rtl-sdr and GNU Radio w/Realtek RTL2832U, E4000 and R820T

Originally meant for television reception and streaming the discovery and exploitation of the separate raw mode used in FM reception was perhaps first noticed by Eric Fry in March of 2010 and then expanded upon by Antti Palosaari in Feb 2012 who found that these devices can output unsigned 8bit I/Q samples at high rates.

The drivers and userspace tools that made rtlsdr what it is today were created by the osmocom people,

- librtlsdr - contains the major part of the driver
- rtl_sdr, rtl_tcp, rtl_test, etc - command line capture tools and utilities
- gr-osmosdr - gnuradio compatible module
- and a bunch of other stuff.

keenerd is the author of many other rtl_* tools: rtl_fm, rtl_power (heatmap.py), rtl_adsb and code changes accepted into the mainline.

patchvonbraun is the author and maintainer of the build-gnuradio script that made it easy for me, and multitudes of others, to get started with rtlsdr under GNU Radio.

rtl-sdr.com has the latest news and tutorials.

RF, DSP, and USB details

The dongles with an E4000 tuner can range between 54-2147 MHz (in my experience) with a gap over 1100-1250 MHz in general. The R820T and R820T2 go from 24-1760 MHz (but with reduced performance above 1500 MHz). The R820T dongles use a 3.57 MHz or 4.57 MHz intermediate frequency (IF) while the E4000s use a IQ pair Zero-IF. For both kinds the tuner error is ~30 +-20 PPM, relatively stable once warmed up, and stable from day to day for a given dongle. All of the generic dongle antenna inputs are 75 Ohm impedance (some SDR branded versions have 50 ohm input). The RTL2832 ADC differential input impedance is ~3,300 Ohm. The dynamic range for most dongles is around 45 dB. The sensitivity is somewhere around -110 dBm typically. The highest safe sample rate is 2.56 MS/s but in some situations up to 3.2 MS/s works without USB dropping samples (RTL2832U drops them internally). Because the devices use complex sampling (I/Q) the sample rate is equal to the bandwidth instead of just half of it. For the data transfer mode USB 2 is required, 1.1 won't work. Antti Palosaari's measurements show the R820T use ~300mA of 5v USB power while the E4000 devices use only ~170mA. You can cut the leads to the LED to drop usage ~10%.

The rtlsdr RTL2832U chips use a phased locked loop based synthesizer to produce the local oscillator required by the quadrature mixer. The quadrature mixer produces a complex-baseband output where the signal spans from -bandwidth/2 to +bandwidth/2 and bandwidth is the analog bandwidth of the mixer output stages. (Datasheets, general refs: But what is the Fourier Transform? A visual introduction by 3Blue1Brown, Quadrature Signals: Complex, But Not Complicated by Richard Lyons) This is complex-sampled (I and Q) by the ADC. The Sigma-Delta ADC samples at some high rate but low precision. From this a 28.8 Msps stream at 8 bits is produced. That can be resampled inside the RTL2832U to present whatever sample rate is desired to the host PC. This resampled output can be up to 3.2 MS/s but 2.56 MS/s is the max recommended to avoid losing samples. The minimum resampled output is 0.5 MS/s. Check this reddit thread for caveats and details. The actual output is interleaved; so one byte I, then one byte Q with no header or metadata (timestamps). The samples themselves are unsigned and you subtract 127 from them to get their actual {-127,+127} value. You'll almost certainly notice a stable spike around DC. It's from either the 1/f noise of the electronics or if it's a Zero-IF tuner (E4000) the LO beating with itself in the mixer.

Popular software

My favorite way to explore the spectrum is using rtl_power to do very wideband multi-day surveys. For general use SDR# is probably the best application for windows with secondary mono-based linux and Mac support. I normally use Gqrx but it requires GNU Radio dependencies. Luckily there are Linux and OS X native binaries packages with all dependencies (ie, gnuradio) these days. For doing diagnostic and low signal level work Linrad is full featured and fast. osmocom_fft comes with GNU Radio module gr-osmosdr and is the natural and best way to use gr-fosphor; a GPU accelerated display. multimode has a very full and configurable GUI (it works great with GPU accelerated displays like gr-fosphor). For command line and low power devices try keenerd's rtl_fm.

