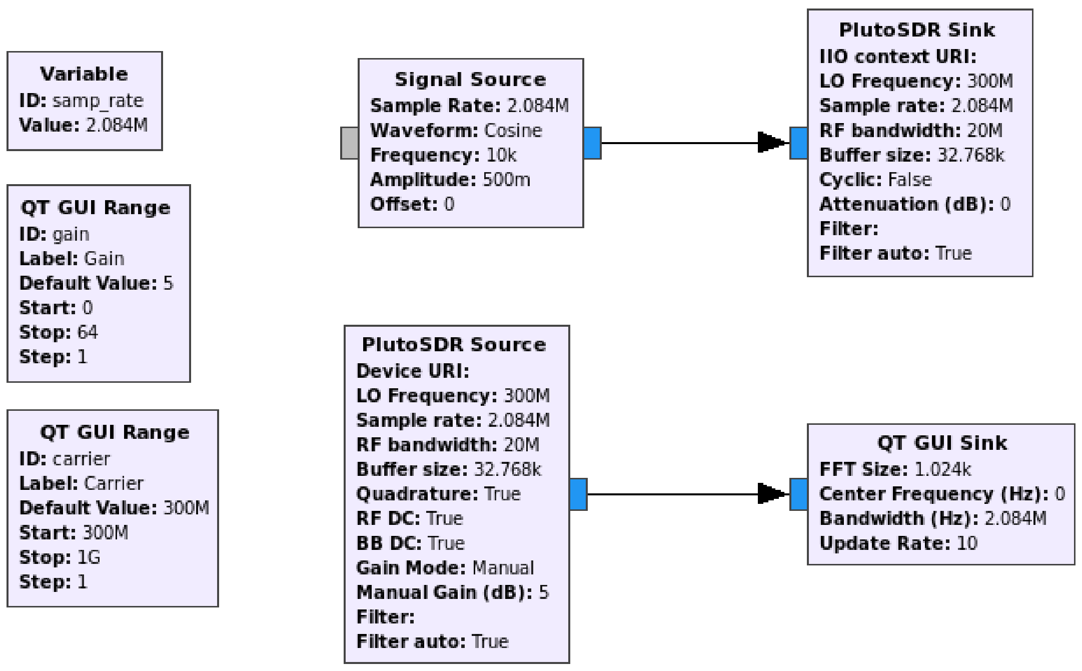
The software package we currently use is GNURadio. This package communicates with the PlutoSDR by using the LIBIIO library (over usb). Below you will find a straight-forward block diagram to transmit and receive a single sine wave with the same device:



Variable
ID: samp_rate
Value: 2.084M

QT GUI Range
ID: gain
Label: Gain
Default Value: 5
Start: 0
Stop: 64
Step: 1

QT GUI Range
ID: carrier
Label: Carrier
Default Value: 300M
Start: 300M
Stop: 1G
Step: 1

