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].

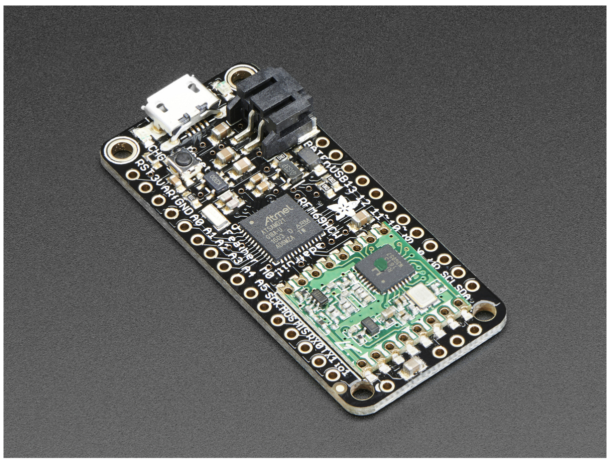


Figure 1: An Adafruit Feather M0 with RFM69 Packet Radio

[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 electro-magnetic 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].

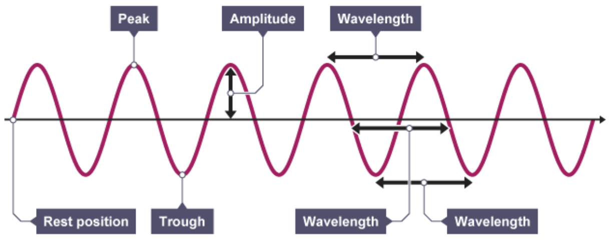
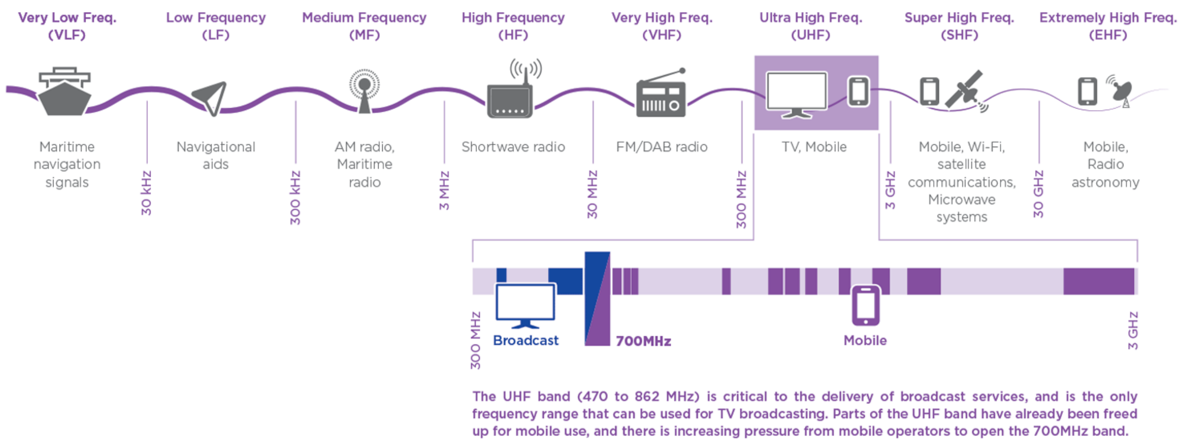


Figure 2: The RF spectrum

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].



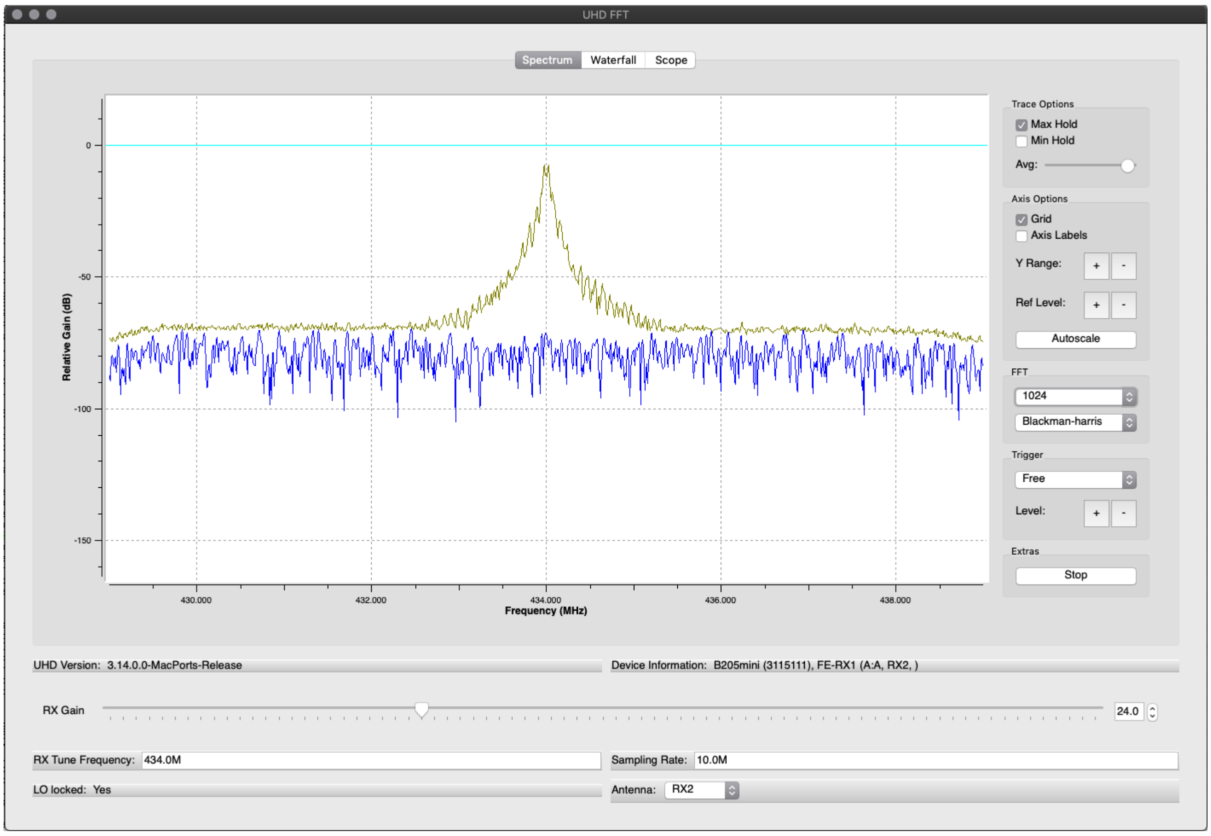
The UHF band (470 to 862 MHz) is critical to the delivery of broadcast services, and is the only frequency range that can be used for TV broadcasting. Parts of the UHF band have already been freed up for mobile use, and there is increasing pressure from mobile operators to open the 700MHz band.

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

[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. 4.



