Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik und Kommunikationsnetze

Frequency Spectrum Monitoring for FlockLab

Semester Thesis

Florian Wernli
fwernli@ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

Supervisors:
Roman Trüb
Dr. Jan Beutel
Prof. Dr. Lothar Thiele

June 28, 2019

Acknowledgements

I heartily thank my supervisor, Roman Trüb, doctorate at D-ITET, TIK, for his encouragement, guidance and especially for always making time for extensive weekly meetings.

Abstract

The Computer Engineering Group (TEC) operates the FlockLab testbed, for developing and evaluating wireless sensor network protocols. With nodes distributed over the campus the sensors are exposed a real-world environment. Currently, the system relies on GPIO and power tracing to evaluate the system's performance, thus being unable to spot interference caused by third-party wireless devices. This project provides the foundations to extend the current setup with the possibility to sense and characterize the radio frequency spectrum while the tests are running. Based on GNU Radio an application has been developed to record the electromagnetic spectrum. The focus of the project was to achieve measurements that are reproducible and comparable while reducing the amount of produced data to a manageable size. The software works with either an RTL-SDR or a USRP B210, the characteristic of both devices has been analyzed and compared to the output of the Tektronix RSA306. Data reduction is achieved by accumulating instantaneous spectra to larger blocks and only saving the averaged power spectrum, as well as the minimal and maximal observed power per frequency bin of these blocks. Information loss due to the averaging is partly compensated by annotating anomalies in the noise distribution and an experimental indication of whether an FSK or a LoRa modulated signal was recorded.

Keywords:
spectrum sensing, power spectral density, RTL-SDR, USRP

Contents

Chapter 1

Introduction

1.1 Motivation

The development of wireless sensor networks requires a proper test environment. For that reason, the Computer Engineering Group (TEC) operates the FlockLab testbed [1] since 2012. The testbed reduces the effort of repeatedly deploying test networks and improves the reproducibility of experiments.

Up to now, FlockLab does not provide a service for monitoring the radio frequency spectrum. The additional information can be used as a further method to verify the correct operation. Mostly the effect of the environment, such as interference, has not been captured by the current setup.

The problem when recording the electromagnetic spectrum is the large amount of data. This obvious solution of constantly sampling signals within the bandwidth of interest, would create roughly 450 MiB per minute per MHz bandwidth. Thus it is highly impractical for longer observations.

1.2 Goals

The goal of the project is to develop a system that is able to capture the radio frequency spectrum with a frequency resolution of 10 kHz and a time resolution or 0.1 s. The data acquisition is performed by a software-defined radio (SDR), either an RTL-SDR Blog v3[1] or a USRP B210[2] can be used as hardware frontends. In the following the devices will be called RTL-SDR and USRP respectively.

To counteract the loss introduced by the low frequency resolution and the long averaging time, the samples are processed beforehand and possible signals are annotated.

[1] `https://www.rtl-sdr.com/buy-rtl-sdr-dvb-t-dongles/`
[2] `https://www.ettus.com/all-products/UB210-KIT/`

The output should be comparable and reproducible, thus the characteristics of the devices are analyzed and used. The difference to most SDR projects is that decoding the signal is not a goal. Instead, the power levels of the noise and signals in the environment are of interest.

1.3 Related work

The availability of cheap SDRs motivated the start of different projects observing electromagnetic emissions. Most available software does not bother saving the spectrum in any useful format but instead focuses on demodulating the signal to either a byte or audio stream. Examples for such software are *GQRX*[3] or *SDR#*[4]. There are also web-based receiver applications like *OpenWebRX*[5]. They usually allow exporting raw IQ data which can be viewed with *inspectrum*[6] or processed with Python or Matlab. However, the fast growth of data makes the acquisition in raw format impractical.

The *ElectroSense* project [2] is similar to this project. Their goal is to monitor the spectrum to analyze the usage of the electromagnetic spectrum. Everyone can participate in the project. It is designed to use an RTL-SDR as receiver to keep the costs low and attract volunteers. A major difference, however, is the resolution at which they record the spectrum. They use the device to observe bandwidths much larger than the devices bandwidth, thus they have to change the centre-frequency. The tuning process of the RTL-SDR is slow, thus there are not only unobserved segments, but also a considerable amount of unusable samples during the retuning. Such a loss would not be acceptable for the use with FlockLab.

Something similar to ElectroeSense was run by Microsoft under the name *Microsoft Spectrum Observatory*[7]. Unfortunately, the project seems to be discontinued. A probable reason for the shutdown is the small number of participant, only a few US universities took part. The project used USRP devices thus the setup cost of a receiver station was a few thousand dollars.

When it comes to spectrum sensing the paper of T. Yücek and Hüseyin Arslan [3] provides a good overview of the problems and possible algorithms. A more detailed description of such an algorithm is found in QuickSense [4]. They propose a method for wideband multichannel sensing. For the classification of signals in the SRD860 band by their communication standard, the work of M. Kuba [5] is promising. He also contributed to a project that turned an Android

[3] `gqrx.dk`
[4] `https://airspy.com/`
[5] https://www.sdr.hu/openwebrx
[6] `https://github.com/miek/inspectrum`
[7] `https://www.microsoft.com/en-us/research/event/spectrum-observatory-thinktank/` and `http://observatory.microsoftspectrum.com/` (offline)

tablet and an RTL-SDR into a tool for traffic monitoring with automatic communication standard recognition [6]. With the data collected by the Spectrum Observatory, M. Zheleva et al. created TxMiner[7] to identify transmitter using a Rayleigh-Gaussian mixture model, that outperforms edge detection in both occupancy and bandwidth estimation accuracy.

1.4 About this Document

This document starts with a short introduction to software-defined radios and a reminder on how to read power spectral density plots. The next chapter presents the device characteristics the resulting limitations and the recommended mode of operation. This is followed by details on the implementation and the choice of default parameters. The final chapter presents test results showing the limits of comparability and reproducibility of the acquired data.

Chapter 2

Fundamentals of SDR and PSD

2.1 Software-defined Radio

Software-defined radios (SDR) are radio communication systems, where certain signal processing parts are performed digitally. The definition is rather loose and is applied to a wide range of setups including everything from devices only performing the decoding of baseband signal using a sound card connected to computer up to devices where the radio frequency is sampled directly, reducing the analog front-end to an amplifier, anti-aliasing filter and an ADC.

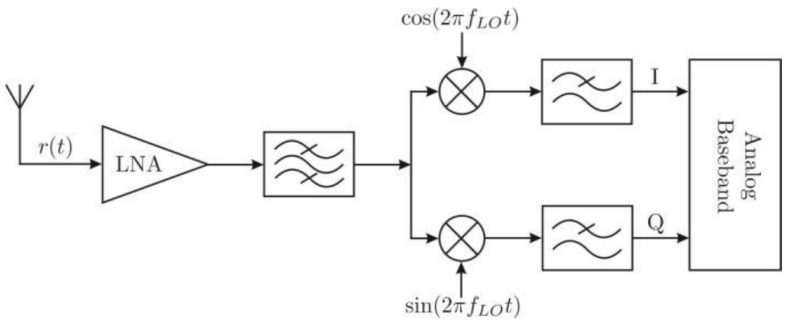


Figure 2.1: SDR Working principle

Most devices that are advertised as SDR work as depicted in Fig. 2.1. The RF signal is filtered and amplified, the mixer stage consists of two mixers working with a phase offset of 90° creating a quadrature signal. The in-phase and the quadrature component are filtered sampled by two different ADCs. The cost of using two ADCs is compensated by the advantages of the quadrature signal, twice the bandwidth is covered compared to a single ADC with the same sampling rate. Also, the IQ sample contains the phase of the signal.

The schematic representations of SDR often oversimplify the inner workings.

Mixing the signal directly to the baseband brings the disadvantage that DC voltages have to be handled properly. To circumvent this difficulty the signal is sampled at an intermediate frequency and shifted to the baseband with a digital down-converter.

2.2 Power Spectral Density

The most common method to estimate the power spectral density is the method proposed by Peter D. Welch [8]. The method works by splitting the time signal into segments which may overlap. For each segment, the squared magnitude of the DFT is calculated and then the average over all these transformed segments is taken. PSD estimates for wide-sense stationary processes are incorrect because the segments are approximately uncorrelated and averaging reduces the variance.

$$X_k = \mathcal{F}\left\{x_k[i]w[i]\right\} \tag{2.1}$$

$$P = \frac{1}{N \cdot M \cdot f_s \cdot G_N} \sum_k |X_k|^2 \tag{2.2}$$

Equation (2.2) shows the implementation of Welch's method as used by Matlab, Octave and SciPy, where

- x_k k-th segment
- w some window function
- N number of points used by the FFT
- M number of segments
- G_N noise gain, mean square of the window
- f_s sample rate

2.2.1 Windows

Choosing the right window for a proper readout requires a lot of specialized knowledge. Main lobe width, side lobe suppression and scallop loss are important window parameters which cannot be optimized at the same time. The choice of the window depends on factors such as how close two signals at different frequencies are, how large the amplitude difference of the signals are.

