4.34.



MSO-X 4104A, MY53110138, 07.40.2021031202: Wed Jun 14 19:16:36 2023

1 2.00V/ 0.0V

H 3.00ns/ -600ps

T 1 43.8mV Auto?

Acquisition
Normal
5.00GSa/s 1GHz

Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 10.0:1

Figure 4.34 – *High state in pin 35 to switch LED1 on*

On the other hand, LED2 is used to know when there is a DC 5 V signal available in the main PCB, which means the Raspberry Pi 4 is switched on. In Figure 4.35, both LEDs working are shown.

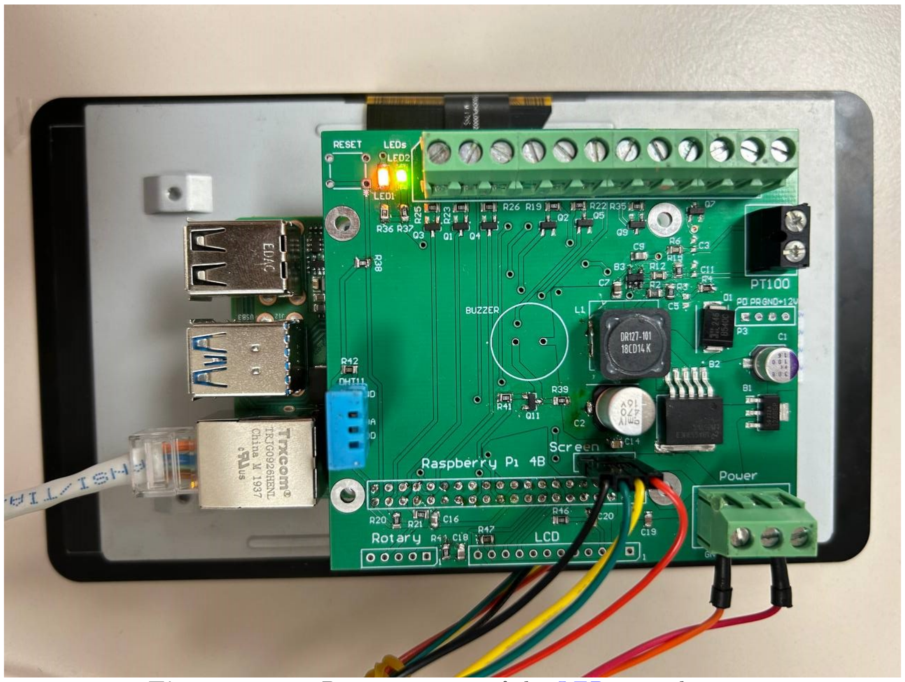


Figure 4.35 – *Demonstration of the LEDs switching on*

4.3.5 Process BLAS control signals

For this task, the Main PCB will be connected to the BLAS interface while connected to the Raspberry Pi. All of the BLAS control signals will be tested at the same time with a loop code. This code shares a default test. The code is the following:

```

import RPi.GPIO as GPIO
import time
import numpy as np

Fail_0 = 11
Fail_1 = 7
FlowPulses = 13
ILockFlow = 16
ILockPatchPanel = 20

Vsel_0 = 12
Vsel_1 = 5
DriverStart = 8
RFAmpStart = 6

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)


## IN PINS

GPIO.setup(Fail_0, GPIO.IN)  ## Pin assigned to collect signal Fail
 0 from BLAS
GPIO.setup(Fail_1, GPIO.IN)  ## Pin assigned to collect signal Fail
 1 from BLAS
GPIO.setup(FlowPulses, GPIO.IN)  ## Pin assigned to collect signal
 Flow Pulses from BLAS
GPIO.setup(ILockFlow, GPIO.IN)  ## Pin assigned to collect signal
 ILockFlow from BLAS
GPIO.setup(ILockPatchPanel, GPIO.IN)  ## Pin assigned to collect
 signal ILockPatchPanel from BLAS


## OUT PINS

GPIO.setup(Vsel_0, GPIO.OUT)  ## Pin assigned to transmit signal
 Vsel 0 to BLAS
GPIO.setup(Vsel_1, GPIO.OUT)  ## Pin assigned to transmit signal
 Vsel 1 to BLAS
GPIO.setup(RFAmpStart, GPIO.OUT)  ## Pin assigned to transmit signal
  RFAmpStart to BLAS
GPIO.setup(DriverStart, GPIO.OUT)  ## Pin assigned to transmit
 signal DriverStart to BLAS


## Algorithm for IN signals


while True:
```

4

```
    # Fail 0 and Fail 1 analysis

    if GPIO.input(Fail_0) and GPIO.input(Fail_1):
        print("Failure in both RF pallet")

    elif GPIO.input(Fail_0):
        print("Failure in RF pallet 0")

    else:
        print("Failure in RF pallet 1")

    # ILockFlow

    if GPIO.input(ILockFlow):
        print("Correct water flow")
        GPIO.output(RFAmpStart, True)
    else:
        print("Check water flow")
        GPIO.output(RFAmpStart, False)

    # ILockPatchPanel

    if GPIO.input(ILockPatchPanel):
        print("Correct configuration in patch panel")
        GPIO.output(DriverStart, True)
    else:
        print("Check patch panel configuration")
        GPIO.output(DriverStart, False)

    # FlowPulses

    timeout = 1   # [seconds]

    timeout_start = time.time()

    while time.time() < timeout_start + timeout:

        lasttime=0

        def callback_up(channel):
            global lasttime
            if lasttime==0:
                lasttime=time.time()
            else:
                now = time.time()
                gap=now-lasttime
                freq_PWM = np.round(1/gap)
                caudal_l_min = freq_PWM/50*60
                print("The PWM frequency of Flowpulses is", freq_PWM
, "Hz, which means a flow ",caudal_l_min,"l/min.")
                lasttime=now

        GPIO.add_event_detect(FlowPulses, GPIO.RISING, callback=
callback_up)
         while 1:
             time.sleep(1)

    # Vsel 0 and Vsel 1 analysis

    # Possible values for inner voltages of the amplifying modules.
    # Select by fixing the corresponding variable to 1.

    V_50 = 1  # Default value. Inner voltage of 55 V
    V_48 = 0  # Inner voltage of 50 V
    V_43 = 0  # Inner voltage of 45 V
    V_41 = 0  # Inner voltage of 40 V

    if (V_50 + V_48 + V_43 + V_41) == 1:
         if V_50== 1:
             print("50 V selected for the amplifying module. Default
value.")
         elif  V_48 == 1:
             print("48 V selected for the amplifying module.")
         elif  V_43 == 1:
             print("43 V selected for the amplifying module.")
         else:
             print("41 V selected for the amplifying module.")
```

Listado 4.2 – *BLAS Control Signals Test*

4.3.6 LCD Test

In order to test the WH2004D LCD, the following code is used:

```


# The wiring for the LCD is as follows:
# 1 : GND
# 2 : 5V
# 3 : Contrast (0-5V)*
# 4 : RS (Register Select)
# 5 : R/W (Read Write)        - GROUND THIS PIN
# 6 : Enable or Strobe
# 7 : Data Bit 0              - NOT USED
# 8 : Data Bit 1              - NOT USED
# 9 : Data Bit 2              - NOT USED
# 10: Data Bit 3              - NOT USED
# 11: Data Bit 4
# 12: Data Bit 5
# 13: Data Bit 6
# 14: Data Bit 7
# 15: LCD Backlight +5V**
# 16: LCD Backlight GND

#import
import RPi.GPIO as GPIO
import time

# Define GPIO to LCD mapping
LCD_RS = 4
LCD_E  = 14
LCD_D4 = 15
LCD_D5 = 17
LCD_D6 = 27
LCD_D7 = 22
LED_ON = 18

# Define some device constants
LCD_WIDTH = 20    # Maximum characters per line
LCD_CHR = True
LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

def main():
  # Main program block

  GPIO.setmode(GPIO.BCM)       # # To respect the BCM numeration
 given by the GPIO number names
  GPIO.setup(LCD_E, GPIO.OUT)  # E
  GPIO.setup(LCD_RS, GPIO.OUT) # RS
  GPIO.setup(LCD_D4, GPIO.OUT) # DB4
  GPIO.setup(LCD_D5, GPIO.OUT) # DB5
  GPIO.setup(LCD_D6, GPIO.OUT) # DB6
  GPIO.setup(LCD_D7, GPIO.OUT) # DB7

  GPIO.setup(LED_ON, GPIO.OUT) # Backlight enable
  GPIO.setwarnings(False)     #disable warnings
  pi_pwm = GPIO.PWM(LED_ON,1000)     #create PWM instance with
 frequency
  pi_pwm.start(4)        #start PWM of required Duty Cycle


  # Initialise display
  lcd_init()

  # Toggle backlight on-off-on
  lcd_backlight(True)
  time.sleep(0.5)
  lcd_backlight(False)
  time.sleep(0.5)
  lcd_backlight(True)
```

```
    time.sleep(0.5)

    while True:

      # Send some centred test
      lcd_string("GranaSAT",LCD_LINE_1,2)
      lcd_string("BLAS CONTROL LCD",LCD_LINE_2,2)
      lcd_string("Andoni Perez Segura",LCD_LINE_3,2)
      lcd_string("MUIT TFM 2023",LCD_LINE_4,2)

      # Blank display
      # lcd_byte(0x01, LCD_CMD)



def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font
  size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)

def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #        False for command

  GPIO.output(LCD_RS, mode) # RS

  # High bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
  if bits&0x80==0x80:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()

  # Low bits
  GPIO.output(LCD_D4, False)
```

4

```
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()

def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)

def lcd_string(message,line,style):
  # Send string to display
  # style=1 Left justified
  # style=2 Centred
  # style=3 Right justified

  if style==1:
    message = message.ljust(LCD_WIDTH," ")
  elif style==2:
    message = message.center(LCD_WIDTH," ")
  elif style==3:
    message = message.rjust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

def lcd_backlight(flag):
  # Toggle backlight on-off-on
  GPIO.output(LED_ON, flag)

if __name__ == '__main__':

  try:
    main()
  except KeyboardInterrupt:
    pass
  finally:
    lcd_byte(0x01, LCD_CMD)
    lcd_string("Goodbye!",LCD_LINE_1,2)
    GPIO.cleanup()
```

Listado 4.3 – *LCD Test*

In Figure 4.36, the LCD working powered by the Main PCB and with its code, it is shown.

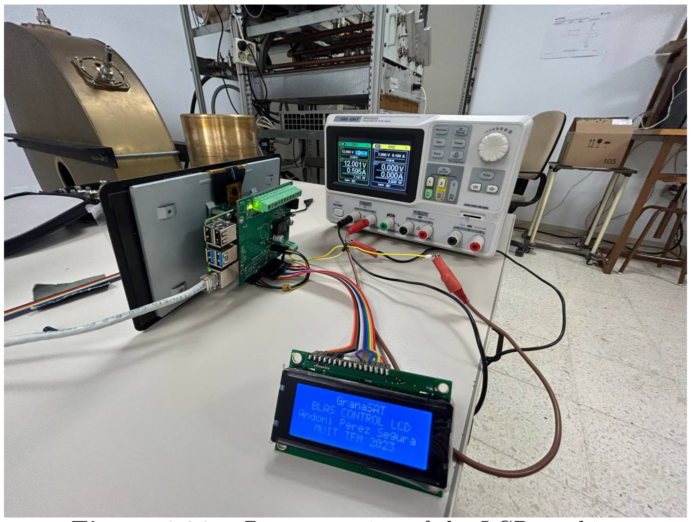


Figure 4.36 – *Demonstration of the LCD working*

4.3.6.1 RF Voltmeter

As mentioned before, the RF voltmeter application is to be able to process the 175 MHz signals, coming from the directional couplers. In Figure 4.39, the circuit design is depicted. The RF signal will be introduced using 50 Ω BNC female ports. It is crucial to let the RF signal always see a 50 Ω impedance, so there are no impedance mismatches. That's why there is a resistance of this value just before the decoupling capacitor. This capacitor is used to separate the diode polarizing DC signal coming from the power supply, and the RF signal. Then, the diode is used to have a half-wave signal, and the low-pass filter to finally correct the signal.

The circuit has been simulated in LTSpice, Figure 4.37. In Figure 4.38, the results given are shown for four different power inputs.

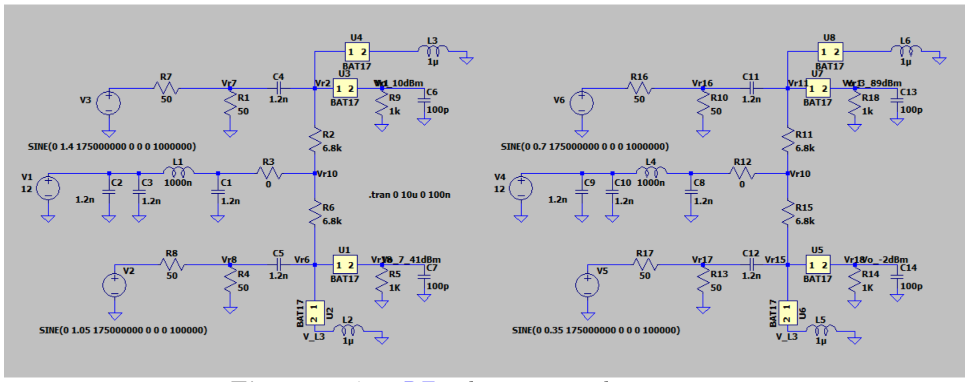

SINE(0 1.4 175000000 0 0 0 1000000)
SINE(0 1.05 175000000 0 0 0 100000)
SINE(0 0.7 175000000 0 0 0 1000000)
SINE(0 0.35 175000000 0 0 0 100000)
.tran 0 10u 0 100n
BAT17
Vr10
V_L3

Figure 4.37 – *RF voltmeter simulation circuit*

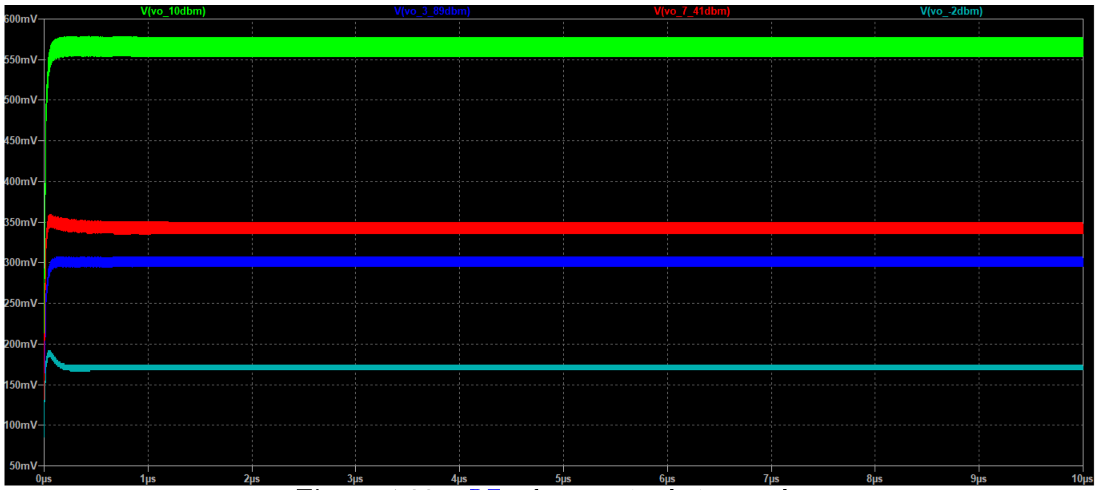


Figure 4.38 – *RF voltmeter simulation results*

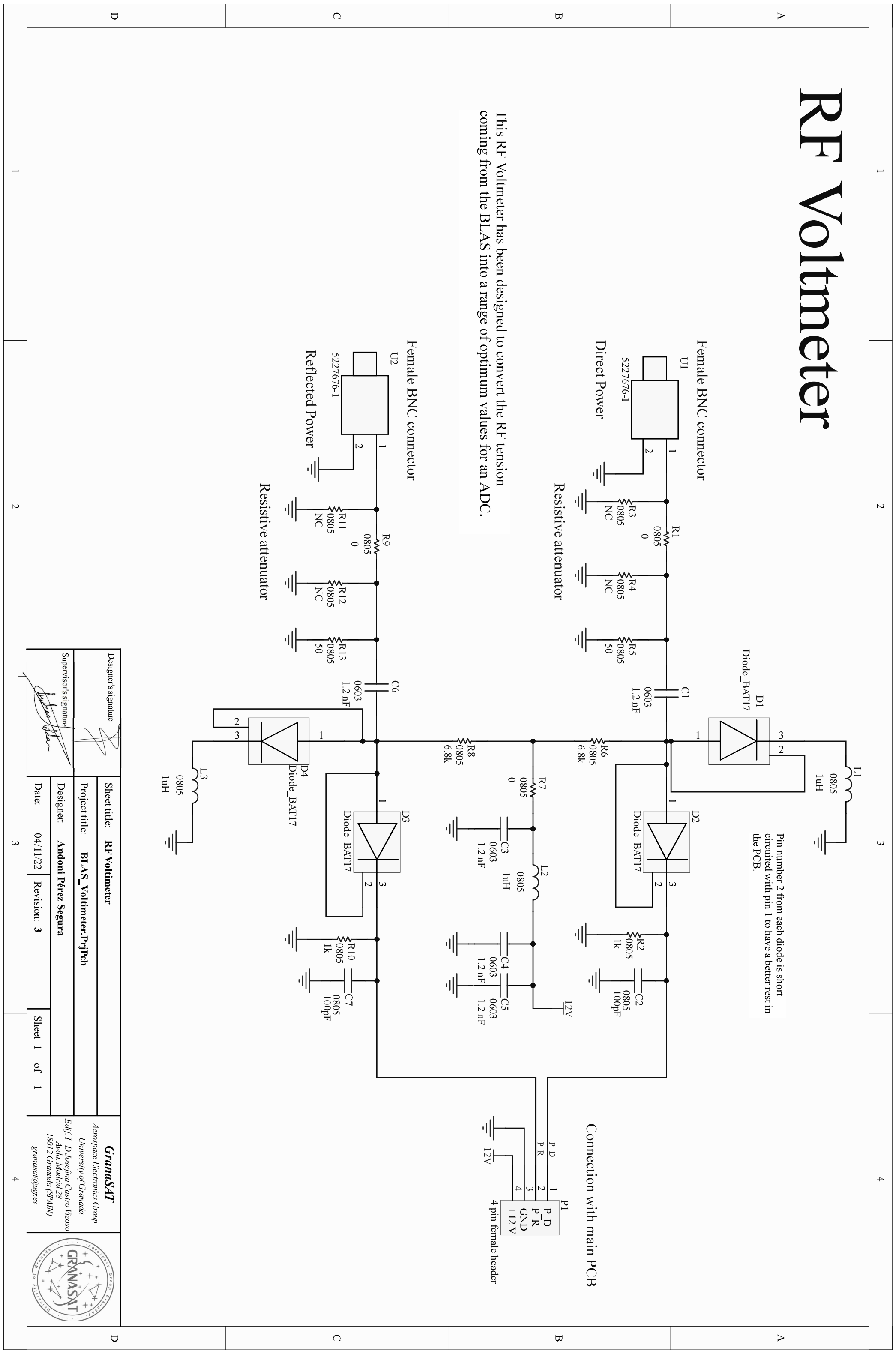


Figure 4.39 – *Voltmeter SCH file*

Once the circuit designing process is finished, the placement of the components and routing starts. As this PCB contains RF signals, the track width is crucial, so the tracks have an impedance of 50 Ω. In order to achieve this, the track width must be 2.78 mm considering a frequency of 175 MHz, 1.51 mm of dielectric thickness, and 35 mm of copper thickness. That’s why the tracks before the capacitor decoupling and after, are so different in width, Figure 4.40.

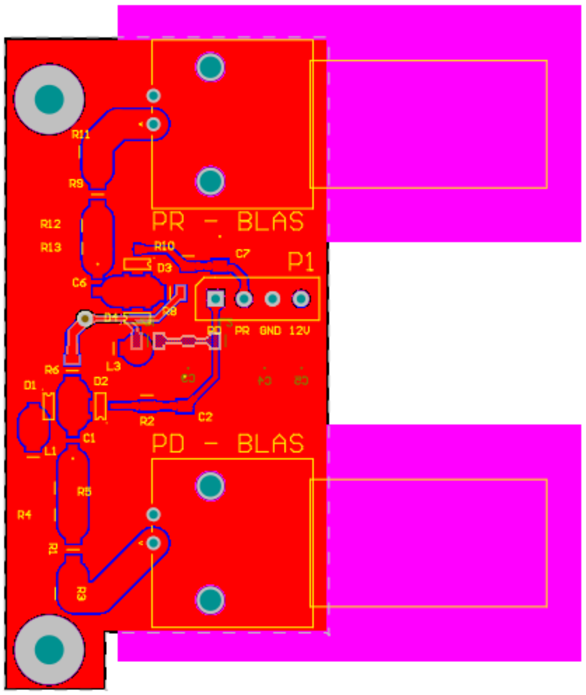


Figure 4.40 – *RF voltmeter top layer*

4.3.6.2 Assembly and testing - RF Voltmeter

As in previous sections, the assembly steps were already detailed, only figures of the voltmeter will be shown.

Figure 4.41 – *RF voltmeter with solder paste applied*

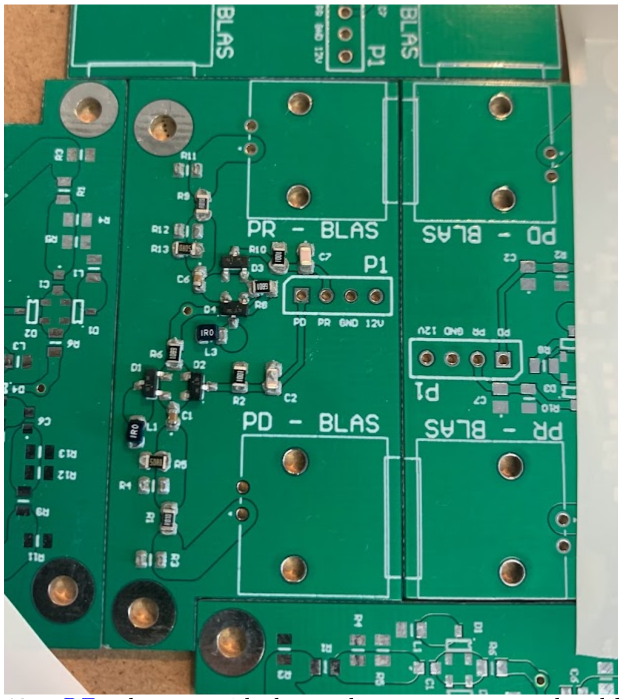


Figure 4.42 – *RF voltmeter with the top layer components placed before oven*

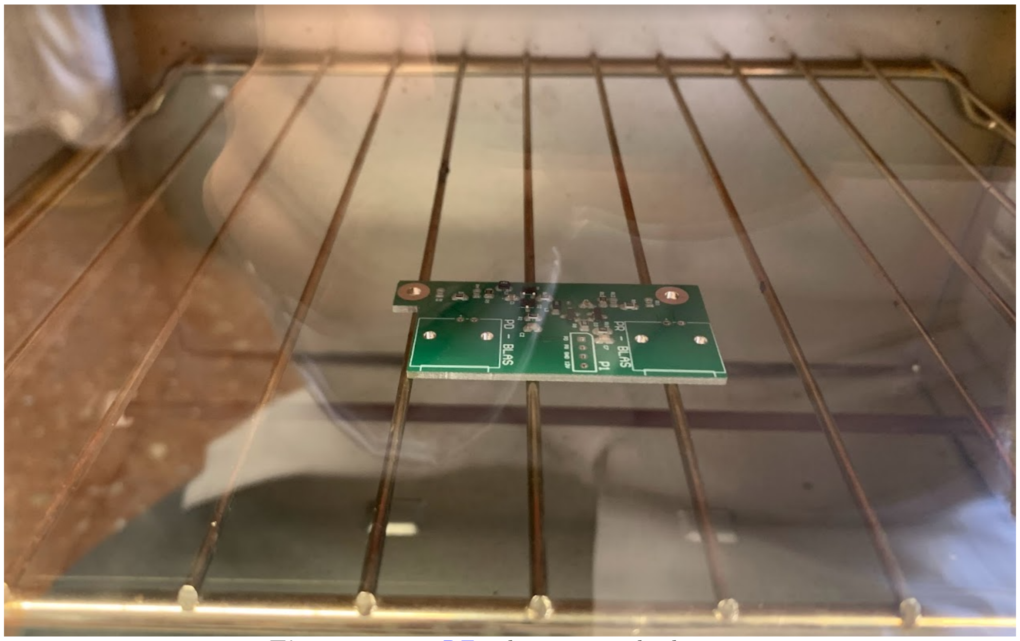


Figure 4.43 – *RF voltmeter inside the oven*

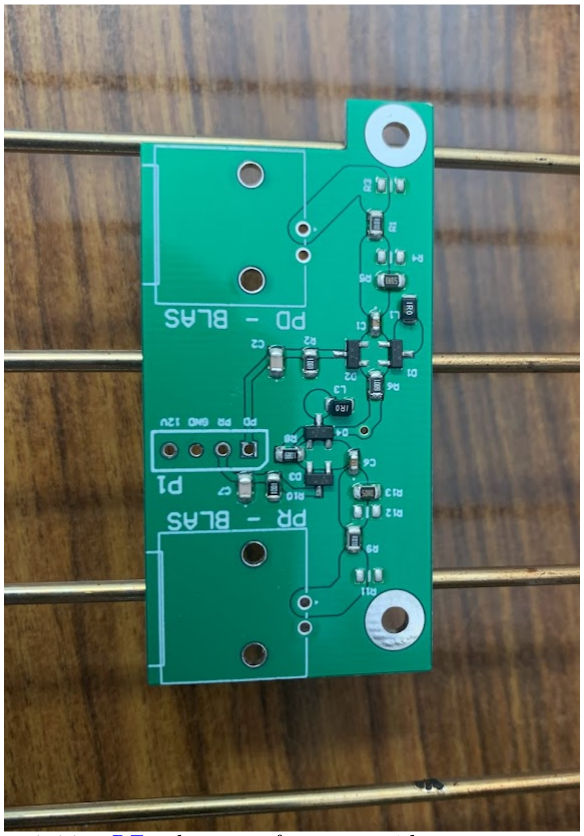


Figure 4.44 – *RF voltmeter after oven with components soldered*

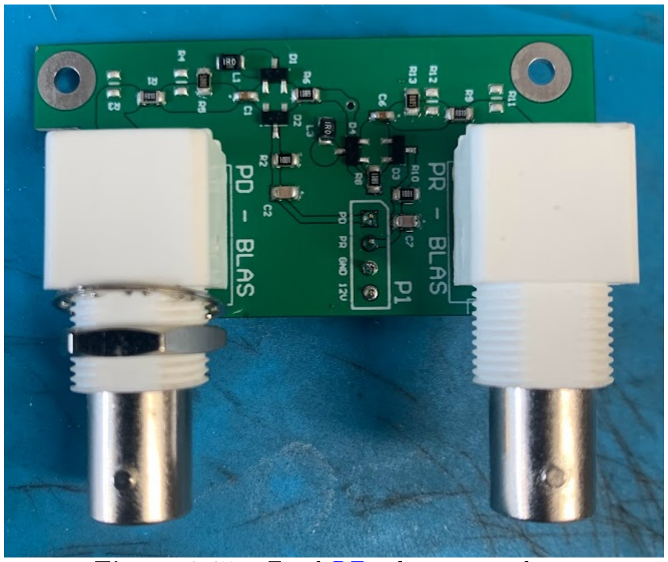


Figure 4.45 – *Final RF voltmeter top layer*

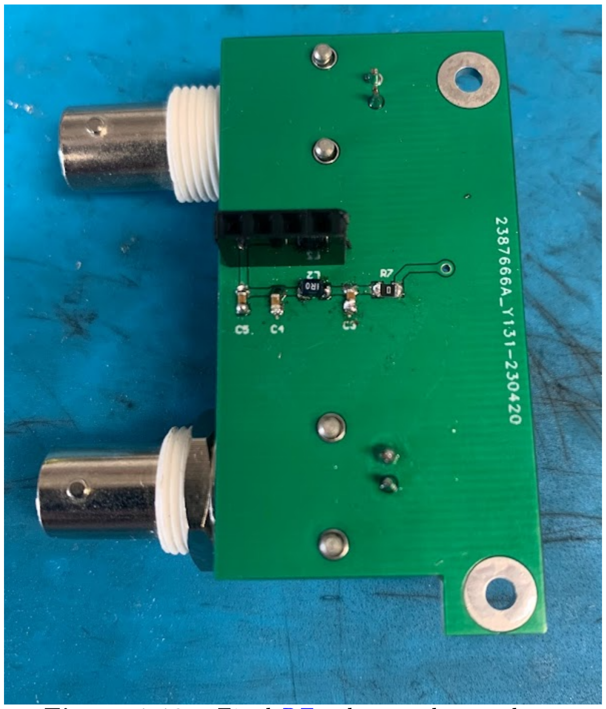


Figure 4.46 – *Final RF voltmeter bottom layer*

Once, the PCB is finished, testing will be done, to check the behaviour depending on the input signal level and its similarity to the preliminary simulations. Firstly, it is relevant to remember that the main goal of this voltmeter is to convert the RF signals coming from the BLAS' directional couplers, to DC signals to transmit them to an ADC subsequently. The mentioned test will be executed as follows. For the RF input signals the signal generator 2022D from Marconi Instruments will be used, and for the 12 V supply the DC power supply SPD3303X from Siglent. Finally, to measure the output signals from the voltmeter the MSO-X-4104A oscilloscope from Keysight was chosen. The test block diagram is shown in Figure 4.47, where it can be seen how every device is interconnected. The RF output from the signal generator is connected to the required BNC input port from the RF voltmeter, while the other one is adapted using a 50 Ω load. Using the pinout designed for connecting this PCB with the main one, 12 V and GND is supplied to the circuit and the DC signals are measured with the oscilloscope.

