Dep. Electrical and Computer Engineering

ECE448 - Advanced topics in Computer Networks

Distributed Spectrum Sensing

Ioannis Roumpos
AEM: 2980
iroumpos@uth.gr

Fall Semester 2023/2024

Contents

1 Configuration of PlutoSDR

In this project, the implementation of the spectrum sensing algorithm is done by using the Python API that PlutoSDR offers. In more detail, the following libraries should be installed before executing the software - the python scripts - in order to provide the right communication between PlutoSDR and the input/output of our system.
libiio, Analog Device's "cross-platform" library for interfacing hardware.
libad9361-iio, AD9361 is the specific RF chip inside the PlutoSDR.
pyadi-iio, the Pluto's Python API, this is our end goal, but it depends on the previous two libraries.
More information about how the setup was done can be found in this link PlutoSDR.Also, it should be mentioned that using the right arguments as they are described in the link above, we can unlock the full frequency range of the chip. This will allow us to extend the range of the spectrum we want to analyze.

2 Received signal using PlutoSDR

Since we want to analyze the spectrum understanding how Pluto receives the signal is vital in order to perform the appropriate signal processing techniques that will allow us to visualize the signal.

2.1 Processing of the Signal

In the figure below we can see how the PlutoSDR handles the incoming signal at a given frequency.

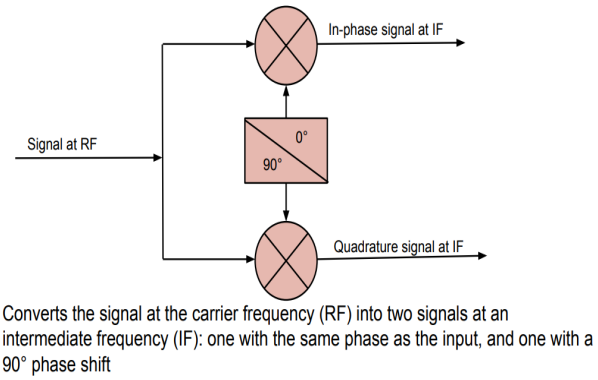

Figure 1: Phase shifter

The signal is divided into two signals at an intermediate frequency one with same phase as the input and one with a 90° phase shift. The first signal is denoted as **I**(In-phase) since the signal's phase remains the same and the other as **Q**(Quadratic) since its phase is shifted. Therefore, the output vector is an **IQ**-signal which is in the time domain and can be represented as an 1-D array of complex numbers.

2.2 Frequency Domain

At the end of the previous stage we got a signal in time domain and we want to convert it into the frequency domain. This will be done using Fast Fourier Transform (**FFT**) algorithm. Initially, raw IQ samples, representing the signal's amplitude and phase at discrete time intervals, are collected. These samples provide a snapshot of the signal's behavior over time. By applying the **FFT** algorithm to this time-domain data, the PlutoSDR efficiently computes the frequency spectrum of the signal. The **FFT** decomposes the time-domain signal into its constituent frequency components, revealing the signal's frequency content and distribution. This transformation enables users to gain insights into the signal's characteristics, such as identifying dominant frequencies, detecting modulation schemes, and analyzing

spectral occupancy.
In the images below, we can see how the FFT works in our case and will be used to calculate in the next chapter the Power Spectral Density which is the metric that will use to inspect the spectrum.

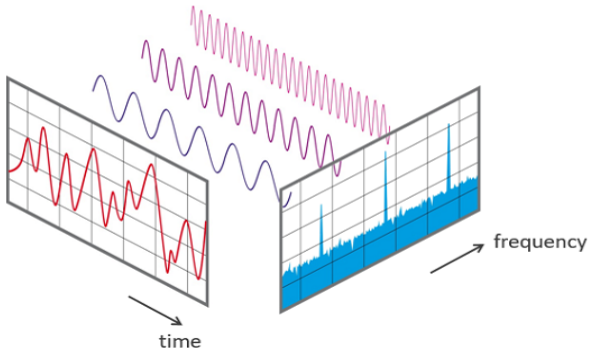

Figure 2: FFT Time Frequency

In the figure below which was taken from PlutoSDR we can observe that the output of the FFT will be produce a vector placed between the half of the sampling rate and centered at zero. That means we can only see signals up to the half of the sampling rate, regardless of how many samples we feed into the FFT.