These sites maintain the best list of rtlsdr device supporting applications: https://sdr.osmocom.org/trac/wiki/rtl-sdr#KnownApps, https://sdr.osmocom.org/trac/wiki/GrOsmoSDR#KnownApps, and lately http://www.rtl-sdr.com/big-list-rtl-sdr-supported-software/

Assuming you're on linux, but applicable in general, do not use the OS DVB drivers. Those are for the DVB-T mode and not the debug mode that outputs raw samples. Linux 3.x kernel should check with "$ lsmod | grep dvb_usb_rtl28xxu" and if found at least "$ sudo modprobe -r dvb_usb_rtl28xxu" to unload it.

While the sampling bandwidth is only 2.56 MHz the frequency can be re-tuned up to ~40 times a second. With frequency hopping you can survey very large bandwidths. See tholin's annotated 24 hour rtl_power spectrogram. Below is a zoomable 37200*31008 pixel 5 day long spectrogram I made using rtl_power's FFT mode and heatmap.py. It starts very far zoomed out. It might load a bit slow too. (view full window)

This page is mostly just notes to myself on how to use rtlsdr's core applications, 3rd party stuff using librtlsdr and wrappers for it, and lots on using the gr-osmosdr source in GNU Radio and GNU Radio Companion. This isn't a "blog", don't read it sequentially, just search for terms of interest or use the topics menu. For realtime support on the same topics try Freenode IRC's ##rtlsdr and reddit's r/rtlsdr.

These days for most people doing most things you want to get an dongle with an R820T2 tuner. They'll come with MCX coaxial connectors. On sites like eBay shipping from China the average price is about ~$10 shipped. These work fine for most things. At a bit higher price of ~$20 some come with improvements like SMA or F connectors, metal cases and heatsinks on the tuner for stability above 1500 MHz, temperature controlled crystal oscillators, extra breakouts on the PCB, and the like.

I bought two E4000 based rtlsdr usb dongles for $20 each in early 2012. Then many months later I bought two more R820T tuner based dongles for ~$11 each. There's photos of the E4ks up at the top of the page and of an R820T based dongle in the "mini" format off to the left (most minis do not have eeproms for device ID). It and the Newsky E4k dongle up top are MCX. Back in 2012 some of the cheaper dongles occasionally miss protection diodes but that is no longer an issue. The antenna connector on the E4k ezcap up top is IEC-169-2, Belling-Lee. I usually replace it with an F-connector or use a PAL Male to F-Connector Female. F to MCX for the other style dongles. The default design has the tuner taking 75 Ohm so that's what they all are except SMA.

Tuners

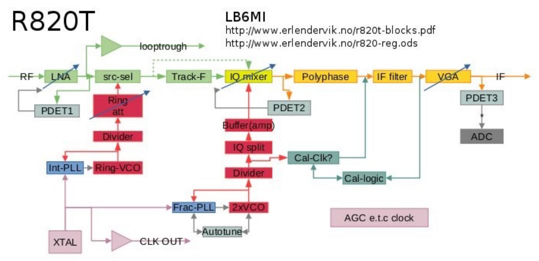
RTL-SDR Tuner Type	Frequency Range
Elonics E4000 (E4K)	54 - 2200 MHz (1100 MHz-1250 MHz gap)
Rafael Micro R820T	24 - 1766 MHz (>1500 MHz is bad w/o tuner cooling)
Rafael Micro R820T2	24 - 1766 MHz (>1500 MHz is bad w/o tuner cooling)
Rafael Micro R820T2	13 - 1864 MHz (mutability's driver)
Fitipower FC0013	22 - 1100 MHz (FC0013B/C, FC0013G has seperate L band input)
Fitipower FC0012	22 - 948 MHz
FCI FC2580	146 - 308 MHz and 438 - 924 MHz

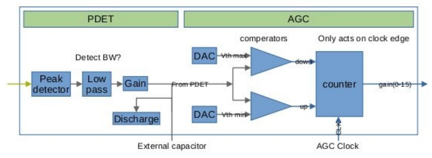
Only three tuners are very desirable at this time. The Elonics E4000 and the Raphael Micro R820T/R820T2. In general they are of equal performance but the sticks with R820T2 chips are easy to find, cheaper (~$10 USD), and they have a smaller DC spike due to the use of a non-zero intermediate frequency but must have cooling for the tuner to PLL lock above ~1500 MHz or so. The E4K is better for high end (>1.7GHz) while the R820T can tune down to 13 MHz without any hardware mods (mutability's driver). The tuners themselves are set up and retuned with I2C commands. E4000 tuners used to re-tune twice as fast as R820T tuners, but this was fixed in keenerd's experimental branch where R820T actually tune a tiny bit faster than the E4Ks. These changes were later adopted by the main rtlsdr.

