



# GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO  
RFSoc For RF Environment Monitoring

Autor: Jaime Mohedano Aragón  
Director: Alan Pisano (Boston University)

Madrid



Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título  
RFSoc for RF Environment Monitoring

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021-2022 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es  
plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.: Jaime Mohedano Aragón

Fecha: 29/06/2022



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Alan Pisano (Boston University) Fecha: 29/06/2022







# GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO  
RFSoc for RF Environment Monitoring

Autor: Jaime Mohedano Aragón  
Director: Alan Pisano (Boston University)

Madrid



# Acknowledgements

Firstly, I would like to express my gratitude to the clients of this project, John Swoboda and Sharanya Srinivas, from the MIT Haystack Observatory, for their consistent support and assistance during the development of this project. I wish to express my thanks to Professor Joshua Semeter, from the ECE department at Boston University, for providing the equipment and laboratory resources needed for the project, and also, to Professor Alan Pisano, the supervisor of this project, who has been providing feedback and guidance during the development of the project.

I am also grateful to my colleagues and friends, with whom I have shared incredible moments, challenges, and experiences during the four years of my bachelor's degree.

Finally, special mention to my family, I am thankful to them for helping me achieve everything I am and have today, this journey would not have been possible without them.





# **SISTEMAS DE RADIOFRECUENCIA EN UN CHIP PARA MONITORIZACIÓN DEL ENTORNO DE RADIOFRECUENCIA**

**Autor: Mohedano Aragón, Jaime**

Director: Pisano, Alan

Entidad Colaboradora: MIT Haystack Observatory

## **RESUMEN DEL PROYECTO**

En este proyecto se ha desarrollado una aplicación web interactiva enfocada a la monitorización del espectro de radio frecuencia. La aplicación es capaz de reproducir datos en formato Digital RF; y de interactuar con una tarjeta RFSoc, incluyendo la transmisión de datos en vivo y la descarga de datos de la tarjeta en dicho formato.

**Palabras clave:** Radio definida por software, procesamiento de señales, monitorización espectral, RFSoc, Digital RF

### **1. Introducción**

El espectro de radiofrecuencia (RF) está cada vez más congestionado, lo cual dificulta que los investigadores de las comunidades geoespacial y de la radioastronomía obtengan las medidas de alta fidelidad necesarias para su trabajo. Para superar la congestión, es necesario implementar técnicas de mitigación de interferencias en radiofrecuencia, las cuales requieren del uso de herramientas de monitorización del espectro.

### **2. Definición del proyecto**

El proyecto consiste en una aplicación web que puede ser usada junto con la tarjeta RFSoc de Xilinx para un conjunto de tareas de monitorización del espectro de radiofrecuencia, incluyendo la reproducción de datos pre-guardados en formato Digital RF, transmisión de datos en vivo desde la placa y la descarga de datos en formato Digital RF para su posterior reproducción. La aplicación muestra el espectro a través de una representación gráfica del mismo y de un espectrograma, permitiendo que el usuario interactúe con los gráficos.

### **3. Descripción del modelo/sistema/herramienta**

La aplicación se compone de cuatro componentes principales:

- Una aplicación web desarrollada en Dash (interfaz de usuario): interfaz a través de la cual el usuario interactúa con la aplicación.
- Un servidor de Redis: canal de comunicación entre la interfaz de usuario con la tarjeta, así como con el backend.

- Un script escrito en Python en el backend: código responsable de procesar las peticiones relacionadas con Digital RF.
- Scripts corriendo en la tarjeta: código responsable de añadir los datos que captura la tarjeta al servidor (stream) de Redis.

La Figure 1- Diagrama de bloques del sistema muestra el diagrama de bloques de los componentes de la aplicación y como interactúan entre ellos:

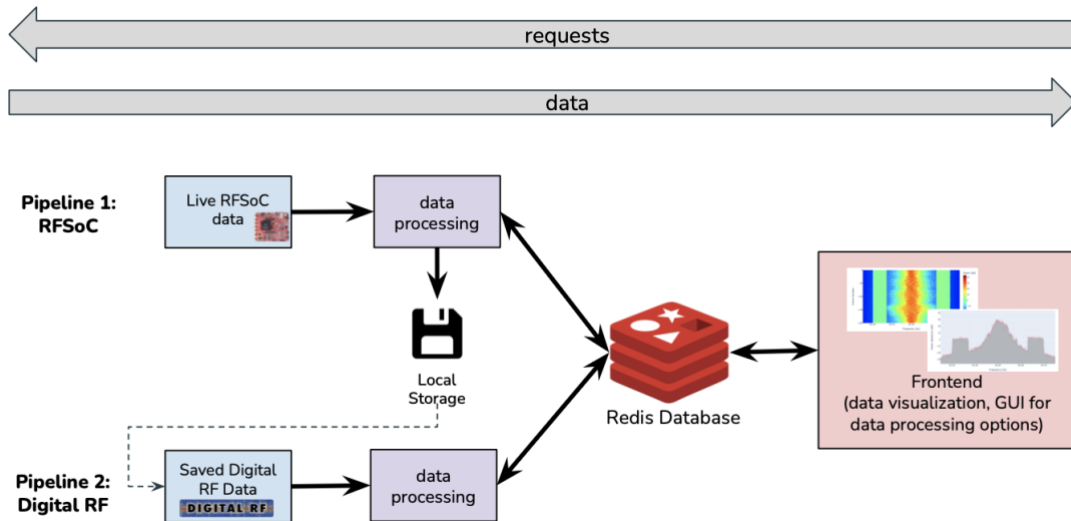


Figure 1- Diagrama de bloques del sistema

#### 4. Resultados

El producto final es una pantalla principal web con gráficos interactivos (espectro y espectrograma) que muestra datos de frecuencia. El usuario puede interactuar con los gráficos cambiando los límites de los ejes, marcando los puntos máximo y mínimo de los datos y cambiando el esquema de color del espectrograma. La aplicación proporciona compatibilidad con el formato Digital RF, el cual está presente en dos de las principales funcionalidades: reproducción de datos pre-guardados y descarga de datos en directo. La otra funcionalidad principal permite al usuario transmitir datos en directo directamente desde la tarjeta.

En la Figure 2 - Interfaz de usuario de la aplicación web se muestra la interfaz de la aplicación web mientras se están transmitiendo en directo datos desde la tarjeta.

## Spectrum Monitoring Dashboard

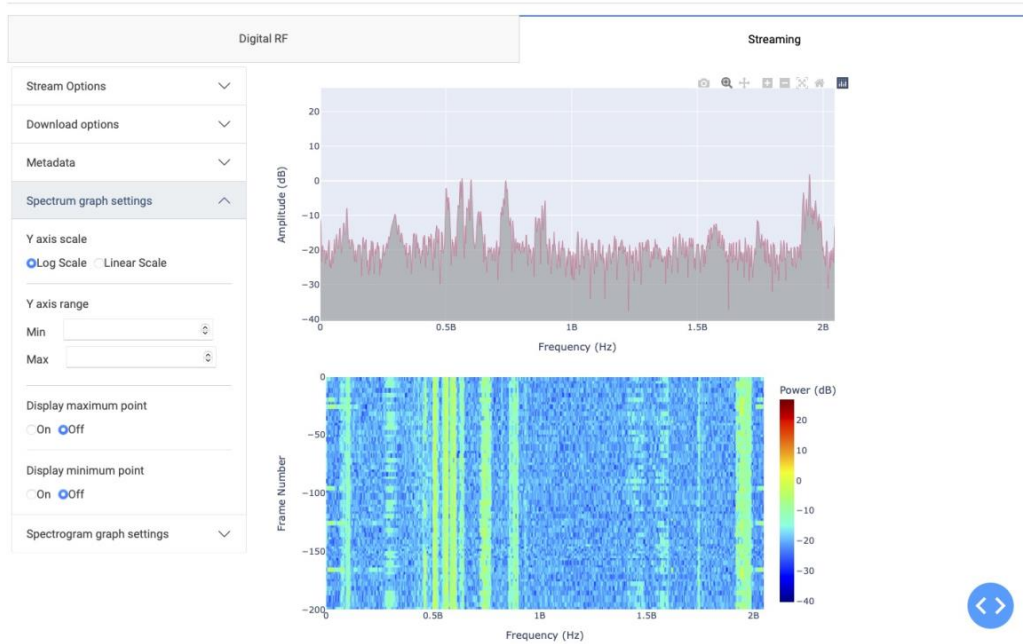


Figure 2 - Interfaz de usuario de la aplicación web

## 5. Conclusiones

La aplicación web desarrollada es una base práctica que se puede ampliar según sea necesario para atender a futuras necesidades del ámbito de la investigación. Ofrece herramientas bien documentadas y fáciles de usar para la monitorización del espectro de radiofrecuencia. El proyecto ha sido de carácter exploratorio y se ha ido adaptando a las peticiones/necesidades del cliente.

Se han propuesto algunas sugerencias o ideas para ampliar las capacidades de la aplicación web y para mejorar la experiencia del usuario. Aun así, las funcionalidades implementadas constituyen una infraestructura sólida, a partir de la cual se pueden implementar rápidamente las ideas propuestas.

# **RFSOC FOR RF ENVIRONMENT MONITORING**

**Author: Mohedano Aragón, Jaime**

Supervisor: Pisano, Alan

Collaborating Entity: MIT Haystack Observatory

## **ABSTRACT**

In this project, an interactive web application meant for radio frequency (RF) spectrum monitoring has been developed. The application is capable of playing back Digital RF data, and interacting with an RFSoc board, the latter includes live streaming data as well as downloading data from the board in the Digital RF format.

**Keywords:** Software-Defined Radio, signal processing, spectrum monitoring, RFSoc, Digital RF

### **1. Introduction**

The radio frequency (RF) spectrum is becoming increasingly congested, which makes it difficult for researchers in the geospace and radio astronomy communities to obtain the high-fidelity measurements necessary for their work. To overcome congestion, it is necessary to deploy RF interference mitigation techniques, which require the use of RF spectrum monitoring tools.

### **2. Project definition**

The project consists of a web application that can be used in conjunction with the Xilinx RFSoc board for a variety of RF spectrum monitoring tasks, including playing back pre-recorded Digital RF data, live streaming data from the board, and downloading data in the Digital RF format to be played back later. The application displays the radiofrequency spectrum through a spectrum plot and a spectrogram, allowing the user to interact with the graphs.

### **3. Model/system/tool description**

There are four main components to the application:

- A Dash-based web application (frontend): interface through which the user interacts with the application.
- A Redis server: channel of communication between the Dash front end with the board, as well as with the Python back end.
- A Python backend script: code responsible for processing Digital RF requests.
- Scripts running on the RFSoc: code responsible for adding the data that the board captures to the Redis stream.

Figure 3 - Block diagram of the system shows, through a block diagram, the components of the web application with their interactions:

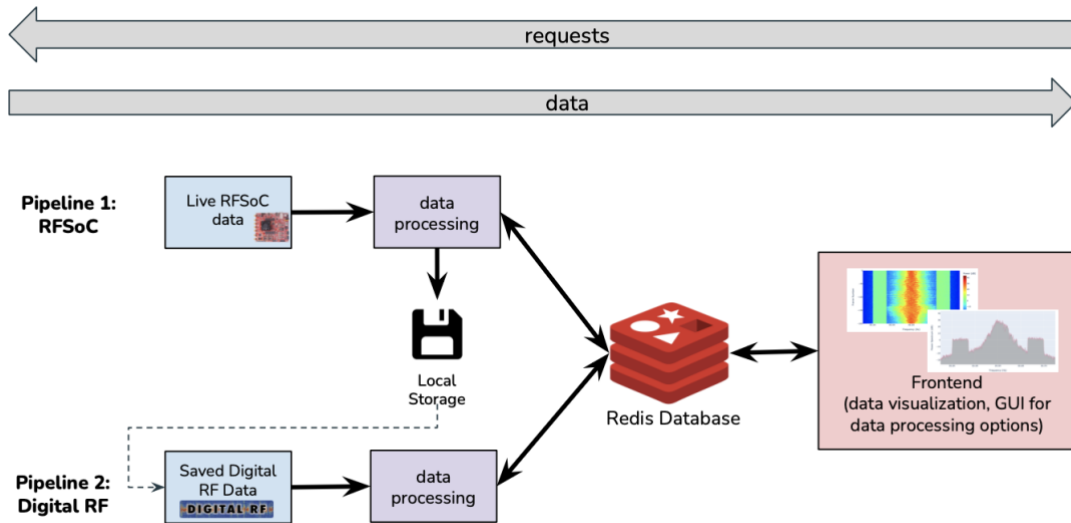


Figure 3 - Block diagram of the system

#### 4. Results

The deliverable is a web-based dashboard with colorful interactive plots (spectrum and spectrogram graphs) that displays frequency data. The user can interact with the graphs by changing the axis boundaries, tracking minimum and maximum data points, and changing the spectrogram color scheme. The application provides compatibility with the Digital RF format, which is present in two of the main functionalities: playing back pre-recorded data and downloading live data. The other main functionality allows the user to live stream data directly from the board.

The figure below, Figure 4 - User interface of the web application, shows the interface of the web application while streaming live data captured by the RFSoc board:

## Spectrum Monitoring Dashboard

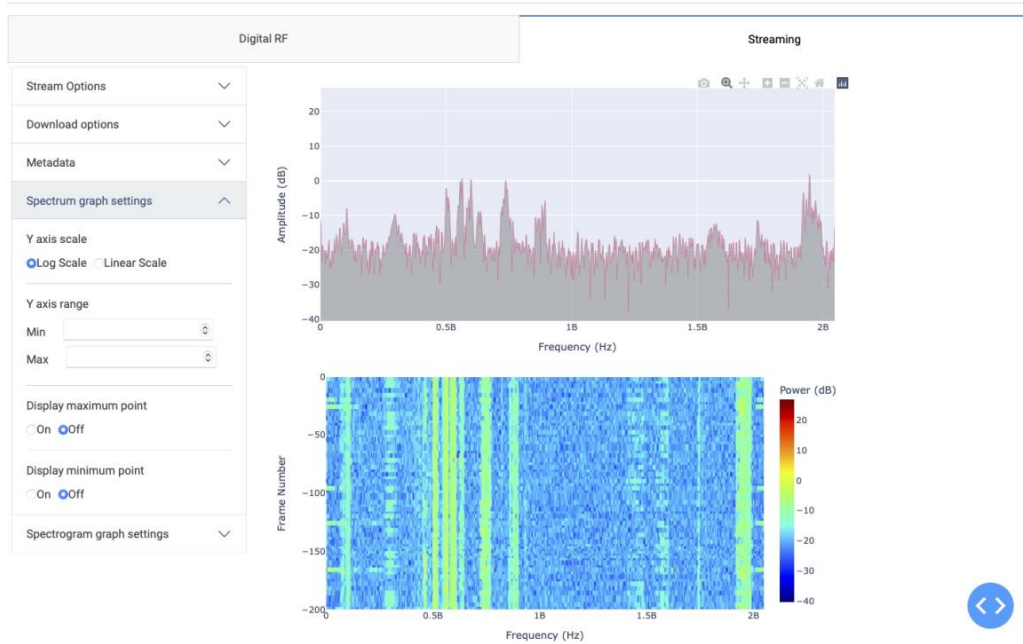


Figure 4 - User interface of the web application

## 5. Conclusions

The web application created is a practical base that can be extended as needed to serve future research needs. It offers well-documented, easy-to-use tools for RF environment monitoring. The project has always been exploratory in nature and has adapted to the requests/needs of the client.

Some suggestions or ideas have been proposed to broaden the capabilities of the web application and to improve the user experience. Even so, the features implemented work as a solid framework from which these additional suggestions could quickly be implemented.

