RF Workshop

Patrick Mintram

August 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.

Text in red boxes are things you need to type in.

Text in blue boxes are instructions, but you might not have to type them in.

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

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 raise an issue or create a pull request.

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 repository.
   (d) gnuradio[3].
   (e) A clone of this repository 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].

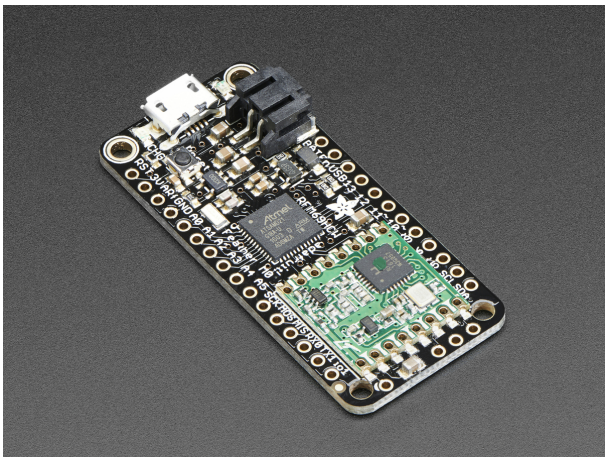

Figure 1: An Adafruit Feather M0 with RFM69 Packet Radio

2.1 SDRs

An SDR is a really cool way of providing a configurable radio. Whereas a radio implemented in a circuit would have lots of fixed value components such as resistors and capacitors all tuned to specific frequencies, such as that in fig. 2. The downside of the is that it's not reconfigurable, and that's something important in cyber security. A typical SDR architecture, as shown in fig. 3 is able to be changed using digitally defined filtering and hardware which is configurable on the fly. We are going to configure an SDR to look at our secret message.

[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`

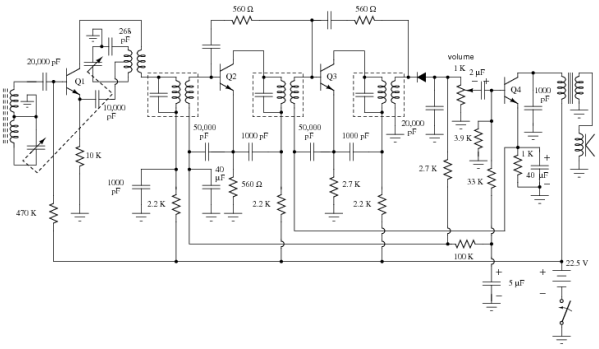

Figure 2: A hardware implemented radio circuit

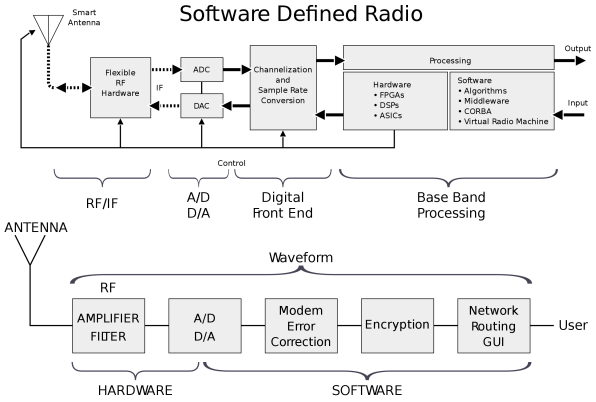

Figure 3: A generic SDR block diagram.

2.2 gnuradio

This is a piece of software which provides a drag and drop graphical user interface (GUI) for performing our SDR reconfiguration. We can also process the data after it's been received using special blocks.

2.3 RadioHead-Extras repo

This is a well known repository for programatically configuring an RF transceiver chip, such as the one on the Feather module we will be using. This library is specific to the project as I have added some functionality to help us to send data without lots of extra features which will complicate things. t

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. 4[8].

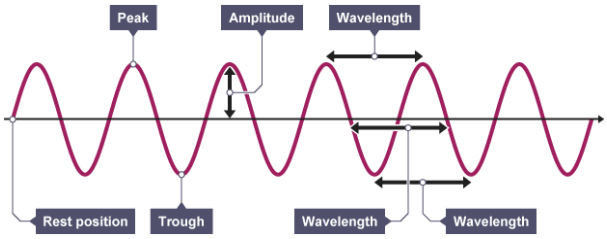

Figure 4: The RF spectrum

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. 5[9].

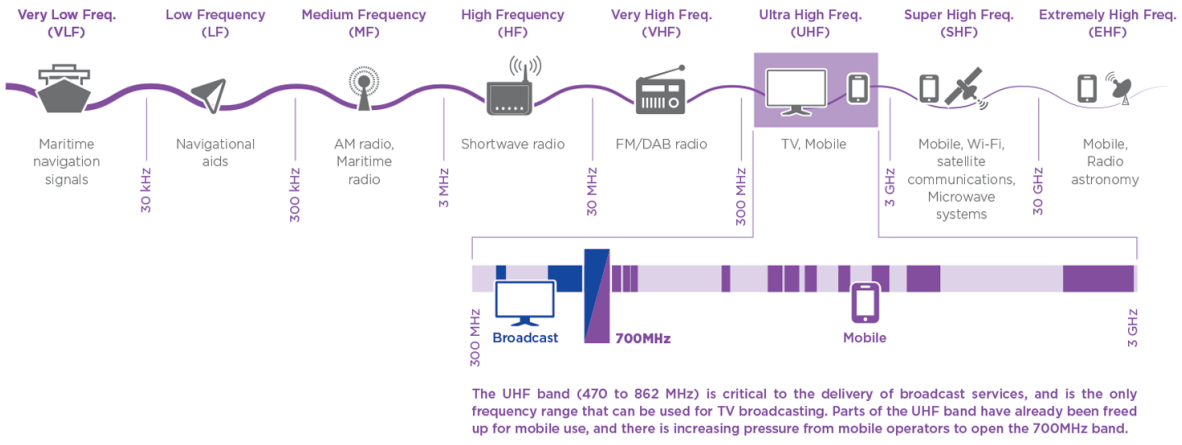

Figure 5: The RF spectrum

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

[8] https://www.bbc.com/bitesize/guides/zgf97p3/revision/1
[9] https://www.ecnmag.com/blog/2017/06/understanding-rf-spectrum

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. 6.