For a detailed comparison of the E4000 vs R820T see the article, Some Measurements on E4000 and R820 Tuners: Image Rejection, Internal Signals, Sensitivity, Overload, 1 dB Compression, Intermodulation by HB9AJG in 2013.

Tuner registers

Here's a dump of all the leaked datasheets and research by rtlsdr community members on the E4000 and R820T tuners I have as of 2020.





R820T

- R820T_datasheet-Non_R-20111130_unlocked.pdf
- http://www.erlendervik.no/r820-reg.ods LB6MI (local)
- http://www.erlendervik.no/r820t-blocks.pdf (local)
- http://prkele.prk.tky.fi/~peltolt2/820T.xls (local)
- R820T2_Register_Description.pdf
- R820T_registers_attempt.xls
- gat3way / r820tweak
- SDRSharp.R820T-master.zip
- R820T2 Breakout Board

E4000

- Elonics-E4000-Low-Power-CMOS-Multi-Band-Tunner-Datasheet.pdf
- e4000_refsch_rev4.pdf

Re-tune speed

In the old days rtlsdr sticks re-tuned relatively slowly. As time passed re-tuning speed has been increased by clean-ups in code and specifically keenerd's changes so the tuner doesn't wait nearly as long for the pll to settle. More recently tejeez's mod made the re-tuning even faster by updating all the changed registers for a re-tune in one r82xx_write I2C call. With this done you can re-tune at rates of up to 41(!) hops per second; a ~2x improvement over then-existing drivers. Since then all of these re-tuning changes have been incorporated into the main rtlsdr.

Further massive speed-ups can be had at the cost of pretty much all reliability. By not waiting for PLL lock at all and always leaving the i2c repeater register enabled tejeez reports retuning speeds of up to 300 jumps per second are possible.

Tuning range

As of Aug. 2014 a handful of people have found ways to extend the r820t frequency range as well. Initially thought to top out at 1700 MHZ the R820T driver has has re-written to tune from 22 to 1870(!) MHz. While efforts have been made to extend the lower range as well, with the PLL seeming to lock down to 8 MHz in some cases, this range turns out to be full of images and repeats of the higher frequency range. A later effort with the addition of driver tweaks to the RTL2832 downconverter pushed the low end down to ~15 MHz. The code is at https://github.com/mutability/rtl-sdr/.

After tejeez worked out the no-mod HF reception a couple people have noted that the tuners with fc0013 receive HF even better than the R820T board designs. So if you have one of those laying around you might want to try HF with it.

Gain settings

The E4K has settings for LNA (-5..+25dB), mixer (4 or 12dB) and total of 6 IF gain stages with various gains allowing for 1dB steps between 3 and 57dB. The software only deals with LNA and mixer gain and not independently. IF gain can be set through the API.

R820T also has LNA, mixer and IF gain settings - the exact steps are not known. The numbers in the library code are through measuring the gain at a fixed frequency. That gave 0..33dB for the LNA, 0..16dB for the mixer and -4.7..40.8dB for the IF gain. The current library does not expose these settings through an API, only LNA and mixer are set through some algorithm. IF gain is set to a fixed value.

bofh__ gives more detail about the R820T step size,

The mixer gain step is 1dB (matches the empirical data passably, but not great) and the IF/VGA gain step is 3.5dB (matches mine basically dead-on). LNA gain step is not mentioned, all it says is "1111 - max, 0000 - min"

Frequency error

All of the dongles have significant frequency offsets from reality that can be measured and corrected at runtime. My ezcap with E4000 tuner has a frequency offset of about ~44 Khz or ~57 PPM from reality as determined by checking against a local 751 Mhz LTE cell using LTE Cell Scanner. Here's a plot of frequency offsets in PPM over a week. The major component of variation in time is ambient temperature. With the R820T tuner dongle after correctly for I have has a ~ -17 Khz offset at GSM frequencies or -35 ppm absolute after applying a 50 ppm initial error correction. When using kalibrate for this the initial frequency error is often too large and the FCCH peak might be outside the sampled 200 KHz bandwidth. This requires passing an initial ppm error parameter (from LTE scanner) -e . Another tool for checking frequency corrections is keenerd's version of rtl_test which uses (I think) ntp and system clock to estimate it rather than cell phone basestation broadcasts.

Also very cool is the MIT Haystack people switching to rtlsdr dongles (pdf) for their SRT and VSRT telescope designs, Use of DVB-T RTL2832U dongle with Rafael R820T tuner (pdf). The first of these characterizes the drift of the R820T clock and gain over time as well as a calibration routine.