Trace Options: [✓ Max Hold] Min Hold

Axis Options: [✓ Grid] Axis Labels

FFT: 1024

Blackman-harris

Trigger: Free

UHD Version: 3.14.0.0-MacPorts-Release

Device Information: B205mini (3115111), FE-RX1 (A:A, RX2,)

RX Gain 24.0

RX Tune Frequency: 434.0M

Sampling Rate: 10.0M

LO locked: Yes

Antenna: RX2

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
```

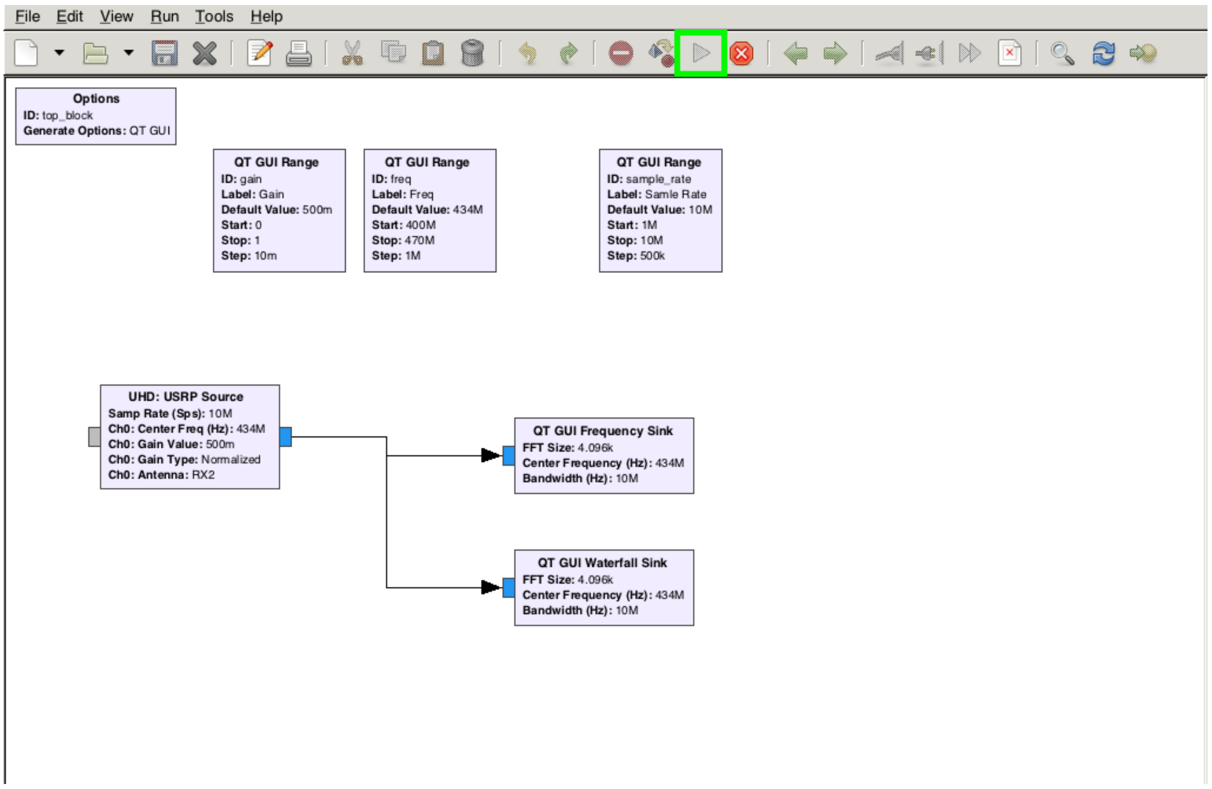
Try looking somewhere around 2.4GHz and seeing how busy it is. What do you think these signal might be?

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.



Options
ID: top_block
Generate Options: QT GUI

QT GUI Range
ID: gain
Label: Gain
Default Value: 500m
Start: 0
Stop: 1
Step: 10m

QT GUI Range
ID: freq
Label: Freq
Default Value: 434M
Start: 400M
Stop: 470M
Step: 1M

QT GUI Range
ID: sample_rate
Label: Samle Rate
Default Value: 10M
Start: 1M
Stop: 10M
Step: 500k

UHD: USRP Source
Samp Rate (Sps): 10M
Ch0: Center Freq (Hz): 434M
Ch0: Gain Value: 500m
Ch0: Gain Type: Normalized
Ch0: Antenna: RX2

QT GUI Frequency Sink
FFT Size: 4.096k
Center Frequency (Hz): 434M
Bandwidth (Hz): 10M

QT GUI Waterfall Sink
FFT Size: 4.096k
Center Frequency (Hz): 434M
Bandwidth (Hz): 10M

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.

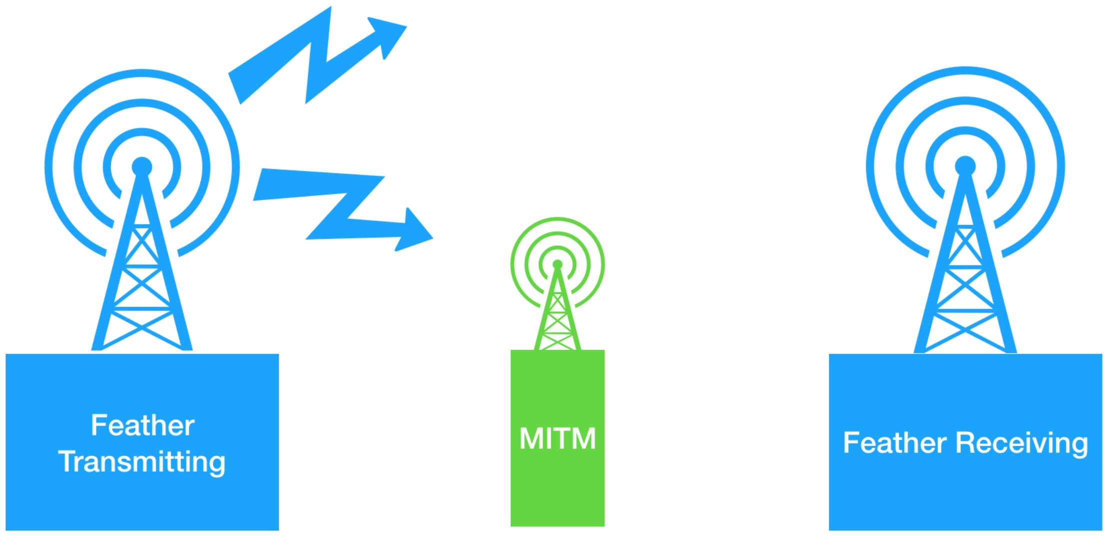


Figure 6: MITM attack on our two feathers

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

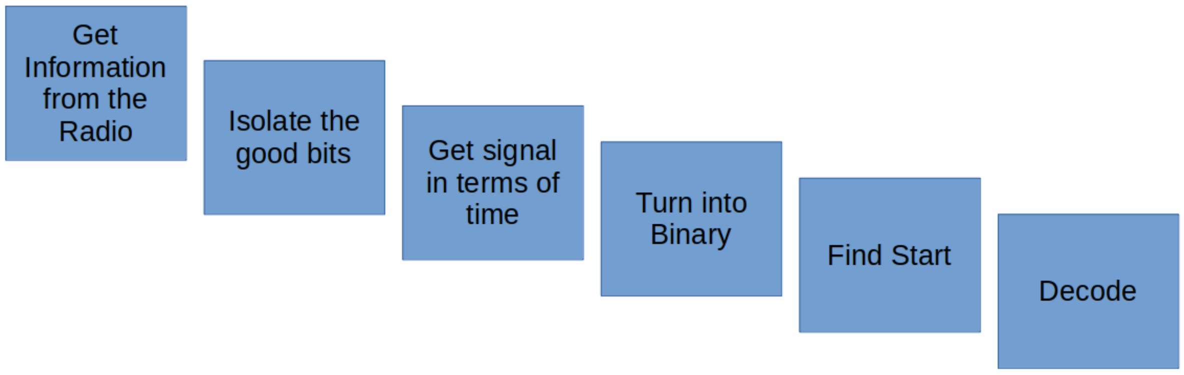


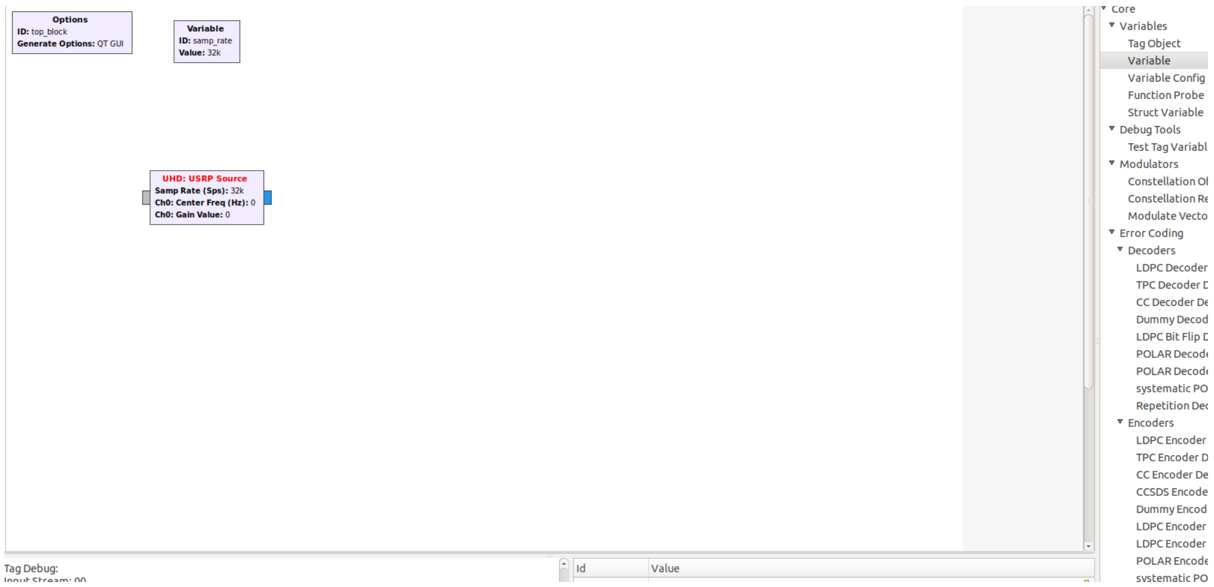
Figure 7: The steps to looking at the data sent