## *Table of contents*

<b>1. Introduction .....</b>	<b>6</b>
<b>2. Description of technologies .....</b>	<b>8</b>
2.1 Xilinx RFSoc 2x2 board.....	8
2.2 Dash.....	9
2.3 Redis.....	10
2.4 Digital RF.....	11
2.5 PYNQ framework .....	11
2.6 StrathSDR .....	12
2.7 JSON.....	12
2.8 JupyterLab.....	13
<b>3 State of the art.....</b>	<b>14</b>
3.1 Project context.....	14
3.2 Competing technologies.....	15
3.2.1 <i>OpenWebRX</i> .....	15
3.2.2 <i>KiwiSDR</i> .....	16
3.2.3 <i>SDRSharp</i> .....	17
3.2.4 <i>GNU Radio</i> .....	18
3.2.5 <i>StrathSDR</i> .....	19
<b>4 Description of the work .....</b>	<b>21</b>
4.1 Justification .....	21
4.2 Objectives.....	21
4.3 Working methodology .....	22
4.4 Budget estimate.....	24
<b>5 System developed .....</b>	<b>26</b>
5.1 Overview block diagram.....	26
5.2 Hardware of the project.....	27
5.2.1 <i>Board setup</i> .....	27
5.2.2 <i>Power requirements</i> .....	29
5.3 Installation and setup.....	30

---

5.3.1	<i>Hardware</i> .....	30
5.3.2	<i>Software</i> .....	31
5.4	Operation of the project .....	36
5.4.1	<i>Operating mode 1: Normal operation</i> .....	37
5.4.2	<i>Operating Mode 2: Abnormal Operation</i> .....	44
<b>6</b>	<b><i>Results..</i></b> .....	<b>46</b>
6.1	Test 1: Digital RF playback .....	46
6.2	Test 2: Live Streaming RFSoc data.....	47
6.3	Test 3: Downloading and playing back Digital RF data .....	50
<b>7</b>	<b><i>Conclusions and future work</i></b> .....	<b>56</b>
<b>8</b>	<b><i>References</i></b> .....	<b>58</b>
	<b><i>Annex I: Integration of the SDGs into the project</i></b> .....	<b>61</b>
	<b><i>Annex II: Dev/build tool information</i></b> .....	<b>65</b>



## *List of figures*

Figure 1 - Diagrama de bloques del sistema.....	8
Figure 2 - Interfaz de usuario de la aplicación web.....	9
Figure 3 - Block diagram of the system .....	11
Figure 4 - User interface of the web application .....	12
Figure 5 - RFSoc 2x2 board by Xilinx [5].....	8
Figure 6 - User interface of OpenWebRX [19] .....	16
Figure 7 - User interface of SDR# [22] .....	18
Figure 8 - User interface of GNU Radio [24].....	19
Figure 9 - User interface of StrathSDR .....	20
Figure 10 - Gantt Chart with the planning of the project .....	24
Figure 11 - Overview block diagram of the web application .....	26
Figure 12 - RFSoc board setup .....	28
Figure 13 - RFSoc board setup with RF components.....	29
Figure 14 - Block diagram of live streaming data setup.....	31
Figure 15 - Block Diagram of Digital RF playback setup.....	33
Figure 16 – Stream Options tab .....	37
Figure 17 - Metadata tab.....	38
Figure 18 - Digital RF playback tab .....	39
Figure 19 - Digital RF playback request form access .....	39
Figure 20 - Digital RF playback request form initially upon opening .....	40
Figure 21 - Digital RF playback form once a valid Digital RF file has been selected .....	40
Figure 22 - Play, pause, and rewind buttons under the “DigitalRF Options” tab .....	41
Figure 23 - Metadata of the Digital RF file shown under “Metadata” tab .....	42
Figure 24 - Digital RF download request form access .....	43
Figure 25 - Digital RF download request form initially upon opening .....	44
Figure 26 - Pre-recorded Digital RF shown on the web application.....	47
Figure 27 - Live RFSoc data - Loopback mode .....	49
Figure 28 - Live RFSoc data - Captured with an antenna .....	50

Figure 29 - Data request form – Loopback mode.....	51
Figure 30 - Form for playing back the Digital RF data downloaded in Figure 29 .....	52
Figure 31 – Web application with the data downloaded in Figure 29 being played back...	53
Figure 32 - Data request form – Captured with an antenna.....	54
Figure 33 - Form for playing back the Digital RF data downloaded in Figure 32 .....	54
Figure 34 - Web application with the data downloaded in Figure 32 being played back ...	55
Figure 35 - The wedding cake model for SDGs diagram [29] .....	61

## *List of tables*

Table 1. Budget estimate of the project .....	25
Table 2 - SDGs integrated into this project. SDGs obtained from [28] .....	63

## 1. INTRODUCTION

The radiofrequency spectrum consists of a specific range of frequencies that is used to communicate information. There are plenty of services and applications that strongly depend on the allocation of frequencies in this spectrum such as radio and television broadcasting, satellites, defense, emergency services, and many more. The exponential growth in data quantities that is transmitted over the internet has seriously increased the demand for radiofrequency spectrum. This increase in demand is also caused by the enormous increment in the number of devices connected to networks, including smartphones, tablets, computers, and IoT devices, to name a few.

The electromagnetic waves belonging to this part of the spectrum are called radio waves. These waves can be modulated to encode information, then they are transmitted through the channel of communication and received at the destination which can be some distance away, where the information can be decoded. Different radio frequencies act differently, some have better properties in terms of propagation range, building penetration, resistance to atmospheric conditions, or power efficiency, depending on the use case, the radio frequencies can be more suitable for specific applications. The downside is that radiofrequency waves that have the same frequency can and do interfere with each other. Additionally, stronger signals can silence weaker ones. These are some of the reasons why the radio spectrum needs to be monitored and supervised [1].

This project was developed in collaboration with the MIT Haystack Observatory, a multidisciplinary radio, and radar remote sensing laboratory. The increasing congestion of the radiofrequency spectrum is creating challenges for researchers in the geospace and radio astronomy communities, as congestion makes it difficult to collect high-fidelity measurements. Radiofrequency interference (RFI) mitigation techniques are essential to carry out work in these fields [2].

The members of the MIT Haystack Observatory are interested in a wide variety of RFI mitigation techniques related to cancellation over space, time, and frequency. These techniques require the ability to monitor the wideband RF spectrum.

There is a specific need for monitoring tools that are both inexpensive and with high bandwidth, and the recent progression of software-defined radio (SDR) has the potential to fill this gap. SDR systems are characterized by having components implemented in software that traditionally have been implemented in hardware. SDR allows for more flexibility in changing radio parameters on the fly such as bandwidth or center frequency. SDR is becoming increasingly popular for RF applications due to having potential for rapid design cycles and hardware reusability [3]

One particular SDR device which shows promise for RF monitoring is the Xilinx radio frequency system-on-chip (RFSoc). The device has multiple inputs and can be phase-synced with other boards allowing for spatial filtering applications, which is a major focus of current research [4]. The device has a large potential bandwidth of up to 2.5 GHz. It was designed to minimize energy cost per RF channel, which is critical since current techniques for wideband spectrum monitoring tend to be power intensive. It is relatively cheap compared to other boards with similar capabilities, with a price of \$2,149 per board. All of these factors will help facilitate the deployment of this technology in many different areas, furthering its potential use cases for radio astronomy research.

The Xilinx RFSoc board also comes with the Xilinx PYNQ framework, which simplifies development by allowing it to be carried out using mostly Python, an extremely popular high-level programming language. Traditionally, development of this kind is done in a hardware description language such as Verilog or VHDL, which requires a specialized skill set. The PYNQ framework allows for users who do not have this niche skill set to develop the boards themselves without having to farm out the work to a digital hardware engineer (and having to pay for that engineer's time). The addition of the PYNQ framework makes the RFSoc device a promising candidate for developing complex RF applications that can be prototyped easily and cost-effectively.

The MIT Haystack Observatory scientists would like to investigate this technology, the RFSoc board, and its relevance to various RF sensing applications within the geospace and radio astronomy communities. One application of interest is monitoring the RF spectrum environment over wide bandwidths. This type of system, with proper phase and time reference signals, could also be useful for nulling interference signals recording the RF environment. This system could be a valuable tool for other projects taking place in the observatory related to interference mitigation research and, in general, for the radio astronomy research community.

## 2. DESCRIPTION OF TECHNOLOGIES

### 2.1 XILINX RFSOC 2x2 BOARD

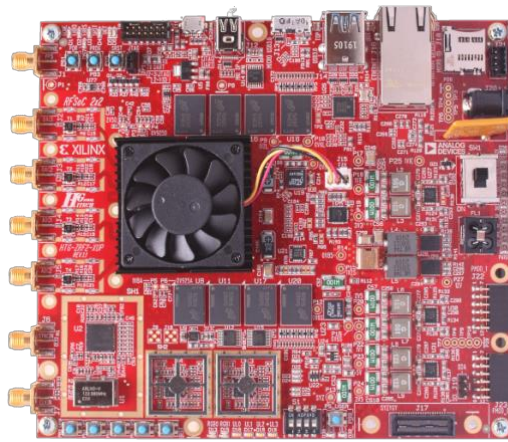


Figure 5 - RFSoc 2x2 board by Xilinx [5]

Software-defined radio (SDR) is a radio communication system where components that have been traditionally implemented in hardware are instead implemented in software. One particular SDR device which shows promise for RF monitoring is the Xilinx Radio Frequency System-on-Chip (RFSoc).

The Xilinx Radio Frequency System-on-Chip (RFSoc), shown in Figure 5 - RFSoc 2x2 board by Xilinx, is a type of integrated circuit that can be used for communications and instrumentation [6].

RFSocs generally feature high-accuracy ADCs (analog-to-digital converters) and DACs (digital-to-analog converters) operating at Giga samples per second (Gsps), an ARM-based processing system, and an FPGA programmable logic facility. In this case, it is an RFSoc 2x2 board, which indicates that it has 2 RF DAC and 2 RF ADC channels. A key feature of an RFSoc's DAC or ADC is its ability to receive and transmit signals in higher-order Nyquist bands, for example, on the RFSoc 2x2 by Xilinx, the 2nd order Nyquist bands can be used covering the spectrum from 2GHz to 4GHz.

The device has a large potential sampling rate of up to 4 GHz. It was designed to minimize energy cost per RF channel, which is critical since current techniques for wideband spectrum monitoring tend to be power intensive. It is relatively cheap compared to other boards with similar capabilities, with a price of \$2,149 per board as of June 2022. This price suffered an increase due to the supply chain issues that the hardware industry is experiencing due to the coronavirus pandemic. All these factors will help facilitate the deployment of this technology in many different spaces, furthering its potential use cases for radio astronomy research.

The RFSoc 2x2 is primarily intended for academic users and is currently only available to verified academics. For this project, this board has been used as the source to capture data, however, almost all of the code written for this project is board-agnostic. As such, it should be easy to expand the web application to work with any SDR device.

## 2.2 *DASH*

Dash is a framework that was created for rapidly building interactive and complex dashboard visualizations in Python, R and Julia by using Plotly graphs. Dash is ideal for building and deploying data apps with customized user interfaces. It is built upon React.js for the front end, Plotly.js for the interactive graphs, and Python Flask for the Web server [7].

Through a couple of simple patterns, Dash abstracts away all the technologies and protocols that are required to build a full-stack web app with interactive data visualization. It allows you to write a web application entirely in Python without having to write any front-end code in JavaScript, which significantly simplifies development. Python has many data-processing libraries and toolkits relevant to this project (including the Digital RF library), those libraries and tools are not available in JavaScript and ReactJS. Writing code entirely in Python makes it easier to carry out the complicated data processing necessary for the dashboard graphs of this project.

Another benefit of Dash is the reusable user input components such as buttons, sliders, and dropdown menus. These reusable components streamlined development of the dashboard and are a major benefit of using the Dash framework.

Because of all the advantages mentioned, Dash suited the requirements for the front-end portion of the spectrum monitoring dashboard and, thus, it was the framework used for the front-end of the web application.

## 2.3 *REDIS*

Redis is an open-source in-memory data structure store that can be used as a database, cache, message broker, and streaming engine. It provides several data structures such as strings, hashes, lists, sets, bitmaps, streams, etc. These data structures allow to run atomic operations on them, like appending to a string, incrementing the value in a hash, pushing an element to a list, etc.

Redis is written in the C programming language and works on most POSIX systems such as Linux, BSD, and Mac OS X without external dependencies. Since it requires a UNIX machine, to run Redis on Windows, a Windows Subsystem for Linux is needed [8].

### REDIS STREAMS

Redis Streams is a data type that allows for streaming operations between producers and consumers. It models a log data structure more abstractly but maintains the essence of the log intact: like a log file, often implemented as a file open in append-only mode, Redis Streams are primarily an append-only data structure [9].

Redis Stream is well suited for time series and message queues, which fits well with graphs that are updated over time.

This project uses Redis for message passage and data storage. The data is stored in the Redis database using the JavaScript Object Notation (JSON) format. A Redis database is used for communication between the front end and the back end, which provides a single clean interface for a variety of data processing pipelines.



## ***2.4 DIGITAL RF***

Digital RF is a standardized format for reading and writing radio frequency data. It is designed to be self-documenting for data archive and to allow rapid random access for data processing. The Digital RF project contains the software for doing so, its libraries include compatibility with programming languages such as C, Python (with blocks for GNU radio), and MATLAB. It was developed by the clients of this project, MIT Haystack Observatory [10].

This web application uses Digital RF in two parts: (i) when playing back previously recorded data in the Digital RF format, and (ii) when converting and storing raw data from the RFSoc board in Digital RF format via a ZIP file.

The first part involves playing back locally stored RF data in the Digital RF format, in which the user can choose which Digital RF file to play along with additional configurable options for the displaying process. The second part includes converting live raw data obtained from the board into a Digital RF file and downloading it as a ZIP file so that it can later be displayed as in the first part. Both parts will be further explained in future sections.

## ***2.5 PYNQ FRAMEWORK***

PYNQ is an open-source project from Xilinx. It uses the Python language and libraries, allowing designers to exploit the benefits of programmable logic and microprocessors to build more capable and exciting electronic systems [11].

PYNQ can be used with Xilinx platforms to create high-performance applications with hardware-accelerated algorithms, real-time signal processing, high bandwidth IO, and low latency control, among other characteristics. If higher performance is needed, Python and PYNQ can be combined with C/C++ modules.

A PYNQ-enabled board can be easily programmed in Jupyter Notebook using Python, which allows developers to use hardware libraries and overlays on the programmable logic, implying an increase in the speed of the software running on the board.

An advantage of the PYNQ framework is that the only software needed to start programming in it with Python is a compatible web browser with Jupyter notebooks such as Firefox, Chrome, or Safari.

## **2.6 STRATHSDR**

StrathSDR was developed by the University of Strathclyde Software Defined Radio Lab, which partnered with Xilinx for the project that developed the RFSoc 2x2 and its PYNQ libraries. It is an open-source library that comes installed on the Xilinx RFSoc board and contains a variety of graphs and visualizations for the board, including spectrum and spectrogram plots [12].

StrathSDR's lab has also developed a variety of educational resources to support the Zynq RFSoc, which is another Xilinx board, and the RFSoc 2x2. Among them, we can find RFSoc introduction notebooks about how to work with the data converters of the board, and Digital Signal Processing (DSP) notebooks that cover a wide range of DSP topics such as Sampling and Quantization or, Modulation and Demodulation.

Their repository offers various examples of PYNQ applications, including an RFSoc Spectrum Analyzer Module, an RFSoc Frequency Planner, and a QPSK radio transceiver using the RFSoc, to name a few.

StrathSDR has been used as a base to work from because the library is open source, contains code that pertains directly to the RFSoc board, and produces outputs similar to the desired outputs of this project. The capabilities of this library have been modified and extended to further meet the needs of the clients.

## **2.7 JSON**

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It presents advantages for both humans and machines. Regarding humans, it is easy to read and write. And, regarding machines, it is easy to parse and generate [13].

JSON is completely independent of any programming language. However, it uses conventions from several languages from the C-family such as C, C++, Java, JavaScript, Python, and many others. These are the main reasons why JSON is a great fit for data-interchange use cases.

About its structure, JSON supports two data structures: (i) a collection of name/value pairs, in programming languages called for example object, dictionary, and hash table. And, (ii) an ordered list of values, in programming languages called array, vector, list. Since there are universal data structures, all modern programming languages support them [14].

Related to this Project, in order for there to be communication between multiple Redis instances, the data must first be serialized. Then it can be transmitted, de-serialized, and displayed on the front-end web application. For this serialization process, the technology used is JSON.

## **2.8 JUPYTERLAB**

Project Jupyter is a non-profit, open-source project. The goal of the community behind it is to "develop open-source software, open-standards and services for interactive computing across dozens of programming languages". Project Jupyter encompass interactive computing products such as Jupyter Notebook, JupyterHub, and JupyterLab [15].

JupyterLab is the next generation of the Jupyter Notebook. It uses Julia, Python, R, or one of many other languages. As a curiosity, Jupyter's name is a reference to these three core programming languages mentioned, which are the main ones supported by Jupyter.

JupyterLab also improved the interface of the notebooks, but it went a step further. It provides the same interface in the browser to different common tools such as file browsers, consoles, terminals, text editors, Markdown editors, CSV editors, JSON editors, among others. This flexibility in the interface position JupyterLab as an excellent tool for data science, scientific computing, computational journalism, and machine learning [16].

Regarding compatibility, it shares server and file format with Jupyter Notebook, allowing full compatibility between notebooks and kernels from both.

## 3 STATE OF THE ART

This section explains the environment in which the project operates, including an analysis of the project context and the competing technologies. It describes the more significant technologies that have requirements similar to the ones this project has.

### 3.1 PROJECT CONTEXT

To explain the project context, it is important to understand the transition from communication systems that have been traditionally implemented in hardware to the same systems implemented in software, this new implementation defines a software-defined radio (SDR) [17].

