BUECE

Boston University
Electrical & Computer Engineering
EC 464 Senior Design Project

Final Prototype Testing Report
4/8/2022

RF Spectrum Analyzer

By
Team 26

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

Overview

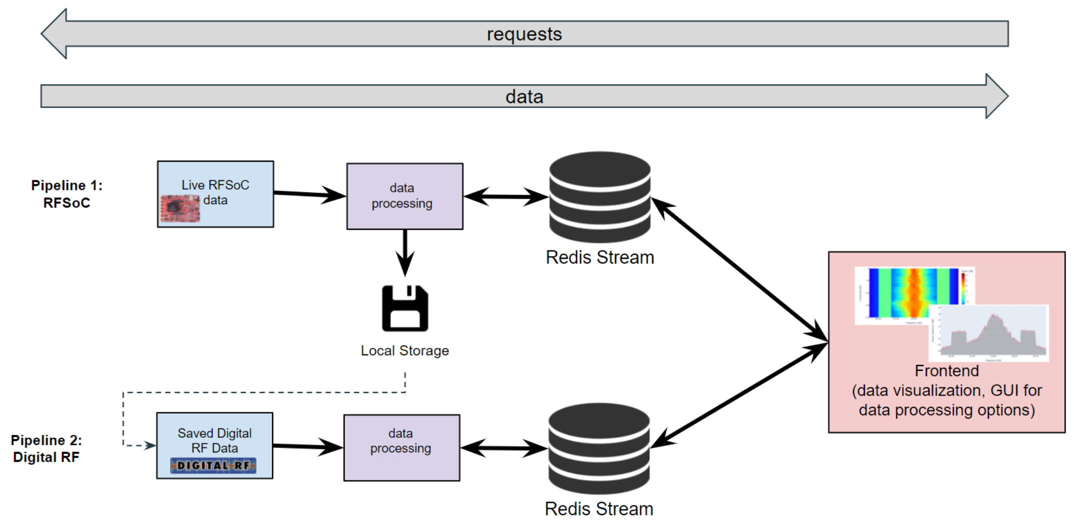


Figure 1: A block diagram of our design

Our team has developed and tested a web application which can display spectrum data for two different pipelines of RF data ("Pipeline 1" and "Pipeline 2", respectively). Pipeline 1 processes live data incoming from a Xilinx RFSoC board, and Pipeline 2 processes locally-stored RF data in the Digital RF format. We also implemented an additional feature, which lets the user download data from the RFSoC board in the DigitalRF format, which can be played back later.

For our final prototype testing, we conducted three different tests. In Test 1, We tested the Digital RF pipeline by playing back Digital RF data which had been originally provided by our clients. We also demonstrated the full interactive capabilities of our web application. In Test 2, we tested the RFSoC pipeline by demonstrating the live streaming feature displaying live data coming in from the RFSoC board on our web application. In Test 3, we used the front end to request data from the RFSoC board, which was downloaded as DigitalRF data in a zip file, and then played back the data stored in the zip file in a similar manner as Test 1.

Overall, our tests were successful, though there are still minor bugs in our application.

Equipment and Setup

<u>Test 1: Digital RF playback</u>

This setup consists of a single PC running a redis database, a back end python server, and a front end Dash web app. The back end streams requested Digital RF data into the front end to be displayed on the graphs there, through the Redis database. All three components (the Redis database, the back end, and the front end) can be run on different machines, but for the purposes of simplicity, we run all three on the same machine in this test.

Test 2: Live Streaming

Our setup involves two main components:

- The hardware side consists of the RFSoC board loaded with a data collection Jupyter notebook script. The board is connected to power and the internet via ethernet cable. One RX channel is connected to one of the TX channels, in "loopback" mode. 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 Fast Fourier Transforms, 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.