The Hann (sometimes Hanning) window is a popular choice. It provides a good spectral resolution and some leakage reduction. In low SNR cases the Blackman-Harris window with lower side lobes may bring better results. The frequency response of some popular windows is visible in figure 2.2.

[1] `http://rfmw.em.keysight.com/wireless/helpfiles/89600B/WebHelp/subsystems/gui/Content/MeasSetup_ResBw_WindowTypes.htm`

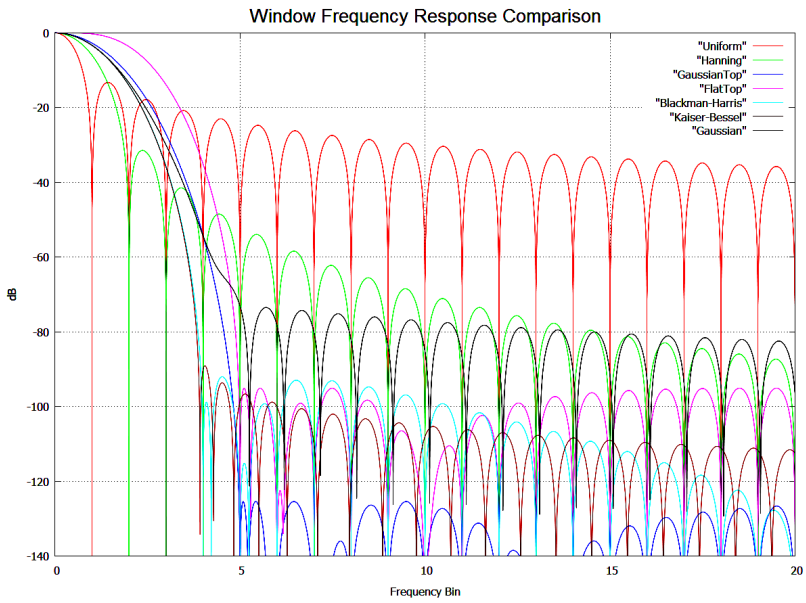


Figure 2.2: Window frequency response[1]

2.2.2 Constant Overlap Add

A major problem when working with short-time Fourier transfrom (STFT) is the weighting of the time signal caused by the window function. Signals with lengths comparable to the window length are suppressed by windowing and almost invisible in the spectral estimate.

The positive effects are easily shown by the following example:

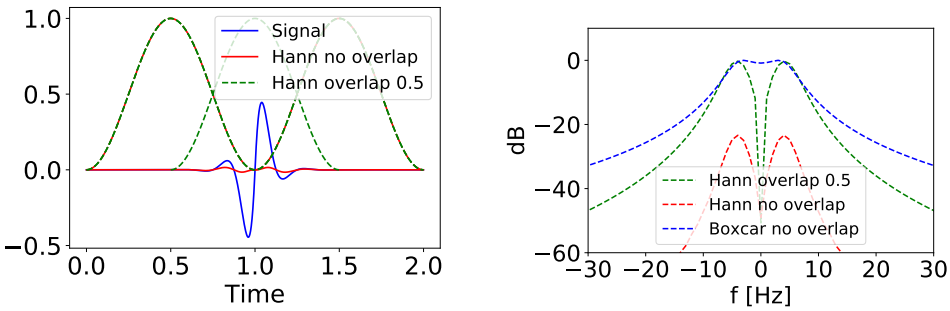
$$f_s = 1024 \text{ s}^{-1} \tag{2.3}$$

$$T = 2 \text{ s} \tag{2.4}$$

$$x(t) = \begin{cases} 0, & t <= 0.5, t > 1.5 \\ \sin(2\pi \cdot 4t) \cdot \exp\left(-\frac{|t-1|^2}{0.25^2}\right) & 0.5 < t <= 1.5 \end{cases} \tag{2.5}$$

When calculating the PSD estimate using Welch's method, non-overlapping windows can have devastating effects. Figure 2.3 shows the signal before and after applying the window function. The time signal after windowing without overlap is much lower, therefore, the 25 dB difference in the frequency domain is not surprising.

The appropriate amount of overlap depends on the window function. When chosen incorrectly some points are either over-counted or under-counted, thus the calculated spectral density heavily varies when the signal is shifted in time. This is a common problem when working with the STFT, especially when a perfect reconstruction is required.



(a) Time

(b) Spectrum

Figure 2.3: Advantages of overlapping windows

$$\sum_l w[n - LH]v[n - lH] = \text{const.}, n \in \{1, 2, ...N\} \tag{2.6}$$

Equation (2.6) is called OLA-constraint. For a given hop size H, if the product of the analysis window w and the synthesis window v sum up to a constant, perfect reconstruction is possible.

An important special case is the so-called Constant-OLA (COLA). In this case $v[n] = 1, n \in \{1, 2, ...N\}$, thus the overlapping have to add to a constant. The required overlap to fulfill the COLA-constraint depends on the window. In the following table, some of the popular windows and the minimal required overlap are listed [9]

Window	Overlap	Scallop Loss [dB]
Rectangular	0	3.92
Bartlett	1/2	-
Hann	1/2	-
Hamming	1/2	1.78
Blackman	2/3	-
Blackman-Harris	3/4	0.83
FlatTop	4/5	-

Table 2.1: Common windows with required overlap for COLA-constraint

Using overlapping windows is mostly beneficial when working with deterministic signals. When the measurement is supposed to adequately describe random processes, non-overlapping windows should be preferred.[10]

2.2.3 Normalization

When reading values from a PSD plot it is important to know what normalization has been applied. The power spectral density calculated according to equation (2.2) are normalized for reading noise values. That means that independent of the number of points of the FFT and independent of the chosen window the noise level stays the same.

Calculating the power of a signal can be done by using equation ((2.7)). This is the preferred normalization for comparing deterministic signals. Windowing not only causes a noise gain (NG) and a coherence gain (CG) but also a scallop loss. Thus the displayed signal amplitude varies depending on its position inside the frequency bin.

$$P_{\text{sig}} = P[i] \cdot \frac{NG \cdot f_s}{CG^2 \cdot N} \tag{2.7}$$

$$CG = \frac{1}{N} \sum_{i=0}^{N-1} w[i] \tag{2.8}$$

$$NG = \frac{1}{N} \sum_{i=0}^{N-1} w[i]^2 \tag{2.9}$$

Chapter 3

Device Characteristics

The device characteristics of the USRP B210 and the RTL-SDR Blog v3 have been explored near their supposed operating point in the SRD 860 band. In order to receive comparable results, the bandwidth was limited to 2.4 MHz. The reason is the RTL-SDR, which only works reliable up to this bandwidth. The USRP supports up to 56 MS/s, although this is only achievable when connected to a USB 3.0 port.

3.1 Timings

FlockLab provides highly time synchronized measurements, thus any extension to the system should also be tested on its accuracy. The time synchronization is not part of this project, yet proceeding with an ill-suited device would render the project useless.

3.1.1 Clock accuracy and stability

The clock accuracy has been measured with a tool called *kalibrate*. It uses the GSM FCCH signal as reference.

The RTL-SDR Blog v3 is a improved version of a DVB-T receiver with a clock error of less than 2 ppm and less than 1 ppm temperature drift. The available device shows an error of 287 Hz after directly plugging it in. Once it is heated up the error increases to 295 Hz. These are errors of 0.306 ppm and 0.314 ppm respectively for the GSM channel 26 at 940.2 MHz. The output can be found in appendix E.3.

The frequency error has been confirmed in a 24 h long observation. The test was conducted with a sample rate of 2.4 MHz, after every 2.4 million samples the system time of the Linux host system was printed to a file. The plot in Fig. 3.1 shows the timestamp minus the starting time minus the number of seconds that

were supposed to have passed, i.e. the number of samples divided by the sample rate.

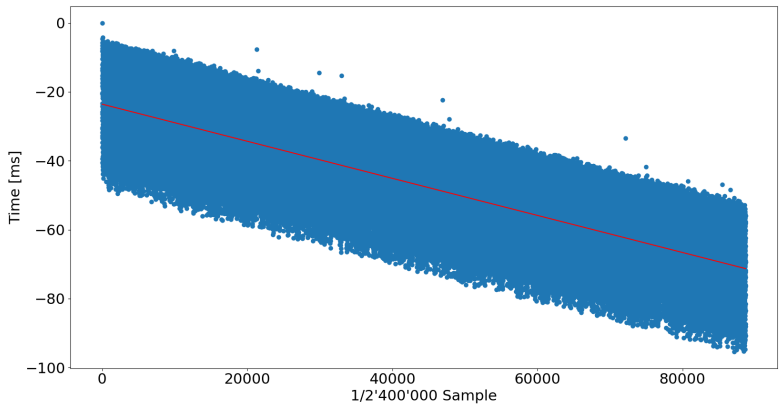


Figure 3.1: RTL-SDR: Difference between counted samples and the system time

In Fig. 3.1 it is clearly visible, that the reported error not only affects the tuning frequency, but also the sampling frequency. It also shows that the error is linearly growing with the number of samples, thus the error between two samples is constant. Aligning the sample based timestamps with the FlockLab timestamp is therefore trivial once the offset of one sample has been determined. The slope is -0.538 ms per 2.4 mega-samples.

