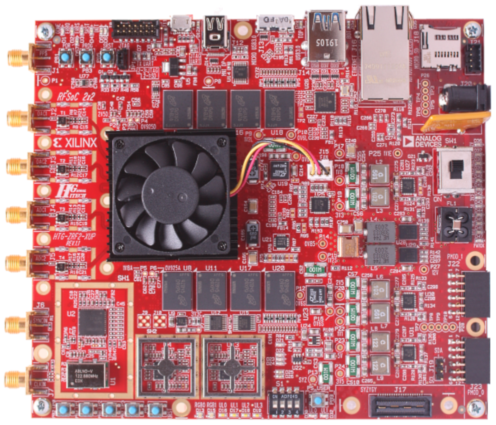
Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

User's Manual

RFSoC For RF Environment Monitoring

Submitted to

John Swoboda
swoboj@mit.edu
Sharanya Srinivas
ssrini@mit.edu



by

Team # 26
Team RFSoC

Team Members

Yana Galina yanag@bu.edu
Jaime Mohedano jaimem@bu.edu
Kakit Wong kakit@bu.edu
Isaac Yamnitsky isaacy@bu.edu

Submitted: 4/18/22

RFSoC for Environment Monitoring

Table of Contents

Executive Summary (Yana)

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. We have created an interactive Web application meant for RF spectrum monitoring. Our application can display the RF spectrum from a variety of RF sources, and is also capable of parsing and storing RF data. Our main focus for this project has been on the application of the Xilinx Radio Frequency System-on-Chip device to RF spectrum monitoring, as well as other techniques of interest to current interference mitigation research. We also handle previously recorded data in the Digital RF format.

1 Introduction (Yana)

The radio frequency (RF) spectrum is becoming increasingly congested due to usage from both the public and private sectors. This congestion creates challenges for researchers in the geospace and radio astronomy communities, as congestion makes it difficult to collect high-fidelity measurements. Radio Frequency Interference (RFI) mitigation techniques are essential to carry out work in these fields. Our clients at MIT Haystack Observatory are interested in a wide variety of RFI mitigation techniques relating to cancelation 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 high bandwidth, and the recent progression of Software-Defined Radio (SDR) has the potential to fill this gap. SDR is becoming increasingly popular for RF applications due to its potential for rapid design cycles and hardware reusability. SDR systems are characterized by having components implemented in software which traditionally have been implemented in hardware. SDR allows for more flexibility in changing radio parameters such as bandwidth or center frequency.

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. 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 deployment of this technology in many different areas, furthering its potential use cases for radio astronomy research.

The ultimate goal of our project was to create an interactive web application that can be used in conjunction with the Xilinx RFSoC board for a variety of RF spectrum monitoring tasks.

Our project has split into two separate but related data pipelines. In one pipeline, our application interacts with the RFSoC board. Our application is capable of live streaming data from the board, as well as downloading data from the board in the Digital RF format. DigitalRF is a standardized format for reading and writing RF data and was developed by our clients at MIT Haystack Observatory. Although our team has only worked with the RFSoC board, almost all of the code written for this project is board-agnostic. As such, it should be easy for our application to be expanded to work with any SDR device.

The second data pipeline of our application involves playback of Digital RF data. Our application allows the user to choose which Digital RF file to play along with additional configurable options, such as which samples within the file to display, and to what extent resulting data points should be averaged out.

Both pipelines display data using two different graphs: A spectrum plot, and a waterfall spectrogram plot. Both of these graphs display frequency-domain RF data, with frequency of a signal being plotted against its spectral power.

Both pipelines allow graph interactivity. The user can toggle between logarithmic and linear scales, adjust the y-axis bounds for the graphs, track maximum and minimum points on the spectrum graph, and change the color scale of the spectrogram graph.

This project has ultimately been exploratory in nature. We aimed to create a flexible base which can be extended as needed to serve future research needs, and we hope that we achieved this goal!

