

Abgabe PHYSEC 2

1. FM-Empfänger

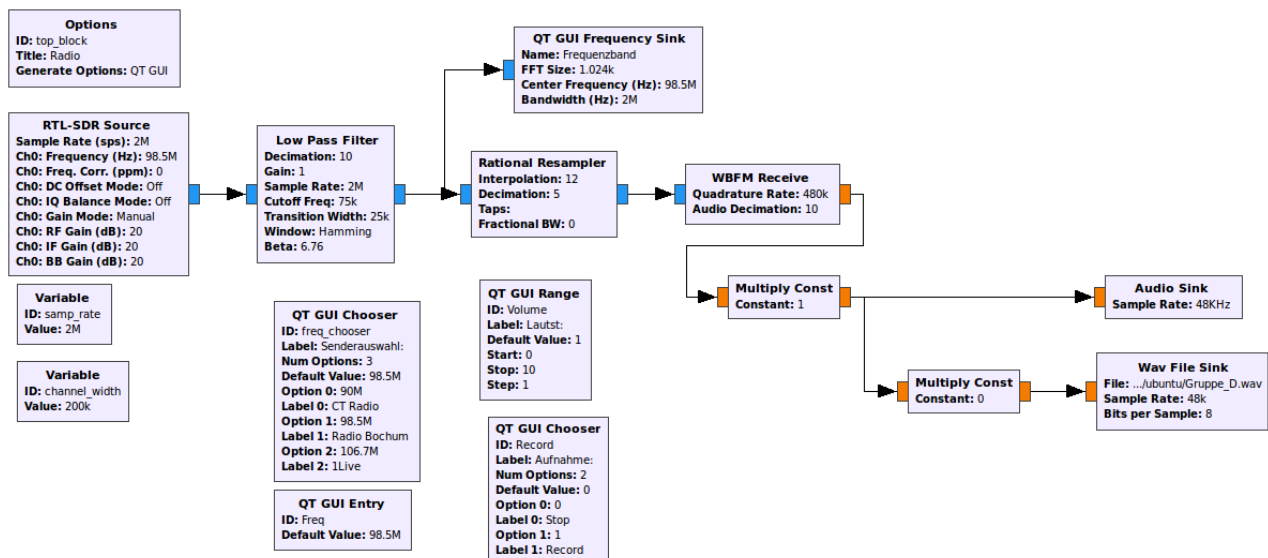


Abbildung 1: Schaltung in GRC

Wie in Abbildung 1 zu erkennen ist, wurden verschiedene Bausteine verwendet, die nun aufgezählt werden. Dabei wird auch darauf eingegangen, welche Aufgaben sie bewältigen.

- *Options*: Ermöglicht einzustellen, welche Art von GUI, hier QT, verwendet werden soll
- *RTL-SDR Source*: Es ermöglicht die Verarbeitung von empfangenen Signalen, mithilfe von angeschlossener Hardware, hier *HackRF*
- *Variable* (oben und unten): Bequeme Steuerung der Werte, die in Baublöcke einfließen
- *Low Pass Filter*: Dient zur Verfeinerung des Signals durch Filtern. Laut Software Defined Radio with HackRF, Lesson 1: Welcome ist es Teil der Demodulation. Es dient als Vorbereitung der eigentlichen Demodulation
- *QT GUI Chooser* (links) und *QT GUI Entry*: Dient zur Senderauswahl, oder genauer, der Frequenzauswahl
- *QT Frequency Sink*: Visuelle Verdeutlichung des Signals. Es zeigt die Wirkung des *Low Pass Filters*

- *Rational Resampler*: Da der *Low Pass Filter* nur eine ganzzahlige *Decimation* unterstützt wird zusätzlich dieser *Resampler* benötigt
- *QT GUI Range* und *Multiply Const* (links): Dient der Lautstärke-Einstellung
- *GT GUI Chooser* (rechts), *Multiply Const* (rechts) und *Wav File Sink*: Regelt die Aufnahme mit *Record* und *Stop* des Empfangs als auch die Abspeicherung als *Gruppe_D.wav*
- *WBFM Receive*: Der eigentliche Demodulierer laut
Software Defined Radio with HackRF, Lesson 1: Welcome.
Es formt das Signal in Audio um

Zusatz:

Samp_rate ist hier 2 Mio. Die *Decimation* beim *Low Pass Filter* ist 10:

$$\frac{2000000}{10} = 200000$$

Der *Rational Resampler* hat eine *Interpolation* von 12 und eine *Decimation* von 5:

$$200000 * \frac{12}{5} = 480000$$

Der *WBFM Receiver* hat eine *Decimation* von 10:

$$\frac{480000}{10} = 48000$$

48kHz ist eine gängige *Sample Rate*, die von nahezu allen *Soundcards* unterstützt wird.