As of 2015 there are a number of SDR-enthusiast targeting dongles produced with temperature controlled oscillators (TXCO) that run at less than 1 PPM with no start-up drift.

R820T2 variant

I recently (06-15-2014) found out from prog (of SDR# and airspy) that there are actually two different versions of the R820T tuner. The normal one and the R820T2. The T2 has different intermediate frequency filters allowing for wider IF bandwidths and apparently slightly better sensitivity (a few dB lower noise floor?). For rtlsdr dongles this difference in IF filter bandwidth usually doesn't matter much since all of them are larger than the RTL2832U's debug/SDR mode bandwidth of ~3 MHz. But there are certain situations where a larger tuner bandwidth is advantageous: such as when using Jowett's HF tuning mod. As of Sept. 2014 some of the new R820T2 have been showing up in Terratec "E4000 upgrade" model sticks. But don't count on it. I bought one from ebay seller "smallpartsbigdifference" which had a *photo showing an R820T2* and it was just an R820T. Since ~2015 R820T2 have become far more available. Here's a pdf with the R820T2 Register Descriptions.

As of 2017-12-11 Rafael Micro has been sending out emails saying the R820T2 has been discontinued. Alternate versions of the series, not pin compatible, are the R836, R840, or R828D. I've already seen SDR-targeted dongles using the R828D.

As of March 2018 it's uncertain how many non-defect bin R820T2 tuners Rafael Micro has left but rtlsdrblog has said that if very large bulk orders could be made Rafael Micro might be willing to produce more.

R820T/2 IF Filter Settings

In Feburary 2015 Leif sm5bsz (of linrad) relased a modified librtlsdr with changes to the rtlsdr R820T tuner code to allow for finer grained control over IF filter settings.

The IF filter which actually is a low pass filter and a high pass filter can be set for a bandwidth of 300 kHz. Dynamic range increases by something like 30 dB for the second next channel 400 kHz away. It is also possible to get some more improvement by changing the gain distribution.

Following this gat3way's patched gr-osmosdr and Vasili_ru's SDR# driver were released. gat3way made the IF filter width variable from within gqrx by presenting it as a gain value. Vasili's rtlsdr SDR# driver also moves the SDR# decimation normally applied during demodulation to the front of the IQ stream. This gives better dynamic range for the visual FFT but demodulated quality is not changed. So far this is all experimental but expect it to be brought mainline on both sides soon.

keenerd's experimental branch automatically set IF filter width based on sample rate but had not exposed them as manually set values.

TLeconte.github.io made some great plots of the R820T2 IF filter shapes in, "Playing with the Airspy R820T IF bandwidth". It should apply just the same for rtlsdr with changes.

R828D variant

In late 2013 Astrometra DVB-T2 dongles with the R828D tuner (pic) paired RTL2832U have begun to appear (2). The DVB-T2 stuff is done by a separate Panasonic chip on the same I2C bus. merbanan wrote a set of patches, rtl-astrometa, for librtlsdr has better support these tuners. The performance hasn't been characterized but it at least works for broadcast wide FM via SDR. steve|m's preliminary testing suggests bad performance in the form of the crystal for the DVB-T2 demodulator leaking fixed spurs 25 dB above noise floor in the IF at approximately 196 and -820 KHz. He was able to mitigate these with the hardware mod of removing the crystal for the DVB-T2 chip (ref). Official support was added to the rtl-sdr on Nov 5th while testing support was added on Nov 4th. In April 2017 u/strangerwithadvice on reddit made a quality post on the r/rtlsdr subreddit where he characterized the noise floor of an R828D dongle stock and with a number of modifications to reduce noise.

Double FC0013 tuner PCI DVB card

randomsdr reported on Freenode ##rtlsdr IRC on 2015-09-03 that the Leadtek Winfast DTV2000DS PLUS **pci card** has 2x FC0013 tuners and 2x rtl2832u chips like 2 normal rtlsdr dongles. Performance is not good but tools like rtl_fm work if the VID/PID is added to the rtlsdr driver table and udev rules set. It isn't recommended except as a novelty.

E4000 datasheet

2012-08-22: The E4000 tuner datasheet has been released into the wild. Elonics-E4000-Low-Power-CMOS-Multi-Band-Tunner-Datasheet.pdf, but...