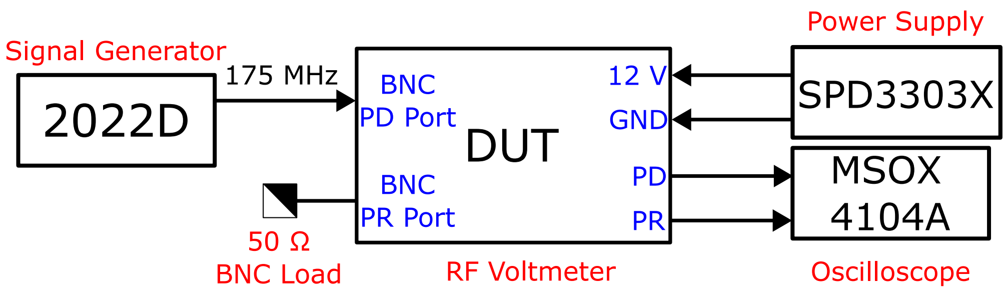


Figure 4.47 – *2 male BNC connectors*

4.3.7 Measurement system

As stated, the 2022D signal generator will generate the input signal. The input ports are female BNC type, so the pigtail used is a male BNC one. The output port of the signal generator is N type, which generates the need to use an adaptor BNC-N, shown in Figure 4.48.

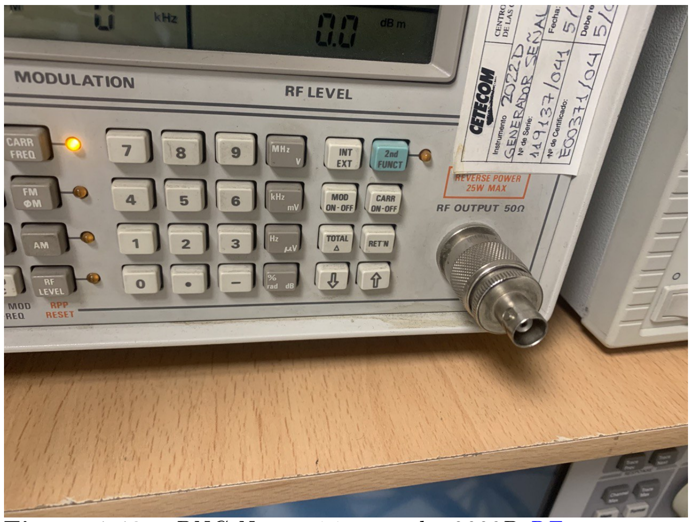


Figure 4.48 – *BNC-N transition on the 2022D RF output port*

As the mentioned pigtail used is of a considerable length and the 2022D generator is quite old, the output signal from the generator will be introduced into the N9020A signal analyzer to know more precisely the input power to the PCB. Fixing an output power of 0 dBm, -1.525 dBm is obtained in the signal analyzer, so there is 1.6 dB of attenuation combining the pigtail and the signal generator. In Figure 4.49, the spectrum from the signal analyzer is depicted, data obtained via the Python library coded for this project. It can be observed the RF voltmeter has a linear behaviour between the input and output voltage.

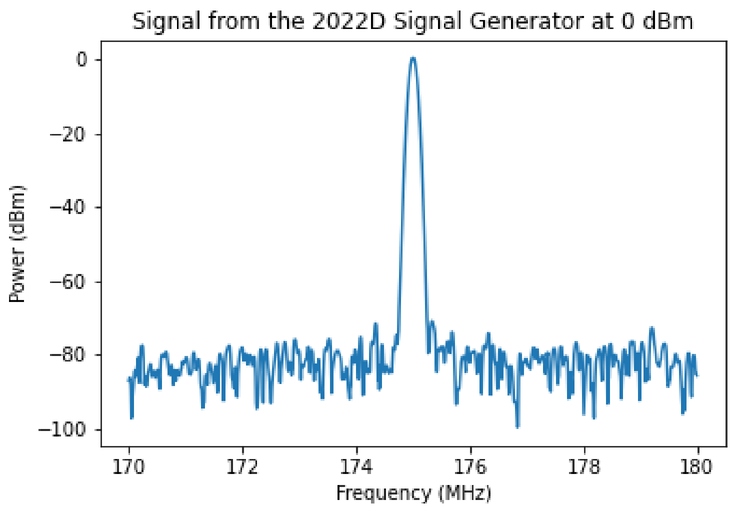


Figure 4.49 – *2022D's 0 dBm output signal measured by the N92020A signal analyzer*

4.3.8 Voltmeter measurements

An input power sweep will be done to characterize the behaviour of the voltmeter in both BNC input ports. BLAS is prepared for achieving 63 dBm (2 kW) of transmitted power, considering the 53 dB attenuation from the directional couplers, this leaves 10 dBm of power for the input of the voltmeter. Moreover, the sweep will be done between 10 dBm and -15 dBm. Figure 4.50 describes how the RF voltmeter was prepared to be measured. The port that is not going to be used needs to be adapted, so a 50 Ω load is placed. The voltmeter is powered with 12 V from the DC power supply using the pins designed for its connection to the main PCB. Finally, using a BNC pigtail, the voltmeter is connected to the 2022D signal generator for the 175 MHz input signal.

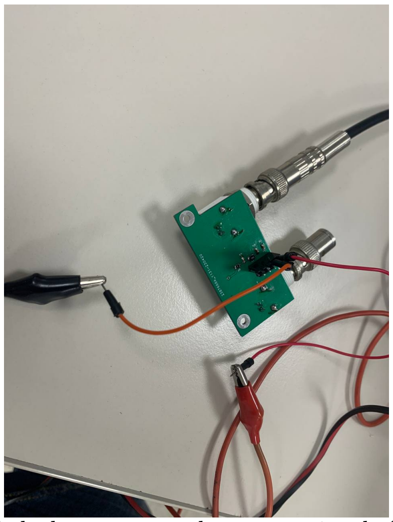


Figure 4.50 – *Method to measure the output signals from RF voltmeter*

In Table 4.2, the output voltage measured is represented, depending on the input power expressed in dBm and V. With the aim of easing the reading of Table 4.2, in Figure 4.51 the measurements have been plotted.

P_{in} (dBm)	V_{in} (V)	V_{out} (mV)
10	1	690
9	0.89	620
8	0.79	560
7	0.71	510
6	0.63	460
5	0.56	420
4	0.5	390
3	0.45	360
2	0.4	320
1	0.35	310
0	0.32	290
-1	0.28	260
-2	0.25	240
-3	0.22	220
-4	0.2	210
-5	0.18	190
-6	0.16	190
-7	0.14	170
-8	0.13	170
-9	0.11	170
-10	0.1	160
-11	0.09	160
-12	0.08	160
-13	0.07	140
-14	0.06	140
-15	0.06	140

Table 4.2 – *RF voltmeter output voltage depending on the input power expressed in dBm and V.*

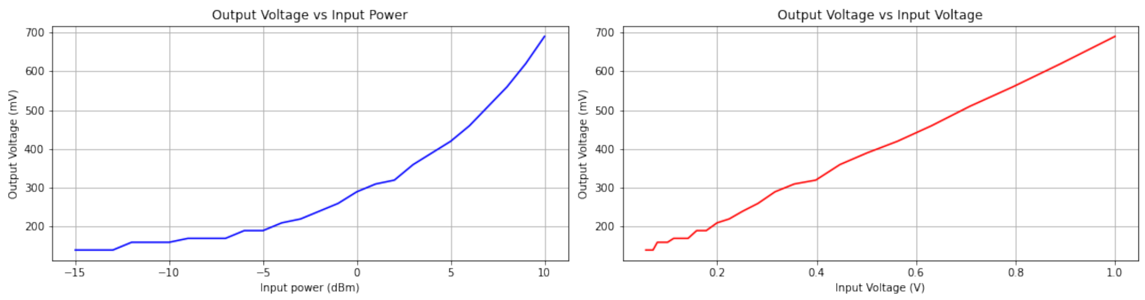
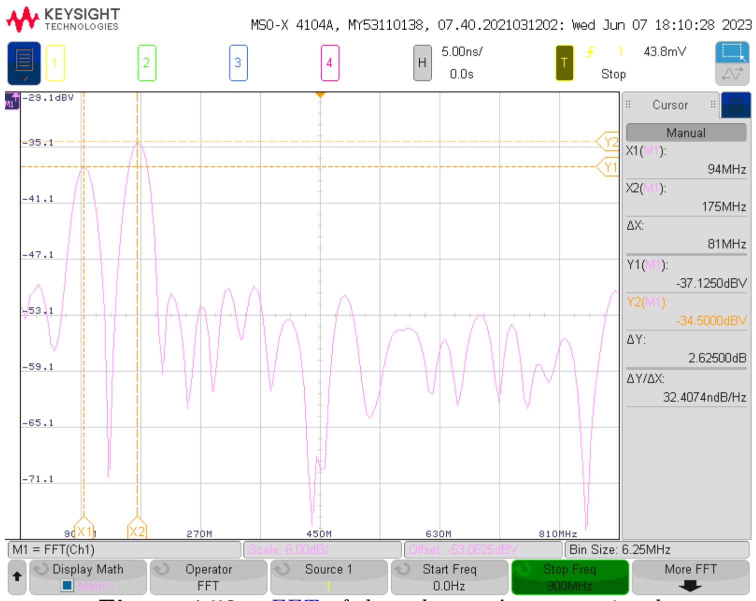


Figure 4.51 – *RF voltmeter's output voltage behaviour*

4.3.9 Simulation comparison

4.3.10 Characterization

In order to check if the output signal is DC and has no AC component, the FFT of the mentioned signal was calculated using the MSO-X-4104A oscilloscope. The output signal to analyse is of 690 mV. This transformation can be seen in Figure 4.52. Most of the power is concentrated at 175 MHz, which is logical as it is the original frequency of the input signal. This frequency has a power of -34.5 dBV (19 mV), which can be seen as insufficient to take it into account. However, compared to the total power of the signal, -3.22 dBV (690 mV), it can slightly differ the final voltage measure. Furthermore, another peak of power can be seen at 94 MHz. This can be associated with the FM transmissions, as when there is no input signal this power is still measured. As the 175 MHz peak power, the power at 94 MHz is too low to take IT into account, -37.125 dBV (14 mV), and it may slightly differ the final voltage measures. Finally, it has been proven that the output signal of the RF voltmeter has a small AC component, which can disturb if high precision is required.



X1(M1): 94MHz
X2(M1): 175MHz
ΔX: 81MHz
Y1(M1): -37.1250dBV
Y2(M1): -34.5000dBV
ΔY: 2.62500dB
ΔY/ΔX: 32.4074ndB/Hz

M1 = FFT(Ch1) Scale: 6.00dB/ Offset: -53.0625dBV Bin Size: 6.25MHz
Operator FFT · Source 1 · Start Freq 0.0Hz · Stop Freq 900MHz

Figure 4.52 – *FFT of the voltmeter's output signal*

As in every RF device, it is of high importance that the device is adapted to avoid the reflection of the input signals. This reflection could affect the behaviour of the directional couplers, as the output ports of these couplers are supposed to be adapted, and the behaviour of the proper RF voltmeter as not all the power is being transmitted. A way to determine if the RF voltmeter is adapted is measuring its reflected S parameters. In order to do so, the E5071C vector network analyzer from Agilent Technologies has been used. Ideally, if a one port device is adapted, its reflected S parameter, S_{11}, has to be $-\infty$ dB. Nevertheless, this value is complicated to achieve in real devices. Normally, a port is said to be adapted when S_{11} is somewhere near -25 dB, which means only 5% of the total power is being reflected.

It is important to remember that the RF voltmeter was designed to work at 175 MHz, the track width of the PCB was thought to work in this frequency. So, the behaviour of the RF voltmeter will differ depending on the frequency used. To fully characterize the voltmeter, both input ports were used. Each input port will act as characterization ports, so it is expected that both S_{11} and S_{22} are around -25 dB, to be adapted and the transmission S parameters, S_{21} and S_{12}, to be really low to ensure each port is independent of the other.

As both ports are female BNC type, and the E5071C VNA has N type ports, ideally a BNC-N pigtail needs to be used. As a pigtail of these characteristics was not available in the laboratory, the male N-type pigtail used for the cavity characterization was used. To be able to connect this pigtail to the RF voltmeter, a transition from male N to female BNC was chosen. In Figure 4.53, this transition is depicted.

(a) *BNC to N transition*

(b) *BNC to N transition*

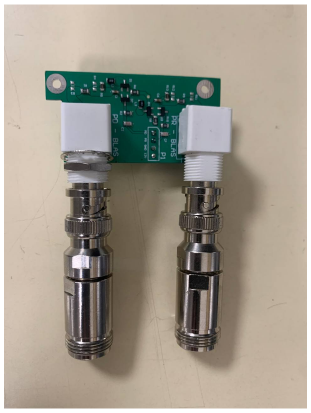

(c) *RF voltmeter and BNC to N transition*

Figure 4.53 – *BNC to N transition needed to measure the RF voltmeter with the VNA*

In Figure 4.54, the way the RF voltmeter is characterized is shown, with both input ports connected to the VNA, and the power supply connected to introduce the 12 V needed for polarization, using the pins designed for connecting this PCB to the main one.

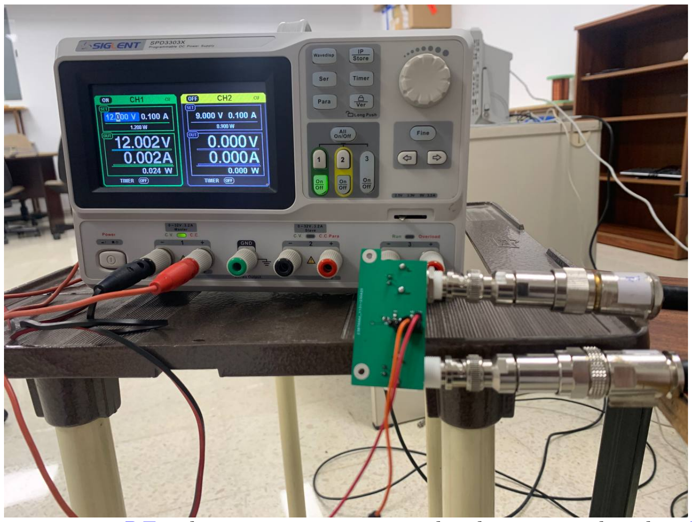


Figure 4.54 – *RF voltmeter connections to be characterized with a VNA*

Firstly, the characterization was done between 10 MHz and 8 GHz, to probe the different behaviour depending on the frequency, results shown in Figure 4.55. The desired performance is achieved in the frequencies between both vertical black lines, around 175 MHz, as expected.

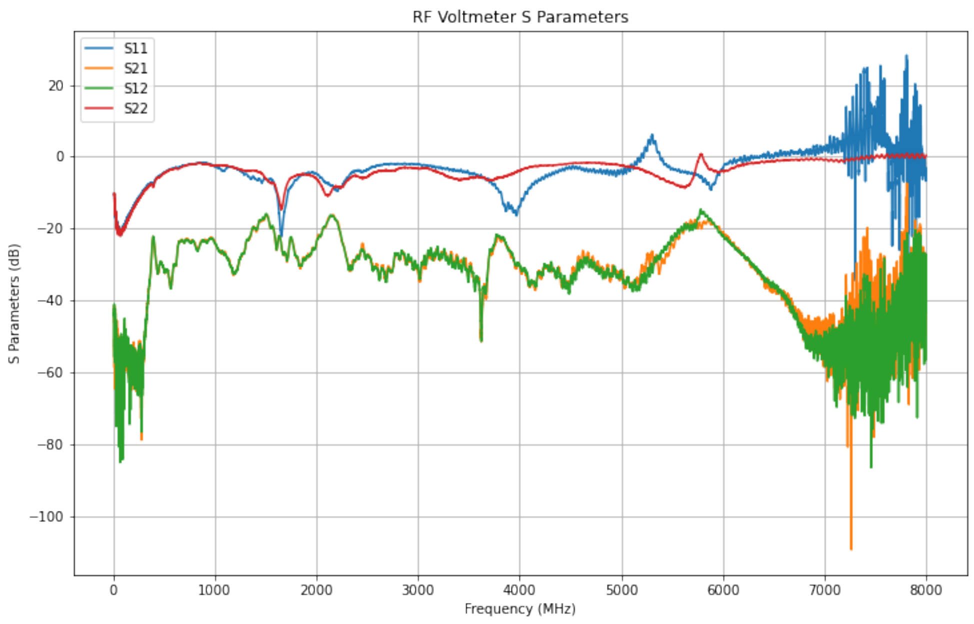


Figure 4.55 – *RF voltmeter full band characterization*

Next, to have a more precise picture of the voltmeter's behaviour, a narrow bandwidth characterization was executed, between 170 MHz and 180 MHz to know how it will work with the frequencies used in BLAS. Figure 4.56 shows the S parameters obtained. The voltmeter is working properly in the bandwidth desired, as the reflected S parameters are around -17 dB, which means 14 % of the power will be reflected. Considering the PCB is a low-cost one, and there are better dielectric materials in the market and this PCB is a first approach, there are not bad results. Moreover, both ports are totally independent as the transmission S parameters are extremely low, somewhere near -60 dB. All the results were obtained remotely using the dedicated Python libraries elaborated for this project.

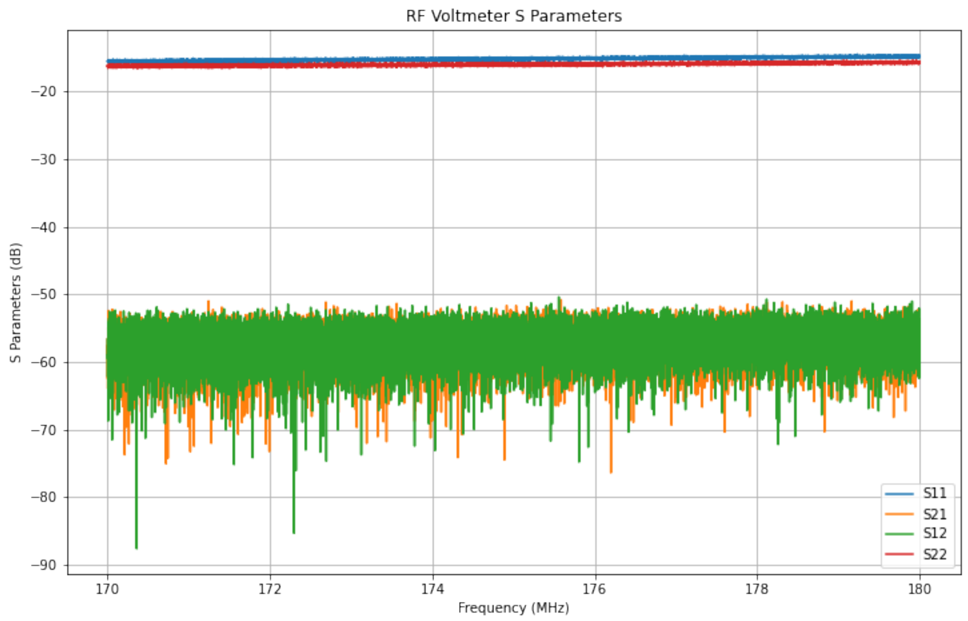


Figure 4.56 – *RF voltmeter narrow band characterization*

4.4 EPICS

4.4.1 Introduction

The Experimental Physics and Industrial Control System (EPICS) is a collection of tools and software components which goal is to create distributed control systems. EPICS' most relevant capabilities are the following [4]:

- Remote control and monitoring of facility equipment
- Facility mode and configuration control
- Alarm reporting, logging and reporting
- Closed loop control
- Modeling and simulation
- Automatic sequencing of operations
- Data analysis

EPICS was designed to be able to control enormous scientific installations, but can be scaled to small systems. One characteristic big systems have, is the huge amount of data that has to be processed, between several devices, and needs to be failure-tolerant. That means if one of the sensors, PCs, or any piece of equipment stops working the control system must be capable of maintaining most of the system alive. For more modest installations, EPICS can be used without requiring complex infrastructure components.

One of the most significant advantages EPICS has, is the worldwide collaboration that exists by being a free open-source software. Moreover, members of the design team are always prone to help and listen to new ideas in order to improve their implementation.

4.4.2 Architecture

EPICS provides the capability of creating servers and client applications. Data is accessed through these servers that can be reached locally or over a network by clients. These clients can store, manipulate and display data as desired. This can be done using the various types of client software available, that go from graphical or command-line user interface to powerful services for data management.

An EPICS based control system consists of the following elements:

- **IOC:** Input/Output Controller is the I/O server of EPICS. It consists of components such as databases and network communication that can be handled by almost any computing platform. For example, a regular desktop computer, or even low-cost hardware such as a Raspberry Pi.

- **CWS:** Client Workstation is a device that can handle EPICS tools and client applications as the user interface tools mentioned before. Usually, these CWSs are run in devices with "typical" operating systems such as Windows, Linux or MacOs.

- **LAN:** A standard local area network, Ethernet based, is needed to interconnect the IOCs with the CWSs.

An example of a simple EPICS control system is depicted in Figure 4.57. As stated previously the LAN is used to interconnect all the other elements from the control system. There can be as many IOCs and CWSs needed, and they can be removed or added without harming the system.

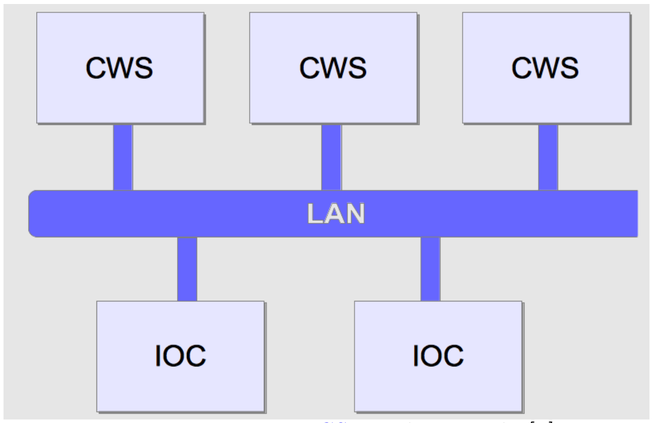


Figure 4.57 – *EPICS simple example [4]*

Furthermore, other basic components are reachable apart from these "classical" ones. There are more servers or services for data processing not linked to process I/O information or attached to hardware. These servers offer the possibility to be used as simulation, calibration, or computing services, such as particle beam modeling, which is a really extended practice.

The communication regarding the mentioned services is done thanks to the next two EPICS components: Channel Access (CA) and pvAccess (PVA). The structure and protocols implemented, so the different CWSs and IOCs speak the same language, are provided by the CA and PVA, which will be discussed in further subsections.

The main characteristic of EPICS is it is process-variable based. Rather than behaving as an object-oriented program, modeling control system devices as objects, they are assumed as data entities that describe single aspects of the device or process that is being controlled. The name associated with these data entities is "process variable" (PV). PV can describe any type of data required, for instance, a spectrum analyzer can have associated several PVs managing power values, a description of the device or bandwidth.

4.4.3 IOC

Understanding the behaviour of EPICS IOC is crucial to exploit the several advantages this software offers. The different software parts that build up an IOC are depicted in Figure 4.58 and are the following:

- IOC Database: This database is in charge of containing the records used in the IOC. **Each record hosts a PV**.
- Record support: There are different types of records, each with characteristic functionalities. This component implements the routines required for each type.
- Scanners: The method to process the records from the database.
- Device support: Routines bind I/O data to the database records.
- Device Drivers: Handle access to external devices.
- CA or PVA: Communicate the other EPICS components with the IOC, via LAN.
- Sequencer: A finite state machine. External module not included in the EPICS distribution.

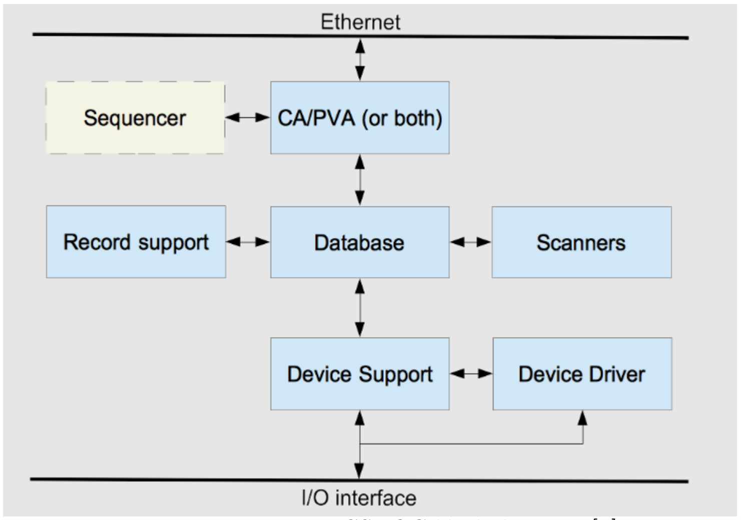


Figure 4.58 – *EPICS IOC block diagram* [4]

4.4.3.1 IOC Databases

As Figure 4.58 reflects, a record database can be seen as the heart of an EPICS IOC. Unlike other common databases, such as SQL, record databases are memory resident, not stored on permanent memory devices. The IOC's behaviour is determined by the records stored in the database, which can be of any number and type. The most common types are the following.

- Analog Input and Output (ai and ao): These records support the reading of data from hardware devices, transformation into engineering units, and setting of alarms or limits.
- Binary Input and Output (bi and bo): Normally used to indicate the status of a device or parameter, like Open/Closed, On/Off, and so on.
- Calc and Calcout: Perform calculations based on other records values.

As records define the IOC's behaviour, "fields" configure a record's behaviour. Fields are metadata that some are common to any type of record, but some are specific for certain types of records. So, every record has a name and every field has a name. In order to avoid undesired errors in CA, every record name has to be unique across all the IOCs, so it is possible to access to their different fields. An example of record is shown in code 4.4.

```

record(ai, "Cavity1:T") #type = ai, name = “Cavity1:T"
{
  field(DESC, “Cavity Temperature") #description
  field(SCAN, “1 second") #record update rate
  field(DTYP, “XYZ ADC") #Device type
  field(INP, “#C1 S4") #input channel
  field(PREC, “1") #display precision
  field(LINR, “typeJdegC") #conversion spec
  field(EGU, “degrees C") #engineering units
  field(HOPR, “100") #highest value on GUI
  field(LOPR, “0") #lowest value on GUI
  field(HIGH, “65") #High alarm limit
  field(HSV, “MINOR") #Severity of “high" alarm
}
```

Listado 4.4 – *EPICS record example [4]*

A record is defined by a type and a name. Inside each record, there are a set of fields, each defined by a type and a value according to it. Examples of fields common to every record are:

- DESC: Offers a description of the record
- SCAN: Indicates the record's update rate
- DTYP: Indicates the type of device the record is a data entity

Moreover, records can be associated with each other. For instance, the execution of one record could into the execution of another one, and the value of one record could be an input parameter of another one. These links convert the database into a programming tool. In resume, a record is an object with:

- A unique name
- A behaviour defined by its type
- Controllable properties (fields)
- Optional associated hardware I/O (device support)
- Links to other records

4.4.3.2 Scanner

As mentioned previously, inside each IOC there is a scanning software component that is in charge of processing the set of records available in the database, depending on the field "SCAN" that each record holds. EPICS provides four basic scanning methods.

- Periodic: A record can be fixed to be processed periodically at a specified frequency.

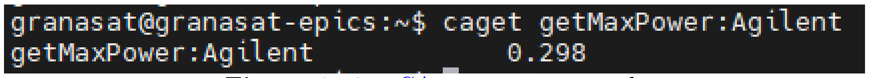
- Event: A record is processed when a IOC component posts an event, such as a new humidity value or a change in a certain parameter.
- I/O Event: As the previous type, but it is based on external events, like processor interrupts.
- Passive: Processed when other records, linked to the passive one, are processed or when it changes its proper value via CA.

4.4.4 Channel Access (CA)

CA is based on a client/server model on TCP/IP. Every IOC implements a CA server used to communicate with the other CAs implemented in the CWSs of the network.

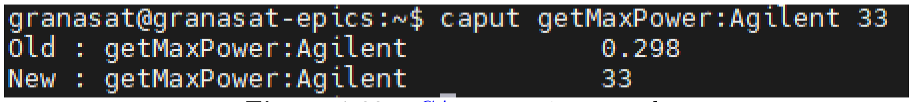
The basic client services provided by CA are:

- Search: Locate the desired PV among all the IOCs available in the network.
- Get: Get the value of a certain PV. An example is depicted in Figure 4.59, where is asked to get the value of a PV called "getMaxPower:Agilent".
- Put: Modify the value of a certain PV. An example is depicted in Figure 4.60, where is asked to change the value of a PV called "getMaxPower:Agilent".
- Monitor: Whenever a PV changes its value, the client receives an update. An example is depicted in Figure 4.61, where is asked to be notified every time the value from the PV called "getMaxPower:Agilent" changes.



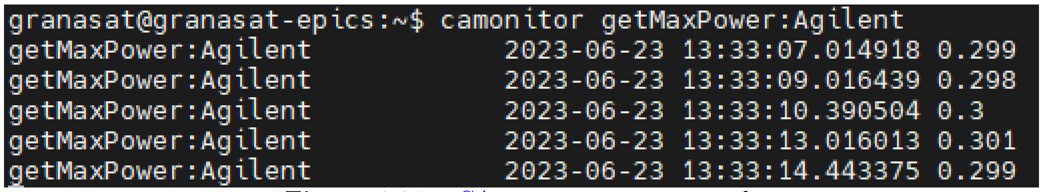
```
granasat@granasat-epics:~$ caget getMaxPower:Agilent
getMaxPower:Agilent            0.298
```

Figure 4.59 – *CA get service example*



```
granasat@granasat-epics:~$ caput getMaxPower:Agilent 33
Old : getMaxPower:Agilent            0.298
New : getMaxPower:Agilent            33
```

Figure 4.60 – *CA put service example*



```
granasat@granasat-epics:~$ camonitor getMaxPower:Agilent
getMaxPower:Agilent            2023-06-23 13:33:07.014918 0.299
getMaxPower:Agilent            2023-06-23 13:33:09.016439 0.298
getMaxPower:Agilent            2023-06-23 13:33:10.390504 0.3
getMaxPower:Agilent            2023-06-23 13:33:13.016013 0.301
getMaxPower:Agilent            2023-06-23 13:33:14.443375 0.299
```

Figure 4.61 – *CA monitor service example*

4.4.5 EPICS implementation

Foremost, the EPICS software must be installed, guide in Appendix F. Once, the software is installed, the first step on EPICS is creating an IOC, guide to do this in Appendix **??**. However, before creating an IOC, it is important to have clear what are the goals that want to be achieved. In this case, EPICS will be used to monitor and control two signal analyzers and the BLAS control signals through a Raspberry Pi 4, Figure 4.62.