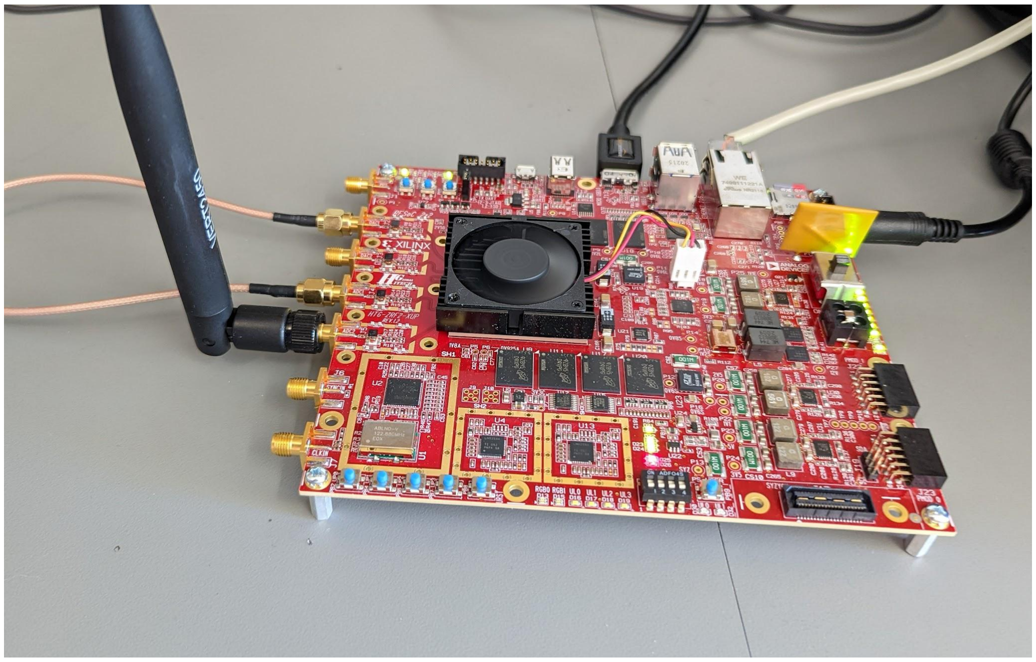


Figure 2: An image of our RFSoC board. On the left side, the TX-RX loopback connection over the beige cable can be seen.

Test 3: Downloading and playing back board data

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

- The hardware side consists of the RFSoC board loaded with a data collection Jupyter notebook script. The board is connected to power and the internet via ethernet cable. One RX channel is connected to one of the TX channels, in “loopback” mode. 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, the Redis server, and the python backend. 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.

Detailed Measurements

Test 1: Digital RF playback

In this test, we played back the “openradar_antennas_lband_ads_b” Digital RF file provided by our client. We were able to pause, play, and restart the file. We were able to see metadata for the file in the designated accordion tab. We were able to toggle the scaling for both the spectrum and spectrogram between logarithmic and linear scales. We were able to modify y axis boundaries on both graphs. We were able to track minimum and maximum points on the spectrum graph. We were able to change the color scale of the spectrogram graph.