2 System Overview and Installation (Yana, Jaime)

2.1 *Overview block diagram*

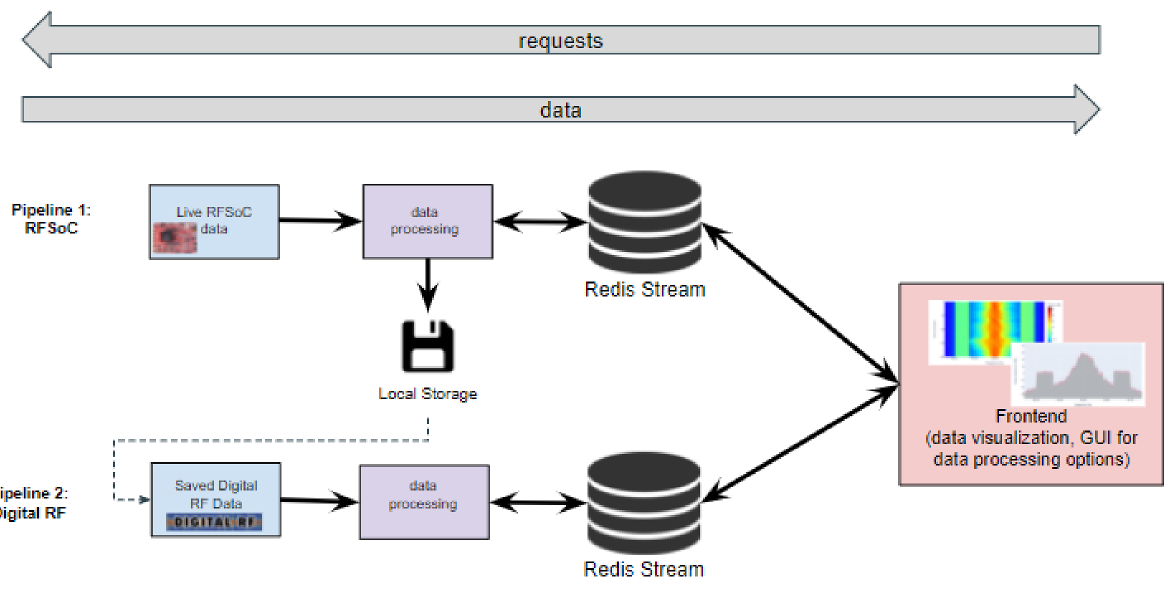
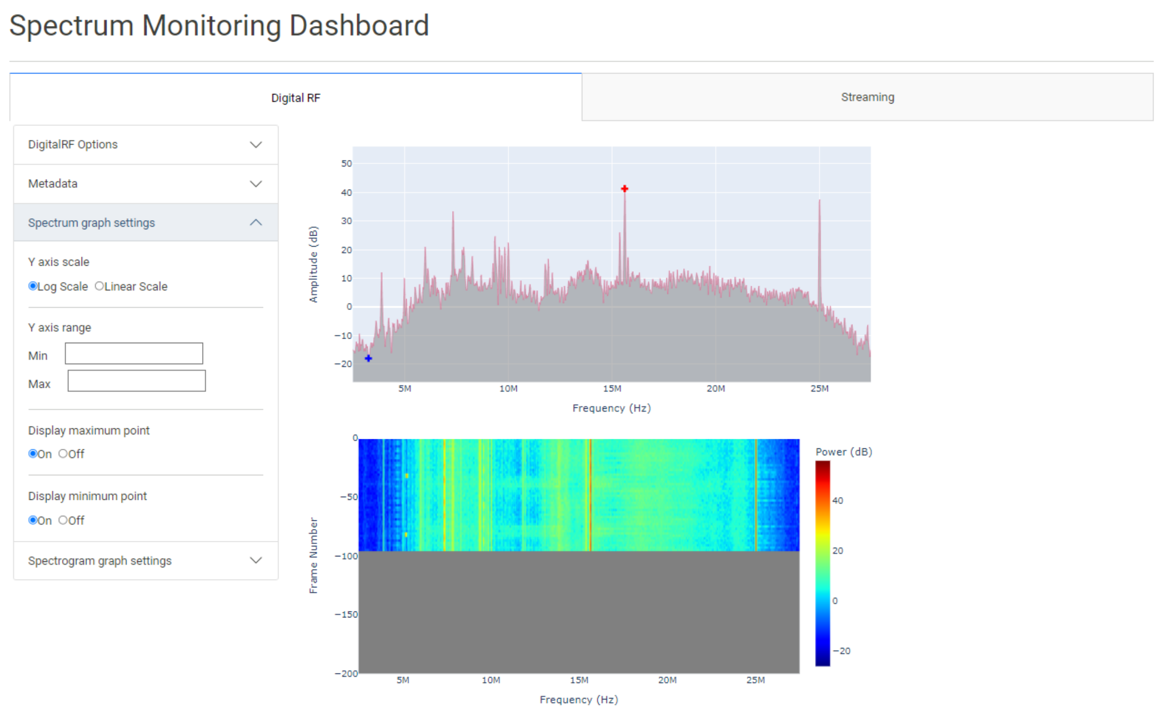


Figure 2.1 Block Diagram of our system. The diagram shows the flow of data between the front end and the different back-end data pipelines.

2.2 *User interface.*


Spectrum Monitoring Dashboard

Digital RF | Streaming

DigitalRF Options

Metadata

Spectrum graph settings

Y axis scale
[✓ Log Scale] Linear Scale

Y axis range
Min
Max

Display maximum point
[✓ On] Off

Display minimum point
[✓ On] Off

Spectrogram graph settings

Figure 2.2 DigitalRF Playback UI. This diagram shows the UI of our digital RF playback system.

