What's on the Wireless? Automating RF Signal Identification

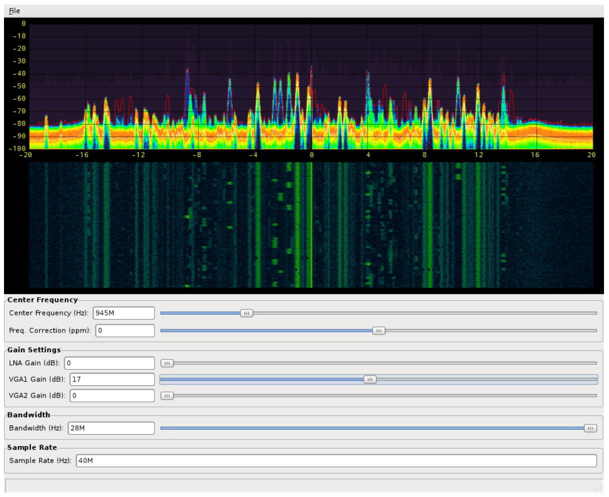
Michael Ossmann Dominic Spill
Great Scott Gadgets

Abstract

Software Defined Radio (SDR) equipment is well suited to support spectrum monitoring applications. We present new software that makes spectrum monitoring with SDR easier and more effective.

1 Introduction

Users of Software Defined Radio receivers tend to be familiar with software that presents a waterfall visualization of radio signals. A waterfall plot is a two-dimensional moving spectrogram, showing frequency on one axis and time on another with signal strength represented by color or brightness.



Center Frequency
Center Frequency (Hz): 945M
Freq. Correction (ppm): 0
Gain Settings
LNA Gain (dB): 0
VGA1 Gain (dB): 17
VGA2 Gain (dB): 0
Bandwidth
Bandwidth (Hz): 28M
Sample Rate
Sample Rate (Hz): 40M

Figure 1: gr-fosphor, waterfall visualization software for SDR

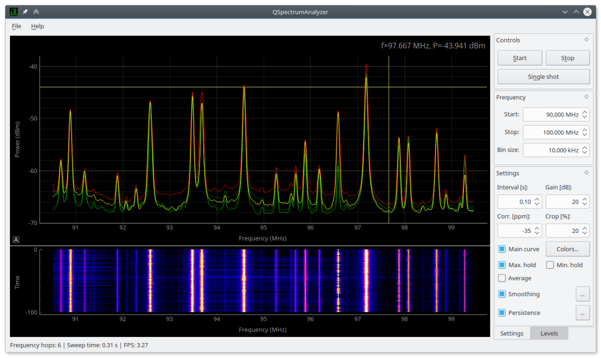
While useful for many applications, waterfalls suffer from limitations that curb their utility for spectrum monitoring. They present information about only a small time window, and they are also limited in the amount of RF bandwidth displayed. A typical waterfall implementation displays bandwidth equal to the sample rate of the SDR platform. For example, operation at 20 million samples per second yields a visible bandwidth of 20 MHz, only a small fraction of the tuning range of a general-purpose SDR platform.

SDR experts are often able to identify signal features including modulation with a glance at a waterfall plot, but this ability requires considerable knowledge and experience. Additional experience is needed to be able to distinguish atypical signals from those normally present in a given environment. For a spectrum monitoring solution to be usable by non-experts, it must have features supporting signal identification and annotation.

2 Spectrum Analysis

A traditional tool for spectrum monitoring is the spectrum analyzer, a test instrument that is more expensive than many of today's SDR platforms. Spectrum analyzers work by rapidly sweeping through a wide range of frequencies, measuring signal strength at one frequency (or over a narrow range of frequencies) at a time. The sweeping function of a spectrum analyzer allows it to analyze and visualize a very wide range of frequencies.

