Real-Time Spectrum Analysis and STFT Visualization using USRP B210 SDR

Sai Krishna Shanigarapu
Electrical Engineering
Indian Institute of Technology, Hyderabad
Hyderabad, India
(ee23btech11054@iith.ac.in)

***Abstract*—In the following project, we implement a programmable, real-time spectrum analyzer using the Software Defined Radio (SDR). We capture the complex base-band IQ samples in the 2.4 GHz ISM band and process them using a Short-Time Fourier Transform (STFT) to visualize the time domain signal. By utilizing the UHD Python API, the system achieves controllable center frequency, channel width and time duration.**

***Index Terms*—Wi-Fi beacon, Frequency-hopping spread spectrum (FHSS), Software Defined radio (SDR), Short Time Fourier Transform (STFT)**

I. INTRODUCTION

The growth of wireless devices in 2.4 GHz ISM band has created a need for tools to monitor the spectral usage. The traditional hardware spectrum analyzers are highly expensive and they also lack the flexibility for custom signal processing. As technology advanced, now we can implement the same traditional hardware components (mixers, filters, modulators) in software using much cheaper Software Defined Radio (SDR) boards [1]. Previously, low-cost SDR's like RTL-SDR were used in this domain. However, it has a very limited bandwidth; around 2MHz [2]. This project utilizes the Ettus Research USRP-B210 (B200 family), a high performance direct conversion transceiver.

In this project, we try to utilize the Python UHD API to receive the complex base-band IQ samples and then we will try to compute the STFT. However, we will face a lot of hurdles while implementing this. When we receive a signal, there is a Local Oscillator (LO) present in the board which gives a DC offset. How to suppress it? What window should be used while computing STFT to get the minimum spectral leakage? We will try to find out one by one using the core Digital Signal Processing Techniques. You can find the complete installation and setup for USRP-B210 in APPENDIX.

II. CONSTRAINTS

- **Programmability:** The user should be able to change the parameters (center frequency, sampling rate and dwell time) for every run.
- **Frequency resolution:** Need to make sure that FFT bin size should be 25 kHz. In STFT, the frequency resolution (Δf) is determined by sampling rate (F_s) and FFT size (N_{fft})

$$\Delta f = \frac{F_s}{N_{fft}} \tag{1}$$

  Here, we want each pixel in the spectrogram to represent exactly 25 kHz of bandwidth. Thus,

$$N_{fft} = \frac{F_s \times 10^{-3}}{25} \tag{2}$$

- **Overlap:** While computing STFT, make sure to use 50% overlap with the previous block to ensure all data is processed and to prevent any significant features from being missed.

III. MATHEMATICAL FORMULATION

As the USRP-B210 receives the signal, it down-converts the continuous time domain signal $x(t)$ into a discrete-time signal $x[n]$. This $x[n]$ can be represented as:

$$x[n] = I[n] + j\,Q[n] \tag{3}$$

where $I[n]$ and $Q[n]$ are real (In-phase) and imaginary (Quadrature-phase) components respectively.

To visualize the frequency content change over time, we implemented the discrete STFT. For a signal $x[n]$ and a sliding window function $w[n]$, the STFT is computed as:

$$X(m,k) = \sum_{n=0}^{N-1} x[n+mH]w[n]e^{-j\frac{2\pi}{N}kn} \tag{4}$$

where:

- m is the time index of the window.
- k is the frequency index.
- H is the hop size (overlap).
- N is the FFT size .

The STFT output $X[m,k]$ is a complex quantity containing both magnitude and phase information. For spectral analysis, we are interested in the power distribution. The spectrogram $P[m,k]$ is computed as follows:

$$|X[m,k]|^2 = \text{Re}\{X[m,k]\}^2 + \text{Im}\{X[m,k]\}^2 \tag{5}$$

To visualize the wide dynamic range of RF signals, the magnitude is converted to a logarithmic decibel (dB) scale:

$$P_{dB}[m,k] = 20\log_{10}\left(|X[m,k]| + \epsilon\right) \tag{6}$$

where $\epsilon \approx 10^{-12}$ is a small regularization constant added to prevent mathematical singularities ($\log(0) \to -\infty$).

IV. DC OFFSET SUPPRESSION

In any SDR board, a Local Oscillator (LO) is present which gives a DC offset. When ever we tune a center frequency, say, f_c, this DC offset causes LO leakage. In the STFT spectrum, we see a horizontal line at f_c. If we fail to remove these spurious frequency components, there's a danger that any DSP/ML algorithm will mistake this to be a real signal. Thus, it must be removed.