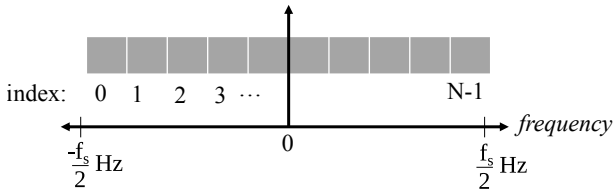

Figure 3: Frequencies Vector

3 Spectrum Sensing

3.1 Energy Detection

Spectrum sensing employing energy detection with Power Spectral Density (PSD) analysis is a fundamental technique in cognitive radio networks. By analyzing the energy level of received signals across different frequency bands, the presence or absence of primary users can be inferred. In this method, the received signal is sampled and its PSD is estimated, providing valuable insights into the spectral characteristics. By comparing the observed energy levels against a predefined threshold, spectrum opportunities can be identified, facilitating dynamic spectrum access and efficient utilization of the available frequency spectrum.
Energy detection with Power Spectral Density (PSD) analysis is closely connected with the Fast Fourier Transform (FFT) in spectrum sensing. By decomposing the time-domain signal into its frequency components, the FFT provides the magnitude and phase information of the signal at different frequencies. This frequency-domain representation allows for the estimation of the PSD, which indicates the distribution of signal power across the frequency spectrum.
In the code below we see a snippet of the code that calculates the psd for a given sample of received signal.

```
def psd_value(rx_samples):
    # Hamming window
    rx_samples = rx_samples * np.hamming(len(rx_samples))
    fft_result = np.fft.fft(rx_samples)
    psd = np.abs(np.fft.fftshift(fft_result))**2
```

```
    return psd

psd_shifted = psd_value(rx_samples)
# Value of PSD in decibel(dB)
psd_dB = 10*np.log10(psd_shifted)
```

3.2 Software Implementation

In this section, the software implementation will be presented explaining the utility of the code that led to successful spectrum sensing.

Firstly, it is important to configure the PlutoSDR attributes with the appropriate values in order to be inline with the devices' specification and user's needs. In this point, i run a script in order to find the max throughput of the device which will be the maximum sample rate that we can cofigure our device.

```
# Configuration for PlutoSDR
    num_samples = 100000

    sdr.gain_control_mode_chan0 = 'manual'
    if start < 4e9:
        sdr.rx_hardwaregain_chan0 = 64.0
    else:
        sdr.rx_hardwaregain_chan0 = 50.0
    sdr.rx_lo = int(start - sample_rate)
    sdr.sample_rate = int(sample_rate)
    sdr.rx_rf_bandwidth = int(sample_rate)
    sdr.rx_buffer_size = num_samples
```

In order to observe the existence or absence of signal in the spectrum we should calculate a threshold value of PSD outside of it. The next block of code is responsible of this calculation.

```
    # Value outside the spectrum
    value = start - 10e6
    sdr.rx_lo = int(value)
    rx_samples = sdr.rx()

    # Threshold value of psd
    threshold = np.mean(psd_value(rx_samples))
    print("Threshold: ", threshold)

    # Number of center frequencies
    num = int((sample_rate/100e3) + 1)

    # Array of frequencies
    freqs = np.linspace(-sample_rate/2, sample_rate/2, num)

    # Array of frequency values of each FFT bin
    fft_freq = np.fft.fftshift(np.fft.fftfreq(len(rx_samples),
    ↪  d=1/sample_rate))
```