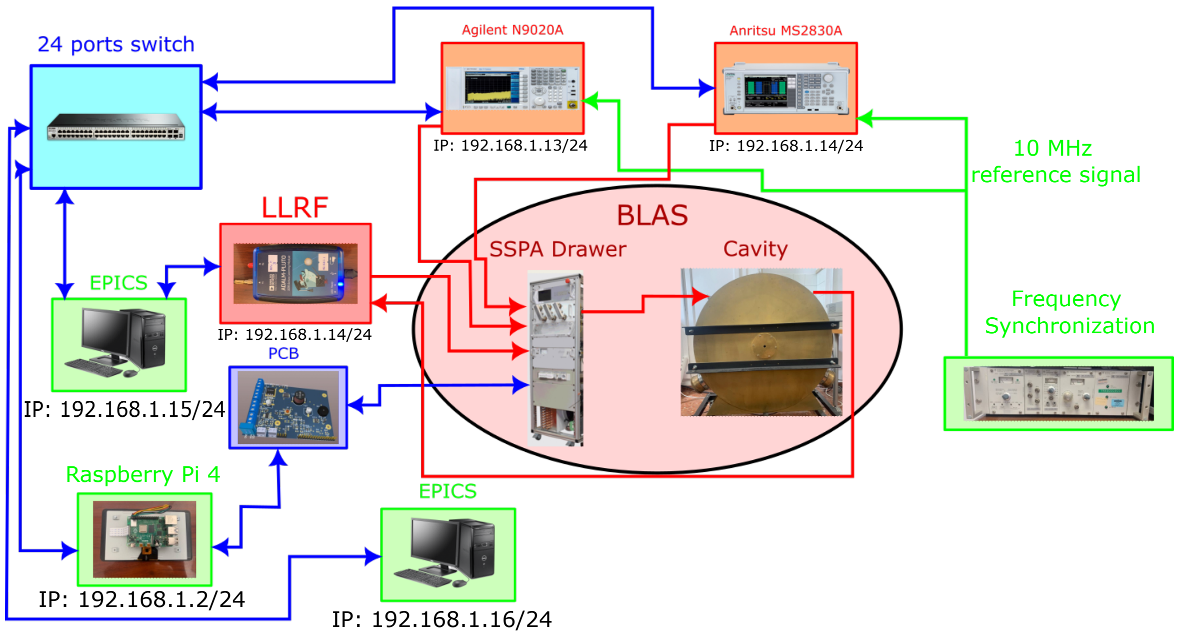


Figure 4.62 – *System control block diagram*

As the IOC will need to communicate via TCP with other devices (the two analyzers to communicate with the SCPI commands), the modules *StreamDevice* and *AsynDriver* need to be installed and added to the IOC when creating. Both steps are covered in Appendixes F and ??. The communication between the Raspberry Pi 4 and the IOC will be done via the Channel Access (CA), without needing any extra module.

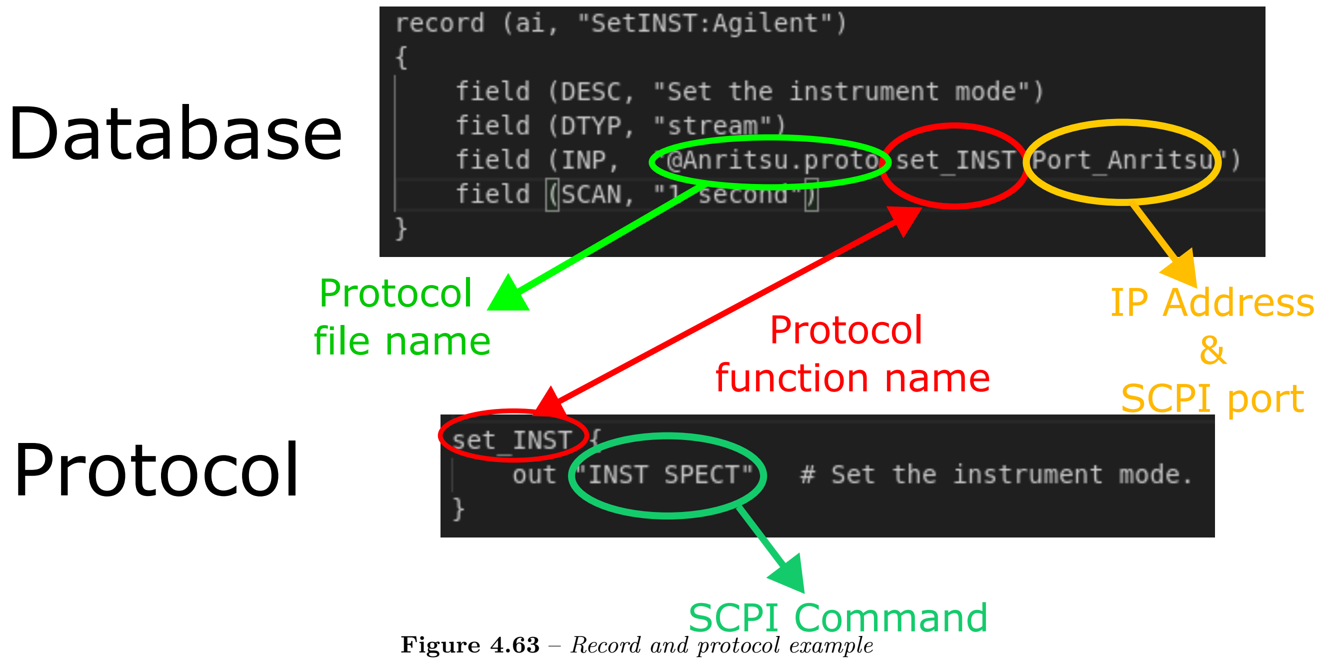
The three most relevant files inside an *Streamdevice* IOC are:

- The database (filename.db): The database of the IOC. This file contains all the records, so it dictates the behaviour of the system control.

- The protocol file (filename.proto): Every SCPI command that is going to be used has a protocol function associated. In turn, these protocol functions are associated with records of *Stream* type. These records are used in *StreamDevice* IOCs, so whenever that record is processed the protocol function and the SCPI command will sent. So, a protocol file is a collection of protocol functions that have an SCPI command associated.

- st.cmd: This file is like the executable file of the IOC. To start an IOC, the st.cmd file is called. This file contains information about the path toward the database, the protocol file and the IP address of the devices to connect.

Next, to clarify these concepts, an example will be presented. In Figure 4.63, the relationship between the database and the protocol file is clearly shown. For instance, there is a record called "SetINST:Agilent" which is used to change the instrument mode of a device, every time it processes. This is done via an SCPI command. So, it is necessary to use a record with the field *DTYPE* in *stream*, and in the *INP* (input) field sets the protocol file name where the protocol function is, the name of the proper protocol function, and the IP address where it has to send it. In this example, as the *SCAN* field is *1 second*, the record will be processed each second and the IOC will send the SCPI every second.



```
record (ai, "SetINST:Agilent")
{
    field (DESC, "Set the instrument mode")
    field (DTYP, "stream")
    field (INP, "@Anritsu.proto set_INST Port_Anritsu")
    field (SCAN, "1 second")
}
```

```
set_INST {
    out "INST SPECT"    # Set the instrument mode.
}
```

Figure 4.63 – *Record and protocol example*

4.4.6 Graphical User Interface - CSS

EPICS is able to work with various tools for Client WorkStation (CWS), to have an easier look and understanding of what's happening in the system they are trying to control. A graphical user interface available for EPICS is Control System Studio (CSS). This tool uses widgets that are associated to different Process Variable (PV) from the IOCs running in the same LAN. CSS uses CA to get or change the values from the different.

For this project, the following CSS window has been designed. It is divided in three parts. The first one is for monitoring of the BLAS Control Signals, Figure 4.64. Every signal has a LED associated, except for the *Flow Pulses* signal which indicates the water flow in l/min, and the *PD_MI* signal which has not been able to recover.

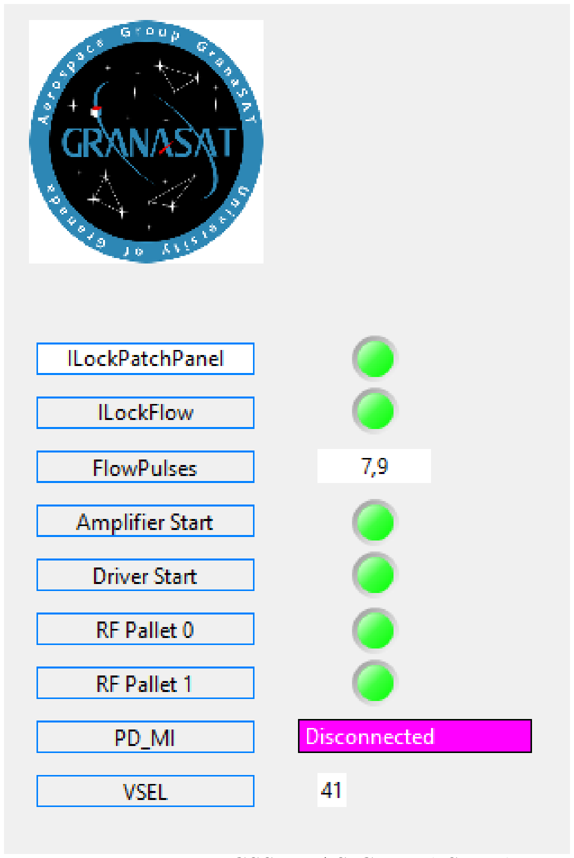


Figure 4.64 – *CSS BLAS Control Signals*

The second part is for the MS2830A signal analyzer, Figure 4.65. In this case, functions for the generator and the spectrum analyzer have been developed, as the active change of the spectrum analysed or the power and frequency of the signal generated.

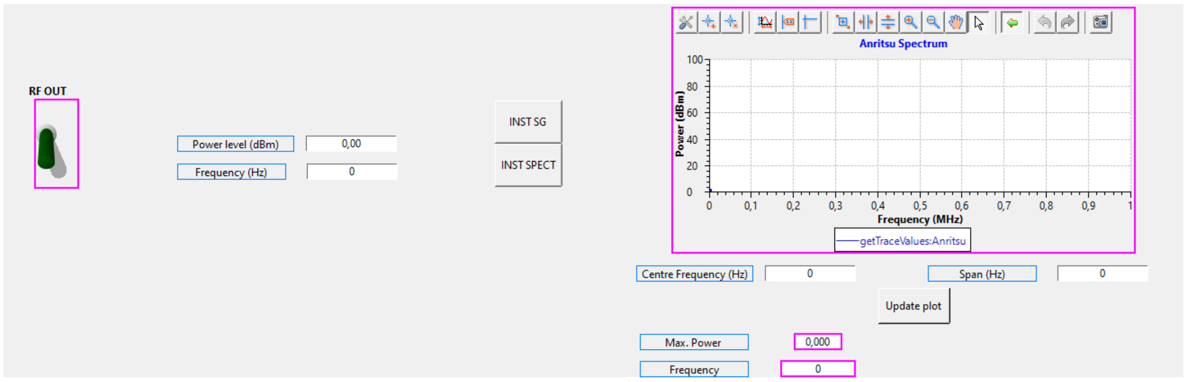
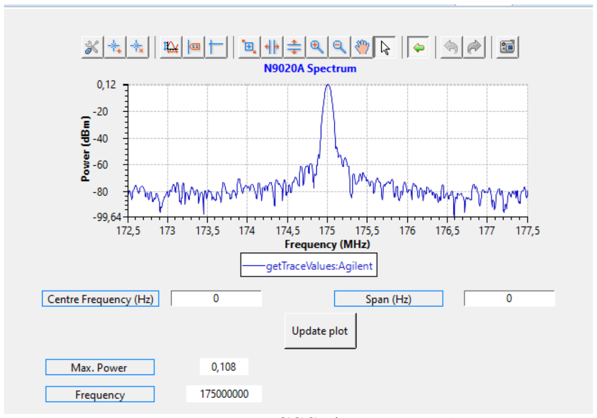


Figure 4.65 – *CSS Anritsu analyzer*

Finally, a part dedicted to the N9020A signal analyzer, Figure 4.66.



Centre Frequency (Hz) 0 Span (Hz) 0

Max. Power 0,108

Frequency 175000000

Figure 4.66 – *CSS Agilent analyzer*

Bibliography

[1] ABOUT CIRCUITS, A. https://www.allaboutcircuits.com/tools/skin-depth-calculator/#:~:text=%22Skin%20depth%22%20refers%20to%20the,permeability%20of%20the%20conductive%20material.

[2] BTESA. *Manual de operación y puesta en marcha. Plan básico de mantenimiento.*

[3] ANSYS. *An introduction to HFSS.*

[4] EPICS. *EPICS Overview.*

[5] F. SIERRA. *RF systems testing solutions for scientific facilities.* CWRF 2016, 2016.

[6] AGILENT TECHNOLOGIES. *E5071C Manual.*

[7] AGILENT TECHNOLOGIES. *N9020A Manual.*

[8] ANRITSU. *MS280A Manual.*

[9] HEWLETT PACKARD (HP). *5087A Manual.*

[10] INFINEON TECHNOLOGIES. *ITS4200S-ME-P*, September 2012. Available at https://datasheet.lcsc.com/lcsc/2202031830_Infineon-Technologies-ITS4200SMEPHUMA1_C2150065.pdf Accessed 6-January-2023.

[11] KEYSIGHT TECHNOLOGIES. *MSOX4104A Manual.*

[12] MARCONI INSTRUMENTS. *2022D Manual.*

[13] TEXAS INSTRUMENTS. *ADS111x Ultra-Small, Low-Power, I2C-Compatible, 860-SPS, 16-Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator*, January 2018. Available at https://www.ti.com/lit/ds/symlink/ads1115.pdf?ts=1668897832069 Accessed 15-November-2022.

[14] TEXAS INSTRUMENTS. *LM2576xx Series SIMPLE SWITCHER® 3-A Step-Down Voltage Regulator*, May 2021. Available at https://www.ti.com/lit/ds/symlink/lm2576.pdf?ts=1673176524840&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM2576%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dapp-null-null-GPN_EN-cpc-pf-google-wwe%2526utm_content%253DLM2576%2526ds_k%253Dlm2576%2526DCM%253Dyes%2526gclid%253DCjwKCAiA8OmdBhAgEiwAShr40yxn_lUOOLObhVJDYb3Wg4DalPwqpISEVE2cGdIOzXB_xLG49x98DxoCV4cQAvD_BwE%2526gclsrc%253Daw.ds Accessed 6-January-2023.

[15] JLCPCB. *PCB manufacturer*, 2006. Available at https://jlcpcb.com/ Accessed 15-November-2022.

[16] M. WEBER. *"Módulo amplificador de radiofrecuencia, de disponibilidad mejorada para los aceleradores de deuterones del dispositivo experimental Ifmif"*. Universidad de Oviedo, 2015.

[17] MINI CIRCUIT. *ZSC-2-4+ Manual.*

[18] notes, E. https://www.electronics-notes.com/articles/basic_concepts/resistance/electrical-resistivity-table-materials.php.

[19] Pirostar. *FC2001 Manual*.

[20] Raspberry Pi Foundation. *Raspberry Pi 4 Model B*, June 2019. Available at https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf Accessed 6-January-2023.

[21] Rohde & Schwarz. *Rubidium Frequency Standard XSRM Manual*, February 1996.

[22] Thomas P. Wangler. *"RF Linear Accelerators."*. Wiley-VCH, 2008.

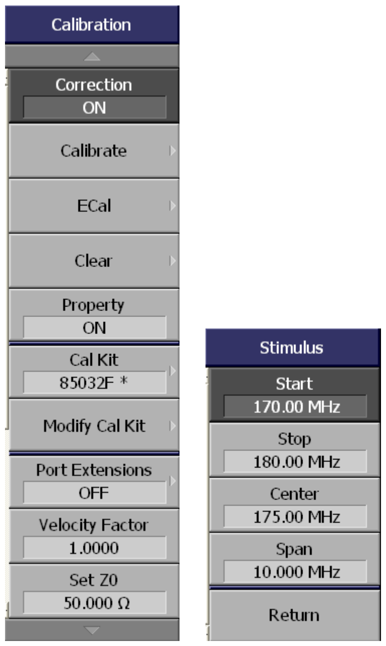
[23] University, I. S. https://www.nde-ed.org/Physics/Materials/Physical_Chemical/Permeability.xhtml#:~:text=For%20non%2Dferrous%20metals%20such,the%20relative%20permeability%20is%20one.

Appendix A

E5071C VNA Calibration

For the calibration of the E5071C VNA, the 85032F kit from Agilent will be used, which contains connectors simulating short circuit, open circuit, and 50 Ω load in both female and male versions. All connectors are N-type. Before performing the typical calibration steps of a network analyzer (placing a short, open. . .), it is necessary to ensure that two parameters are properly adjusted:

- System Z_0: It is necessary to specify the value of the characteristic impedance being used (bottom part of Figure A.1).
- Stimulus: The signal used to characterize our Device Under Test (DUT) and for which the ports will be calibrated. It is necessary to indicate the frequencies being analyzed and the power level of the stimulus, if the power level is critical for the operation of the DUT. Figure A.1.



Correction: ON
Property: ON
Cal Kit: 85032F *
Port Extensions: OFF
Velocity Factor: 1.0000
Set Z0: 50.000 Ω

Start: 170.00 MHz
Stop: 180.00 MHz
Center: 175.00 MHz
Span: 10.000 MHz

(a) *System Z_0 option* **(b)** *VNA stimulus*

Figure A.1 – *System Z_0 and stimulus option*

Next, it is necessary to select the calibration kit to be used, as shown in Figure A.1. As mentioned earlier, the kit is the 85032F from Agilent. Figure A.2 displays the female-type connectors of the kit. When calibrating, a thru is required, but since it is not available in our kit, a stainless steel barrel connector has been used as a simulation.

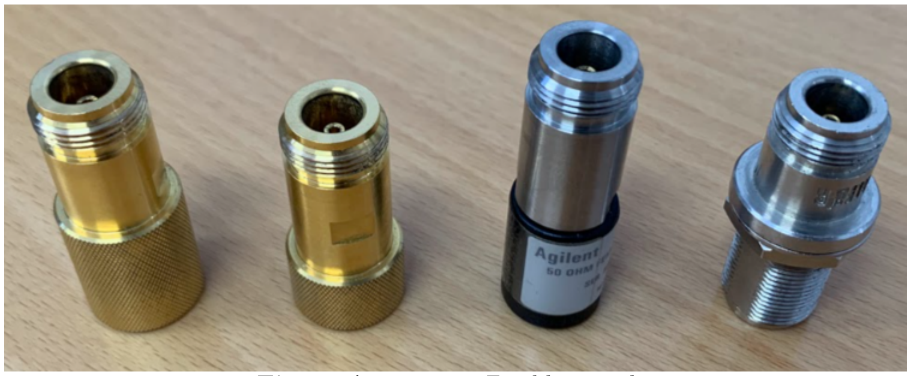


Figure A.2 – *85032F calibration kit*

There are different types of calibrations depending on the intended use of the analyzer and the type of DUT. In this case, the DUT is a three-port cavity, so the calibration will be performed using the *3-Port Cal* method, as shown in Figure A.3.

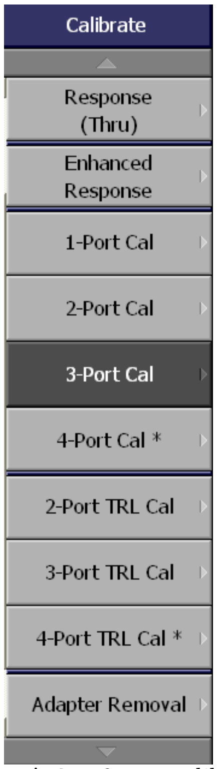


Figure A.3 – *3 port calibration*

The reflection calibration, as shown in Figure A.4, involves placing the open, short, and load on each individual port.

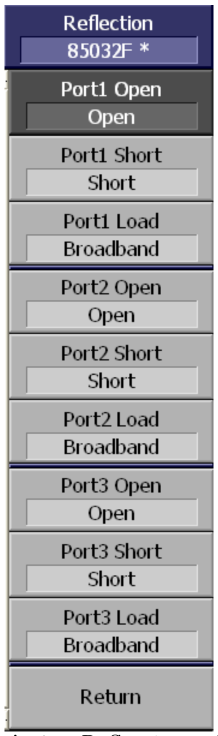


Figure A.4 – *Reflection calibration*

On the other hand, the transmission calibration involves analyzing the behavior of two ports connected together using a thru, Figure A.5.

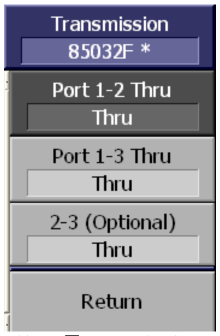
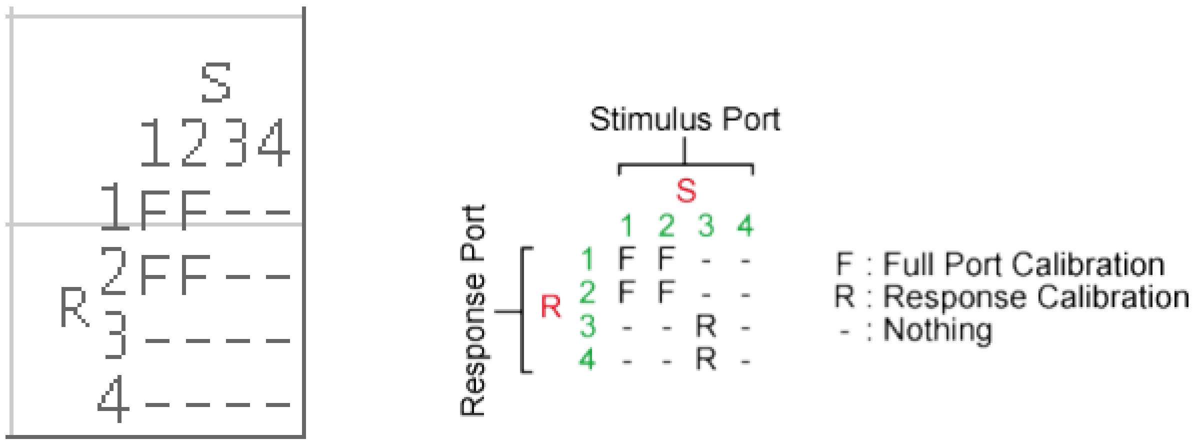


Figure A.5 – *Transmission calibration*

Once these steps have been completed, it is necessary to click on "Done" to apply our calibration. To ensure that it has been configured correctly, the table shown in Figure A.6 should appear in the bottom right part of the screen.



R \ S	1	2	3	4
1	F	F	-	-
2	F	F	-	-
3	-	-	-	-
4	-	-	-	-

Stimulus Port
Response Port

R \ S	1	2	3	4
1	F	F	-	-
2	F	F	-	-
3	-	-	R	-
4	-	-	R	-

F	Full Port Calibration
R	Response Calibration
-	Nothing

(a) *State table* **(b)** *VNA stimulus*

Figure A.6 – *State calibration table [6]*

Appendix B

7/16" male pigtail assembly

The cable used to connect the cavity with the is a 7/8" coaxial cable of 50 Ω, with two 7/16" RF male connectors. Firstly, 2.5 cm of external dielectric is cut off with a cutter, Figure B.1.

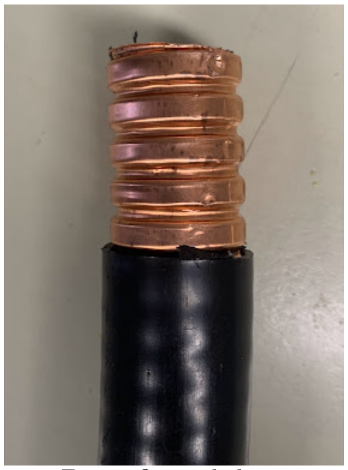


Figure B.1 – *Outer dielectric cut off*

The 7/16" connector has two pieces, the body and the proper male connector. The body is assembled to the cable, Figure B.2.

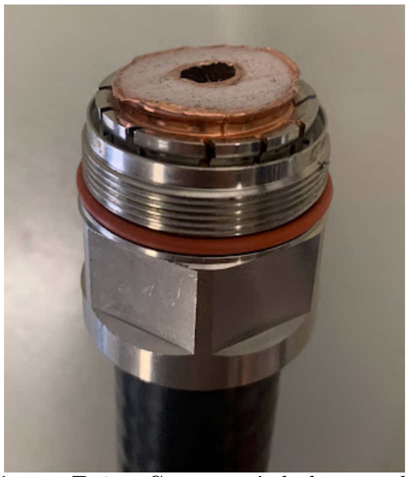


Figure B.2 – *Connector's body assembly*

Finally, using two wrenches, the second piece of the connector is assembled to the body, Figure B.3.

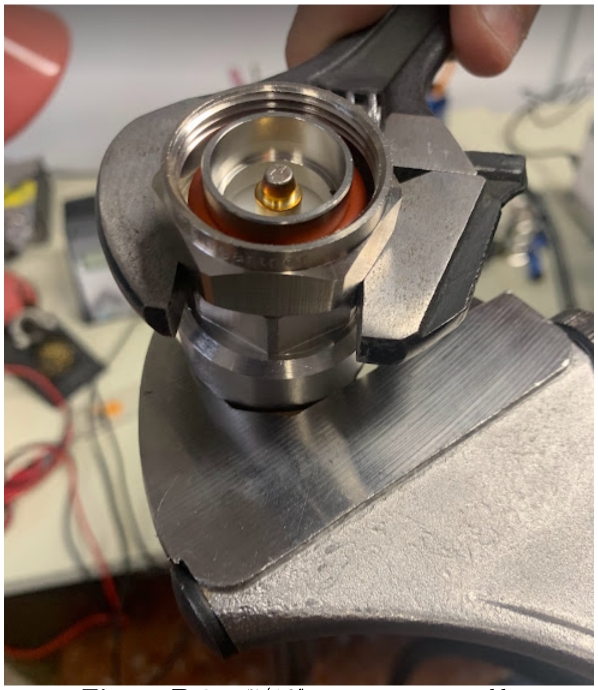


Figure B.3 – *7/16" connector assembly*

In order to check if the cable is correctly assembled, the S parameters were calculated, Figure B.4. As expected, the transmission S parameters are near zero, so there is not a high attenuation in the cable, and the reflected S parameters are very low indicating a good adaptation.

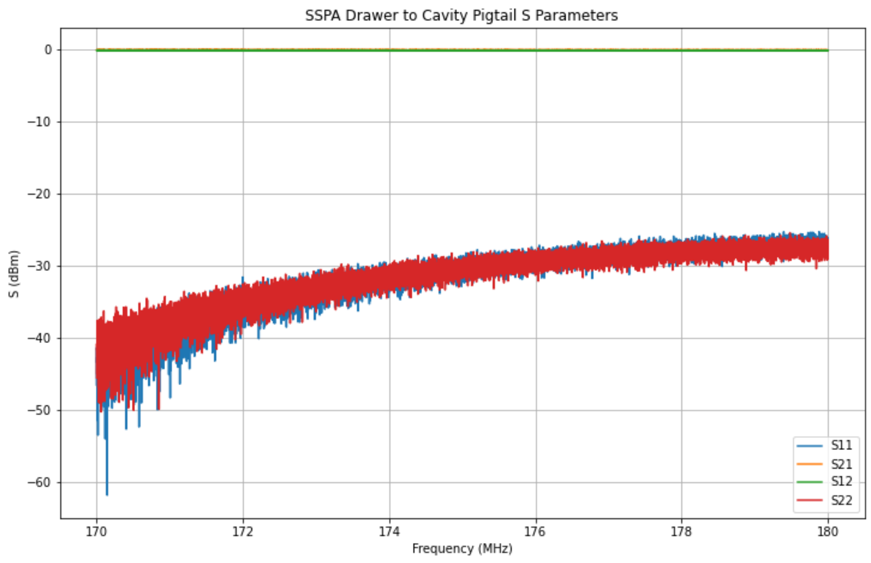


Figure B.4 – *7/16" pigtail's S parameters*

Appendix C

Frequency synchronization

C.1 Introduction

Due to the cavity's high , which induces a need to design a system extremely precise in frequency, it is crucial that all the equipment that conforms the measurement and signal generator system are synchronized in frequency, for the different measures obtained, and the signal generated to be as reliable as possible. RF devices work with a 10 MHz signal generated by their local oscillator. All of these devices have the option to introduce an external 10 MHz reference signal via BNC ports, Figure C.1. By using the same reference signal for all the equipment, there will be less frequency drifts between measures and the signals used. The assembly of the BNC pigtail is explained in Appendix D

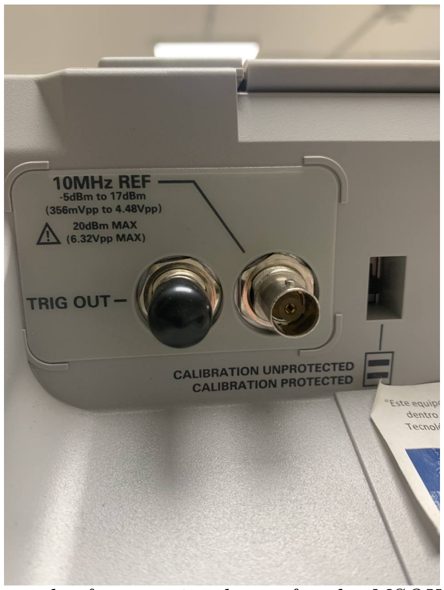


Figure C.1 – *External reference signal port for the MSOX4104A oscilloscope*

C.2 10 MHz reference signal generation

Therefore, to synchronize the system we need to introduce the same 10 MHz signal to all the devices. First, we need to generate a stable reference signal, as all the measurement and signal generator system will depend on the mentioned signal. For this, the Rubidium Frequency Standard XSRM 238.4011.02 from Rohde

& Schwarz has been chosen, Figure C.2, which consists of different modules. The XSRM module supplies a 5 MHz sinusoidal whose frequency is very accurate, stable and of high spectral purity. The XSRM uses the atomic resonance frequency, 6.834682641 GHz, of rubidium 87, which is extremely precise and scarcely influenced by ambient conditions, to regulate the 5 MHz oscillator [21]. Moreover, the XSRM-Z module, converts this 5 MHz signal into a 10 MHz, with all the important characteristics of the first one. This module can also convert 5 MHz signals into 100 kHz and 1 MHz ones, if needed. The final 10 MHz signal is of 1 V_{rms} on 50 Ω. Finally, it is important to note that the XSRM 238.4011.02 needs a warm up time of 35 minutes, to ensure a good reference signal quality.