A. *Single-Tap IIR Filter*

In this manual [3], they have explicitly stated to use **Single-Tap IIR** (Infinite Impulse Response) filter. Here, the current input sample $x\left[k\right]$ is subtracted from a weighted portion of the previous output $y\left[k-1\right]$ to get a new output $y\left[k\right]$.

$$y\left[k\right] = x\left[k\right] \,-\, \alpha\, y\left[k \,-\, 1\right] \tag{7}$$

In the above difference equation, α is a very small number (2^{-20}). As the α becomes very small, in the frequency domain, it acts as a **notch filter** which removes a particular tone. The smaller the alpha is, the narrower the notch will be.
To implement this in the USRP, we can simply add this line to our code.

```
usrp_dev.set_rx_dc_offset(True, 0)
```

B. *Statistical Mean Subtraction*

According to [3], a DC offset is a fixed voltage that is permanently present on the signal. This acts as an additive error. Let $x\left[n\right]$ be the desired signal and $y\left[n\right]$ be the received signal, then

$$y\left[n\right] = x\left[n\right] \,+\, D \tag{8}$$

where D is the offset.
Since all these Wi-Fi and Bluetooth Low Energy (BLE) signals are high frequency signals that oscillate between +ve and -ve values, over a sufficiently large samples (N), their mean tends to zero

$$\frac{1}{N}\sum_{n=0}^{N-1} x\left[n\right] \;\approx\; 0 \tag{9}$$

Thus, we can estimate the value of this D. Let $\hat{D}$ be the estimated DC-offset.

$$\hat{D} = \frac{1}{N}\sum_{n=0}^{N-1} y\left[n\right] \tag{10}$$

We can then subtract this $\hat{D}$ from $y[n]$ to get the estimated original signal.
Note that since we use sampling rate, say 6 MHz, for a well time of 0.1 second, we have 600,000 samples which ensures the estimation error is minimized.

V. Window Function Convergence Analysis for minimum spectral leakage

Choosing a window is the most critical task while computing the STFT. There are several windows which have their own applications. For a classification problem, the STFT plot should be clear so that the ML algorithms can successfully detect the signal. Thus, choosing window is critical. For this, we have captured Bluetooth signals at 2.467 GHz and computed the STFT with various window functions.

The following functions, main lobe widths, side lobe widths, trade-offs and uses of each window can be referred from this paper [4].

A. *Rectangular Window Analysis*

The rectangular window is simplest window function, equivalent to uniformly weighting the signal with unity gain over the observation interval. Mathematically defined as:

$$w[n] = \begin{cases} 1 & 0 \le n \le N-1 \\ 0 & \text{else} \end{cases} \tag{11}$$

where N is the length of the STFT block.

- **Main Lobe Width:** $4\pi/N$ bins. This is the narrowest main lobe of all common window functions, providing the highest possible frequency resolution.
- **Side-Lobe Level:** -13 dB. This is the highest side-lobe level among standard windows, resulting in the poorest dynamic range and severe spectral leakage.

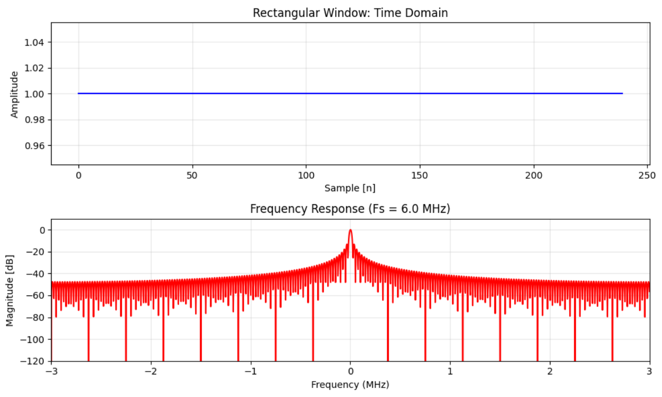

Fig. 1: Frequency and Time domain characteristics of the Rectangular Window.