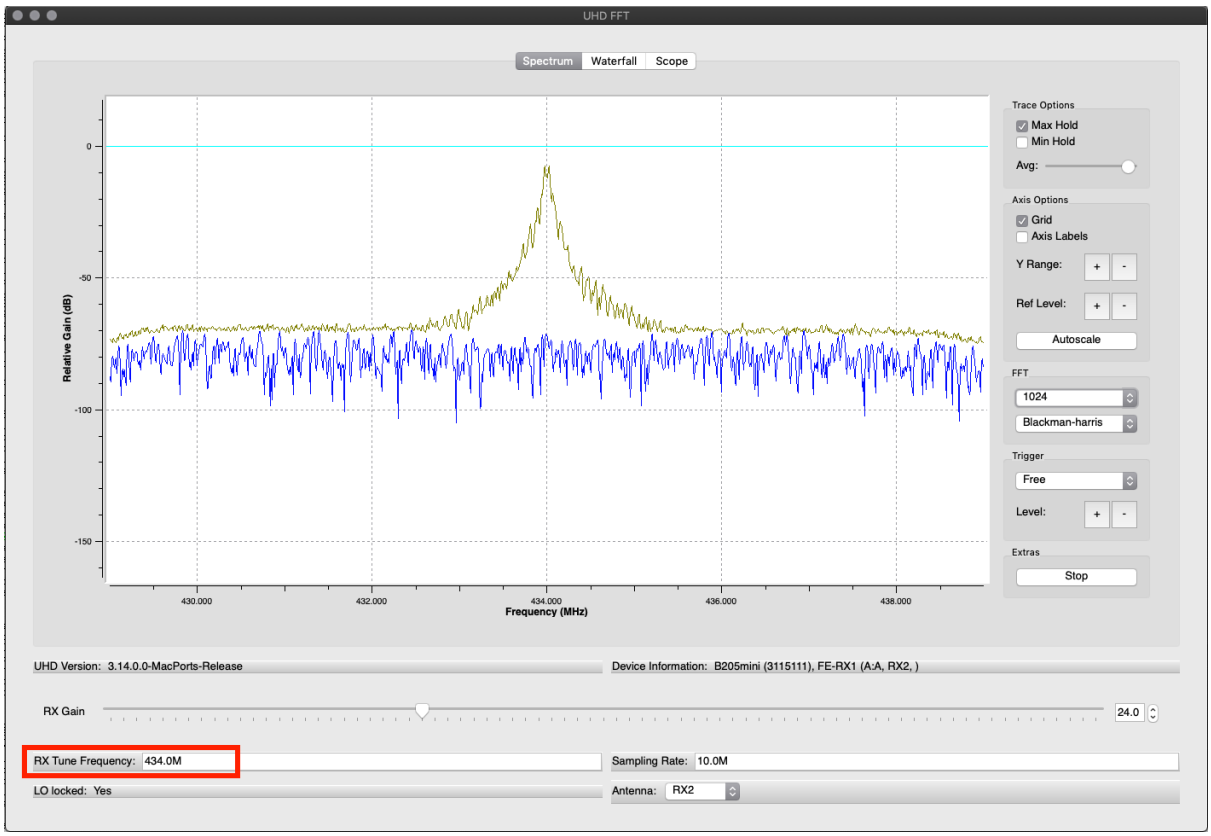

Figure 6: The output of the uhd_fft command with a 434MHz signal present.

The spike on the yellow line in fig. 6 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` is a provided command which opens a spectrum analyser like the one in fig. 6

```
uhd_fft -f 434000000 -s 10000000
```

Try looking somewhere around 2.4GHz and seeing how busy it is, by changing the rx tune frequency highlighted in fig. 6. Alternatively you can run the command again with different values, the number after the `-f` is the centre frequency, and the number after the `-s` is the sample rate, or how wide the spectrum is.

3.1 gnuradio flows

Rather than using a prepackaged command like `uht_fft` we can make our own GUIs 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

Open up the provided file FFT.grc and click the play button highlighted.

You should 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.

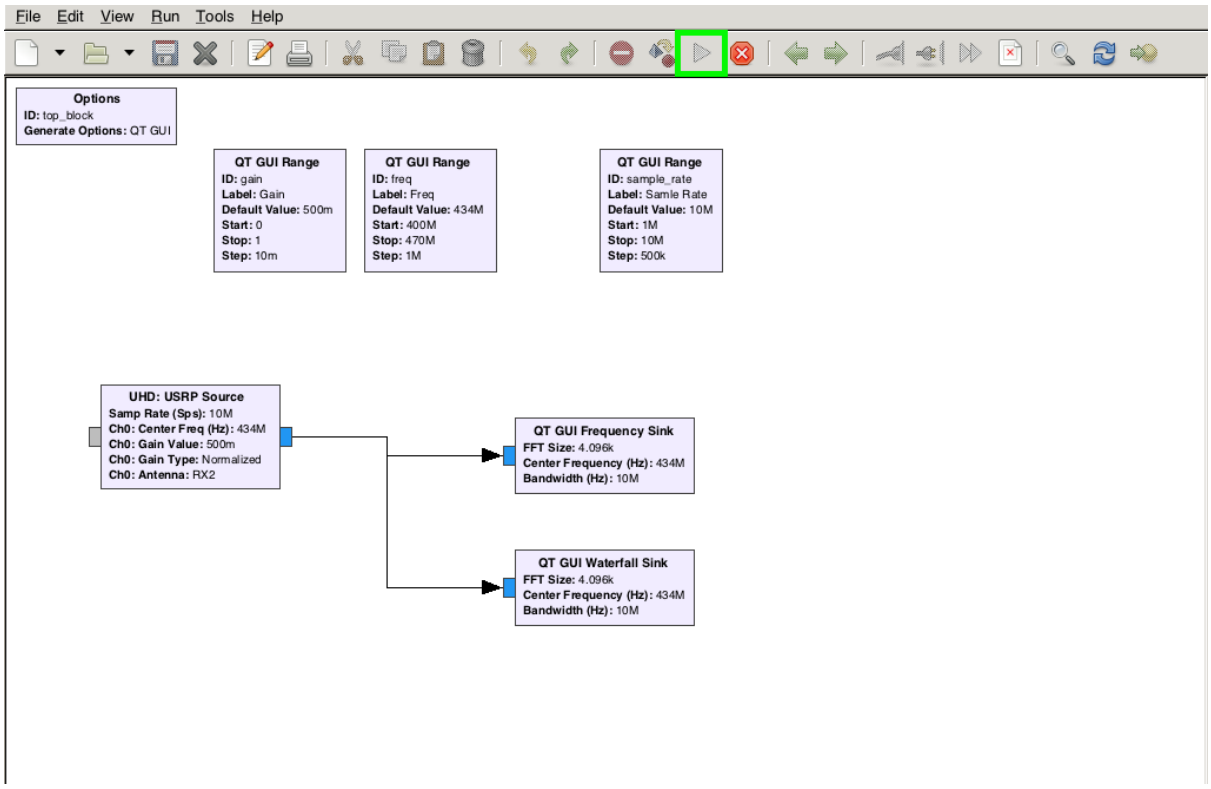

Figure 7: gnuradio flow for viewing the spectrum

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. For further information on Arduino libraries consult the Arduino website[10]

Open up the provided 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! Try changing the code to send another message but for now keep the settings of the RF chip the same, we will come back and change them later.

[10] `https://www.arduino.cc/en/Main/Libraries`

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. 9.