The USRP B210 is equipped with the optionally available GPS-disciplined, temperature-controlled oscillator. Its frequency accuracy is 75 ppb and when GPS locked its pulse per second (PPS) aligned to UTC with ± 50 ns accuracy. The tool for the USRP had to be patched, such that the GPS adjusted clock was used (see appendix E.2). It started with an error of 11 Hz and later a difference of only 5 Hz was reported by the software. The same 24 hour setup was conducted with the USRP, as expected the slope is less steep than the one of the RTL-SDR. Also, Fig. 3.2 shows a much smaller variance, a possible reason are smaller buffer sizes in the USB driver. The fluctuations at the beginning are most likely caused by either adjusting the GPS-disciplined oscillator, or by synchronizing the system time with the local timeserver. Even though the GPS antenna was directly at the window there were two sections (orange bars) where the GPS receiver was not fully locked to the satellites.

GPS

The USRP has a board mounted GPS-disciplined, temperature-controlled oscillator. In unlocked condition the crystal oscillates with frequency accuracy of 75

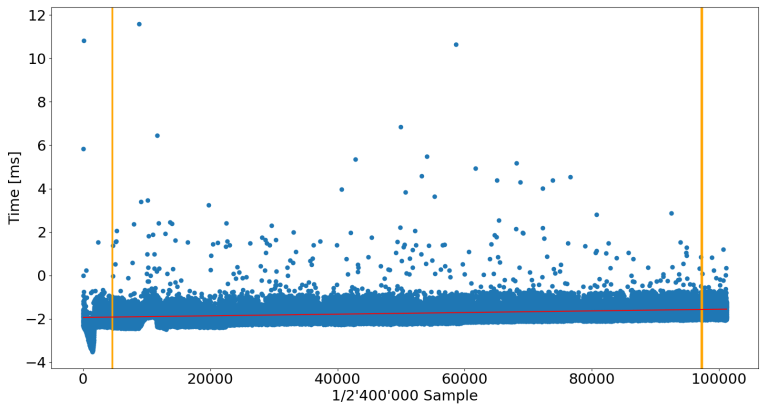


Figure 3.2: USRP: Difference between counted samples and the system time

ppm at 10 MHz. Once it is locked to enough GPS satellites it promises a 1 PPS accuracy of ±50 ns to UTC.

The increased accuracy, and the possibility to have an exact starting time are certainly in favor of the module, however the integration of the GPS module in the standard firmware is implemented rather poorly. The FPGA seems to be unable to interpret any of GPS string. The official way to set the device time to the GPS time is to retrieve the GPSDO time in the application and writing it back to the device. [11]

All communication with the module happens with the *get_mboard_sensor()* function which takes following strings table 3.1

gps_time	Returns the epoch time in seconds
gps_locked	Lock status (for some unknown reason differs from LED lock indicator)
gps_gprmc	NMEA Recommended minimum specific GPS/Transit data
gps_gpgga	NMEA GPS Fix Data
gps_servo	Summary of GPS status

Table 3.1: GPS commands

Apart from the gps_time command the gps_servo command appears to be the most useful. Strangely the only documentation is a response in the Ettus mailing list [12], hence the interpretation of the 9 space separated values is listed below:

- date
- PPS count
- fine DAC
- UTC offset in nano seconds

- Freq. error estimate in Hertz
- Nr. of visible satellites
- Nr. of triacked satellites
- Locked State
  - 0 OCXO heat-up
  - 1 Holdover (no lock)
  - 2 Locking (training)
  - 5 Holdover, oscillator still PLL'ed (GPS lost for less than 100s)
  - 6 Fully locked
- Health state (bit mask)
  - 1<<2 phase offset to UTC $> 0.25\,\mu$s
  - 1<<3 run time < 5 min
  - 1<<4 in holdover for > 1 min
  - 1<<5 freq. estimate implausible
  - 1<<8 ADEV at 100 s > 100 ns (drift > 1 ppb)

The USRP adds a `rx_time` tag to the first received sample, setting the time- and clock-source to 'gpsdo' is not sufficient.

Waiting for the device to achieve a full lock when freshly powered on is very time consuming. The GPSDO module has no memory, after every start the time is reset to January 2006. With the antenna placed directly at the window and 7 to 9 visible satellites, it takes roughly between one and two minutes to synchronize the time. This seems to be sufficient for the `gps_lock` command to return true, although the lock state indicates that the internal model is still being trained. The time to achieve a full lock under these conditions is close to ten minutes.

3.1.2 Frequency hopping

One limiting factor of the RTL-SDR is the small bandwidth. To extend the bandwidth with a single device, certain projects change the center frequency in short intervals, thus only observing parts of the spectrum for a given time. Some receivers were designed to support fast switching of the tuning frequency, and have a tune and lock time of less than one millisecond.[13]

The RTL-SDR has a large time lag when the changes concern tuner settings. According to D.Pfammatter et al. [14], retuning the dongle to a new center frequency takes 55 ms. Own measurements have shown that the tuning between two frequencies that are only 2 MHz apart is achieved 10 ms faster. For over 35 ms after sending the command the samples show no change, the changing and locking to the new frequency takes less then 10 ms. Fig. 3.3 shows the tuning process when changing the frequency from 866 MHz to 868 Mhz. A continuous source at 867 MHz was used to make the change visible.

The USRP supports a much broader bandwidth, thus its frequency tuning

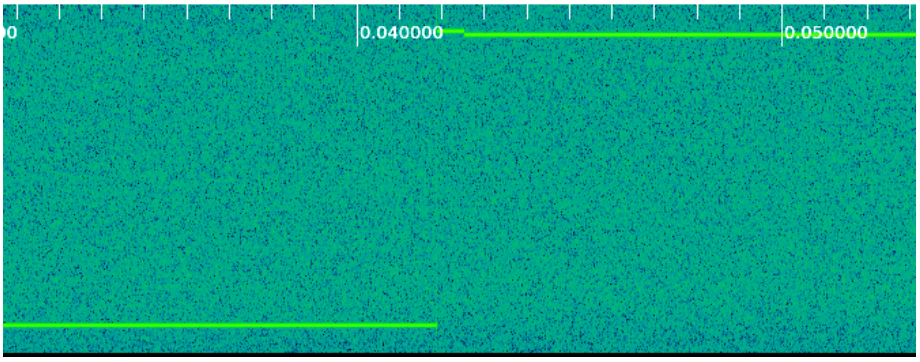


Figure 3.3: Retuning of the RTL-SDR. Waterfall diagram with time on x-axis.

capabilities were not investigated. However, times listed in [13] show that the B210 series with a retune time of 3 ms was not designed for fast hopping.

3.2 Distortions

Ideally the recorded spectrum would be completely flat. However, anti-aliasing filter and internally generated noise distort the spectrum. The frequency response of the USRP and the RTL-SDR have been measured using a BG7TBL noise source (version 2016-03-06).

3.2.1 USRP

The USRP has two stages of tuning. The RF front-end mixes the signal from the radio frequency down to an intermediate frequency, using the local oscillator (LO). The IF signal is sampled and converted to the baseband. The frequency shift is performed in DSP on the FPGA.

In Fig. 3.4 the benefits of using a LO frequency outside the observed bandwidth are clearly visible. With the LO set to the center frequency the USRP operates in the same manner as a direct conversion receiver. The hump at this frequency is typical for these kind of receivers. The result is an overall flatter passband.

Offset tuning depends on the sampling frequency of the ADC, if the offset is larger than half the sample rate it wraps around. The USRP B210 used in this project can sample with up to 61.44,MS/s, thus there are enough possibilities to place the LO frequency outside of the 2.4 MHz observed bandwidth. [15]

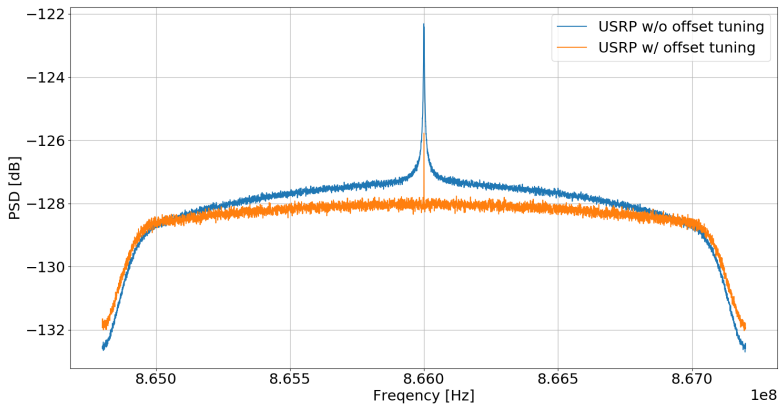


Figure 3.4: The frequency response of the USRP with and without LO offset

3.2.2 RTL-SDR

The frequency response of the RTL-SDR shows more fluctuation.