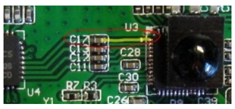
"All the ones that are documented in the DS are 'explained'in the driver header file ... And the rest, the datasheet call them "Ctrl2: Write 0x20 there" and no more details"

R820T original, support, etc

2012-09-07: Experimental support for dongles with the Rafael Micro R820T tuner that started appearing in May has been added to rtl-sdr source base by stevem. These tuners cover 24 MHz to 1766 MHz. They also don't have the DC spike caused by the I/Q imbalance since they use a different, non-zero, IF. On the other hand, they might have image aliasing due to being superheterodine receivers. See stevem's tuner comparisons. On 2012-09-20 the R820T datasheet was leaked to the ultra-cheap-sdr mailing list. The R820T2 Register Description pdf was provided by luigi tarenga to the ultra cheap sdr mailing list after he received it from RafaelMicro. The official range is 42-1002 Mhz with a 3.5 dB noise figure. On 2012-10-04 my order arrived. I'm liking this tuner very much since it actually works well, locking down to 24 Mhz or so *without* direct sampling mode. Here's a rough gnuplot spectral map of 24 to 1700 Mhz over 3 days I made with some custom perl and python scripts. Don't judge the r820t on the quality of that graph, it is just to show the range. You can see what I think is either front-end mixer filters not attenuating enough or actual intermodulation as RFI. I do almost no processing of the signal (ie, no IQ correction), don't clear the buffer between samples (LSB probably bad), and use a hacky way to display timeseries data in gluplot. Real SDR software like SDR# shows them to be equal or better in quality to E4ks.

stevem did gain measurement tests with a few dongles using some equipment he had to transmit a GSM FCCH peak, "which is a pure tone." This includes the E4000 and R820T tuners. In addition he measured the mixer, IF and LNA for the R820T.

High Frequency (0-30Mhz) Direct Sampling Mod



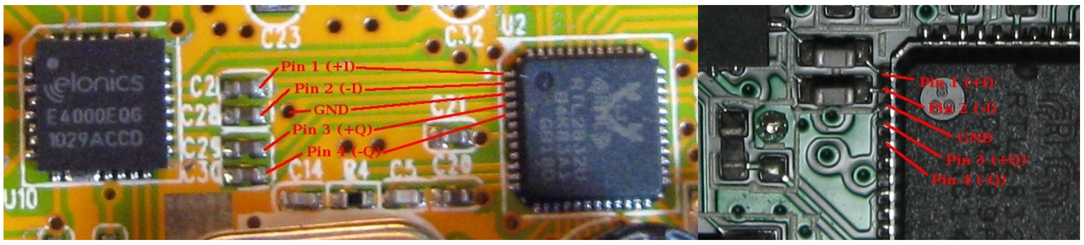
Steve Markgraf of Osmocom has created an experimental software and hardware modification to receive 0~30Mhz(*) by using the 28.8 MHz RTL2832U ADCs for RF sampling and aliasing to do the conversion. In practice you only get DC-14.4MHz in the first Nyquist zone but the upper could be had by using a 14.4 MHz to 28.8 MHz bandpass filter. In the stereotypical ezcap boards you can test this by connecting an appropriately long wire antenna to the right side of capacitor 17 (on EzTV668 1.1, at least) that goes to pin 1 of the RTL2832U. That's the one by the dot on the chip surface. Apparently even pressing a wet finger onto the capacitor can pick up strong AM stations. This bypasses static protection among other things so there's a chance of destroying your dongle. For gr-osmosdr the parameter direct_samp=1 or direct_samp=2 gives you the two I or two Q inputs.

No hardware change, software mod direct sampling

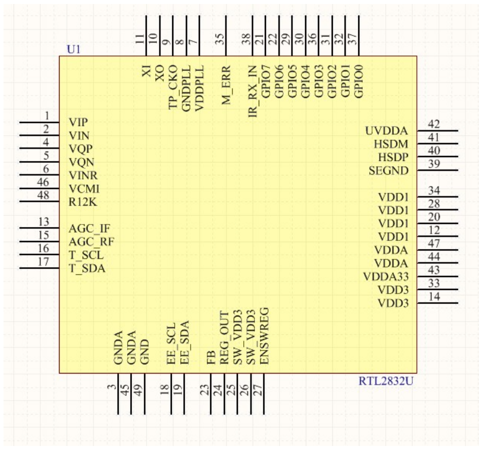
It has recently become possible to use direct sampling with no hardware modifications at all. It is still very experimental and performance is bad. In Oct 2012 Anonofish on the r/rtlsdr subreddit had discovered the PLL would lock for a small ~ 3686.6 MHz - 3730 MHz range far outside the normal tuning range and there seemed to be signals there. In January 2014 ##rtlsdr IRC channel user tejeez figured out this bypassed the tuner (mixer leakage) and implemented a set of register settings (R820T IF frequency, IF filter bandwidths, r82xx_write_reg_mask(priv, 0x12, val, 0x08) replaced with r82xx_write_reg_mask(priv, 0x12, val|0x10, 0x18)) that would exploit this to enable HF reception. Shortly thereafter keenerd assembled everything into a relatively easy to use patch-set.