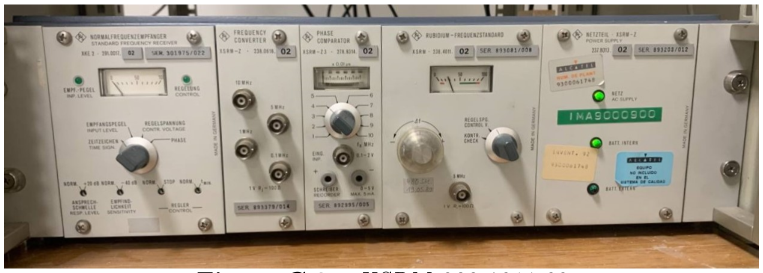


Figure C.2 – *XSRM 238.4011.02*

In order to probe the quality of our 10 MHz signal, the frequency meter FC2001 from Pirostar has been used. This frequency meter can measure signals from 10 Hz to 3 GHz, input impedance of 50 Ω for signals between 1 MHz and 3 GHz, and 1 MΩ between 10 Hz and 50 MHz with a timebase of less than 1 ppm at room temperature [19]. In Figure C.3, the measure given by the frequency meter can be seen, which reaffirms the accuracy of our reference signal, measuring a signal of 10.0000037 MHz.

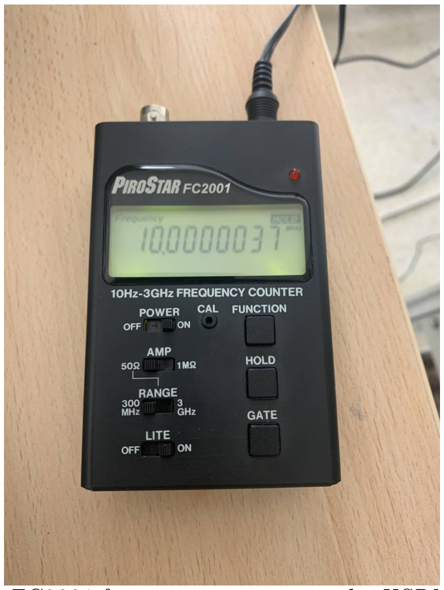


Figure C.3 – *FC2001 frequency measure on the XSRM output signal*

C.3 10 MHz reference signal distribution

After obtaining the stable reference signal, the next step consists on introducing it into each device. The pieces of equipment used are listed below:

- E5071C VNA (Agilent Technologies)
- MXA N9020A 20 Hz to 8.4 GHz signal analyzer (Agilent Technologies)

- MS2830A 9 kHz to 6 GHz signal analyzer and generator (Anritsu)
- MSOX4104A 1 GHz 5 GSa/s oscilloscope (Keysight Technologies)
- 2022D 10 kHz to 1 GHz signal generator (Marconi Instruments)

However, not all the devices have the same requirements for the external reference signal. In table C.1 these specifications are shown.

10 MHz signal power specifications				
Device	P_{max} **(dBm)**	$V_{rms-max}$ **(V)**	P_{min} **(dBm)**	$V_{rms-min}$ **(V)**
E5071C VNA [6]	10	0.71	-3	0.16
MXA N9020A SA [7]	10	0.71	-5	0.13
MS2830A SAG [8]	20	2.2	-15	0.04
MSOX4104A Osc. [11]	17	1.57	-5	0.13
2022D SG [12]	-	-	13	1

Table C.1 – *10 MHz power specifications for the measurement and signal generator system, based on 50 Ω*

Concerning the different power levels required by each device, the 5087A distribution amplifier from HP is used. The 5087A is a narrow-band amplifier capable of receiving up to four different input frequencies (100 kHz, 1 MHz, 5 MHz and 10 MHz) on three input channels (A, B, C). It amplifies these frequencies and supplies outputs in any selected combination on a total of 12 output lines (1-12) [9]. The input level has to be of 0.3 to 3.0 V_{rms} and has an input impedance of 50 Ω. Moreover, the output level into a 50 Ω load can be configured between 0 and 3 V_{rms}. The three input channels and the 12 output ones have a preamplifier associated, with variable gain, changing depending on the voltages needed. In the 5087A front panel, there is a rotary and a voltmeter to check the output voltage (V_{rms}) of the 15 preamplifiers, shown in Figure C.4.

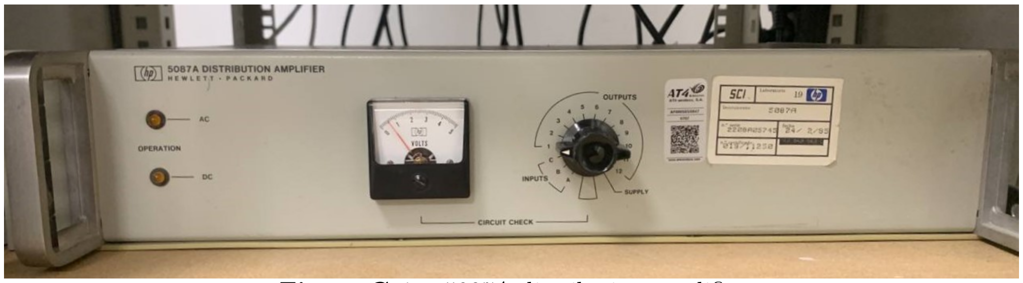


Figure C.4 – *5087A distribution amplifier*

As mentioned before, the distribution amplifier can work with several configurations, option 33 is the one to be installed. This configuration consists on using only one input channel, the 10 MHz signal coming from the XSRM, and having 12 output lines with 10 MHz signals with different levels, as seen in Figure C.5.

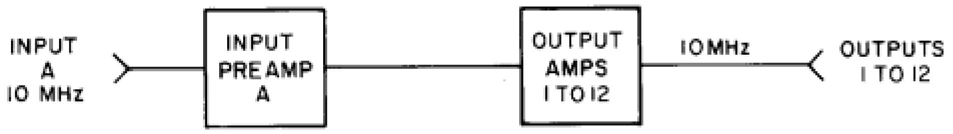


Figure C.5 – *5087A option 33 diagram [9]*

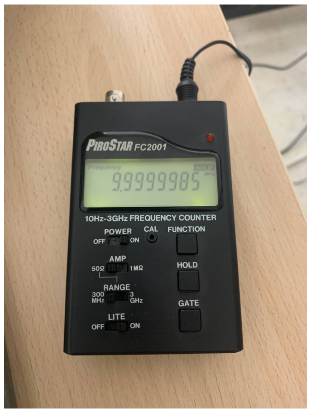
The set-up procedure is as follows:

1. Remove the instrument top cover and connect the 10 MHz signal to the A input channel. Modify the input preamplifier gain until the output voltage reading is 0.3 V.

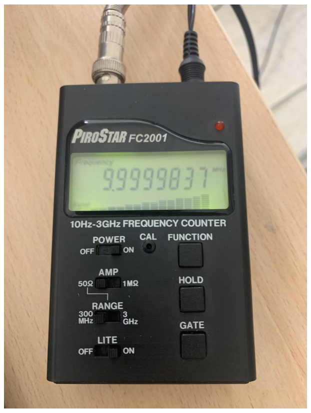
2. Modify the output voltage preamplifiers gain to 0 before connecting the output channels to the different devices, not to damage their external reference signal port.

3. Connect the devices to the output ports of the distribution amplifier and then modify the gain, checking with the voltmeter, to get the voltage needed for each device. It is important to change the gain after connecting the devices, as the final output voltage is the one given when there is a 50 Ω load in the preamplifiers output. When connecting the device's 50 Ω input impedance, the output voltage will decrease.

Finally, the output channels' signals are checked to ensure the signal's quality. As it was done with the 10 MHz XSRM signal, the FC2001 frequency meter was used to measure the signal's frequency, results for channel 2 and 3 shown in Figure C.6.



(a) *Output channel 2*



(b) *Output channel 3*

Figure C.6 – *FC2001 frequency measures on output channels*

Between output channels, there is less than 10 ppm of difference, an assumable error considering all the devices will be synchronized. In Figure C.7, a block diagram of the frequency synchronization system is depicted.

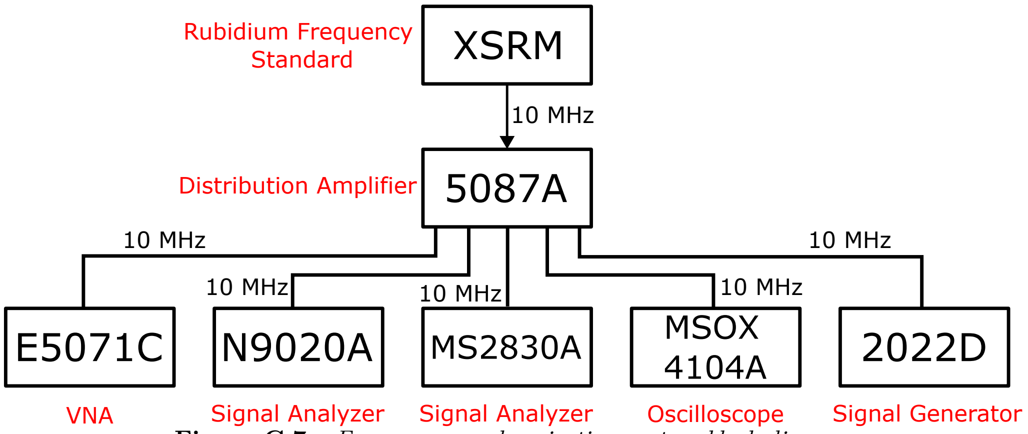


Figure C.7 – *Frequency synchronization system block diagram*

C.4 Devices configurations

It is important to be sure the devices are using the external reference signal, as some pieces of equipment do not swipe between the internal and the external automatically.

- **E5071C VNA:** Changes automatically to the external reference signal. Check if the below *ExtRef* box is highlighted, if not it is using the internal reference because the external one is not inside the specifications, Figure C.8.

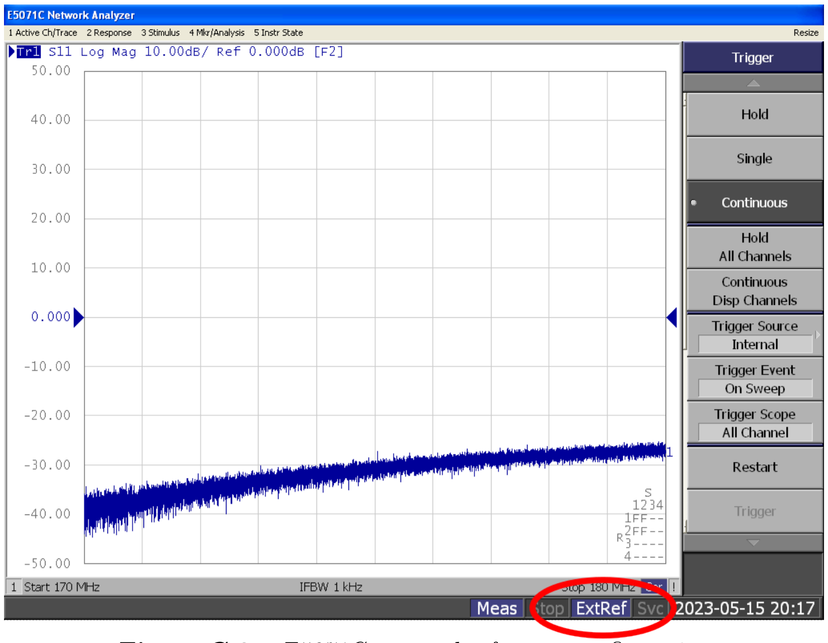


Figure C.8 – *E5071C external reference configuration*

- **MXA N9020A signal analyser:** Changes automatically to the external reference signal. Check if the top *SENSE* variable is fixed to *SENSE:EXT*, if not it is using the internal reference because the external one is not inside the specifications, Figure C.9.

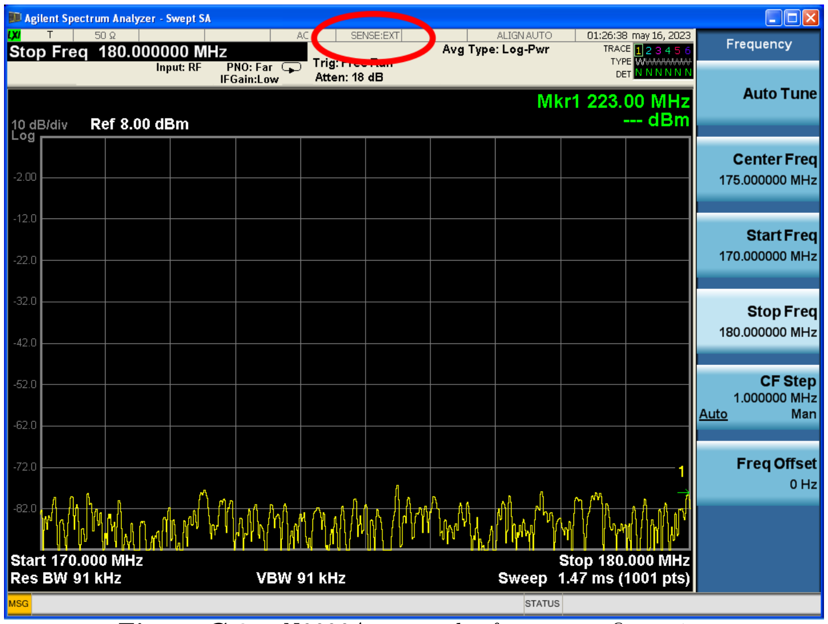


Figure C.9 – *N9020A external reference configuration*

- **MSOX4104A oscilloscope:** Does not change automatically. The option to set the external reference signal on is in *Utilities* ⟶ *Utility menu* ⟶ *Options* ⟶ *Rear panel* ⟶ *10 MHz Ref Signal Output and Input off* ⟶ *Input On*, Figure C.10.

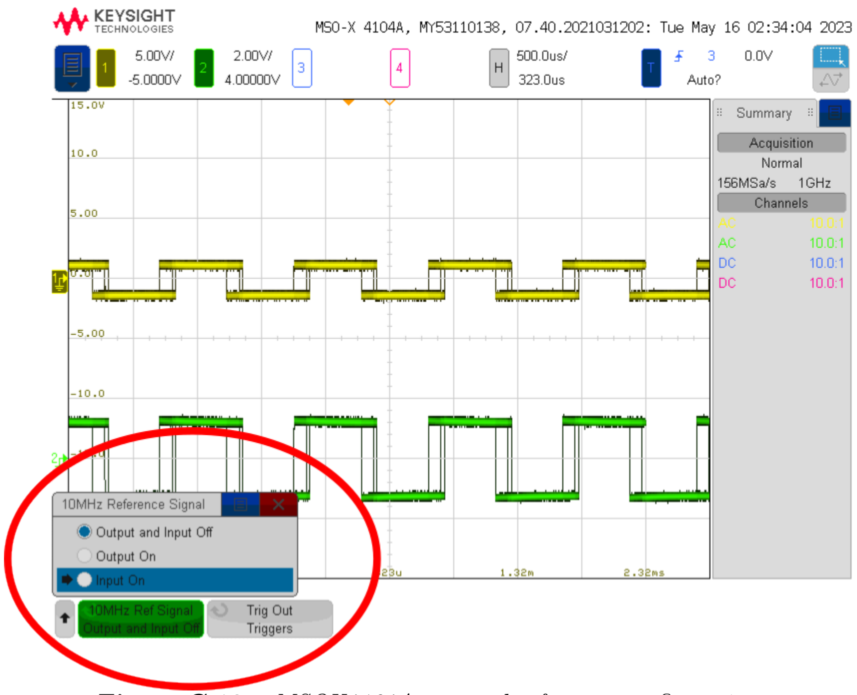


Figure C.10 – *MSOX4104A external reference configuration*

- **2022D signal generator:** Does not change automatically. The option to set the external reference signal on is pressing the button *INT EXT* from the front panel. A *EXT STD* text will appear in the bottom right corner of the display, Figure C.11 However, there can occur two type of errors, shown in the display as:

  1. Error 11: The external reference is selected but not applied.
  2. Error 12: The external reference signal is not locking.

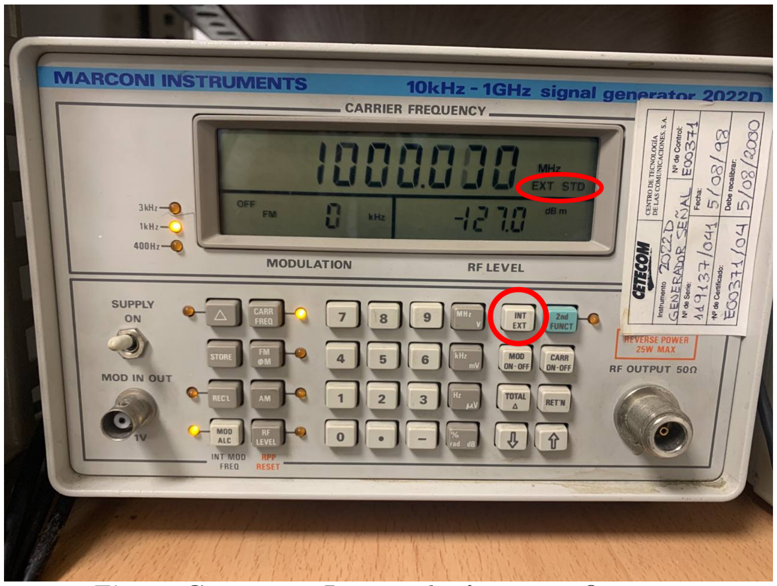


Figure C.11 – *2022D external reference configuration*

Appendix D

BNC pigtail assembly

In this appendix, the assembly of a male BNC pigtail will be detailed. Firstly, it is important to choose the correct cable. As the pigtail is going to be used for RF purposes, with 50 Ω loads, the cable's characteristic impedance has to be of 50 Ω. In this case, the URM43 coaxial cable has been selected, widely used in the telecommunications industry. Now the material needed for a male BNC pigtail will be listed:

- 2 50 Ω male BNC connectors
- URM43 coaxial cable (length as desired)
- Cutting pliers
- Screwdriver
- Crimping tool

In Figure D.1, the 2 BNC connectors that are going to be used are shown, the one on the left is disassembled. It is divided in two parts. The first one has a central whole where the coaxial cable's inner conductor is introduced and tighten with a screw. Moreover, on the opposite part of the BNC head, there is a metallic support for the coaxial cable and for the outer conductor, to fix the ground level on the connector. The second part of the connector acts as a shield.

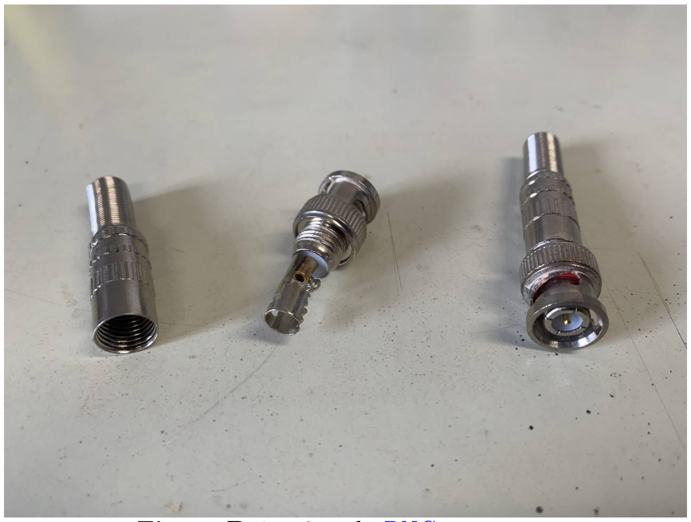


Figure D.1 – *2 male BNC connectors*

The first step to assembly the desired pigtail is to strip the coaxial. To start, the outer dielectric will be stripped for 13 mm, leaving the start of the outer dielectric somewhere near the middle of the metallic part for ground level, as seen in Figure D.2.

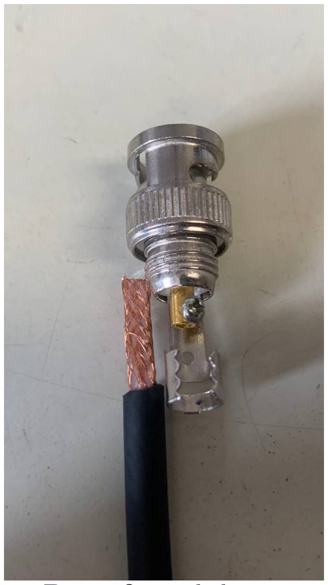


Figure D.2 – *Outer dielectric stripped*

Once the outer dielectric is stripped, the outer conductor wires are compacted towards the outer conductor, as seen in Figure D.3.

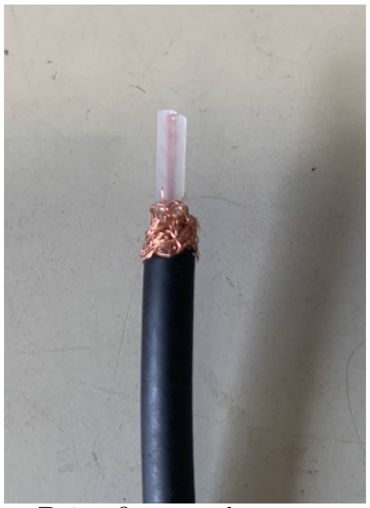


Figure D.3 – *Outer conductor put together*

Next, the inner conductor dielectric will be stripped, leaving 7 mm of dielectric, and the inner conductor without shield. For this particular connector only 3 mm of inner conductor is needed, so the rest is cut. Before, connecting the connector and the cable it is important to get the connector's shield part through the cable, as when the coaxial is tightened to the connector it is not going to be possible to do so, Figure D.4.

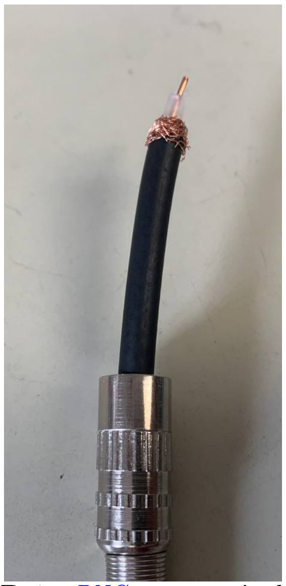


Figure D.4 – *BNC connector's shield part*

Following these instructions, the BNC connector and the coaxial cable should look as Figure D.5. The inner dielectric up to the start of the inner conductor whole, and the outer conductor sharing space with the outer dielectric in the metallic part.

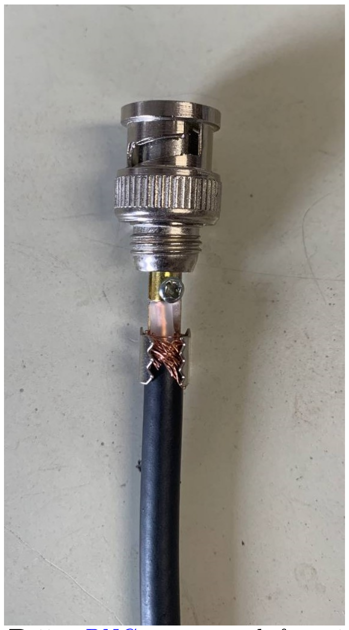


Figure D.5 – *BNC connector before tightening*

The next step is to tight the inner conductor to the connector using the screw available and crimp by the metallic part, Figure D.6. It is recommended to screw first as when crimping the coaxial or the connector could move and change the setup of Figure D.5.

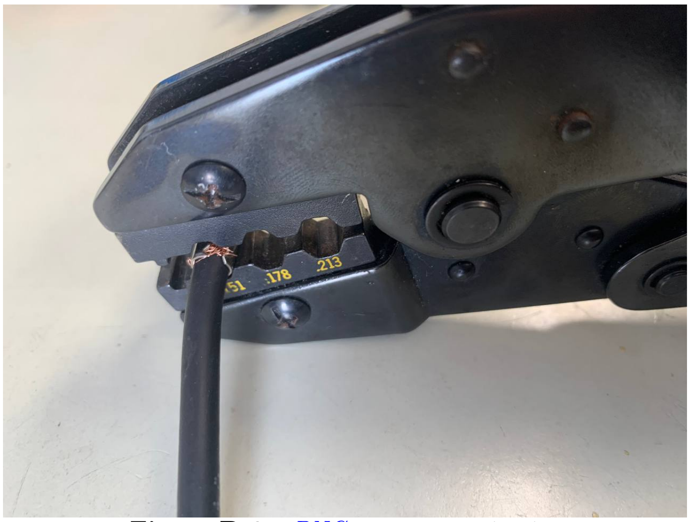


Figure D.6 – *BNC connector crimping*

Finally, the shield part is threaded, and the BNC connector is ready to use, Figure D.7. After following the same steps for the other connector, the male BNC pigtail is ready to use. A multimeter can be used to check there are no shorts in the cable between ground and signal.

Figure D.7 – *Finished male BNC pigtail*

Appendix E

Electronics Instrumentation Notebooks

E.1 N9020A Signal Analyzer

[4]:
```
import pyvisa as visa
import numpy as np
from struct import unpack
import pylab
import time
import math
from matplotlib import pyplot as plot

# Establish Connection
rm = visa.ResourceManager('@py') # Calling PyVisaPy library
#scope = rm.open_resource('USB0::0x0699::0x0409::C010730::INSTR') #␣
 ↪Connecting via USB
# 192.168.1.200 => IP de la máquina Agilent en el sistema
#scope = rm.open_resource('TCPIP::172.25.2.52::5025::SOCKET') #␣
 ↪Connecting via LAN
scope = rm.open_resource('TCPIP::192.168.1.13::INSTR') # Connecting via␣
 ↪LAN
print(scope)
```

```
TCPIPInstrument at TCPIP0::192.168.1.13::inst0::INSTR
```

Pyvisa es la librería que voy a usar para comunicarme con la máquina, puesto que es una librería sencilla de usar en Python que permite una comunicación ethernet (que es lo que me interesa) con dispositivos mediante su IP como vemos antes.

Se instala con:

- pip install PyVISA
- pip install PyVISA-py

Pyvisa consta de dos principales funciones una vez hemos establecido conexión:

- write() : es una función con la cual podemos enviar los comandos SCPI (los comandos o lenguaje que entiende la máquina) a la máquina, esta función se usa para cambiar parámetros en la máquina, es decir, cuando usas esta función es para modificar algo, no para esperar una respuesta por parte de la máquina, pues esta función no recoge dato alguno de la máquina salvo el estado de éxito del envío del comando.

- query() : esta función es la que usaremos para solicitar datos o información a la máquina, mediante esta función podemos solicitar datos a la máquina y lo más importante, la máquina nos devuelve estos datos, por lo que cuando usemos esta función será para recoger la información que nos dé la máquina en una variable o imprimirla directamente, puesto que todo lo que nos devuelve la máquina son strings.