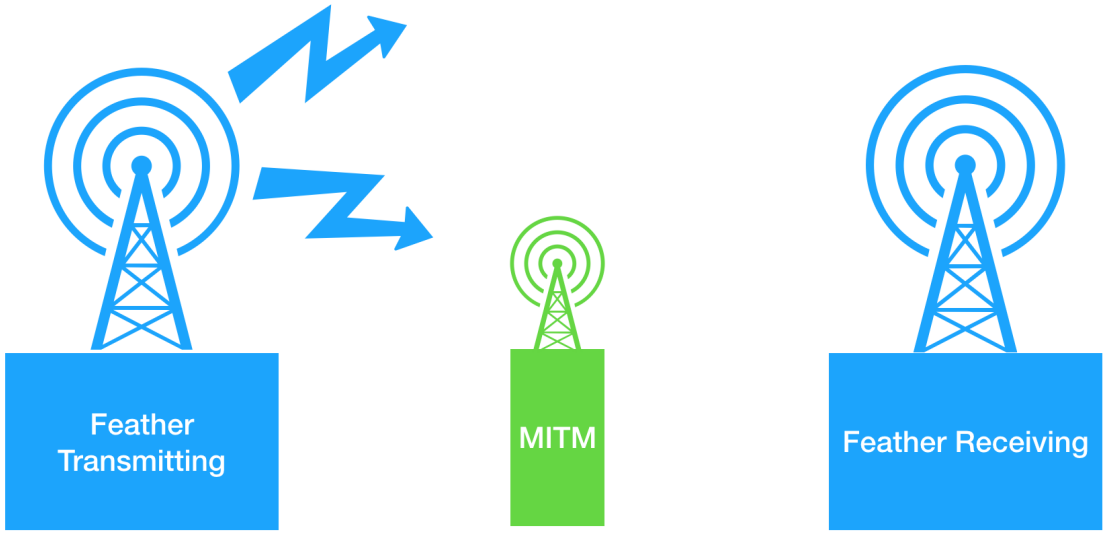

Figure 8: MITM attack on our two feathers

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

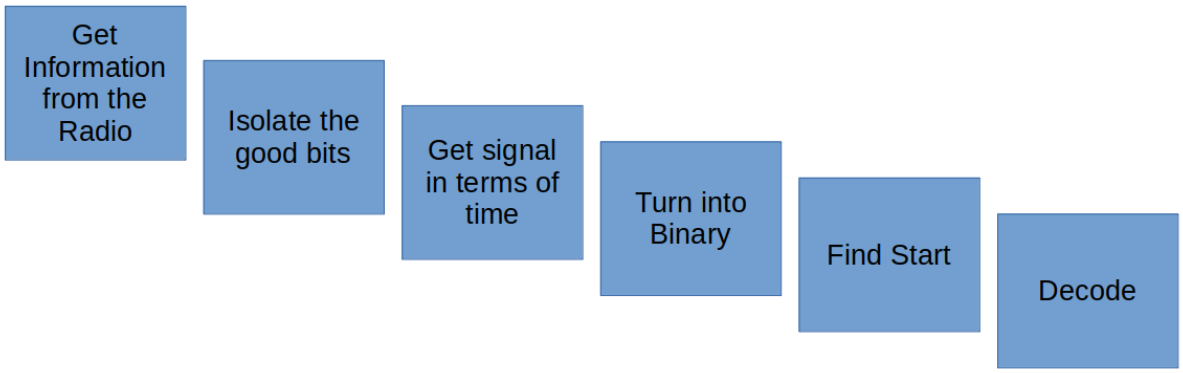

Figure 9: The steps to looking at the data sent

5.1 Connecting the Radio

In order to get data into the flow from the SDR we must use a source (just like a river, in gnuradio all data has a source). In this case, it's the RTL-SDR source because we are using the RTL-SDR dongle. If there are issues there are plenty of resources available online to work through the RTL-SDR specifics[11].

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

If you don't have an RTL-SDR the manufacturers website will have documentation of how to use the SDR in gnuradio.

Find the source in gnuradio using the search button shown in fig. 10.

Figure 10: gnuradio-companion search icon

Drag in the source to the canvas, the canvas should look like the one in fig. 11.

Figure 11: A gnuradio flow with just a radio source

Double click the source block.

Here you will get some options, and 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 gives us a slider on the GUI to change settings at runtime.

Add a QT GUI Range block.

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 as shown in fig. 12.

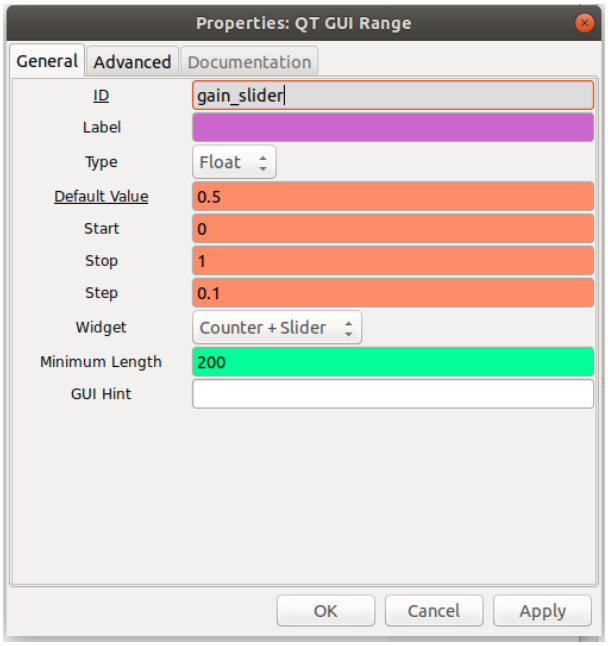

Figure 12: Setting up a slider

Now you can enter it's unique ID in the radio's settings under *gain* as shown in fig. 13. This is effectively telling the radio to get it's gain value from the slider called `gain_slider`.

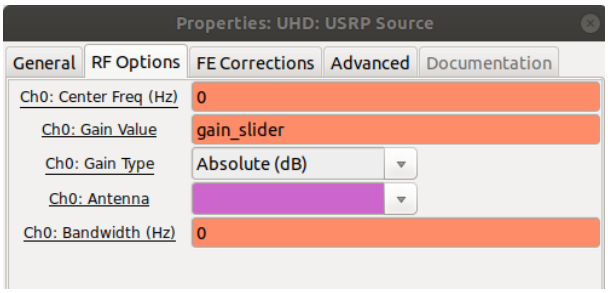

Figure 13: Using a slider value for the gain in the radio source.

Create a slider for the *centre frequency* and sample rate.

Set up your new sliders with the following settings, noting how the sample rate is a lot lower than the centre frequency:

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* a.k.a. FFT sink is required.

Add a GT GUI Frequency Sink and connect it shown in fig. 14.

Drag this in and connect it to the radio by clicking the blue tabs.