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.

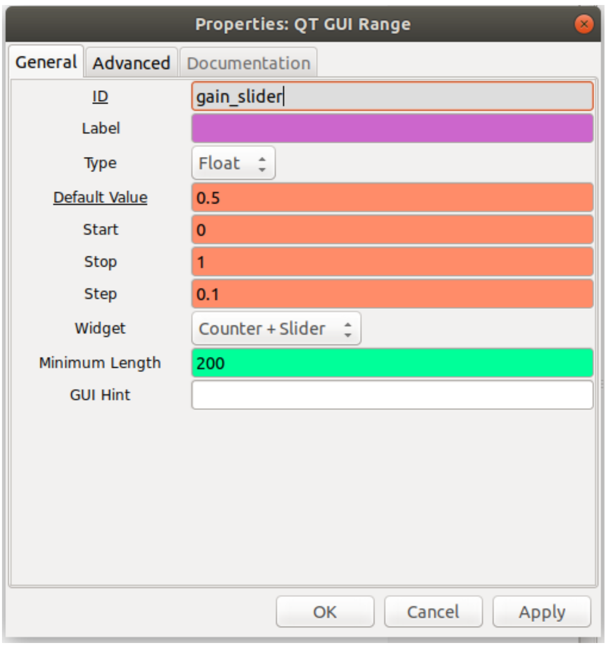

Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

UHD: USRP Source
Samp Rate (Sps): 32k
Ch0: Center Freq (Hz): 0
Ch0: Gain Value: 0

Figure 8: A gnuradio flow with just a radio source

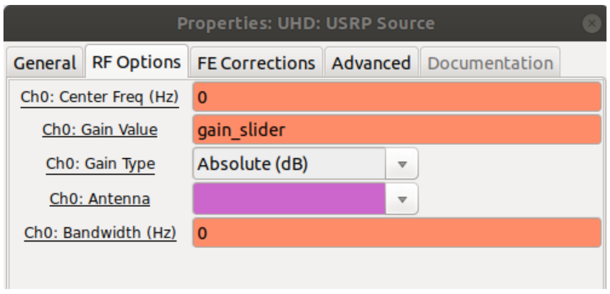
If you double click the block you will get some options, in here the key things to find are the *sample rate*, *gain*, and *centre frequency*. The sample rate affects how quickly the radio is taking measurements (the rate at which it samples!) and as a result how wide the spectrum we are monitoring is. 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.


Properties: QT GUI Range

ID	gain_slider
Label	
Type	Float
Default Value	0.5
Start	0
Stop	1
Step	0.1
Widget	Counter + Slider
Minimum Length	200
GUI Hint	

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*.


Properties: UHD: USRP Source

Ch0: Center Freq (Hz)	0
Ch0: Gain Value	gain_slider
Ch0: Gain Type	Absolute (dB)
Ch0: Antenna	
Ch0: Bandwidth (Hz)	0

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.

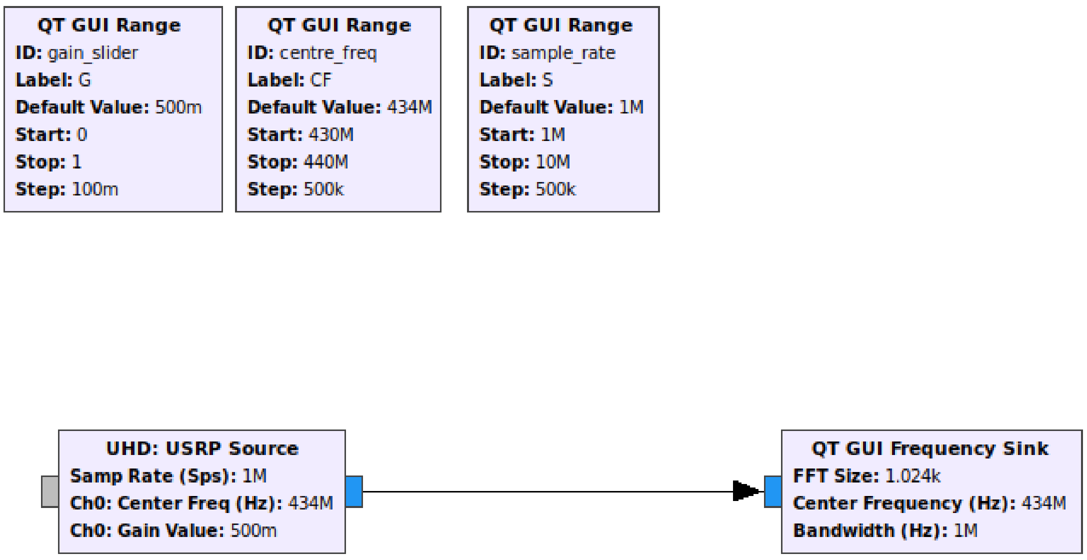
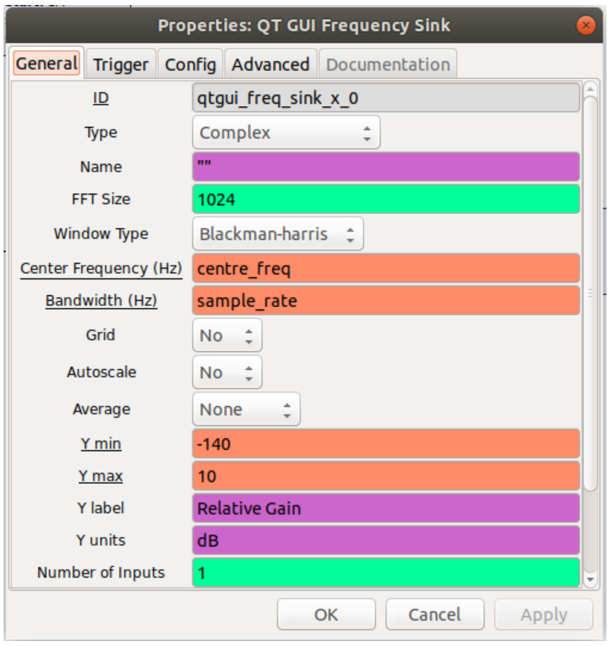


Figure 11: A basic flow.

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:



ID	qtgui_freq_sink_x_0
Type	Complex
Name	""
FFT Size	1024
Window Type	Blackman-harris
Center Frequency (Hz)	centre_freq
Bandwidth (Hz)	sample_rate
Grid	No
Autoscale	No
Average	None
Y min	-140
Y max	10
Y label	Relative Gain
Y units	dB
Number of Inputs	1

