Work Experience - RF Guide

Patrick Mintram

May 31, 2019

Contents

List of Figures

1 Introduction

This guide has been produced to help you work through the Radio Frequencies (RF) workshop as part of your work experience. In this workshop you will learn

1. What things use RF.

2. How we can make something that uses RF.

3. How using RF can expose your projects to vulnerabilities.

4. What tools we can use to help when using RF.

1.1 What you will be doing

In following this workshop you will be using tools available at home, to look at some of the information being sent through the air as RF. You will be able to see the different frequencies used by different kinds of devices, such as doorbells, WiFi, remote control cars and bluetooth connected items. You will then send some secret messages between some microcontrollers and use these tools to spy on the message as part of a man in the middle (MITM) attack.

1.2 Using this guide

There may part of this guide which aren't explained very in depth, that is because the subject of RF and signals is really complicated, so the detail has been left out. If you want to find out more there are some good overviews available online[1]. This guide is meant at more of a practical workshop than an academic exercise, so if something is glossed over a useful link will be provided in the footnotes, as you have already seen. It isn't expected that you full understand the subjects covered, but it is expected that you'll take some time in the future to have a play with the tools and techniques and learn a bit more about the things you've touched on.

1.3 Feedback

The author of this guide is keen to know what you think; the good, the bad and the ugly. Please feel free to send any comments their way, or if you're that way inclined use the github system and raising an issue or creating a pull request.

---

[1]http://www.ti.com/lit/ml/slap127/slap127.pdf, for example

2 Equipment

In order to complete this guide you will need the following equipment.

1. Laptop with the following:

    (a) The Software Defined Radio (SDR) Drivers. These are usually available from the manufacturers website.

    (b) The Arduino Integrated Development Environment (IDE)[2].

    (c) The RadioHead-Extras library should be installed and made available to the Arduino IDE. The library is in the `src` folder of this repo.

    (d) gnuradio[3].

    (e) A clone of this repo and performed recursively[4].

2. An SDR with an appropriate antenna for looking at the 430-440MHz frequency range. Its up to you which you use, there are loads available for a reasonable price[5].

3. Two Adafruit Feathers with an RFM69 packet radio module attached[6] as shown in fig. 1. These should ideally have antennas attached as described in the Adafruit documentation[7].
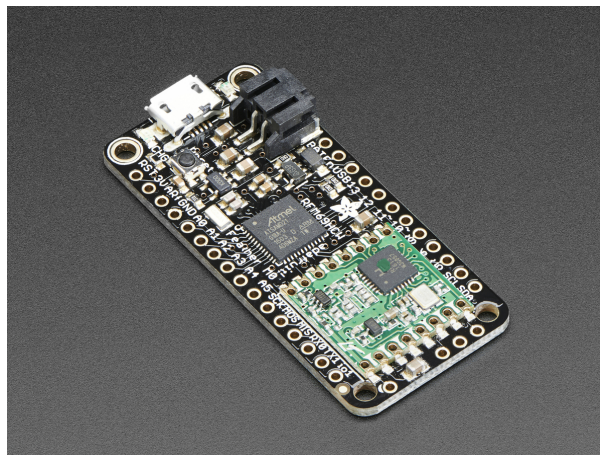

Figure 1: An Adafruit Feather M0 with RFM69 Packet Radio

---

[2]https://www.arduino.cc/en/Main/Software

[3]https://www.gnuradio.org

[4]git clone –recursive https://github.com/geekskick/wex-guide

[5]https://www.rtl-sdr.com

[6]https://learn.adafruit.com/adafruit-feather-m0-radio-with-rfm69-packet-radio/overview

[7]https://learn.adafruit.com/adafruit-feather-m0-radio-with-rfm69-packet-radio/antenna-options

3 Looking at the Spectrum

The first thing we need to understand is what the RF spectrum looks like. There are plenty of electromagnetic waves around, which you may or may not be aware of. Let's quickly revise what a wave looks like, by looking at fig. 2[8].
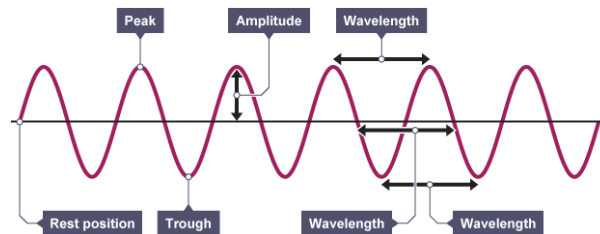

Figure 2: The RF spectrum

The key thing we care about from this diagram is the wavelength because that determines how long it takes for the wave to happen; it's period. This can be used to calculate how many times it' repeats in a second, this is measured in *Hertz* and is a result of the equation shown in eq. (1).

$$\text{Frequency (Hz)} = \frac{1}{\text{Time for one cycle of the wave (s)}} \tag{1}$$

For example a signal that repeats every 2.309469 nanoseconds has a frequency of 434MHz - it repeats 434 million times a second. There are loads of different frequencies in the RF spectrum and 434MHz fits in the ultra high frequency (UHF) part of this, as shown in fig. 3[9].
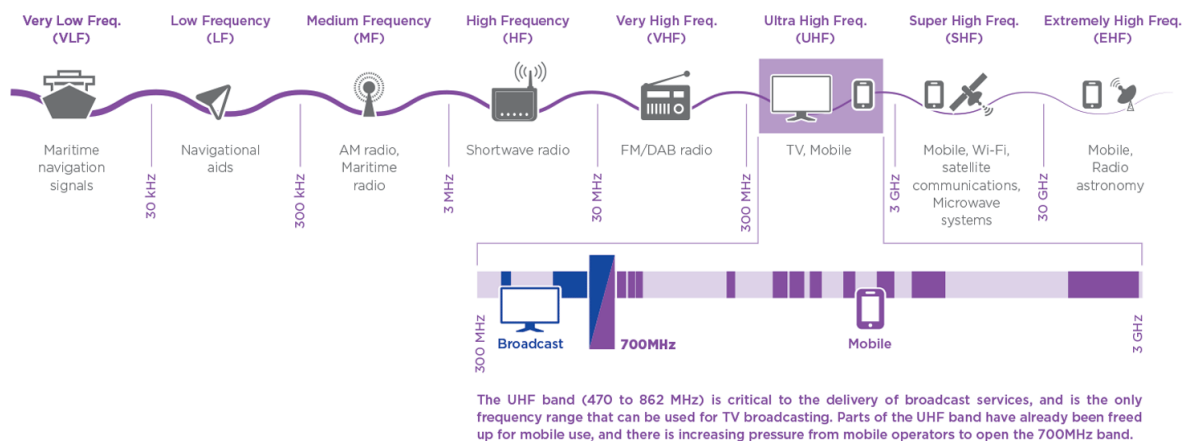