If you want to give HF listening a try with no risk keenerd has added these changes rtl_fm and rtl_power in his experimental rtlsdr repository. To use the no mode mode with rtl_ tools append the argument, "-E no-mod". To use the no-mod direct sampling in something that uses gr-osmosdr, like gqrx or GRC flowgraphs, add the following to the the "device string" parameters: ie "direct_samp=3". Plug your HF antenna into the normal connector, no hardware mods needed.

Differential input



I've been told my pin numbering doesn't correspond to the datasheets, so take that with salt. The relative positions are correct regardless of the numbering. The RTL2832 ADC differential input impedance is ~3,300 Ohm.



A number of people have tried to match the ADC's input impedance and both differential inputs by using baluns of various sorts. The datasheet seemed to say 200 Ohm so a lot of people (myself included) tried 4:1 baluns which did improve performance. But better matches can be found using the 1800 Ohm (36:1) Mini-circuits T36-1-KK81 with a 3900 Ohm resistor in parallel with the secondary to bring the RTL impedance down to 1800 Ohm (ref: G8JNJ).

Dekar has a page showing how to use an ADSL transformer to generate signal for the ADCs differential input *using pin 1 (+I) and 2(-I)* on the RTL2832. mikig has a useful pdf schematic with part numbers for using wide band transformers or toroids for winding your own. Here's a series of posts from bh5ea20tb showing how to use a FT37-43 ferrite core. And another example from IW6OVD Fernando. PY4ZBZ as well. The ADC has a differential/balanced input so this is done mainly for the unbalanced->balanced conversion. But the ADC input pins also have a DC offset so you can't just connect one to GND for that.

Tom Berger (K1TRB) used multiple core materials with trifilar wire and performed tests using his N2PK virtual network analyzer on May 19th (2013).

Hams love type 43 ferrite, but for almost every application, there is a better choice. For broadband HF transformers Steward 35T is generally a better choice. Therefore, I wound a couple transformers and did the comparison. Type 43 and 35T Transformer Material Compared

For my tests with direct sampling mode I ordered a couple wideband transformers from coilcraft. The PWB-2-ALB and PWB-4-ALB to be specific. I sampled the PWB-4-ALB for free and ordered 4 of the PWB-2-ALB for ~$10 shipped. Both seem to work fine though I have no means of comparative testing.

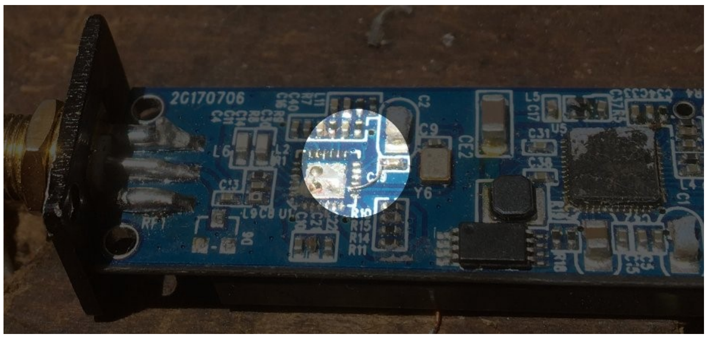
If you're particularly interested in HF work then an upconverter would be better than the HF mod. With the mod there will be aliases(*) for any frequency over 14.4 Mhz (1/2 the 28.8 clock rate). So you'd want a 14 MHz lowpass for the low end or a 14-28 MHz bandpass for the high end. And probably other little idiosyncracies. A lot of people chose to just use an upconverter instead. KF7LE wrote up short summaries comparing 16 popular upconverters.

Another alternative is to make a diplexer so that you get both HF via direct sampling and VHF+/etc without any switches. G8JNJ has a detailed guide with annotated photos on how to build the appropriate circuit and modify the latest R820T2 type dongles with it. He reports being able to receive from 15 KHz to 1.8 GHz with this mod.