Because SDR platforms now feature tuning ranges comparable to spectrum analyzers, it is possible to implement spectrum analysis with SDR. Software such as rtl_power [3] implements a basic spectrum analysis function by measuring signal strength over a bandwidth less than or equal to the sample rate and then making such measurements successively across a range of tuning frequencies. The output of rtl_power can by post-processed for visualization by heatmap.py [2] or visualized in real time with QSpectrumAnalyzer [4].



f=97.667 MHz, P=-43.941 dBm

Start: 90.000 MHz
Stop: 100.000 MHz
Bin size: 10.000 kHz
Interval [s]: 0.10
Gain [dB]: 20
Corr. [ppm]: -35
Crop [%]: 20
Main curve
Max. hold
Min. hold
Average
Smoothing
Persistence

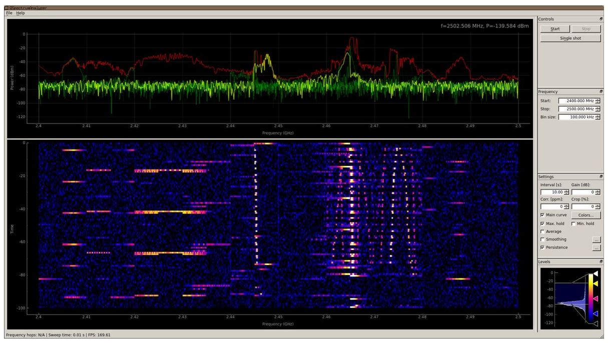
Frequency hops: 6 | Sweep time: 0.31 s | FPS: 3.27

Figure 2: QSpectrumAnalyzer using rtl_power to monitor the FM broadcast band

3 Rapid Sweeping

A significant drawback of SDR-based spectrum analysis software compared to special-purpose spectrum analyzers is the sweep rate achieved. The sweep rate of a spectrum analyzer can vary widely depending on configuration but is typically many sweeps per second. In contrast, the sweep rate of rtl_power over a wide tuning range is many seconds per sweep. The SDR device must be re-tuned many times per sweep, and each tuning requires time not only for the device to settle on a new frequency but also for the tuning command to be communicated (over USB in most cases) to the device by the software.

We have achieved a much faster sweep implementation in the open source HackRF [1] project by locating the frequency control in firmware of HackRF One, eliminating the need for the host computer to issue tuning commands over USB many times per sweep. This approach results in a sweep rate of 8 GHz per second (0.75 seconds per sweep across the entire 6 GHz tuning range), a much more useful rate for spectrum analysis. Our hackrf_sweep tool produces output comparable to rtl_power and is compatible with QSpectrumAnalyzer for real-time spectrum monitoring.



Start: 2400.000 MHz
Stop: 2500.000 MHz
Bin size: 100.000 kHz

Figure 3: QSpectrumAnalyzer using hackrf_sweep to monitor the 2.4GHz ISM band

In order to avoid problems with both the DC offset and the band-edge roll-off typically found in the output of SDR receivers, hackrf_sweep uses an interleaved tuning scheme. It operates at 20 million samples per second, capturing 20 MHz of bandwidth at each tuning step. Instead of analyzing the entire 20 MHz, however, hackrf_sweep only produces output for two 5 MHz bands, separated by a 5 MHz gap in the center of the 20 MHz bandwidth. The next tuning step analyzes another pair of 5 MHz bands interleaved with those of the previous step. In this way, 20 MHz of bandwidth is analyzed with two interleaved steps instead of a single step.

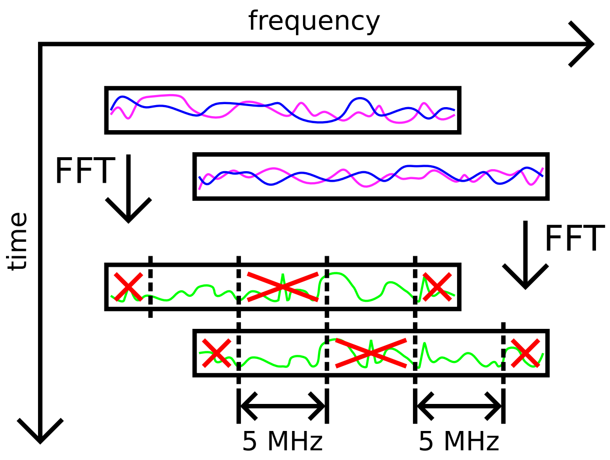


Figure 4: hackrf_sweep's interleaved tuning scheme captures time domain samples at two different center frequencies, converts them to the frequency domain with the Fast Fourier Transform (FFT), and then discards portions of the output of each FFT.

