3 USRP2 Hardware Implementation

This section of the laboratory will familiarize you with some of the useful GNURadio tools for digital communication system design via SDR using the USRP2 platforms. Specifically, you will implement several simple digital communication scripts that wirelessly transmit packets of information.

3.1 Basic Setup

If you have not gone through the host computer configuration steps described in the laboratory tutorial, please do so at this time.

3.2 Frequency Offset

Oscillator crystals are not perfect, since they do not oscillate at exactly the specified frequency. Due to imperfections and tolerances in the manufacturing process, oscillator crystals typically have inaccuracies of about 20 or 50 parts-per-million (ppm), which are guaranteed by the manufacturer. We will ignore other sources of inaccuracy, such as load capacitance, that change the RLC characteristics of the oscillator circuit.

Oscillator circuits are used in the radio frequency (RF) front-end electronics to modulate and demodulate signals to and from RF. Inaccuracies in the oscillator crystal frequency cause errors in the RF carrier frequency. As you may recall from previous classes, even relatively small differences in frequency between the transmitter and the receiver can prevent the receiver from correctly decoding the transmission.

It is also worth noting that oscillator circuits provide the clock source for the digital electronics. In particular, the clocks for the analog-to-digital and digital-to-analog converters affect their sampling rates. Therefore, accurate oscillators are essential for both the RF and digital systems.

The GNURadio FAQ [1] stats that the "USRP2 reference clock stability" is "about 20 ppm unless you lock to an external references." Note that the USRP2 has an input connector for an external frequency reference, but we will not be using it in these labs. Suppose that we select a carrier frequency of 2.45 GHz for our USRP2. An offset of 20 ppm may sound small, but it is quite significant at high frequencies; note that 20 ppm is equal to 20,000 parts-per-billion. Working out the math, 2.45×10^{9} times $20{,}000\times10^{-9}$ = 49,000. In other words, when using a carrier frequency of 2.45 GHz, the worst case frequency offset could be as much as about 50 kHz! Note that if you have two USRPs trying to communicate with each other, where one has an offset of -50 kHz and the other has an offset of +50 kHz, the overall difference in frequency will be 100 kHz!

It is necessary to compensate for this frequency offset in order to achieve successful digital communication between the USRPs. For example, we have measured some of the USRP2s to have frequency offsets of as much as 45 kHz (45 kHz/2.45 GHz = 18 ppm). With compensation, the USRP2 works fine, but without it, digital communication does not work at all.

Please do the following steps to measure and compensate for the frequency offset.

1. Arbitrarily select a carrier somewhere within either of the supported bands of the XCVR2450 daughtercard (2.4 to 2.5 GHz and 4.9 to 5.9 GHz). For example, suppose 2.45 GHz. Note: You probably do not want to choose this exact frequency since everyone in the class will be transmitting at once; there will be interference if everyone chooses this frequency.
2. From the course website, download and run basic_siggen.grc on one USRP and observeFFT.grc on another USRP, as shown in the figures below. Set the freq variable to the center frequency that you chose in step 1. You can change the frequency of the tone being transmitted by changing the *Frequency* parameter in the *Signal Source* block in basic_siggen.grc (it is currently set to 40 kHz).