Removing the tuner entirely

Since the tuner is not used in direct sampling it can be removed entirely (especially in a case where it is dead). Goatman contacted me on ##rtlsdr IRC about the process and pin jumpers needed to feed the rtl2832u the clock directly from the onboard oscillator in his case.

```
<Goatman> with a simple wire from pin 8 to 10 on the tuner pad
<Goatman> rtl_test reports all is well with direct sampling
<Goatman> ... Gqrx works fine btw with direct sampling even if no tuner is found
```



Using External Clocks and coherent sampling in general...

Multiple coherent dongles

The most exciting development in rtlsdr that has happened recently are Juha Vierinen's discuss-gnuradio mailing list and blog posts about a simple and inexpensive method to distribute the clock signal from one dongle to multiple others for coherent operation.

"I recently came up with a trivial hack to build a receiver with multiple coherent channels using the RTL dongles. I do this basically by unsoldering the quartz clock on the slave units and cable the clock from the master RTL dongle to the input of the buffer amplifier (Xtal_in) in the slave units (I've attached some pictures)."

Since I've seen a lot of people asking, the dongles he used were Newsky TV28T v2 w/R820T tuners.

- $16 dual-channel coherent digital receiver
- Passive radar with $16 dual coherent channel rtlsdr dongle receiver

Max Manning also implemented a passive radar system using clock sharing rtlsdr. Ben Silverwood also did so as, " Low cost RTL-SDR passive multistatic DAB radar." an implementation in matlab. The youtube video description has links to photos of the setup.

Also, there's a Japanese seller with high precision SMD 28.8 MHz crystals. And an ebay seller with high precision 28.8 MHz oscillators for around ~$30 shipped.

Things again became exciting in June of 2014. Going beyond simple clock sharing and it's max of 3 dongles, YO3IIU put up a great post his build of a 4+ dongle RTL2832u based coherent multichannel receiver using a CDCLVC1310-EVM dev board from TI for clock distribution. His post shows the results of a gnuradio block he coded that does all the correlation math to align the samples from each receiver (which are out of step due to the way USB works). Unfortunately the software was never released.

steve|m's experiments were the first I heard about back in 2011. He used his 13MHz cell-phone clock as a reference for a PLL to generate 28.8MHz. He said he used 1v peak to peak. He also related it was possible to not even use the PLL and just the 13 MHz clock if w/E4000 tuners if you don't care about sample rate offset.

```
<steve|m> not really, just a picture and a short clip: http://steve-m.de/projects/rtl-sdr/osmocom_clocksource.webm http://steve-m.de/pictures/rtlsdr_external_clock.jpg
<steve|m> a motorola C139
```

The Green Bay Public Packet Radio guys have written up an interesting article on using 14.4 MHz temperature controlled crystal oscillators sent through a passive (two diode) frequency doubler followed by crystal filters made out of the old rtlsdr clock crystals to provide a low PPM error clock for rtlsdr devices. Since their mirror was missing images I cut them out of the Zine pdf and made a mirror here.

I first heard about the GBPPR article from patchvonbraun who implemented one and performed tests which he posted about on the Society for Amateur Radio Astronomy list. It turns out that even with a good distributed clock the 2x R820t rtlsdr dongles still have large phase error for some reason, see: Phase-coherence experiments with RTLSDR dongles and the photo post: Progress towards using RTLSDR dongles for interferometry.

Alex Paha has also done clock distribution but unlike the others he used E4000 tuner based receivers for his dual coherent receiver. He also seems to be using only half the I/Q pairs. This post is in Russian.

Actually maintaining coherence over re-tunes and USB2 latency

rtl_coherent

In October 2015 teejez uploaded his rtl_coherent code for maintaining multi-dongle coherence using external antenna switches to disconnect the antennas and connect all to a common noise source for correlation calibration. Here's a video of him using it to make a 3 dongle direction finder.

Each dither-disabled rtl-sdr is fed from the same reference clock. They still have unknown phase shifts and sampling time differences relative to each other. This is calibrated by disconnecting them from antennas and connecting every receiver to the same noise source. Cross correlation of the noise gives their time and phase differences so that it can be corrected. Currently the signal is received and processed in short blocks with each block starting with a burst of calibration noise.

As I understand it the switch chips are sa630 that "look" for dongle i2c traffic. There are controlled by two RC delay circuits so that every time you change frequency (causing i2c traffic) it disconnects antennas, waits for some time, feeds a pulse (just one edge from the logic chip) into all dongles, waits a bit more and connects the antennas back. You can see the evolution of his setup from this earlier prototype to this later prototype and finally the version used in his direction finder.