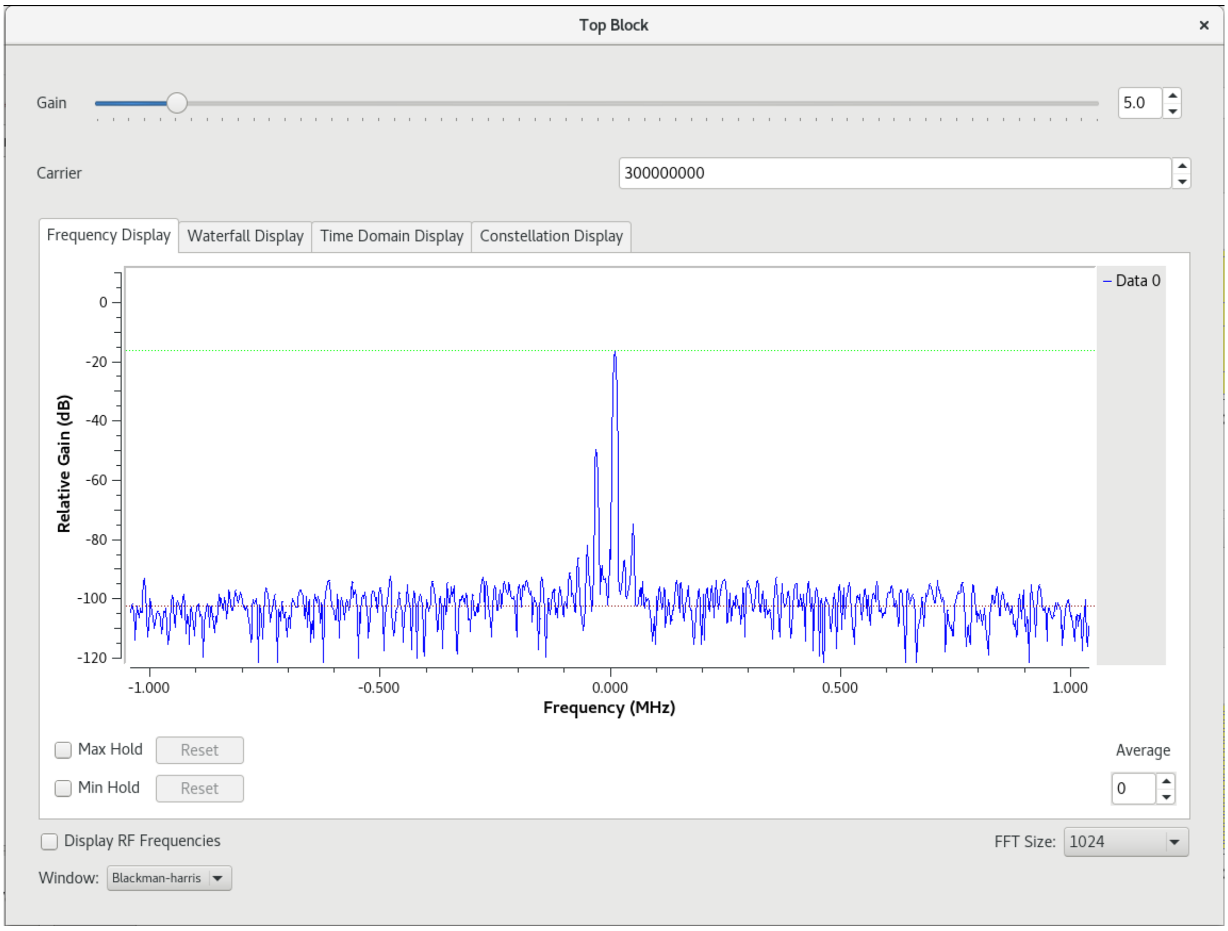
Signal Source
Sample Rate: 2.084M
Waveform: Cosine
Frequency: 10k
Amplitude: 500m
Offset: 0

PlutoSDR Sink
IIO context URI:
LO Frequency: 300M
Sample rate: 2.084M
RF bandwidth: 20M
Buffer size: 32.768k
Cyclic: False
Attenuation (dB): 0
Filter:
Filter auto: True

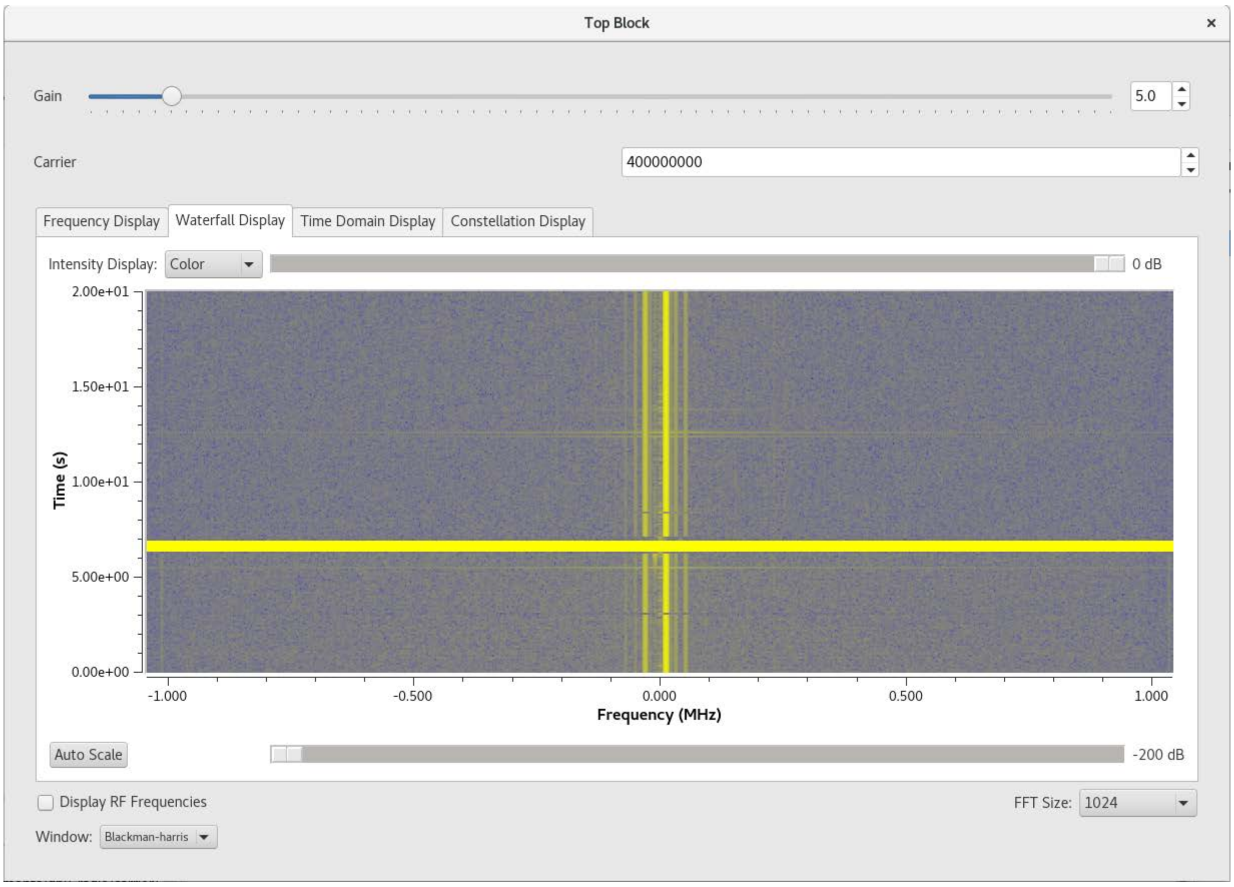
PlutoSDR Source
Device URI:
LO Frequency: 300M
Sample rate: 2.084M
RF bandwidth: 20M
Buffer size: 32.768k
Quadrature: True
RF DC: True
BB DC: True
Gain Mode: Manual
Manual Gain (dB): 5
Filter:
Filter auto: True