People needing to communicate (data, voice and video communications, broadcast messaging, etc.) has been in exponential growth, thus, it has become a priority to modify radio devices to become more cost-effective. Part of this growth has been covered by SDRs, which provide flexibility and cost-efficiency

Conventional radio devices, hardware-based, require physical presence to be modified, which derives in higher costs and lower flexibility for applications with different waveform standards. On the other hand, SDR devices provides a cheaper and more efficient solution to this issue by allowing multi-functional wireless devices to use software updates.

In other words, traditionally, when creating a radio communication device, the engineer had to deal with problems such as creating a particular circuit for detection depending on the transmission, designing a particular circuit that would decode/encode that specific signal or debugging the device with special equipment. With SDR, all these problems become accessible via software by using algorithms to process the signal on a computer.

There are several advantages and beneficiaries of SDR. For Radio Equipment Manufacturers, SDR provides software to be reused across radio products and allows for reprogramming and bug fixes to occur while the radio is operational. For Radio Service

Providers, SDR enables: (i) new features to be added to devices without requiring major costs and, (ii) the use of a common radio platform for multiple markets. For end-users, SDR technology aims to reduce costs and allows for more flexibility in changing radio parameters on the fly, such as bandwidth or center frequency.

A few examples of the adoption of SDR technologies are military applications, amateur radio, and mobile communications [18].

SDR is becoming increasingly popular for radiofrequency applications due to its potential for rapid design cycles along with its allowance for reusing hardware between multiple applications.

Furthermore, the radio frequency spectrum is becoming increasingly congested. This congestion makes it difficult to collect high-fidelity measurements, which are needed for the radio astronomy community. That is why Radio Frequency Interference mitigation techniques are essential to carry out work in these fields. There is a specific need for RF monitoring tools to be able to carry out these techniques, and the recent progression of Software-Defined Radio has the potential to fill this gap.

## **3.2 *COMPETING TECHNOLOGIES***

Other similar concepts/projects/technologies approach the topic of this project, they could be seen as competing technologies in some way but, since this project consists of the development of a web application and some of those projects are open-source, some of their capabilities will be used in this project to continue providing open-source code to the community.

### **3.2.1 *OpenWebRX***

OpenWebRX is an open-source multi-user Software-Defined Radio receiver. One of its advantages is that it does not need any downloadable software because it can be operated from any web browser, i.e., the only requirements to use it are a computer, network access, and an SDR device [19].

OpenWebRX is a remote spectrum monitoring tool that was originally designed for the amateur radio community. It uses a waterfall plot or spectrogram that shows the frequency spectrum over time as can be seen in Figure 6 - User interface of OpenWebRX.

While it has functionality similar to the one this project wishes to implement, these web-based SDR lacks some of the features and characteristics the client wants, such as showing both spectrum and spectrogram, more interactivity with the graphs, and Digital RF compatibility [20].

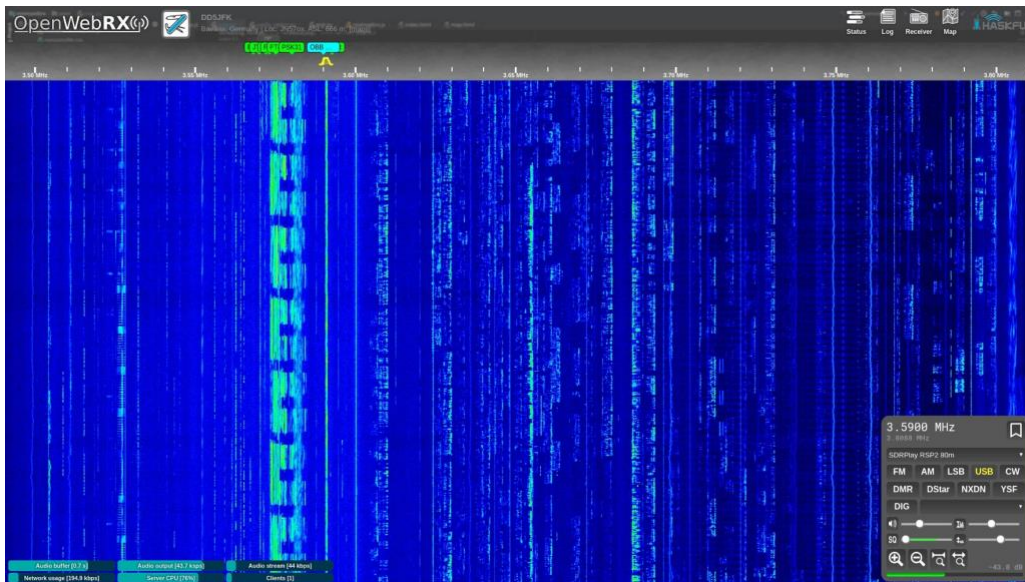


Figure 6 - User interface of OpenWebRX [19]

### 3.2.2 KiwiSDR

KiwiSDR is a software-defined radio that attaches to a specific embedded computer, the Seed BeagleBone Green embedded computer. It is a standalone device that attaches to your local network and is optionally accessed through the Internet. A browser is used to connect to the user interface [21].

One interesting feature about KiwiSDR is that the owners of Kiwis, instead of having them set up locally, can make them publicly available. By doing this, everyone can access them via the internet. Right now, there are over three hundred available worldwide.

Another interesting characteristic about KiwiSDR is that it allows software extensions, providing more customization to the user. These include signal strength graph, IQ display, and WSPR decoder, among others. The most popular is the WSPR decoder, used for weak-signal radio communication between amateur radio operators.

Regarding compatibility, KiwiSDR is compatible with the following browsers: Firefox, Chrome, Safari, and Opera. And with the following operating systems: Windows, Linux, and Mac. The browser interface depends on the length of the display so currently, it works on iPads and Android devices but not on mobile devices because the screen is not large enough. The mobile device update is under development.

It uses OpenWebRX as one of its tools for displaying the frequency data. The KiwiSDR is the SDR itself, not the web application that handles the displaying of the data, so it can be compared with the RFSoc 2x2. The most important difference is that the typical digitized bandwidth of the KiwiSDR is limited to about 30 MHz, which is substantially smaller compared to the 2.5GHz of bandwidth that the RFSoc is capable of.

### **3.2.3 SDRSharp**

SDR# is a PC-based DSP application, developed by Airspy, for Software-Defined Radio written in C#. It supports several third-party SDRs such as the RTL-SDR, although it was designed for Airspy devices.

The main purpose of SDRSharp is to offer a simple proof of concept application to get hands on DSP techniques. As the development platform, it uses .NET 6, the newest version of the Microsoft framework [22].

Some of the advantages of SDR# include that it offers high levels of customization, allowing third-party developers to easily code plugins. It also provides a high level of user interaction with the application as can be seen in the image below. Finally, the process of installation, setup, and uninstallation is easy, it doesn't require advanced knowledge in DSP or SDR, even the most inexperienced user can easily start with SDR# and even with the most sophisticated plugins. A screenshot of its user interface is shown in Figure 7 - User interface of SDR#.



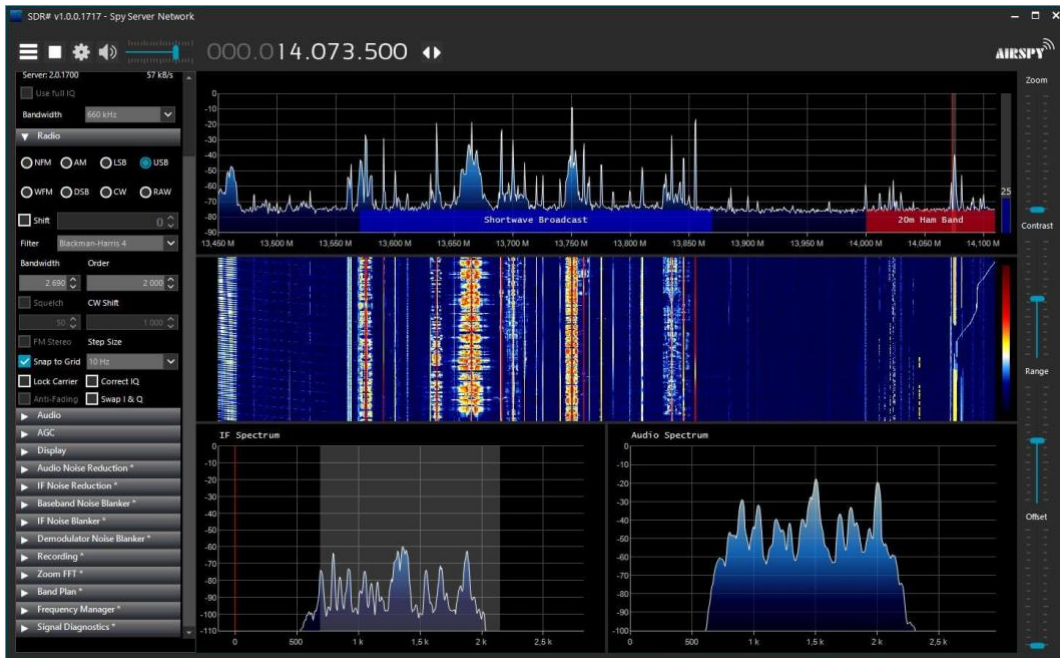


Figure 7 - User interface of SDR# [22]

### 3.2.4 GNU Radio

GNU Radio is a widely used open-source software development suite that provides signal processing blocks for SDRs and other systems. These blocks can be connected together to achieve the desired functionality generating a GNU radio application called flowgraph. These flowgraphs can be written in either C++ or Python. The suite contains many pre-made and pre-tested radio components which facilitate the development of SDRs [23].

GNU Radio enables the design, simulation, and deployment of radio systems. It serves as the signal-processing software that handles the processing specific to each different radio application. An image of the user interface of GNU Radio is shown in Figure 8 - User interface of GNU Radio.

However, GNU Radio blocks can only run on general-purpose CPUs, as opposed to the Xilinx PYNQ framework which can directly program the logic circuits of the RFSoc board. As such, programming through the PYNQ framework is necessary to make full use of the board's resources.



As for compatibility, GNU radio works better on Linux and Mac OS X devices. For Windows, it doesn't have direct maintenance support, although it will build and run under it. As for the system requirements, GNU Radio in its core is C++ with lots of user functionality relying on Python. So basically, as long as the platform has a feasible compiler, it can work.

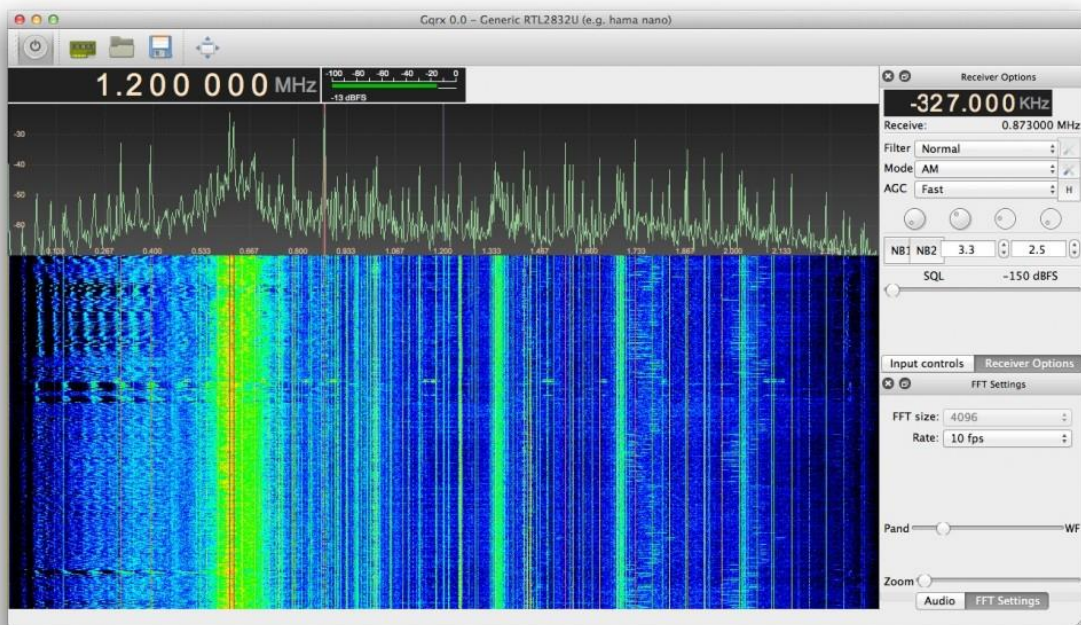


Figure 8 - User interface of GNU Radio [24]

### 3.2.5 StrathSDR

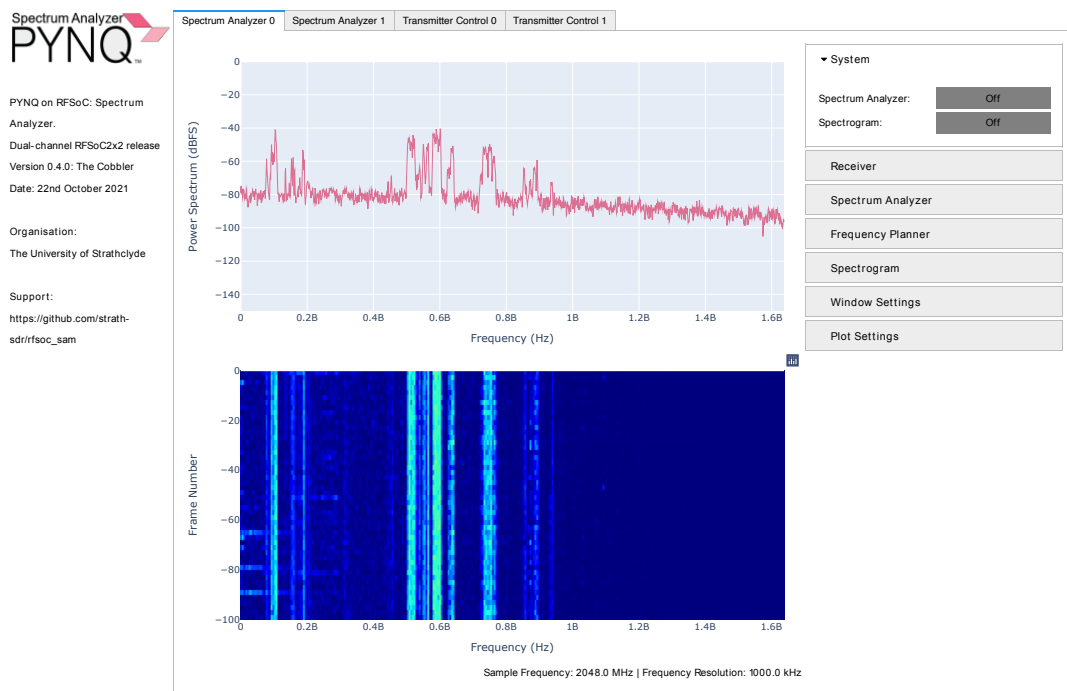
StarthSDR was already explained in section 2.7 under “Description of Technologies”, but it can also be seen as a Competing Technology since the use cases it covers are very similar to the ones this project aims to fulfill.

StrathSDR is a library created by The University of Strathclyde SDR Lab team. It includes a Jupyter-based web application for the Xilinx 2x2 RFSoc. It includes various examples of

applications developed with PYNQ such as an RFSoc Spectrum Analyzer Module or an RFSoc Frequency Planner, among others.

Since the client's request was an interactive, well-documented, easy-to-expand web application that could display the spectrum and spectrogram plots, the Spectrum Analyzer application developed by StrathSDR was an interesting starting point that also served as an objective of how the interface of this project could look like. So, rather than starting from scratch, the first steps of the front end were piecing together related existing components from StrathSDR open-source libraries.

StrathSDR's interface of the Spectrum Analyzer is shown in *Figure 9 - User interface of StrathSDR*.



*Figure 9 - User interface of StrathSDR*

## 4 DESCRIPTION OF THE WORK

### 4.1 *JUSTIFICATION*

The client of this project is the MIT Haystack Observatory. The scientists at the observatory wanted a web application to monitor the RF spectrum similar to StrathSDR in terms of interface and user interactivity but with a higher grade of customization and with well-documented code. This last feature is important since this project aimed to create a flexible base to be extended as needed to serve future research needs. MIT Haystack Observatory scientists want to continue developing this web application so that it can be used in its Radio Frequency Interference mitigation techniques.

The focus of this project is on the functionality of the different components, rather than any particular deployment scheme. This focus is due to the intended use case of this work. As it has been explained, this project is meant to serve as a base that can be easily extended to suit future research needs. The original requirements and the new ones that appeared during the process of developing the web application were set by the observatory, but the target audience is not only the client of the project but also software-savvy radio frequency researchers who will likely tweak the codebase to suit whatever particular goals they may have. As such, these researchers will presumably use their own deployment methods, whether that be running parts of the code on the cloud, or on machines in a lab.