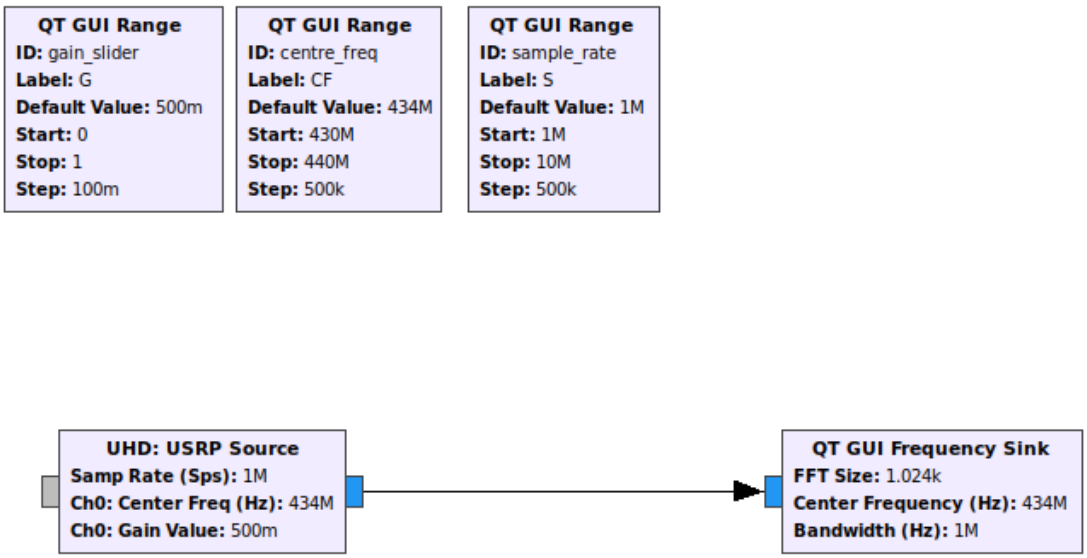

Figure 14: A basic flow.

Again, open the FFT sink block and set the centre frequency to the same slider as your radio.

Set the bandwidth to your sample rate slider as shown in fig. 15.

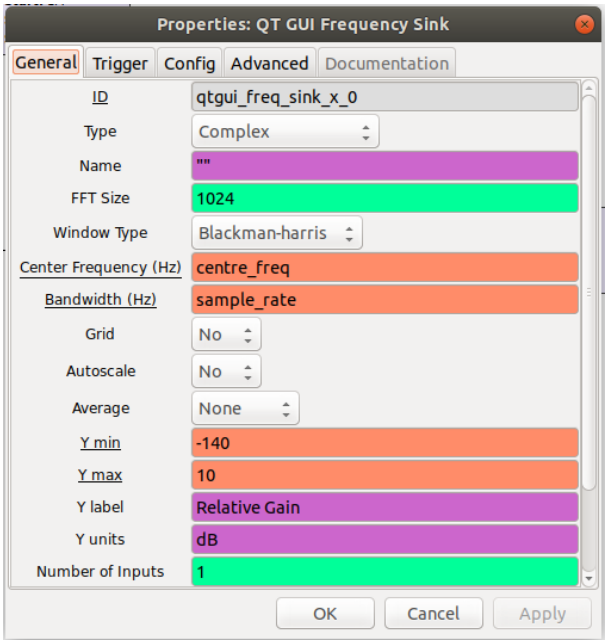

Figure 15: FFT Sink Configuration.

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 pops 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 a low pass filter (LPF) to make sure we receive only the parts of the transmission we are interested in.

Drag an LPF into the gnuradio flow, and add another FFT sink to see the output.

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).

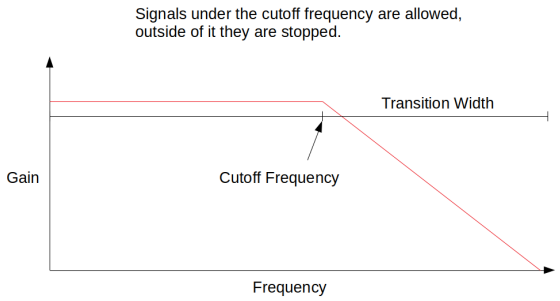

Figure 16: Where the cutoff frequency and transition width matter.

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

Figure 17: Sauron's home.

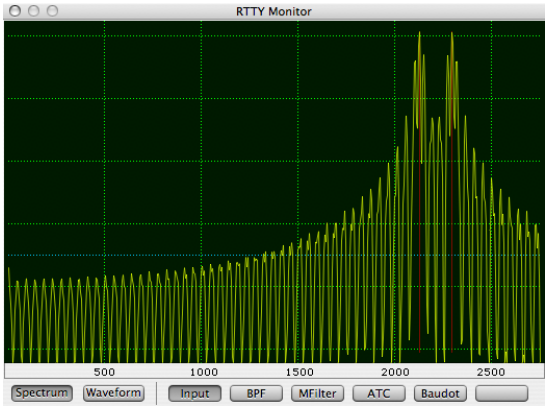

Figure 18: FSK Signal as seen on a frequency plot.

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 experimentation. 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.

Add a simple squelch 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.

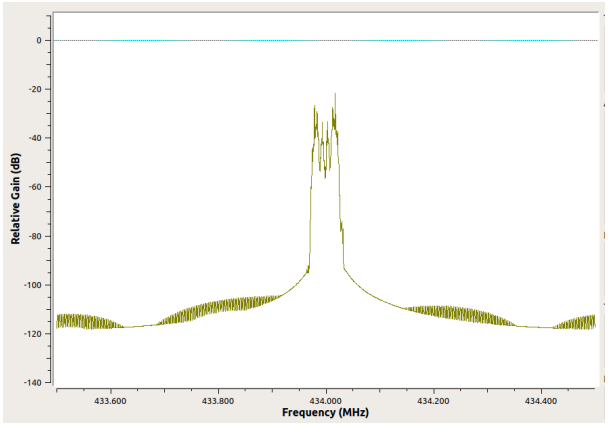

Figure 19: An ideal output from the LPF

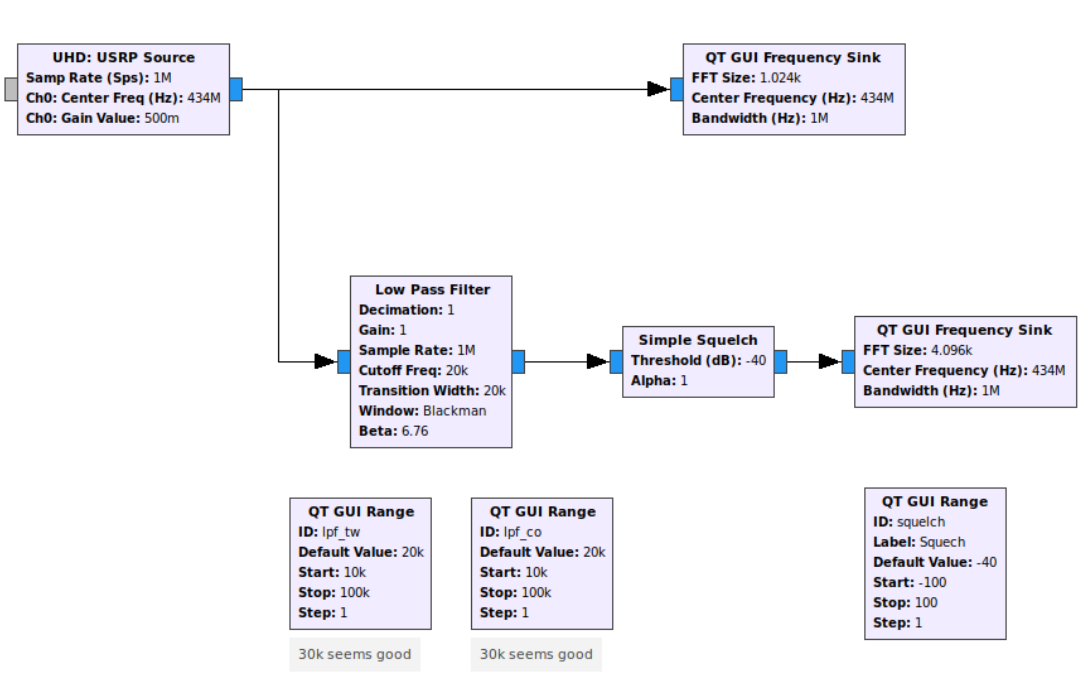

Figure 20: Radio - LPF - Squelch - Frequency Sink

5.3 Get signal in terms of time

Now we have isolated the peaks, we can turn these into bits by using a Quadrature demod block with a gain setting of 1.