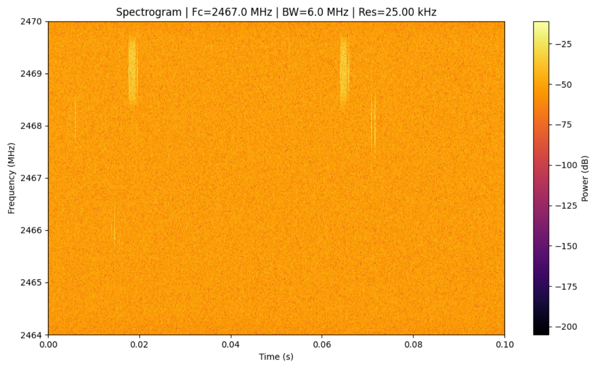

Fig. 2: STFT of the captured Bluetooth signals with rectangular window

From the above figure, it would be very difficult for ML algorithms to detect the Bluetooth signals present in it.

B. *Hanning Window Analysis*

The Hanning (or Hann) window is a raised-cosine window designed to touch zero at both ends, eliminating the discontinuity at the block boundaries. Mathematically defined as:

$$w[n] = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right), \quad 0 \le n \le N-1 \tag{12}$$

- **Main Lobe Width:** $8\pi/N$ bins. The main lobe is twice as wide as the rectangular window, resulting in a slight reduction in frequency resolution.
- **Side-Lobe Level:** -31.5 dB. This offers a significant improvement in leakage suppression compared to the rectangular window (-13 dB).

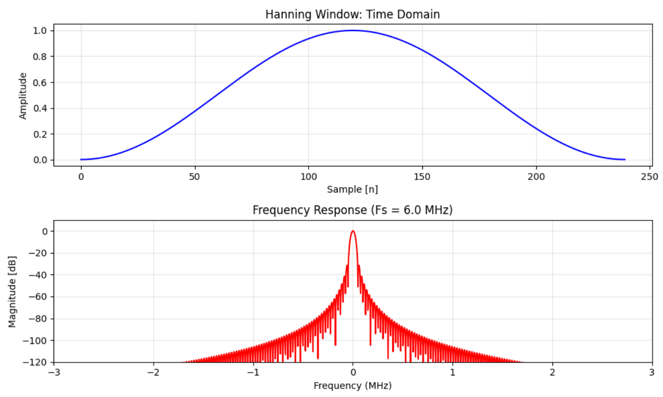

Fig. 3: Frequency and Time domain characteristics of the Hanning Window.

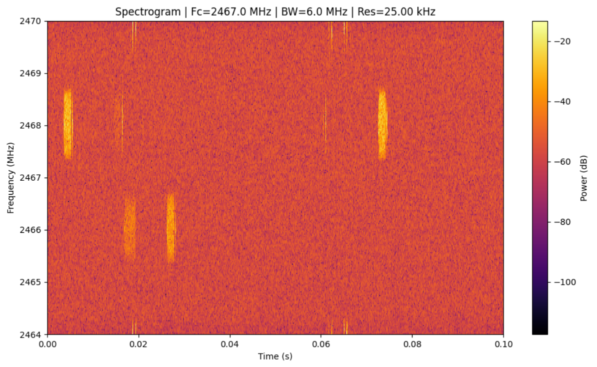

Fig. 4: STFT of the captured Bluetooth signals with Hanning window

Compared to the rectangular window, this plot is much more clear.

C. Hamming Window Analysis

The Hamming window is a modification of the Hanning window. It is also a raised-cosine window but uses optimized coefficients to minimize the maximum (nearest) side-lobe level. Mathematically defined as:

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \le n \le N-1 \tag{13}$$

- **Main Lobe Width:** $8\pi/N$ bins. Similar to the Hanning window.
- **Side-Lobe Level:** -43 dB. This provides better immediate suppression than Hanning (-31 dB)

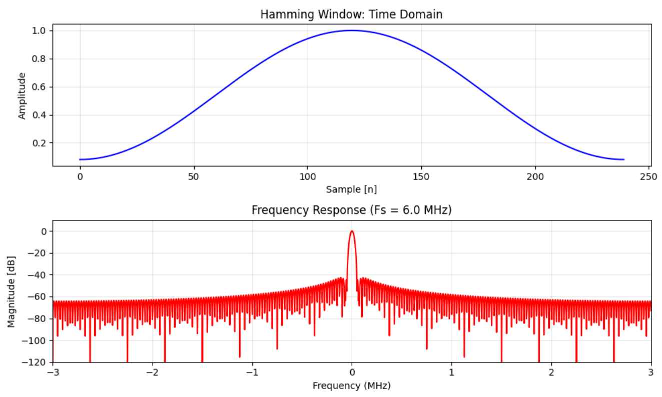

Fig. 5: Frequency and Time domain characteristics of the Hamming Window.