Figure 3: The RF spectrum

We can easily see the effect of these signals by using equipment which uses them; if our radio doesn't work then we know that the signals aren't present at 90MHz. What about if we want to see a signal at

434MHz though? The radios in our car only tune into parts of the very high frequency (VHF) frequencies so we can't use those. Here is where our SDR comes in useful because we can tell it a frequency to tune into (centre frequency) and we can tell it how quickly to process data (sample rate). We can then plug this into the gnuradio software to see on a graph which frequencies are most powerful, as seen in fig. 4.
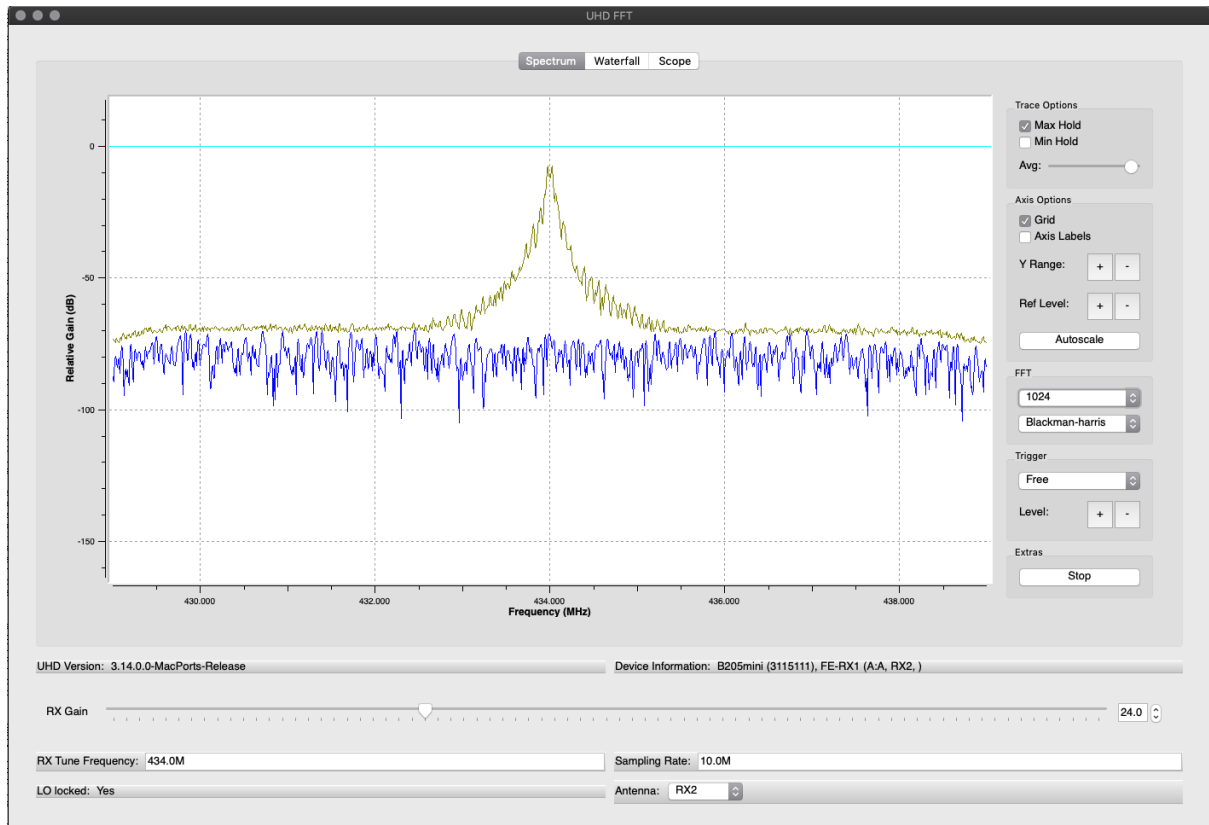

Figure 4: The output of the uhd_fft command with a 434MHz signal present.

The spike on the yellow line in fig. 4 shows that there is some signal present at the 434MHz frequency. We can change the settings on this window to see other signals which might be present, but first you need to run the following command from the command line to open the window.

```
uhd_fft −f 434000000 −s 10000000
```

Try looking somewhere around 2.4GHz and seeing how busy it is. What do you think these signal might be?

3.1 gnuradio flows

Rather than using a prepackaged command like uht_fft we can make our own graphical user interface (GUI)s using gnuradio companion. This can be a bit weird, so to start you off one has been provided. From the command line enter:

```
gnuradio−companion
```