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 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 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).

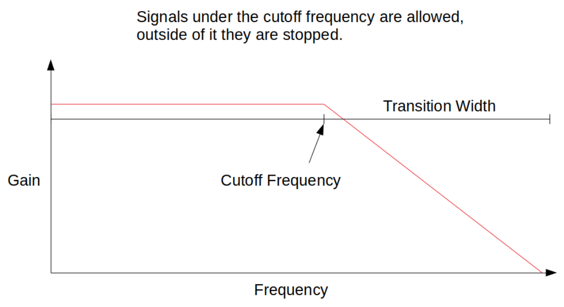

Signals under the cutoff frequency are allowed, outside of it they are stopped.
Transition Width
Gain
Cutoff Frequency
Frequency

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.

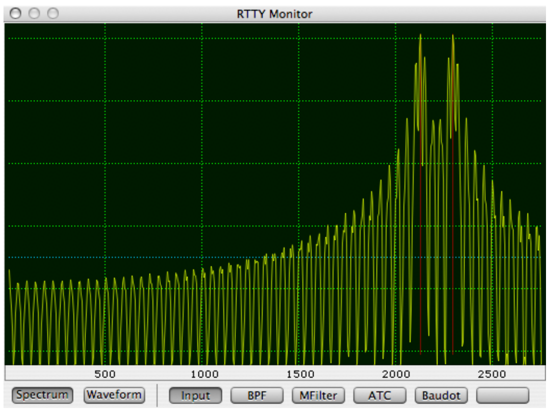


Figure 15: FSK Signal as seen on a frequency plot.

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.

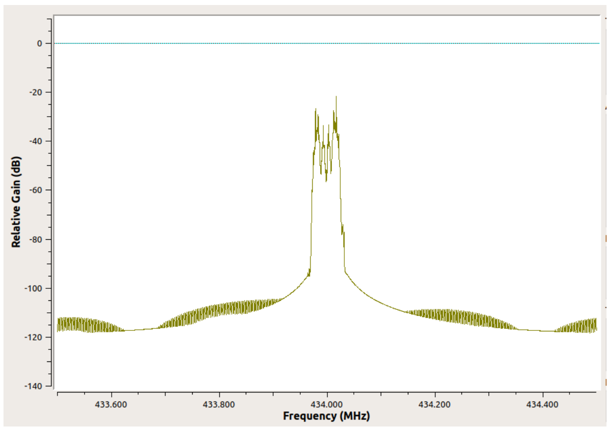
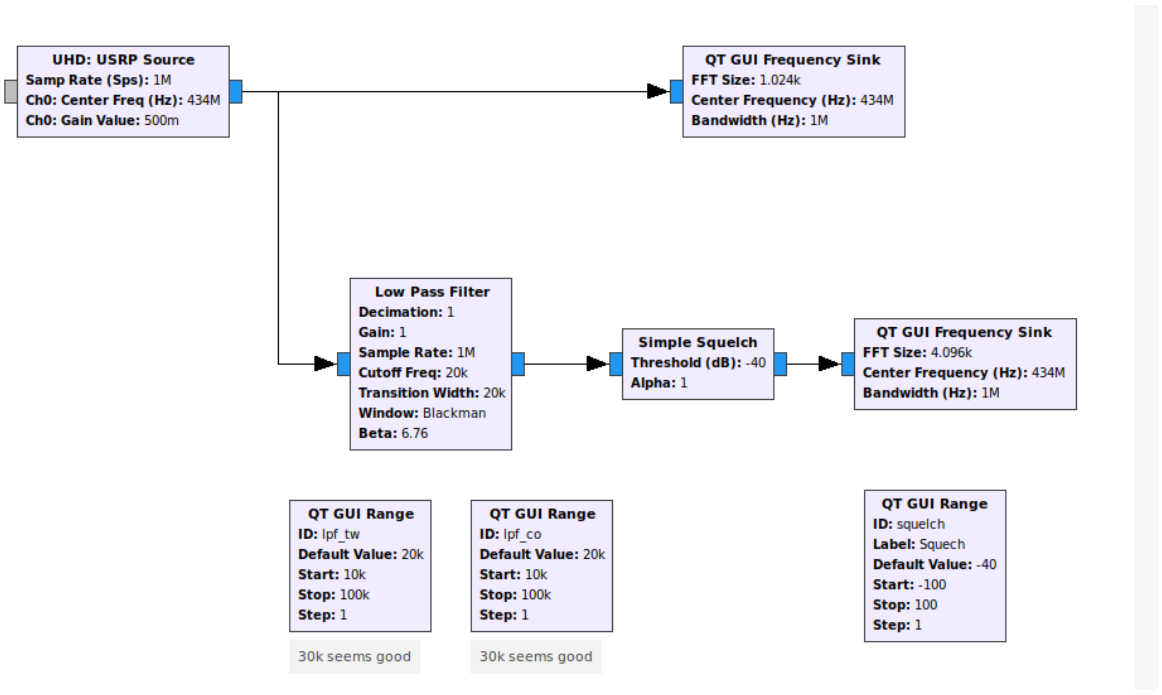


Figure 16: An ideal output from the LPF



UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 434M
Ch0: Gain Value: 500m

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 434M
Bandwidth (Hz): 1M

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Cutoff Freq: 20k
Transition Width: 20k
Window: Blackman
Beta: 6.76

Simple Squelch
Threshold (dB): -40
Alpha: 1

QT GUI Frequency Sink
FFT Size: 4.096k
Center Frequency (Hz): 434M
Bandwidth (Hz): 1M

QT GUI Range
ID: lpf_tw
Default Value: 20k
Start: 10k
Stop: 100k
Step: 1

30k seems good

QT GUI Range
ID: lpf_co
Default Value: 20k
Start: 10k
Stop: 100k
Step: 1

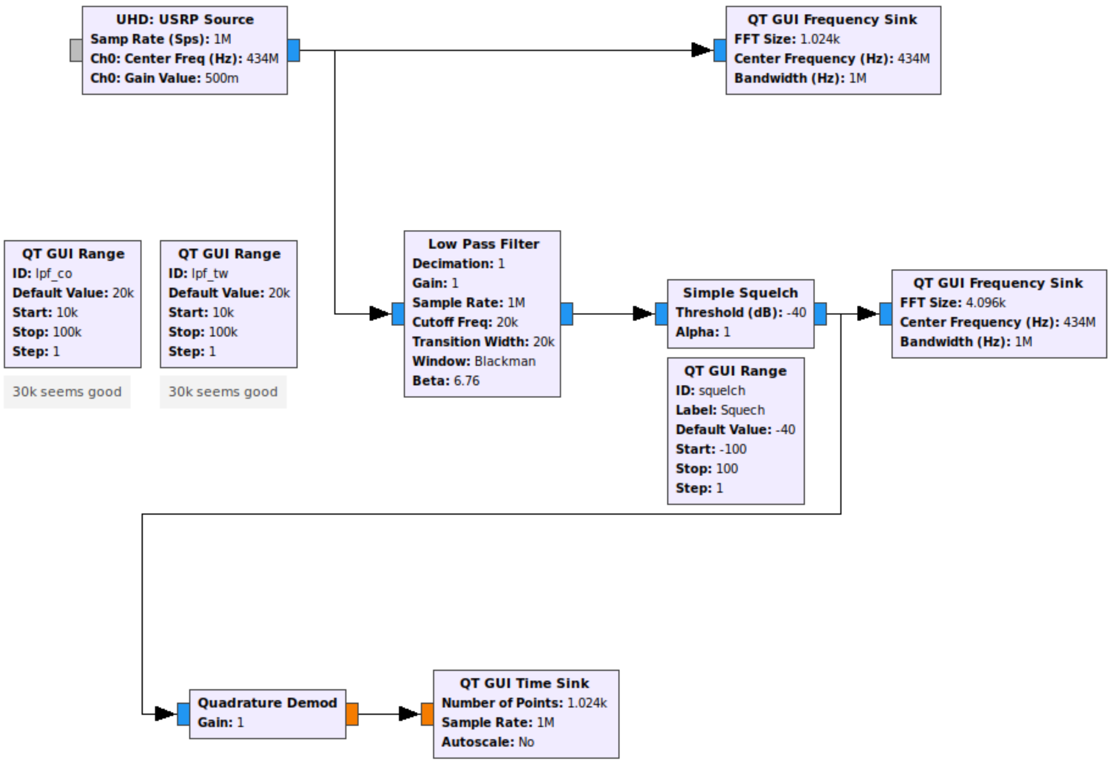
30k seems good

QT GUI Range
ID: squelch
Label: Squech
Default Value: -40
Start: -100
Stop: 100
Step: 1

Figure 17: 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.



UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 434M
Ch0: Gain Value: 500m