### 4.2 *OBJECTIVES*

This section explains the objectives of the project. As it was mentioned in the previous section, this project has been developed for a client, MIT Haystack Observatory, thus, the objectives or requirements of the project were set by them. The user requirements shown in the list below are the final delivered specifications. For a better understanding and more precise use of the requirements, to the extent possible, all of them should indicate quantifiable measures.

- The library must be open-source and easily accessible to other members of the geospace and radio astronomy communities.

- The web application must display a full range of live data from the RFSoc, meaning, a bandwidth of up to 2.5GHz (depending on the antenna and programmed bandwidth)
- The delay between data entering the RFSoc board and data being displayed on the dashboard must be lower than 1 second.
- The rate of data frames displayed while live streaming data must be around 200 msec, i.e., the web application should display new data frames every 200 msec
- Data must be downloaded from the RFSoc in Digital RF format via a ZIP file
- The user must be able to interact with the graphs:
  - Can adjust scales between logarithmic and linear values
  - Can adjust y-axis bounds
  - Can track maximum and minimum points on the graph
  - Can change the color scale of spectrogram graph
- Easy setup: the codebase is thoroughly documented, including installation and setup procedures
- Cost:
  - There should be no extra cost to the user to run the web application if they already have the appropriate hardware.
  - The software portion of the application is free to run

### **4.3 WORKING METHODOLOGY**

This project has been developed in collaboration with MIT Haystack Observatory. They handed in a document that briefly explained the motivation they had for the project, Radio Frequency spectrum monitoring to be used in RF interference mitigation. And some initial requirements, they wanted a web application that could show the spectrum and spectrogram plots with user interactivity, compatibility with Digital RF data, and high grade of customization, everything with well-documented code.

After analyzing these requirements, the methodology chosen to be followed was the Agile methodology, since it is a software project, it is one of the most appropriate and successful ones for this type of project.

In the Agile methodology, the planning is incremental, several tasks are done in parallel, and it is easier to modify the process to reflect the changing needs of the client. The requirements and the pertinent solution evolve and change in time in relation to the client's needs.

All the tasks of the project are collected in the product backlog, these tasks or features are agreed to be done in certain periods of time, and each period is called a Sprint. Similar to the product backlog, the collection of tasks belonging to a Sprint is called the Sprint backlog. The Sprints divide the timeline of the project and guarantee that the client is constantly receiving partial but constant deliveries. At the beginning of each Sprint, a meeting to organize everything takes place, and, at end of each Sprint, the progress made is presented with the objective to do a Sprint review. In this case, the duration of each sprint was one month, so, since the duration of the project was seven months, there were seven sprints.

Some of the core values of the Agile Manifesto were followed:

- Working software over comprehensive documentation: prioritizing weekly deliveries with working software so that the client can say what else should be added to the web application.
- Customer collaboration over contract negotiation: weekly meetings with the client so that they can lead the project in the direction they want since they will be using the web application/libraries in other projects they have, the project had to be developed based on the needs of those others.
- Responding to change over following a plan: as the RFSoc board didn't arrive until later in the first semester, new objectives had to be adopted during that time.

The project was firstly based on designing the web application for the Xilinx RFSoc 2x2 but unfortunately, the board arrived at the end of the first semester, mid-December. That's why the Gantt Chart only illustrates the project estimated schedules for all the tasks for the second semester. During the first semester, I focused on doing research about the project, including: SDR workshops, StrathSDR code analysis, Redis tutorials and more, which helped accelerate the development of the web application for the second semester.

To illustrate the project schedule, a Gantt chart is shown in Figure 10 - Gantt Chart with the planning of the project. It includes all the tasks that have been carried out with their corresponding start and end/due date, together with the percentage of completion. In this case, since the project is finished, all the percentages show 100% but the chart was being



Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
1	1	Xilinx RFSoc 2x2 Kit	\$2,149	\$2,149
2	1	Ethernet Cable	\$8	\$8
3	1	SMA Cable	\$6	\$6
4	1	OmniLOG PRO 1030 Antenna	\$270	\$270
5	2	Mini-Circuits ZX60-P105LN+ Low Noise Amplifier	\$90	\$180
6	1	Mini-Circuits VLF-1400+ LTCC Low Pass Filter	\$25	\$25
7	1	Mini-Circuits FW-6+ 6 dB Fixed Attenuator	\$21	\$21
8	1	Mini-Circuits VLM-33W-2W-S+ LIMITER	\$50	\$50
Beta Version-Total Cost				\$2709

*Table 1. Budget estimate of the project*

The cost of the software components of this project is negligible since the web application is not hosted on any third-party cloud service. The dominating expense is the cost of the RFSoc board, even more with the supply chain issues that the hardware industry is experiencing due to the coronavirus pandemic.

In addition to the cost of the board, an antenna and other RF equipment are needed to be able to receive signals. The cost of these accessories will depend strongly on the user's needs and budget, i.e., a rather basic antenna would be enough to try the web application but in order to receive signals with better quality, a better antenna must be used. The antenna listed in the table above is a high-end antenna for this application. Again, this would depend on the user's requirements.

A few RF accessories which can be used for a basic monitoring setup are included in the table above. These accessories were recommended by the client. A further explanation of the board set up with these accessories is shown in Section 5.2. Any additional specialized equipment will likely cost more.



## 5 SYSTEM DEVELOPED

### 5.1 OVERVIEW BLOCK DIAGRAM

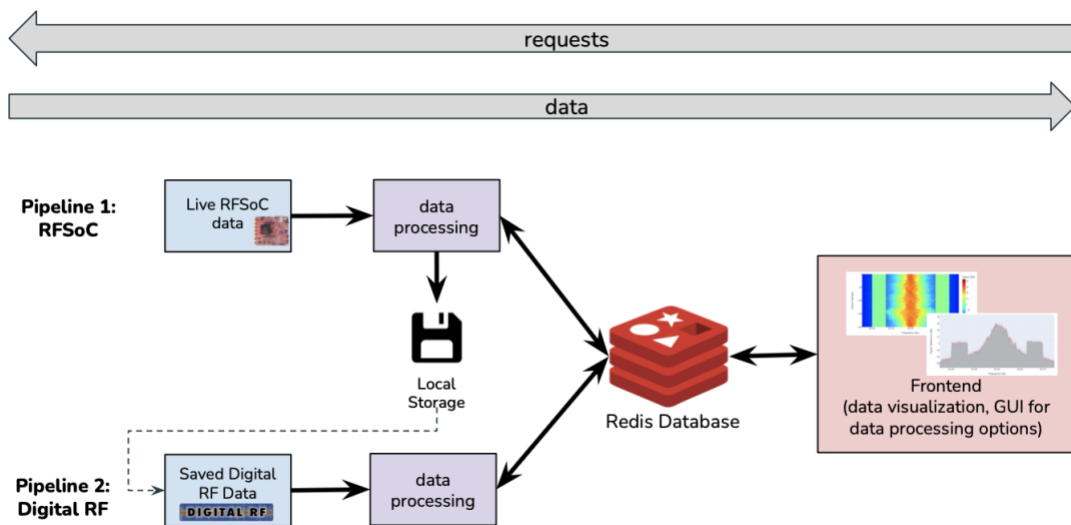


Figure 11 - Overview block diagram of the web application

The overview block diagram is presented in Figure 11. The diagram shows the flow of data between the front end and the different back-end data pipelines of the web application.

The web application can display spectrum data for two different pipelines.

Pipeline 1 processes live data incoming from the RFSoc. The board receives raw IQ data through an antenna, some data processing is done on the board including performing the FFT on the data with the corresponding previous windowing to avoid undesirable artifacts in the FFT amplitude because of the signal not being periodic. Then, the FFT data together with some metadata is streamed into Redis, this stream is read by the front end and all the data is displayed in the web application, the metadata is used for actions such as adjusting the vertical and horizontal axis. The streaming of data can be paused at any moment and resumed whenever the user wants to.



Pipeline 2 processes locally stored RF data in the Digital RF format. This pipeline was a request from the clients at MIT Haystack Observatory, since they developed the Digital RF format, they wanted to be able to display data in that format. To do so, the part including the board is not used, in the front end a file in that format is selected and the back end handles the request to open it and process it. It is then streamed into Redis, and the front end reads the stream and displays the data. The data can be paused, rewound to the beginning of the file, or resumed after being paused.

There is another pipeline that combines the previous two, this feature lets the user download data from the RFSoc board in the Digital RF format, which can be played back later. The process would be as follows: the RFSoc is constantly receiving raw IQ data through the antenna and it's continuously waiting for the front end to send a request. When the front end sends a request to the board asking for x seconds of data, the board dumps the next x seconds of raw IQ data in Redis (together with the corresponding metadata). The front end creates a Digital RF directory from the data and metadata, zips the directory into a single .zip file, and then has the user's browser download it. Finally, the user can play back the data stored in the zip file similarly to Pipeline 1.

Further explanations and instructions on how to install the web application, set it up and run it for all the pipelines are present in Section 5.3.

## **5.2 *HARDWARE OF THE PROJECT***

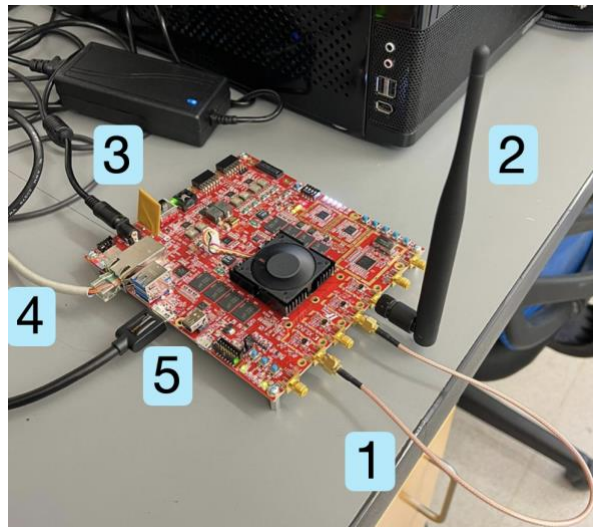
This project has not built or created any hardware. The hardware used mainly consists of the Xilinx RFSoc 2x2 board and an antenna. Any other radio frequency accessory or component is not required but is present on the budget estimate shown in section 4.4.

### **5.2.1 *Board setup***

The RFSoc has two channels, each with one transmitter and one receiver. For the demos/tests that will be explained in the results section, the setup was as follows:

- In channel 0: an antenna is connected to ADC2 (a receiver), to monitor the RF spectrum.

- In channel 1: an SMA cable connecting DAC1 (a transmitter) and ADC1 (a receiver) in "loopback mode". This is used for testing purposes. Connecting one transmit channel to a receive channel allows to control exactly what data goes into the receive channel so that it can easily be tested whether or not the data being displayed on the web application is accurate.



*Figure 12 - RFSoc board setup*

Figure 12 shows the RFSoc board setup. It includes:

- SMA cable: connects, for the same channel, the transmitter (DAC) to the receiver (ADC), loopback.
- Basic SDR antenna: connected to the 2nd receive channel to be able to monitor the RF environment (not the OmniLOG PRO 1030 Antenna shown in the Budget Estimate).
- Power cable.
- Ethernet cable: connects the board to the internet. It can be connected to a router or to a PC with internet access.
- Micro USB cable: connects the board to the PC used to control the board. Specifically, the PC accesses the RFSoc via JupyterLab, allowing it to run Python scripts on the board.

In Figure 13 the setup is the same as in the previous example, but some RF components have been added, they were connected between the antenna and the input of ADC2.

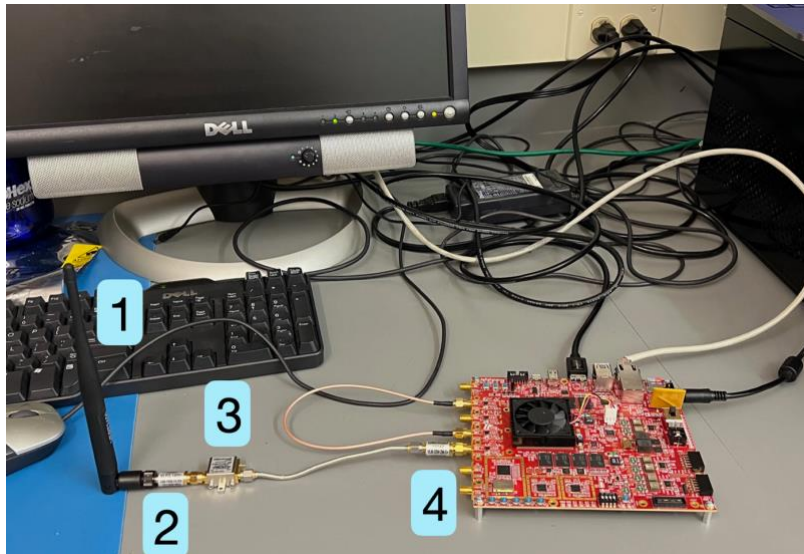


Figure 13 - RFSoc board setup with RF components

The RF components present for this setup are:

1. Basic SDR Antenna (not the OmniLOG PRO 1030 Antenna shown in the Budget Estimate).
2. Low Pass Filter: passes signals with a frequency lower than its cutoff frequency, in this case, 1400MHz
3. Low Noise Amplifier: amplifies a very low-power signal
4. Limiter: prevents damage to the LNA due to excessive power at the input of the system

Note: In the image, the LNA is not connected to the power supply, it should be connected for the LNA to work on the circuit.

For information about these components visit the Mini-circuits webpage [25] and search for the specific component (the full name of each component is provided in the Budget Estimate in section 4.4)

### 5.2.2 Power requirements

In this project there are two components that require power:

- RFSoc board: requires 12V. The RFSoc board kit comes with a 12V-72W power supply unit, no external power supply must be bought.

- Low Noise Amplifier (LNA<sup>1</sup>): requires 5V DC. To supply this voltage, the following power supply has been used, DC POWER SUPPLY HY3005D. Some characteristics of this power supply:
  - It can operate in 115 AC or 230 AC
  - It has adjustable outputs: 0-30V and 0-5A

## 5.3 *INSTALLATION AND SETUP*

### 5.3.1 *Hardware*

The hardware for this project is a Xilinx RFSoc 2x2 board (hereafter referred to as “the RFSoc” or “the board”). The getting started guide for the board is located at [https://www.rfsoc-pynq.io/rfsoc\\_2x2\\_getting\\_started.html](https://www.rfsoc-pynq.io/rfsoc_2x2_getting_started.html). Follow all these instructions to get your board running.

The first step of the previous link describes how to install the latest PYNQ image onto the board. An up-to-date PYNQ image is necessary to run the board scripts involved in this project. This image can be flashed onto the board’s SD card using the balenaEtcher software, which can be installed onto your PC from their official website. Once you have installed and run balenaEtcher, insert the board’s microSD card into an SD card reader on your PC, select ‘Flash from file’, choose the PYNQ image that you downloaded in the first step, choose the proper microSD card under ‘Select target’, and finally press the ‘Flash!’ button. The PYNQ image should now be flashed onto the card, and you can now insert the card into the board. Afterward, continue with the remaining steps on the “Getting Started” page linked above. After finishing the last step described on the “Getting Started” page, you will be on the JupyterLab interactive development environment (IDE). In order to get the relevant scripts for this project onto the board, go to this project’s GitHub repository located at <https://github.com/kitkatkandybar/RFSoc-Spectrum-Monitoring/>. Under the **board/** folder, there are two files, one called **stream.ipynb** and one called **download.ipynb**. For each of these two files, create a new file in JupyterLab and paste the corresponding code into the file in the IDE.

Your RFSoc is now ready to run scripts!

---

<sup>1</sup> This component is not required for the use of the web application but since the web application has been tested with RF components such as the LNA, its specifications have been added for information purposes

### 5.3.2 Software

#### 5.3.2.1 Software installation

Git needs to be installed on your computer in order to download the software. To install the codebase of the software, open a terminal window that has git installed and run:

```
git clone
https://github.com/kitkatkandybar/RFSoc-Spectrum-Monitoring.git
```

This repository requires Anaconda to manage python versions and packages. You can install Anaconda from its official website [26]. Once you have installed Anaconda, navigate to the project repository in a terminal window and run:

```
conda env create -f environment.yaml
```

This creates the Anaconda environment and installs the necessary packages.

The software also requires a Redis server. You can download Redis at their official website [27]. Redis requires a UNIX machine to run, it can be run on a Linux machine, a macOS machine or in an Ubuntu environment running on Windows Subsystem for Linux.