QT GUI Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 2.084M
Update Rate: 10

As you can see, I also implemented a numeral input variable for the local frequency (ID: carrier). When this block diagram is compiled into the corresponding python code, the following GUI is created, which allows us to change the local frequency 'on the fly':



In the waterfall display below, we can see the FFT plot over time where the amplitude is represented by a color. I think this is where the miscommunication happened; Whenever the local frequency is changed, the device cannot transmit or receive any samples to or from the device for ~1 second (between ~5.1 and ~5.2 seconds in the graph below). We called this a 'reboot' in our previous e-mail because the device is unresponsive for a noticable amount of time, but we probably should have called it an 'unavailability' since the device is not actually rebooting, but merely changing the local frequency attribute.



Our purpose is to have many adalm plutos communicate with each other over a set of carrier frequencies. For this to happen, each unit needs to claim an available (unused) frequency and do a frequency sweep to find other devices transmitting on other frequencies. If we need one second to switch per frequency, the sweeping process will take too long. We need to get this 'transition' time down to a few milliseconds - preferably even less - otherwise the device won't be usable for our purpose.

I looked through the code and found the following:

The python function that we get from GNURadio to change the local frequency is this:

```
def set_carrier(self, carrier):
    self.carrier = carrier
    self.pluto_source_0.set_params(int(self.carrier), self.samp_rate, 20000000, True, True, True, "manual", self.gain, '', True)
    self.pluto_sink_0.set_params(int(self.carrier), self.samp_rate, 20000000, 0, '', True)
```

The 'set_params' method above calls the following C function from the gr-iio library: https://github.com/analogdevicesinc/gr-iio/blob/85cf3200691ddf57f84001f0489f0a0ce4c34532/lib/pluto_sink_impl.cc#L66

If I follow the chain of all functions that call each other, the C function linked from the gr-iio repository above links back to this command from the libiio repository: https://github.com/analogdevicesinc/libiio/blob/560027bfa271c7c994ca5c383c9030f26a5b1d88/examples/ad9371-iiostream.c#L95

Assuming I observed this chain of functions correctly, does that mean that since GNURadio is directly writing the lo_fr (local frequency) attribute to the ad9371 chip using the libiio library, that there is no more optimal or faster way of doing this? Does this mean that we are stuck with the delay of 1 second to change between frequencies? Is this a limitation by the method of communicating between the petalinux mcu with the ad9371 chip inside the adalm pluto or a limitation from communicating with the device externally over usb? Is there other software we can use (than GNURadio) that allows us to do a frequency sweep at a higher speed?