2.3 Physical description.

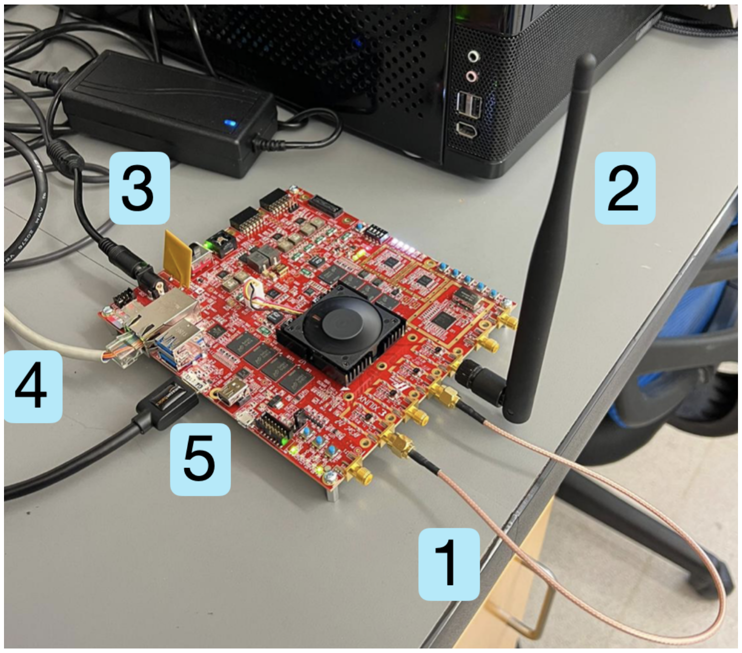


Figure 2.3 RFSoC board setup

The image above shows the RFSoC board setup that our team has used. It includes:

1. SMA cable: This cable connects, for the same channel, the transmitter (DAC) to the receiver (ADC). This setup is used for testing purposes. Connecting one transmit channel to a receive channel allows us to control exactly what data goes into the receive channel, so that we can easily test whether or not the data being displayed on our application is accurate.
2. Basic SDR antenna: We connect an antenna to the 2nd receive channel to be able to monitor the RF environment.
3. Power cable.
4. Ethernet cable: This cable connects the board to the internet. It can be connected to a router or to a PC with internet access.
5. Micro USB cable: This cable connects the board to the PC which we use to control the board. Specifically, the PC accesses JupyterLab on the RFSoC, which allows us to run Python scripts on the board.

2.4 Installation, setup, and support

2.4.1 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/getting_started.html. Follow all of 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 https://www.balena.io/etcher/. 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.

Afterwards, 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 our GitHub repository located at https://github.com/kitkatkandybar/RFSoC-Spectrum-Monitoring/. Under the **`board/`** folder, there are two files, one called **`simplestream.ipynb`** and one called **`commands.py`**. 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!

2.4.2 Software

2.4.2.1 Software Installation

Our codebase is located at https://github.com/kitkatkandybar/RFSoC-Spectrum-Monitoring/.

Git needs to be installed on your computer in order to download our software. You can install Git at https://git-scm.com/downloads.

To install 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 at https://anaconda.org/.

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.

Our software also requires a Redis server. You can download Redis at https://redis.io/download. Redis requires a UNIX machine to run. Our team has successfully run Redis on a MacOS machine and in an Ubuntu environment running on Windows Subsystem for Linux.