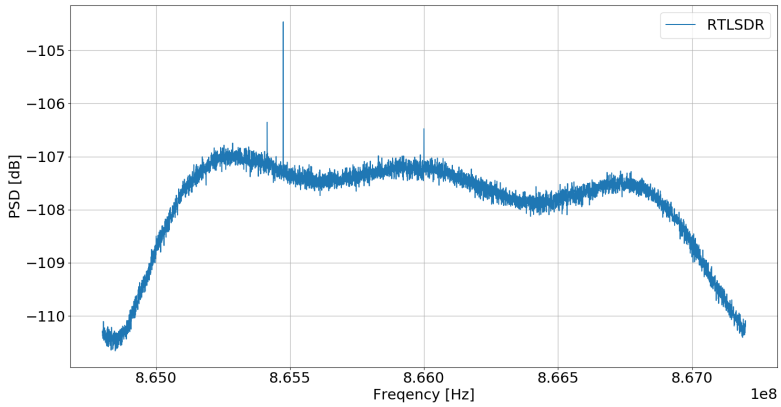


Figure 3.5: The frequency response of the RTL-SDR

Fig. 3.5 shows that in the passband a non symmetric ripple of almost 1 dB is to be expected. The passband at which the signal is attenuated less than 1 dB is only 1.86 MHz, whereas the USRP has 2.1 Mhz at the same sample frequency.

3.3 Gain

The gain of the devices has been measured and compared with the output of the spectrum analyzer Tektronix RSA306. The CW mode of the DPP2 Lora board was used as source the output power was −9 dBm and additional 70 dB attenuation in form of a 30 dB and a 40 dB in-line SMA attenuator were added.

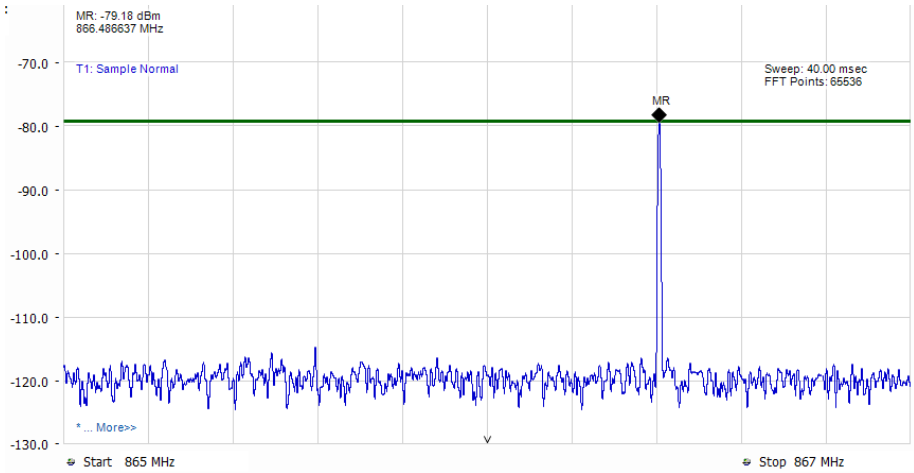


Figure 3.6: Tek Signal-Vu -9 dBm source with additional 70 dB attenuation

Fig. 3.6 shows that the output power and attenuation is pretty accurate. The peak power was −79.18 dBm.

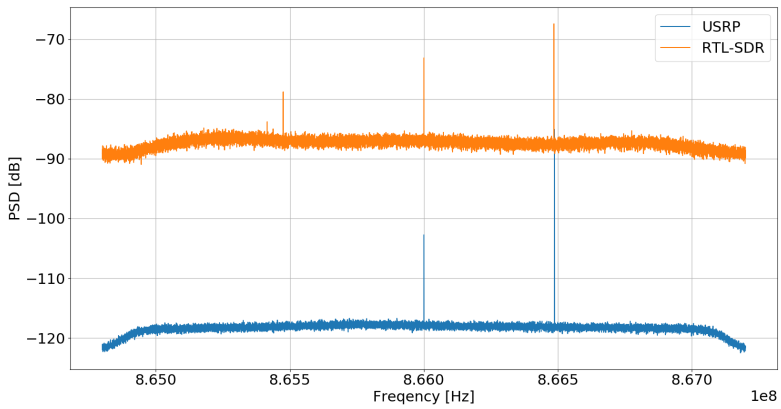


Figure 3.7: RTL-SDR and USRP set to 0 dB gain, -9 dBm source with additional 70 dB attenuation

The spectra recorded with the two SDR devices in Fig. 3.7 show that the RTL-SDR has a gain of 11.75 dB and the USRP has one of −5.93 dB. It also

shows that the signal to noise ratio is about 13 dB larger for the USRP, the higher internal sampling rate, 61.44 MS/s versus 28.8 MS/s, and the higher ADC resolution, 12 bit versus 8 bit are a huge advantage.

3.3.1 RTL-SDR

The name RTL-SDR describes all devices that use a Realtek RTL2832 DVB-T IC. When used as an SDR, its task is to sample the RF signal at an intermediate frequency, down-convert it and send the IQ samples over USB. The gain settings and the mixing of the signal is performed by a tuner. The two most common tuners are the Elonics E4000 (often called E4k) and the Rafael Micro R820T. Former is well documented, yet the RTL-SDR Blog v.3 dongle uses latter. Possible reasons are a tuning gap at between 1.1 and 1.4 GHz, or the fact that the E4k mixes the signal to a zero IF frequency, i.e. directly to the baseband making it more susceptible to IQ balance problems. The lack of documentation limited the usefulness of the tuner until someone figured out how to disable the automatic gain control. [16] As of today, some documentation has been leaked, though the manufacturer does not specify the gain and filter characteristics [17].

In the currently used drivers the VGA gain is fixed and only the LNA and the mixer gain are used. This choice of the VGA gain can be justified by interpreting Fig. 3.8. All gains have 16 gain steps (4 bit). Both lines have been plotted with a LNA and a mixer gain of 0x9. With a low VGA gain and without signal or

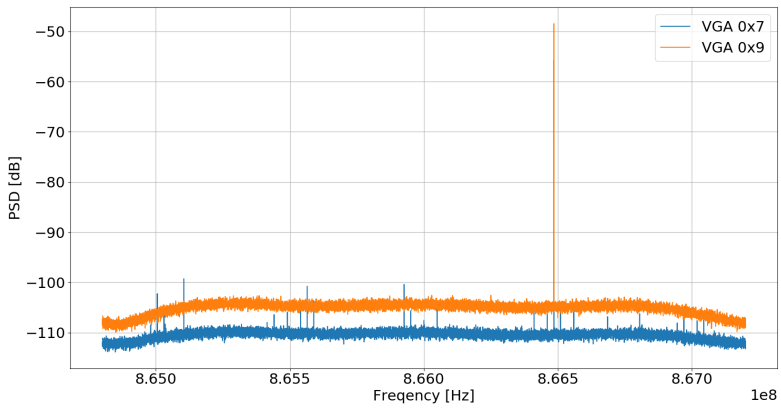


Figure 3.8: Signal at 866.5 MHz, “birdies” across the spectrum for VGA 0x7

only weak signals the quantization introduces spikes in the spectrum. The VGA gain is the last gain step, thus all the noise generated in the previous stages is amplified as well. In this case the amplified noise helps to suppress these phantom signals often referred to as birdies. The VGA gain of 0x8 as chosen by the author of the library, is a compromise between amplitude of phantom

signals and overall noise. Additionally, setting the VGA gain to about half the maximum, makes it more likely that the linearity of the amplifier is better, this reduces the likelihood of intermodulation.

The algorithm to choose the LNA and Mixer gain, however, is rather primitive. Steve Markgraf measured the amplitude of a sine wave generator for every given setting. The algorithm now increases the LNA and mixer gain step by step taking turns until the gain is at least as high as demanded. On first glance this is a perfectly reasonable method, however there are some details that have been overlooked. Firstly, the number of possible gains is extremely limited. Secondly, the mixer gain is not monotonically increasing, the highest configuration value has a lower gain than its predecessor.

For the project a new gain algorithm has been written which uses a look-up table. Additional reachable gain values have been introduced and it was ensured that the gain does not decrease when the its value is increased. The algorithm also works with the VGA gain fixed to 0x8. The source-code of the updated version is on GitHub[1]. If at some point someone needs to control the gains independently, there is a minor patch for gr-osmosdr allowing to control the gains directly from GNU Radio[2].

[1] `https://github.com/0pq76r/rtl-sdr`
[2] `https://github.com/0pq76r/gr-osmosdr`

Chapter 4

Implementation

The goal is to record activities in the radio frequency spectrum with a frequency resolution of about 10 kHz and aggregated in time to blocks of then to hundreds of milliseconds. At high data rates multiple different senders can be active in the same time block, thus, some processing steps are necessary before the data is reduced.