From here open up the provided file FFT.grc and you can click the play button highlighted in fig. 5 and see the spectrum as we did before. At this point it's worth taking some time to familiarise yourself with gnuradio using the tutorial available here: `https://wiki.gnuradio.org/index.php/Guided_Tutorial_GRC`, that way it wont be a shock if the instruction is to 'add in the Throttle block', for example.
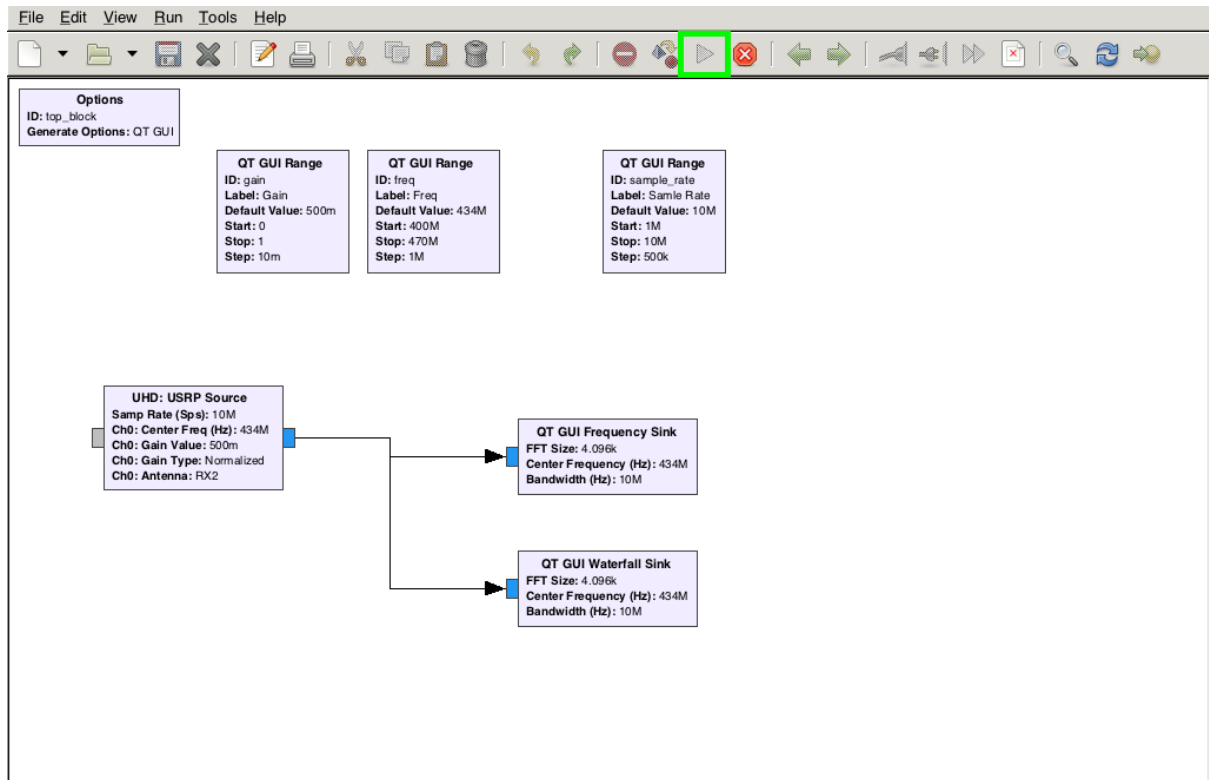
Figure 5: gnuradio flow for viewing the spectrum

4 Sending a secret message

In this section we will use our feathers to send some messages to each other. Fortunately this is pretty simple to get started with thanks to the code provided by the RadioHead-Extras library provided.

Open up the *SimpleFSKSend.ino* sketch and the *SimpleFSKRx.ino* files in the arduino editor and load them on to different feathers. You should see that one is sending a message and the other is receiving it, and printing it to the serial monitor. This is our secret message!

Verify that it is working, and then try changing the code to send another message. For now, keep the settings of the RF chip the same, we will come back and change them later.

5 Spying on a secret message

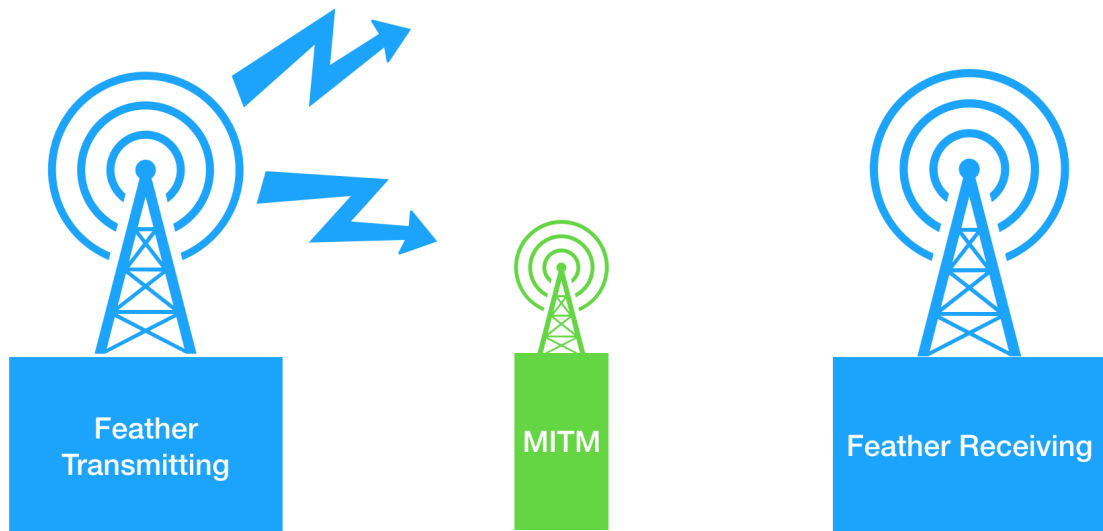In this section we will make a custom gnuradio flow to perform a MITM attack on our two feathers, as shown in fig. 7.