2.4.2.2 Running Pipeline 1 - Live streaming RFSoC data

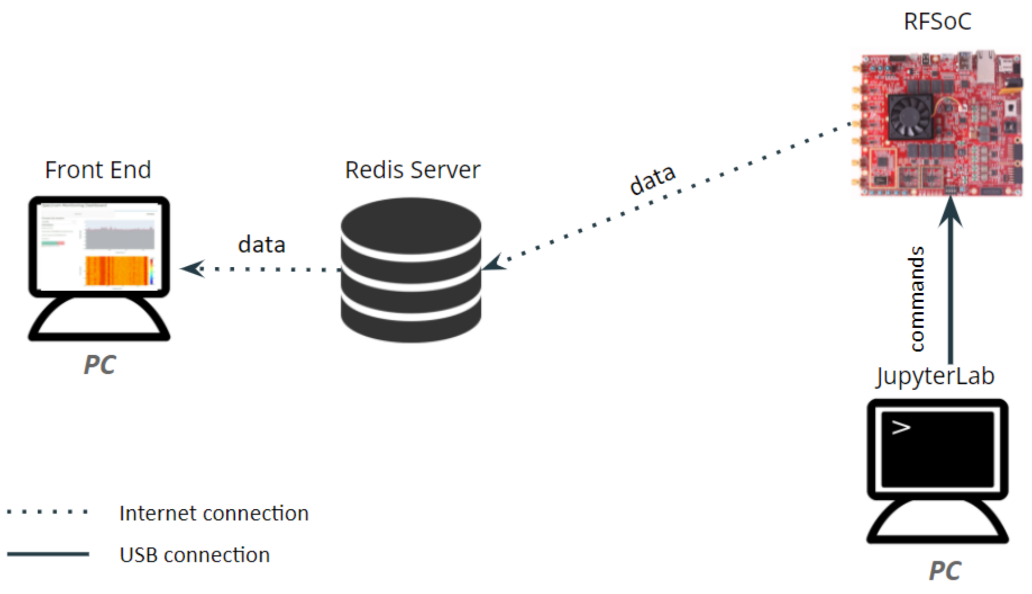


Figure 2.4 Block Diagram of the setup for the RFSoC data pipeline. A PC commands the RFSoC to run scripts through a JupyterLab environment. The RFSoC dumps data into a Redis stream, which the front end reads and displays in graph format.

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.

To run this pipeline:

1) Start the Redis Server

To start a Redis database, open a terminal which 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 our repository. You can find details for configuration parameters at https://redis.io/docs/manual/config/. We have kept "protected-mode" off in **`redis.conf`** to facilitate easy setup and connection to the database. However, "protected-mode" should be turned on when this project is used in production. By default, Redis listens on 'localhost' on port 6379.

2) Run the board scripts

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" in Google. Then, in the **`simplestream.ipynb`** file you created on the board in 2.4.1, enter the IP address in the line of code which sets up the redis connection :

```
r = redis.Redis(host='ip_address_here', port=6379, db=0)
```

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 and port values and that "protected-mode" is set to "no" in the Redis configuration parameters.

3) Run the Front end

First, set the IP address of the Redis server (which you found in step 2) 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/.

2.4.2.2 Running Pipeline 2 - Digital RF Playback

The Digital RF Playback has three components:

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

All three of these components can be run either on the same machine or on separate machines.

(1) Start the Redis Server

To start a Redis database, open a terminal which 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 our repository. You can find details for configuration parameters at https://redis.io/docs/manual/config/. We have kept "protected-mode" off in **`redis.conf`** to facilitate easy setup and connection to the database. However, "protected-mode" should be turned on when this project is used in production. By default, Redis listens on "localhost" on port 6379.