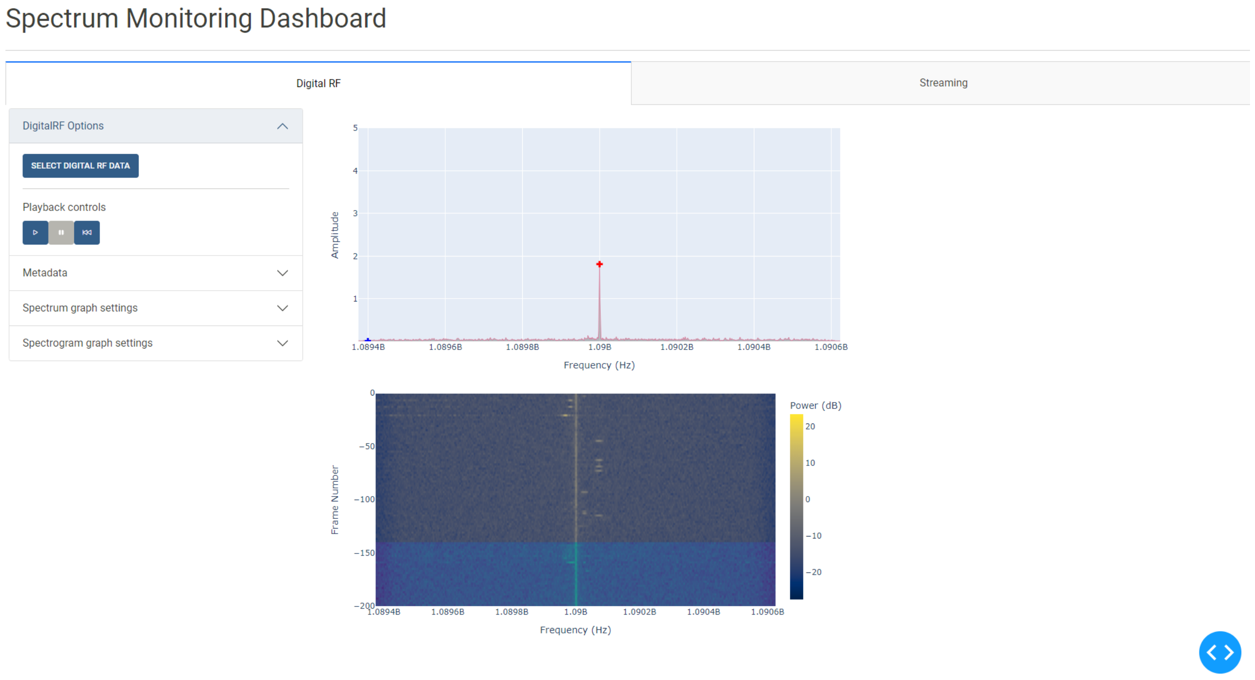


Figure 3: Pre-recorded Digital RF data being displayed on our web application. The spectrum graph (top) has been converted to a linear scale, and the minimum and maximum points are being displayed (the blue and red crosses, respectively). The spectrogram graph (bottom) had its color scale changed part way through the play back.

	T/F
User can select a Digital RF Directory	**T**
Channels and sample ranges appropriate for the file appear in the selection form	**T**
Metadata appears after a request has been processed	**T**
User can pause data	**T**
User can play data	**T**
User can rewind data back to beginning	**T**
User can toggle graph between linear and logarithmic scales	**T**
User can change y-axis limits of graph	**T**

User can track minimum and maximum points on the spectrum graph	**T**
User can change colors scheme of spectrogram graph	**T**

Test 2: Live Streaming

In this test, we tested the live streaming feature. Using a script running on the RFSoC, we set the TX channel, which is connected to our RX channel, to output a pulse at 600 MHz. When we connected to the stream on our front end, we were able to clearly see the peak at 600 MHz. The stream was smooth, with no major lags or bumps.