The implementation is base on GNU Radio 3.7[1]. The abstraction of the driver interface makes it easy to switch between the two SDR devices. It provides a lot of DSP blocks and can be extended with additional out-of-tree modules written in C++ or Python. It allocates and manages the sample buffers and the graphical companion application allows connecting the DSP block in a drag and drop fashion, similar to Matlab's Simulink.

The Python script generated by the GNU Radio Companion (GRC) is wrapped by another script making it easier to run the app with non-default parameters on headless setups. Its usage is described in the appendix D.1.

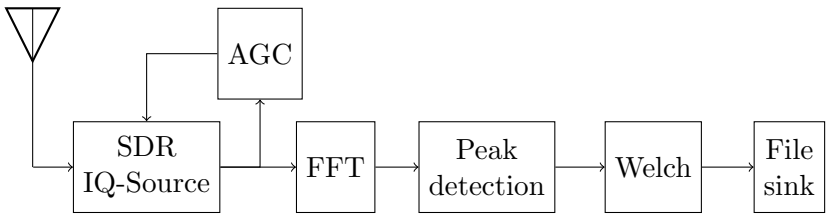


Figure 4.1: System overview

The processing pipeline is depicted in Fig. 4.1. The SDR delivers

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

4.1 gr-FlockLab

The GNU Radio library includes `gr_modtool`, a script that manages the creation of new modules and blocks. The gr-FlockLab module contains new blocks extending the capabilities of GNU Radio with the required tools necessary for this project.

4.1.1 AGC

Setting the gain is a major problem, it should be as high as possible without causing the signal to clip. The hardware AGC does not tell the instantaneous gain, thus it is disabled. The AGC calculates the power over some samples and calculates the gain, such that a pure sine wave with equivalent power would return an peak-to-peak amplitude of one.

A major problem of this implementation is the huge dead-time. For the RTL-SDR from processing until the gain was adjusted and the new samples traversed through all the buffers a time delay of 0.25 s is necessary.

There are three proposed usages of the AGC:

- Unknown signal strengths, strongest signal should not clip:
  The AGC only sets the gain when the difference to the calculated ideal gain is above a certain threshold. The thresholds for increasing and decreasing can be set independently. In this scenario setting the increase threshold to a very high gain (e.g. larger than half of the maximal gain), the decrease threshold to a few dB (e.g. 3 dB) and starting with maximum gain is recommended. The gain will be reduced as soon as signals starts to clip. The drawback is that one single strong pulse can reduce the gain for the whole duration of the experiment.
- Signals longer than dead-time:
  The AGC thresholds should never be too low because during the dead-time the gain could take arbitrary values. Five to ten dB threshold should provide a good trade-off between dynamic range and number of fluctuations.
- Strong signals immediately followed by short weak signals:
  In some cases, disabling the AGC can improve the observed spectrum. The clipping of the long signal causes phantom signals to pop up and the power estimate is too low. On the plus side the weak signals will be visible once the strong one has stopped.

MSG Variable is a variable that accepts a message and sets its value accordingly. GNU Radio introduced messages as a more efficient way to change parameters while the application is running. Not all blocks support this interface, they still rely on the Python callback calls. In order to use the value it has

to be wrapped in a `msg_variable(var_name)` function call. The AGC cannot set the gain variable directly thus this workaround was implemented.

4.1.2 FFT

There are different libraries with highly platform optimized algorithms available. The most famous is the FFTW library. It is freely available under GPL, however a license can be bought which allows closed source linking.

On the laptop, a ThinkPad with a 7th Generation Intel i7 Processors, the size is not limited by the computation power but rather by the amount of samples required compared to the symbol time.

The computation power of the Raspberry Pi is much lower thus it is necessary to limit the size to powers of two. The Raspberry Pi 3 has Cortex-A53 ARM cores which can be used in 32-bit or 64-bit mode. There are some distributions that make use of the 64-bit architecture, in experiments with single precision transformations no significant gain was measurable. The first version of the Raspberry Pi had a single core CPU without NEON (SIMD) instructions in order to compute FFT efficiently the GPU was used. From experiments with the library on the Raspberry Pi one can conclude that for single transformations the overhead from copying the data and setting up the GPU is too large. For sizes up to 2^{13} the FFTW3 library from the Raspbian repository outperforms the GPU implementation by a factor of roughly 1.8. The difference is even larger for transformations with fewer points. Details are in appendix E.1.

The default parameters for the FFT block are a size of 2048 and a Hann window with 0.5 overlap. The Hann window has been chosen because it has fewer leakage than the boxcar window, and it reaches the COLA-constraint already at 50% overlap. At first the number of bins of the FFT had been chosen higher, however, each bin requires a sample. At high data rates, the symbol or sweep times were lower than the time required to gather the required amount of samples. Thus, instead of the characteristic chirp, numerous narrow spikes appeared in the spectrum. An example is visible in Fig. 4.2.

Vector Shift is a block with the same task as `fftshift` in other DSP libraries. It shifts the zero-frequency component to the center. As the name suggest this is a more general implementation allowing an arbitrary shift.

4.1.3 Signal classification

Unfortunately there was not enough time to implement a working signal classification.

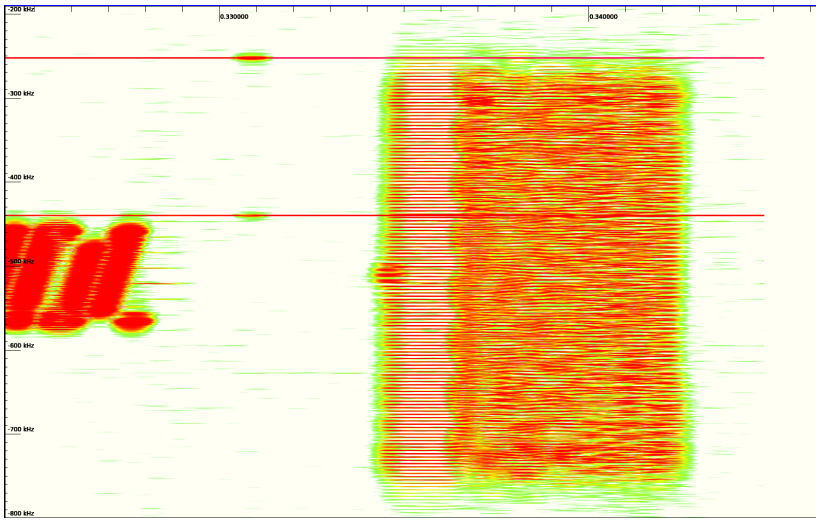


Figure 4.2: The LoRa signal on the right performs multiple sweeps in one FFT segment

Peak Tag was created to tag bins that exceed a certain threshold. The efficiency of the block is too poor, even on the laptop samples are dropped.

4.1.4 Stream Tags

Parallel to the sample buffer GNU Radio manages a second buffer where additional data can be passed with the sample. It is less efficient thus only a fraction of the samples should be tagged.

Saturation Tag this block adds a tag when the signal is above or below a given thresholds. Its intended use is to tag samples where the ADC is driven to its limit. When the signal is clipped the power estimate is wrong.

Time Tag this block adds the system time to each sample that is a multiple of the sample rate, i.e. every second a new tag is added. If the source is a USRP then every 10 seconds the response of `gps_servo` is added as tag

Variable Tag is a general purpose block that can be used to monitor one or multiple run-time variables. The GRC automatically registers the callback on all the variable setters. Once the callback is triggered, the next processed sample is tagged with the new value. In this project this method is used to track the gain variable.

4.1.5 PWelch

The name is somewhat misleading as it expects Fourier transformed vectors at its input. The vectors are processed in groups of the same size. The block has four outputs which return the average of the group, the element wise maximal and minimal value observed in the group, and the element wise variance. The values are normalized, such that the output is equivalent to Matlabs `pwelch` function.

Vector subsample is a block just inserted before the PWelch block. Its task is to reduce the vector size from the internal processing length to the requested output length. In its current state it can only divide the length by a factor of two. The reduction is based on averaging.

4.1.6 File Sink

The file sink for FlockLab is optimized to store the output of the PWelch block. The block can have multiple vector inputs with the same vector size. The file starts with a text segment, where the user can place arbitrary messages for documentation purposes. Immediately thereafter is a line in CSV format containing following information:

- Device: Either 'usrp_b210', 'rtl_sdr', or 'other'. the viewer (**??**) uses this information to subtract the intrinsic device gain.
- Type: The stream data type. This project only uses 'float32' as type.
- Vector size.
- Sample rate.
- Hop size: The overlap of the FFT expressed as number of samples between two transforms.
- Decimation: Usually the group size/avg length of the PWelch block.
- Center frequency.
- Number of ports.
- Window noise gain.
- Window coherence gain.
- Gain tag: name of the gain tag.
- Other tag names: Every tag name is a CSV element. Only tags listed here will be recorded by the files sink.

The header ends with a newline and the data block begins. The data is binary and of the type as described in the header and as long as a vector. This binary blob is followed by a CSV string starting with the port number, followed by an element for each tag defined in the header, starting with the gain tag. The tag value is always a string, if the tag is not assigned to the vector, the field is an empty string.