#### 5.3.2.2 Running Pipeline 1 - Live streaming RFSoc data

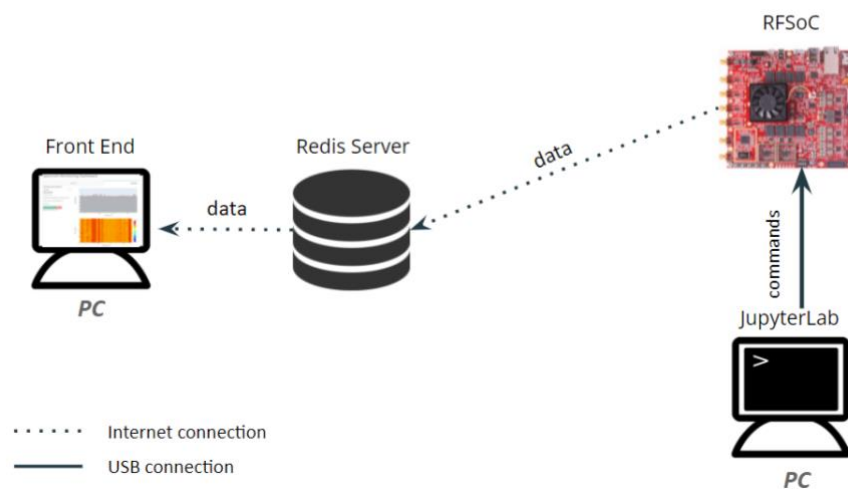


Figure 14 - Block diagram of live streaming data setup

Figure 14 shows the block diagram of the setup for live streaming RFSoc data, Pipeline 1. A PC commands the RFSoc to run scripts through a JupyterLab environment. The RFSoc dumps data into a Redis stream, which the front end reads and displays in graph format.

The live-streaming pipeline requires four major components:

1. The Xilinx RFSoc 2x2 Board
2. A computer to command the RFSoc Board connected to it
3. A computer to run the Redis Server
4. A computer to run the front-end Web application

2-4 can all be run on the same machine or on separate machines. The potential of Redis is shown here because the three steps can run on three different machines. Going into more detail, we must have a PC connected to the RFSoc to run the board scripts, another PC in which we would like to see the graphs, and the third PC could be anywhere around the world with just one requirement, it should have internet connectivity. So, as long as the front-end PC (the one showing the graphs) and the RFSoc (not the PC connected to the board but the board itself) can ping the IP address of the PC running the Redis server, everything would work.

To run this pipeline:

#### I. Start the Redis Server

To start a Redis database, open a terminal that has Redis installed, navigate to this project's repository and run the following command:

```
redis-server ./redis.conf
```

You can specify configuration parameters for the database in **redis.conf**, which is located in the repository. You can find details for configuration parameters at <https://redis.io/docs/manual/config/>. For the Redis connection to be private and provide security to the data that is being streamed, a password for connecting to the server has been set. This password is located in the redis.conf file and can be changed as long as it is also changed in the config.yaml file located in the /frontend folder (for live streaming the back-end configuration file is not being used, so it needn't be changed there as well). By default, Redis listens on 'localhost' on port 6379.

## II. Run the board script

First, obtain the IP address of the computer running the Redis server. You can find the public IP address of a computer by running **ipconfig** (on Windows) or **ifconfig** (on Unix) in a terminal. Alternatively, you can get your IP address by searching “What’s my IP” on Google. Then, in the **simplestream.ipynb** file you created on the board in 5.3.1, enter the IP address in the line of code which sets up the Redis connection:

```
r = redis.Redis(host='155.41.48.219', port=6379, db=0,  
password='Qsh4r9.VlMj5_HrvY#0f36')
```

Run the cells by pressing the play button in each cell. After a few seconds, data from the board should start streaming into Redis. If there is a connection issue with the database, make sure you have specified the proper host IP, port, and password values in the Redis configuration parameters.

## III. Run the front end

First, set the IP address of the Redis server (which you found in step II) in **front\_end/config.yaml**. You can also configure the location of where the front end will be located. To start the front-end web application, run the following in a terminal window:

```
conda activate rfsoc  
python ./front_end/app.py
```

The web application should now be running in the location you specified in the configuration file and can be accessed via a web browser. By default, it should be at <http://127.0.0.1:8050/>

### 5.3.2.3 *Running Pipeline 2 - Digital RF Playback*

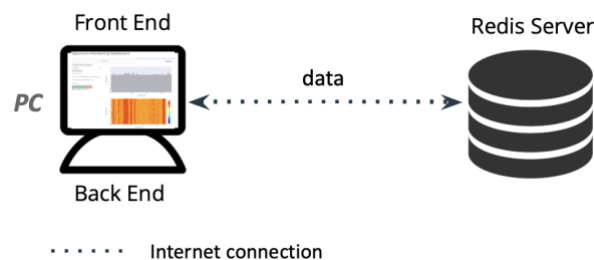


Figure 15 - Block Diagram of Digital RF playback setup



Figure 15 shows the block diagram of the setup for playing back data in the Digital RF format. The Digital RF Playback has three components:

- The back end
- The Redis server
- The front end

For this Pipeline, the front end and the back end must run on the same machine since the user will be selecting in the web application (front end) which Digital RF file to load from its PC. The Redis server can run on the same or in a different machine.

### I. Start the Redis Server

To start a Redis database, open a terminal that has Redis installed, navigate to this project's repository and run the following command:

```
redis-server ./redis.conf
```

You can specify configuration parameters for the database in **redis.conf**, which is located in the repository. You can find details for configuration parameters at <https://redis.io/docs/manual/config/>. For the Redis connection to be private and provide security to the data that is being streamed, a password for connecting to the server has been set. This password is located in the redis.conf file and can be changed as long as it is also changed in the two config.yaml files, both located in the /frontend and /backend folders (it must be changed in both files). By default, Redis listens on 'localhost' on port 6379.

### II. Run the Back end

First, obtain the IP address of the computer running the Redis server. You can find the public IP address of a computer by running **ipconfig** (on Windows) or **ifconfig** (on Unix) in a terminal. Alternatively, you can get your IP address by searching "What's my IP" on Google.

Next, set the IP address and port of the Redis server in **back\_end/config.py**. Then, to start the back-end script, run the following in a terminal:

```
conda activate rfsoc  
python ./back_end/drfs_request_handler.py
```



### III. Run the Front end

First, set the address of the Redis server (which you found in step II) in **front\_end/config.yaml**. You can also configure the location of where the front end will be located. To start the front-end web application, run the following in a terminal window:

```
conda activate rfsoc  
python ./front_end/app.py
```

The web application should now be running in the location you specified in the configuration file and can be accessed via a web browser. By default, it should be at <http://127.0.0.1:8050/>

#### **5.3.2.4 Running extra Pipeline – Downloading Digital RF data**

The downloading pipeline requires four major components:

1. The Xilinx RFSoc 2x2 Board
2. A computer to command the RFSoc Board connected to it
3. A computer to run the Redis Server
4. A computer to run the front-end Web application

2-4 can all be run on the same machine or on separate machines.

To run this pipeline:

#### I. Start the Redis Server

To start a Redis database, open a terminal that has Redis installed, navigate to this project's repository and run the following command:

```
redis-server ./redis.conf
```

You can specify configuration parameters for the database in **redis.conf**, which is located in the repository. For the Redis connection to be private and provide security to the data that is being streamed, a password for connecting to the server has been set. This password is located in the redis.conf file and can be changed as long as it is also changed in the config.yaml file located in the /frontend folder (for live streaming, the back-end configuration file is not being used, so it needn't be changed there as well). By default, Redis listens on 'localhost' on port 6379.

## II. Run the board script

First, obtain the IP address of the computer running the Redis server. You can find the public IP address of a computer by running **ipconfig** (on Windows) or **ifconfig** (on Unix) in a terminal. Alternatively, you can get your IP address by searching “What’s my IP” on Google. Then, in the **download.ipynb** file you created on the board in 5.3.1, enter the IP address in the line of code which sets up the Redis connection :

```
r = redis.Redis(host='155.41.48.219', port=6379, db=0,  
password='Qsh4r9.VlMj5_HrvY#0f36')
```

Run the cells by pressing the play button in each cell. The board is now available for data- and it is subscribed to all requests coming from the front end directed to it.

## III. Run the front end

First, set the IP address of the Redis server (which you found in step II) in **front\_end/config.yaml**. You can also configure the location of where the front end will be located. To start the front-end web application, run the following in a terminal window:

```
conda activate rfsoc  
python ./front_end/app.py
```

The web application should now be running in the location you specified in the configuration file and can be accessed via a web browser. By default, it should be at <http://127.0.0.1:8050/>. The front end sends a request to the board after a user fills out the download request form.

## **5.4 OPERATION OF THE PROJECT**

This section goes chronologically after the previous section, it explains the steps related to interactivity that the user needs to follow after the hardware and software have been installed and set up, i.e., it describes how to use the web application. It explains in detail the operating mode of the web application in response to how the user interacts with it, explaining the normal and abnormal operations of each of the modes or pipelines.

## 5.4.1 OPERATING MODE 1: NORMAL OPERATION

### 5.4.1.1 Normal RFSoc Streaming Operation

1. Follow the steps outlined in 5.3.2.2 to get the board and the web application running.
2. Open the web application in a browser. Navigate to the “Streaming” tab located in the top right of the dashboard.
3. In the sidebar, under the “Stream Options” accordion tab, the name of your stream should appear in the dropdown. Select it.
4. You should now be able to click “Play stream data” and “Pause” to play and pause data, respectively. These buttons are located under the “Stream Options” accordion tab.

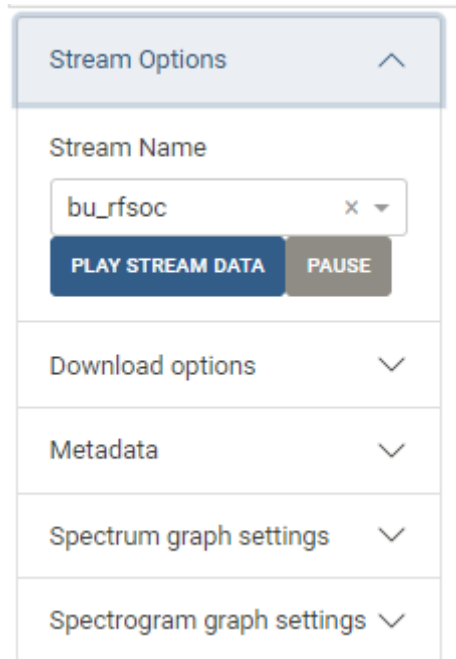
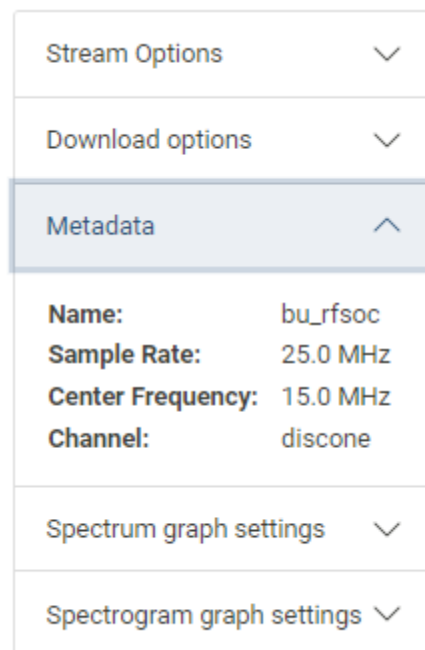


Figure 16 – Stream Options tab

Figure 16 shows the Stream Options tab, which displays the name of a board that is available for streaming, in this case, the name of the board selected is bu\_rfsoc, as well as the “play” and “pause” buttons.

- The metadata for the live RFSoc data should be displayed in the sidebar under the “Metadata” tab. Figure 17 shows an example of what the Metadata sidebar tab looks like after live streaming from a board begins.



*Figure 17 - Metadata tab*

- Additional settings for the graphs can be found under the “graph settings,” “spectrum graph settings,” and “spectrogram graph settings” accordion tabs. These settings include changing between linear and logarithmic scales choosing the vertical limits, displaying the maximum and/or minimum points of the spectrum, and changing the color scheme of the spectrogram plot.

#### **5.4.1.2 Normal Digital RF Playback Operation**

- Follow the steps outlined in 5.3.2.3 to get the web application running.
- Open the application in a browser. Make sure to stay on the “Digital RF” tab, shown in Figure 18.

### Spectrum Monitoring Dashboard

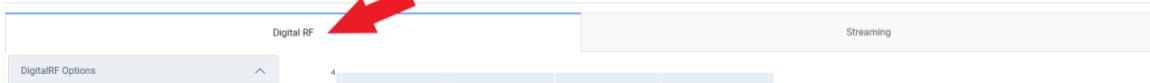


Figure 18 - Digital RF playback tab

3. Click the “Select Digital RF Data” button in the sidebar under the “Digital RF Options” accordion dropdown, the dropdown is shown in Figure 19. A modal form should pop up.

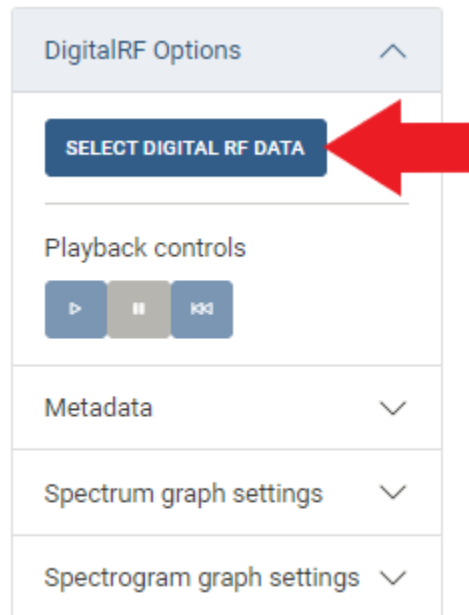
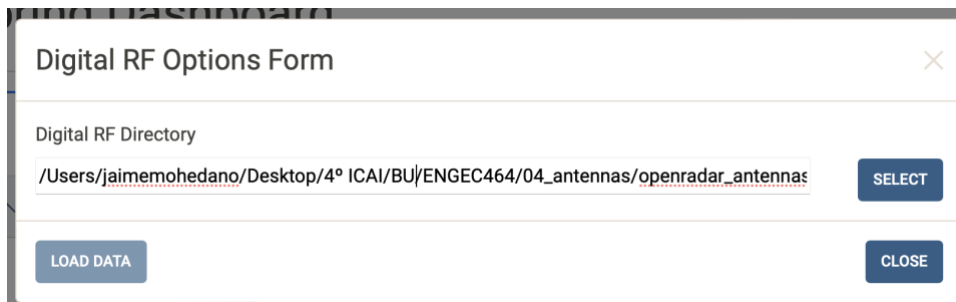


Figure 19 - Digital RF playback request form access

4. Enter the file path of the Digital RF data you wish to display in the form shown in Figure 20. Hit “select”.

NOTE: This file path is for the *back end*. In other words, the data being played back should be located in whatever machine is running the back end.

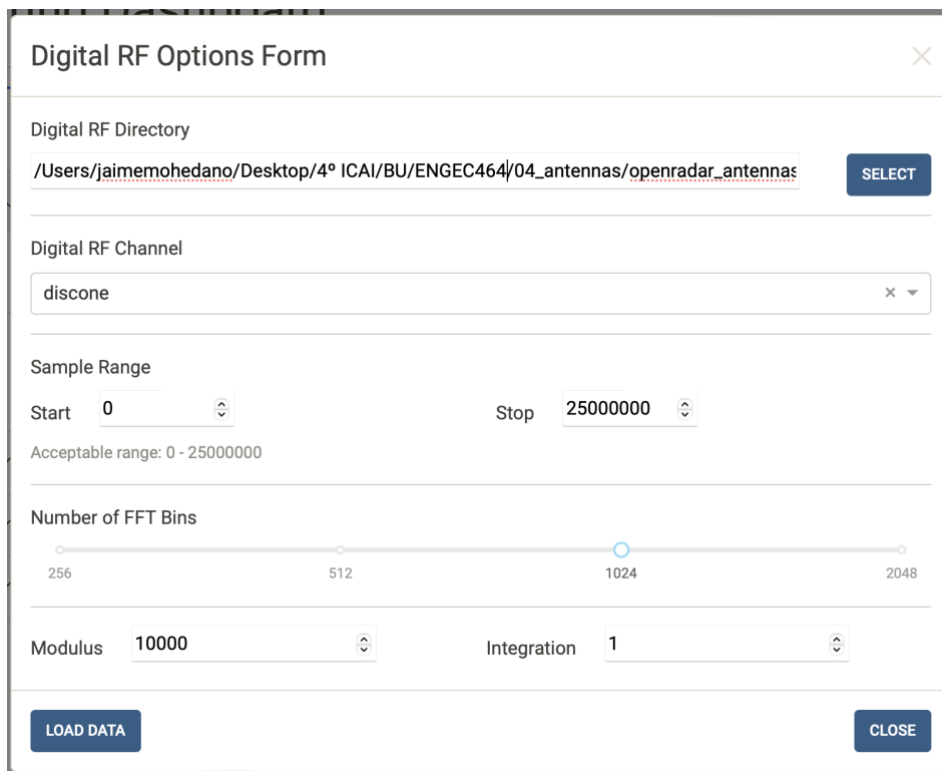


Digital RF Options Form

Digital RF Directory

Figure 20 - Digital RF playback request form initially upon opening

- Additional form options should pop up as shown in Figure 21. “Digital RF Channel” lets you select between the different channels found within the directory you selected in step 4. “Sample Range” lets you pick which samples to play in the file. For example, selecting 0-200k means the first 200,000 samples in the file will be played. “Number of FFT bins” represents the number of Fast Fourier Transform bins carried out on the raw data. “Modulus” and “Integration” represent how many samples get skipped over or averaged together, respectively. Hit “Load Data” to load data for playback. Hit “close” to cancel and close the form.