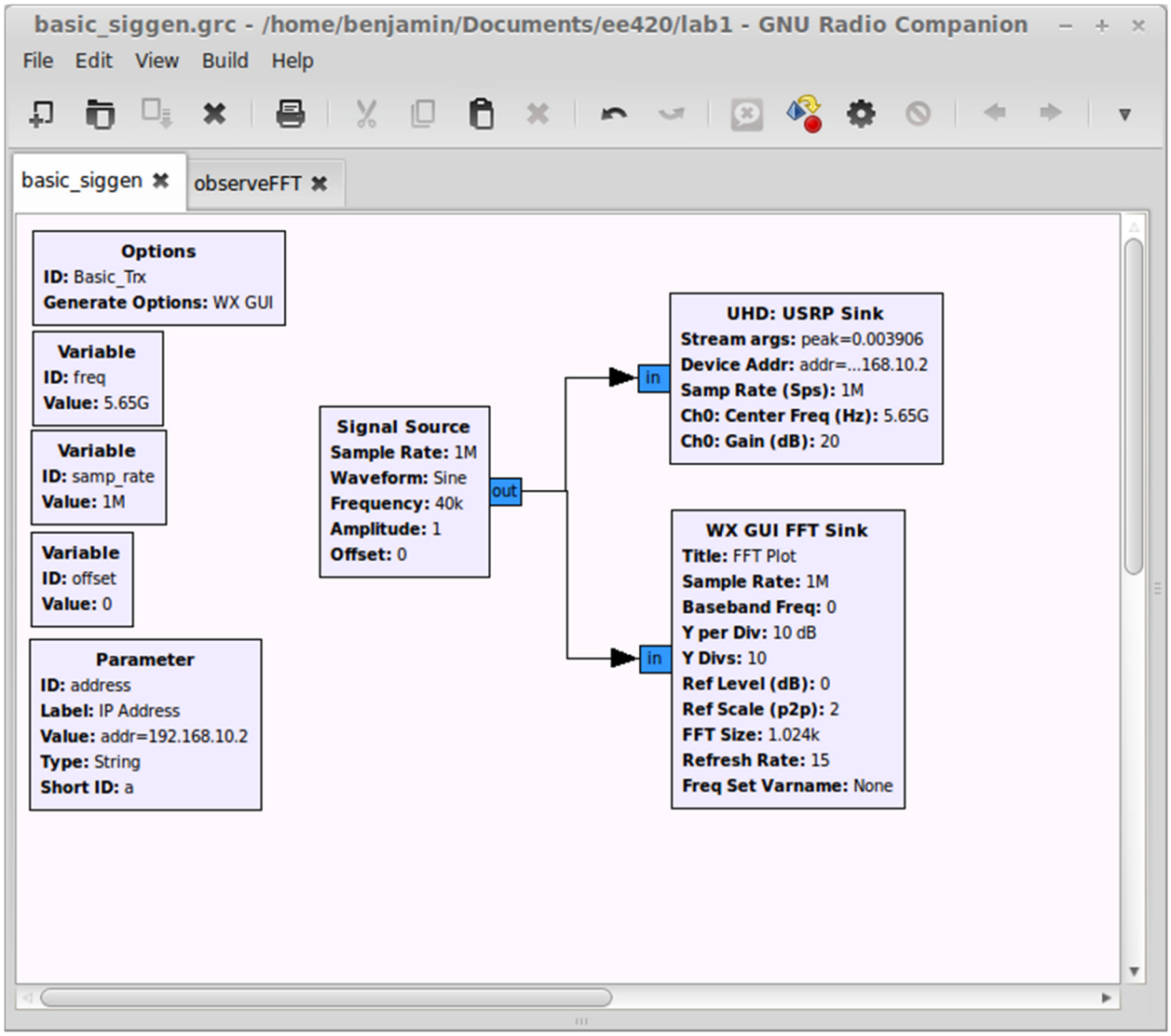

Figure 1: basic_siggen.grc

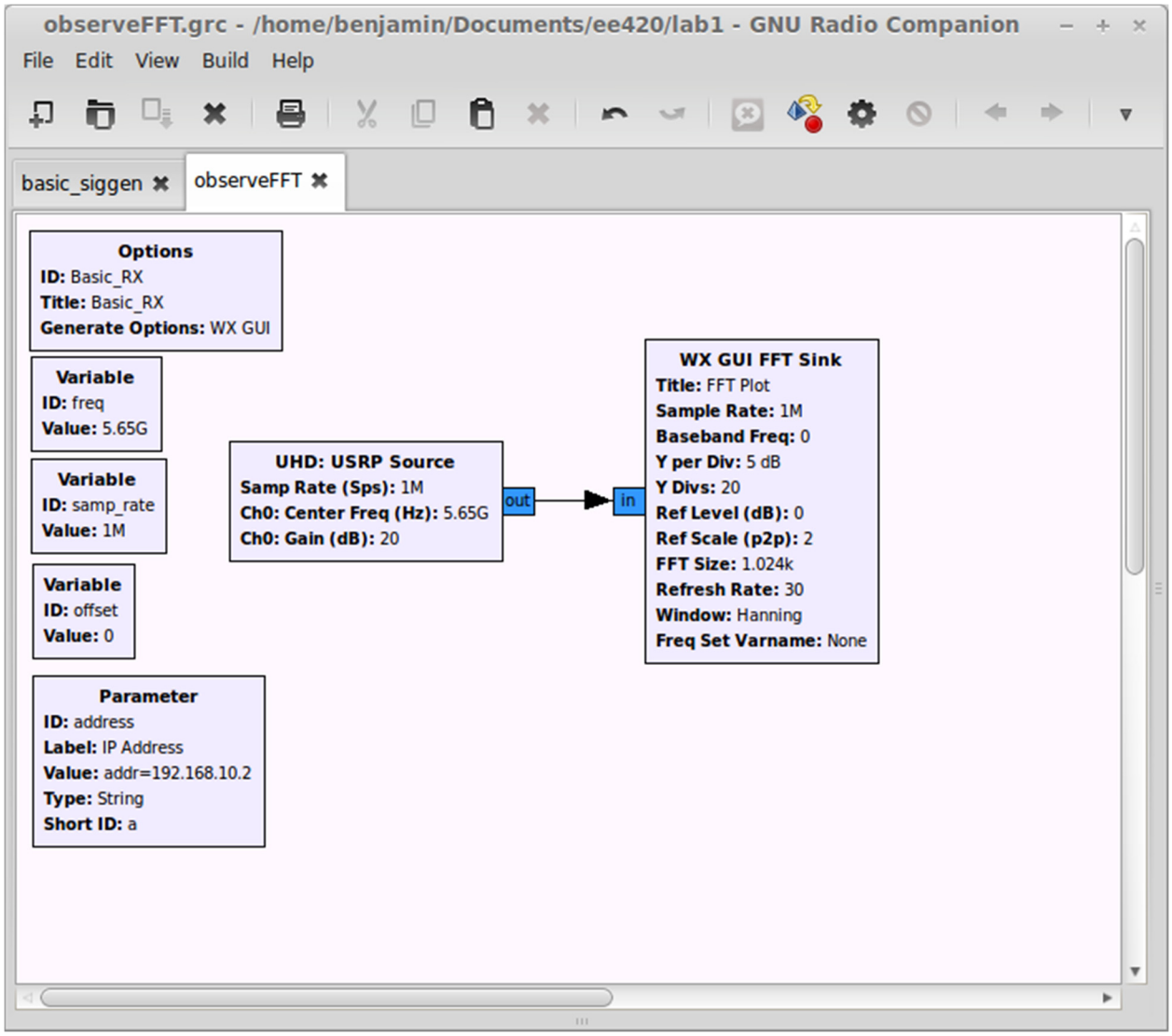

Figure 2: observeFFT.grc

3. Using the FFT plot from WX GUI FFT Sink, measure the frequency offset between the received signal and the desired carrier. In order to have a better observation from the plot, you can click the "Autoscale" button on the right side.

4. Now adjust the center frequency of the carrier frequency of the transmitter by the amount of this offset by changing the *offset* variable. (*Note: if you open up the USRP Sink in basic_siggen.grc you will notice that the value of the CH0: Center Freq (Hz) is "freq+offset" – hence by changing the *offset* variable you are directly changing the center transmit frequency.)

5. By iteratively adjusting the carrier frequency and observing the result, you should be able to determine the carrier frequency that you must use on the transmitter in order to observe the correct carrier frequency on the receiver.

Keep in mind that both the transmitter and the receiver have frequency offsets. For example, "2.45 GHz" on the receiver is somewhere within 2.45 GHz ± 20 ppm, not exactly "2.45 GHz". When you tune the transmitter to eliminate the frequency offset on the receiver, you are essentially compensating for the

offset between them. It is the relative offset between the transmitter and the receiver that matters. Note that you can adjust either the transmitter or the receiver (or both) to compensate for the offset. The basic idea is to get the two devices synchronized in carrier frequency with each other. If you have more than two USRPs communicating with each other, you have to make sure that each pair of transmitters and receivers are all tuned to each other.