Add a quadrature demod block.

Plug the output from this into a *QT GUI Time Sink*, and enable the time sink's control panel the same way you did with the FFT sink. You may have to change the `type` of the the time sink in it's config panel so that the blocks join up correctly.

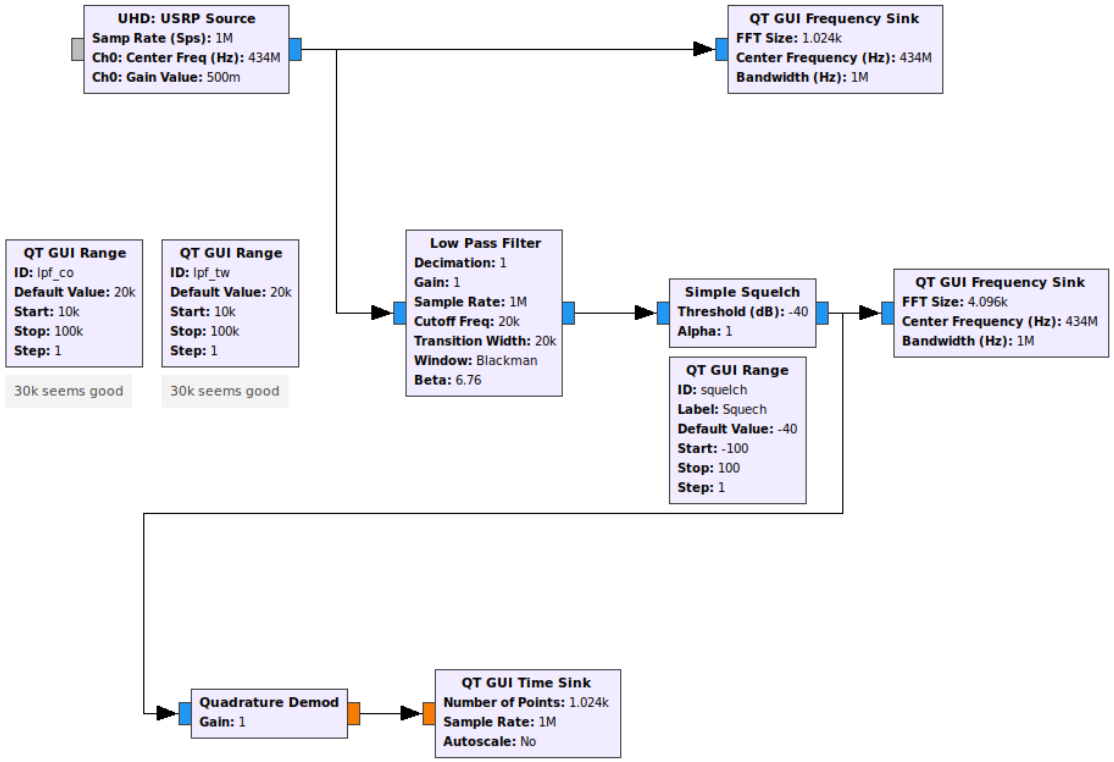

Figure 21: Radio - LPF - Squelch - Quad Demod - Time Sink

This new graph will have turned the frequency changes into changes against time. This will be easier to see in the time sink if you set the trigger to auto shown in fig. 22, and when the flow is running click '+ X Max' loads until you can see what looks like something interesting like that in fig. 23.

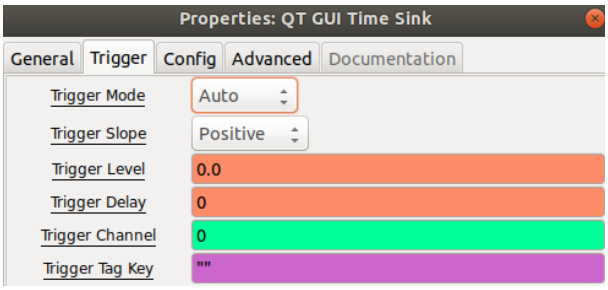

Figure 22: Setting the timesink trigger to auto in the GUI

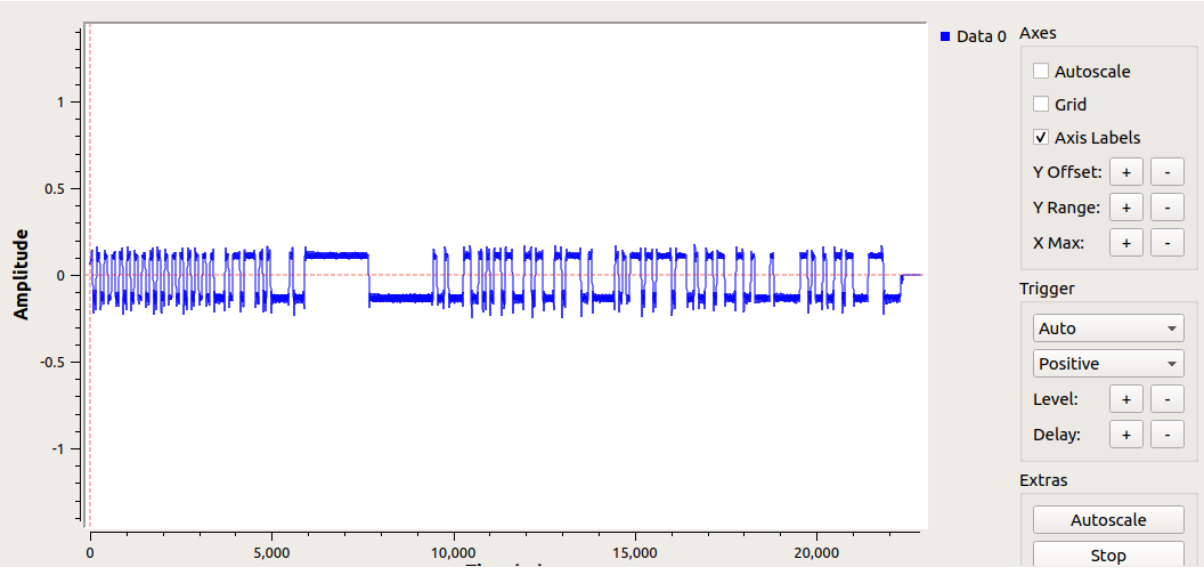

Figure 23: Our signal against time

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 MM 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 relatively 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.

Fill the variables with the information in table 2.