Digital RF Options Form

Digital RF Directory

Digital RF Channel

Sample Range

Start

Stop

Acceptable range: 0 - 25000000

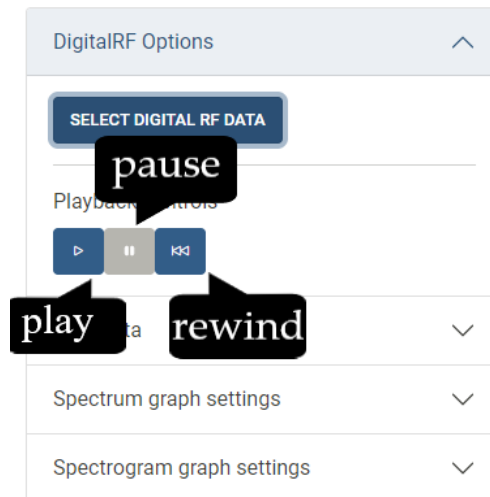
Number of FFT Bins

Modulus

Integration

Figure 21 - Digital RF playback form once a valid Digital RF file has been selected

6. You should now be able to play back data. To start playing data, under the “Digital RF Options” accordion tab, shown in Figure 22, press the play button. You can also pause and rewind the data back to the beginning using the corresponding buttons.



*Figure 22 - Play, pause, and rewind buttons under the “DigitalRF Options” tab*

7. The metadata for the digital RF file should be displayed in the sidebar under the “Metadata” tab, Figure 23.

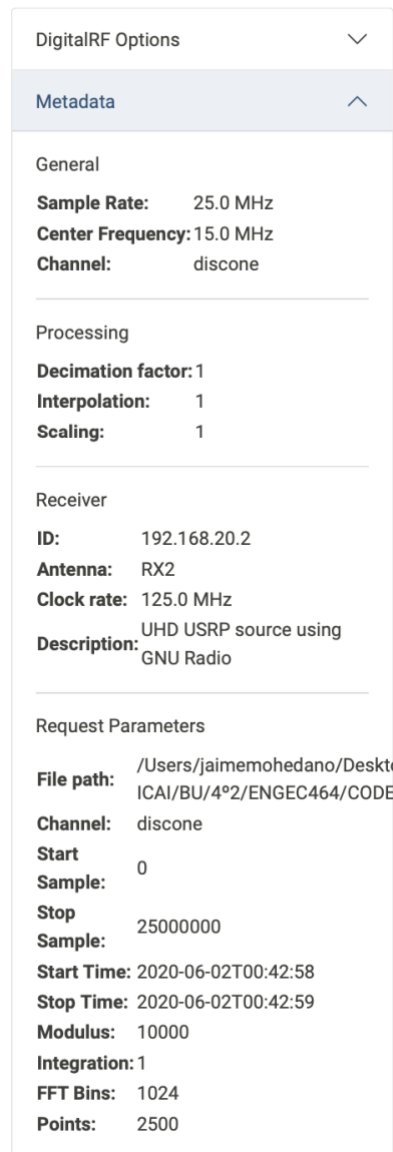


Figure 23 - Metadata of the Digital RF file shown under “Metadata” tab

8. Additional settings for the graphs can be found under the “graph settings,” “spectrum graph settings,” and “spectrogram graph settings” accordion tabs. These settings include changing between linear and logarithmic scales, choosing the vertical limits, displaying the maximum and/or minimum points of the spectrum, and changing the color scheme of the spectrogram plot.
9. To play back a different file, or the same file with different settings, repeat steps 3-5.



### 5.4.1.3 Normal Downloading Digital RF data Operation

1. Follow the steps outlined in 5.3.2.4 to get the board and the web application running.
2. Open the web application in a browser. Navigate to the “Streaming” tab located in the top right of the dashboard.
3. In the sidebar, under the “Download Options” accordion tab, click the “Download data from board” button, Figure 24.

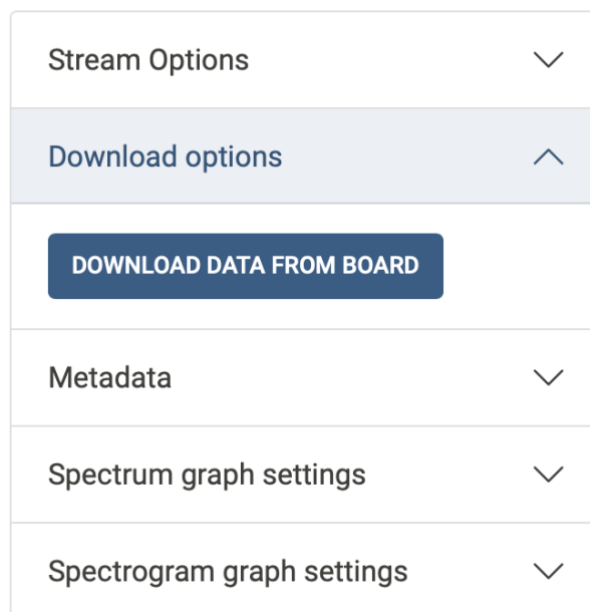


Figure 24 - Digital RF download request form access

4. The “Download data from board” form should pop up as in Figure 25. The board denotes that it's available for data downloading by adding its name. The name of your board should appear in the “Choose board” dropdown. Select it. You must enter the time duration (by default it is in seconds, but it can be changed to milliseconds or even microseconds) you want of data to be downloaded in the “Duration” option. Enter the name of the folder that will have the Digital RF data. Hit “Download Data”.

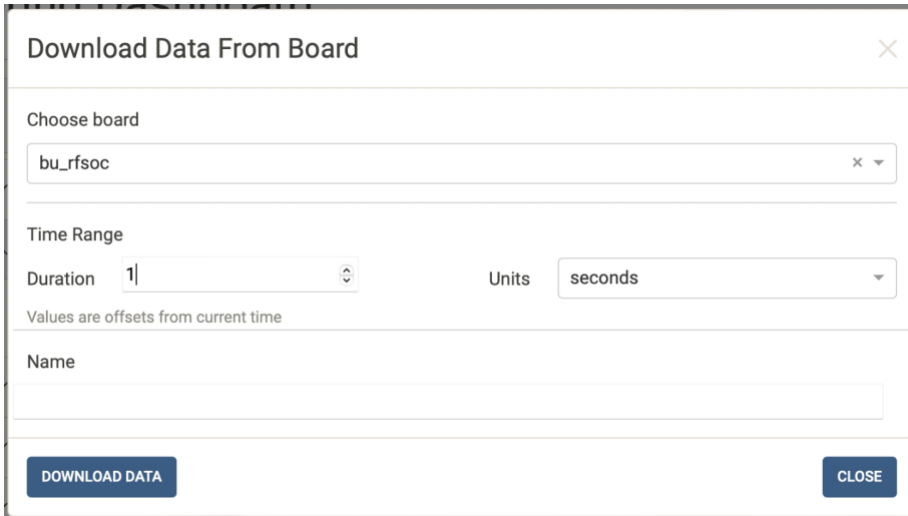


Figure 25 - Digital RF download request form initially upon opening

5. After a few seconds (depending on the time duration you entered), the browser should start downloading a zip file with the requested data in the “Downloads” folder of your PC. Extract the zip file and playback the Digital RF data contained within as described in section 5.4.1.2. In step 4 make sure to enter the uncompressed folder that was downloaded.

## 5.4.2 OPERATING MODE 2: ABNORMAL OPERATION

### Software Issues

If there are issues with major portions of the application, it is likely due to a disconnect between the front end, the Redis server, and the back end(s). Make sure that the Redis server and the back end(s) are running. Confirm that the front and back-end configuration parameters are pointing to the correct location of the Redis server.

Occasionally minor bugs can arise within the Dash application. One example of this is the y-axis of one of the graphs being misaligned with the data. Minor bugs happen most frequently when Dash drops a particular callback, or when the application is misused somehow. Most issues can be resolved by either refreshing the web page or rebooting the Dash application.

### Hardware Issues

Most of the issues related to JupyterLab can be resolved by restarting the kernel and re-running the notebook. If JupyterLab is not responding, restart the board by switching it off and back on again.

## 6 RESULTS

This section shows the results after installing, setting up, and running the web application. The steps to do everything mentioned are explained in the previous section.

### ***6.1 TEST 1: DIGITAL RF PLAYBACK***

This test consists of playing back pre-recorded Digital RF data, this data was originally provided by the client of the project. The setup consists of a single PC with three components:

- A Redis database
- A back-end python server
- A front-end Dash web application

The Redis database can be hosted on another PC but for simplicity it is running on the same PC as the other two components.

The back-end streams Digital RF data requested by the front end through a Redis server, the front end displays the frequency data on the web application on both a spectrum graph and a spectrogram graph.

In this test, the data that was played back was captured by a D130J antenna (the Digital RF channel selected is discone when choosing the Digital RF options from the dropdown menu of the form) from the narrowband VHF centered at 99.5 MHz. Again, this data was provided by the clients.

The file can be paused, played, and restarted with the corresponding effect on the graphs. The metadata can be seen in the designated accordion tab and the scaling can be toggled for both the spectrum and spectrogram between logarithmic and linear scales. The y-axis boundaries on both graphs can be modified and the minimum and maximum points on the spectrum graph can be tracked. Also, the color scale of the spectrogram graph can be changed.

Below, in Figure 26, an image of the test is shown. The spectrum graph (top) has been converted to a logarithmic scale, and the minimum and maximum points are being displayed (the blue and red crosses, respectively). The spectrogram graph (bottom) had its color scale changed part way through the playback.

## Spectrum Monitoring Dashboard

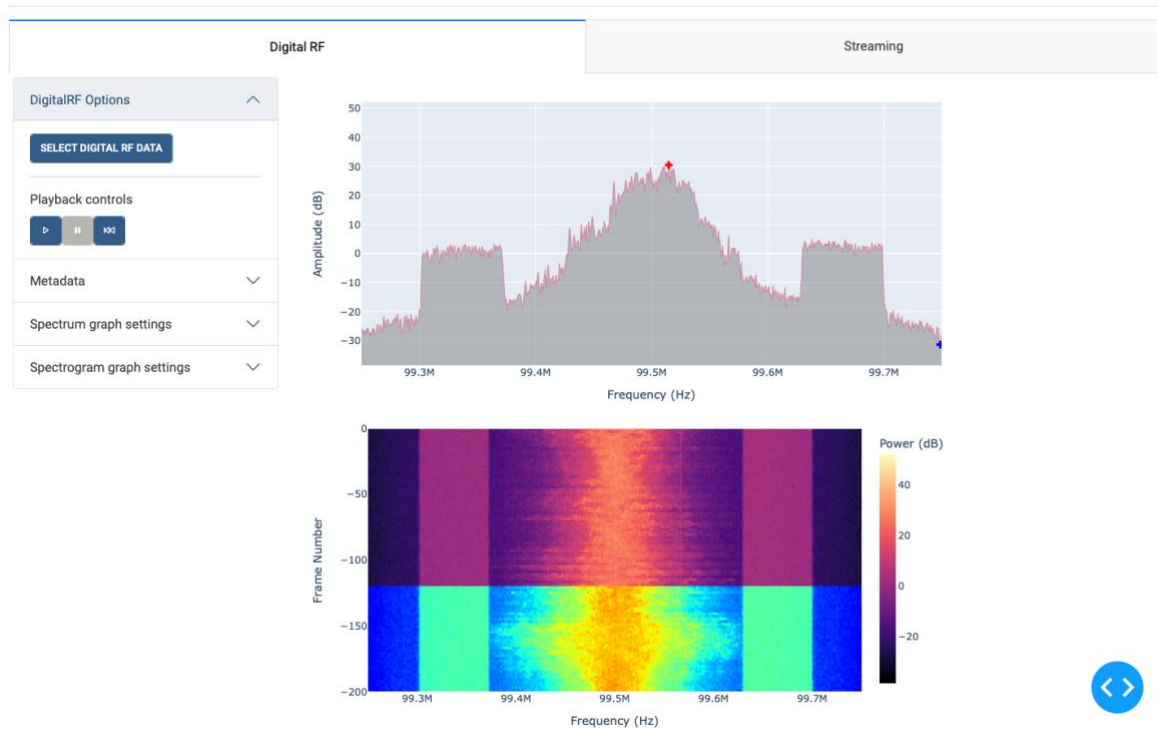


Figure 26 - Pre-recorded Digital RF shown on the web application

## 6.2 TEST 2: LIVE STREAMING RFSOC DATA

This test consists of live streaming data from the RFSoc board. The test will be divided into two sub-tests, both for live streaming data. The first one will live stream data from the board itself (from a DAC, transmitter) and receive it with an ADC (receiver). This is achieved by connecting both ADC and DAC of the same channel with an SMA cable, it is connected in loopback. The second sub-test will consist of live streaming data that is being captured with an antenna attached to the board.

Both setups consist of two main components:

- The hardware side consists of the RFSoc board loaded with a data collection JupyterLab script. One computer (PC1) is needed to connect to the board and run the necessary scripts on the board. When the board is powered on and running, the script takes incoming data, converts it to spectrum data using the FFT, and sends that data over ethernet into a Redis database running on a second computer (PC2).
- The software side consists of a PC2 running the Dash web application as well as the Redis database. This web application processes data streaming into the Redis database from the board and displays it in the browser with an interactive graph format.

### ***6.2.1 Streaming live data - Loopback mode***

In this case, the incoming data is generated by the script that is running on the board. First, the script transmits a tone at a certain frequency with a certain amplitude through one of the transmitters of the board. The receiver, which is connected to the transmitter that is generating the tone, receives the data, the board performs the corresponding time-frequency conversion with the FFT, and sends the data through Redis to the front end.

The transmitter was set, in this case, to output a tone at 600 MHz. The left panel is showing that the data is coming from the “bu\_rfsoc” stream. The peak in Figure 27 is clearly at 600 MHz, the frequency value that was set on the transmitter to generate a tone at. This means that the web application is correctly processing and displaying the data.

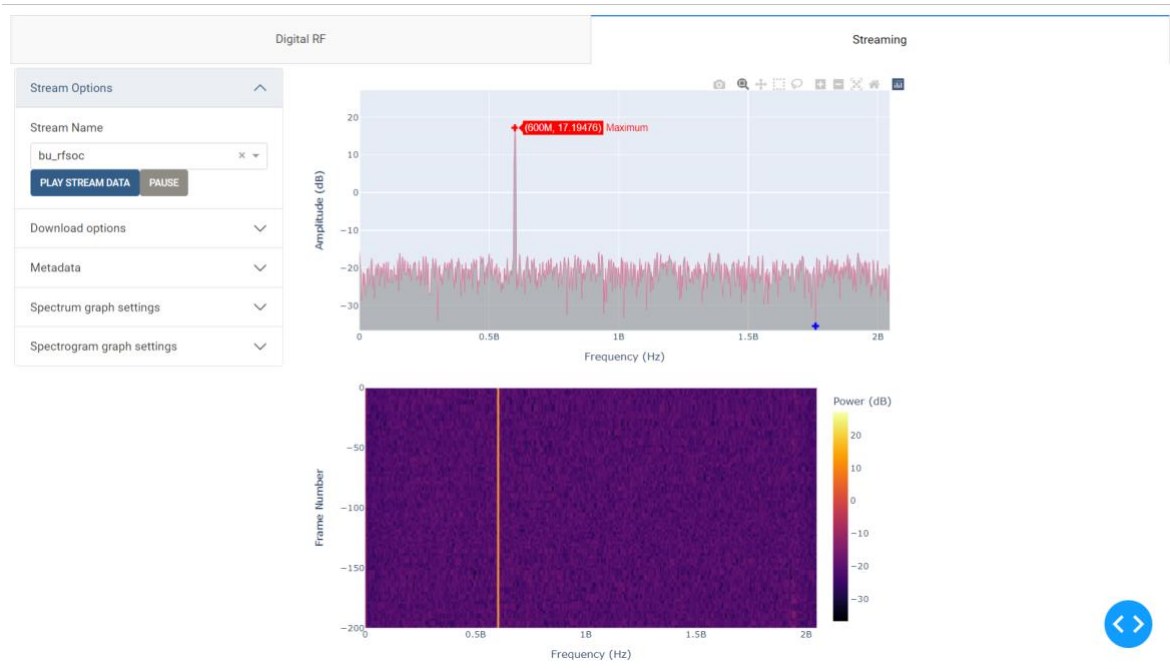


Figure 27 - Live RFSoc data - Loopback mode

### 6.2.2 Streaming live data - Captured with an antenna

In this case, the incoming data is captured with a rather basic antenna that was provided by the client for testing purposes, specifically the antenna was a USRP B200 antenna. The test works similarly to the previous example, the only variation is that the receiver is not connected to a transmitter but to an antenna. The antenna receives the data, the board performs the corresponding time-frequency conversion with the FFT and sends the data through Redis to the front end.