4.1.7 Debugging

For debugging the project contains a special pseudo variable `GDB`. Instead of a variable initialization the code generated by the GRC causes the process to print the process id and to pause, such that a debugger can connect.

4.2 FlockLab PSD Viewer

The PSD Viewer is a Python script that interprets and displays the files created by the gr-FlockLab File Sink (Sec. 4.1.6). The GUI is created with TkInter and the Matplotlib is used for creating the plots. Fig. 4.3 shows a screenshot of the viewer. The plots share the same x-axis and the y-axis of the upper plot controls the color gradient of the waterfall diagram. In the upper plot the instantaneous spectrum with all available variations are displayed, i.e. if the maximum output of the Welch block was not saved to the file it is simply ignored.

To improve the performance, the waterfall diagram only contains the immediately required data for the visualization. The file parser caches a limited number of samples, and the offsets at which a data entry is found. Latter is necessary since the unknown content of the stream tags can have arbitrary lengths.

Figure 4.3: PSD Viewer

Chapter 5

Results

Collecting and visualizing the power spectrum, works with both devices. The direct comparison of the RTL-SDR and the USRP in Fig. 5.1 confirm the limitations described in chapter 3.

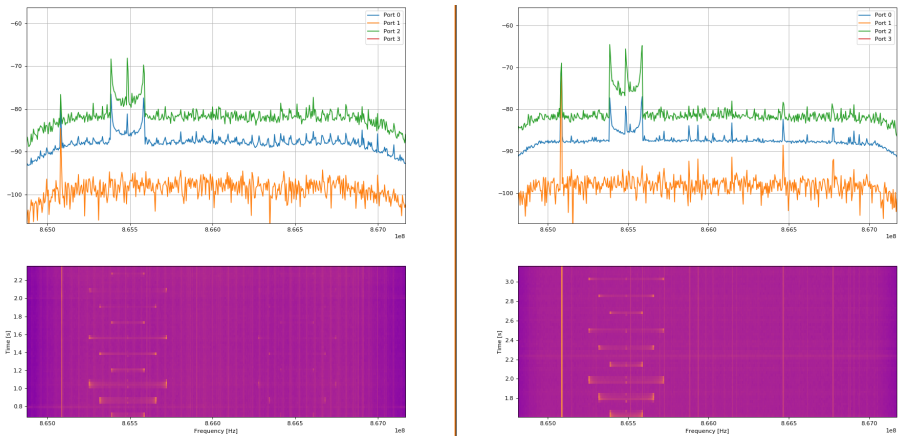


Figure 5.1: RTL-SDR left and USRP on the right

Fig. 5.1 was created by placing the two SDRs close together and recording the signal of a FlockLab DPP2 LoRa module on its $-9\,\mathrm{dBm}$ setting, with a $50\,\Omega$ SMA load connected instead of an antenna. That fact that the frequency plots show almost the same amplitude is pure coincidence. Both devices operate at their maximal gain, which has not been subtracted. After normalizing the gain, a difference of 12 dB would be visible. The explanation of the difference is most likely found in the different antennas and the slightly different positions.

Fig. 5.2 shows the same transmit sequence recorded at two different times. The peaks in the plot show a difference of 2.5 dB, Differences of $2-3\,\mathrm{dB}$ were observed in most blocks where signals with a length longer than the time block wee captured. For short signals, e.g. the pulse at the center frequency immediately before the FSK transmission start, it is possible that it lies in a different time block, thus causing differences of more than 5 dB.

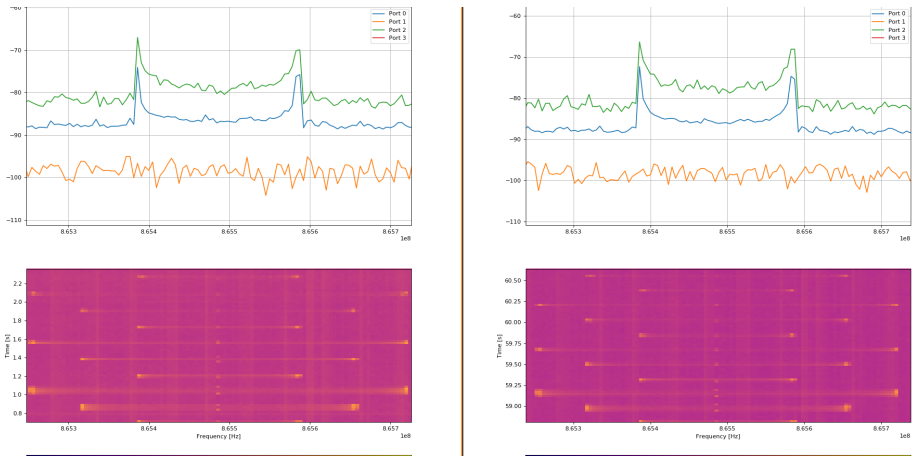


Figure 5.2: RTL-SDR repeated sequence

Chapter 6

Conclusions

The USRP clearly shows a better performance, the SNR is 5 to 10 dB higher and the delay when changing the gain is much lower. However the RTL-SDR works surprisingly well, if the reduced SNR still meets the requirements of the application, which I think it does. then the much lower price of the RLT-SDR speaks certainly in its favor.

Chapter 7

Future Work

Signal classification: Unfortunately there was not enough time to do a proper signal identification implementation. Some experiments with the GNU Radio gr-lora packet have been performed, although the module instructions have been followed and different spreading factors and bandwidths have been tested, no successful decoding was possible.

PSD Viewer: The Matplotlib does not provide the necessary optimizations for smooth scrolling through the waterfall diagram. Also, many features are still missing, for example the tags are only printed to the command line. Using the mouse to scroll is not only slow but also the scale of the other plot changes.

AGC: The current implementation is very primitive and has a very high delay. Moving the AGC implementation into the driver could reduce the delay.

Driver improvements: This concerns especially the librtlsdr. The tuner communicates over i^2c, even though it allows sequential writing of its registers the driver addresses them individually.

Time synchronisation: Before the integration of the system into FlockLab can proceed a proper time synchronization, as well as compensations for the clock error in long term measurements are required.

Bibliography

[1] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. pages 153–165. IEEE. ISBN 978-1-4503-1959-1. doi: 10.1145/2461381.2461402.

[2] Sreeraj Rajendran, Roberto Calvo-Palomino, Markus Fuchs, Bertold Van den Bergh, Hector Cordobes, Domenico Giustiniano, Sofie Pollin, and Vincent Lenders. Electrosense: Open and big spectrum data. 56:210–217. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1700200.

[3] Tevfik Yucek and Huseyin Arslan. A survey of spectrum sensing algorithms for cognitive radio applications. 11:116–130. ISSN 1553-877X. doi: 10.1109/SURV.2009.090109.

[4] Sungro Yoon, Li Erran Li, Soung Chang Liew, Romit Roy Choudhury, Injong Rhee, and Kun Tan. Quicksense: Fast and energy-efficient channel sensing for dynamic spectrum access networks. pages 2247–2255. IEEE. ISBN 978-1-4673-5946-7. doi: 10.1109/INFCOM.2013.6567028.

[5] Matthias Kuba. *Automatische Klassifikation von Kommunikationsstandardsim europäischen 868 MHz Short Range Device-Band.* PhD thesis, Universität Erlangen-Nürnberg, 2012.

[6] Jens Saalmüller, Matthias Kuba, and Andreas Oeder. A user-friendly android-based tool for 868 mhz rf traffic- and spectrum-analysis. volume embedded wolrd 2015. Fraunhofer Institute for Integrated Circuits IIS Nuremberg, 2015.

[7] Mariya Zheleva, Ranveer Chandra, Aakanksha Chowdhery, Ashish Kapoor, and Paul Garnett. Txminer: Identifying transmitters in real-world spectrum measurements. pages 94–105. IEEE. ISBN 978-1-4799-7452-8. doi: 10.1109/DySPAN.2015.7343893.

[8] P. Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. 15:70–73. ISSN 0018-9278. doi: 10.1109/TAU.1967.1161901.

[9] Hristo Zhivomirov. On the development of stft-analysis and istft-synthesis routines and their practical implementation. *TEM Journal*, 8(1):56–64, February 2019. ISSN 2217-8309.

[10] Hanspeter Schmid. How to use the fft and matlab's pwelch function for signal and noise simulations and measurements. Technical report, ime Institute of Microelectronics, August 2012.

[11] Ettus Research. Usrp hardware driver and usrp manual: Device synchronisation. `https://files.ettus.com/manual/page_sync.html`, . [Online; Version: 3.15.0.0-13-gb89f76bd4; accessed 27-June-2019].

[12] Marcus Müller. [USRP-users] gps_servo sensor. `http://lists.ettus.com/pipermail/usrp-users_lists.ettus.com/2016-December/051048.html`, December 2016. [Online; accessed 27-June-2019].

[13] Richard Bell. Maximum supported hopping rate measurements using the universal software radio peripheral software defined radio. In *Proceedings of the GNU Radio Conference*, volume 1, 2016.