Figure 6: MITM attack on our two feathers

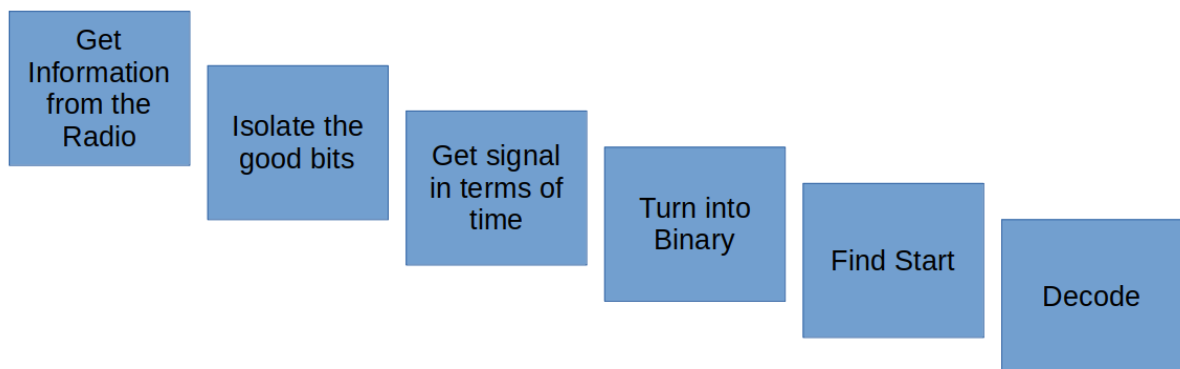In this section will will walk through the steps to looking at the data send over the air.

Figure 7: The steps to looking at the data sent

5.1 Connecting the Radio

In order to get data into the flow from the SDR we must use a source. In this case, it's the RTL-SDR source (In my examples I will be using a slightly different one, however the main points are the same) if there are issues there are plenty of resources available online to work through the RTL-SDR specifics [10].

---

[10] https://www.instructables.com/id/RTL-SDR-FM-radio-receiver-with-GNU-Radio-Companion/

Drag in the RTL-SDR source to the canvas.

Figure 8: A gnuradio flow with just a radio source

If you double click the block you will get some options, in here the key things to find are the *sample rate*, *gain*, and *centre frequency*. The sample rate affects how quickly the radio is taking measurements (the rate at which it samples!) and as a result how wide the spectrum we are monitoring is. The gain is how loud that measurement is. The centre frequency is where the middle of the spectrum we are observing sits. The best way for us to experiment with these settings is to use the *QT GUI Range* widget, this widget givs us a slider on the GUI to change settings at runtime. In order to do this, first, double click the widget's block and give it a meaningful ID, something like *gain_slider*. Give it a start value of 0, a stop value of 1, and a step of 0.1.
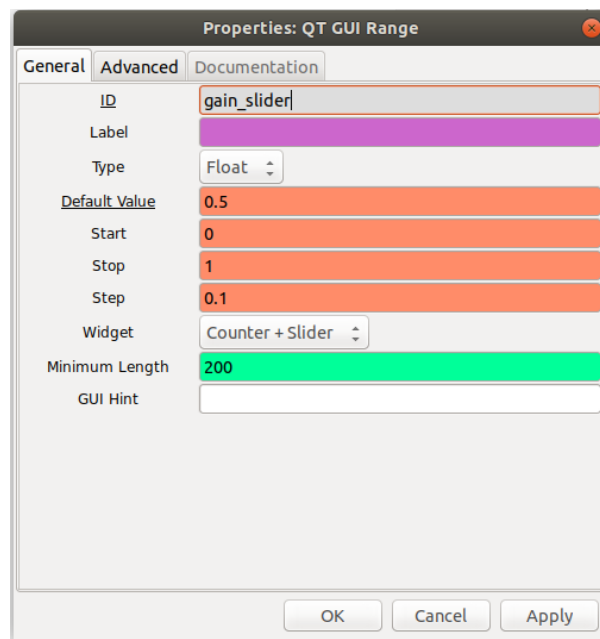

Figure 9: Setting up a slider

Now you can enter it's unique ID in the radio's settings under *gain*. This is effectively telling the radio to get it's gain value from the slider called *gain_slider*.
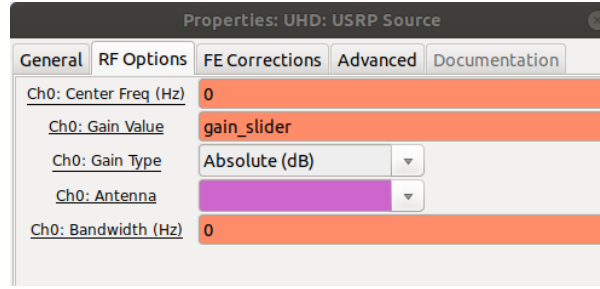
Figure 10: Using a slider value for the gain in the radio source.

Do the same, creating a slider for the *centre frequency* and *sample rate* with the following settings:

Setting	Start	Stop	Default	Step
Centre Frequency	430000000	440000000	435000000	500000
Sample Rate	1000000	10000000	2000000	500000

Table 1: Radio slider settings

5.1.1   Viewing the spectrum

The next thing is to actually be able to see the spectrum! For this a *QT GUI Frequency Sink* is required. Drag this in and connect it to the radio by clicking the blue tabs.
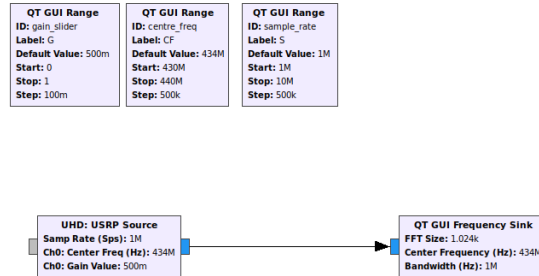

Figure 11: A basic flow.

Again, open the frequecy sink block and set the centre frequency to the same slider as your radio. Set the bandwidth to your sample rate slider:
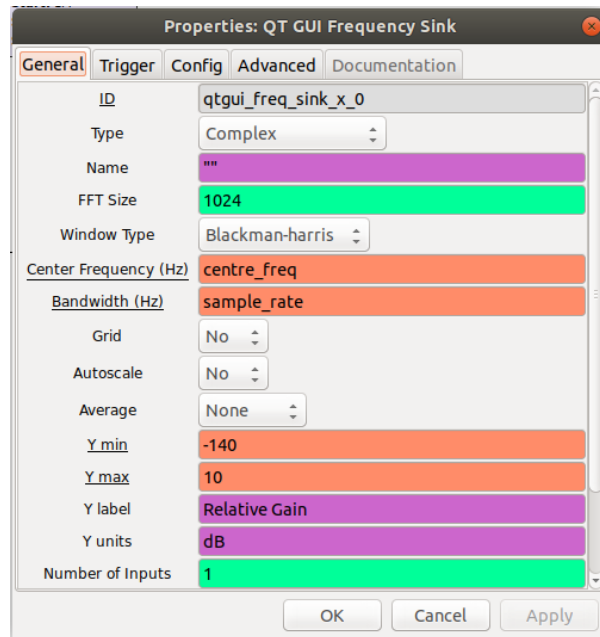
Figure 12: FFT Sink Configuration.