In the next box of code, we are presenting the main functionality of the spectrum sensing. The code is accompanied with proper comments that explain the algorithm. In more detail, the logic behind is to iterate through the entire spectrum the user wants to by breaking it into chunks of frequency based on the sample rate and the bandwidth. The collection of received signals is done multiple times over time to avoid a collection of repeated samples. We save the wanted data to the databases in order to plot the results both for PSD-Frequency and a Spectrogram/Waterfall.

```
while temp_freq <= end:
    rx_lo_change.append(start + counter)
    rcv_time = time.time()

    iter = 0

    # 50 samples of received signal
    while iter < 50:
        rx_samples = sdr.rx()
        iter += 1


    psd_shifted = psd_value(rx_samples)
    psd_dB = 10*np.log10(psd_shifted)
    f_axis =
    ↪  np.linspace(next_value-sample_rate/2,next_value+sample_rate/2,
    ↪  len(psd_shifted))

    spectrogram = []
    time_axis = []

    spectrogram.append(psd_dB)
    time_axis.append(time.time())

    # Stores the appropriate data in the databases
    with h5py.File(output_file, 'a') as f:
        f.create_dataset(f'psd_dB_{next_value}', data=psd_dB)
        f.create_dataset(f'f_{next_value}', data=f_axis)

    with h5py.File(spect_file,'a') as fi:
        fi.create_dataset(f'spect_{next_value}', data=spectrogram)
        fi.create_dataset(f'time_{next_value}',
        ↪  data=np.array(time_axis))

    # This loop help us to find the frequency in chunk based on
    ↪  bandwidth and sample rate and will terminate the loop until it
    ↪  reaches the end
    for i in range(num):
        if i == 0:
            avg =
            ↪  np.mean(psd_shifted[frequency_dict[i][0]:frequency_dict[i][1]])
            temp_freq = math.ceil(fft_freq[frequency_dict[i][0]] +
            ↪  sdr.rx_lo)
            list_of_freqs.append(temp_freq)
        elif i == num - 1:
            end_value =
            ↪  np.mean(psd_shifted[frequency_dict[0][0]:num_samples])
        else:
            avg =
            ↪  np.mean(psd_shifted[frequency_dict[i][0]:frequency_dict[i][1]])
            temp_freq = math.ceil(((fft_freq[frequency_dict[i][1]] -
            ↪  fft_freq[frequency_dict[i][0]])/2)
                                  + fft_freq[frequency_dict[i][0]])
                                  ↪  + sdr.rx_lo
            list_of_freqs.append(temp_freq)

    # Next value represents the center frequency in the frequency chunk
    ↪  we examine in each iteration
    counter += int(sample_rate/1e6)
    next_value = start + counter*100e3
    sdr.rx_lo = int(next_value)
```

3.3 Distributed Spectrum Sensing

Distributed spectrum sensing in Software Defined Radio (SDR) systems entails a multi-step process for effectively monitoring the radio frequency (RF) environment across distributed nodes. In the case of this project, each node initializes its SDR hardware, tunes into a specific frequency band, and independently senses the local spectrum to detect primary users and estimate occupancy. The PlutoSDRs have their own local IP in order to be distinguished from each other and in the code below it is shown how they are initialized. The option 2 in the arguments gives the opportunity to the user to leverage both of the SDR devices.

```
sdr = adi.Pluto('ip:192.168.4.1')
if(int(sys.argv[1]) == 2):
    sdr2 = adi.Pluto('ip:192.168.2.1')
spectrum_sensing(sdr, start, end, sample_rate, "sdr1" + filepath_data,
↪  "sdr1" + filepath_spect, bandwidth)
if(int(sys.argv[1]) == 2):
    spectrum_sensing(sdr2, start, end, sample_rate, "sdr2" +
    ↪  filepath_data, "sdr2" + filepath_spect, bandwidth)
```

After the spectrum sensing the results are stored in a database and from there using different methods like energy detection and PSD and a spectrogram the results are vizualised. In that way the use can extract useful information about the spectrum sensing.