<u>(2) Run the Back end</u>
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" in 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/drf_back_end.py
```

<u>(3) Run the Front end</u>
First, set the address of the Redis server (which you found in step 2) 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/.

3 Operation of the Project (Yana)

3.1 Operating Mode 1: Normal Operation

3.1.1 Normal RFSoC Streaming Operation

1) Follow the steps outlined in 2.4.2.1 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.

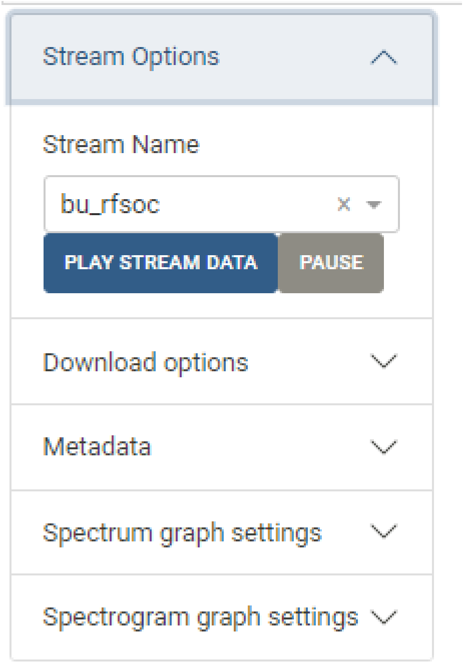
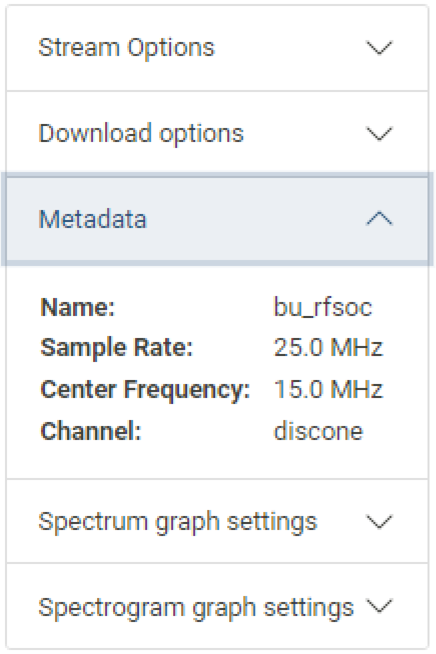


Figure 3.1 Stream Options tab. This image shows the Stream Options tab, which displays the name of a board which can be accessed, as well as the “play” and “pause” buttons.

5) The metadata for the live RFSoC data should be displayed in the sidebar under the “Metadata” tab.



Name:	bu_rfsoc
Sample Rate:	25.0 MHz
Center Frequency:	15.0 MHz
Channel:	discone

Figure 3.2 Metadata tab. This image shows an example of what the Metadata sidebar tab looks like after live streaming from a board begins.

6) Additional settings for the graphs can be found under the "graph settings," "spectrum graph settings," and "spectrogram graph settings" accordion tabs.

3.1.2 Normal Digital RF Playback Operation

1. Follow the steps outlined in 2.4.2.2 to get the web application running.
2. Open the application in a browser. Make sure to stay on the "Digital RF" tab.

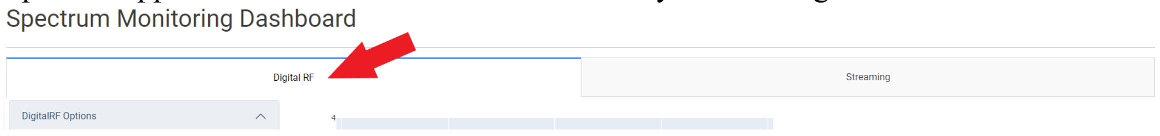


Figure 3.2 This image shows where the Digital RF playback tab is located.

3. Click the "Select Digital RF Dabta" button in the sidebar under the "Digital RF Options" accordion dropdown. A modal form should pop up.

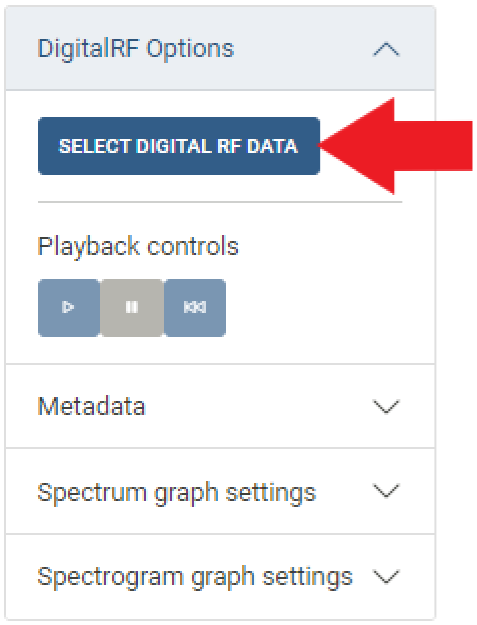
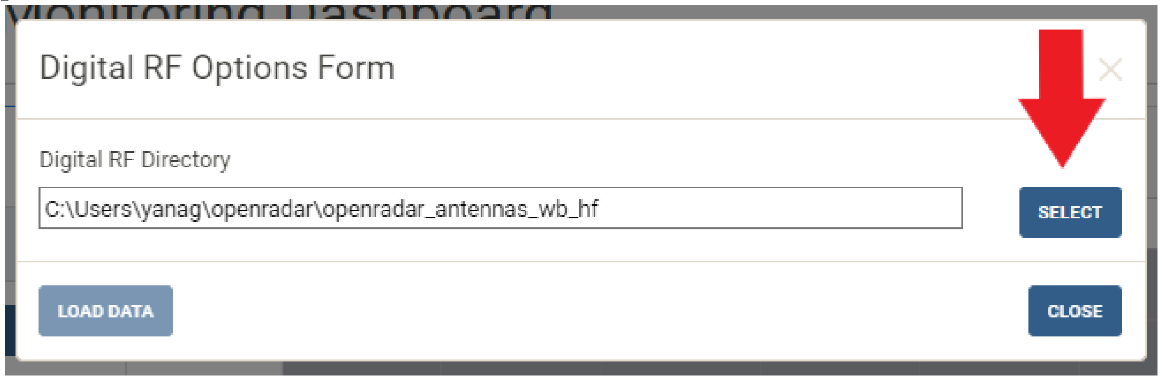


Figure 3.3 This image shows where to access the Digital RF playback request form.

4. Enter the file path of the Digital RF data you wish to display. Hit "select". NOTE: This file path is with respect to the *back end*. In other words, the data being played back should be located in whatever machine is running the back end. A future feature that should be implemented is having the user upload data to be processed from the front end.

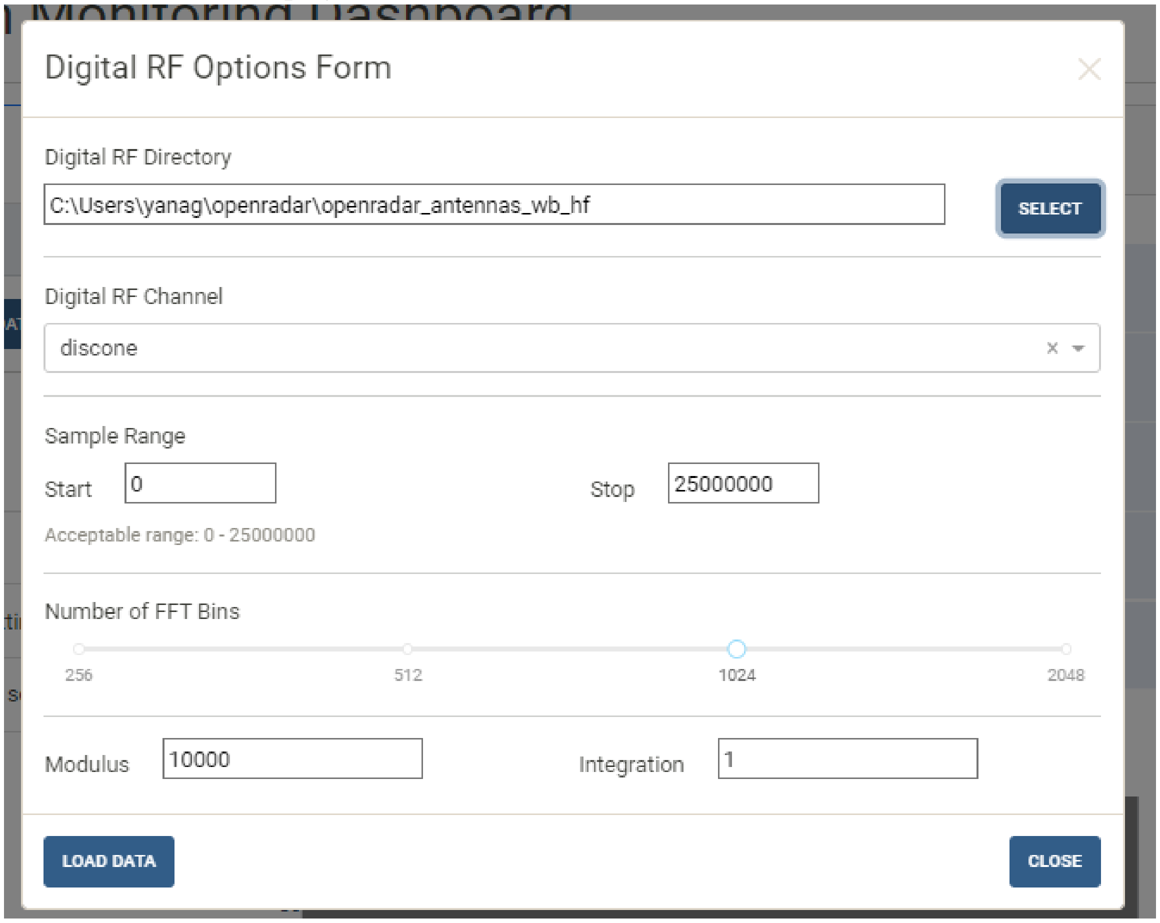


C:\Users\yanag\openradar\openradar_antennas_wb_hf

Figure 3.4 This image shows what the Digital RF playback request form initially looks like upon opening.