3.3 Digital Communication

Now that you have learned how to compensate for the frequency offset, you are ready to experiment with some of the digital communications examples. Do the following tasks concerning a BPSK example:

1. From the course website, download and run `bpsk_tx.pyc` on one USRP and `bpsk_rx.pyc` on another USRP. In order to do this follow the steps below:
   i. Open a terminal and use `cd` to navigate to the directory that the files are in.
   ii. Make sure that the files are executable (you can check this by running the command `ls -l`. If they are not executable run the command `chmod +x bpsk_tx.pyc bpsk_rx.pyc`
   iii. To run the transmitter with a center frequency of 5.31 GHz, run the command `./bpsk_tx.pyc –f 5.31e9` (alternatively, you can invoke the `.pyc` files with python – i.e. `python bpsk_tx.pyc –f 5.31e9`). Do the same for the receiver on the other USRP. You can change the center frequency by changing the value after the `–f`

Please note that for the carrier frequency you should use the values that you found in Section 3.2 that compensate for the frequency offset (this can be done on the transmitter by specifying a frequency of 5.310004e9 if for example you wanted to transmit at 5.31 GHz + 4 kHz).

In the terminal of the receiver computer you should see the received packets. What do you observe?

Note: you might need to start the transmitter before starting the receiver as well as restart the receiver if you stop and restart the transmitter.