Hinweise:

- Die Hyperlinks oben sind schon mit den passenden Timestamps versehen
- Der von GRC ausgegebene Python2 Sourcecode und die entsprechende .grc-Datei können <hier> heruntergeladen werden

Abbildung 2 zeigt die fertige GUI. Oben lässt sich die Lautstärke einstellen. Ist der Slider ganz links, dann ist das Radio stumm. Darunter lässt sich mit *Record* und *Stop* die Aufnahme steuern. Bei *Freq.* kann man die gewünschte Frequenz eingeben. Bei *Senderauswahl* kann man zu *CT Radio*, *Radio Bochum* und *ILive* schalten.

Es folgen danach die Einstellungen der wichtigsten Bauteile, die in Abbildung 1 zu sehen sind in Form von Screenshots.

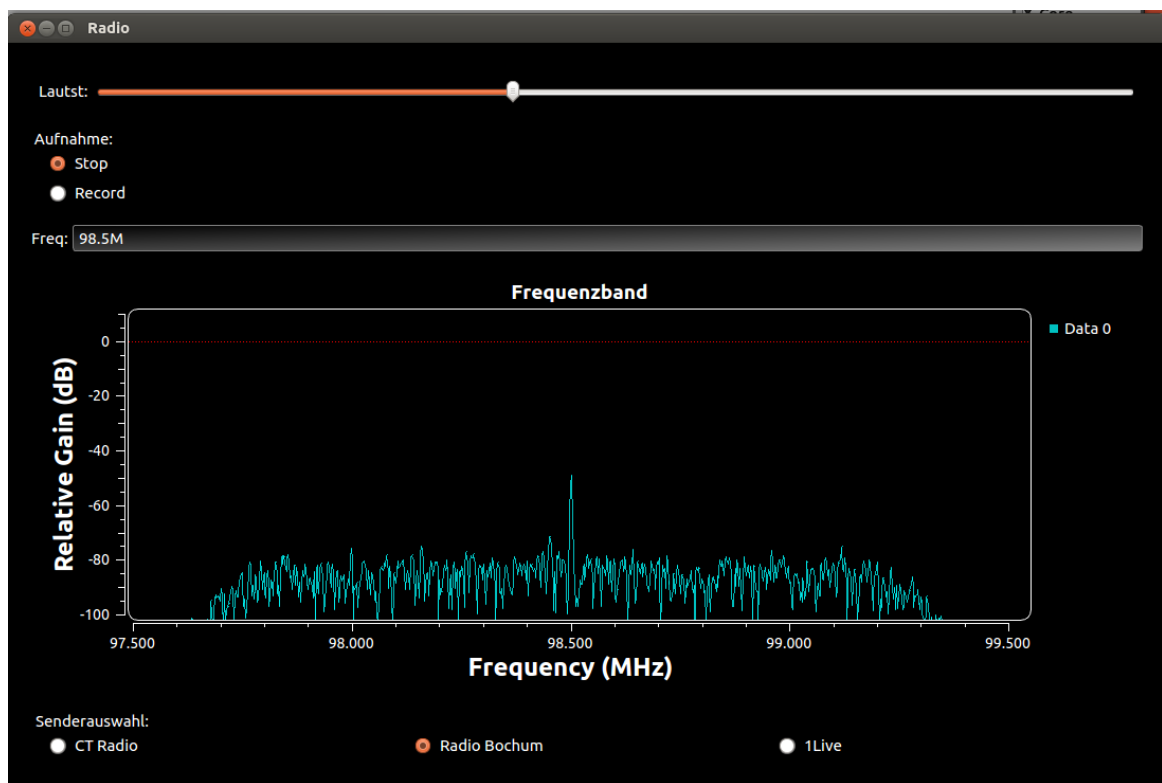
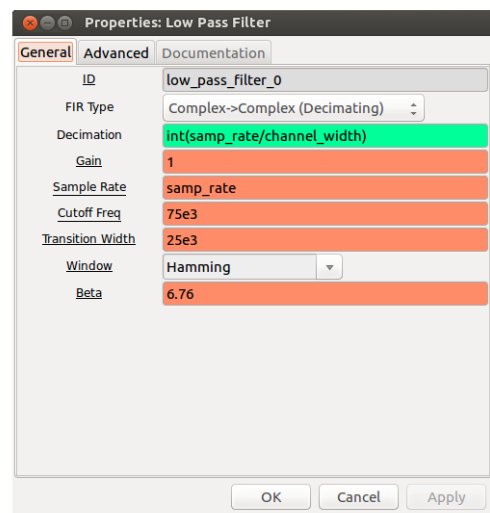
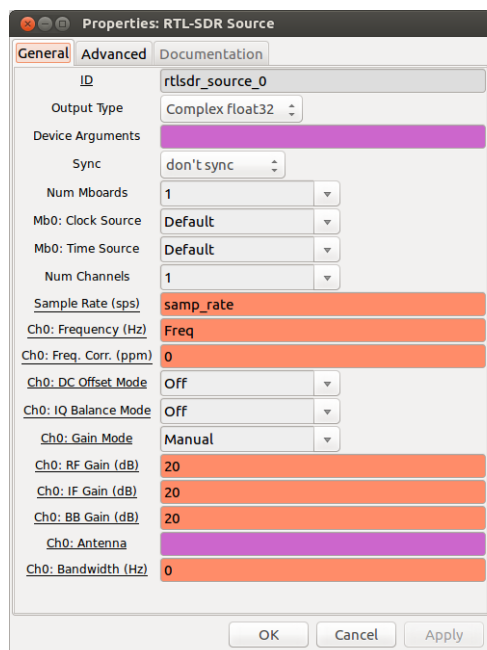