5. Additional form options should pop up. "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. For example, choosing "1024" means that each frame of the graphs will have 1024 data points. "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: C:\Users\yanag\openradar\openradar_antennas_wb_hf

Digital RF Channel: discone

Sample Range

Start: 0 Stop: 25000000

Acceptable range: 0 - 25000000

Number of FFT Bins: 1024

Modulus: 10000 Integration: 1

Figure 3.4 This image shows what the Digital RF playback form looks like 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, press the play button. You can also pause and rewind the data back to the beginning using the corresponding buttons.

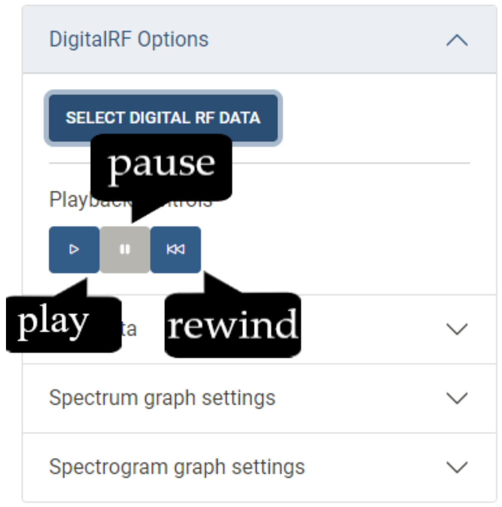


Figure 3.4 This image shows where the play, pause, and rewind buttons are located under the "DigitalRF Options" tab.

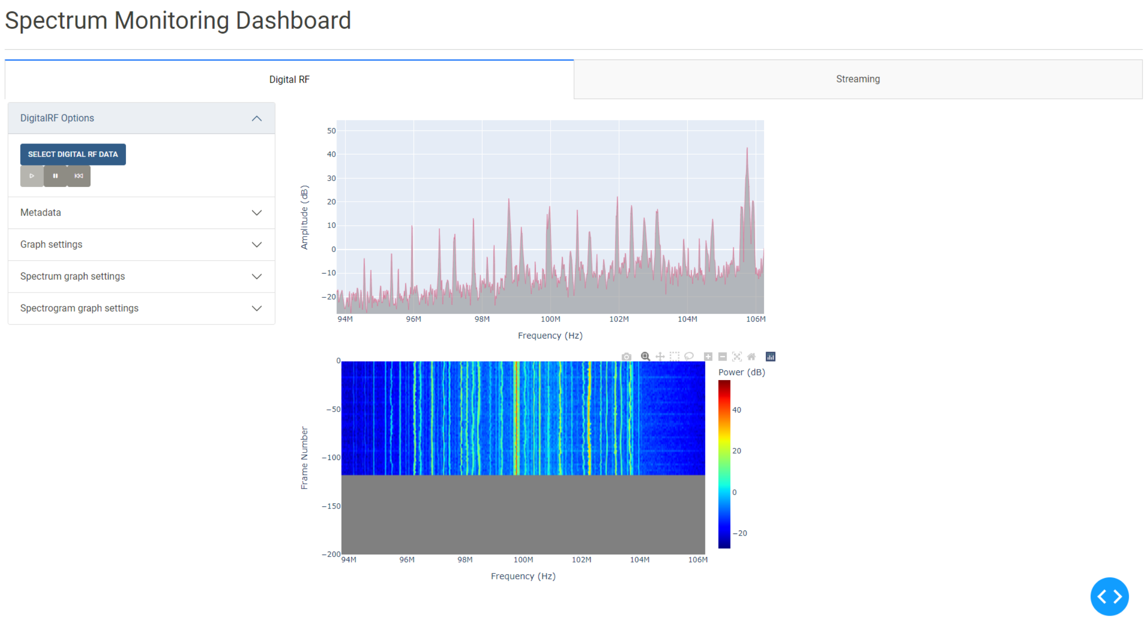
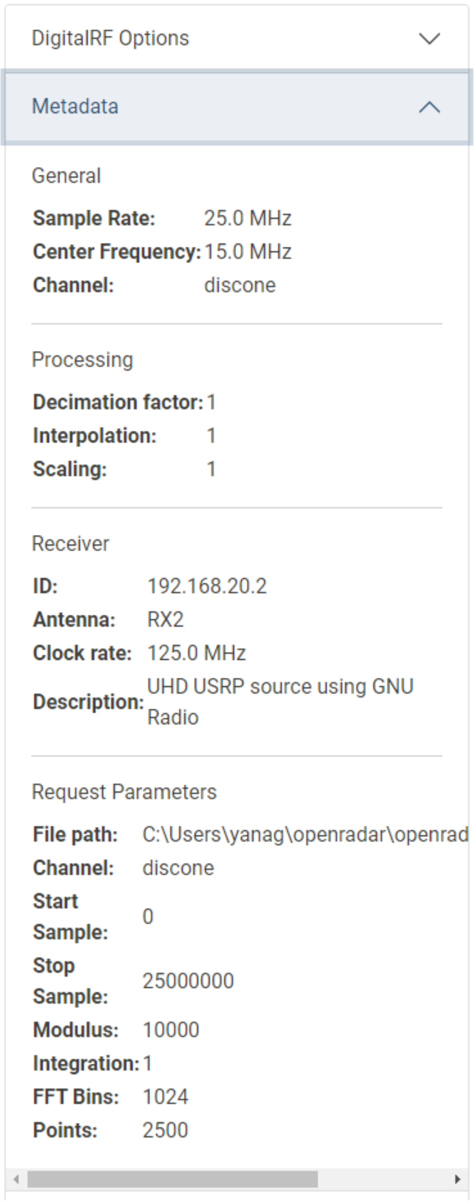


Figure 3.5 This image shows an example of what the dashboard looks like while data is being played back.

7. The metadata for the digital RF file should be displayed in the sidebar under the "Metadata" tab.



General

Sample Rate:	25.0 MHz
Center Frequency:	15.0 MHz
Channel:	discone

Processing

Decimation factor:	1
Interpolation:	1
Scaling:	1

Receiver

ID:	192.168.20.2
Antenna:	RX2
Clock rate:	125.0 MHz
Description:	UHD USRP source using GNU Radio

Request Parameters

File path:	C:\Users\yanag\openradar\openrad
Channel:	discone
Start Sample:	0
Stop Sample:	25000000
Modulus:	10000
Integration:	1
FFT Bins:	1024
Points:	2500

Figure 3.4 This image shows where the play, pause, and rewind buttons are located under the "DigitalRF Options" tab.

8. Additional settings for the graphs can be found under the "spectrum graph settings", and "spectrogram graph settings" accordion tabs.
9. To play back a different file, or the same file with different settings, repeat steps 3-5.

3.2 Operating Mode 2: Abnormal Operations

<u>Software Issues</u>

If there are issues with major portions of the application, it is likely due to a connection problem 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 the 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.

3.3 Safety Issues

Assuming reasonable handling of the RFSoC board, there should not be any significant safety concerns with this project.

4 Technical Background (Isaac, Kakit)

Xilinx RFSoC 2x2 Board

The RFSoC is a type of integrated circuit which can be used for communications and instrumentation. It has 2 built-in high-accuracy RF ADCs and DACs operating at Giga samples per seconds. It also utilizes the PYNQ framework with Jupyter Lab. It has a USB 2.0 UART/JTAG, 2 USB 3.0 Host, and USB 3.0 Slave port. It is also 4GB DDR4, 64 bit PS and PL accessible with a MicroSD card reader slot. It contains 4 switches, 2 RGB LEDs, 4 LEDs, and 5 push buttons. The clocks are 128 MHz for PL, 33 MHz for PS, 12.288 MHz for RF.