Variable Name	Value
baudrate_hz	9600
deviation_hz	19200
samples_per_symbol	sample_rate/baudrate_hz

Table 2: Variable values

Add a Clock recovery MM block.

Then, in your clock recovery block set the *omega* to be `samples_per_symbol` like in fig. 24

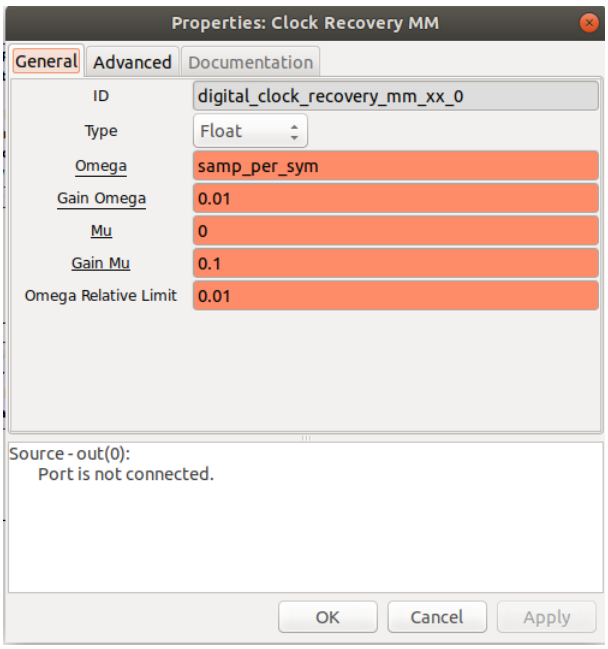

Figure 24: Clock recovery block settings

Connect a time sink to this to see the output.

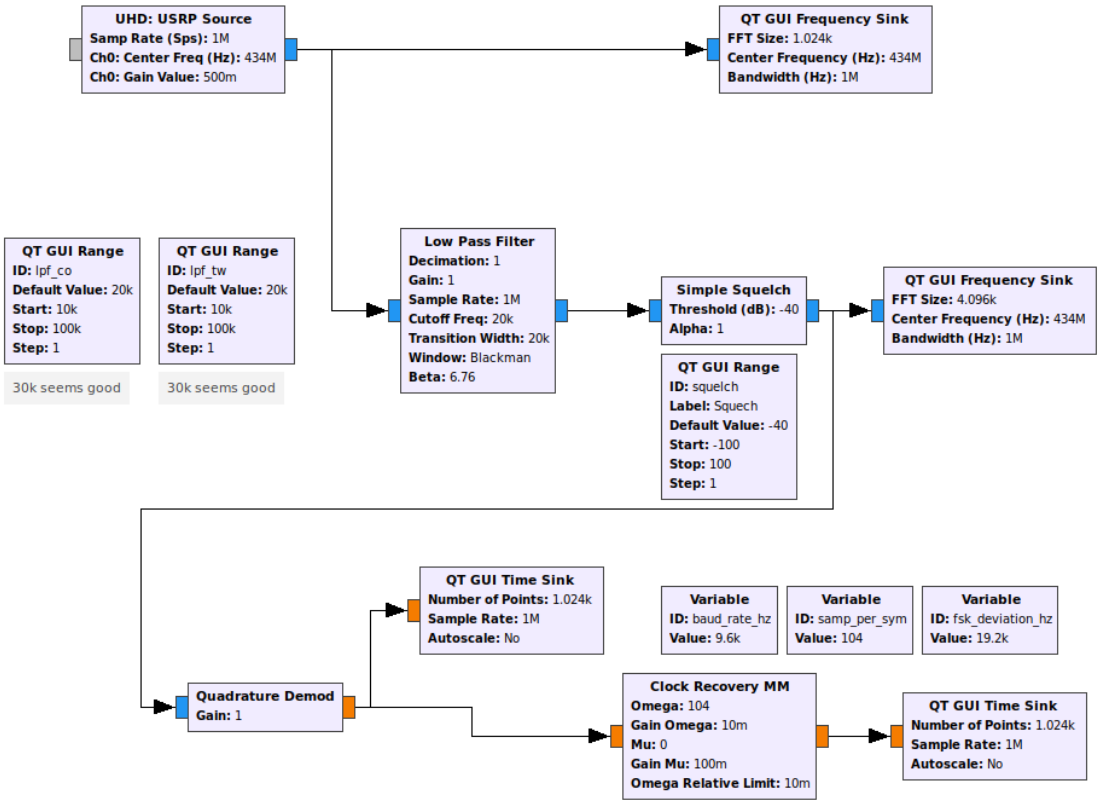

Figure 25: Radio - Filter - Squelch - Quad Demod - Clock Recovery

What you should see is a slight difference between the time sinks for just the quadrature demod and the clock recovery MM. The clock recovery MM 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 MM this part of this signal has "fatter" highs and lows. A comparison can be seen in fig. 26 and fig. 27.