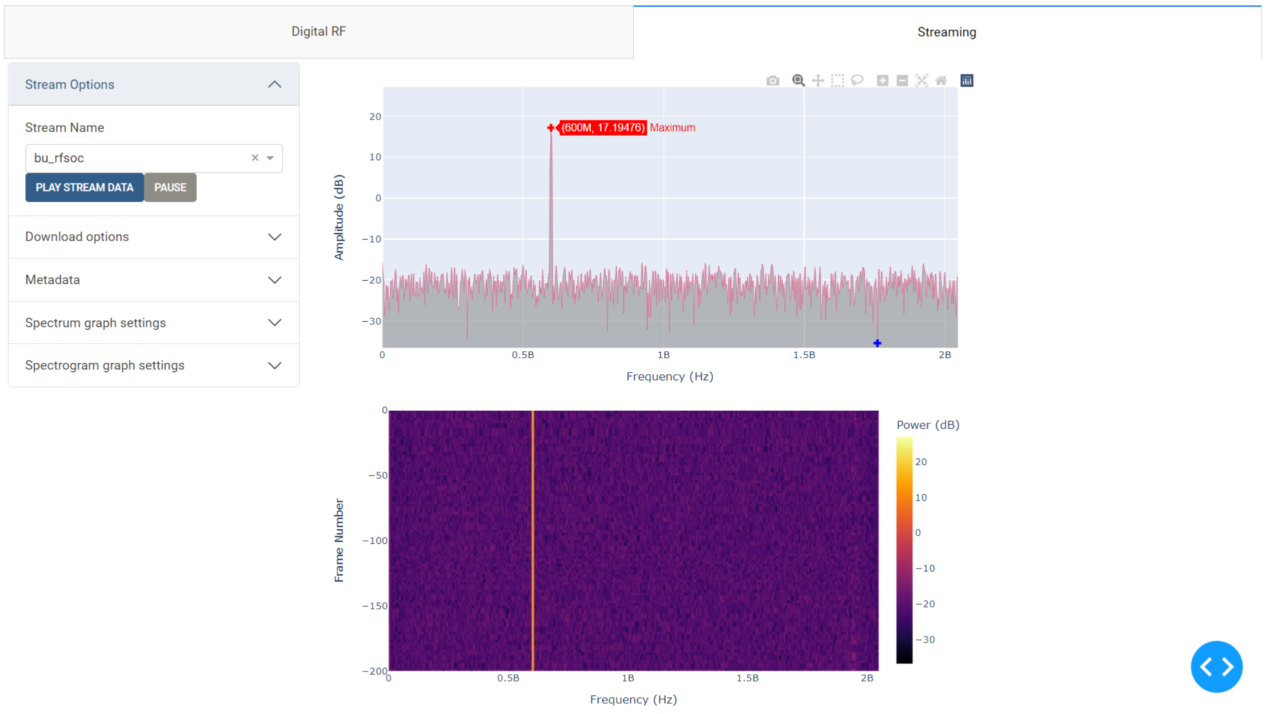


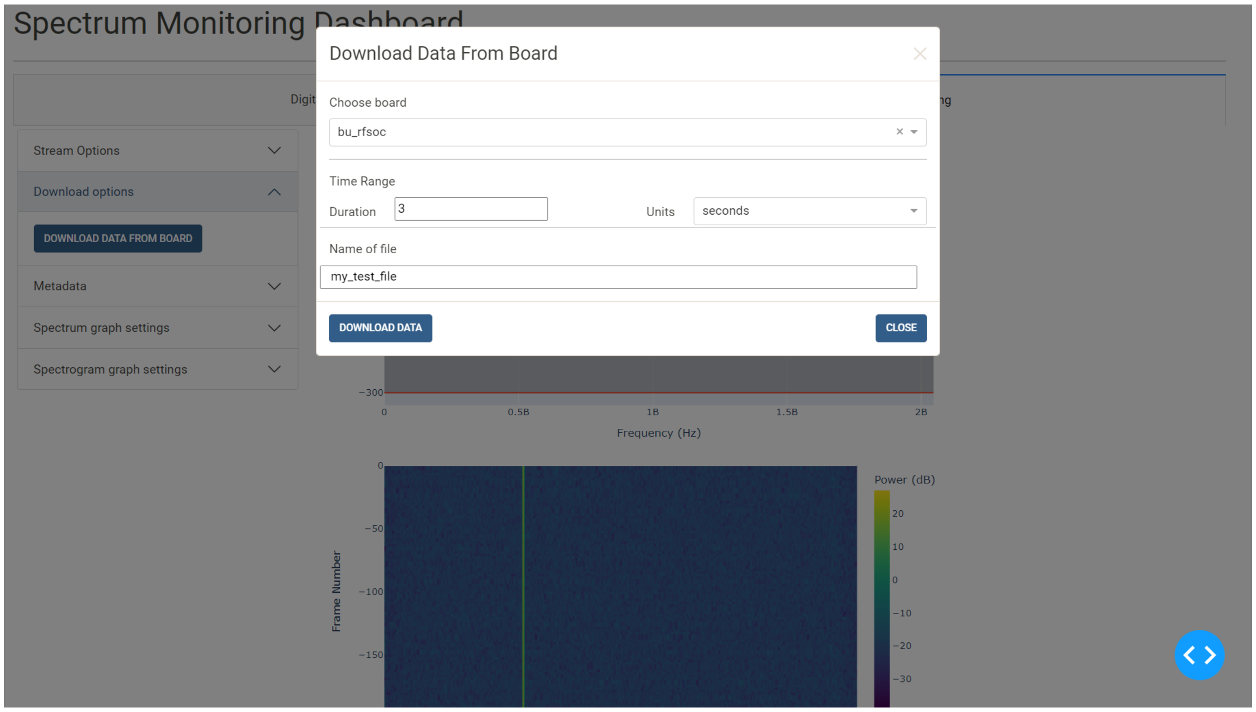
Figure 4: Live RFSoC data being displayed on our dashboard. The left panel is showing that the data is coming from the "bu_rfsoc" stream. The peak is clearly at 600 MHz, the frequency value we set the TX channel we set a pulse at. This means that we are correctly processing and displaying the data in our application.

	T/F
User can find "bu_rfsoc" under "Stream Options" tab	**T**
User can pause and play data	**T**

Graph frames are updated at rate greater than 1 per second	**T**
The graphs depict accurate data (eg a peak at 600 MHz)	**T**
Metadata should appear on the sidebar	**T**
Interactive graph options as described in Test 1 criteria work properly	**T**

Test 3: Downloading board data and playing back

In this test, we request 3 seconds of data from the RFSoC. The RFSoC had an RX channel in loopback with a TX channel. For this test, the TX channel was set to have a pulse at 750MHz. We submitted a request for 3 seconds of data using the front end request form. The RFSoC processed that request, and dumped the raw data into the Redis stream. 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. We extracted the zip file, and then played its contents back in the Digital RF playback section of our web application. This playback clearly displayed the peak at 750 MHz.