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 434M
Bandwidth (Hz): 1M

QT GUI Range
ID: lpf_co
Default Value: 20k
Start: 10k
Stop: 100k
Step: 1

30k seems good

QT GUI Range
ID: lpf_tw
Default Value: 20k
Start: 10k
Stop: 100k
Step: 1

30k seems good

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Cutoff Freq: 20k
Transition Width: 20k
Window: Blackman
Beta: 6.76

Simple Squelch
Threshold (dB): -40
Alpha: 1

QT GUI Frequency Sink
FFT Size: 4.096k
Center Frequency (Hz): 434M
Bandwidth (Hz): 1M

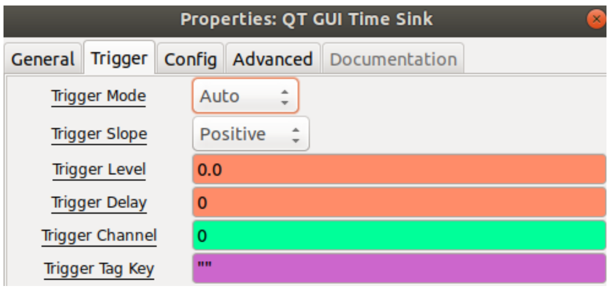
QT GUI Range
ID: squelch
Label: Squech
Default Value: -40
Start: -100
Stop: 100
Step: 1

Quadrature Demod
Gain: 1

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 1M
Autoscale: No

Figure 18: 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, and when the flow is running click '+ X Max' loads until you can see what looks like something interesting.



Properties: QT GUI Time Sink

General | Trigger | Config | Advanced | Documentation

Trigger Mode	Auto
Trigger Slope	Positive
Trigger Level	0.0
Trigger Delay	0
Trigger Channel	0
Trigger Tag Key	""

Figure 19: Setting the timesink trigger to auto in the GUI

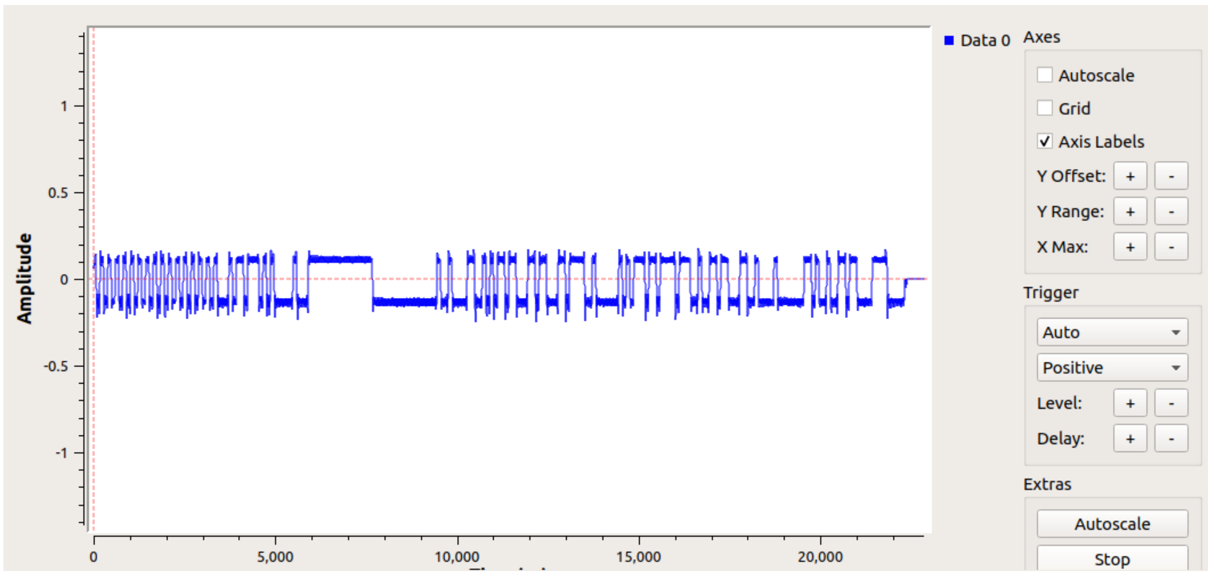


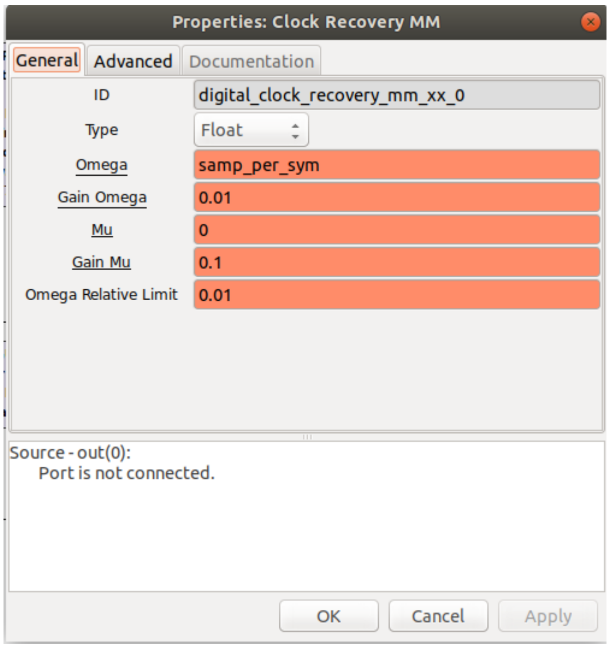
Figure 20: Our signal against time

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*.



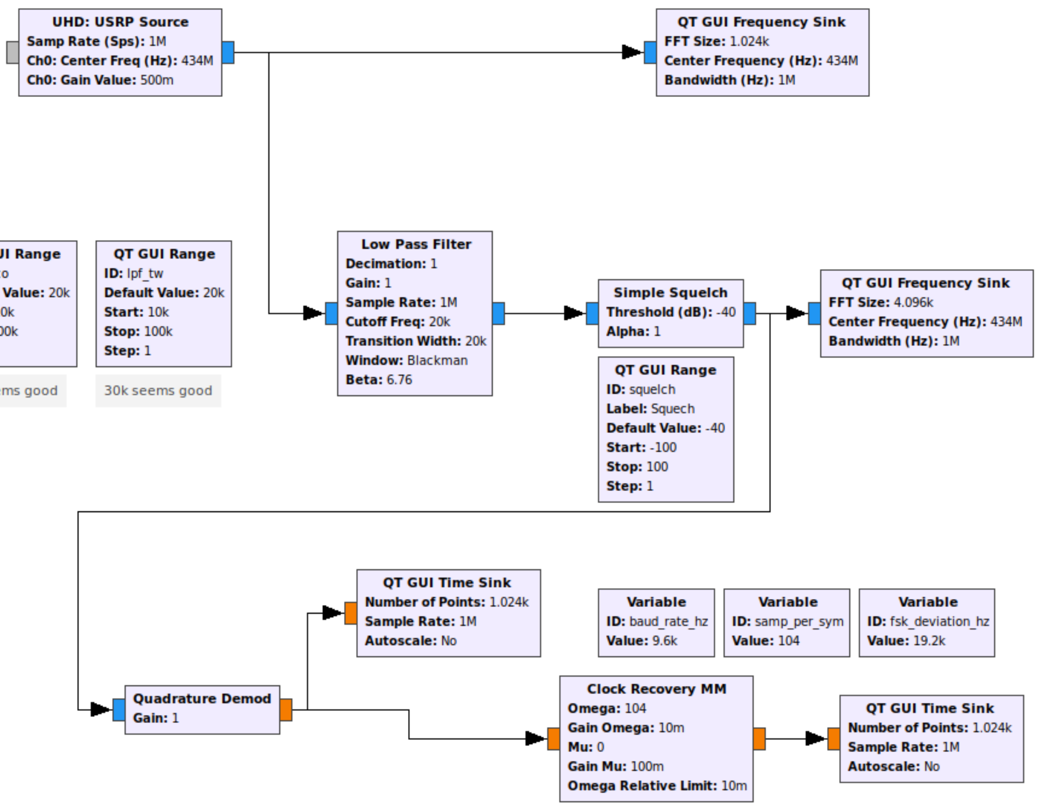
Properties: Clock Recovery MM

ID	digital_clock_recovery_mm_xx_0
Type	Float
Omega	samp_per_sym
Gain Omega	0.01
Mu	0
Gain Mu	0.1
Omega Relative Limit	0.01