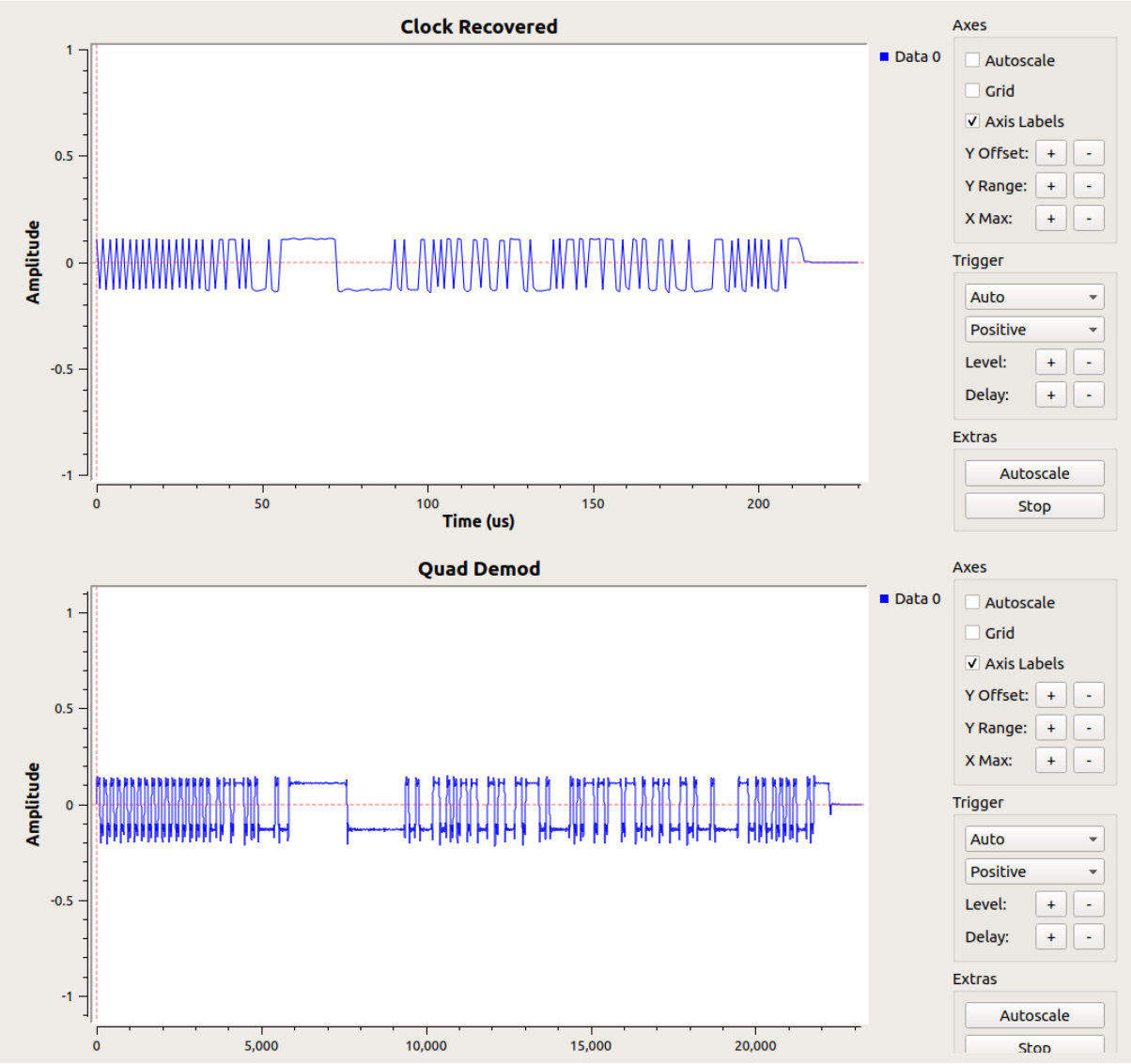

Figure 26: The signal before and after clock recovery

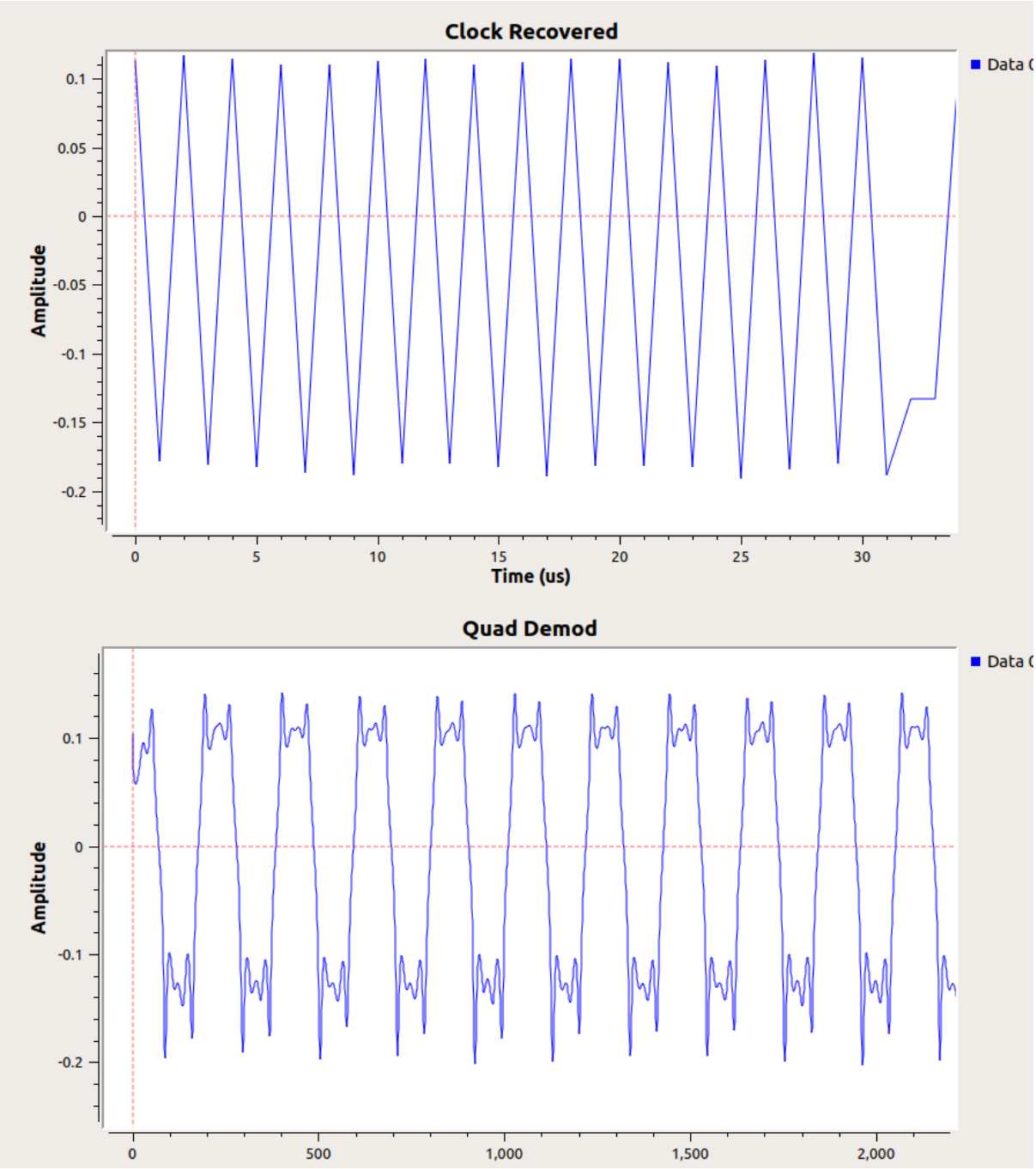

Figure 27: Detail of a series of 1 and 0 before and after clock recovery

Finally add a *binary slicer* to the end of the clock recovery block.

Done.

5.5 Find start

Finding the start of the signal is relatively easy as a block called Correlate access code - tag will do this by searching for a pattern in the binary stream coming in. Through inspection of the time sinks before we can see that the signal starts with a long stream of 1 and 0. This is known as the preamble and is present in lots of signals; it's the radio's way of saying "LISTEN TO ME I'M TRANSMITTING" before it sends any useful data. In your new block set the `Access Code` to be 10101010101010101010101010101000101101 (or 4 bytes of 10101010 then a byte of 00101101), a threshold of 0, and call the tag `preamble`.

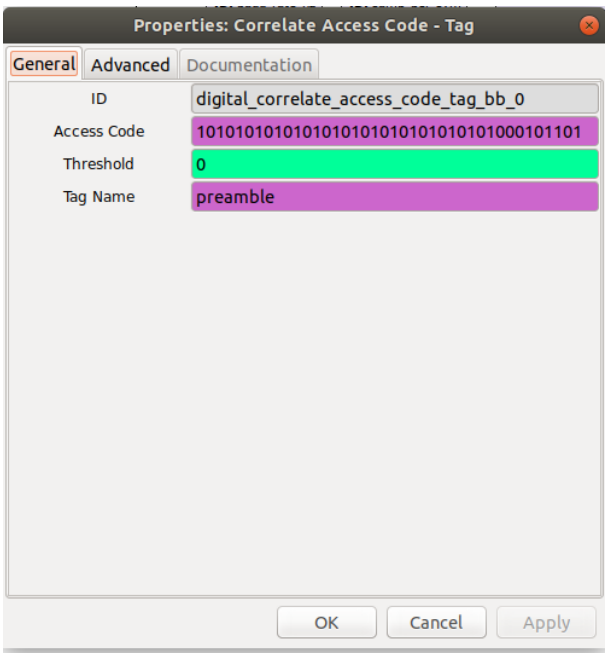

Figure 28: Correlate Access Code Settings

This block, when it sees a stream of data matching the access code will insert a special tag into the data stream with the name "preamble" to show that it's detected a preamble. Finally output this to a file using a *File Sink*. This file may grow quite large so I recommend making something like `tmp/file_sink` so that it's deleted when you turn off your computer.