The result of this test is shown in Figure 28.

## Spectrum Monitoring Dashboard

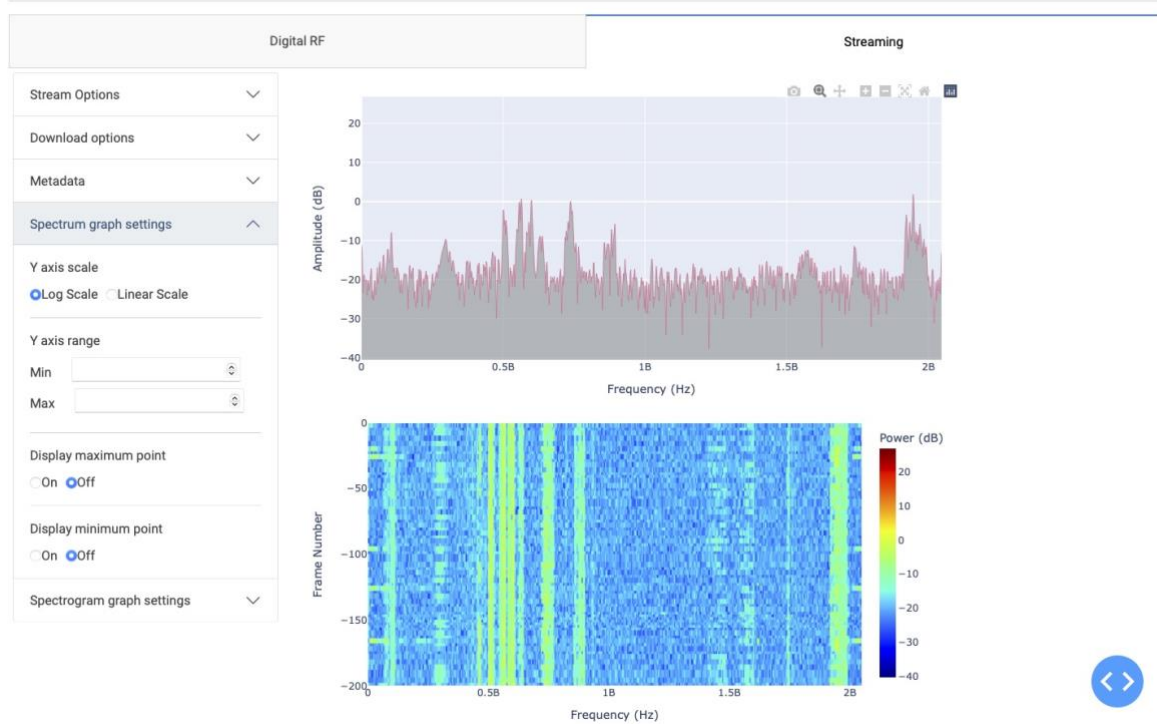


Figure 28 - Live RFSoc data - Captured with an antenna

### 6.3 TEST 3: DOWNLOADING AND PLAYING BACK DIGITAL RF DATA

This test consists of downloading live data from the RFSoc board in Digital RF format and then playing it back. The test will be divided into two sub-tests. The first one will download live data that is transmitted from the board itself (from a DAC, transmitter) and receive it with an ADC (receiver). The second sub-test will consist of downloading live data that is being captured with an antenna attached to the board.

The setup involves a combination of the setups from Tests 1 and 2:

- The hardware side consists of the RFSoc board loaded with a data collection JupyterLab script. One computer (PC1) is needed to connect to the board and run the necessary scripts on the board. When the board is powered on and running, the script waits for an incoming request, records the requested data, and dumps the raw



recorded data over ethernet into a Redis database running on a second computer (PC2).

- The software side consists of PC2 running the Dash web application and the Redis server. The web application downloads data requested from the board in a ZIP file, the contents of which can be played back as in Test 1.

### 6.3.1 Downloading live data - Loopback mode

In this case, the incoming data is generated by the script that is running on the board. First, the script transmits a tone at a certain frequency with a certain amplitude through one of the transmitters of the board. The receiver, which is connected to the transmitter that is generating the tone, receives the raw IQ data and sends it through Redis to the front end.

In this test, 3 seconds of data from the RFSoc were requested, shown in Figure 29. The RFSoc had an RX channel in loopback with a TX channel. For this test, the TX channel was set to output a tone at 750 MHz. The RFSoc processed that request and dumped the raw data into the Redis streams.

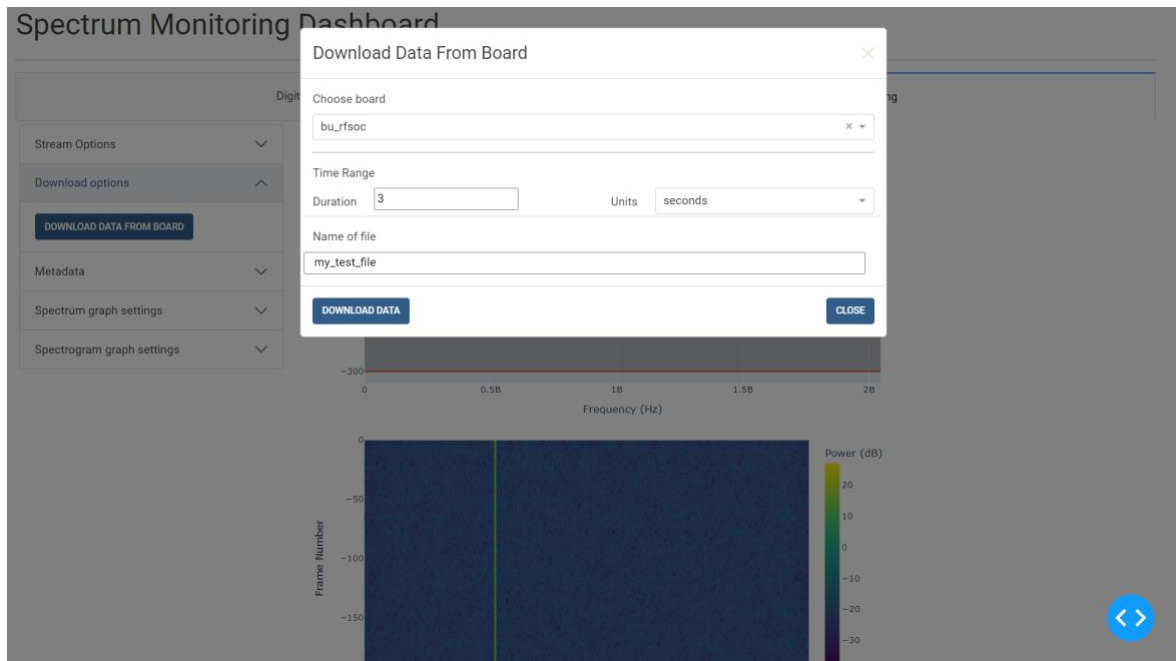


Figure 29 - Data request form – Loopback mode

Note: it should be mentioned that to transmit the data from the back end to the front end it must be serialized as it is explained in section 2.7 when describing JSON. You can't serialize complex data with JSON (raw IQ data is complex data), so the data in this Pipeline is being transmitted through Redis with two streams, one for the real part of the data and one for the imaginary part.

The front end got the data from the Redis server, converted it into a digital RF file, and pushed it to the user's browser, where it was downloaded as a ZIP file. The zip file is extracted, and then its contents are played back in the Digital RF playback section of the web application as Figure 30 shows.

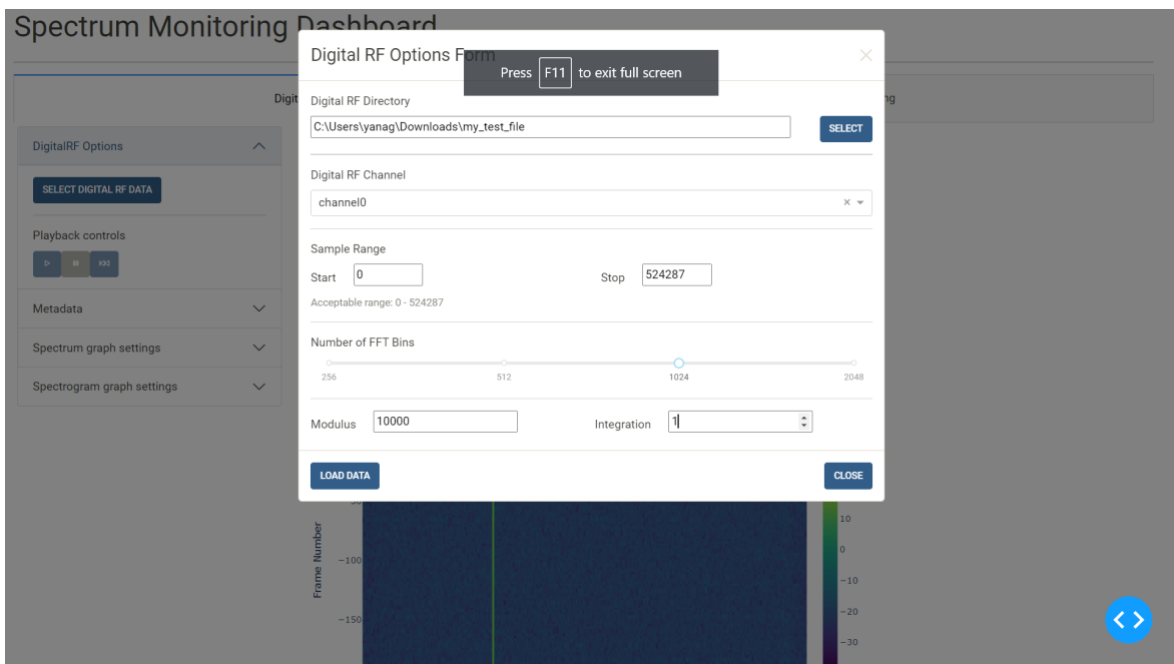


Figure 30 - Form for playing back the Digital RF data downloaded in Figure 29



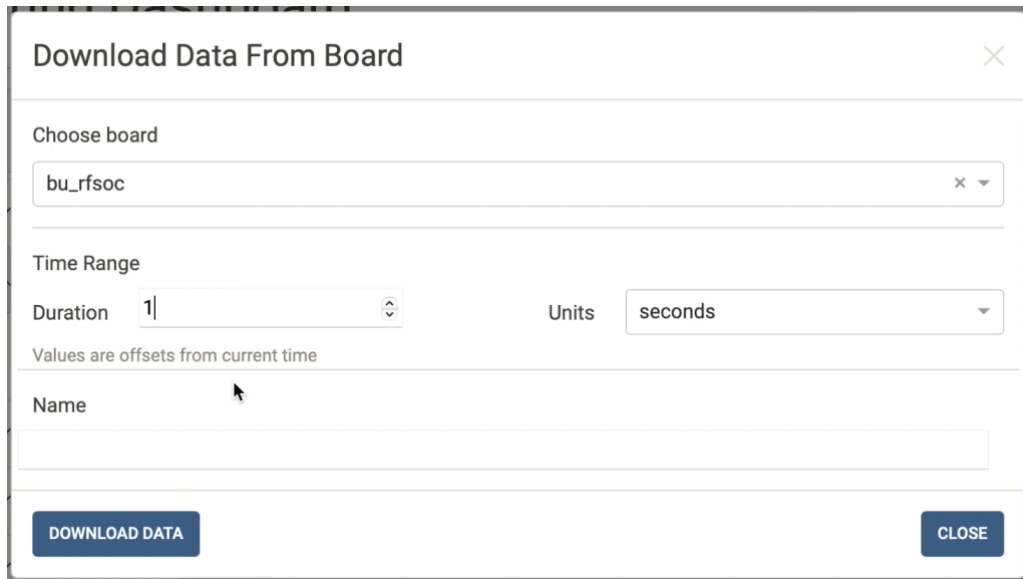
Figure 31 – Web application with the data downloaded in Figure 29 being played back

In Figure 31 a peak at 750 MHz is clearly visible, meaning that the data has likely been transmitted and played back accurately.

### 6.3.2 Downloading live data – Captured with an antenna

In this case, the incoming data is captured with the same antenna as in 6.2.2, a USRP B200 antenna. The test works similarly to the previous example, the only variation is that the receiver is not connected to a transmitter but to an antenna. The antenna receives the raw data, and the board sends it through Redis to the front end.

Figure 32 shows how 1 second of data is requested from the board “bu\_rfsoc”.



**Download Data From Board**

Choose board  
 bu\_rfsoc

Time Range  
 Duration 1 Units seconds

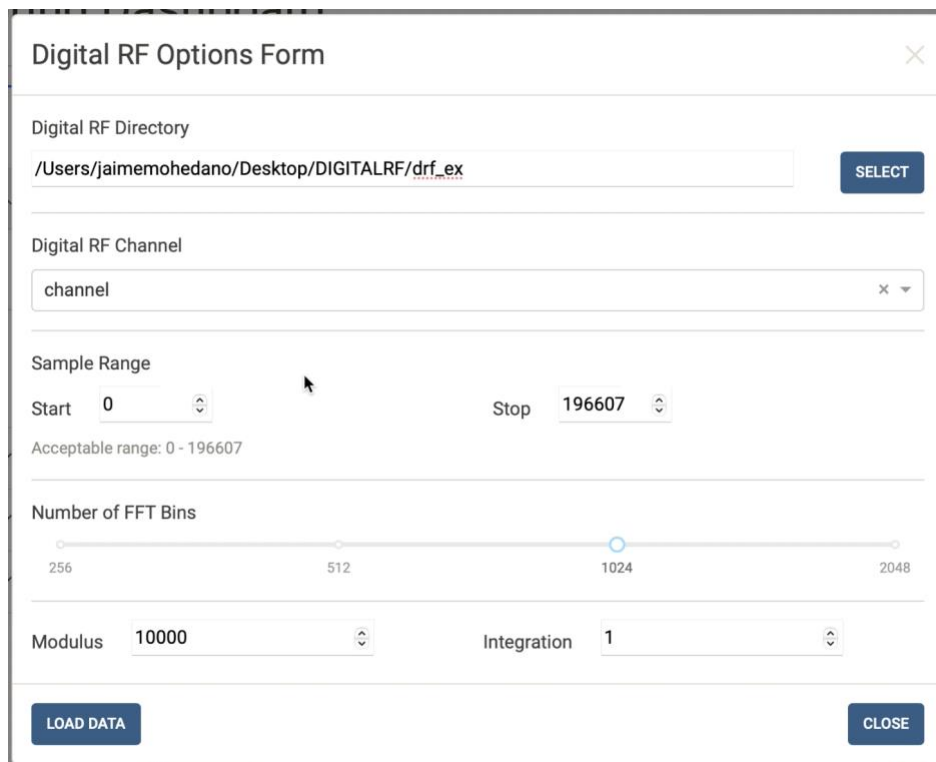
Values are offsets from current time

Name

DOWNLOAD DATA CLOSE

Figure 32 - Data request form – Captured with an antenna

Figure 33 shows how the file downloaded (drf\_ex by default as a Name was not specified) is played back as in the previous example.



**Digital RF Options Form**

Digital RF Directory  
 /Users/jaimemohedano/Desktop/DIGITALRF/drf\_ex

Digital RF Channel  
 channel

Sample Range  
 Start 0 Stop 196607

Acceptable range: 0 - 196607

Number of FFT Bins  
 256 512 1024 2048

Modulus 10000 Integration 1

LOAD DATA CLOSE

Figure 33 - Form for playing back the Digital RF data downloaded in Figure 32

The final result shown in Figure 34 shows spectrum monitoring when playing back the downloaded data. The data was captured with an antenna.