3.4 Database

To store the data in a database the library **h5py** of python is used. The **h5py** package is a Pythonic interface to the HDF5 binary data format. It lets you store huge amounts of numerical data, and easily manipulate that data from NumPy. For example, you can slice into multi-terabyte datasets stored on disk, as if they were real NumPy arrays. Thousands of datasets can be stored in a single file, categorized and tagged however you want.

The next figure represents the structure of our database. The group is the parent directory where the (meta)data is stored which are the data that are used for plotting the figures. After each iteration, a new group is created in this hierarchical structure and will ensure that each data from a different frequency band will be stored and edited in a way that the final plot contains all the spectrum the user wanted.

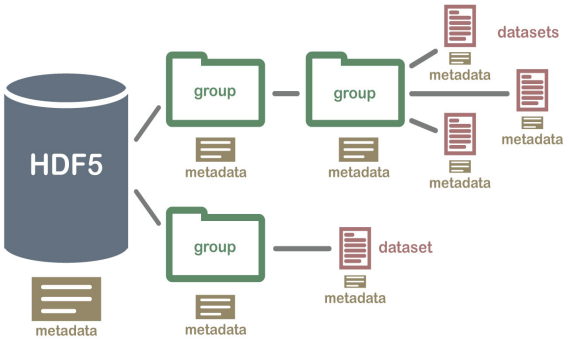

Figure 4: Structure of database

The code below shows how the data are stored in the database and how the plots are created from the database files.

```
    # Stores the appropriate data for spectrum analysis
    with h5py.File(output_file, 'a') as f:
        f.create_dataset(f'psd_dB_{next_value}', data=psd_dB)
        f.create_dataset(f'f_{next_value}', data=f_axis)
```

```
    # Stores the data for the spectrogram
    with h5py.File(spect_file,'a') as fi:
        fi.create_dataset(f'spect_{next_value}', data=spectrogram)
        fi.create_dataset(f'time_{next_value}',
        ↪  data=np.array(time_axis))


def spectrum_results(filepath):
        # Open the HDF5 file
        with h5py.File(filepath, 'r') as f:
            frequencies = []
            psd_values = []
            for key in f.keys():
                    if key.startswith('f_'):
                            frequencies.append(f[key][:])
                    elif key.startswith('psd_dB'):
                            psd_values.append(f[key][:])

        # Concatenate the arrays
        all_frequencies = np.concatenate(frequencies)
        all_psd_values = np.concatenate(psd_values)


        print(all_frequencies)
        # Plot the spectrum
        plt.plot(all_frequencies, all_psd_values)
        plt.xlabel("Frequency")
        plt.ylabel("PSD (dB)")
        plt.title("Complete Spectrum")
        plt.show()

def spectrogram(filepath,sample_rate,start,end):
        # Open the HDF5 file
        with h5py.File(filepath, 'r') as f:
            spectrogram = []
            time_axis = []
            for key in f.keys():
                    if key.startswith('spect'):
                            spectrogram.append(f[key][:])
                    elif key.startswith('time'):
                            time_axis.append(f[key][:])

        # Concatenate the arrays
        all_spectrogram = np.concatenate(spectrogram)
        all_time_axis = np.concatenate(time_axis)

        plt.imshow(np.array(all_spectrogram).T, aspect='auto',
        ↪  extent=[0, len(all_time_axis) * 0.01, start - sample_rate /
        ↪  2, end + sample_rate / 2])
        plt.xlabel('Time')
        plt.ylabel('Frequency (Hz)')
        plt.title('Spectrogram')
        plt.colorbar(label='PSD (dB)')
        plt.show()

```

4 Experiments and Results

In order to test our implementation of this software we performed various experiments across the spectrum. The visualization of the experiments include two diagrams one that shows the Power Spectral Density(PSD) for the spectrum we want to sense, which the main tool of spectrum sensing and one that plot that shows

frequency over time. It is simply a bunch of FFTs stacked together vertically.
The frequencies bands for the experiments are used to represent well-known bands used in every day life.
The comments regarding the absence or presence of the spectrum should be based on the threshold value we get during the calculations and the user can see it while running the script that has the respected software.