While you're there, go to the *Config* tab and select *yes* to control panel. Now run your flow by clicking the play button at the top of the gnuradio window. This will turn your flow into a python script and run it. If there are errors address these before continuing. All being well, the window that pop up should have three sliders, and a very familiar looking spectrum. Take some time to play with the sliders and see what they do to the spectrum view. (It helps if you enable *max hold* in the spectrum control panel.

5.2 Isolating the good bits

The next step is to use an low pass filter (LPF) to make sure we receive only the transmission we are interested in. Drag an LPF into the gnuradio flow, and add another QT Frequency Sink to see the output. In addition, it's a good idea to add some more QT Range Widgets so you can change the parameters of the filter as it's running. The parameters of interest here are the *cutoff frequency* (how wide the filter is) and the *transition width* (how slopey it's edges are).
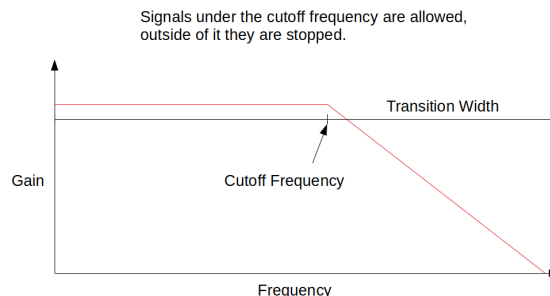

Figure 13: Where the cutoff frequency and transition width matter.

Our aim for the LPF is to have an output that looks abit like barad-dur.
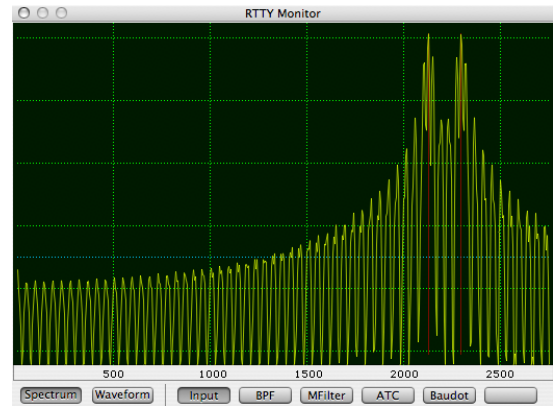

Figure 14: Sauron's home.

Figure 15: FSK Signal as seen on a frequency plot.

Run your new flow and you should be able to use the filter sliders to isolate specific parts of your signal. I found a value of around 20000 for both to give pretty good results during experimenation. We can further help our filter by using *squelch*. This block will only let signals above a certain amplitude through, effectively cutting out all the noise. Put this block after your filter, and set it's *alpha* to 1. Again, make a slider for it's squelch value and play with it - experiments proved that a value of -40 gave good results for me.
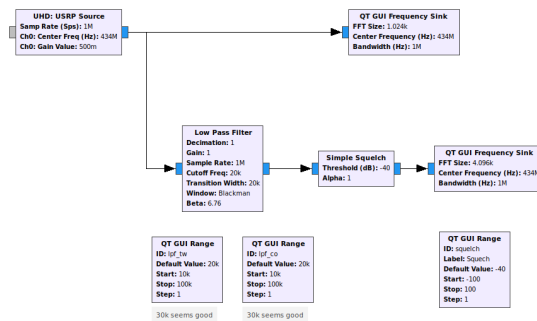

Figure 16: Radio - LPF - Squelch - Frequency Sink

5.3 Get signal in terms of time

Now we have isolated the peaks, we can turn these into bits by using a *Quadradure Demod* block with a gain setting of 1. Plug the output from this into a *QT GUI Time Sink*, and enable the time sink's control panel the saem way you did with the frequency sink.
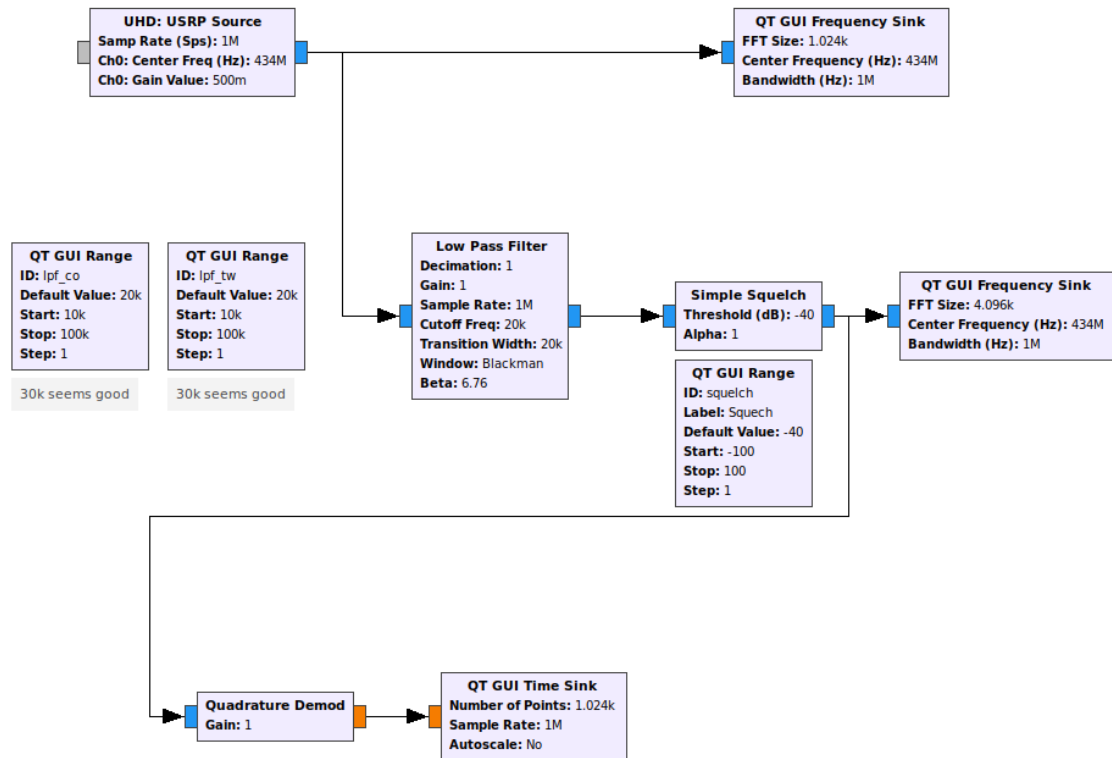
Figure 17: Radio - LPF - Squelch - Quad Demod - Time Sink