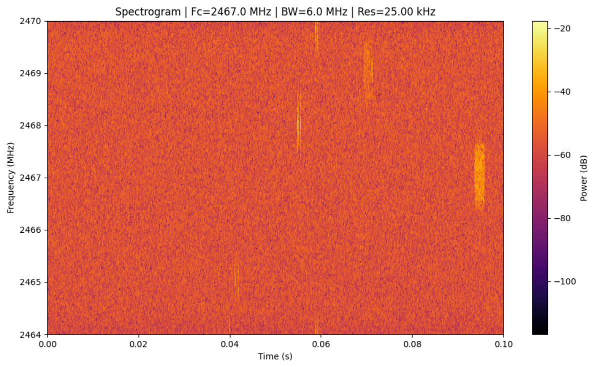

Fig. 6: STFT of the captured Bluetooth signals with Hamming window

D. Bartlett Window Analysis

The Bartlett window is a triangular window that effectively performs linear weighting, tapering from unity at the center to zero at the ends. Mathematically defined as:

$$w[n] = 1 - \left|\frac{n - \frac{N-1}{2}}{\frac{N-1}{2}}\right|, \quad 0 \le n \le N-1 \tag{14}$$

- **Main Lobe Width:** $8\pi/N$ bins. Same width as Hanning and Hamming windows.
- **Side-Lobe Level:** -26 dB. While this is an improvement over the Rectangular window (-13 dB), it performs worse than the Hanning (-31 dB) and Hamming (-43 dB) windows.

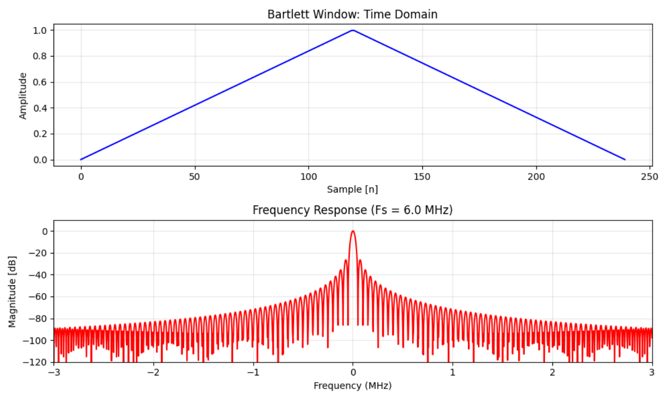

Fig. 7: Frequency and Time domain characteristics of the Bartlett Window.

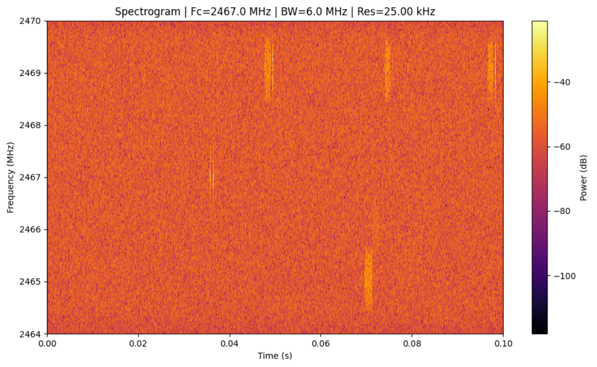

Fig. 8: STFT of the captured Bluetooth signals with Bartlett window

E. Kaiser Window Analysis

The Kaiser window is a family of windows based on the zeroth-order modified Bessel function of the first kind, denoted as I_0. Mathematically defined as:

$$w[n] = \frac{I_0\left(\beta\sqrt{1-\left(\frac{2n}{N-1}-1\right)^2}\right)}{I_0(\beta)}, \quad 0 \le n \le N-1 \qquad (15)$$

- **Parameter** $\beta = 14$**:** The parameter β controls the shape. A value of $\beta = 0$ corresponds to a rectangular window, while $\beta \approx 6$ approximates a Hanning window. According to Harris [4], increasing β significantly reduces side-lobe levels. We selected a high value of $\beta = 14$ to achieve a side-lobe attenuation exceeding -100 dB.