```
Source - out(0):
    Port is not connected.
```

Figure 21: Clock recovery block settings

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



UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 434M
Ch0: Gain Value: 500m

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 434M
Bandwidth (Hz): 1M

QT GUI Range
ID: lpf_co
Default Value: 20k
Start: 10k
Stop: 100k
Step: 1

30k seems good

QT GUI Range
ID: lpf_tw
Default Value: 20k
Start: 10k
Stop: 100k
Step: 1

30k seems good

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Cutoff Freq: 20k
Transition Width: 20k
Window: Blackman
Beta: 6.76

Simple Squelch
Threshold (dB): -40
Alpha: 1

QT GUI Range
ID: squelch
Label: Squech
Default Value: -40
Start: -100
Stop: 100
Step: 1

QT GUI Frequency Sink
FFT Size: 4.096k
Center Frequency (Hz): 434M
Bandwidth (Hz): 1M

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 1M
Autoscale: No

Variable
ID: baud_rate_hz
Value: 9.6k

Variable
ID: samp_per_sym
Value: 104

Variable
ID: fsk_deviation_hz
Value: 19.2k

Quadrature Demod
Gain: 1

Clock Recovery MM
Omega: 104
Gain Omega: 10m
Mu: 0
Gain Mu: 100m
Omega Relative Limit: 10m

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 1M
Autoscale: No

Figure 22: 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.

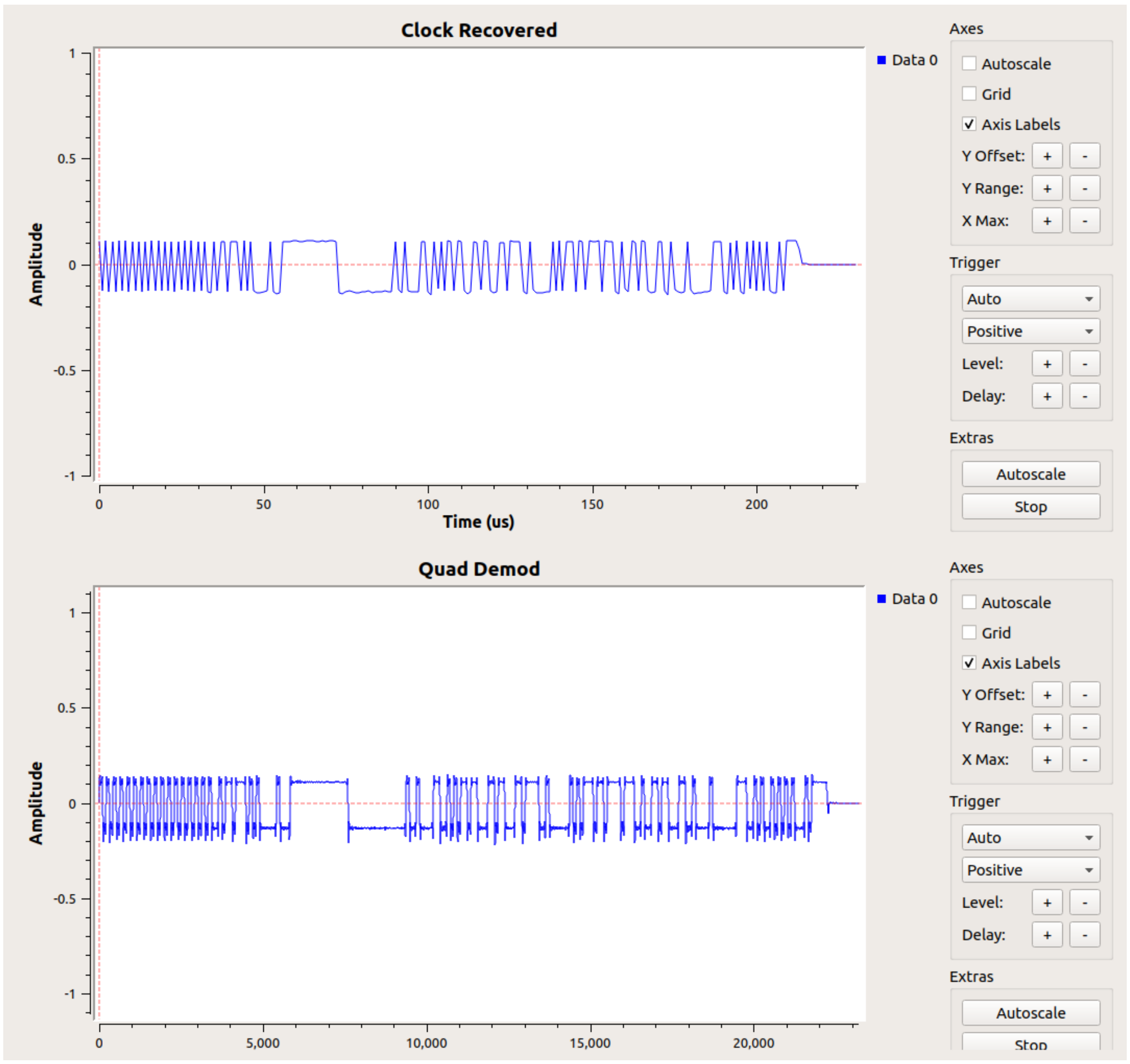


Figure 23: The signal before and after clock recovery

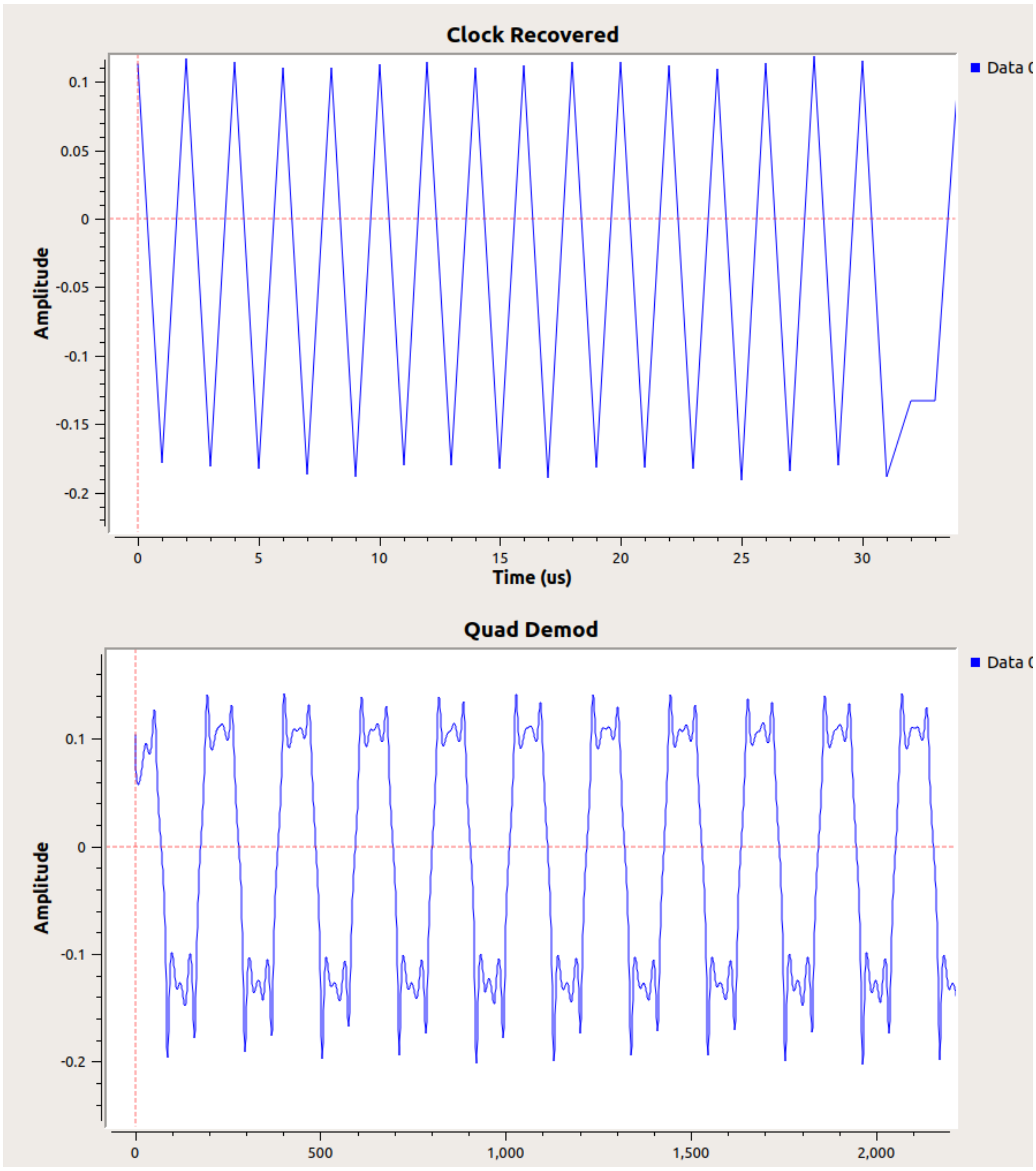