4 Integration with Existing Software

hackrf_sweep features a novel Inverse Fast Fourier Transform (IFFT) mode that produces time domain output instead of frequency domain output, allowing it to integrate with existing software intended for analysis and visualization of time domain samples produced by SDR receivers. At each tuning step, one or more frequency bins are measured within each 5 MHz band. After each sweep, the bins from each step are concatenated together into a contiguous set of frequency domain samples spanning the entire sweep range. This set is processed with the IFFT, resulting in a simulated time domain signal at a sample rate equal to the sweep range.

For example, when configured to sweep across 6 GHz, hackrf_sweep's IFFT mode produces a short burst of simulated samples at 6 billion samples per second every 0.75 seconds (the sweep period). The number of samples in each burst is related to the bin width configuration; when configured to the default 1 MHz bin width, 6000 samples are produced in each burst.

This technique makes it possible to integrate hackrf_sweep with existing, unmodified software such as gr-fosphor [5] or inspectrum [7] for spectrum monitoring applications. However, it is important for the user to recognize the difference between the simulated time domain signal produced by hackrf_sweep and an actual high sample rate signal. The IFFT output causes events that occurred at different times across a single sweep to appear to have occurred at the same time. It is also sparse in the time domain, skipping a significant amount of time between each burst of samples.

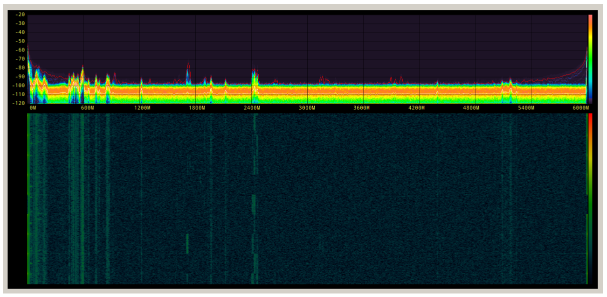


Figure 5: hackrf_sweep monitoring 0 to 6 GHz with IFFT output visualized in real time with gr-fosphor

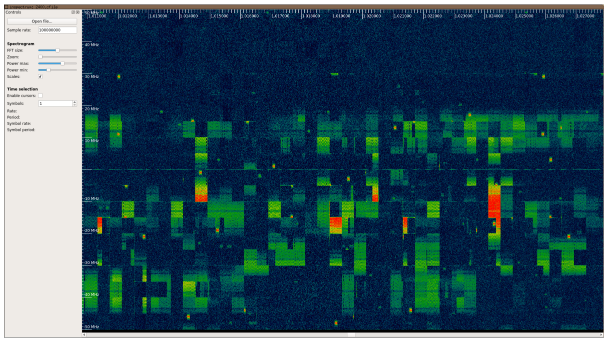


Figure 6: hackrf_sweep monitoring the 2.4 GHz band with IFFT output visualized by inspectrum

5 Integrated Waterfall and Sweep

When monitoring RF spectrum it is often useful to be able to more closely analyze a particular frequency of interest. To support this workflow we have extended ShinySDR [6], a web-based waterfall application, to enable switching back and forth between sweep mode and traditional waterfall mode. This allows a user to drill down and investigate a signal that was observed in sweep mode.

To further support spectrum monitoring, we have enhanced ShinySDR's annotation system, making it easier to import spectrum usage information from regulatory agencies or other external data sources. Additionally we support user annotation, allowing users to document spectrum usage in their own environments and easing the detection of anomalous activity.