Abbildung 2: Fertige GUI



Properties: QT GUI Chooser

General Advanced Documentation

ID: freq_chooser

Label: Senderauswahl:

Type: Float

Num Options: 3

Default Value: 98.5e6

Option 0: 90e6

Label 0: CT Radio

Option 1: 98.5e6

Label 1: Radio Bochum

Option 2: 106.7e6

Label 2: 1Live

Widget: Radio Buttons

Orientation: Horizontal

GUI Hint:

OK Cancel Apply

Properties: QT GUI Frequency Sink

General Trigger Config Advanced Documentation

ID: qtgui_freq_sink_x_0

Type: Complex

Name: "Frequenzband"

FFT Size: 1024

Window Type: Blackman-harris

Center Frequency (Hz): Freq

Bandwidth (Hz): samp_rate

Grid: No

Autoscale: No

Average: None

Y_min: -100

Y_max: 10

Y label: Relative Gain

Y units: dB

Number of Inputs: 1

Update Period: 0.30

GUI Hint:

Show Msg Ports: No

OK Cancel Apply

Properties: Rational Resampler

General Advanced Documentation

ID: rational_resampler_xxx_0

Type: Complex->Complex (Complex Taps)

Interpolation: 12

Decimation: 5

Taps:

Fractional BW: 0

OK Cancel Apply

Properties: WBFM Receive

General Advanced Documentation

ID: analog_wfm_rcv_0

Quadrature Rate: 480e3

Audio Decimation: 10

OK Cancel Apply

Properties: QT GUI Chooser

General Advanced Documentation

ID: Record

Label: Aufnahme:

Type: Integer

Num Options: 2

Default Value: 0

Option 0: 0

Label 0: Stop

Option 1: 1

Label 1: Record

Widget: Radio Buttons

Orientation: Vertical

GUI Hint:

OK Cancel Apply

Properties: QT GUI Range

General Advanced Documentation

ID: Volume

Label: Lautst:

Type: Float

Default Value: 1

Start: 0

Stop: 10

Step: 1

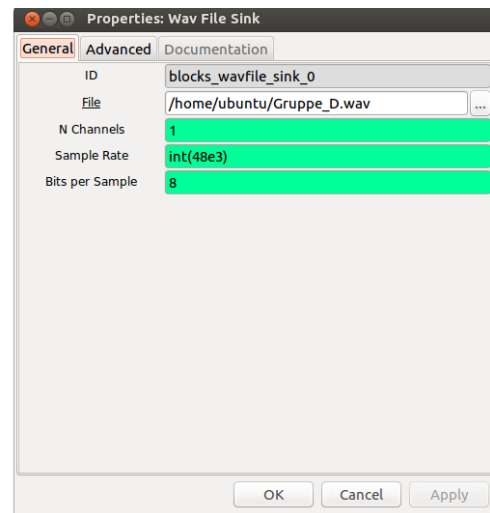
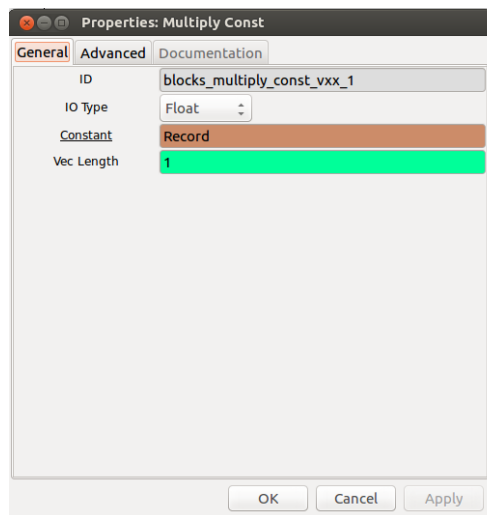
Widget: Slider

Orientation: Horizontal

Minimum Length: 200

GUI Hint:

OK Cancel Apply



2. Spectrum Spraying

```
import scipy.ndimage as img
import numpy as np

def makeBinary(input_filename, output_filename='out.bin'):
    # Options for the FFT
    Fs = 18000000
    T_line = 0.005

    # Read input image
    image = img.imread(input_filename)

    # Set FFT size to be double the image size so that the
    # edge of the spectrum stays clear
    # preventing some bandfilter artifacts
    NFFT = 2*image.shape[1]

    # Repeat image lines until each one comes often enough to
    # reach the desired line time
    repetitions = int(np.ceil(T_line * Fs / NFFT))
    ffts = ((np.repeat(image[:, :, 0], repetitions, axis=0)/16.)**2.)/256.

    # Embed image in center bins of the FFT
    fftall = np.zeros((ffts.shape[0], NFFT))
    startbin = int(NFFT/4)
    fftall[:, startbin:(startbin+image.shape[1])] = ffts

    # Generate random phase vectors for the FFT bins to prevent high peaks
    # in the output. The phases won't be visible in the spectrum
    phases = 2*np.pi*np.random.rand(*ffts.shape)
    rffts = fftall * np.exp(1j*phases)

    # Perform the FFT per image line, then concatenate them to form the
    # final signal
    timedata = np.fft.ifft(np.fft.ifftshift(
        rffts, axes=1), axis=1)/np.sqrt(float(NFFT))
    linear = timedata.flatten()
    linear = linear / np.max(np.abs(linear))

    # Match the requirements
    res = np.zeros(2*linear.size, dtype=np.float32)
    res[0::2], res[1::2] = np.real(linear), np.imag(linear)

    # Get the Output
    res.tofile(output_filename)

    return

# Create the Signals
makeBinary('signal_kreis.png', 'signal_kreis.bin')
makeBinary('physec.png', 'physec.bin')
```