Dash Framework

Our Web application was created using Dash. Dash is a framework that was created for rapidly building interactive dashboard visualizations. Dash suits the requirements for our front end portion of our spectrum monitoring dashboard. Dash is built upon React.js for the front end, Plotly.js for the interactive graphs, and Python Flask for the Web server. Dash 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 a large amount of data-processing libraries and toolkits which are not available in JavaScript and ReactJS. Writing code entirely in Python made it easier for us to carry out the complicated data processing necessary for our dashboard graphs.

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.

Redis

Redis is an open source data structure store which can be used as a database, cache, message broker, and streaming engine. Redis is written in ANSI C and works on most POSI X systems such as Linux, *BSD, and Mac OS X without external dependencies.

Redis can be easily installed from the Redis website. Depending on the operating system there are different guides for installing the software. The website has a detailed guide of the steps needed in order to set up the server and have it running.

After the Redis server is set up on the host computer, the RFSoC will be collecting live data that will be processed and sent through the Redis server that will then be sent to the web application to be displayed.

5 Relevant Engineering Standards (Yana)

The front end of our application is a Web application which uses Hypertext Transfer Protocol (HTTP). This application is displayed on a user’s browser using HyperText Markup Language (HTML) and Cascading Style Sheets (CSS). The graphs which display RF data in our application are rendered using the Scalable Vector Graphics (SVG) format.

Our Web application was created using the Dash framework. The Dash framework allows the user to write a Web application using the Python programming language. Each Dash application incorporates a Python Flask server and a ReactJS front end. ReactJS, in turn, is a framework built using the Javascript programming language.

We use Redis for message passage and data storage. Redis is an in-memory, non-relational key-value database system. We store data structures within the Redis database using the JavaScript Object Notation (JSON) format.