For the first experiment we sense between 87.9 MHz to 100MHz with bandwidth of 1MHz.

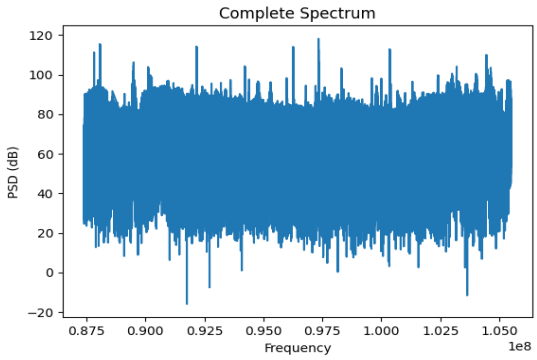

Figure 5: PSD for spectrum from 87,9 to 100 MHz

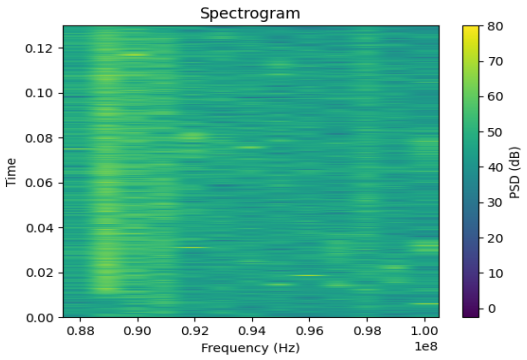

Figure 6: Spectrogram from 87,9 to 100 MHz

For the second experiment we sense between 2,43 GHz to 2,7 GHz with bandwidth of 30MHz.

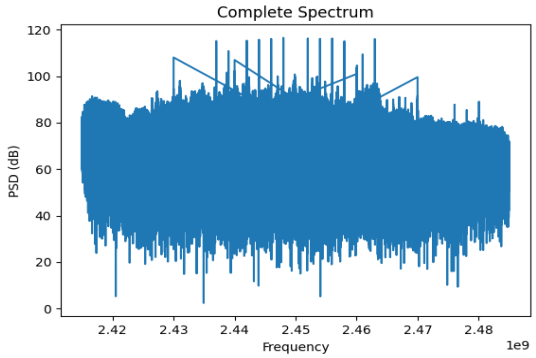

Figure 7: PSD for spectrum from 2,43 to 2,7 GHz

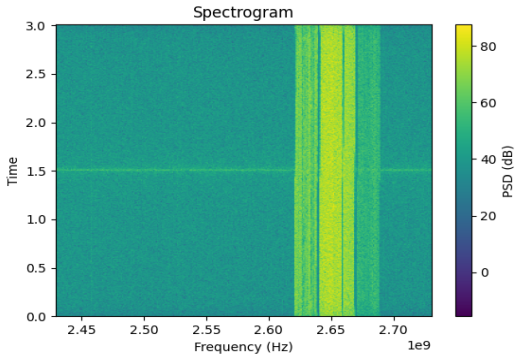

Figure 8: Spectrogram from 2,43 to 2,7 GHz

For the third experiment we sense between 800 MHz to 1 GHz with bandwidth of 15MHz.