Spectrum Monitoring Dashboard

Download Data From Board

Choose board: bu_rfsoc

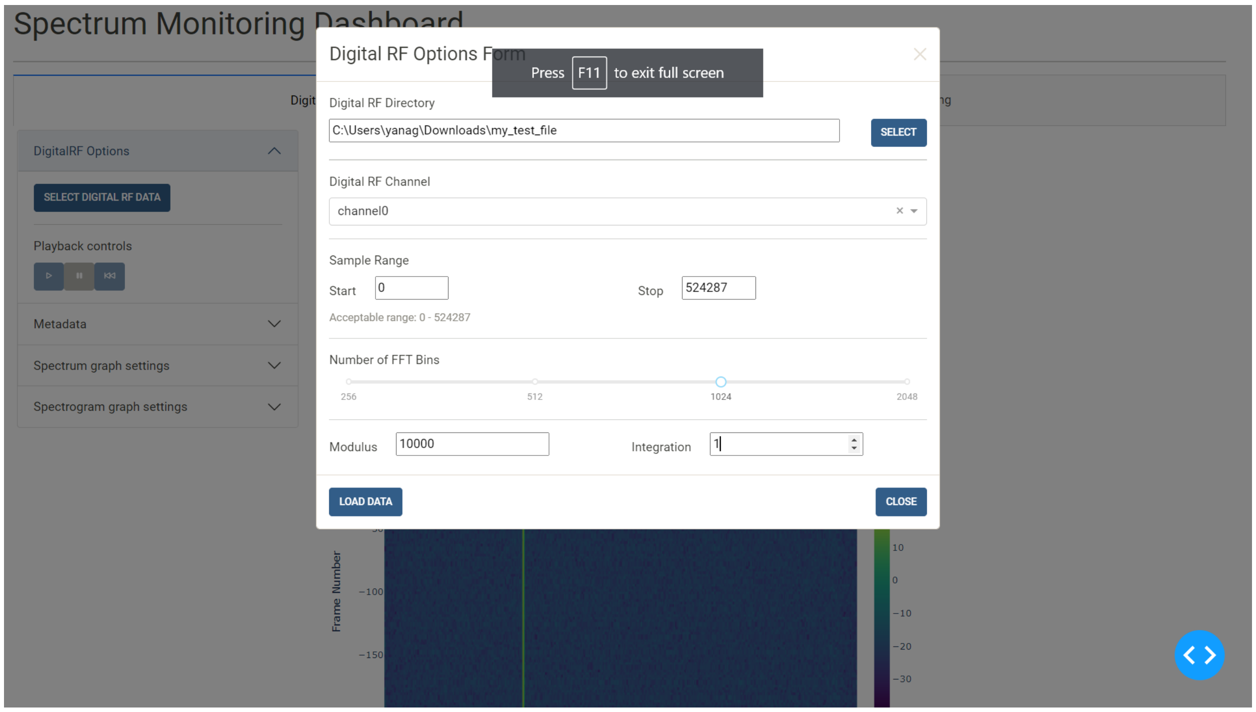
Time Range

Duration: 3

Units: seconds

Name of file: my_test_file

Figure 5: A screenshot of the data request form.