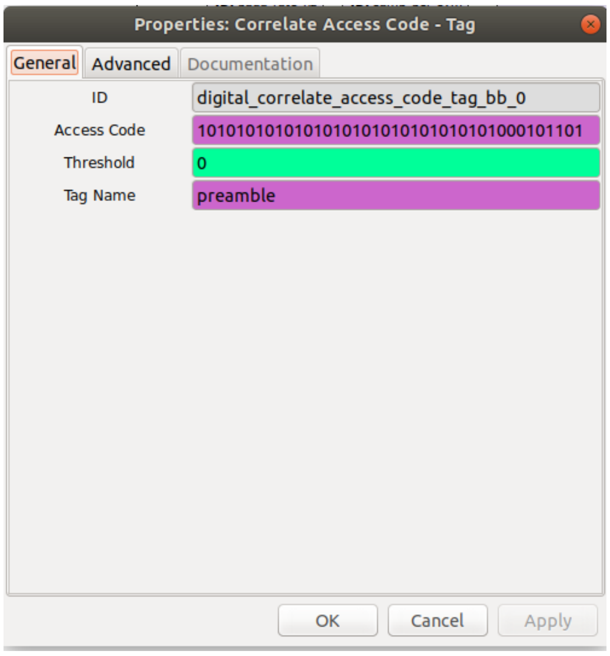


Figure 24: Detail of a series of 1 and 0 before and after clock recovery

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 1010101010101010101010101010101000101101 (or 4 bytes of 10101010 then a byte of 00101101), a threshhold of 0, and call the tag 'preamble'.



ID	digital_correlate_access_code_tag_bb_0
Access Code	1010101010101010101010101010101000101101
Threshold	0
Tag Name	preamble

Figure 25: 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 Meta 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.

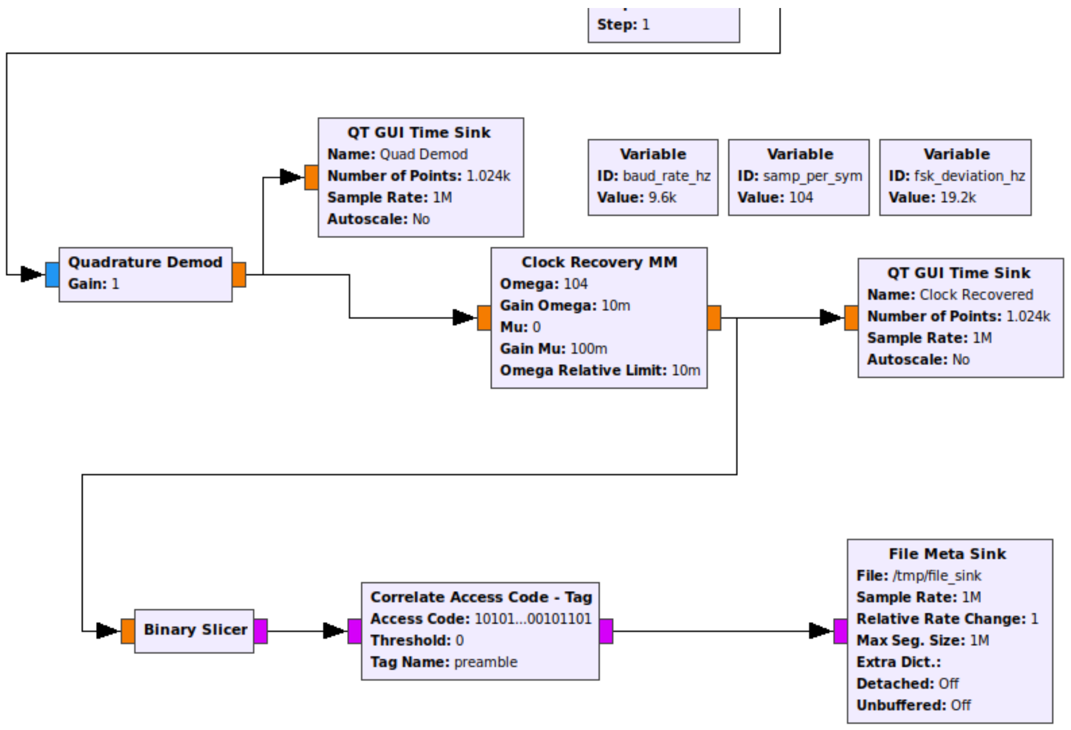
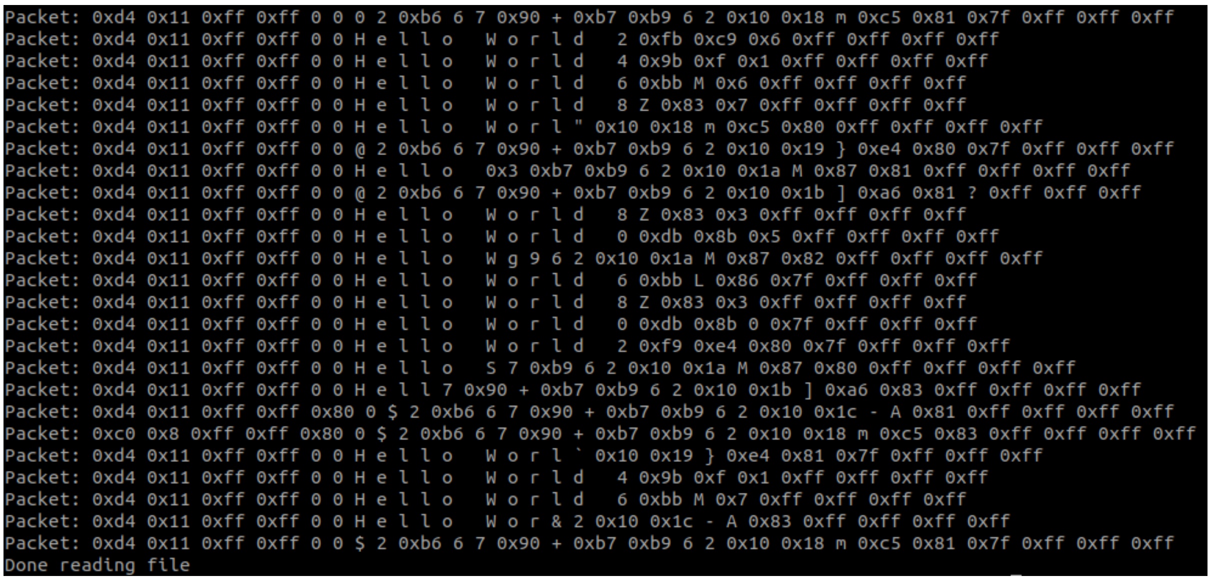


Figure 26: Radio - LPF - Squelch - Quad Demod - Clock Recovery - Binary Slicer - Correlate Access Code - File Sink

5.6 Decode

This is where it can get abit tougher! I have cobbled together a small tool to help. It's not that great and misses some of the messages, however it's a start! In the repo you downloaded use the analyser to examine the contents of the file you created. This tool looks for the 'preamble' tag in the file and then prints off a certain number of bytes afterwards to the screen. In order to build the tool navigate to it's directory and enter the command 'source doit.sh'. This will creat a folder called 'build' and make the application there. Then you can use it by entering the command './analyser /tmp/file_sink' and it should print off some of the packets.