We also use the Digital RF format in multiple parts of our project. Digital RF is a standardized format for reading and writing RF data which was developed by our clients at MIT Haystack Observatory. Our application can play back Digital RF data, and is also able to convert and store raw data from the RFSoC board as a Digital RF file.

6 Cost Breakdown (Yana, Jaime)

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

The cost of the software components of this project is negligible, since the Web application is not hosted on any 3rd party cloud service. The dominating expense is the cost of the RFSoC board.

In addition to the cost of the board, you need an antenna and other RF equipment in order to be able to receive signals. The cost of these accessories will depend strongly on the user’s needs and budget. We have included a few RF accessories which can be used for a basic monitoring setup. These accessories were recommended to us by our client. Any additional specialized equipment will likely cost more.

7 Appendices (Yana)

7.1 *Appendix A - Specifications*

Requirement	Value, range, tolerance, units
Displaying full range of live data from the RFSoC	● Displays a bandwidth of up to 2.5GHz (depending on the antenna and programmed bandwidth)
Delay between data entering RFSoC board and data being displayed on dashboard	● < 1 second
Rate of data frames displayed while live streaming data	● Displaying new data frames every 200 msec
Downloading board data	● Data can be downloaded from the RFSoC in DigitalRF file via a ZIP file
Smoothly playing back locally-stored DigitalRF data	● Displaying new data frames every 150 msec ● User is able to pause, play, and rewind data
User can interact with graphs	● Can adjust scales between logarithmic and linear values ● Can adjust y-axis bounds ● Can track maximum and minimum points on graph ● Can change color scale of spectrogram graph
Easy setup	● Codebase is thoroughly documented, including installation and setup procedures ● A new user should be able to set up and install the necessary software and equipment within 2 hours
Cost	● The software portion of the application is free to run

7.2 *Appendix B – Team Information*

Yana Galina is a Computer Engineering major at Boston University. She mainly worked on the Web application portion of the project.

Jaime Mohedano Aragon is a Telecommunications Engineering major at Comillas ICAI (Madrid, Spain). He mainly worked on the RFSoC portion of the project.

Kakit Wong is an Electrical Engineering major at Boston University.

Isaac Yamnitsky is a Computer Engineering major at Boston University.