Spectrum Monitoring Dashboard

Digital RF Options Form

Press F11 to exit full screen

Digital RF Directory: C:\Users\yanag\Downloads\my_test_file

Digital RF Channel: channel0

Sample Range

Start: 0

Stop: 524287

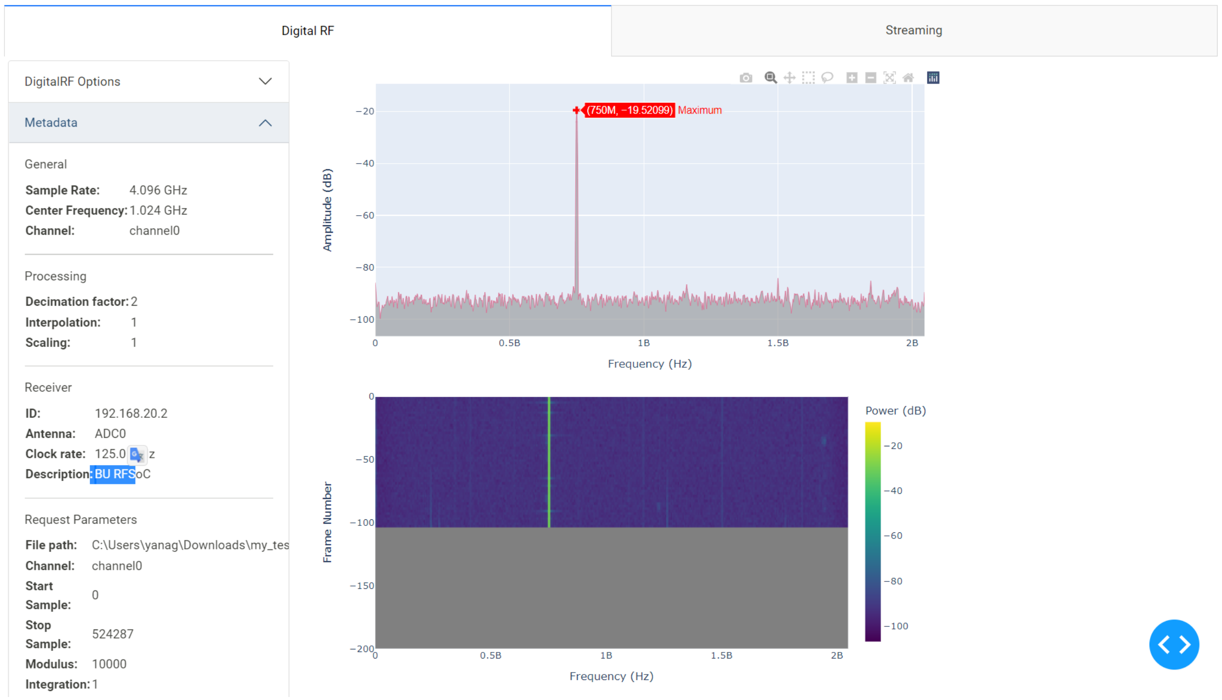
Acceptable range: 0 - 524287

Number of FFT Bins: 1024

Modulus: 10000

Integration: 1

Figure 6: A screenshot of the form for playing the back the Digital RF data downloaded in Figure 5.


Metadata

General

Sample Rate: 4.096 GHz

Center Frequency: 1.024 GHz

Channel: channel0

Processing

Decimation factor: 2

Interpolation: 1

Scaling: 1

Receiver

ID: 192.168.20.2

Antenna: ADC0

Clock rate: 125.0 z

Description: BU RFSoC

Request Parameters

File path: C:\Users\yanag\Downloads\my_tes

Channel: channel0

Start Sample: 0

Stop Sample: 524287

Modulus: 10000

Integration: 1

Figure 7: A screenshot of the data downloaded in Figure 5 being played back. A peak at 750 MHz is clearly visible, meaning that the data has likely been transmitted and played back accurately.

	T/F
User can find "bu_rfsoc" in the download request form	**T**
A zip file is downloaded after several seconds	**T**
After zip file extraction, the Digital RF Playback feature of the web application is able to read and process the data	**T**
The graphs depict the data accurately (eg there should be a clear peak at 750 MHz)	**T**

Conclusions

In our first semester report, we wrote that "we aim to create a practical base that can be extended as needed to serve future research needs." We consider ourselves to have succeeded in this goal and consider our project to be a success overall. The direction of our project has shifted several times since the beginning of the year, but this was expected, since our project has always been exploratory in nature.

One major requirement we had was to be able to play back Digital RF data. We successfully demonstrated this capability in Test 1.

Another requirement was to be able to display the full spectrum of data coming out of the RFSoC board. We successfully demonstrated this capability in Test 2 with our live streaming feature.

An additional requirement we had was to be able to store data from the board in the Digital RF format, and we successfully demonstrated this capability in Test 3.

There are still some minor bugs with the web application. 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. Overall, the web application could use further robustness and error handling.

Last semester, we considered implementing a wide variety of features, many of which we did not have time to implement during this semester. Overall, we were able to make more progress on the front end, software side of the project as opposed to the hardware side. This was mainly due to receiving the RFSoC board only at the end of our first semester.

One capability we did not implement was commanding the board for live streaming, for example by specifying a particular bandwidth and center frequency to get data from. Our live streaming feature is “passive” in that the board sets all of the parameters for itself, and the front end only passively listens into the data being dumped into the Redis stream.

We had originally planned to do more signal processing work with the board. This would have entailed being able to download, or play back, data from the board which had been processed, such as being digitally down-converted or filtered in some other way. We were not able to get this point with the board.

However, the features we *did* implement work as a solid framework from which these additional features could quickly be implemented. Commanding the board for live streaming and signal processing work would be an extension of the scripts we wrote and capabilities we implemented for Tests 2 and 3. Therefore, our overall goal of creating an extendable “base” application was accomplished.