[14] Damian Pfammatter, Domenico Giustiniano, and Vincent Lenders. A software-defined sensor architecture for large-scale wideband spectrum monitoring. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, IPSN '15, pages 71–82, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3475-4. doi: 10.1145/2737095.2737119. URL `http://doi.acm.org/10.1145/2737095.2737119`.

[15] Ettus Research. Usrp hardware driver and usrp manual: General application notes. `https://files.ettus.com/manual/page_general.html`, . [Online; Version: 3.15.0.0-13-gb89f76bd4; accessed 27-June-2019].

[16] 'superkuh'. Rtl-sdr and gnu radio with realtek rtl2832u [elonics e4000/raphael micro r820t] software defined radio receivers. `http://www.superkuh.com/rtlsdr.html`, June 2018. [Online; accessed 27-June-2019].

[17] *R820T High Performance Low Power Advanced Digital TV Silicon Tuner Datasheet*. Rafael Micro, November 2011.

Appendix A

Assignment

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Semester Thesis at the
Department of Information Technology and
Electrical Engineering

for

Florian Wernli

Frequency Spectrum Monitoring for FlockLab

Advisors: Jan Beutel
Roman Trüb
Reto Da Forno

Professor: Prof. Dr. Lothar Thiele

Handout Date:	11. 03. 2019
Official Start Date:	22. 03. 2019
Due Date:	28. 06. 2019
Initial Presentation (tentative):	04.04.2019
Final Presentation (tentative):	TBD

1 Project Description

Since 2012, the Computer Engineering Group (TEC) operates the FlockLab testbed [4] for developing and evaluating wireless sensor network protocols. A testbed helps to reduce the effort of repeatedly deploying test networks when developing protocols for wireless sensor networks. Furthermore, such a testbed improves the reproducibility of experiments and allows to share infrastructure.

The FlockLab testbed features different services such as GPIO and power tracing and GPIO actuation. Up to now it does however not provide a service for monitoring the frequency spectrum. This can provide additional information whic is important to verify the correct operation of the wireless communication protocol under test as well as to understand the environment (e.g. interference) during the tests.

The idea of this project is to use a software-defined radio (SDR), e.g. an rtl-sdr dongle or an USRP, to sense and characterize the spectrum while tests are running on the FlockLab testbed. Recording the frequency spectrum with an SDR generates a lot of data. The challenges are to process/aggregate the recorded data in order to obtain interesting metrics and to synchronize the timestamps of the collected data with the traces of the other FlockLab services. The aggregation could among other methods include the classification or even decoding of the sensed transmissions (e.g. with `gr-lora` [3]).

2 Project Tasks

- Formulate a time schedule and milestones for the project. Discuss and approve this time schedule with your supervisors.
- Perform a literature review to get an overview of related work (including [5, 6, 1]).
- Familiarize yourself with the SDR hardware and software tools [2].
- Compare the advantages/disadvantages of different SDR hardware (cheap rtl-sdr vs. USRP) and decide which one is used in the rest of the thesis.
- With a single transmitting device, collect frequency power spectrum traces, log them as csv and visualize them.
- Investigate the feasibility of logging multiple frequency channels at the same time.
- Investigate packet classification / decoding possibilities, e.g. using [2, 3], *or* investigate the time synchronization between the logged spectrum data and other data logged on the FlockLab observers.
- Evaluate the performance of the proposed system by logging the environment while running a FlockLab test with a known behavior.
- Document your project with a written report. As a guideline, your documentation should be as thorough to allow a follow-up project to build upon your work, understand your design decisions taken as well as recreate the experimental results.

3 Project Organization

Deliverables

- Time schedule (at the end of first 2 weeks)
- Initial Presentation (3 min)
- Final Presentation (15 min)
- Code of implementation including documentation
- Written report which includes: Introduction, Analysis of related work, Documentation of decisions, Evaluation, Description and HowTo guide of the developed software.

Offers

- The supervisors offer the student the opportunity to do a rehearsal of the initial and the final presentation. The supervisors offer to give feedback how to improve the presentations.
- The supervisors offer to proof-read a draft of the final report. The draft is not required to be complete. The draft should be handed in no later than 1 week before the deadline of the thesis.

General Requirements

- The project progress shall be regularly monitored using the time schedule. Unforeseen problems may require adjustments to the planned schedule. Discuss such issues openly and timely with your supervisor.
- Use the work environment and IT infrastructure provided with care. The general rules of ETH Zurich (BOT) apply. In case of problems, contact your supervisor.
- Discuss your work progress regularly with your supervisor. In addition to such meetings, a short weekly status email to your supervisors is required containing your current progress, problems encountered and next steps.

Handing In

- Hand in a single PDF file of your project report. In addition, hand in the signed declaration of originality on paper.
- Clean up your digital data in a clear and documented structure using the provided GitLab repository. In the end, all digital data should be contained in the student's GitLab repository for the thesis. This includes: developed software, measurements, presentations, final report, etc. An exception is

large amounts of measurement data which is stored separately (ask your supervisors!).

References

[1] ElectroSense. `https://electrosense.org/`.

[2] GNU Radio Companion. `https://wiki.gnuradio.org/index.php/GNURadioCompanion`.

[3] gr-lora. `https://github.com/rpp0/gr-lora`.

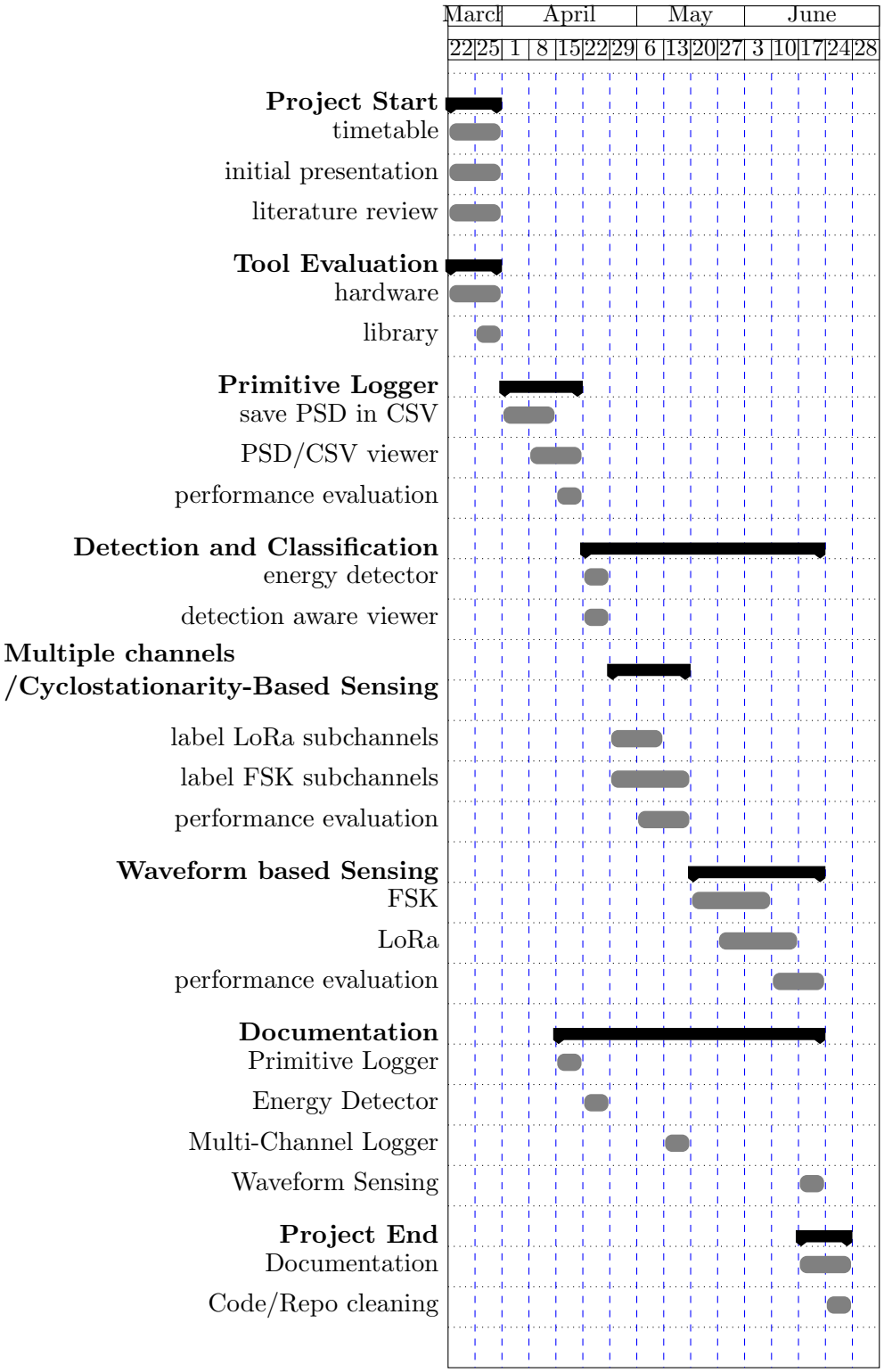
[4] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, IPSN '13, pages 153–166, New York, NY, USA, 2013. ACM.