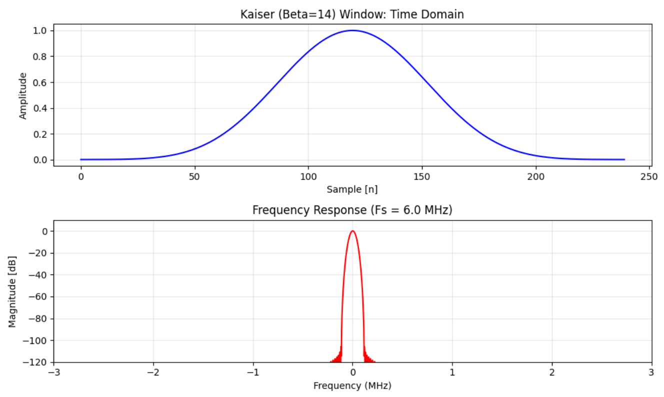

Fig. 9: Frequency and Time domain characteristics of the Kaiser Window ($\beta = 14$).

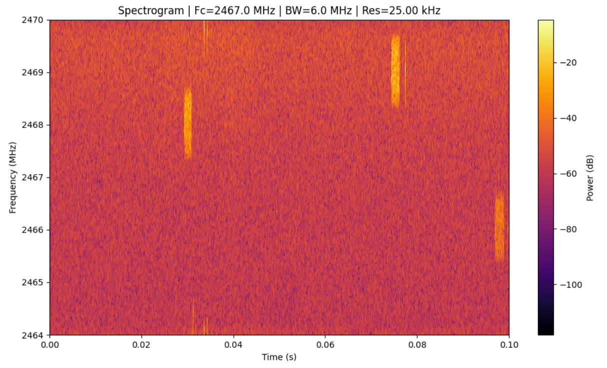

Fig. 10: STFT of the captured Bluetooth signals with Kaiser window ($\beta = 14$).

F. Blackman-Harris Window Analysis

The Blackman-Harris window is a general term for a family of 3-term or 4-term windows. The 4-term variation used in this experiment is a sum of cosines designed to minimize side-lobe levels. Mathematically defined as:

$$w[n] = a_0 - a_1 \cos\left(\frac{2\pi n}{N-1}\right) + a_2 \cos\left(\frac{4\pi n}{N-1}\right) - a_3 \cos\left(\frac{6\pi n}{N-1}\right) \qquad (16)$$

where the coefficients are $a_0 = 0.35875$, $a_1 = 0.48829$, $a_2 = 0.14128$, and $a_3 = 0.01168$. [7]

- **Main Lobe Width:** $16\pi/N$ bins. The main lobe is twice as wide as the Hanning window. This results in "fatter" signal points in the spectrogram (reduced frequency resolution).
- **Side-Lobe Level:** -92 dB. This offers excellent spectral leakage suppression.

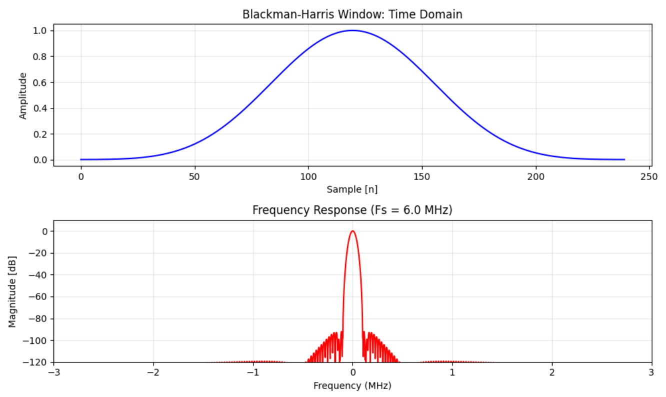

Fig. 11: Frequency and Time domain characteristics of the Blackman-Harris Window.

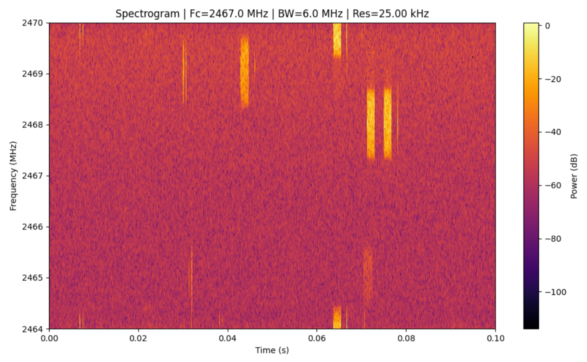

Fig. 12: STFT of the captured Bluetooth signals with Blackman-Harris window

Results:
It is evident from the figures and as well as the side lobe levels, that Kaiser window (β = 14) and Blackmanharris perform exceptionally well compared to other windows. Here, the Bluetooth signals are clearly visible when these two windows were used compared to others. Thus, ML algorithms won't face difficulty in detecting them. So, choosing any one of the windows - **Kaiser** or **Blackmanharris** - will be a good choice for minimal spectral leakage.