```
Packet: 0xd4 0x11 0xff 0xff 0 0 0 2 0xb6 6 7 0x90 + 0xb7 0xb9 6 2 0x10 0x18 m 0xc5 0x81 0x7f 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   2 0xfb 0xc9 0x6 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   4 0x9b 0xf 0x1 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   6 0xbb M 0x6 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   8 Z 0x83 0x7 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l " 0x10 0x18 m 0xc5 0x80 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 @ 2 0xb6 6 7 0x90 + 0xb7 0xb9 6 2 0x10 0x19 } 0xe4 0x80 0x7f 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   0x3 0xb7 0xb9 6 2 0x10 0x1a M 0x87 0x81 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 @ 2 0xb6 6 7 0x90 + 0xb7 0xb9 6 2 0x10 0x1b ] 0xa6 0x81 ? 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   8 Z 0x83 0x3 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   0 0xdb 0x8b 0x5 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W g 9 6 2 0x10 0x1a M 0x87 0x82 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   6 0xbb L 0x86 0x7f 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   8 Z 0x83 0x3 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   0 0xdb 0x8b 0 0x7f 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   2 0xf9 0xe4 0x80 0x7f 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   S 7 0xb9 6 2 0x10 0x1a M 0x87 0x80 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l 7 0x90 + 0xb7 0xb9 6 2 0x10 0x1b ] 0xa6 0x83 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0x80 0 $ 2 0xb6 6 7 0x90 + 0xb7 0xb9 6 2 0x10 0x1c - A 0x81 0xff 0xff 0xff 0xff
Packet: 0xc0 0x8 0xff 0xff 0x80 0 $ 2 0xb6 6 7 0x90 + 0xb7 0xb9 6 2 0x10 0x18 m 0xc5 0x83 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l ` 0x10 0x19 } 0xe4 0x81 0x7f 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   4 0x9b 0xf 0x1 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r l d   6 0xbb M 0x7 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 H e l l o   W o r & 2 0x10 0x1c - A 0x83 0xff 0xff 0xff 0xff
Packet: 0xd4 0x11 0xff 0xff 0 0 $ 2 0xb6 6 7 0x90 + 0xb7 0xb9 6 2 0x10 0x18 m 0xc5 0x81 0x7f 0xff 0xff 0xff
Done reading file
```

Figure 27: Output of the analyser

It appears to skip every other packet at this stage so, however we can clearly see a bunch of 'Hello World' messages decoded - congratulations you have successfully performed a MITM attack on your feather!

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

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/`