[10]:
```
!pip install PyVISA
# Versión: 1.11.3
#!pip show PyVISA
```

```
Collecting PyVISA
  Downloading PyVISA-1.13.0-py3-none-any.whl (175 kB)
     -------------------------------------- 175.7/175.7 kB 2.7 MB/s eta 0:
 ↪00:00
Requirement already satisfied: typing-extensions in
d:\programas\winpython\wpy64-31050\python-3.10.5.amd64\lib\site-packages␣
 ↪(from
PyVISA) (4.3.0)
Installing collected packages: PyVISA
Successfully installed PyVISA-1.13.0
```

[7]:
```
#pip install --upgrade pip
```

```
Requirement already satisfied: pip in
d:\programas\winpython\wpy64-31050\python-3.10.5.amd64\lib\site-packages
(22.1.2)
Collecting pip
  Downloading pip-22.3.1-py3-none-any.whl (2.1 MB)
     ---------------------------------------- 2.1/2.1 MB 4.5 MB/s eta 0:00:
 ↪00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.1.2
    Uninstalling pip-22.1.2:
      Successfully uninstalled pip-22.1.2
Successfully installed pip-22.3.1
Note: you may need to restart the kernel to use updated packages.
```

[11]:
```
!pip install PyVISA-py
# Versión: 0.5.2
!pip show PyVISA-py
```

```
Collecting PyVISA-py
  Downloading PyVISA_py-0.6.0-py3-none-any.whl (69 kB)
     ---------------------------------------- 69.0/69.0 kB 1.9 MB/s eta 0:
↪00:00
Requirement already satisfied: typing-extensions in
d:\programas\winpython\wpy64-31050\python-3.10.5.amd64\lib\site-packages␣
↪(from
PyVISA-py) (4.3.0)
Requirement already satisfied: pyvisa>=1.12.0 in
d:\programas\winpython\wpy64-31050\python-3.10.5.amd64\lib\site-packages␣
↪(from
PyVISA-py) (1.13.0)
Installing collected packages: PyVISA-py
Successfully installed PyVISA-py-0.6.0
Name: PyVISA-py
Version: 0.6.0
Summary: Pure Python implementation of a VISA library.
Home-page:
Author:
Author-email: "Hernan E. Grecco" <hernan.grecco@gmail.com>
License: The MIT License

        Copyright (c) 2014 PyVISA-py Authors and contributors. See AUTHORS

        Permission is hereby granted, free of charge, to any person␣
↪obtaining a
copy of
        this software and associated documentation files (the "Software"),␣
↪to
deal in
        the Software without restriction, including without limitation the
rights to
        use, copy, modify, merge, publish, distribute, sublicense, and/or␣
↪sell
copies
        of the Software, and to permit persons to whom the Software is␣
↪furnished
to do
        so, subject to the following conditions:
```

```
        The above copyright notice and this permission notice shall be␣
 ↪included
in all
        copies or substantial portions of the Software.

        THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,␣
 ↪EXPRESS
OR
        IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF␣
 ↪MERCHANTABILITY,
        FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT␣
 ↪SHALL
THE
        AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR␣
 ↪OTHER
        LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,␣
 ↪ARISING
FROM,
        OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER␣
 ↪DEALINGS
IN THE
        SOFTWARE.

Location: d:\programas\winpython\wpy64-31050\python-3.10.5.amd64\lib\site-
packages
Requires: pyvisa, typing-extensions
Required-by:
```

[12]:
```
#!pip install matplotlib # Versión: 3.5.0
!pip show matplotlib
```

```
Name: matplotlib
Version: 3.5.2
Summary: Python plotting package
Home-page: https://matplotlib.org
Author: John D. Hunter, Michael Droettboom
Author-email: matplotlib-users@python.org
License: PSF
Location: d:\programas\winpython\wpy64-31050\python-3.10.5.amd64\lib\site-
packages
Requires: cycler, fonttools, kiwisolver, numpy, packaging, pillow,␣
 ↪pyparsing,
python-dateutil
Required-by: astroML, Cartopy, fastai, ipympl, mizani, mlxtend, mpl-scatter-
```

```
density, mpld3, mpldatacursor, nbconvert_reportlab, plotnine, pygad,␣
 ↪seaborn,
wordcloud
```

```
[5]: # PÁGINA 153
     # Ejecución de identificador del dispositivo
     print(scope.query('*IDN?'))
     #print(scope.query(':SYST:PRES'))
```

```
Agilent Technologies,N9020A,MY49100673,A.03.08
```

```
[3]: # PÁGINA 1176
     # Ejecución para modificar la frecuencia central
     scope.write('FREQ:CENT 180MHZ')
```

```
[3]: 18
```

```
[10]: # Ejecución para averiguar la frecuencia central // PAG 244
      print(scope.query('FREQ:CENT?'))
```

```
2.00000000E+08
```

```
[47]: # PÁGINA 461
      # Ejecución para modificar el SPAN
      scope.write('FREQ:SPAN 10MHZ')
```

```
[47]: 17
```

```
[28]: # Ejecución para averiguar el SPAN
      print(scope.query("FREQ:SPAN?"))
```

```
1.000000000E+07
```

```
[37]: # PÁGINA 1182 FREQ:STAR
      # PÁGINA 1184 FREQ:STOP
      #scope.write('FREQ:START 17MHZ')  # Modifica la frecuencia inicial o␣
       ↪mínima
      #scope.write('FREQ:STOP 18MHZ')  # Modifica la frecuencia final o máxima
```

```
[48]: print(scope.query('FREQ:START?')) # Muestra la frecuencia incial o mínima
      print(scope.query('FREQ:STOP?')) # Muestra la frecuencia final o máxima
```

```
1.70000000E+08
```

```
1.80000000E+08
```

```
[3]: # PÁGINA 368
     # Con esto configuramos el instrumento para que en los siguientes comandos
     # cuando le pidamos la coordenada X e Y, nos de las del punto máximo␣
      ↪(frecuencia/potencia)
     scope.write('CALC:MARK:MAX')
```

```
[3]: 15
```

```
[5]: # PÁGINA 404
     print("Frecuencia donde se encuentra la potencia máxima: ",scope.
      ↪query('CALC:MARK:X?'))
     print("Potencia máxima: ",scope.query('CALC:MARK:Y?'))
```

```
Frecuencia donde se encuentra la potencia máxima:  1.74920000000E+08

Potencia máxima:  3.047E+00
```

```
[67]: # Para averiguar la potencia de una determinada frecuencia lo que debemos␣
       ↪hacer
      # es modificar el valor de la coordenada x (frecuencia), de tal forma que␣
       ↪si luego
      # pedimos la coordenada y (la potencia), se nos devolverá la potencia␣
       ↪asociada a la
      # frecuencia definida anteriormente
      scope.write('CALC:MARK:X 178.5MHZ')
```

```
[67]: 22
```

```
[68]: print("Frecuencia definida anteriormente: ",scope.query('CALC:MARK:X?'))
      print("Potencia asociada a la frecuencia definida: ",scope.query('CALC:
       ↪MARK:Y?'))
```

```
Frecuencia definida anteriormente:  1.78500000000E+08

Potencia asociada a la frecuencia definida:  -8.7075E+01
```

```
[5]: #PÁGINA 1137
     # Para cambiar el nivel de referencia
     scope.write('DISP:WIND:TRAC:Y:RLEV 10.00')
```

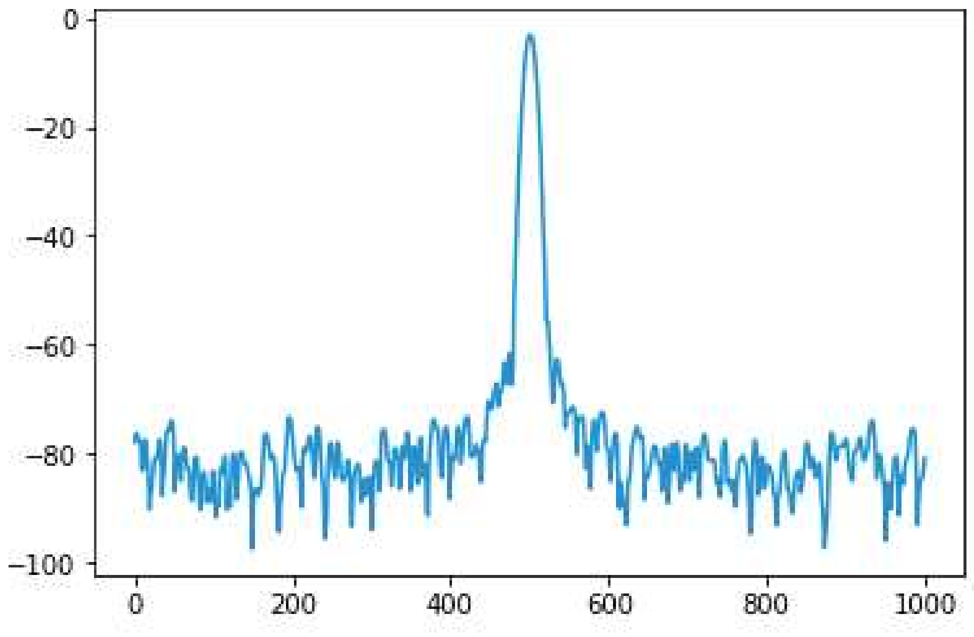
```
[5]: 29
```

[3]:
```
# Para recoger información de los datos del spectrum analyzer
# FREQ:CENT ==> la frencuencia central
# FREQ:START ==> la frencuencia inicial
# FREQ:STOP ==> la frencuencia final
# DISP:WIND:TRAC:Y:RLEV ==> el nivel de referencia
# SWE:POIN? ==> el número de puntos de los que se recoge información

print(scope.query('FREQ:CENT?'))
print(scope.query('FREQ:START?'))
print(scope.query('FREQ:STOP?'))
print(scope.query('DISP:WIND:TRAC:Y:RLEV?'))
print(scope.query('SWE:POIN?'))


freqCentral = float(scope.query('FREQ:CENT?'))
freqInicial = float(scope.query('FREQ:START?'))
freqFinal = float(scope.query('FREQ:STOP?'))
nivelRef = float(scope.query('DISP:WIND:TRAC:Y:RLEV?'))
nPuntos = int(scope.query('SWE:POIN?')) ## PÁGINA 1369
```

```
1.75000000E+08

1.70000000E+08

1.80000000E+08

1.000E+01

1001
```

[66]:
```
# PÁGINA 509
# VAMOS A PLOTEAR LA INFORMACIÓN


puntos = int(scope.query('SWE:POIN?')) # guardamos el número de puntos a␣
 ↪representar con plot
initialFreq = float(scope.query('FREQ:START?')) # guardamos la frecuencia␣
 ↪inicial en ese momento
finalFreq = float(scope.query('FREQ:STOP?')) # guardamos la frecuencia␣
 ↪final en ese momento

scope.write('FORM ASC') # Pide a la máquina que lo que se le pide lo␣
 ↪devuelva en formato ASCII
```

```
datos = scope.query('TRAC? TRACE1') # Pide a la máquina los datos del␣
 ↪Spectrum (los datos son las potencias de los 10001 puntos observables y␣
 ↪medibles)
datosManipulables = datos.split(",") # separa todos los datos separados␣
 ↪por comas y los guarda en una lista
datosManipulables = [float(i) for i in datosManipulables] # transformo␣
 ↪los datos de string a float para poder trabajar con ellos


freq = finalFreq - initialFreq  # definimos la amplitud del intervalo de␣
 ↪frecuencias a representar
pointWidth = freq / float(puntos) # defino la anchura que debe ocupar␣
 ↪cada punto en la imagen (para saber donde colocar cada frecuencia en la␣
 ↪imagen)


# Creo una lista de todas las frecuencias a representar
frequencies = []
count = 0
while len(frequencies) != puntos:
    frequencies.append(initialFreq +(pointWidth*count))
    count+=1

# Label for x-axis
plot.xlabel("Frequency (MHz)")

# Label for y-axis
plot.ylabel("Power (dBm)")

# title of the plot
plot.title("Output of Spectrum Analyzer")

#Genero la imagen con las frecuencias calculadas y las potencias␣
 ↪ofrecidas por la máquina

plot.plot(frequencies,datosManipulables)
plot.savefig('./images/graphAgilent.png') # Dirección relativa donde se␣
 ↪quiere que se guarde la imagen creada
plot.show()
```



E.1.1 Definición de clase para máquina Agilent MXA N9020A 20 Hz-8.4 GHz

```
[69]: # Librerías o Modulos necesarios a importar
      import pyvisa as visa
      import numpy as np
      from struct import unpack
      import pylab
      import time
      from matplotlib import pyplot as plot
```

```
[82]: ##########################################################################################
      ##############                 CLASS AGILENT_N9020A    ␣
       ↪####################################
      ##############         SPECTRUM ANALYZER CONTROLLED    ␣
       ↪####################################
      ##############  BY ETHERNET CONTROL                   ␣
       ↪####################################
      ##########################################################################################

      class Agilent_MXA_N9020A:

          # La medida con la que vamos a trabajar van a ser los MHz
```

```
    medida ='MHZ'

#######################################################################################
############## CONSTRUCTOR␣
 ↪#################################################################
#######################################################################################

    # Cuando creemos un objeto de la clase, ejecutaremos el setup()␣
 ↪conectándose automáticamente a la máquina
    # mediante conexión TCP

    def __init__(self):

        self.scope= self.setup()
        #sel.nPoints = int(self.scope.query('SWE:POIN?'))

#######################################################################################
############## IDENTITY & SETUP␣
 ↪############################################################
#######################################################################################

    # Muestra la información propia ofrecida por la máquina

    def identity(self):
        info= self.scope.query('*IDN?')
        info = info.split(",")
        print("Fabricante: ",info[0])
        print("Modelo: ",info[1])
        print("Número de serie: ",info[2])
        print("Firmware: ",info[3])

    # Establece una conexión TCP con la máquina mediante su IP␣
 ↪devolviendo el objeto conectado para poder manejarlo

    def setup(self):
        # 192.168.1.200 IP AGILENT MACHINE
        rm = visa.ResourceManager('@py') # Calling PyVisaPy library
        scope = rm.open_resource('TCPIP::192.168.1.200::INSTR') #␣
 ↪Connecting via LAN
        return scope


#######################################################################################
```

```
# AL CONTRARIO QUE LA MÁQUINA ANRITSU, AGILENT SOLO TIENE MODO SPECTRUM␣
 ↪ANALYZER, POR LO QUE NO ES NECESARIO ACTIVAR NINGÚN MODO
# EL MODO SPECTRUM ANALYZER VIENE POR DEFECTO
######################################################################################

    # Activa el modo Spectrum en la máquina (por si acaso)

    def setSpectrum(self):
        self.scope.write('INST SA')
        #print("Ha seleccionado el Spectrum Analyzer")


######################################################################################
######### FUNCTIONS FOR SPECTRUM ANALYZER␣
 ↪#################################################
######################################################################################

    # Muestra todos los parámetros del Espectro en ese momento

    def getParamsSpectrum(self):
        print("Frecuencia central: ",self.centralFreq ,"MHz") # Muestra␣
 ↪la frecuencia central
        print("Frecuencia inicial: ",self.inicialFreq,"MHz") # Muestra la␣
 ↪frecuencia inicial
        print("Frecuencia final: ",self.finalFreq,"MHz") # Muestra la␣
 ↪frecuencia final
        print("Nivel de referencia: ",self.referenceLevel,"dBm") #␣
 ↪Muestra el nivel de referencia

    # Modifica todos los parámetros del Spectrum Analyzer (la medida de␣
 ↪las frecuencias está en MHz)

    def setParamsSpectrum(self,inicialFreq , finalFreq , referenceLevel):
        # Guarda todos los datos en variables del objeto de la clase
        self.inicialFreq = float(inicialFreq)
        self.finalFreq = float(finalFreq)
        self.referenceLevel = float(referenceLevel)
        self.centralFreq = (self.finalFreq + self.inicialFreq)/2.0

        self.scope.write('FREQ:START '+ str(inicialFreq) + self.medida) #␣
 ↪Modifica la frecuencia inicial
        self.scope.write('FREQ:STOP '+ str(finalFreq) + self.medida) #␣
 ↪Modifica la frecuencia final
```

```
        self.scope.write('FREQ:CENT '+ str(self.centralFreq)+ self.
↪medida) # Modifica la frecuencia central
        self.scope.write('DISP:WIND:TRAC:Y:RLEV '+ str(referenceLevel)) #␣
↪Modifica el nivel de referencia
        # Se define el número de puntos observables y medibles al valor␣
↪que tenga la máquina en ese momento (por defecto son 10001)
        self.nPoints = int(self.scope.query('SWE:POIN?'))



    # Modifica todos los parámetros del Spectrum Analyzer mediante el uso␣
↪de span (la medida de las frecuencias está en MHz)

    def setParamsSpectrumSpan(self, centralFreq , span , referenceLevel):
            self.setCentralFreqMHz(centralFreq) # Modifica la frecuencia␣
↪central y su atributo de la clase
            self.setSpanMHz(span) # llama a la función modificando el␣
↪valor de span inicialFreq y finalFreq y los atributos de la clase
            self.setReferenceLevelDBM(referenceLevel) # llama a la␣
↪función modificando el valor de referencelevel y el atributo de la␣
↪clase
            # Se define el número de puntos observables y medibles al␣
↪valor que tenga la máquina en ese momento (por defecto son 10001)
            self.nPoints = int(self.scope.query('SWE:POIN?'))



    # Modifica el valor del span, inicialFreq y finalFreq y aparte los␣
↪atributos de la clase correspondientes a la frecuencia inicial y final␣
↪y el span
    # (Hace falta haber definido la frecuencia central)

    def setSpanMHz(self, span):
        self.span = float(span)
        mitad = self.span / 2.0
        self.scope.write('FREQ:SPAN '+ str(span)+ self.medida)

        self.inicialFreq = self.centralFreq - mitad
        self.scope.write('FREQ:START '+ str(self.inicialFreq) + self.
↪medida)
        self.finalFreq = self.centralFreq + mitad
        self.scope.write('FREQ:STOP '+ str(self.finalFreq) + self.medida)
    # Muestra el Span en ese momento

    def getSpanMHz(self):
        print("Span: ", self.span , " MHz")

    # Modifica el valor de la frecuencia central y el atributo del objeto␣
↪de la clase correspondiente

    def setCentralFreqMHz(self,centralFreq):
        self.centralFreq = float(centralFreq)
        self.scope.write('FREQ:CENT '+ str(centralFreq)+ self.medida)

    # Muestra la frecuencia central en ese momento

    def getCentralFreqMHz(self):
        print("Frecuencia central: ",self.centralFreq," MHz")

    # Modifica la frecuencia incial y el atributo del objeto de la clase␣
↪correspondiente

    def setInicialFreqMHz(self,inicialFreq):
        self.inicialFreq = float(inicialFreq)
        self.scope.write('FREQ:START '+ str(inicialFreq) + self.medida)
        self.centralFreq = (self.inicialFreq + self.finalFreq)/2.0

    # Muestra la frecuencia inicial en ese momento

    def getInicialFreqMHz(self):
        print("Frecuencia inicial: ",self.inicialFreq," MHz")

    # Modifica la frecuencia final y el atributo del objeto de la clase␣
↪correspondiente

    def setFinalFreqMHz(self,finalFreq):
        self.finalFreq = float(finalFreq)
        self.scope.write('FREQ:STOP '+ str(finalFreq) + self.medida)
        self.centralFreq = (self.inicialFreq + self.finalFreq)/2.0

    # Muestra la frecuencia final en ese momento

    def getFinalFreqMHz(self):
        print("Frecuencia final: ",self.finalFreq," MHz")
    # Modifica el nivel de referencia y el atributo del objeto de la␣
↪clase correspondiente

    def setReferenceLevelDBM(self,referenceLevel):
        self.referenceLevel = float(referenceLevel)
        self.scope.write('DISP:WIND:TRAC:Y:RLEV '+ str(referenceLevel))

    # Muestra el nivel de referencia en ese momento

    def getReferenceLevelDBM(self):
        print("Nivel de referencia: ",self.referenceLevel," dBm")

    # Muestra y devuelve el número de puntos observables y medibles en␣
↪ese momento

    def getNumPoints(self):
        puntos = int(self.scope.query('SWE:POIN?'))
        #print("Número de puntos: ",puntos)
        return puntos

    # Modifica el número de puntos observables y medibles y el atributo␣
↪del objeto de la clase correspondiente

    def setNumPoints(self,npoints):
        self.nPoints = int(npoints)
        self.scope.write('SWE:POIN '+ str(npoints))

    # Muestra y devuelve la frecuencia donde se encuentra la potencia␣
↪máquina y la potencia máxima

    def getMaxFreqPower(self):
        self.scope.write('CALC:MARK:MAX')
        print("Frecuencia donde se encuentra la potencia máxima: ",self.
↪scope.query('CALC:MARK:X?')," MHz")
        print("Potencia máxima: ",self.scope.query('CALC:MARK:Y?')," dBm")
        freq = float(self.scope.query('CALC:MARK:X?'))
        power = float(self.scope.query('CALC:MARK:Y?'))
        return freq,power

    # Función que devuelve la potencia referente a una frecuencia dada

    def getPowerDBM(self, freq):
        self.scope.write('CALC:MARK:X '+ str(freq)+ self.medida)
        print("Potencia asociada a la frecuencia dada: ",self.scope.
↪query('CALC:MARK:Y?'), " dBm")

########################################################################################
############## PLOT INFORMATION ␣
↪############################################################
########################################################################################

    # Función que guarda una imagen png y la muestra en pantalla de la␣
↪señal del Spectrum completa en ese momento

    def plotInfo(self):

        puntos = self.getNumPoints() # guardamos el número de puntos a␣
↪representar con plot

        self.scope.write('FORM ASC') # Pide a la máquina que lo que se le␣
↪pide lo devuelva en formato ASCII
        datos = self.scope.query('TRAC? TRACE1') # Pide a la máquina los␣
↪datos del Spectrum (los datos son las potencias de los 10001 puntos␣
↪observables y medibles)
        datosManipulables = datos.split(",") # separa todos los datos␣
↪separados por comas y los guarda en una lista
        datosManipulables = [float(i) for i in datosManipulables] #␣
↪transformo los datos de string a float para poder trabajar con ellos


        freq = self.finalFreq - self.inicialFreq # definimos la amplitud␣
↪del intervalo de frecuencias a representar
        pointWidth = freq / float(puntos) # defino la anchura que debe␣
↪ocupar cada punto en la imagen (para saber donde colocar cada␣
↪frecuencia en la imagen)


        # Creo una lista de todas las frecuencias a representar
        frequencies = []
        count = 0
        while len(frequencies) != puntos:
            frequencies.append(self.inicialFreq+(pointWidth*count))
            count+=1

        # Label for x-axis
        plot.xlabel("Frequency (MHz)")
        # Label for y-axis
        plot.ylabel("Power (dBm)")

        # title of the plot
        plot.title("Output of Spectrum Analyzer")

        #Genero la imagen con las frecuencias calculadas y las potencias␣
 ↪ofrecidas por la máquina

        plot.plot(frequencies,datosManipulables)
        plot.savefig('./images/graphAgilent.png') # Dirección relativa␣
 ↪donde se quiere que se guarde la imagen creada
        plot.show()
```

[83]:
```
agilent = Agilent_MXA_N9020A()
#agilent.setParamsSpectrum(centralFreq , inicialFreq , finalFreq ,␣
 ↪referenceLevel)
```

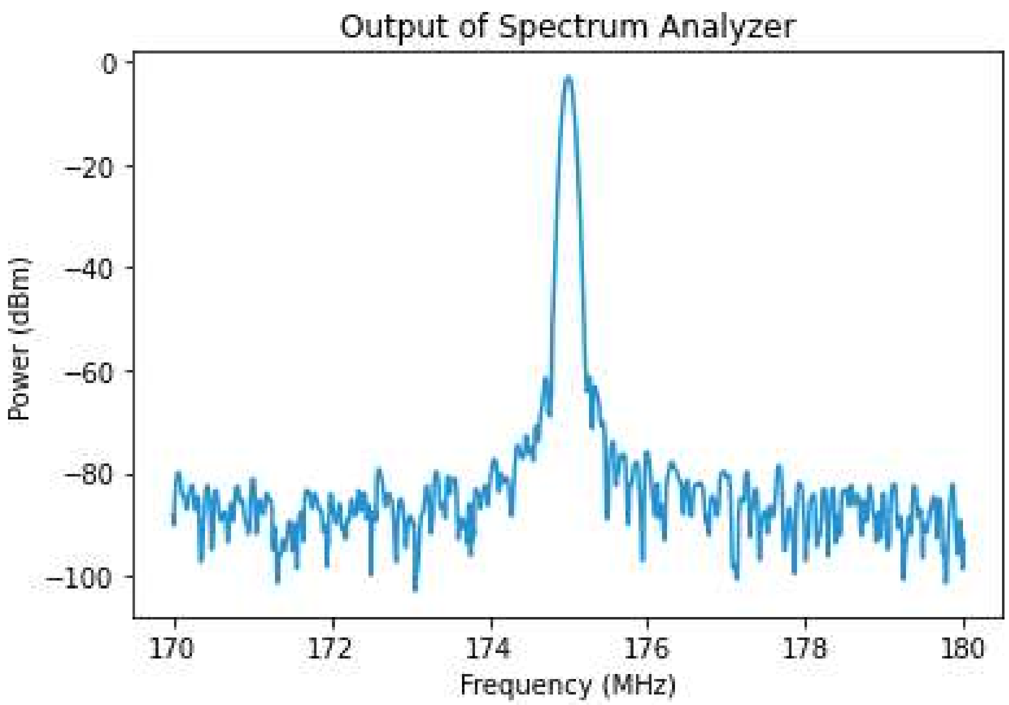
[84]:
```
agilent.identity()
```

```
Fabricante:  Agilent Technologies
Modelo:  N9020A
Número de serie:  MY49100673
Firmware:  A.03.08
```

[85]:
```
agilent.setParamsSpectrum(175,170,180,0.00)
agilent.getParamsSpectrum()
```

```
Frecuencia central:  175.0 MHz
Frecuencia inicial:  170.0 MHz
Frecuencia final:  180.0 MHz
Nivel de referencia:  0.0 dBm
```

[86]:
```
agilent.plotInfo()
```



E.2 MS2830A Signal Analyzer

[6]:
```python
import pyvisa as visa
import numpy as np
from struct import unpack
import pylab
import time
import math
from matplotlib import pyplot as plot

# Establish Connection
rm = visa.ResourceManager('@py') # Calling PyVisaPy library
#scope = rm.open_resource('USB0::0x0699::0x0409::C010730::INSTR') #␣
 ↪Connecting via USB
scope = rm.open_resource('TCPIP::192.168.1.14::INSTR') # Connecting via␣
 ↪LAN
print(scope)
```

```
TCPIPInstrument at TCPIP0::192.168.1.14::inst0::INSTR
```

PyVISA es la librería que voy a usar para comunicarme con la máquina, puesto que es una librería sencilla de usar en Python que permite una comunicación ethernet (que es lo que me interesa) con dispositivos mediante su IP como vemos antes.

Se instala con:

- pip install PyVISA
- pip install PyVISA-py

Pyvisa consta de dos principales funciones una vez hemos establecido conexión:

- write() : es una función con la cual podemos enviar los comandos SCPI (los comandos o lenguaje que entiende la máquina) a la máquina, esta función se usa para cambiar parámetros en la máquina, es decir, cuando usas esta función es para modificar algo, no para esperar una respuesta por parte de la máquina, pues esta función no recoge dato alguno de la máquina salvo el estado de éxito del envío del comando.
- query() : esta función es la que usaremos para solicitar datos o información a la máquina, mediante esta función podemos solicitar datos a la máquina y lo más importante, la máquina nos devuelve estos datos, por lo que cuando usemos esta función será para recoger la información que nos dé la máquina en una variable o imprimirla directamente, puesto que todo lo que nos devuelve la máquina son strings.

[3]:
```
#!pip install PyVISA # Versión: 1.11.3
!pip show PyVISA
```

```
Name: PyVISA
Version: 1.13.0
Summary: Python VISA bindings for GPIB, RS232, TCPIP and USB instruments
Home-page:
Author: Gregor Thalhammer
Author-email: Torsten Bronger <bronger@physik.rwth-aachen.de>
License: The MIT License

        Copyright (c) 2005-2022 PyVISA Authors and contributors. See AUTHORS

        Permission is hereby granted, free of charge, to any person␣
 ↪obtaining a
copy of
        this software and associated documentation files (the "Software"),␣
 ↪to
deal in
        the Software without restriction, including without limitation the
rights to
        use, copy, modify, merge, publish, distribute, sublicense, and/or␣
 ↪sell
copies
        of the Software, and to permit persons to whom the Software is␣
 ↪furnished
```

```
to do
        so, subject to the following conditions:

        The above copyright notice and this permission notice shall be␣
 ↪included
in all
        copies or substantial portions of the Software.

        THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,␣
 ↪EXPRESS
OR
        IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF␣
 ↪MERCHANTABILITY,
        FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT␣
 ↪SHALL
THE
        AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR␣
 ↪OTHER
        LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,␣
 ↪ARISING
FROM,
        OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER␣
 ↪DEALINGS
IN THE
        SOFTWARE.

Location: d:\programas\winpython\wpy64-31050\python-3.10.5.amd64\lib\site-
packages
Requires: typing-extensions
Required-by: PyVISA-py
```

[4]:
```
#!pip install PyVISA-py # Versión: 0.5.2
!pip show PyVISA-py
```

```
Name: PyVISA-py
Version: 0.5.2
Summary: Python VISA bindings for GPIB, RS232, and USB instruments
Home-page: https://github.com/pyvisa/pyvisa-py
Author: Hernan E. Grecco
Author-email: hernan.grecco@gmail.com
License: MIT License
Location: c:\users\salva\appdata\local\packages\pythonsoftwarefoundation.
 ↪python.
3.9_qbz5n2kfra8p0\localcache\local-packages\python39\site-packages
Requires: pyvisa, typing-extensions
```

```
Required-by:
```

[5]:
```
#!pip install matplotlib # Versión: 3.5.0
!pip show matplotlib
```

```
Name: matplotlib
Version: 3.5.0
Summary: Python plotting package
Home-page: https://matplotlib.org
Author: John D. Hunter, Michael Droettboom
Author-email: matplotlib-users@python.org
License: PSF
Location: c:\users\salva\appdata\local\packages\pythonsoftwarefoundation.
 ↪python.
3.9_qbz5n2kfra8p0\localcache\local-packages\python39\site-packages
Requires: cycler, fonttools, kiwisolver, numpy, packaging, pillow,␣
 ↪pyparsing,
python-dateutil, setuptools-scm
Required-by: QT-PyQt-PySide-Custom-Widgets
```

[11]:
```
scope.write('*RST;*CLS')
```

[11]: 11

[14]:
```
scope.write('INST SG')
scope.write('FREQ 175MHZ')
scope.write('UNIT.POW DBM')
scope.write('POW -3.2')
scope.write('OUTP 1')
```

[14]: 8

[13]:
```
# Ejecución de identificador del dispositivo
print(scope.query('*IDN?'))
#print(scope.query(':SYST:COMM:LAN:SCPI:SOCK:CONT?'))
```

```
ANRITSU,MS2830A,6200929104,94.05.01A
```

[36]:
```
# Ejecución para cambiar el instrumento // PAG: 111
# SPECT ==> Spectrum Analyzer
# SIGANA ==> Signal Analyzer
scope.write('INST SPECT')
```

[36]: 12

[35]:
```
# Ejecución para ver que instrumento está activo // PAG: 113
print(scope.query('INST?')) #INST:SYST? SIGANA ==>ç para ver estado␣
 ↪(activo o inactivo)
```

```
SPECT
```

[11]:
```
# Ejecución para modificar la frecuencia central // PAG 244
scope.write('FREQ:CENT 175MHZ')
```

[11]:
```
18
```

[12]:
```
# Ejecución para averiguar la frecuencia central // PAG 244
print(scope.query('FREQ:CENT?'))
```

```
175000000
```

[13]:
```
# Ejecución para modificar el SPAN // PAG 46
scope.write('FREQ:SPAN 10MHZ')
```

[13]:
```
17
```

[15]:
```
# Ejecución para averiguar el SPAN
print(scope.query("FREQ:SPAN?"))
```

```
10000000
```

[37]:
```
#scope.write('FREQ:START 17MHZ')  # Modifica la frecuencia inicial o␣
 ↪mínima
#scope.write('FREQ:STOP 18MHZ')  # Modifica la frecuencia final o máxima
```

[16]:
```
print(scope.query('FREQ:START?')) # Muestra la frecuencia incial o mínima
print(scope.query('FREQ:STOP?')) # Muestra la frecuencia final o máxima
```

```
170000000

180000000
```

[14]:
```
# Con esto configuramos el instrumento para que en los siguientes comandos
# cuando le pidamos la coordenada X e Y, nos de las del punto máximo␣
 ↪(frecuencia/potencia)
scope.write('CALC:MARK:ACT ON; CALC:MARK:RES PEAK; CALC:MARK:MAX')
```

[14]:
```
53
```

```
[15]: print("Frecuencia donde se encuentra la potencia máxima: ",scope.
       ↪query('CALC:MARK:X?'))
      print("Potencia máxima: ",scope.query('CALC:MARK:Y?'))
```

```
Frecuencia donde se encuentra la potencia máxima:  176796000.00

Potencia máxima:  -91.350
```

```
[19]: # Para averiguar la potencia de una determinada frecuencia lo que debemos␣
       ↪hacer
      # es modificar el valor de la coordenada x (frecuencia), de tal forma que␣
       ↪si luego
      # pedimos la coordenada y (la potencia), se nos devolverá la potencia␣
       ↪asociada a la
      # frecuencia definida anteriormente
      scope.write('CALC:MARK:ACT OFF; CALC:MARK:ACT ON; CALC:MARK:X 179MHZ')
```

```
[19]: 57
```

```
[16]: print("Frecuencia definida anteriormente: ",scope.query('CALC:MARK:X?'))
      print("Potencia asociada a la frecuencia definida: ",scope.query('CALC:
       ↪MARK:Y?'))
```

```
Frecuencia definida anteriormente:  176796000.00

Potencia asociada a la frecuencia definida:  -92.911
```

```
[17]: # Para cambiar el nivel de referencia
      scope.write('DISP:WIND:TRAC:Y:RLEV 10.00')
```

```
[17]: 29
```

```
[18]: # Para recoger información de los datos del spectrum analyzer
      # FREQ:CENT ==> la frencuencia central
      # FREQ:START ==> la frencuencia inicial
      # FREQ:STOP ==> la frencuencia final
      # DISP:WIND:TRAC:Y:RLEV ==> el nivel de referencia
      # SWE:POIN? ==> el número de puntos de los que se recoge información

      print(scope.query('FREQ:CENT?; FREQ:START?; FREQ:STOP?; DISP:WIND:TRAC:Y:
       ↪RLEV? ; SWE:POIN?'))

      freqCentral = int(scope.query('FREQ:CENT?'))
```