## Spectrum Monitoring Dashboard

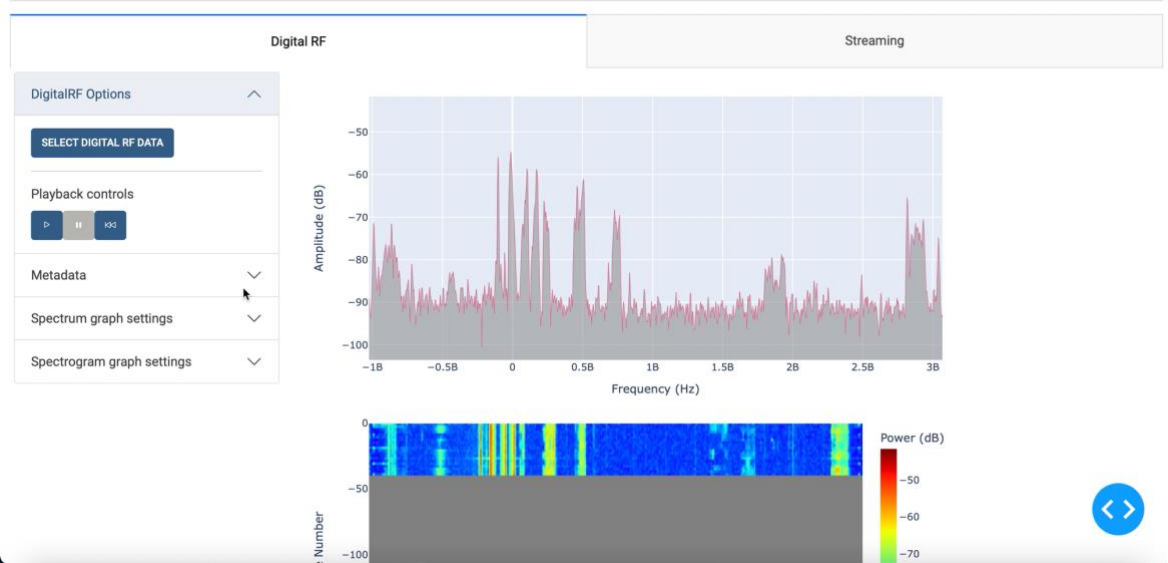


Figure 34 - Web application with the data downloaded in Figure 32 being played back

## 7 CONCLUSIONS AND FUTURE WORK

The web application created is a practical base that can be extended as needed to serve future research needs. It offers well-documented, easy-to-use tools for RF environment monitoring. The direction of the project has shifted several times since the beginning, but this was expected since the project has always been exploratory in nature and has adapted to the requests/needs of the client.

However, the current state of the project involves some known issues that may need some time to work on so that the user experience is better, these known issues are listed below:

- The error handling for the application is not robust. Bugs and errors can and do happen, especially if the user misclicks or inputs some invalid values. If a user clicks wildly at the graph options in the sidebar, the graphs are likely to behave oddly. For example, toggling many times between the logarithmic and linear scales may result in the y-axis bounds being misaligned. The best way of dealing with an issue is to refresh the page or restart the components involved (the Dash application, the back-end scripts, etc.)
- Sometimes, a Dash callback gets dropped, and as such the action involved does not get completed. This can lead to misaligned graph axes, a data point being missed on the graph, information not being populated, etc. The fix for this is to just repeat the action again or to refresh the page if the error is bad enough. This is likely to happen due to Dash throttling callbacks when the rate gets too high, a consistent fix has not been found.
- The way the data is being pulled out of the board uses the following command:  

```
base.radio.receiver.channel[board_channel].transfer(number_samples)
```

with base being an instance of the PYNQ Base Overlay class. However, this function only lets you transfer a few tens of thousands of samples at a time, and takes about 200 milliseconds to run, making it a poor candidate for downloading a large chunk of data (eg over 0.5 seconds). A better way of downloading data from the board should be investigated.

Together with the known issues, there are some tasks or topics that can be worked on in the future with the same objective, to improve the user experience and, also to broaden the capabilities of the web application. Some of these tasks are listed below:

- Currently, live streaming data from the board is entirely "passive". This means that the front-end web application is not able to set any parameters for the board (center frequency, bandwidth, etc). These parameters are set by the board itself. The front end passively receives the data dumped by the board into the Redis server.
- Additional filtering and processing scripts for the RFSoc can be added, including:
  - o Streaming a band and time-limited set of raw IQ data
  - o Digital down-converting the data
- At the moment, downloading data from the board and live streaming board data require two different scripts for the board. This means that only one of these features can be available at any given time, depending on which script is being run. In the future, it would be good to combine these features into a single script.
- At this point in time, this project was developed and tested only by handling one user at a time. Being able to process multiple users will likely involve having to restructure certain portions of the codebase. Specifically, the global variables found in `front_end/config.py` will have to be either cached or stored locally per user for this application to work for multiple users. The most difficult challenge will likely be to restructure the Spectrum and Spectrogram classes (or their corresponding data) to be stored/cached.
- Extending the web application to work with other software-defined radios. Almost all of the code written for this project is board-agnostic. As such, it should be easy to expand the web application to work with any SDR device.

Even so, the features implemented work as a solid framework from which these additional features could quickly be implemented. Therefore, the overall goal of creating an extendable “base” application was accomplished.

## 8 REFERENCES

- [1] Radio spectrum - A key resource for the Digital Single Market. European Parliament. March 2015
- [2] A. Leshem and A. -. van der Veen, "Radio-astronomical imaging in the presence of strong radio interference," in *IEEE Transactions on Information Theory*, vol. 46, no. 5, pp. 1730-1747, Aug. 2000, doi: 10.1109/18.857787.
- [3] A. M. Wyglinski, D. P. Orofino, M. N. Ettus and T. W. Rondeau, "Revolutionizing software defined radio: case studies in hardware, software, and education," in *IEEE Communications Magazine*, vol. 54, no. 1, pp. 68-75, January 2016, doi: 10.1109/MCOM.2016.7378428.
- [4] Veen, Alle-Jan & Wijnholds, Stefan. (2013). Signal Processing Tools for Radio Astronomy. 10.1007/978-1-4614-6859-2\_14.
- [5] RFSoc 2x2 board image. Retrieved the 23rd of October 2021 from <https://www.xilinx.com/support/university/xup-boards/RFSoc2x2.html>
- [6] RFSoc 2x2. Retrieved the 22nd of October 2021 from <https://www.rfsoc-pynq.io>
- [7] Introduction to Dash. Retrieved the 22nd of October 2021 from <https://dash.plotly.com/introduction>
- [8] Introduction to Redis. Retrieved the 22nd of October 2021 from <https://redis.io/docs/about/>
- [9] Redis streams. Retrieved the 22nd of October 2021 from <https://redis.io/docs/manual/data-types/streams/>
- [10] MIT Haystack Observatory's Digital RF GitHub repository. Retrieved the 18th of October 2021 from [https://github.com/MITHaystack/digital\\_rf](https://github.com/MITHaystack/digital_rf)
- [11] PYNQ: Python productivity. Retrieved the 23rd of October 2021 from <http://www.pynq.io>
- [12] University of StrathClyde open-source GitHub repository. Retrieved the 12th of October 2021 from <https://github.com/strath-sdr>
- [13] Introducing JSON. Retrieved the 22nd of October 2021 from <https://www.json.org/json-en.html>
- [14] Structures of JSON. Retrieved the 22nd of October 2021 from <https://www.w3resource.com/JSON/structures.php>
- [15] Jupyter. Retrieved the 22nd of October 2021 from <https://jupyter.org>



- [16] Introducing JupyterLab. Retrieved the 22nd of October 2021 from <https://ipython-books.github.io/36-introducing-jupyterlab/>
- [17] Software-defined radio. Retrieved the 12th of October 2021 from [https://en.wikipedia.org/wiki/Software-defined\\_radio](https://en.wikipedia.org/wiki/Software-defined_radio)
- [18] Software Defined Radio. Retrieved the 1st of June 2022 from <https://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf>
- [19] OpenWebRX web-based software-defined radio. Retrieved the 13th of October 2021 from <https://www.openwebrx.de/news.php>
- [20] Running OpenWebRX on balena to remotely monitor local radio spectrum. Retrieved the 1st of June 2022 from <https://bigpi.vc/running-openwebrx-on-balena-to-remotely-monitor-local-radio-spectrum/>
- [21] Introduction to using the KiwiSDR. Retrieved the 22nd of October 2021 from [http://kiwisdr.com/ks/using\\_Kiwi.html](http://kiwisdr.com/ks/using_Kiwi.html)
- [22] sdrsharp. Retrieved the 14th of October 2021 from <https://www.rtl-sdr.com/tag/sdrsharp/>
- [23] GNU Radio. Retrieved the 14th of October 2021 from [https://en.wikipedia.org/wiki/GNU\\_Radio](https://en.wikipedia.org/wiki/GNU_Radio)
- [24] OS X port of the awesome gqrx SDR software. Retrieved the 23rd of October 2021 from <https://eliasoenal.com/2012/09/30/osx-port-of-the-awesome-gqrx-sdr-software/>
- [25] Mini-circuits webpage. Retrieved the 15th of March 2022 from <https://www.minicircuits.com>
- [26] Anaconda download. Retrieved the 15th of November 2021 from <https://anaconda.org/>.
- [27] Redis download. Retrieved the 15th of November 2021 from <https://redis.io/download>.
- [28] Sustainable Development Goals. Retrieved the 5th of July 2022 from <https://www.un.org/sustainabledevelopment/>
- [29] Azote Images for Stockholm Resilience Centre (CC BY 4.0)
- [30] The SDGs wedding cake. Retrieved the 5th of July from <https://www.stockholmresilience.org/research/research-news/2016-06-14-the-sdgs-wedding-cake.html>
- [31] Sustainable Development Goals: “better model for the future”. Retrieved the 5th of July from <https://blog.dgnb.de/en/sdgs-interview-buckley-part-1/>
- [32] Measuring progress towards the Sustainable Development Goals. Retrieved the 5th of July from <https://sdg-tracker.org/>

- [33] Keeping the Internet up and running in times of crisis. Retrieved the 5th of July from <https://www.oecd.org/coronavirus/policy-responses/keeping-the-internet-up-and-running-in-times-of-crisis-4017c4c9/>

## ANNEX I: INTEGRATION OF THE SDGs INTO THE PROJECT

The Sustainable Development Goals (SDGs) are a collection of 17 global goals designed to achieve a more sustainable future. They were adopted by the United Nations in 2015 as a universal call for action by all countries to promote prosperity while protecting the planet [28]. The SDGs aim to address the global challenges we face, including poverty, climate change, peace, and justice, among others. They are intended to be achieved by 2030.

Shortly before the official publication of the SDGs, Professor Rockstrom, a recognized scientist for his work on global sustainability, and partners presented a model I find interesting, the “wedding cake” model for the SDGs. The diagram that represents this model is shown in Figure 35.

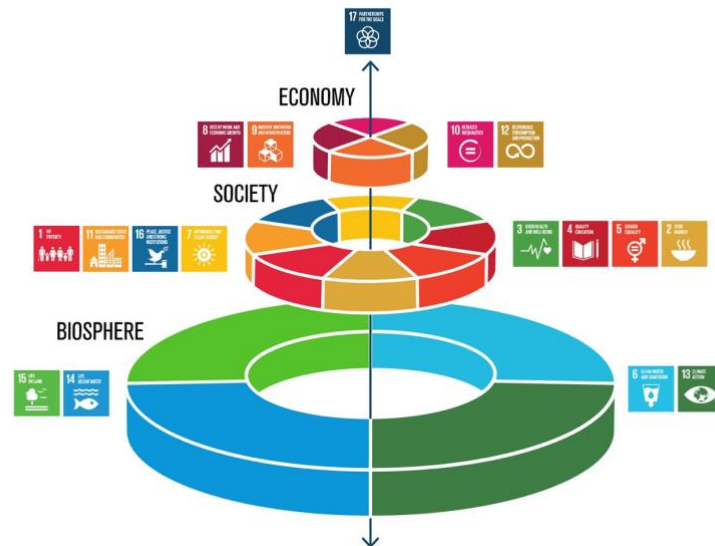


Figure 35 - The wedding cake model for SDGs diagram [29]

The pie model describes how economies and societies should be seen as embedded parts of the biosphere. This approach differs from the current vision in which the social, economic, and ecological areas are seen as independent. An integrated view of social, economic, and ecological development is supported by such a conception [30].

In other words, the illustration shows that the basis of all Sustainable Development Goals is the biosphere, which provides all the resources we need to live. We should do everything in our power to maintain it, to meet the basic needs of the next level, society. Similarly, society is the basis for a well-functioning economy [31].

Although it may seem difficult to relate a software-related project with sustainable goals, if we think more deeply, the ultimate goal of the project is to monitor the radiofrequency spectrum to help with research on mitigating interferences. The project's objective is to help the client, MIT Haystack Observatory, thus, it is directed to the radio astronomy community. However, the radio frequency spectrum is not uniquely used by that community. This web application can be used to monitor the spectrum independently of the user's community.

The number of interferences happening is exponentially increasing every day because of the enormous quantity of data that needs to be transmitted in the frequencies of the radiofrequency spectrum. Some examples of necessary transmissions of data might be radio and television broadcasting, satellites, defense, emergency services, and many more.

So, the problem that is clearly happening is the congestion of the spectrum. The web application developed in this project aims to monitor the spectrum, experts and scientists (of any field) can take advantage of the project and identify what signals are relevant for a specific use case and what signals are interferences. It should be mentioned that depending on the communication, a signal might be an interference or relevant for the analysis. For example, if I'm listening to the Spanish radio "Onda Cero" at 98 MHz, I might find relevant a signal at the same frequency, but, a TV retransmission at 578 MHz of the "Antena 3" TV channel is an interference. On the contrary, if I'm watching that exact TV channel, the signal is not an interference.

With that being explained, data transmission is everywhere, thus, the SDGs that this project, I believe, is more related to, are shown in Table 2:

SDG identified	SDG Dimension	Role
8 - Decent work and economic growth	Economy	Primary
9 - Industry, innovation and infrastructure	Economy	Primary
12 – Responsible consumption and production	Economy	Secondary

*Table 2 - SDGs integrated into this project. SDGs obtained from [28]*

### **SDG #8 – Decent work and economic growth**

The goal this project achieves is target 8.2: Diversify, innovate and upgrade for economic productivity. The UN has defined it as: “Achieve higher levels of economic productivity through diversification, technological upgrading, and innovation, including through a focus on high-value-added and labor-intensive sectors by 2030.” [32].

Since the start of the COVID-19 crisis, demand for broadband communication services has soared, with some operators experiencing as much as a 60% increase in Internet traffic compared to before the crisis. Most networks are coping with the increased demand and changes in utilization patterns with peak periods being stretched out during the day as well as the evening [33].

Telefónica reports nearly 40% more bandwidth in Spain, with mobile traffic growth of 50% and 25% in voice and data, respectively. Cisco Webex, the most prevalent cloud-based videoconferencing application, is peaking at 24 times higher volume.

This project fosters technological upgrading and innovation, by monitoring the radiofrequency spectrum, not focused on the radio astronomy community but on the spectrum itself, i.e., in all communities. Monitoring leads to identifying and mitigating irrelevant signals for a specific use case. So, the project is contributing to improving the efficiency of the allocation of resources/frequencies present in the spectrum, thus, new complex technologies, which will most likely be more bandwidth-consuming, can emerge.

## **SDG #9 – Industry, innovation and infrastructure**

The project integrates SDG number 9, achieving “Universal access to information and communications technology” with target 9.C. Its definition is as follows: “Significantly increase access to information and communications technology and strive to provide universal and affordable access to the Internet in the least developed countries by 2020.”

As it was mentioned in the previous SDG, the project helps with the allocation of frequencies, allowing for more signals to be transmitted, thus, more data. Consequently, increasing access to communications and the internet in general, including in the least developed countries.

Furthermore, related to digital transformation. Data is a key pillar because every interaction in the digital world generates data. This data provides a good indicator of progress. The more data that can be generated/transmitted, the more confidence the businesses owners will have to adopt digital technology in the least developed countries that have not yet started with the digital transformation.

## **SDG #12 – Responsible consumption and production**

Lastly, the Project secondarily integrates SDG number 12, responsible consumption and production. The goal this project achieves is target 12.2: “Sustainable management and use of natural resources”, which has the following definition: “By 2030, achieve the sustainable management and efficient use of natural resources.”

To understand how the project achieves this goal, it can be explained as a chain of events as follows. The web application provides a radiofrequency monitoring tool, as it has been explained, data transmission happens in these frequencies. The better the control over what is being transmitted at each frequency is, the better the mitigation can be done. This derives in less physical infrastructure that is needed to be deployed, thus, less energy consumption, which contributes to a more sustainable and efficient use of the natural resources that are used to generate energy.

## **ANNEX II: DEV/BUILD TOOL INFORMATION**

This project was tested using the following package versions on a PC running Windows 11:

- python 3.9.7
- dash 2.0.0
- dash-bootstrap-components 1.0.1
- dash-core-components 2.0.0
- dash-html-components 2.0.0
- digital\_rf 2.6.7
- matplotlib 3.4.3
- numpy 1.21.2
- orjson 3.6.7
- pyyaml 6.0
- redis 3.5.3
- scipy 1.7.1

The project was also tested using the following package versions on a PC running macOS Monterey Version 12.3.1:

- python 3.9.7
- dash 2.0.0
- dash-bootstrap-components 1.0.1
- dash-core-components 2.0.0
- dash-html-components 2.0.0
- digital\_rf 2.6.7
- matplotlib 3.4.3
- numpy 1.21.4
- orjson 3.6.7
- pyyaml 6.0
- redis 4.2.1
- scipy 1.7.2