This new graph will have turned the frequency changes into changes against time. This will be easyier to see in the Time Sink if you set the trigger to auto, and then click '+ X Max' loads until you can see what looks like something interesting.
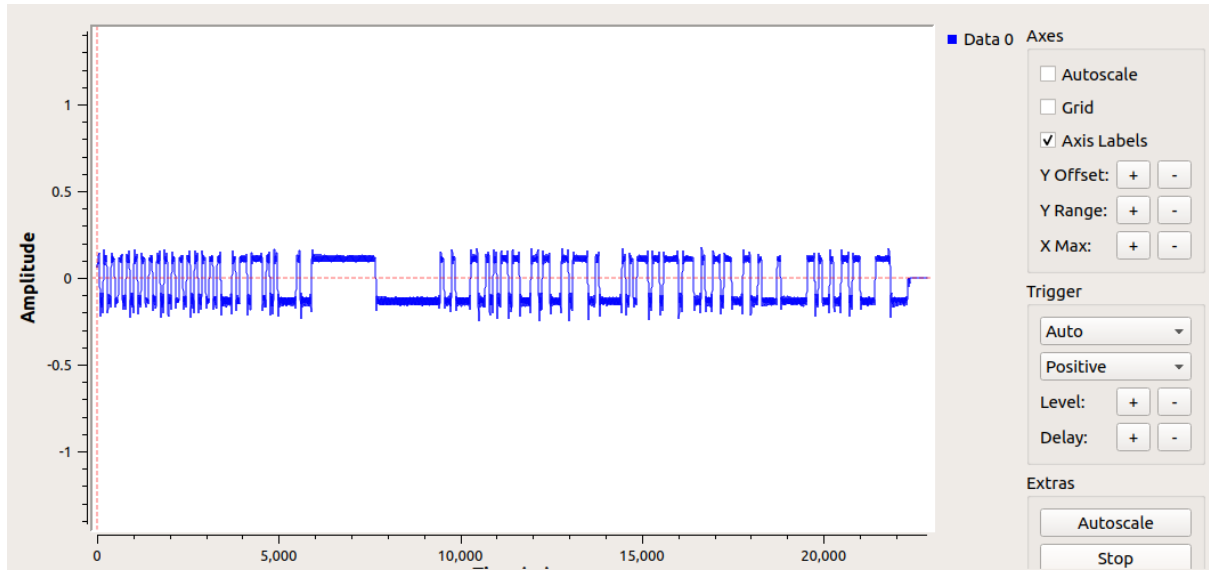
Figure 18: Our signal against time

5.4 Turn into binary

In order to turn the signal into binary we need to know exactly when the transition happens. In this section we will do that using the *Clock Recovery MM* block and a *binary slicer*. The clock recovery relies on knowing the deviation (how far apart your peaks are in the frequency plot) and your baud rate (how quickly bits are sent in the data). Since we made the feathers talk to each other, we know these settings - so it becomes a reltively easy task of plopping numbers into an equation. In the real world however the process of finding this information may take some time and extra snooping around the RF spectrum and settings. Drag in 3 *variable* blocks and use the following information:

Variable Name	Value
baudrate_hz	9600
deviation_hz	19200
samples_per_symbol	sample_rate/baudrate_hz

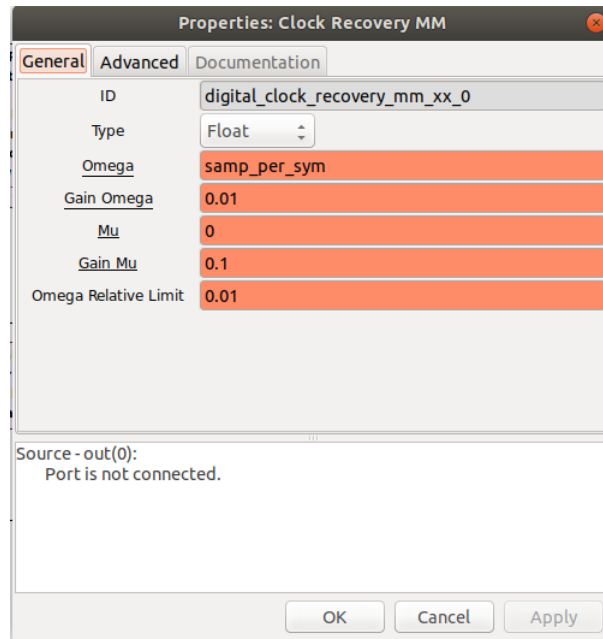Then, in your clock recovery block set the *omega* to be *samples_per_symbol*.