2. Now try running these files using the carrier frequencies **without compensation**. What do you observe? Is it any different?

   Some of the USRPs have more severe frequency offsets than others, depending on the severity of imperfections in the oscillator hardware. On some of the USRPs, you will see an increase in the number of incorrect packets. Some of the USRPs will not receive any packets at all because the frequency offset is so severe.

3. Now on the receiver run the command: `./bpsk_rx.pyc –f 5.31e9 | tee output.txt`. This will copy everything being printed to the terminal to the file `output.txt`. (You can rename your output file to whatever you like.) Repeat the previous two steps. Is there a noticeable difference in the amount of errors in the received packets?

4. As you may recall from previous classes, frequency synchronization between the transmitter and the receiver is crucial. To get a feel for this, try experimenting with a few other carrier frequency offsets, stepping in increments of about 10 kHz or 15 kHz. Which offsets result in many incorrect packets? Is there a threshold between being able to receiver some packets and not being able to receive any? What is this threshold?
5. Try changing the distance between the transmitter and receiver. Try moving the transmitter and receiver to disrupt the line of sight communication between them to increase channel impairments such as multi-path interference. Observe how this affects the packet error rate.

3.4 IEEE 802.11 Wireless Local Area networks

Another interesting experiment is using the USRP2 to observe the spectrum of wireless local area networks within the vicinity (WLANs). Most high population density areas employ numerous wireless communication networks for a variety of applications, such as the WPI wireless network. Consequently, you can use your USRP2 experimentation platform to plot their magnitude spectrum. IEEE 802.11 [3] is one type of WLAN standard that possesses a list of carrier frequencies for a collection of Wi-Fi channels. For example, the IEEE 802.11 standard defines channel 1 of the 2.4 GHz band to be centered at 2.412 GHz.

- Using observeFFT.grc specify the "Center Frequency" parameter of the USRP receiver with this carrier frequency to observe the spectrum. It may also be helpful to increase the sample rate (up to a maximum of 20 MSamples/s – although if this proves difficult for your computer to handle 5 or 10 MSamples/s should work as well).
- Since your XCVR2450 daughter card supports the 5 GHz band as well, use observeFFT.grc to plot spectrum in the 5 GHz band.

What did you observe? Were you able to find any active WIFI channels? Are these WIFI signals present all the time?

Note: You might want to use the Autoscale button on the right hand side of the FFT plot and adjust the amplitude, since the peaks of these wireless signals could be rather low.

4 Open-ended Design Problem: Automatic Frequency Off-set Compensator

4.1 Introduction

Beginning from this laboratory, you will be given an open-ended design problem at the end of each experiment. These problems are related to what you have learned and done in each laboratory, but require additional thinking and investigation. There is no single solution for these problems, and each student group is expected to come up with their own innovative design.

4.2 Objective

The objective of this problem is to design and implement a software-defined radio (SDR) communication system capable of automatically calculating the frequency offset between two USRP platforms.

In Section 3.2, you have already found the frequency offset of two USRPs manually, namely, by comparing the FFT of the transmitted and received signals. In this problem, you are expected to discover this offset in an automatic way. In other words, in the receiver model you have designed, the only thing you need to do is to hit the "start simulation" button, which will result in this simulation providing you with the value of the frequency offset.

4.3 Theoretical Background

1. Before you apply your method to the USRP2 boards, it is highly recommended that you test your method by simulating it in GNURadio without the USRPs first and see whether it works.
2. In the GNURadio-only simulation, you can use a signal source to transmit a Sine Wave at a specified frequency. You can then see if you simulation can detect this frequency.
3. An important hint: If you take the square of a signal, the FFT of the received signal will be shifted double of the frequency offset.
4. In your simulation, you might need the following blocks:
    a. Signal Source: analog.sig_source_c
    b. FFT: fft.fft_vcc
    c. Complex to Magnitude Squared: blocks.complex_to_mag_squared
    d. Argument Max: blocks.argmax
    e. Add Constant: blocks.add_const
    f. Multiply Constant: blocks.multiply_const
    g. Vector Sink: blocks.vector_sink

    Note about vector sink: After the simulation has run, you can access the data in the vector sink by accessing its "data" field. For example, if you named your top_block *as* tb *and inside you had a vector sink called* my_vector_sink*, you could print the data in the vector sink using the command* tb.my_vector_sink.data.

    You are not required to use all of these blocks, they are merely suggestions. You can use all the blocks available in GNURadio.
5. Although you are required to find the frequency offset of the two USRP boards, you are actually trying to find the frequency offset of the received signal.
6. Compare your result here from what you have obtained in section 3.2.