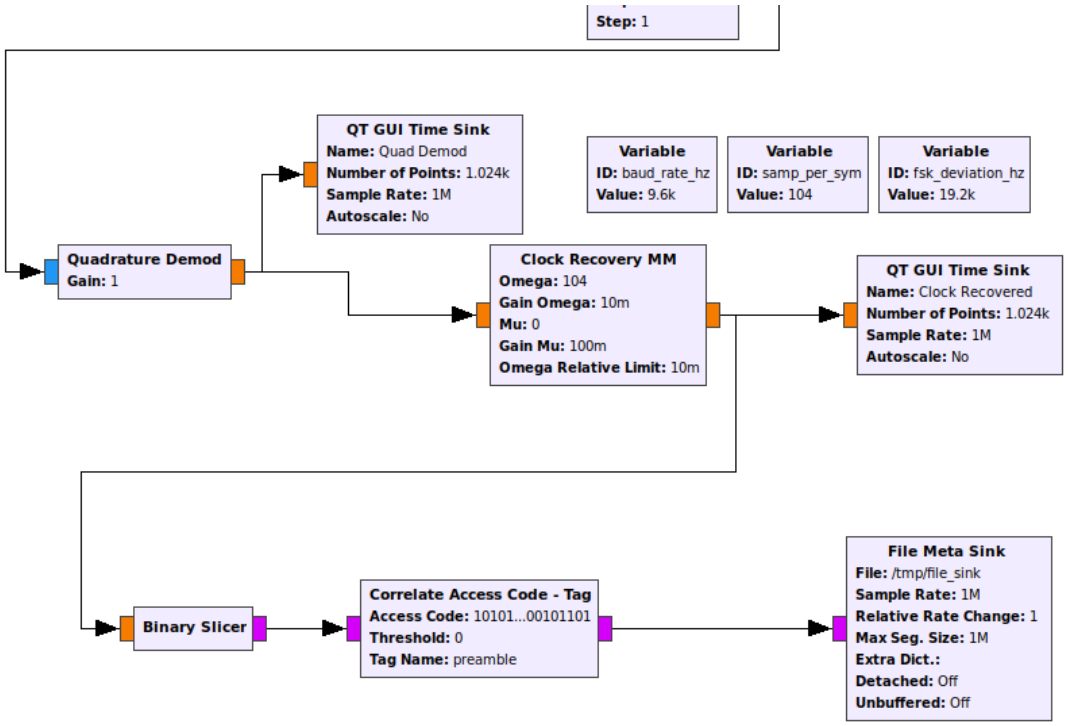

Figure 29: Radio - LPF - Squelch - Quad Demod - Clock Recovery - Binary Slicer - Correlate Access Code - File Sink

A full flow is available to do this in the repository - if you get stuck at any point.

5.6 Decode

Now come the hard part - the decoding. This can be difficult because in the real world you might not know what you're looking for in the file. It is also made trickier by the way that gnuradio companion (GRC) handles the demodulated bit stream. The main thing to know is that each bit is represented by a byte in the output.

Understanding what's in the file might take a bit, so there is a small tool to help demonstrate that a `1` bit will look like a `01` byte in the file, and a `0` bit will look like a `00` byte in the file. The least significant bit (LSB) is the most important one! There is a tool to help demonstrate this process in the `src\tools` directory of this repository. Run it to see how GRC turns the binary `00110101` into bytes and put it in a file called "out".

```
python make_binbyte_file.py 00110101 out
```

Take some time to play with this and understand what's happening here. You can see the output

using the `xxd` tool or some other binary file viewer, a good one is 010editor if you have it. A top tip for this is to ignore the first digit printed, and then every other one after that; so "0001000001" turns into "010001".

xxd out

The other hard bit is unfortunately quite manual in the real world - it's basically looking through the file created and looking for patterns, or things that are out of the ordinary. Fortunately you know the secret message which was sent, so you know to look for a pattern of `1`s and `0`s which spells out the secret message! At this point you will need to compare the bits to the American Standard Code for Information Interchange (ASCII) for them, for example an "a" in hex is `61`, which is `01100001` in binary. If you need a reminder of the ASCII table there are loads of great comparison tables on google. So if you see the pattern `0001010000000001` in the file created, this means an "a" has been demodulated in the RF signal sent.

Use xxd (or your favourite tool) to inspect the output from the GRC flow and find the secret message in the data. This might be difficult to spot, so take plenty of breaks to prevent getting *binary blindness*[a].

[a]Not a real medical condition, just where it gets difficult to see the patterns in loads of 0s and 1s on the screen

5.7 Further things to do with your flow

And that's it! You have now successfully decoded a secret message sent over the air using your SDR. Here's some extra things you can do in pairs:

1. Change the secret message sent and find that in the output file.
2. Look at the messages sent - what is the extra stuff that the adafruit library is sending in addition to the secret message?
3. Change the frequency the secret message is sent on and try to find it, then decode it!

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 Receiver

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/`

Glossary

ASCII American Standard Code for Information Interchange. 29

baud rate how many bits are sent per second. 22

binary slicer a gnuradio block that rounds values to ones and zeros. 22

clock recovery MM a gnuradio block that synchronises the edge of a binary signal to a clock. 22–24

correlate access code - tag a gnuradio block that finds a code in the bit stream. 27

cutoff frequency that max (or min) frequency a filter will let through. 1, 18

deviation how far from the centre frequency a peak it. 22

FFT sink a gnuradio block which shows the signal as amplitude against frequency. 15, 16, 18, 20

GRC gnuradio companion. 28, 29

GUI graphical user interface. 6, 9, 14

IDE Integrated Development Environment. 5

LPF low pass filter. 17, 18

LSB least significant bit. 28

MITM man in the middle. 3, 12

preamble "LISTEN TO ME I'M TRANSMITTING". 27

quadrature demod a gnuradio block that turns frequency changes into amplitude changes. 20, 24

repository a storage location from which software packages may be retrieved and installed on a computer. 5, 28

RF Radio Frequencies. 3, 6, 7, 11, 22, 29

sample rate how quickly the analog value is turned into a digital one and passed to the gnuradio flow. 14–16

SDR Software Defined Radio. 5, 6, 8, 12, 13, 29

sink a gnuradio block that lets data leave the flow. 27

source a gnuradio block which brings data into the flow. 12–14

squelch a gnuradio block that stops any frequency below a set threshold. 19

time sink a gnuradio block which shows the signal as amplitude against time. 20, 21, 23, 24, 27

transition width how slopey the edge of a filter are. 1, 18

UHF ultra high frequency. 7

VHF very high frequency. 8