- physec

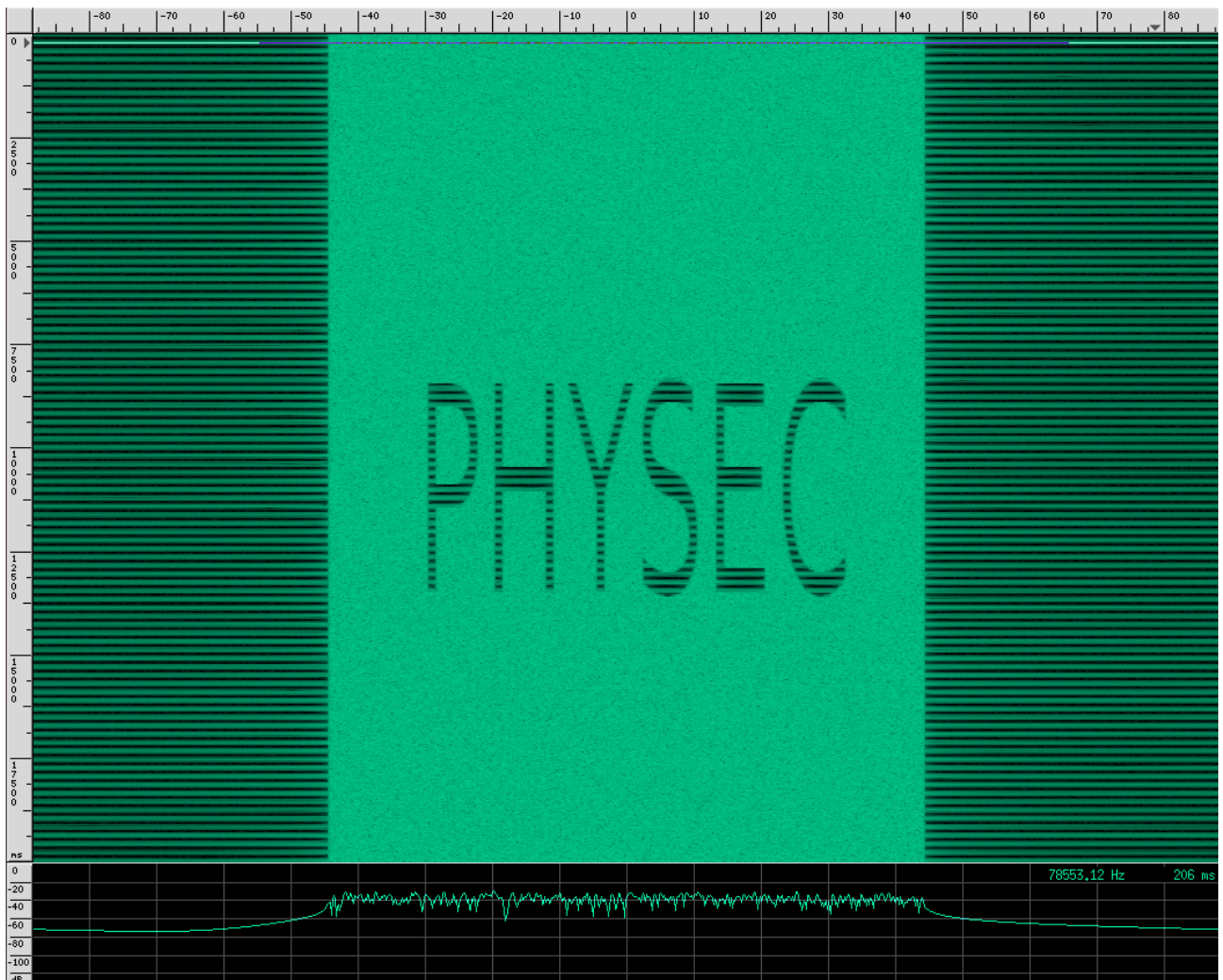


Abbildung 3: physec.bin in Baudline

- kreis

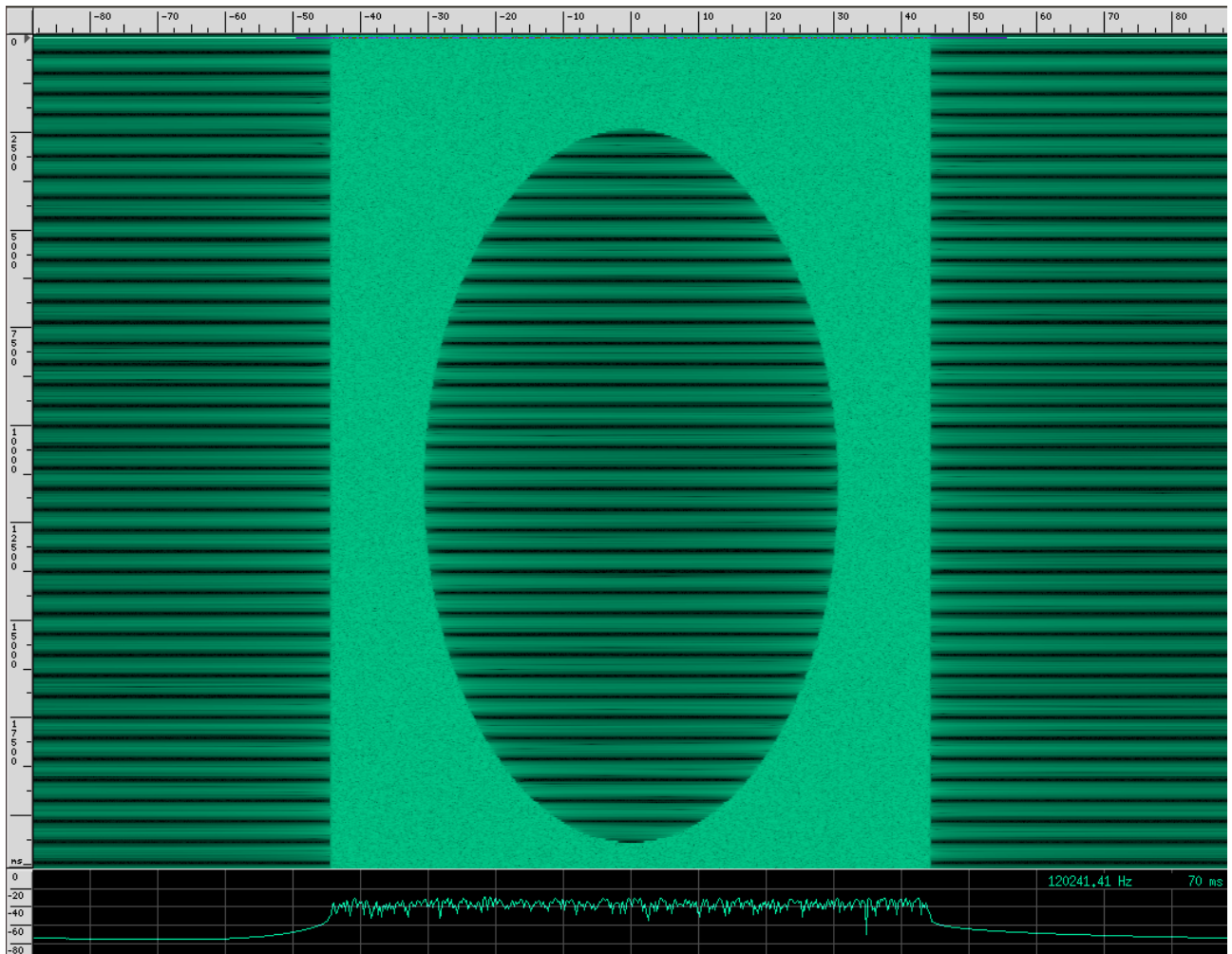


Abbildung 4: signal_kreis.bin in Baudline

<Hier> können die Signale sowie der Pythoncode heruntergeladen werden.

3. Spectrum Sensing

Bei *Spectrum Sensing* oder in deutsch *Spektrumerfassung* handelt sich um ein Scannen eines Frequenzbandes. Hier wird das Spektrum von 75MHz - 1GHz erfasst. Das RTL-SDR durchläuft Stück für Stück (abhängig von der Samplerate) das gesamte Band und durchsucht dieses nach Signalen.