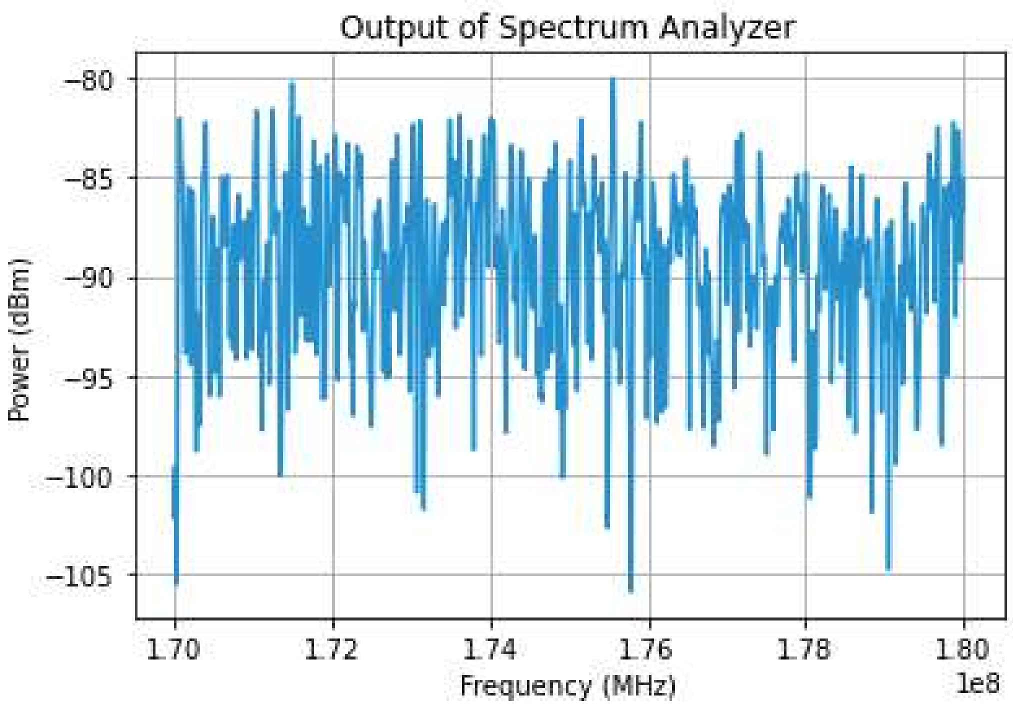
```
freqInicial = int(scope.query('FREQ:START?'))
freqFinal = int(scope.query('FREQ:STOP?'))
nivelRef = float(scope.query('DISP:WIND:TRAC:Y:RLEV?'))
nPuntos = int(scope.query('SWE:POIN?'))
```

```
175000000;170000000;180000000;10.00;10001
```

[33]:
```
# VAMOS A PLOTEAR LA INFORMACIÓN


puntos = int(scope.query('SWE:POIN?')) # guardamos el número de puntos a␣
 ↪representar con plot
initialFreq = float(scope.query('FREQ:START?')) # guardamos la frecuencia␣
 ↪inicial en ese momento
finalFreq = float(scope.query('FREQ:STOP?')) # guardamos la frecuencia␣
 ↪final en ese momento

scope.write('FORM ASC') # Pide a la máquina que lo que se le pide lo␣
 ↪devuelva en formato ASCII
datos = scope.query('TRAC? TRAC1') # Pide a la máquina los datos del␣
 ↪Spectrum (los datos son las potencias de los 10001 puntos observables y␣
 ↪medibles)
print(type(datos[1]))
datosManipulables = datos.split(",") # separa todos los datos separados␣
 ↪por comas y los guarda en una lista
datosManipulables = [float(i) for i in datosManipulables] # transformo␣
 ↪los datos de string a float para poder trabajar con ellos


freq = finalFreq - initialFreq  # definimos la amplitud del intervalo de␣
 ↪frecuencias a representar
pointWidth = freq / float(puntos) # defino la anchura que debe ocupar␣
 ↪cada punto en la imagen (para saber donde colocar cada frecuencia en la␣
 ↪imagen)


# Creo una lista de todas las frecuencias a representar
frequencies = []
count = 0
while len(frequencies) != puntos:
    frequencies.append(initialFreq +(pointWidth*count))
    count+=1
```

```
# Label for x-axis
plot.xlabel("Frequency (MHz)")

# Label for y-axis
plot.ylabel("Power (dBm)")

# title of the plot
plot.title("Output of Spectrum Analyzer")

#Genero la imagen con las frecuencias calculadas y las potencias␣
 ↪ofrecidas por la máquina

plot.plot(frequencies,datosManipulables)
plot.grid()
plot.savefig('./images/graphAgilent.png') # Dirección relativa donde se␣
 ↪quiere que se guarde la imagen creada
plot.show()
```

```
<class 'str'>
```



E.2.1 Definición de librería para máquina Anritsu MS2830A 9 KHz-6 GHz

[30]:
```
# Librerías o Modulos necesarios a importar
import pyvisa as visa
import numpy as np
from struct import unpack
import pylab
import time
from matplotlib import pyplot as plot
```

[34]:
```
########################################################################################
##############             CLASS ANRITSUMS2830A       ␣
 ↪#####################################
##############        SPECTRUM ANALYZER CONTROLLED    ␣
 ↪#####################################
##############       AND SIGNAL GENERATOR CONTROLLED     ␣
 ↪#################################
##############            BY ETHERNET CONTROL                    ␣
 ↪##########################
########################################################################################

class AnritsuMS2830A:

    # La medida con la que vamos a trabajar van a ser los MHz
    medida ='MHZ'

########################################################################################
############## CONSTRUCTOR␣
 ↪##################################################################
########################################################################################

    # Cuando creemos un objeto de la clase, ejecutaremos el setup()␣
 ↪conectándose automáticamente a la máquina
    # mediante conexión TCP

    def __init__(self):

        self.scope= self.setup()
        self.nPoints = int(self.scope.query('SWE:POIN?'))# reogemos el␣
 ↪número de puntos de la máquina

        #if int(mode)==0:
            #self.setSignalGen()
        #else:
            #self.setSpectrum()

#####################################################################################
############## IDENTITY & SETUP␣
↪############################################################
#####################################################################################

    # Muestra la información propia ofrecida por la máquina

    def identity(self):
        info= self.scope.query('*IDN?')
        info = info.split(",")
        print("Fabricante: ",info[0])
        print("Modelo: ",info[1])
        print("Número de serie: ",info[2])
        print("Firmware: ",info[3])

    # Establece una conexión TCP con la máquina mediante su IP␣
↪devolviendo el objeto conectado para poder manejarlo

    def setup(self):
        # 192.168.1.139 IP MÁQUINA ANRITSU
        rm = visa.ResourceManager('@py') # Calling PyVisaPy library
        scope = rm.open_resource('TCPIP::192.168.1.139::INSTR') #␣
↪Connecting via LAN
        #scope.write('*RST;*CLS') # resetea la máquina
        return scope

#####################################################################################
############## GETTERS & SETTERS␣
↪###########################################################
#####################################################################################

    # Activa el modo Spectrum en la máquina

    def setSpectrum(self):
        self.scope.write('INST SPECT')
        #print("Ha seleccionado el Spectrum Analyzer")

    # Activa el modo Generador en la máquina

    def setSignalGen(self):
        self.scope.write('INST SG')
        #print("Ha seleccionado el Signal Generator")

#######################################################################################
######### FUNCTIONS FOR SIGNAL GENERATOR␣
↪##################################################
#######################################################################################

    # Muestra en terminal la información relativa al generador: la␣
↪frecuencia, la potencia y si está o no encendido

    def getParamsGenerator(self):
        print("Frecuencia del generador: ",self.frequency,"MHz")
        print("Potencia del generador: ",self.power,"dBm")
        if self.state == 1:
            print("Estado del generador: Activado")
        else:
            print("Estado del generador: Desactivado")

    # Modifica todos los parámetros del generador de una sola vez

    def setParamsGenerator(self, frequency , power):
        self.scope.write('FREQ '+ str(frequency) + self.medida) #␣
↪Modifica la frecuencia
        self.scope.write('UNIT.POW DBM') # Modifica las unidades de la␣
↪potencia DBM
        self.scope.write('POW ' + str(power)) # Modifica la potencia
        self.scope.write('OUTP 1') # Enciende el Generador
        # Guarda todos los datos en variables del objeto de la clase
        self.frequency = float(frequency)
        self.power = float(power)
        self.state = 1

    # Modifica la frecuencia y el atributo del objeto de la clase␣
↪correspondiente

    def setFrequencyMHz(self, frequency):
        self.scope.write('FREQ '+ str(frequency) + self.medida)
        self.frequency = float(frequency)

    # Muestra la frecuencia en ese instante

    def getFrequencyMHz(self):
        print("Frecuencia del generador: ",self.frequency," MHz")
    # Modifica la potencia y el atributo del objeto de la clase␣
↪correspondiente

    def setPowerGeneratordBm(self,power):
        self.scope.write('POW ' + str(power))
        self.power = float(power)

    # Muestra la potencia en ese instante
    def getPowerGeneratordBm(self):
        print("Potencia del generador: ",self.power," dBm")

    # Función capaz de encender o apagar el generador (1 enciende, 0␣
↪apaga) y guarda el estado en un atributo del objeto de la clase␣
↪correspondiente

    def setStateGenerator(self,state):
        self.scope.write('OUTP '+str(state))
        self.state = int(state)

    # Muestra el estado del generador (1 => encendido , cualquier otro =>␣
↪apagado (0))

    def getStateGenerator(self):

        if self.state == 1:
            print("El Generador está activado | estado: ", self.state)
        else:
            print("El Generador está desactivado | estado: ", self.state)



###########################################################################
######### FUNCTIONS FOR SPECTRUM ANALYZER␣
↪###################################################
###########################################################################

    # Muestra todos los parámetros del Espectro en ese momento

    def getParamsSpectrum(self):
        print("Frecuencia central: ",self.centralFreq ,"MHz") # Muestra␣
↪la frecuencia central
        print("Frecuencia inicial: ",self.inicialFreq,"MHz") # Muestra la␣
↪frecuencia inicial
```

```
        print("Frecuencia final: ",self.finalFreq,"MHz") # Muestra la␣
↪frecuencia final
        print("Nivel de referencia: ",self.referenceLevel,"dBm") #␣
↪Muestra el nivel de referencia

    # Modifica todos los parámetros del Spectrum Analyzer (la medida de␣
↪las frecuencias está en MHz)

    def setParamsSpectrum(self,inicialFreq , finalFreq , referenceLevel):
        # Guarda todos los datos en variables del objeto de la clase
        self.inicialFreq = float(inicialFreq)
        self.finalFreq = float(finalFreq)
        self.referenceLevel = float(referenceLevel)
        self.centralFreq = (self.finalFreq + self.inicialFreq)/2.0

        self.scope.write('FREQ:START '+ str(inicialFreq) + self.medida) #␣
↪Modifica la frecuencia inicial
        self.scope.write('FREQ:STOP '+ str(finalFreq) + self.medida) #␣
↪Modifica la frecuencia final
        self.scope.write('FREQ:CENT '+ str(self.centralFreq)+ self.
↪medida) # Modifica la frecuencia central
        self.scope.write('DISP:WIND:TRAC:Y:RLEV '+ str(referenceLevel)) #␣
↪Modifica el nivel de referencia
        # Se define el número de puntos observables y medibles al valor␣
↪que tenga la máquina en ese momento (por defecto son 10001)
        self.nPoints = int(self.scope.query('SWE:POIN?'))



    # Modifica todos los parámetros del Spectrum Analyzer mediante el uso␣
↪de span (la medida de las frecuencias está en MHz)

    def setParamsSpectrumSpan(self, centralFreq , span , referenceLevel):
            self.setCentralFreqMHz(centralFreq) # Modifica la frecuencia␣
↪central y su atributo de la clase
            self.setSpanMHz(span) # llama a la función modificando el␣
↪valor de span inicialFreq y finalFreq y los atributos de la clase
            self.setReferenceLevelDBM(referenceLevel) # llama a la␣
↪función modificando el valor de referencelevel y el atributo de la␣
↪clase
            # Se define el número de puntos observables y medibles al␣
↪valor que tenga la máquina en ese momento (por defecto son 10001)
            self.nPoints = int(self.scope.query('SWE:POIN?'))
    # Modifica el valor del span, inicialFreq y finalFreq y aparte los␣
↪atributos de la clase correspondientes a la frecuencia inicial y final␣
↪y el span
    # (Hace falta haber definido la frecuencia central)

    def setSpanMHz(self, span):
        self.span = float(span)
        mitad = self.span / 2.0
        self.scope.write('FREQ:SPAN '+ str(span)+ self.medida)

        self.inicialFreq = self.centralFreq - mitad
        self.scope.write('FREQ:START '+ str(self.inicialFreq) + self.
↪medida)
        self.finalFreq = self.centralFreq + mitad
        self.scope.write('FREQ:STOP '+ str(self.finalFreq) + self.medida)

    # Muestra el Span en ese momento

    def getSpanMHz(self):
        print("Span: ", self.span , " MHz")



    # Modifica el valor de la frecuencia central y el atributo del objeto␣
↪de la clase correspondiente

    def setCentralFreqMHz(self,centralFreq):
        self.centralFreq = float(centralFreq)
        self.scope.write('FREQ:CENT '+ str(centralFreq)+ self.medida)

    # Muestra la frecuencia central en ese momento

    def getCentralFreqMHz(self):
        print("Frecuencia central: ",self.centralFreq," MHz")

    # Modifica la frecuencia incial y el atributo del objeto de la clase␣
↪correspondiente

    def setInicialFreqMHz(self,inicialFreq):
        self.inicialFreq = float(inicialFreq)
        self.scope.write('FREQ:START '+ str(inicialFreq) + self.medida)
    # Muestra la frecuencia inicial en ese momento

    def getInicialFreqMHz(self):
        print("Frecuencia inicial: ",self.inicialFreq," MHz")

    # Modifica la frecuencia final y el atributo del objeto de la clase␣
↪correspondiente

    def setFinalFreqMHz(self,finalFreq):
        self.finalFreq = float(finalFreq)
        self.scope.write('FREQ:STOP '+ str(finalFreq) + self.medida)

    # Muestra la frecuencia final en ese momento

    def getFinalFreqMHz(self):
        print("Frecuencia final: ",self.finalFreq," MHz")

    # Modifica el nivel de referencia y el atributo del objeto de la␣
↪clase correspondiente

    def setReferenceLeveldBm(self,referenceLevel):
        self.referenceLevel = float(referenceLevel)
        self.scope.write('DISP:WIND:TRAC:Y:RLEV '+ str(referenceLevel))

    # Muestra el nivel de referencia en ese momento

    def getReferenceLeveldBm(self):
        print("Nivel de referencia: ",self.referenceLevel," dBm")

    # Muestra y devuelve el número de puntos observables y medibles en␣
↪ese momento

    def getNumPoints(self):
        puntos = int(self.scope.query('SWE:POIN?'))
        #print("Número de puntos: ",puntos)
        return puntos

    # Modifica el número de puntos observables y medibles y el atributo␣
↪del objeto de la clase correspondiente

    def setNumPoints(self,npoints):
        self.nPoints = int(npoints)
        self.scope.write('SWE:POIN '+ str(npoints))
    # Muestra y devuelve la frecuencia donde se encuentra la potencia␣
↪máquina y la potencia máxima

    def getMaxFreqPower(self):
        self.scope.write('CALC:MARK:ACT ON; CALC:MARK:RES PEAK; CALC:MARK:
↪MAX')
        print("Frecuencia donde se encuentra la potencia máxima: ",self.
↪scope.query('CALC:MARK:X?'), " MHz")
        print("Potencia máxima: ",self.scope.query('CALC:MARK:Y?'), "␣
↪dBm")
        freq = float(self.scope.query('CALC:MARK:X?'))
        power = float(self.scope.query('CALC:MARK:Y?'))
        return freq,power

    # Función que devuelve la potencia referente a una frecuencia dada

    def getPowerdBm(self, freq):
        self.scope.write('CALC:MARK:ACT OFF; CALC:MARK:ACT ON; CALC:MARK:
↪X '+ str(freq)+ self.medida)
        print("Potencia asociada a la frecuencia dada: ",self.scope.
↪query('CALC:MARK:Y?'), " dBm")

##########################################################################################
############## PLOT INFORMATION ␣
↪##########################################################
##########################################################################################

    # Función que guarda una imagen png y la muestra en pantalla de la␣
↪señal del Spectrum completa en ese momento

    def plotInfo(self):

        puntos = self.getNumPoints() # guardamos el número de puntos a␣
↪representar con plot

        self.scope.write('FORM ASC') # Pide a la máquina que lo que se le␣
↪pide lo devuelva en formato ASCII
        datos = self.scope.query('TRAC? TRAC1') # Pide a la máquina los␣
↪datos del Spectrum (los datos son las potencias de los 10001 puntos␣
↪observables y medibles)
        datosManipulables = datos.split(",") # separa todos los datos␣
↪separados por comas y los guarda en una lista
```

```
        datosManipulables = [float(i) for i in datosManipulables] #␣
↪transformo los datos de string a float para poder trabajar con ellos


        freq = self.finalFreq - self.inicialFreq # definimos la amplitud␣
↪del intervalo de frecuencias a representar
        pointWidth = freq / float(puntos) # defino la anchura que debe␣
↪ocupar cada punto en la imagen (para saber donde colocar cada␣
↪frecuencia en la imagen)


        # Creo una lista de todas las frecuencias a representar
        frequencies = []
        count = 0
        while len(frequencies) != puntos:
            frequencies.append(self.inicialFreq+(pointWidth*count))
            count+=1

        # Label for x-axis
        plot.xlabel("Frequency (MHz)")

        # Label for y-axis
        plot.ylabel("Power (dBm)")

        # title of the plot
        plot.title("Output of Spectrum Analyzer")

        #Genero la imagen con las frecuencias calculadas y las potencias␣
↪ofrecidas por la máquina

        plot.plot(frequencies,datosManipulables)
        plot.savefig('./images/graphAnritsu.png') # Dirección relativa␣
↪donde se quiere que se guarde la imagen creada
        plot.show()
```

[35]:
```
anritsu = AnritsuMS2830A()
#anritsu.setParamsSpectrum(centralFreq , inicialFreq , finalFreq ,␣
↪referenceLevel)
```

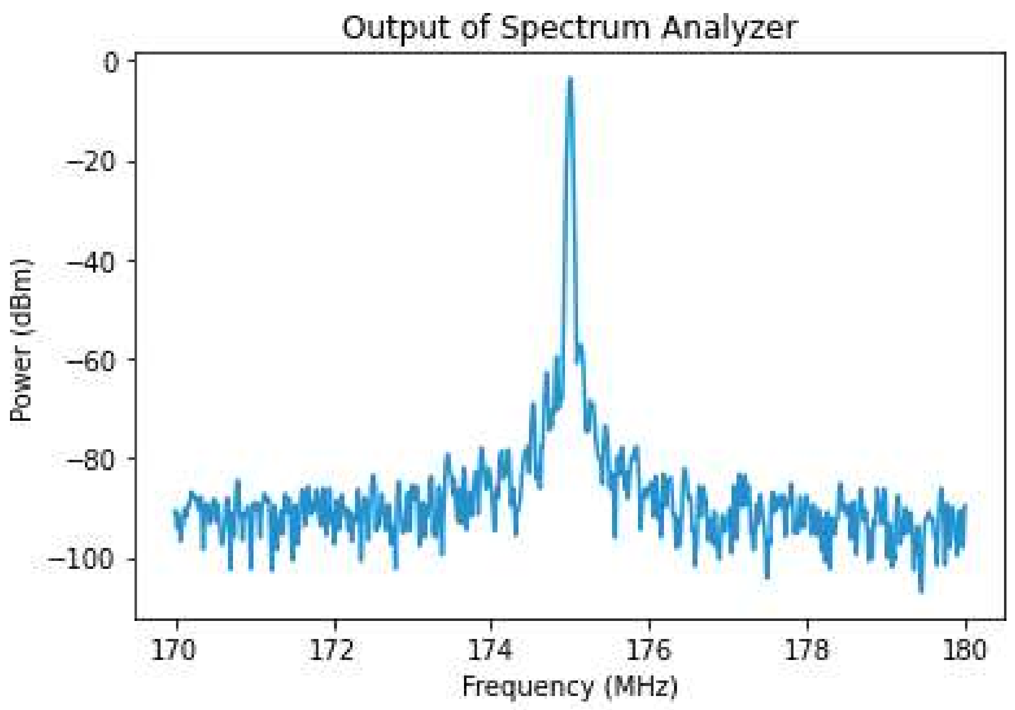
[36]:
```
anritsu.identity()
anritsu.setSignalGen()
#anritsu.setParamsGenerator(175,-3.2)
anritsu.setFrequencyMHz(175)
anritsu.setPowerGeneratordBm(-3.2)
anritsu.setStateGenerator(1)
anritsu.getParamsGenerator()
```

```
Fabricante:  ANRITSU
Modelo:  MS2830A
Número de serie:  6200929104
Firmware:  94.05.01A

Ha seleccionado el Signal Generator
Frecuencia del generador:  175.0 MHz
Potencia del generador:  -3.2 dBm
Estado del generador: Activado
```

[37]:
```
anritsu.setSpectrum()
anritsu.setParamsSpectrum(175,170,180,0.00)
anritsu.getParamsSpectrum()
```

```
Ha seleccionado el Spectrum Analyzer
Frecuencia central:  175.0 MHz
Frecuencia inicial:  170.0 MHz
Frecuencia final:  180.0 MHz
Nivel de referencia:  0.0 dBm
```

[39]:
```
anritsu.plotInfo()
```



E.3 MSO-X-4104A Oscilloscope

Conexión con el osciloscopio

Para establecer conexión con el dispositivo por USB basta con conectarlo con un ordenador por el puerto USB-B de la interfaz HO720 de la parte trasera, el osciloscopio se identificará correctamente. Para poder comunicarse con el osciloscopio y enviar comandos es necesario instalar el driver de la interfaz HO720 aquí.

[1]:
```
# Libraries and modules needed to connect/work with the VNA:
import pyvisa as visa
#import usb.core
#import usb.util
import numpy as np
import math
import pylab
from matplotlib import pyplot as plot
from PIL import Image
import sys
from engineering_notation import EngNumber
```

[2]:
```
# Llamar a pip e imprimir versiones de librerias en uso.
!pip show pyvisa pylab-sdk
```

```
Name: PyVISA

WARNING: Package(s) not found: pylab-sdk


Version: 1.13.0
Summary: Python VISA bindings for GPIB, RS232, TCPIP and USB instruments
Home-page:
Author: Gregor Thalhammer
Author-email: Torsten Bronger <bronger@physik.rwth-aachen.de>
License: The MIT License

        Copyright (c) 2005-2022 PyVISA Authors and contributors. See AUTHORS

        Permission is hereby granted, free of charge, to any person␣
 ↪obtaining a
copy of
        this software and associated documentation files (the "Software"),␣
 ↪to
deal in
```

```
Location: d:\programas\winpython\wpy64-31050\python-3.10.5.amd64\lib\site-
packages
Requires: typing-extensions
Required-by: PyVISA-py
```

[2]:
```
class Agilent_MXO_4104A:
    ch1 = False
    ch2 = False
    ch3 = False
    ch4 = False
    #Creation of a new instance of the object, first stablishes a␣
↪connection via TCP/IP with the VNA, then asks for the identification␣
↪parameters (*IDN? command).
    def __init__(self):
        self.scope = self.startup()
        print(self.scope)

        self.identify()
        if self.selectChannel(1, True)[0:1:1] == '1':
            self.ch1 = True

        if self.selectChannel(2, True)[0:1:1] == '1':
            self.ch2 = True

        if self.selectChannel(3, True)[0:1:1] == '1':
            self.ch3 = True

        if self.selectChannel(4, True)[0:1:1] == '1':
            self.ch4 = True

### Initialization methods:
    # Network/USB connection to the oscilloscope through pyvisa module.
    # Need to specify the connection to the device in the .open_resource␣
↪method:
      # For LAN connection, you must specify the IP and port if necessary␣
↪like this: TCPIP[board]::host address[::INSTR] for only IP and␣
↪TCPIP[board]::host address::port::SOCKET for IP and port.
      # For USB connection use the VISA address indicated by the device:␣
↪USB0::2391::6064::MY53110138::INSTR
    def startup(self):
        rm = visa.ResourceManager('@py') # Calling PyVisaPy library
        newScope = rm.open_resource('TCPIP0::a-mx4104a-10138::INSTR')
        return newScope

    # Returns and prints the details of the device:
    def identify(self):
        idn = self.scope.query('*IDN?')
        idn = idn.split(',')

        print('\nFabricante: ', idn[0])
        print('Modelo: ', idn[1])
        print('Número de serie: ', idn[2])
        print('Versión de firmware: ', idn[3])
    #Commands the serial object to close, allows for other apps or␣
↪instances of this object to use the port
    def close(self):
        try:
            self.scope.close()
            print('Cerrando la conexión con el puerto COM del␣
↪osciloscopio...')
        except:
            print('La conexión ya se encuentra activa.')

    #Commands the serial object to open again.
    def open(self):
        try:
            self.scope.open()
            print('Iniciando la conexión con el  puerto COM del␣
↪osciloscopio...')
        except:
            print('La conexión ya se encuentra activa o el puerto está␣
↪siendo usado por otra conexión.')

  ## Channel methods:
    # Sets/gets the state of the indicated channel: CHAN<ch>:DISP[?]␣
↪[ON|OFF]
    # Needs an integer to indicate which channel to use and a boolean to␣
↪select wheter to ask (True) or set (False) the channel state, when␣
↪question is True, channel is ignored.
    # channel can range from 1 to 4, any value outside this range will␣
↪default to 1.
    def selectChannel(self, channel: int, question: bool):
        if channel < 1 and channel > 4:
            channel = 1
            print('El valor introducido no es válido, volviendo al valor␣
↪por defecto.')

        if question:
            query = self.scope.query(':CHAN' + str(channel) + ':DISP?')

            return query
        else:
            match channel:
                case 1:
                    if self.ch1:
                        print('Cerrando el canal 1...')
                    self.ch1 = False
                    self.scope.write('CHAN1:DISP OFF')
                    print('Canal 1 cerrado.')
                else:
                    print('Abriendo el canal 1...')
                    self.ch1 = True
                    self.scope.write('CHAN1:DISP ON')
                    print('Canal 1 abierto.')

            case 2:
                if self.ch2:
                    print('Cerrando el canal 2...')
                    self.ch2 = False
                    self.scope.write('CHAN2:DISP OFF')
                    print('Canal 2 cerrado.')
                else:
                    print('Abriendo el canal 2...')
                    self.ch2 = True
                    self.scope.write('CHAN2:DISP ON')
                    print('Canal 2 abierto.')

            case 3:
                if self.ch3:
                    print('Cerrando el canal 3...')
                    self.ch3 = False
                    self.scope.write('CHAN3:DISP OFF')
                    print('Canal 3 cerrado.')
                else:
                    print('Abriendo el canal 3...')
                    self.ch3 = True
                    self.scope.write('CHAN3:DISP ON')
                    print('Canal 3 abierto.')

            case 4:
                if self.ch4:
                    print('Cerrando el canal 4...')
                    self.ch4 = False
                    self.scope.write('CHAN4:DISP OFF')
                    print('Canal 4 cerrado.')
                else:
                    print('Abriendo el canal 4...')
                    self.ch4 = True
                    self.scope.write('CHAN4:DISP ON')
                    print('Canal 4 abierto.')
```