VI. Results and Analysis

For the final results, we have used Kaiser (β = 14) window for the STFT computation. We tried to capture the Wi-Fi Hotspot signals at 2.432 GHz and BLE signals at 2.467 GHz. We used our own Mobile Hotspot (2.4 GHz) and streamed a song via Bluetooth headset connected to mobile.
Theoretically, this board works well for up to 61.44 MS/s. [9]

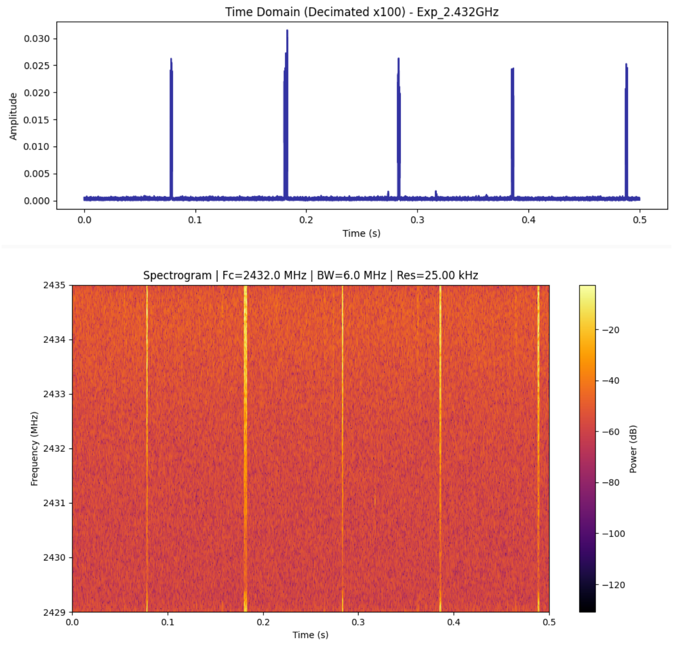

Fig. 13: Time domain and STFT plot of Wi-Fi beacons

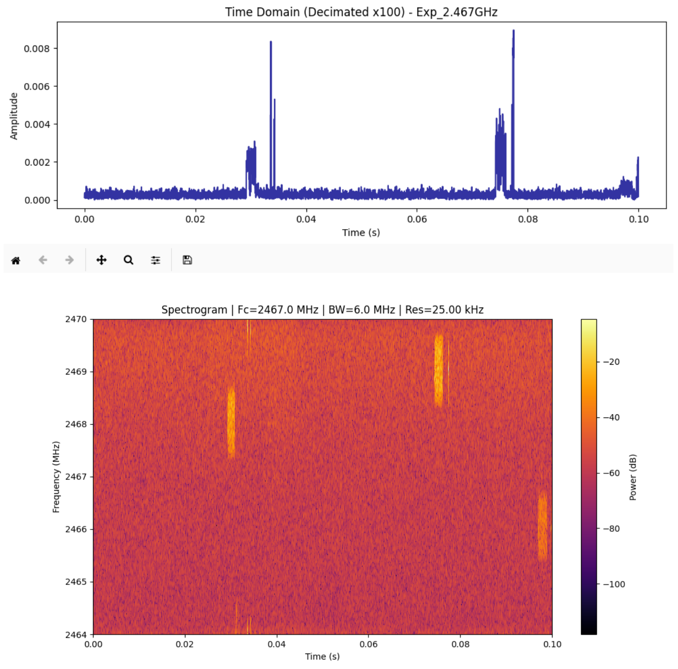

Fig. 14: Time domain and STFT plot of BLE

In Figure 13, the bright yellow vertical lines which are repeating periodically after every 102.4 ms are Wi-Fi beacons. In Figure 14, the two yellow bright box like structures represent BLE signals.