14

Figure 19: Clock recovery block settings

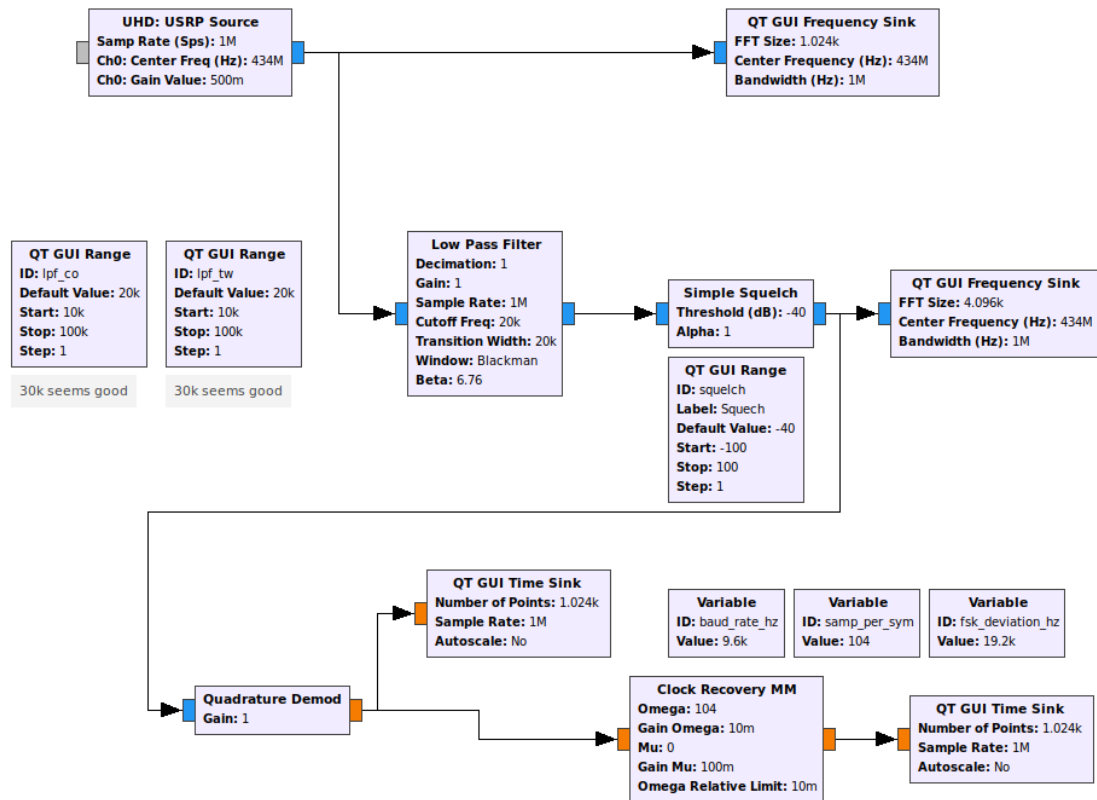Again, connect a time sink to this to see the output.