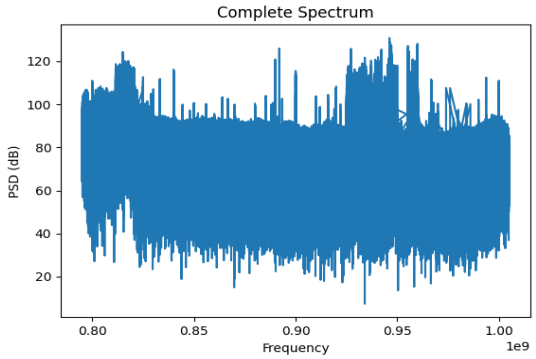

Figure 9: PSD for spectrum from 800 to 1000 MHz

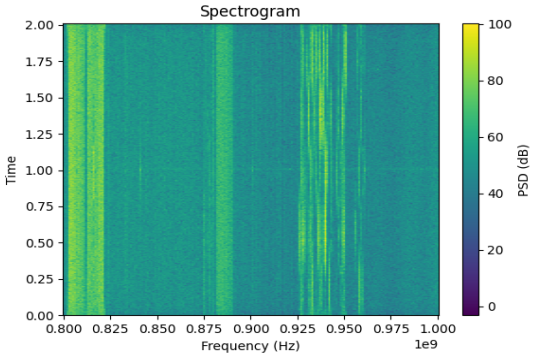

Figure 10: Spectrogram from 800 to 1000 MHz

For the last experiment we sense between 87.9 MHz to 105 MHz with bandwidth of 1MHz but we use both of the SDRs to see and will the contention between those two. This experiment represents the **Distributed Spectrum Sensing** utility of our software.

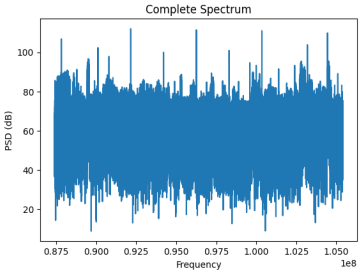

Figure 11: PSD from PlutoSDR #1

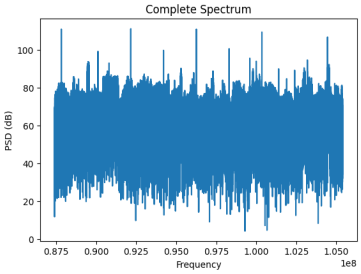

Figure 12: PSD from PlutoSDR #2

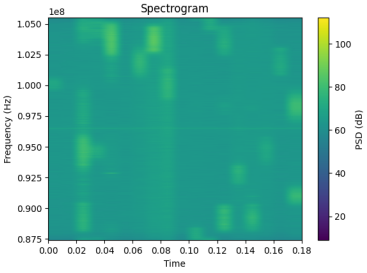

Figure 13: Spectrogram PlutoSDR #1

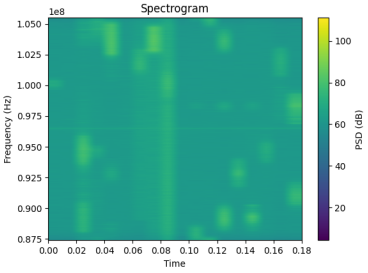

Figure 14: Spectrogram PlutoSDR #2

5 How to run the software

In this chapter, detailed instructions are provided to help the user run the software and visualize the results.
The main software for spectrum sensing is the **spectrum_sensing.py** and is executed like the command below.

python3 spectrum_sensing <option> <file1.h5> <file2.h5>

- **option**: option = 1 there is one PlutoSDR while when option = 2 there are both of the PlutoSDRs.
- **file1.h5**: Provide the name of the database that will be used for plotting the spectrum in respect with PSD.
- **file2.h5**: Provide the name of the database that will be use for plotting the spectrogram.

After running the software the user will be asked to input different parameters of the spectrum like the range of it and the bandwidth.

For visualizing the requested spectrum the user should run the **database_handler.py** script.

python3 database_handler <filename> <option>

- **filename**: Provide the name of the database you want to visualize.
- **option**: option = 1 is for PSD plot and option = 2 for spectrogram.

You should be provide the right pair of arguments since the database name should match the corresponding option. For example, the database file that includes the data for spectrum analysis with PSD should be followed by option 1 and the database with the data for spectrogram should be followed with option 2.

Note: Make sure you have installed the h5py library (pip install h5py) and all the proper dependencies for executing the software.