[5] A. Nika, Z. Zhang, X. Zhou, B. Y. Zhao, and H. Zheng. Towards commoditized real-time spectrum monitoring. In *Proceedings of the 1st ACM workshop on Hot topics in wireless*, pages 25–30. ACM, 2014.

[6] D. Pfammatter, D. Giustiniano, and V. Lenders. A software-defined sensor architecture for large-scale wideband spectrum monitoring. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, IPSN '15, pages 71–82, New York, NY, USA, 2015. ACM.

Appendix B

Timetable


March
April
May
June
22 25 1 8 15 22 29 6 13 20 27 3 10 17 24 28
Project Start
timetable
initial presentation
literature review
Tool Evaluation
hardware
library
Primitive Logger
save PSD in CSV
PSD/CSV viewer
performance evaluation
Detection and Classification
energy detector
detection aware viewer
Multiple channels
/Cyclostationarity-Based Sensing
label LoRa subchannels
label FSK subchannels
performance evaluation
Waveform based Sensing
FSK
LoRa
performance evaluation
Documentation
Primitive Logger
Energy Detector
Multi-Channel Logger
Waveform Sensing
Project End
Documentation
Code/Repo cleaning

Appendix C

Setup

C.0.1 Notebook

Tested on `4.15.0-46-generic #49 16.04.1-Ubuntu`

USRP B210 : instructions `https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_Linux`.

Use mkdeb.sh script from tools to create a deb-packet. The package manager override files that are not part of a packet without warning. Building gr-zeromq test may fail when compiling the GNU Radio, this can be ignored.

C.0.2 Raspberry Pi

Tested on `raspberrypi 4.14.98-v7+`

USRP B210 : Allow the installation of packets for the next release version (buster).

```
$ cat <<EOF > /etc/apt/preferences.d/stretch.pref
Package: *
Pin: release n=stretch
Pin-Priority: 990
```

Software	Version
UHD	v3.13.1.0
gr	v3.7.13.4
libqwt-dev	v6.1.2-5

Table C.1: Software and version

Software	Version
UHD	v3.13.1
gr	v3.7.13.4
libqwt-dev	v6.1.2-5

Table C.2: Software and version

```
EOF
$ cat <<EOF > /etc/apt/preferences.d/buster.pref
Package: *
Pin: release n=buster
Pin-Priority: 750
EOF
$ echo deb http://mirrordirector.raspbian.org/raspbian/ stretch
↪  main contrib non-free rpi >
↪  /etc/apt/sources.list.d/stretch.list
$ echo deb http://mirrordirector.raspbian.org/raspbian/ buster
↪  main contrib non-free rpi >
↪  /etc/apt/sources.list.d/buster.list
$ sudo apt-get install uhd-host libuhd-dev -t buster
$ sudo apt-get install gnuradio -t buster
$ uhd_images_downloader # download firmware
$ sudo uhd_find_devices # send firmware to the device
```

RTL-SDR The gr-osmosdr packet with the RTL-SDR source for GNU Radio on swig:

```
sudo apt install swig cmake -t buster
```

C.1 Common

Edit the file /etc/ld.so.conf, insert `/lib/ /usr/lib/` at the top to prefer those paths. Ubuntu/Raspbian may installs ancient library versions into /usr/lib/..-linux-gnu/

RTL-SDR Install git and cmake and libusb1.0-dev :

```
sudo apt install git
sudo apt install cmake
sudo apt install libusb-1.0-0-dev
```

Install librtlsdr driver:

```
git clone https://github.com/0pq76r/rtl-sdr.git
~/mkdeb.sh rtl-sdr -DINSTALL_UDEV_RULES=ON
sudo dpkg -i rtl-sdr/build/rtl-sdr-0.6.0-2-gf68bb2f.deb
sudo ldconfig
```

Fix issue with udev rules by blacklisting driver/kernel module: Create file '/etc/modprobe.d/rtl-sdr.conf' with the following content:

```
blacklist dvb_usb_rtl28xxu
blacklist rtl2832
blacklist rtl2830
```

```
git clone git.osmocom.org/gr-osmosdr # latest stable (commit
↪  4d83c6067)
~/mkdeb.sh gr-osmosdr -DENABLE_UHD=ON
```

To build kalibrate these build tools are required:

```
sudo apt install autoconf automake libtool
```

Appendix D

Usage

D.1 PSD Recorder

Run

```
./flocklab_headless_recorder --help
```

for a list of all available options.

D.2 PSD Viewer

Run `./simple_viewer.py [file]` if no file is provided as argument a file open dialog shows up.

Appendix E

Experiments

E.1 FFT performance RPI 3

Source: `https://github.com/0pq76r/gpu_fftw`

FFT size 8192:

```
GPU FFTW 0.554983 times faster (1176.04 ffts/sec, 850.314
↪ usec/fft, fftw3: 2119.05 ffts/sec)
```

FFT size 1024:

```
GPU FFTW 0.328558 times faster (12653.5 ffts/sec, 79.0298
↪ usec/fft, fftw3: 38512 ffts/sec
```

E.2 Kalibrate USRP

```
Device: USRP B210 with GPSDO w/o GPS lock
Tool:   https://github.com/ttsou/kalibrate.git (ac0ace8)

Modifications:
================================================================
diff --git a/src/usrp_source.cc b/src/usrp_source.cc
index 4cb6a27..1800e04 100644
--- a/src/usrp_source.cc
+++ b/src/usrp_source.cc
@@ -166,6 +166,7 @@ int usrp_source::open(char *subdev) {

                m_dev->set_rx_rate(m_desired_sample_rate);
                m_sample_rate = m_dev->get_rx_rate();
+                m_dev->set_clock_source("gpsdo");

```

```
                if (m_external_ref)
                        m_dev->set_clock_source("external");


==================================================
COLD Directly after start:
==================================================

user@pc-10745$ ./src/kal -c 26 -g 40
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800;
↪  UHD_3.13.1.HEAD-0-gbbce3e45
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Detecting internal GPSDO....
[INFO] [GPS] Found an internal GPSDO: GPSTCXO , Firmware Rev
↪  0.929a
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Setting master clock rate selection to 'automatic'.
[INFO] [B200] Asking for clock rate 16.000000 MHz...
[INFO] [B200] Actually got clock rate 16.000000 MHz.
[INFO] [MULTI_USRP] Setting master clock rate selection to
↪  'manual'.
[INFO] [B200] Asking for clock rate 52.000000 MHz...
[INFO] [B200] Actually got clock rate 52.000000 MHz.
kal: Calculating clock frequency offset.
Using GSM-900 channel 26 (940.2MHz)
average    [min, max]  (range, stddev)
+  11Hz    [-28, 53]  (82, 21.810984)
overruns: 0
not found: 12

=========================================================
HOT some minutes after start:
=========================================================

user@pc-10745$ ./src/kal -c 26 -g 40

[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800;
↪  UHD_3.13.1.HEAD-0-gbbce3e45
```

```
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Detecting internal GPSDO....
[INFO] [GPS] Found an internal GPSDO: GPSTCXO , Firmware Rev
↪  0.929a
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Setting master clock rate selection to 'automatic'.
[INFO] [B200] Asking for clock rate 16.000000 MHz...
[INFO] [B200] Actually got clock rate 16.000000 MHz.
[INFO] [MULTI_USRP] Setting master clock rate selection to
↪  'manual'.
[INFO] [B200] Asking for clock rate 52.000000 MHz...
[INFO] [B200] Actually got clock rate 52.000000 MHz.
kal: Calculating clock frequency offset.
Using GSM-900 channel 26 (940.2MHz)
average    [min, max]  (range, stddev)
+   5Hz    [-25, 42]  (67, 18.135695)
overruns: 0
not found: 10
```

E.3 Kalibrate RTL-SDR

```
Device: RTL-SDR v.3
Tool:   https://github.com/steve-m/kalibrate-rtl (aae11c8)

===============================================
COLD Directly after start:
===============================================

sdr@raspberrypi:~ $ kal -g20 -c 26
Found 1 device(s):
  0:  Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 270833.002142 Hz
[R82XX] PLL not locked!
```

```
Setting gain: 20.0 dB
kal: Calculating clock frequency offset.
Using GSM-900 channel 26 (940.2MHz)
average         [min, max]      (range, stddev)
- 287Hz         [-321, -244]    (76, 18.524521)
overruns: 0
not found: 41
average absolute error: 0.306 ppm

=========================================================
HOT some minutes after start:
=========================================================
sdr@raspberrypi:~ $ kal -g20 -c 26
Found 1 device(s):
  0:  Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 270833.002142 Hz
[R82XX] PLL not locked!
Setting gain: 20.0 dB
kal: Calculating clock frequency offset.
Using GSM-900 channel 26 (940.2MHz)
average         [min, max]      (range, stddev)
- 295Hz         [-349, -246]    (102, 25.232929)
overruns: 0
not found: 102
average absolute error: 0.314 ppm
```