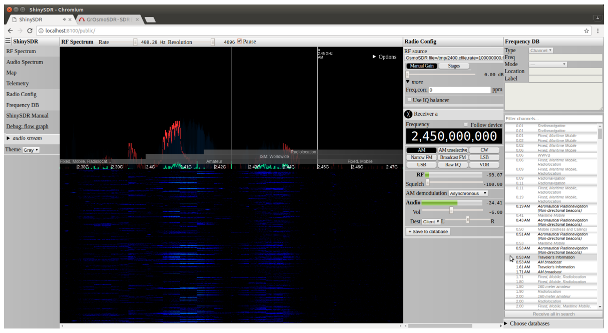


Figure 7: ShinySDR monitoring the 2.4 GHz band with IFFT output from hackrf_sweep

We are currently working on an additional modification to ShinySDR that enables Automatic Modulation Classification (AMC). This feature will further reduce the expertise required to identify radio signals with SDR.

6 Performance

While developing the hackrf_sweep functionality we faced trade-offs between sweep rate and data quality. We were able to make some of these choices user configurable, whereas others are fixed but could be improved with further work.

6.1 FFT Bin Width

The balance between FFT bin width (frequency resolution) and throughput is directly related to the processing power of the host system. By default we use a bin width of 1 MHz, which provides more horizontal resolution in a 6 GHz wide waterfall plot than most monitors can display. It is possible to reduce this to a 4 kHz bin width for even greater resolution. However, modern laptops struggle to process this in real time.

6.2 Interleaved Receive

The interleaved receive method described in section 3 was a decision between data quality and sweep speed. As we receive two overlapping 20 MHz blocks to produce 20 MHz of FFT output, our sweep rate is half of our theoretical maximum rate. Additionally we have implemented a linear (non-interleaved) sweep mode which uses the whole of each 20 MHz block without throwing out the frequency bins at the top and bottom ends or the bins around the central DC spike.

There may be scenarios where the speed of linear mode is a worthwhile trade-off for the lower quality output data featuring DC spikes every 20 MHz, especially when dealing with radio signals that are wideband with short packet lengths or low duty cycles. For example, when monitoring Wi-Fi bands the majority of packets will be several MHz wide, so the DC spike and filter roll-off at the band edges will not prevent packets from being detected. However, while looking for unknown signals it is important to have the highest data quality possible to reduce ambiguity.

6.3 Tuning Delay

As we tune the HackRF One to a new frequency we drop around 820 μs of samples that are received while the microcontroller issues serial commands to the front-end frequency synthesizers and the synthesizers settle. It may be possible to reduce this delay in at least some cases, but we do not believe that it will drastically improve the sweep rate for most uses. For this reason we have elected to fix the delay to 820 μs which accommodates the worst case tuning time of HackRF One.

6.4 Antenna Choice

There are very few antennas which support 0 to 6 GHz, so we are greatly limited in our antenna options for wideband spectrum monitoring. To mitigate this problem we have designed an antenna switching module, codenamed “Opera Cake” that allows the HackRF One to change antennas in less time than is required to tune to a new frequency. We have written firmware to control the module that is aware of which antenna is appropriate for a given frequency band. This tool allows us to use an array of antennas and automatically switch to the most appropriate antenna while re-tuning. Opera Cake is able to switch quickly enough to keep up with hackrf_sweep.

7 Conclusion

With appropriate software, general-purpose SDR systems can rival or even exceed the usefulness of special-purpose spectrum analyzers for spectrum monitoring tasks. Our approach of implementing rapid frequency sweeping in firmware combined with enhancement of user interfaces such as ShinySDR demonstrates the utility of SDR for spectrum monitoring.

8 Acknowledgments

We thank Mike Walters, Kevin Reid, Ellie Puls, Jacob Graves, and Michal Krenek for software contributions to this project.

References

[1] GREAT SCOTT GADGETS. HackRF. http://greatscottgadgets.com/hackrf/.

[2] KYLE KEEN. Rtl Power: Basic scripting. http://kmkeen.com/rtl-power/.

[3] MARKGRAF, S., ET AL. rtl-sdr. https://osmocom.org/projects/sdr/wiki/rtl-sdr.

[4] MICHAL KRENEK. qspectrumanalyzer. https://github.com/xmikos/qspectrumanalyzer.

[5] MUNAUT, S. gr-fosphor. https://osmocom.org/projects/sdr/wiki/Fosphor.

[6] REID, K. ShinySDR. https://kpreid.github.io/shinysdr/.

[7] WALTERS, M. inspectrum. https://github.com/miek/inspectrum.