Komplettes Frequenzband

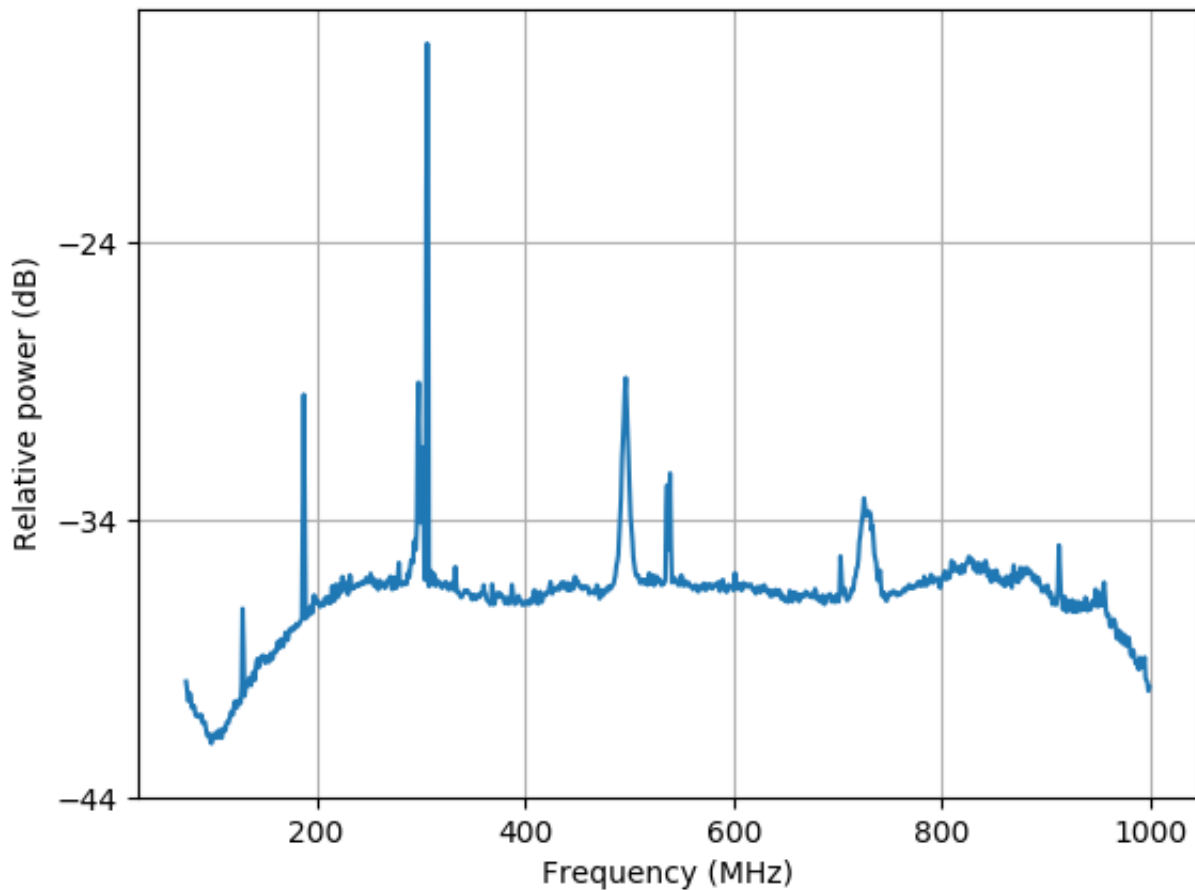


Abbildung 5: Frequenzband

Energy Detection

Signale

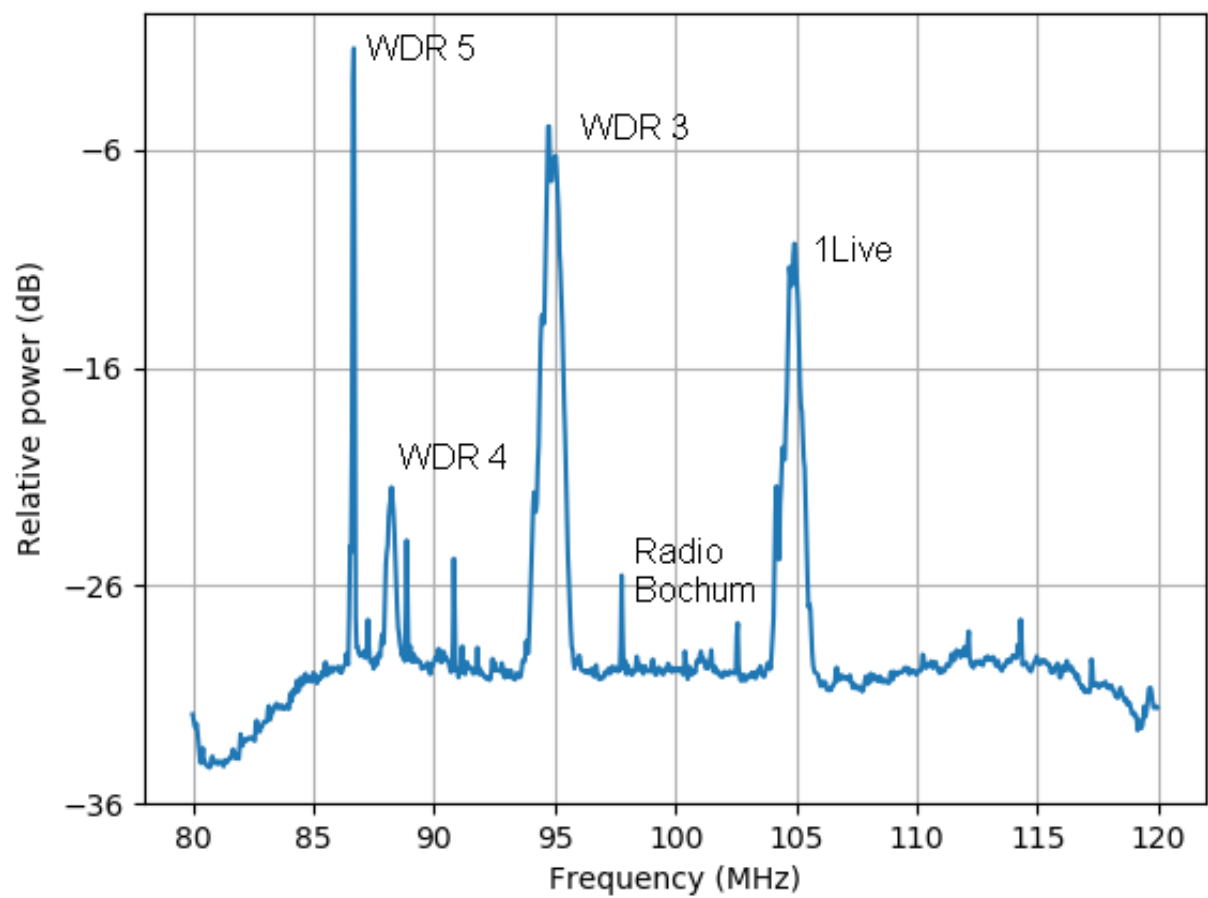


Abbildung 6: Erkannte Radiosignale beim Spectrum Scan

Unterschied zwischen RTL-SDR und HackRF

Es wurde der UKW-Bereich zwischen 80 MHz und 120 MHz mithilfe von *qspectrumanalyzer* gescannt.