VII. Conclusion

We have successfully captured the complex base-band signals through USRP-B210 and computed their STFT. We also detected the Wi-Fi Hotspot and BLE signals at 2.432 GHz and 2.467 GHz respectively. We have also suppressed the DC offset caused by the LO oscillator present in the SDR board using two strategies: (i) Single-Tap IIR filter and (ii) Mean Subtraction. We have analyzed the STFT plots of BLE signals captured at 2.467 GHz with different window functions and came to a conclusion that both - Kaiser and Blackman-Harris - windows give minimum spectral leakage.

To ensure the reproducibility of experiments, all the source codes are present in the below GitHub repository.
https://github.com/ssk1504/SDR_USRP_spectrum_analysis

Appendix A
Installation and Setup of USRP-B210

To utilize the UHD Python API with USRP-B210, **Ubuntu 22.04** and **Python 3.8+** are must. System should have USB 3.0 port. Without this, we can't configure the USB Permissions for USRP.

A. Install Dependencies

```
sudo apt update
sudo apt install -y \
    git cmake g++ python3 python3-dev python3-pip
        \
    libboost-all-dev libusb-1.0-0-dev \
    libncurses5-dev python3-mako libaio-dev
```

B. Clone the UHD Repository

```
cd ~
git clone https://github.com/EttusResearch/uhd.
    git
mv uhd uhd_repo
```

C. Build UHD From Source

Configure the build to enable the Python API.

```
cd ~/uhd_repo/host
mkdir build
cd build

cmake \
  -DENABLE_PYTHON_API=ON \
  -DPYTHON_EXECUTABLE=$(which python3) \
  -DPYTHON3_SITEPKG=/usr/local/lib/python3.10/
      dist-packages \
  ..

make -j$(( $(nproc) / 2 ))
sudo make install
sudo ldconfig
```

D. Install the UHD Python Module

```
cd ~/uhd_repo/host/build/python
sudo python3 setup.py install
```

If no errors occur, then the module is successfully installed.

E. Configure USRP USB Permissions

1. Copy udev rules and reload:

```
sudo cp ~/uhd_repo/host/utils/uhd-usrp.rules /etc
    /udev/rules.d/
sudo udevadm control --reload-rules
sudo udevadm trigger
```

2. Add user to groups:

```
sudo usermod -aG plugdev $USER
sudo usermod -aG dialout $USER
```

3. Reboot the system:

```
sudo reboot now
```

F. Verification

After rebooting, verify the installation.

1. Check Device Recognition:

```
uhd_find_devices
```

Expected output should list the B200/B210 device with its serial number.

2. Check Python Bindings:

```
python3 - << 'EOF'
import uhd
print("UHD␣Python␣API␣is␣working")
print("UHD␣Version:", uhd.get_version_string())
EOF
```

Expected output: `UHD Version: 4.x.x`.

NOTE: If you are using a Virtual machine for running Ubuntu, then ensure the USRP is connected to the VM you are using and not to the local host machine. If you have USB 3.0 port and still facing errors in USB configuration,

- Go to the settings of VM
- Select USB Devices
- Change it to 3.0 OR 3.1

Acknowledgment

I sincerely thank Prof. Sundar Vanka for mentoring our work regarding the signal capture through SDR boards, helping us to optimize our codes for the STFT computation and debug the issues faced throughout the project.

References

[1] J. Mitola, "The software radio architecture," in *IEEE Communications Magazine*, vol. 33, no. 5, pp. 26-38, May 1995.
[2] T. A. Kazi and M. Noman, "Implementation of Wide Band FM Receiver on RTL-SDR," *International Journal of Engineering Research and Technology (IJERT)*, vol. 5, no. 5, pp. 580–584, 2016.
[3] "USRP Hardware Driver and USRP Manual: Device Calibration and Frontend Correction," Ettus Research. [Online]. Available: Here.
[4] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," in *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51-83, Jan. 1978.
[5] Ettus Research, "USRP B200/B210 Knowledge Base," National Instruments, 2024.
[6] B. E. Ward, "USRP Tutorial," *PySDR: A Practical Guide to SDR*, 2025. [Online]. Available: Here. [Accessed: Nov. 30, 2025]
[7] J. O. Smith III, "Three-Term Blackman-Harris Window," in *Spectral Audio Signal Processing*, 2011. [Online]. Available: Here.
[8] M. Lichtman, "USRP in Python," in *PySDR: A Guide to SDR and DSP using Python*, 2025. [Online]. Available: Here.
[9] "USRP B200/B210/B200mini/B205mini Series," in *UHD Manual v3.13.1.0*, Ettus Research. [Online]. Available: Here.