Figure 20: Radio - Filter - Squelch - Quad Demod - Clock Recovery

What you should see is a slight difference between the time sinks for just the quad demod and the clock recovery. The clock recovery one will look a bit pointier, and the lines will be less thick. This is clearest at the start of the graph - one data point will be high, and the next low whereas without the clock recovery this part of this signal has 'fatter' highs and lows.
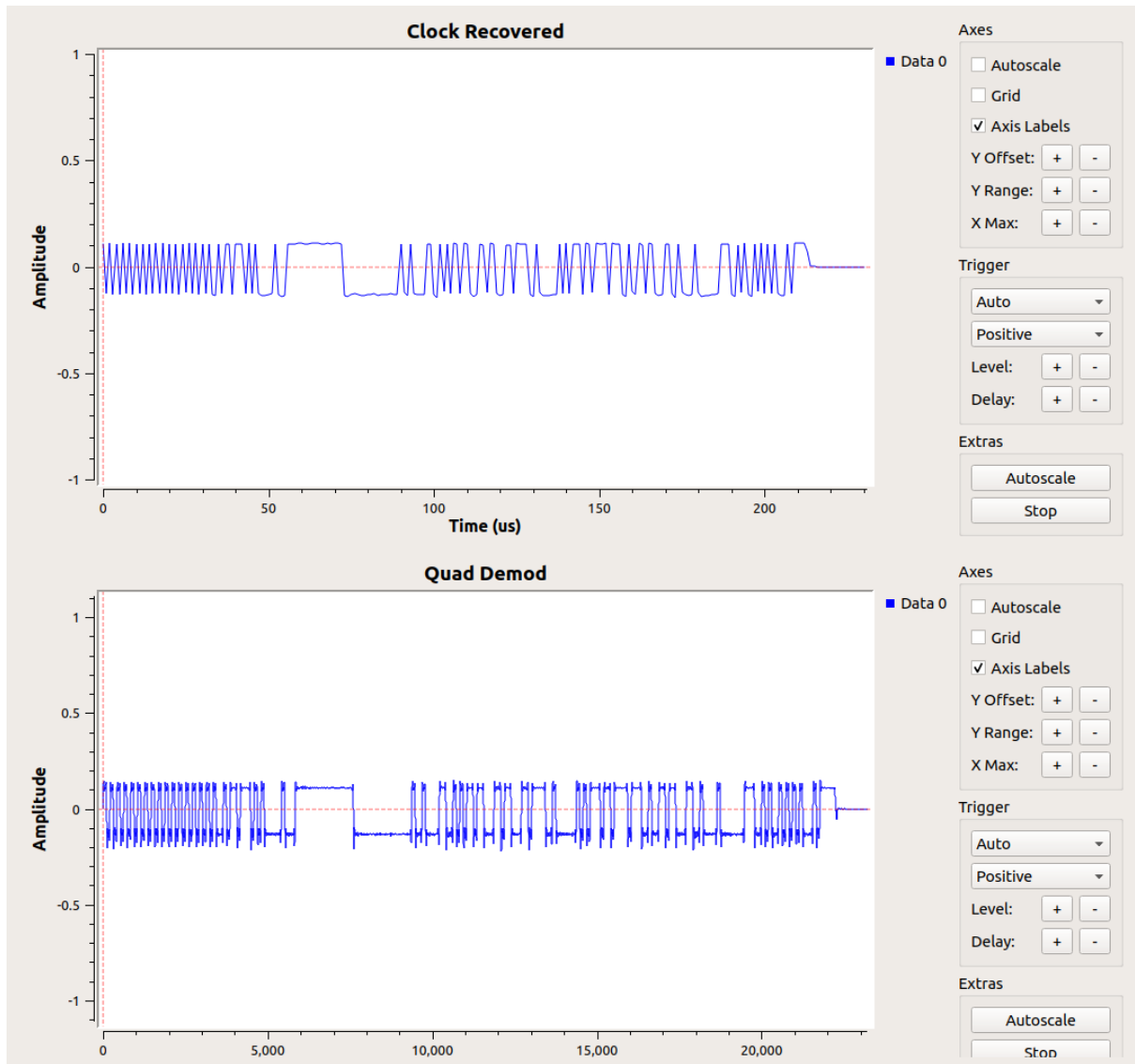
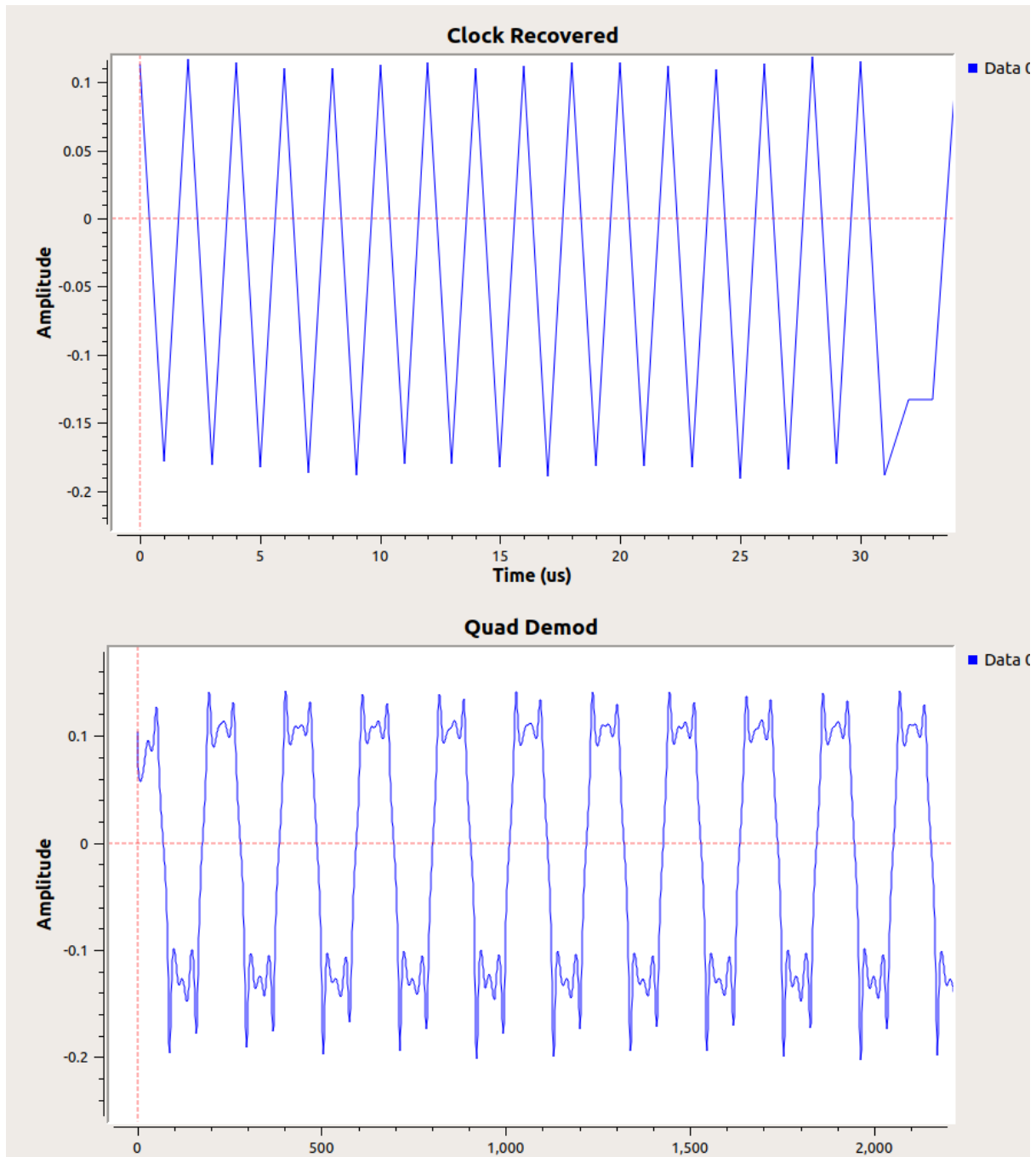Figure 21: The signal before and after clock recovery

Figure 22: Detail of a series of 1 and 0 before and after clock recovery

5.5 Find start

5.6 Decode

6 Real World Examples and Further Projects

6.1 Tyre Pressure Monitoring System

This technique can be used in the 'real world' as demonstrated here: `https://www.sharebrained.com/downloads/toorcon/dude_wheres_my_car_toorcon_sd_2013.pdf` where someone has used the same technology to find out the tyre pressure of different cars near them. Work through the slides and the code provided and see what you can see.

6.2 FM Reciever

You can listen to the radio, and watch the spectrum and signals change as you do it by following the guide here: `https://www.instructables.com/id/RTL-SDR-FM-radio-receiver-with-GNU-Radio-Companion/`

6.3 Satellite Images

There is a rough guide for receiving images from weather satellites available which may require some special configuration of antenna, if you're up for it!

`http://oz9aec.net/radios/gnu-radio/noaa-weather-satellite-reception-with-gnu-radio-and-usrp/`