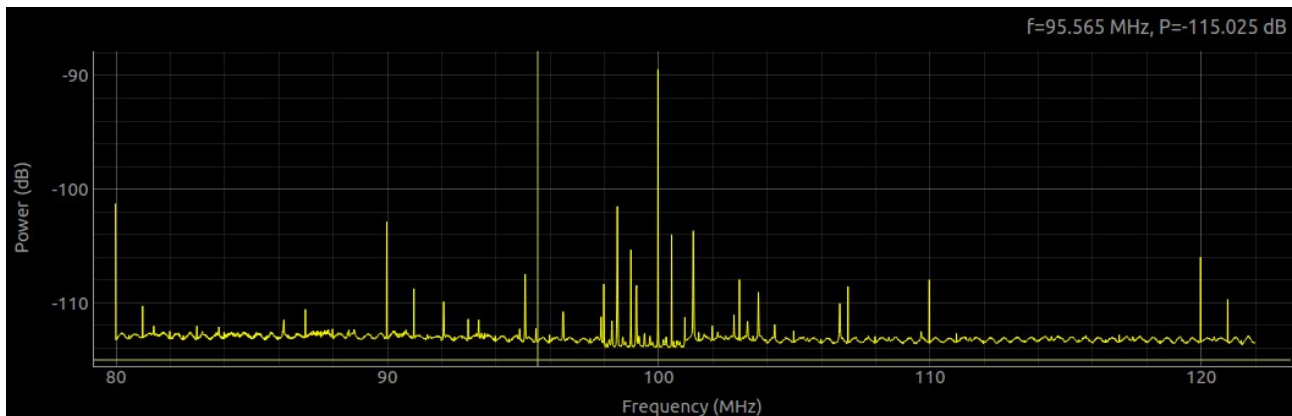


Abbildung 7: HackRF

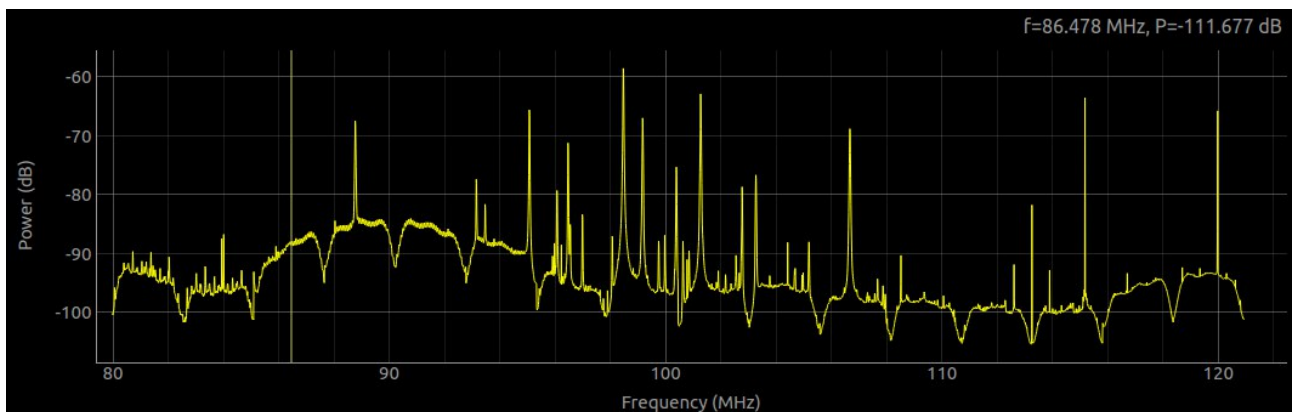


Abbildung 8: RTL-SDR

Wie in den Abbildungen 7 und 8 erkennbar ist, sind die Peaks bei *RTL-SDR* deutlich höher auf der Y-Achse. Außerdem sind in diesem Messbereich bei *RTL-SDR* einige etwa gleich große Slots vermerkbar. Bei *HackRF* ist dieses Verhalten in der jeweiligen Messung nicht beobachtbar, als auch bei wiederholten Messungen. Nicht nur die Peaks, sondern auch das Grundlevel hat bei *RTL-SDR* in der Abbildung 8 wesentlich mehr Leistung.

4. Reading assignment

a)

Siehe Seite 36 des Papers sollen damit *resource depletion attacks* und *masquerade attacks* abgewehrt werden.

b)

Siehe Seite 37 des Papers haben Signalprints die folgenden Eigenschaften:

- Sie sind schwer zu *spoofen*
- Sie weisen eine starke Korrelation in Hinblick auf ihre physikalische Umgebung auf
- Über mittlere Zeitdauern variieren sie nur leicht

c)

Node Induction

Da der Sender nur mithilfe des empfangenen Signalprints, welches mit dem Referenz-Signalprints verglichen wird, identifiziert werden kann, muss das voraussetzen, dass das Referenz-Signalprint existiert. Falls jedoch eine Node zum ersten mal mit dem Netzwerk interagiert, fehlt logischerweise dieses Referenz-Signalprint, was das Identifizieren dieser Node nicht möglich macht. Wenn eine Node sich dem Netzwerk anschließen will, legt sie offen auf welchen Kanälen sie die restlichen Übertragungen senden wird. Die Empfänger stellen sich darauf ein und messen ihre jeweiligen RSSI Werte, die daraufhin über das Netzwerk verteilt werden.

Frequent Hello Messages

Aufgrund der kontinuierlichen Batterieentladung sinkt die Übertragungsstärke zunehmend, was zur Folge hat, dass die RSSI-Werte mit der Zeit abnehmen. Als Konsequenz wird der Vergleich der RSSI-Werte mit dem Signalprint nach einer gewissen Zeit beeinträchtigt und somit die Identifikation. Im Allgemeinen kann die Entladungsrate nicht ohne Weiteres vom Empfänger vorausgesagt werden, da es von der Ladung des Senders abhängt. Deshalb wird das regelmäßige Versenden von „hello“ Paketen empfohlen. Immer wenn ein Empfänger eine Übertragung eines Senders erhält, vergleicht dieser dann den empfangenen RSSI-Wert mit dem des Referenz-Signalprints. Erreicht die Differenz eine gewisse Größe, wird die Generierung eines neuen Referenz-Signalprints angefordert. Oft reicht der Verkehr von genuinen Übertragungen aus, aber ist das nicht der Fall müssen „hello“ Pakete übertragen werden zur erfolgreichen Verifizierung der RSSI-Werte.

Data Transmission

Nicht jede Datenübertragung wird mithilfe eines Signalprints verifiziert. Je mehr Empfänger zusammenarbeiten um ein Signalprint zu generieren, desto höher die Verlässlichkeit des Signalprints. Es

kann passieren, dass zu einem Zeitpunkt nicht genug Empfänger auf einen bestimmten Kanal eingestellt sind um einen ausreichend verlässlichen Signalprint zu erhalten. Deshalb wird eine Methodik bestehend aus zwei Schritten verwendet:

- Das Netzwerk hält Ausschau nach verdächtiger Aktivität
- Falls so eine Aktivität bemerkt wurde, stimmt das Netzwerk eine ausreichende Anzahl an Empfängern auf die entsprechenden Kanäle ein und ein Signalprint wird erzeugt