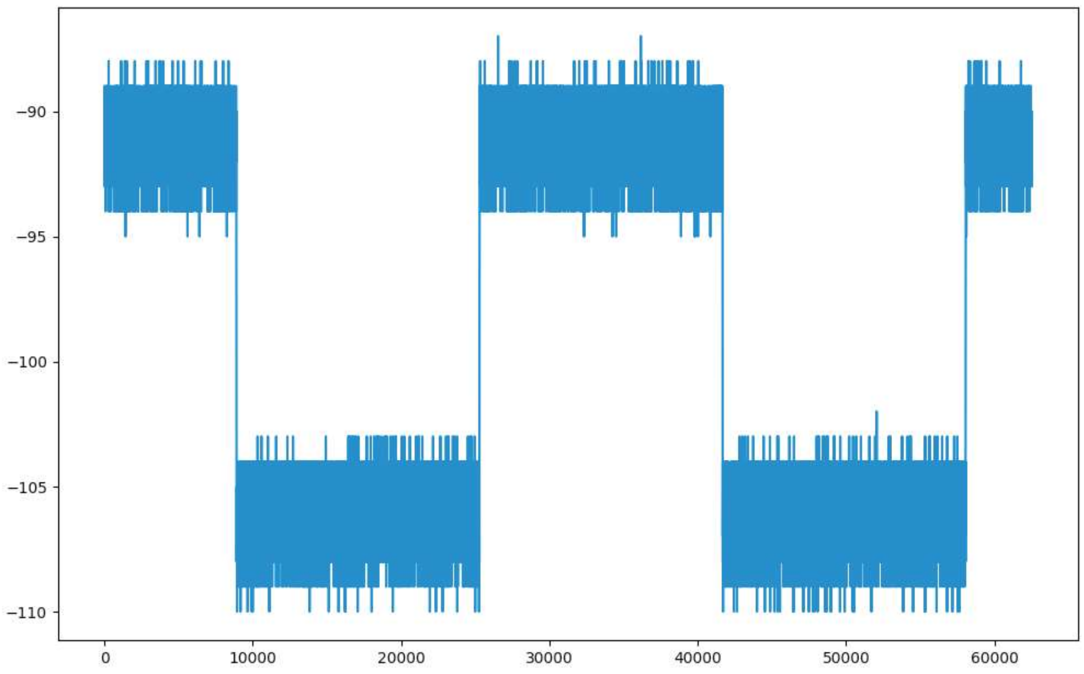
```
    # Gets all the data from the on-screen wavefrom: CHAN<m>:DATA?
    # Doesn't require any parameter.
    def collectData(self, channel: int):
        if channel < 1 and channel > 4:
            channel = 1
            print('El valor introducido no es válido, volviendo al valor␣
↪por defecto.')

        self.scope.write(':WAV:SOUR CHAN' + str(channel))
        data = self.scope.query_binary_values(':WAV:DATA?', datatype='B',␣
↪is_big_endian=False)
        return data

    # Sets/gets the time scale for measurement: TIM_SCAL[?] [float]
    # This scale indicates the time represented in each of the 12␣
↪horizontal grid divisions e.g.: if the device is set to 1 seconds, the␣
↪whole screen will represent a total of 12 seconds.
    # Needs an float to indicate the time scale and a boolean to select␣
↪whether to ask (True) or not (False) for the actual scale, when␣
↪question is true, time is ignored.
    # time can range from 2 nanoseconds (2e-9) to 50 seconds (50), any␣
↪value outside this range will default to 100 microseconds (100e-6).
    def timeScale(self, time: float, question: bool):
        if question:
            query = self.scope.query('TIM:SCAL?')

            return query
        else:
            if time < 2e-9 or time > 50:
                time = 100e-6
                print('La escala indicada no es válida, volviendo al␣
↪valor por defecto.')

            self.scope.write('TIM:SCAL ' + str(time))

    # Sets/gets the voltage scale for measurement: CHAN<ch>:SCAL[?]␣
↪[float]
    # This scale indicates the voltage represented in each of the 8␣
↪horizontal grid divisions e.g.: if the device is set to 2V, the whole␣
↪screen will represent a total of 16V.
```

```
    # Needs an float to indicate the voltage scale and a boolean to␣
↪select whether to ask (True) or not (False) for the actual scale, when␣
↪question is true, volts is ignored.
    # time can range from 1mV (1e-3) to 10V (10), any value outside this␣
↪range will default to 5mV (5e-3).
    def voltScale(self, volts: float, channel: int, question: bool):
        if channel < 1 and channel > 4:
            channel = 1
            print('El valor introducido no es válido, volviendo al valor␣
↪por defecto.')

        if question:
            query = self.scope.query('CHAN' + str(channel) + ':SCAL?')

            return query
        else:
            if volts < 1e-3 or volts > 10:
                volts = 5e-3
                print('El voltaje indicado no es válido, volviendo al␣
↪valor por defecto.')

            self.scope.write('CHAN' + str(channel) + ':SCAL ' +␣
↪str(volts))

        # Sets/gets the number of measurement point: CHAN<m>:DATA:POIN[?]␣
↪[integer]
    # Needs a string to select the possible range of points used, an␣
↪integer to indicate which channel to consult and a boolean to select␣
↪whether to ask (True) or not (False) for the actual number, when␣
↪question is true, range is ignored.
    # The possible values for range are:
      # NORM: The default number of points.
      # MAX: The maximum number of points of the device (10.000.000).
      # RAW: The maximun number of points returned by the :ACQuire:POINts?
↪ command.
    # channel can range from 1 to 4, any value outside this range will␣
↪default to 1.
    def measurementPoints(self, range: str, question: bool):
        if question:
            query = self.scope.query(':WAV:POIN?')

            return query
        else:
            if channel < 1 or channel > 4:
                channel = 1
                print('El canal indicado no es válido, volviendo al valor␣
↪por defecto.')

            self.scope.write(':WAV:POIN?')

    # Sets/gets the offset of the signal from the indicated channel:␣
↪CHAN<m>:OFFS[?] [float]
    # Needs a float to indicate the offset value, an integer to select␣
↪the channel to work with an a boolean to select whether to ask (True)␣
↪or not (False) for the actual number, when question is true, offset is␣
↪ignored.
    # offset can range from -10 to 10, any value outside this range will␣
↪default to 0.
    # channel can range from 1 to 4, any value outside this range will␣
↪default to 1.
    def verticalOffset(self, offset: float, channel: int, question: bool):
        if question:
            query = self.scope.query(':CHAN' + str(channel) + ':OFFS?')
            print('Offset del canal ' + str(channel) + ' es: ' + query)

            return query
        else:
            if channel < 1 or channel > 4:
                channel = 1
                print('El canal indicado no es válido, volviendo al valor␣
↪por defecto.')

            if offset < 1 or offset > 4:
                offset = 1
                print('El valor indicado no es válido, volviendo al valor␣
↪por defecto.')

            self.scope.write(':CHAN' + str(channel) + ':OFFS ' +␣
↪str(offset))

## Aditional Methods:
    # Gets the necessary data to build.
    # Needs a list that contains numeric values.
    def getBounds(self):
        xFlags = []
        yFlags = []
        volts = []
        factors = []

        # Get time scale and formart it.
        time = float(self.timeScale(0.0, True))
        #print('Tiempo por división: ' + str(time))

        if time < 1e-7:
            xScale = 'ns'
            time *= 1e9
        elif time < 1e-6:
            xScale = 'µs'
            time *= 1e6
        elif time < 1e-4:
            xScale = 'µs'
            time *= 1e6
        elif time < 1e-3:
            xScale = 'ms'
            time *= 1e3
        elif time < 1e-1:
            xScale = 'ms'
            time *= 1e3
        elif time < 1e-0:
            xScale = 's'
        #print('Tiempo adaptado: ' + str(time))

        for i in range(0, 13, 1):
            #xFlags.append("{:.1f}".format(time * i))
            xFlags.append(str(EngNumber(time * i))) # To make the number␣
↪appear in engineering notation (most probably will see m after some␣
↪number with certain values).
        #print(xFlags)

        # Get the voltage scale, format the numbers and set the correct␣
↪unit.
        if self.ch1:
            volt1 = float(self.voltScale(0.0, 1, True))
            volts.append(volt1)

            for i in range(-4, 5, 1):
                yFlags.append(str(EngNumber(volt1 * i)))
        else:
            volts.append(0)
        if self.ch2:
            volt2 = float(self.voltScale(0.0, 2, True))
            volts.append(volt2)

            if self.ch1:
                for i in range(-4, 5, 1):
                    if i != 0:
                        yFlags[i + 4] = yFlags[i + 4] + '/' +␣
↪(str(EngNumber(volt2 * i)))
            else:
                for i in range(-4, 5, 1):
                    yFlags.append(str(EngNumber(volt2 * i)))

            factors.append(volt1/volt2)
        else:
            volts.append(0)
            factors.append(0)

        if self.ch3:
            volt3 = float(self.voltScale(0.0, 3, True))
            volts.append(volt3)

            if self.ch1 or self.ch2:
                for i in range(-4, 5, 1):
                    if i != 0:
                        yFlags[i + 4] = yFlags[i + 4] + '/' +␣
↪(str(EngNumber(volt3 * i)))
            else:
                for i in range(-4, 5, 1):
                    yFlags.append(str(EngNumber(volt3 * i)))

            factors.append(volt1/volt3)
        else:
            volts.append(0)
            factors.append(0)

        if self.ch4:
            volt4 = float(self.voltScale(0.0, 4, True))
            volts.append(volt4)

            if self.ch1 or self.ch2 or self.ch3:
                for i in range(-4, 5, 1):
                    if i != 0:
                        yFlags[i + 4] = yFlags[i + 4] + '/' +␣
↪(str(EngNumber(volt4 * i)))
                else:
                    for i in range(-4, 5, 1):
                        yFlags.append(str(EngNumber(volt4 * i)))

                factors.append(volt1/volt3)
            else:
                volts.append(0)
                factors.append(0)

            #print(yFlags)
            #print(factors)

            return xScale, xFlags, yFlags, volts, factors

 ## Main Method:
    # Prints the values of the active channels.
    # Needs an boolean to indicate whether print all the channels in the␣
↪same graphic (False) or in separate graphics (True).
    def plotChannels(self, split: bool):
        # Add this Plot objects to correctly define the plot grid.
        fig = plot.figure(figsize=(12, 7.5)) #and casually define the␣
↪plot size (in inches).
        axes = fig.add_subplot(1, 1, 1)

        data1 = ''
        off1 = ''
        finalData1 = []
        data2 = ''
        off2 = ''
        finalData2 = []
        data3 = ''
        off3 = ''
        finalData3 = []
        data4 = ''
        off4 = ''
        finalData4 = []

        # Check if each channel is active or not, active channels will be␣
↪the ones ploted.
        if self.ch1:
            data1 = self.collectData(1)
            off1 = float(self.verticalOffset(0, 1, True))
            #print(data1)
            #print(off1)
        if self.ch2:
            data2 = self.collectData(2)
            #print(data2)
            off2 = float(self.verticalOffset(0, 2, True))
        if self.ch3:
            data3 = self.collectData(3)
            off3 = float(self.verticalOffset(0, 3, True))
            print(off3)
        if self.ch4:
            data4 = self.collectData(4)
            off4 = float(self.verticalOffset(0, 4, True))
            print(off4)

        #print(points)
        # Add list for the X axys
        points = int(self.measurementPoints('', True))
        x = range(0, points, 1)

        # Check which mode is activated, any oscilloscope math mode or␣
↪FFT(Frequencies) mode.
        #---> ##########revisar esto########## <---#
        #mode = self.scope.query('CALC:MATH?')
        #mode = mode[1:8:1]
        mode = 'NOTFFT'
        #Determine the correct divisions for the plot grid, to resemble␣
↪more closely the screen's divisions
        if mode != 'FFTMAG':
            bounds = self.getBounds()
            xScale = bounds[0]
            xFlags = bounds[1]
            yFlags = bounds[2]
            volts = bounds[3]
            factors = bounds[4]
            print(factors)

            #Determine the correct divisions for the plot grid, to␣
↪resemble more closely the screen's divisions
            xDivs = range(0, points + 1, int(points/12))
            volt = float(self.voltScale(0.0, 1, True))
            yDivs = np.arange(volt * -4, volt * 5, volt)
            #print(yDivs)
            axes.set_xticks(xDivs, xFlags)
            axes.set_yticks(yDivs, yFlags)
            #plot.ylim(volt * -5, volt * 5)
            #print(str(volt * -5) + ', ' + str(volt * 5))
            plot.xlabel('Time (' + xScale + ')')
            plot.ylabel('Voltage (V)')
        else:
            print('algo')


        if split:
            if len(data1) > 0:
                plot.subplot(2, 2, 1)
                plot.title('Output of the channel 1')
                plot.grid()

                for i in range (0, len(data1), 1):
                    finalData1.append(data1[i] + off1 * volts[0])

                plot.plot(x, finalData1, 'y')
            if len(data2) > 0:
                plot.subplot(2, 2, 2)
                plot.title('Output of the channel 2')
                plot.grid()

                for i in range (0, len(data2), 1):
                    finalData2.append(data2[i] * factors[0] + off2 *␣
↪volts[0])

                plot.plot(x, finaData2, 'b')
            if len(data3) > 0:
                plot.subplot(2, 2, 3)
                plot.title('Output of the channel 3')
                plot.grid()

                for i in range (0, len(data3), 1):
                    finalData3.append(data3[i] * factors[1] + off3 *␣
↪volts[0])

                plot.plot(x, finalData3, color='violet')
            if len(data4) > 0:
                plot.subplot(2, 2, 4)
                plot.title('Output of the channel 4')
                plot.grid()

                for i in range (0, len(data4), 1):
                    finalData4.append(data4[i] * factors[2] + off4 *␣
↪volts[0])

                plot.plot(x, finalData4, 'g')
        else:
            plot.title('Output of the oscilloscope')
            plot.grid()

            if len(data1) > 0:
                for i in range (0, len(data1), 1):
                    finalData1.append(data1[i] + off1 * volts[0])
                #print(finalData1)

                plot.plot(x, finalData1, 'y')
            if len(data2) > 0:
                for i in range (0, len(data2), 1):
                    finalData2.append(data2[i] * factors[0] + off2 *␣
↪volts[0])

                plot.plot(x, finalData2, 'b')
            if len(data3) > 0:
                for i in range (0, len(data3), 1):
                    finalData3.append(data3[i] * factors[1] + off3 *␣
↪volts[0])

                plot.plot(x, finalData3, color='violet')
            if len(data4) > 0:
                for i in range (0, len(data4), 1):
                    finalData4.append(data4[i] * factors[2] + off4 *␣
↪volts[0])

                plot.plot(x, finalData4, 'g')

            plot.show()

 ## Screnshot:
   # Gets a screenshot of the oscilloscope display an saves it in PNG␣
↪format in the Notebook folder.
   # Needs a string to indicate the file to save and a boolean to select␣
↪whether set a white (True) or black (False) background for the image.
   def screenshot(self, filename: str, whiteBack: bool):
        if whiteBack:
            self.scope.write(':HARD:INKS ON')
        else:
            self.scope.write(':HARD:INKS OFF')
        rgb = self.scope.query_binary_values(':HCOPY:SDUM:DATA? PNG',␣
↪datatype='B', is_big_endian=False)

        data = bytes(rgb)
        with open(filename + '.png', 'wb') as f:
            f.write(data)
```

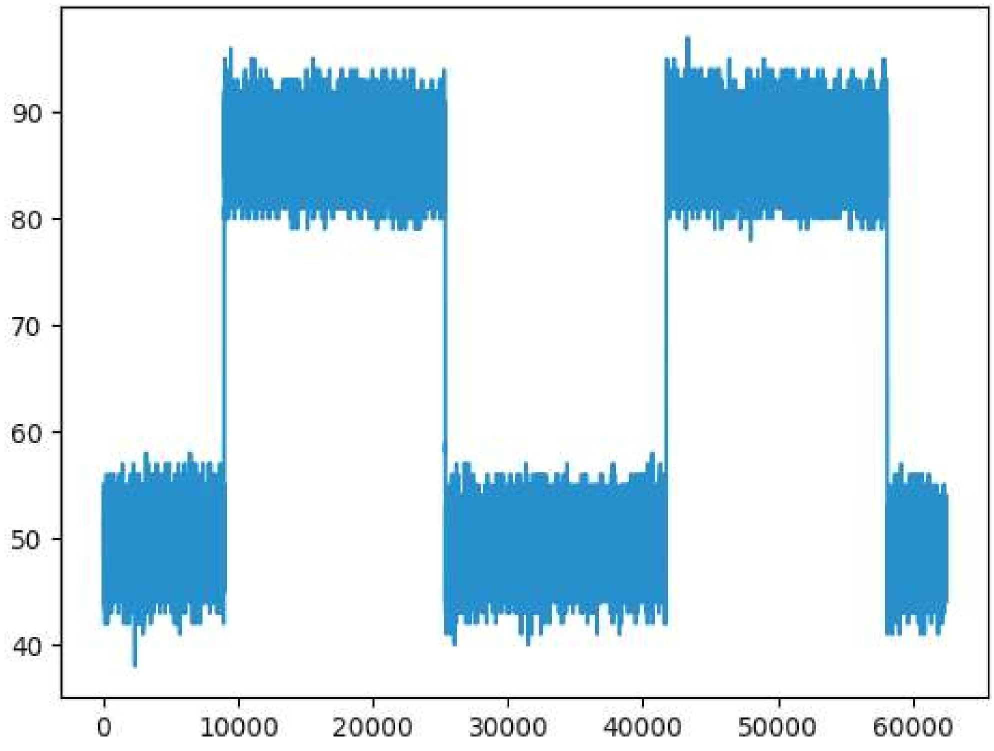
[5]:
```
osci = Agilent_MXO_4104A()
osci.screenshot("Pin_35_LED1",True)
```

```
TCPIPInstrument at TCPIP0::a-mx4104a-10138::inst0::INSTR

Fabricante:  AGILENT TECHNOLOGIES
Modelo:  MSO-X 4104A
Número de serie:  MY53110138
Versión de firmware:  07.40.2021031202
```

[ ]:
```
data1 = osci.collectData(1)
data2 = osci.collectData(2)
fig = plot.figure(figsize=(12, 7.5))
plot.plot(data1)
plot.show()

plot.plot(data2)
plot.show()
```



−90
−95
−100
−105
−110
0
10000
20000
30000
40000
50000
60000



90
80
70
60
50
40
0
10000
20000
30000
40000
50000
60000

E.4 E5071C VNA

[9]:
```
# Libraries and modules needed to connect/work with the VNA:
import pyvisa as visa
import numpy as np
from struct import unpack
import pylab
import time
from matplotlib import pyplot as plot
```

[10]:
```
# Call the pip commando to show the version of all the third-party␣
 ↪libraries used.
#!pip show pyvisa numpy pylab-sdk matplotlib
```

[11]:
```
run Libreria_VNA_E5071C.py
```

[12]:
```
vna = Agilent_ENA_E5071C()
```

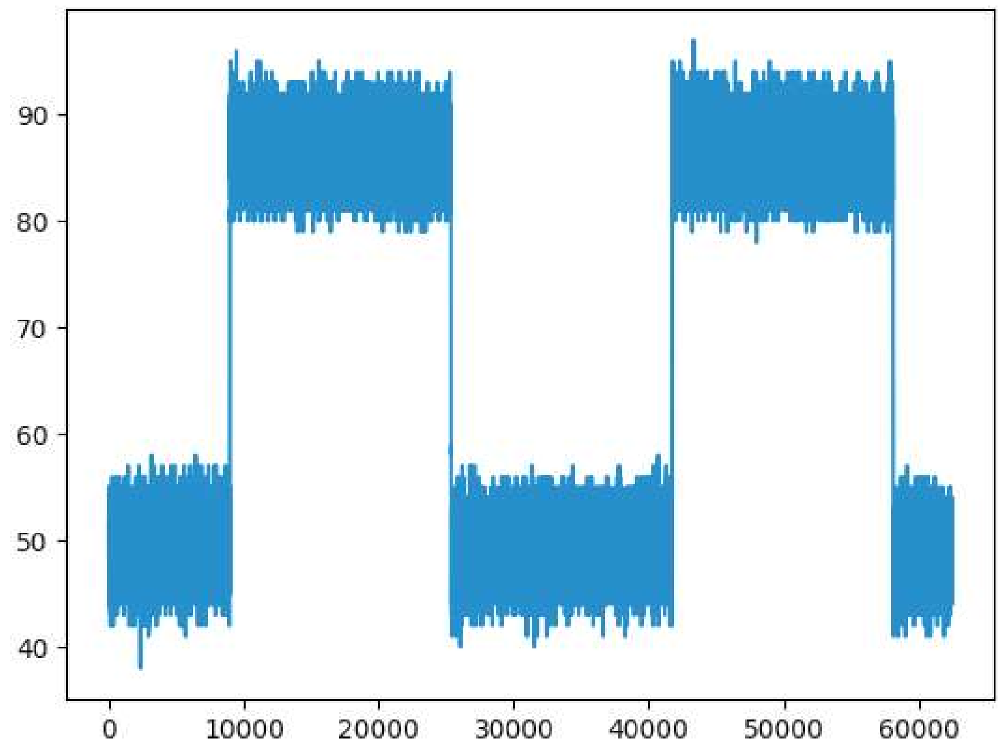
```
TCPIPInstrument at TCPIP0::192.168.1.12::inst0::INSTR

Fabricante:  Agilent Technologies
Modelo:  E5071C
Número de serie:  MY46104277
Versión de firmware:  A.11.36
```

[14]:
```
vna.load('D:/STATE06.STA') #Load state required
```

[ ]:
```
NTraces = 4
vna.numberActiveTraces(NTraces, False) #Set the number of active traces.
Traces_names = ['S11', 'S21', 'S12', 'S22']
# Set the parameter for each trace:

for i in range(0, NTraces):
    vna.setActiveTrace(i+1)
    vna.measParameter(Traces_names[i], False)
"""
vna.setActiveTrace(1)
vna.measParameter(Traces[0], False)
vna.setActiveTrace(2)
vna.measParameter(Trace_2, False)
vna.setActiveTrace(3)
vna.measParameter(Trace_3, False)

"""
```

```
filename = 'Voltmeter_Full_Bandwidth'
title = 'RF Voltmeter S Parameters'
yLabel = 'S Parameters (dB)'
Bandwith = False
Marker = 1
Threshold = 3
annotation = False
vna.plotData(title, yLabel, Bandwith, Marker, Threshold, Traces_names,␣
 ↪NTraces, filename, annotation) #Retrieve the traces data and show in␣
 ↪this notebook
```

```
:CALC1:TRAC1:DATA:FDAT?
:CALC1:TRAC2:DATA:FDAT?
:CALC1:TRAC3:DATA:FDAT?
:CALC1:TRAC4:DATA:FDAT?
```



```
[ ]:
```

E.5 2022D Signal Generator

Ajeno a la biblioteca, son una colección de funciones y pruebas, que ayudan a entender el comportamiento de los dispositivos, en específico del mediador. Ver biblioteca más abajo.

```
[1]: import serial
     import time

     # USAR EL PUERTO EN FUNCIÓN DEL S.O.:

     port_USB2GPIB = 'COM3'   # En Windows = COMX, siendo X={1, 2, ...}
     # port_USB2GPIB = '/dev/ttyUSB0'  # En Linux: /dev/ttyUSBX, siendo X =␣
      ↪{0, 1, ...}
     timeout = 0.5  # Timeout global, segundos

     s = serial.Serial(port_USB2GPIB,         # Puerto
                       115200,                # Baudios
                       serial.EIGHTBITS,      # Payload de 8 bits
                       serial.PARITY_NONE,    # Sin paridad
                       serial.STOPBITS_ONE,   # 1 bit de parada
                       timeout=timeout,       # tiempo de dropout
                       rtscts=False)          # Señales RTS->CTS
```

```
[5]: s.reset_input_buffer()
     s.write(b'++ver' + b'\n')
```

```
[5]: 6
```

```
[6]: bytes_Available=s.in_waiting
     print("Bytes disponibles=" + str(bytes_Available))
```

```
Bytes disponibles=50
```

```
[7]: out=s.read(100)
     type(out)
```

```
[7]: bytes
```

```
[8]: print("Leído=" + str(out))
```

```
Leído=b'ARDUINO GPIB firmware by E. Girlando Version 6.1\r\n'
```

```
[9]: out.split()
```

```
[9]: [b'ARDUINO',
      b'GPIB',
 b'firmware',
 b'by',
 b'E.',
 b'Girlando',
 b'Version',
 b'6.1']
```

[10]:
```
ID_Mediator=out.split()[2]
print("Identify = " + str(ID_Mediator))
```

```
Identify = b'firmware'
```

[11]:
```
length_string_read = 100  # Cte: longitud de caracteres a leer del buffer.

def Identify_USB2GPIB(verbose=True):   # Prof. Andrés Roldán
    """
    Envío de comando para que identificar al intermediario
    Read ++ver
    """

    s.reset_input_buffer()
    s.write(b'++ver' + b'\r\n')
    out = s.read(length_string_read)

    if verbose:
        print("Leído=" + str(out))
    return(out)


def USB2GPIB_AutoResponse_Set(auto_mode=0, verbose=True):   # Prof.␣
↪Andrés Roldán
    """
    Envío de comando para establecer la respuesta automática del mediador␣
↪y no
    tener que hacer un ++read por cada comando enviado.
    Send ++auto <param>, por defecto auto_mode=0 (0: automático, 1: no␣
↪automático)
    """

    #Enviamos comando para que se identifique
    s.reset_input_buffer()
    command_query = f'++auto {auto_mode} '
    s.write(command_query.encode() + b'\r\n')
    if verbose:
        print("Enviado " + "++auto " + str(auto_mode))


def USB2GPIB_AutoResponse_Get(verbose=True):    # Prof. Andrés Roldán
    """
    Pregunta al mediador por su estado de AutoResponse.
    Send ++auto
    """

    s.reset_input_buffer()
    command_query = '++auto'
    s.write(command_query.encode() + b'\r\n')
    out = s.read(length_string_read)

    if verbose:
        print("Leído=" + str(out))
    return(out)


def USB2GPIB_Slave_Address_Set(address, verbose=True):    # Prof. Andrés␣
 ↪Roldán
    """
    Establece la dirección del esclavo.
    Send ++addr Address
    """

    s.reset_input_buffer()
    command_query = '++addr '
    s.write(command_query.encode() + (str(address)).encode() + b'\r\n')

    if verbose:
        print("Enviado " + "++addr " + str(address))


def USB2GPIB_Slave_Address_Get(verbose=True):    # Prof. Andrés Roldán
    """

    Send ++addr
    """

    s.reset_input_buffer()
    command_query = '++addr'
    s.write(command_query.encode() + b'\r\n')
    out = s.read(length_string_read)

    if verbose:
        print("Leído=" + str(out))
    return(out)
```

[12]:
```
# Identificamos al mediador
Identify_USB2GPIB()
```

```
Leído=b'ARDUINO GPIB firmware by E. Girlando Version 6.1\r\n'
```

[12]:
```
b'ARDUINO GPIB firmware by E. Girlando Version 6.1\r\n'
```

[13]:
```
# Configuramos el modo de no pregunta automático "++AUTO 0"
USB2GPIB_AutoResponse_Set(auto_mode=0)
```

```
Enviado ++auto 0
```

[14]:
```
# Preguntamos por la configuración del modo de pregunta automático␣
 ↪"++AUTO"
USB2GPIB_AutoResponse_Get()
```

```
Leído=b'0\r\n'
```

[14]:
```
b'0\r\n'
```

[15]:
```
# Configuramos la dirección del SLAVE GPIB a 10 en byte pero cada␣
 ↪caracter.
Generator_Marconi_2022D_Address = 2  # Dirección del equipo. Aparece al␣
 ↪encerder la fuente.
USB2GPIB_Slave_Address_Set(Generator_Marconi_2022D_Address, verbose=True)
```

```
Enviado ++addr 2
```

[16]:
```
# Leemos la dirección del SLAVE GPIB
USB2GPIB_Slave_Address_Get(verbose=False)
```

[16]:
```
b'2\r\n'
```

E.5.1 Clase para manejar el generador. Implementación de la biblioteca

[3]:
```
# Repetimos estas sentencias del inicio para tener la configuración a mano

import serial
import time
```

```
port_USB2GPIB = 'COM8'  # En Windows = COMX, siendo X={1, 2, ...}
# port_USB2GPIB = '/dev/ttyUSB1'  # En Linux: /dev/ttyUSBX, siendo X =␣
 ↪{0, 1, ...}
timeout = 0.5  # Timeout global, segundos

s = serial.Serial(port_USB2GPIB,        # Puerto
                  115200,               # Baudios
                  serial.EIGHTBITS,     # Payload de 8 bits
                  serial.PARITY_NONE,   # Sin paridad
                  serial.STOPBITS_ONE,  # 1 bit de parada
                  timeout=timeout,      # tiempo de dropout
                  rtscts=False)         # Señales RTS->CTS
```

E.5.2 USB2GPIB: Comandos para el mediador

Sencilla clase que contiene funciones de configuración para el mediador. Uso de comandos "++". No están implementados todos los comandos disponibles, tan sólo los principales para poder acceder al esclavo GPIB.

[4]:
```
class USB2GPIB:
    """
    Colección de métodos para comunicarse con el mediador USB2GPIB.␣
 ↪Arduino con
    firmware "Arduino UNO as a USB to GPIB adapter / controller" v6.1 de
    Emanuele Girlando.
    """

    def __init__(self, length_string_read=100):
        """
        Constructor.
        :param length_string_read: longitud de caracteres a leer del␣
 ↪buffer. Por
            defecto 100.
        :type length_string_read int
        """
        self.length_string_read = length_string_read

    def _send(self, cmd, verbose=True):
        """
        Envío de mensajes al mediador. Resetea el buffer en cada llamada.
        :param cmd: comando a enviar
        :type cmd str
        """
        s.reset_input_buffer()  # reset del buffer
        s.write(cmd.encode() + b'\r\n')  # comando + caracteres de␣
↪parada, bytes
        if verbose:
            print(f"Sent: {cmd}")

    def _read(self, verbose=True):
        """
        Recepción de mensajes del mediador.
        :returns out: mensaje captado.
        :type out str
        """
        out = s.read(self.length_string_read)  # lee del buffer serial
        if verbose:
            print(f"Read: {str(out)}")
        return out

    def identify(self, verbose=True):
        """
        Identificar al intermediario
        Read ++ver
        """
        cmd = "++ver"
        self._send(cmd, verbose)
        return self._read(verbose)

    def set_auto_response(self, auto_mode=0, verbose=True):
        """
        Envío de comando para establecer la respuesta automática del␣
↪mediador y
        no tener que hacer un ++read por cada comando enviado.
        Send ++auto <auto_mode>
        :param mode: parámetro para el comando, por defecto auto_mode=0
        """
        cmd = f"++auto {auto_mode}"
        self._send(cmd, verbose)

    def get_auto_response(self, verbose=True):
        """
        Pregunta al mediador por su estado de AutoResponse.
        Send ++auto
        """
        cmd = "++auto"
        self._send(cmd, verbose)
        return self._read(verbose)

    def set_slave_address(self, address, verbose=True):
        """
        Establece la dirección del esclavo (el generador de señal).
        Send ++addr Address
        :param address: dirección del equipo esclavo a comunicar por GPIB.
        """
        cmd = f"++addr {str(address)}"
        self._send(cmd, verbose)

    def get_slave_address(self, verbose=True):
        """
        Consulta la dirección del esclavo
        Send ++addr
        """
        cmd = "++addr"
        self._send(cmd, verbose)
        return self._read(verbose)
```

Ejemplo de uso

Inicializamos la clase. Identificamos al dispositivo y establecemos el tipo de respuesta, así como la dirección del esclavo al que queremos enviar órdenes.

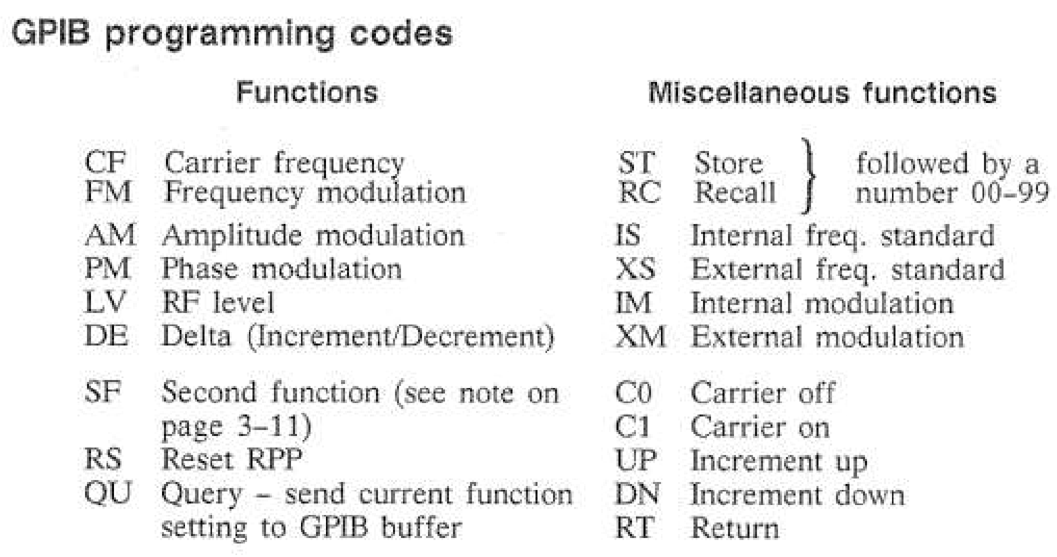
[5]:
```
Generator_Marconi_2022D = USB2GPIB()
Generator_Marconi_2022D.identify()

Generator_Marconi_2022D.set_auto_response(auto_mode=0)
Generator_Marconi_2022D.set_slave_address(address=2)

Generator_Marconi_2022D.get_auto_response()
Generator_Marconi_2022D.get_slave_address()
```

```
Sent: ++ver
Read: b'ARDUINO GPIB firmware by E. Girlando Version 6.1\r\n'
Sent: ++auto 0
Sent: ++addr 2
Sent: ++auto
Read: b'0\r\n'
Sent: ++addr
Read: b'2\r\n'
```

[5]:
```
b'2\r\n'
```

E.5.3 Generator_Marconi_2022D: Comandos para el generador.

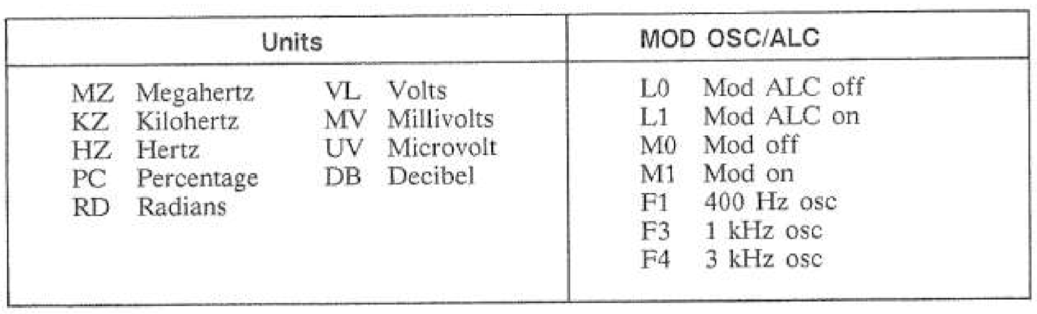
En la siguiente clase se añadirán las funciones necesarias poder enviar los comandos al generador 2022D de Marconi Instruments.



GPIB programming codes

Functions		Miscellaneous functions	
CF	Carrier frequency	ST	Store } followed by a number 00–99
FM	Frequency modulation	RC	Recall } followed by a number 00–99
AM	Amplitude modulation	IS	Internal freq. standard
PM	Phase modulation	XS	External freq. standard
LV	RF level	IM	Internal modulation
DE	Delta (Increment/Decrement)	XM	External modulation
SF	Second function (see note on page 3–11)	C0	Carrier off
		C1	Carrier on
RS	Reset RPP	UP	Increment up
QU	Query – send current function setting to GPIB buffer	DN	Increment down
		RT	Return

Fig.1 - Tabla con las siglas correspondientes a cada función.



Units				MOD OSC/ALC	
MZ	Megahertz	VL	Volts	L0	Mod ALC off
KZ	Kilohertz	MV	Millivolts	L1	Mod ALC on
HZ	Hertz	UV	Microvolt	M0	Mod off
PC	Percentage	DB	Decibel	M1	Mod on
RD	Radians			F1	400 Hz osc
				F3	1 kHz osc
				F4	3 kHz osc

Fig.2 - Tabla con las siglas correspondientes a cada unidad y modo.

[7]:
```
class Marconi_2022D:
    """
    Funciones del generador 2022D de Marconi Instruments

    NOTA: Los métodos de esta clase esperan que la comunicación se␣
 ↪realice por
    serial, siendo necesario un intermediario tipo USB 2 GPIB.
    Para comunicarse con cada equipo conectado se debe especificar␣
 ↪primero la
```

```
    dirección correspondiente al mediador.
    """

    def __init__(self, length_string_read=100):
        """
        Constructor.
        :param length_string_read: longitud de caracteres a leer del␣
↪buffer. Por
            defecto 100.

        Este parámetro afecta a los comandos demasiado extensos como por␣
↪ejemplo
        el envío de muestras de señal.
        :type length_string_read int
        """
        self.length_string_read = length_string_read

    def _send(self, cmd, verbose=True):
        """
        Envío de mensajes al mediador. Resetea el buffer en cada llamada.
        :param cmd: comando a enviar
        :type cmd str
        """
        s.reset_input_buffer()  # reset del buffer
        s.write(cmd.encode() + b'\r\n')  # comando + caracteres de␣
↪parada, en bytes
        if verbose:
            print(f"Sent: {cmd}")

    # GPIB programming codes
    # Funciones para modificar los parámetros de la señal generada:␣
↪frecuencia, modulación...
    # Por lo general, las funciones consisten en dos siglas que indican␣
↪el parámetro que se quiere
    # modificar seguido del valor a fijar y sus unidades si son␣
↪necesarias.
    # En las página 3-20 y 3-21 del manual de operaciones, adjuntado al␣
↪principio del notebook,
    # aparecen unas tablas con todas las siglas necesarias.
    # Para conseguir mayor eficiencia se han insertado dichas tablas en␣
↪la celda anterior.

    def set_carrier_frequency(self, fc, units, verbose=True):
        """
        Con esta función se busca fijar la frecuencia de la portadora,␣
↪indicándole el valor y las unidades.
        IMPORTANTE: Al llamar a la función es necesario escribir las␣
↪unidades entre comillas al ser un string
        """
        cmd = f"CF {fc}" + units
        self._send(cmd, verbose)

    def set_modulation_frequency(self, fm, units, verbose=True):
        """
        Con esta función se busca fijar la frecuencia de modlación,␣
↪indicándole el valor y las unidades.
        IMPORTANTE: Al llamar a la función es necesario escribir las␣
↪unidades entre comillas al ser un string
        """
        cmd = f"FM {fm}" + units
        self._send(cmd, verbose)

    def set_modulation_amplitude(self, Ac, units, verbose=True):
        """
        Con esta función se busca fijar la amplitud de modulación,␣
↪indicándole el valor y las unidades.
        IMPORTANTE: Al llamar a la función es necesario escribir las␣
↪unidades entre comillas al ser un string
        """
        cmd = f"AM {Ac}" + units
        self._send(cmd, verbose)

    def set_modulation_phase(self, Pm, units, verbose=True):
        """
        Con esta función se busca fijar la fase de modulación,␣
↪indicándole el valor y las unidades.
        IMPORTANTE: Al llamar a la función es necesario escribir las␣
↪unidades entre comillas al ser un string
        """
        cmd = f"PM {Pm}" + units
        self._send(cmd, verbose)

    def set_RF_level(self, Px, units, verbose=True):
        """
        Con esta función se busca fijar el nivel de potencia de la señal␣
↪de salida, indicándole el valor y las unidades.
```

```
        IMPORTANTE: Al llamar a la función es necesario escribir las␣
↪unidades entre comillas al ser un string
        """
        cmd = f"LV {Px}" + units
        self._send(cmd, verbose)

    def set_delta(self, param, value, units, verbose=True):
        """
        Con esta función se busca cambiar el valor del incremento o␣
↪decremento de un cierto parámetro.
        IMPORTANTE: Al llamar a la función es necesario escribir las␣
↪unidades y el parámetro entre comillas al ser un string
        """
        cmd = f"DE" + param + f"{value}" + units
        self._send(cmd, verbose)

    def set_carrier_off(self, verbose=True):
        """
        Con esta función se busca desactivar la portadora de nuestra␣
↪señal de salida
        """
        cmd = "C0"
        self._send(cmd, verbose)

    def set_carrier_on(self, verbose=True):
        """
        Con esta función se busca activar la portadora de nuestra señal␣
↪de salida
        """
        cmd = "C1"
        self._send(cmd, verbose)

    def set_increment_up(self, verbose=True):
        """
        Con esta función se busca aumentar el valor anteriormente indicado
        """
        cmd = "UP"
        self._send(cmd, verbose)

    def set_increment_down(self,verbose=True):
        """
        Con esta función se busca disminuir el valor anteriormente␣
↪indicado
        """
        cmd = "DN"
        self._send(cmd, verbose)

    # Funciones para cambiar la modulación

    def set_mod_alc_off(self,verbose=True):
        """
        Con esta función se busca desactivar la modulación ALC
        """
        cmd = "L0"
        self._send(cmd, verbose)

    def set_mod_alc_on(self,verbose=True):
        """
        Con esta función se busca activar la modulación ALC
        """
        cmd = "L1"
        self._send(cmd, verbose)

    def set_mod_off(self,verbose=True):
        """
        Con esta función se busca desactivar la modulación
        """
        cmd = "M0"
        self._send(cmd, verbose)

    def set_mod_on(self,verbose=True):
        """
        Con esta función se busca activar la modulación
        """
        cmd = "M1"
        self._send(cmd, verbose)

    def set_oscillator_400_Hz(self,verbose=True):
        """
        Con esta función se busca fijar el oscilador a 400 Hz
        """
        cmd = "F1"
        self._send(cmd, verbose)

    def set_increment_down(self,verbose=True):
        """
        Con esta función se busca fijar el oscilador a 1 kHz
        """
        cmd = "F3"
        self._send(cmd, verbose)

    def set_increment_down(self,verbose=True):
        """
        Con esta función se busca fijar el oscilador a 3 kHz
        """
        cmd = "F6"
        self._send(cmd, verbose)

    def set_external_freq_standard(self,verbose=True):
        """
        Con esta función se selecciona la señal de 10 MHz externa
        """
        cmd = "XS"
        self._send(cmd, verbose)

    def set_internal_freq_standard(self,verbose=True):
        """
        Con esta función se selecciona la señal de 10 MHz interna
        """
        cmd = "IS"
        self._send(cmd, verbose)
```

Ejemplo de uso.

Inicializamos la clase y utilizamos las funciones definidas anteriormente. El ejemplo consistirá en fijar la frecuencia a 175 MHz, la potencia de salida a -3.2 dBm y seleccionamos referencia externa.

[12]:
```
Generador = Marconi_2022D()
Generador.set_carrier_frequency(175,"MZ")
Generador.set_RF_level(-3.2, "DB")
Generador.set_external_freq_standard()
```

```
Sent: CF 175MZ
Sent: LV -1000DB
Sent: XS
```

Appendix F

EPICS Installation

This appendix will detail the steps to follow for the complete installation of all the necessary software to use EPICS and its add-ons.

Firstly, we need to install the EPICS base. For this, we must have *make*, *c++ compiler*, and *libreadline* installed. As usual before each installation, we run the typical Ubuntu command to update the system and ensure that all packages are up to date, and then install the mentioned dependencies.

- sudo apt update
- sudo apt install make
- sudo apt install g++
- sudo apt-get install libreadline-dev

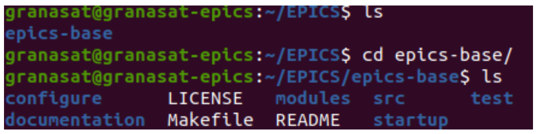
Once they are installed, we can proceed to download the EPICS base. Before that, we will create a folder to host the program files, within our $ HOME folder.

- mkdir EPICS
- cd EPICS

We download the files from GitHub using the following command:

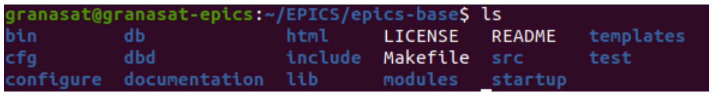
- git clone –recursive https://github.com/epics-base/epics-base.git

Inside our *EPICS* directory, a folder called *epics-base* will be created, which will contain the following subdirectories.



```
granasat@granasat-epics:~/EPICS$ ls
epics-base
granasat@granasat-epics:~/EPICS$ cd epics-base/
granasat@granasat-epics:~/EPICS/epics-base$ ls
configure      LICENSE   modules  src      test
documentation  Makefile  README   startup
```

Now the command *make* is run.

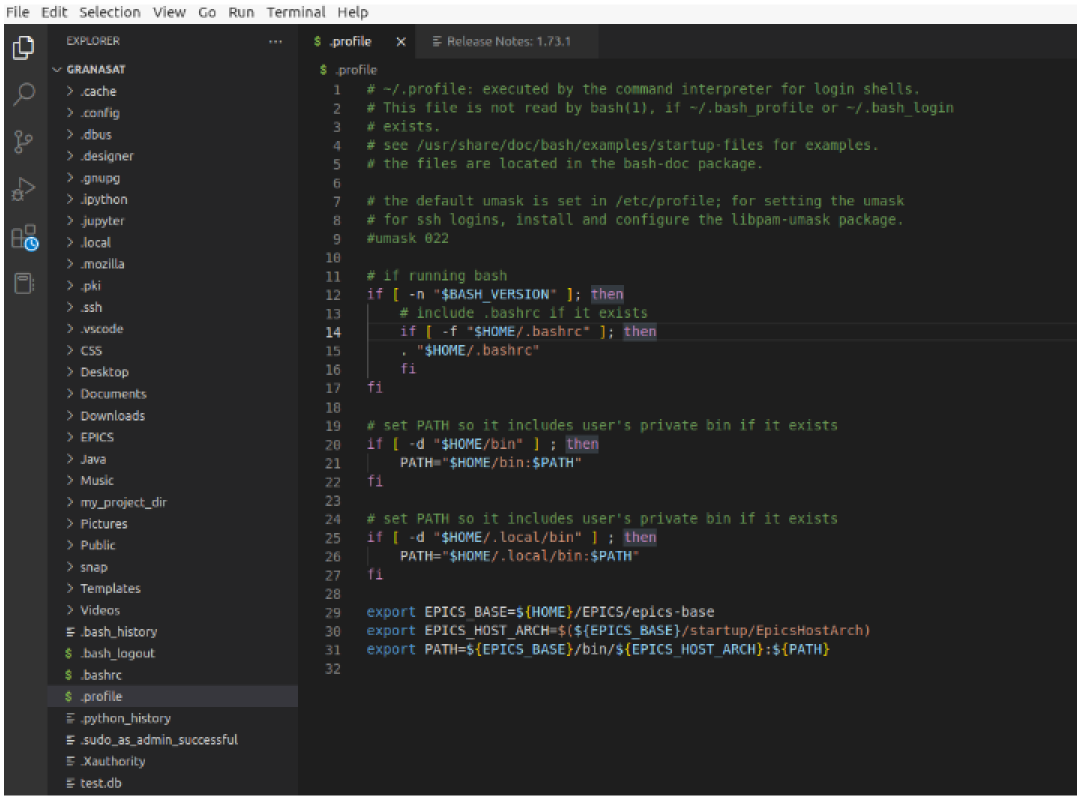


```
granasat@granasat-epics:~/EPICS/epics-base$ ls
bin        db             html     LICENSE   README   templates
cfg        dbd            include  Makefile  src      test
configure  documentation  lib      modules   startup
```

The following directories and files will appear:

```
export EPICS_BASE=${HOME}/EPICS/epics-base
export EPICS_HOST_ARCH=$(${EPICS_BASE}/startup/EpicsHostArch)
export PATH=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}:${PATH}
```

When finished, you need to indicate the following paths in the *$HOME/.profile* or *HOME/.bashrc* file, and the installation of EPICS BASE would be completed.

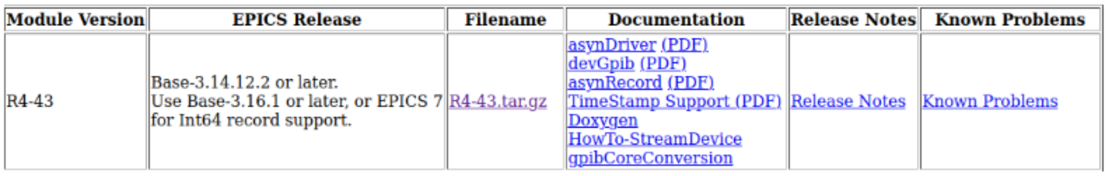


```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
    . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi

export EPICS_BASE=${HOME}/EPICS/epics-base
export EPICS_HOST_ARCH=$(${EPICS_BASE}/startup/EpicsHostArch)
export PATH=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}:${PATH}
```

F.1 StreamDevice

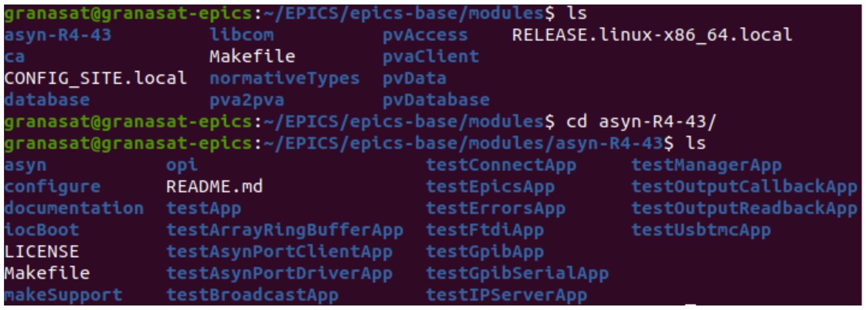
To communicate with devices, we need to use the StreamDevice module, which needs to be installed. Additionally, as it relies on AsynDriver, it is necessary to install this module first.

Download the latest version of AsynDriver, in this case, R4-43, from this link.



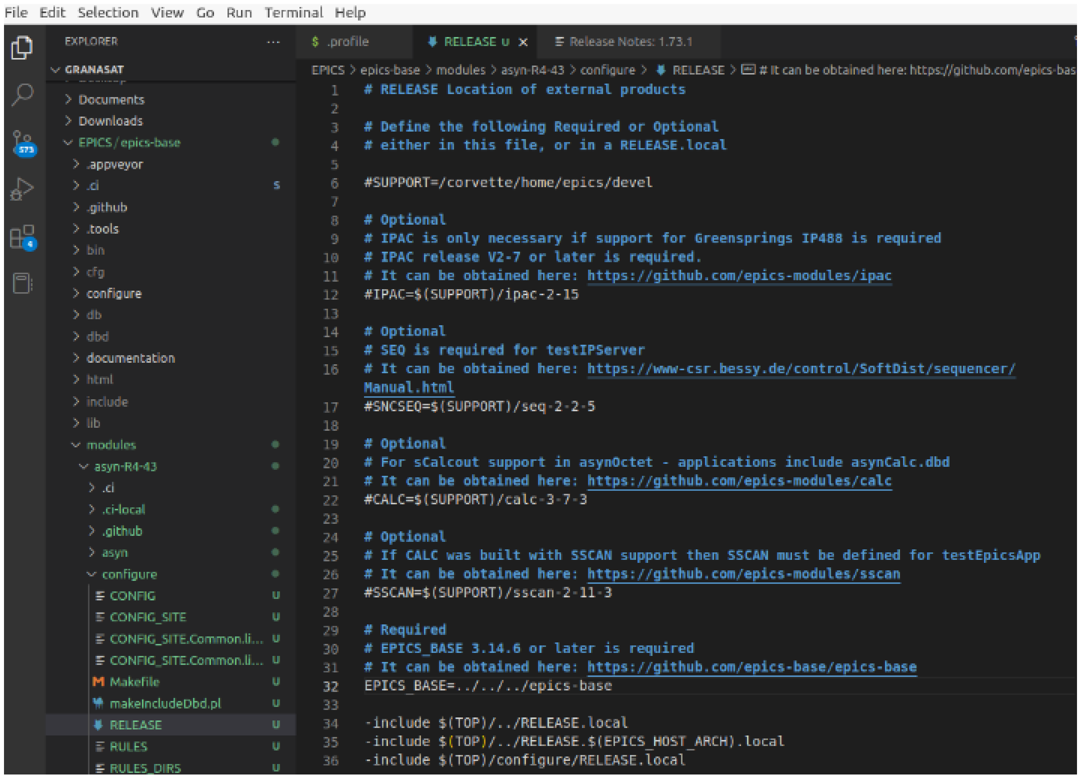
Module Version	EPICS Release	Filename	Documentation	Release Notes	Known Problems
R4-43	Base-3.14.12.2 or later. Use Base-3.16.1 or later, or EPICS 7 for Int64 record support.	R4-43.tar.gz	asynDriver (PDF) devGpib (PDF) asynRecord (PDF) TimeStamp Support (PDF) Doxygen HowTo-StreamDevice gpibCoreConversion	Release Notes	Known Problems

Once the file is downloaded, move it to the directory where you want to store this module. In this case, we will use the already created modules folder within *EPICS BASE*.



```
granasat@granasat-epics:~/EPICS/epics-base/modules$ ls
asyn-R4-43         libcom          pvAccess     RELEASE.linux-x86_64.local
ca                 Makefile        pvaClient
CONFIG_SITE.local  normativeTypes  pvData
database           pva2pva         pvDatabase
granasat@granasat-epics:~/EPICS/epics-base/modules$ cd asyn-R4-43/
granasat@granasat-epics:~/EPICS/epics-base/modules/asyn-R4-43$ ls
asyn           opi                     testConnectApp    testManagerApp
configure      README.md               testEpicsApp      testOutputCallbackApp
documentation  testApp                 testErrorsApp     testOutputReadbackApp
iocBoot        testArrayRingBufferApp  testFtdiApp       testUsbtmcApp
LICENSE        testAsynPortClientApp   testGpibApp
Makefile       testAsynPortDriverApp   testGpibSerialApp
makeSupport    testBroadcastApp        testIPServerApp
```

Next, you need to specify the path to *EPICS BASE* in the *RELEASE* file located in the configure folder.

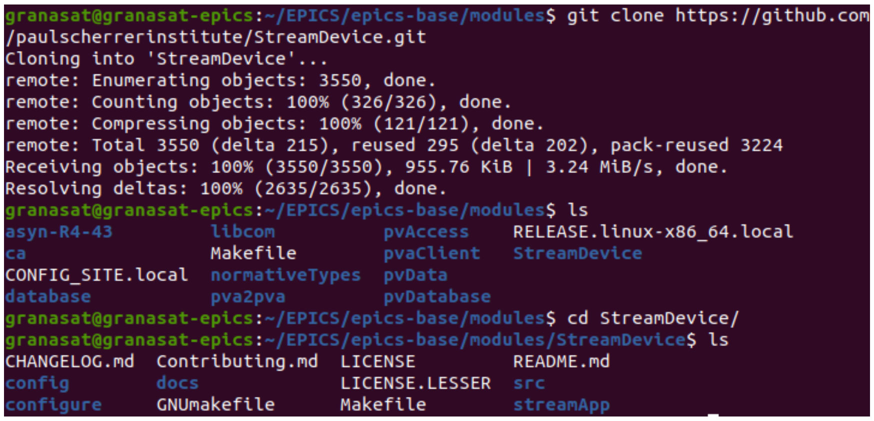


```
# RELEASE Location of external products

# Define the following Required or Optional
# either in this file, or in a RELEASE.local

#SUPPORT=/corvette/home/epics/devel

# Optional
# IPAC is only necessary if support for Greensprings IP488 is required
# IPAC release V2-7 or later is required.
# It can be obtained here: https://github.com/epics-modules/ipac
#IPAC=$(SUPPORT)/ipac-2-15

# Optional
# SEQ is required for testIPServer
# It can be obtained here: https://www-csr.bessy.de/control/SoftDist/sequencer/
Manual.html
#SNCSEQ=$(SUPPORT)/seq-2-2-5

# Optional
# For sCalcout support in asynOctet - applications include asynCalc.dbd
# It can be obtained here: https://github.com/epics-modules/calc
#CALC=$(SUPPORT)/calc-3-7-3

# Optional
# If CALC was built with SSCAN support then SSCAN must be defined for testEpicsApp
# It can be obtained here: https://github.com/epics-modules/sscan
#SSCAN=$(SUPPORT)/sscan-2-11-3

# Required
# EPICS_BASE 3.14.6 or later is required
# It can be obtained here: https://github.com/epics-base/epics-base
EPICS_BASE=../../../epics-base

-include $(TOP)/../RELEASE.local
-include $(TOP)/../RELEASE.$(EPICS_HOST_ARCH).local
-include $(TOP)/configure/RELEASE.local
```

Finally, execute the *make* command in the main directory of your module, and AsynDriver will be installed.

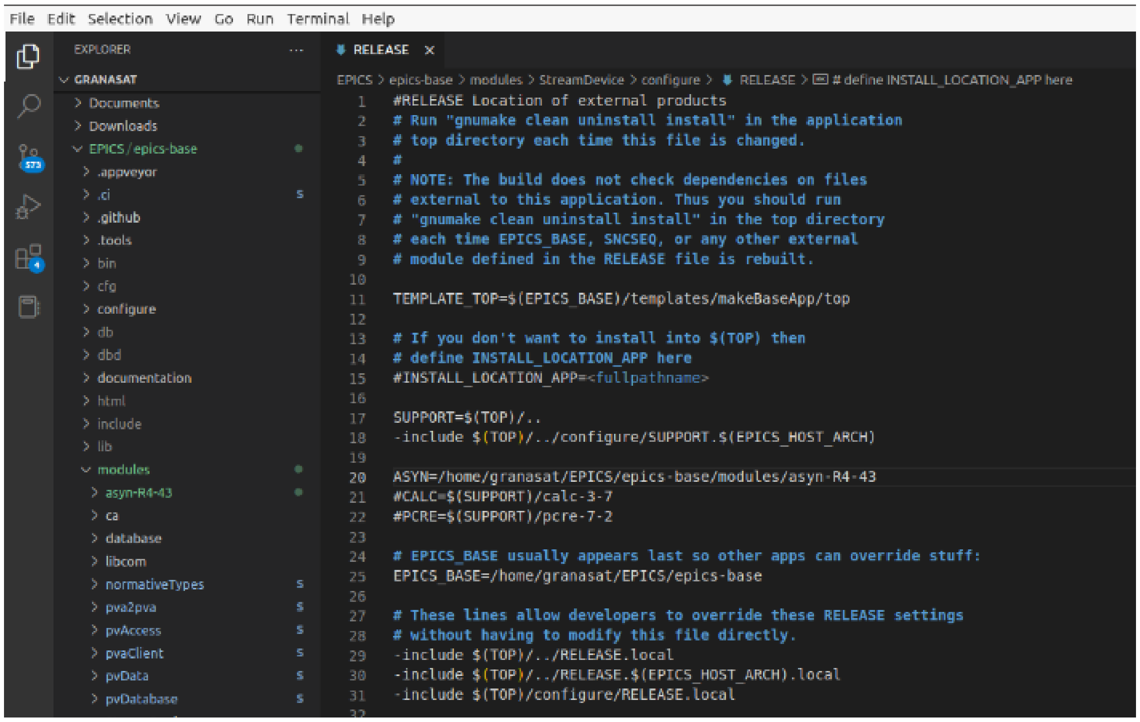
Once AsynDriver is installed, we can proceed with the installation of StreamDevice. To download the files, execute the following command within the modules directory:

- git clone `https://github.com/paulscherrerinstitute/StreamDevice.git`



```
granasat@granasat-epics:~/EPICS/epics-base/modules$ git clone https://github.com
/paulscherrerinstitute/StreamDevice.git
Cloning into 'StreamDevice'...
remote: Enumerating objects: 3550, done.
remote: Counting objects: 100% (326/326), done.
remote: Compressing objects: 100% (121/121), done.
remote: Total 3550 (delta 215), reused 295 (delta 202), pack-reused 3224
Receiving objects: 100% (3550/3550), 955.76 KiB | 3.24 MiB/s, done.
Resolving deltas: 100% (2635/2635), done.
granasat@granasat-epics:~/EPICS/epics-base/modules$ ls
asyn-R4-43          libcom          pvAccess     RELEASE.linux-x86_64.local
ca                  Makefile        pvaClient    StreamDevice
CONFIG_SITE.local   normativeTypes  pvData
database            pva2pva         pvDatabase
granasat@granasat-epics:~/EPICS/epics-base/modules$ cd StreamDevice/
granasat@granasat-epics:~/EPICS/epics-base/modules/StreamDevice$ ls
CHANGELOG.md  Contributing.md  LICENSE         README.md
config        docs             LICENSE.LESSER  src
configure     GNUmakefile      Makefile        streamApp
```

After the StreamDevice folder is created, modify the path to *EPICS BASE* and AsynDriver in the *configure/RELEASE* file and comment out the paths for the CALC and PCRE variables.



```
#RELEASE Location of external products
# Run "gnumake clean uninstall install" in the application
# top directory each time this file is changed.
#
# NOTE: The build does not check dependencies on files
# external to this application. Thus you should run
# "gnumake clean uninstall install" in the top directory
# each time EPICS_BASE, SNCSEQ, or any other external
# module defined in the RELEASE file is rebuilt.

TEMPLATE_TOP=$(EPICS_BASE)/templates/makeBaseApp/top

# If you don't want to install into $(TOP) then
# define INSTALL_LOCATION_APP here
#INSTALL_LOCATION_APP=<fullpathname>

SUPPORT=$(TOP)/..
-include $(TOP)/../configure/SUPPORT.$(EPICS_HOST_ARCH)

ASYN=/home/granasat/EPICS/epics-base/modules/asyn-R4-43
#CALC=$(SUPPORT)/calc-3-7
#PCRE=$(SUPPORT)/pcre-7-2

# EPICS_BASE usually appears last so other apps can override stuff:
EPICS_BASE=/home/granasat/EPICS/epics-base

# These lines allow developers to override these RELEASE settings
# without having to modify this file directly.
-include $(TOP)/../RELEASE.local
-include $(TOP)/../RELEASE.$(EPICS_HOST_ARCH).local
-include $(TOP)/configure/RELEASE.local
```