Every time you tune any two (or more) dongles to a new frequency there *will* be a tiny difference in the frequency each actually tuned to. The offset must corrected before trying to correlate them. If you don't it'll look like there's a constantly varying phase shift. Also don't forget to let the dongles warm up to equilibrium otherwise this additional temperature related frequency shift will cause changes even larger than relative tuning offset and you'll get the "random" phase shift again.

Multi-RTL

As of 2016 Piotr Krysik's "Multi-RTL" (github) has made maintaining coherence of multiple dongles accessible even to the amateur. His GNU Radio block handles all the complex details of keeping multiple rtlsdr coherent even when they're tuned to different frequencies and over re-tunes. It requires no external circuitry. You just have to distribute the clock signal with cable.

PLL Dithering and you.

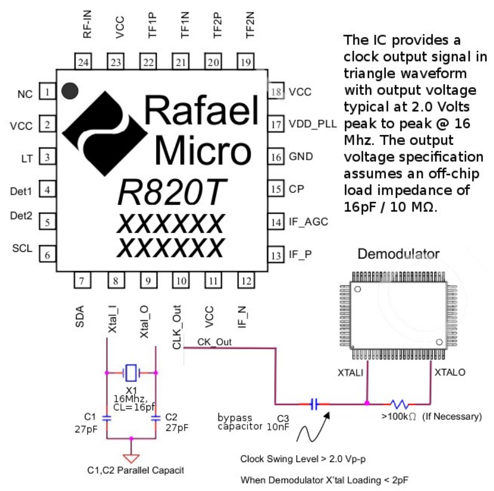
On the clock coherencey side Michele Bavaro's has explored, tweaked, and replaced, librtlsdr's pll setting code, intermediate frequency, and PLL dithering settings, such that the math, and results, work out cleaner. Using this modified driver he was able to minimize frequency setting errors and improve his GPS carrier following code. This is written up with code examples at his blog in, GNSS carrier phase, RTLSDR, and fractional PLLs (the necessary evil). Without dithering you can only tune to increments of 439.45 Hz. With dithering, you can tune to aproximately anything.

tejeez from the ##rtlsdr IRC relates that this can be done in r82xx_set_pll by changing r82xx_write_reg_mask(priv, 0x12, val, 0x08) to r82xx_write_reg_mask(priv, 0x12, val|0x10, 0x18). This has been implemented as an option in rtl_sdr, '-N', in keenerd's experimental branch.

Misc

In the absence of any useful information about the RTL2832U clock here's some information about the R820T's clock system.

Crystal parallel capacitors are recommended when a default crystal frequency of 16 MHz is implemented. Please contact Rafael Micro application engineering for crystal parallel capacitors using other crystal frequencies. For cost sensitive project, the R820T can share crystal with backend demodulators or baseband ICs to reduce component count. The recommended reference design for crystal loading capacitors and share crystal is shown as below.


RF-IN
VCC
TF1P
TF1N
TF2P
TF2N
NC
VCC
LT
Det1
Det2
SCL
VCC
VDD_PLL
GND
CP
IF_AGC
IF_P
SDA
Xtal_I
Xtal_O
CLK_Out
VCC
IF_N
Rafael Micro
R820T
XXXXXX
XXXXXX
The IC provides a clock output signal in triangle waveform with output voltage typical at 2.0 Volts peak to peak @ 16 Mhz. The output voltage specification assumes an off-chip load impedance of 16pF / 10 MΩ.
Demodulator
CK_Out
XTALI
XTALO
X1 16Mhz, CL= 16pf
C1 27pF
C2 27pF
bypass capacitor
C3 10nF
>100kΩ (If Necessary)
Clock Swing Level > 2.0 Vp-p
When Demodulator X'tal Loading < 2pF
C1,C2 Parallel Capacit

Noise, shielding, cables, and why is that FM signal there?!

When you see something weird, like commercial FM broadcasts at 27 MHz, what you are seeing incomplete filtering of mixing products. It's the harmonics of the square wave driving the mixers combined with insufficient rf filtering to suppress the response. You can tell if it is a local oscillator mixer harmonic leakage by sweeping the frequency and seeing how fast the ghost signal moves relative to this; look for linear relationships (ie, 2x the speed, 1/4 the speed, in-depth reference). Sometimes local signals can be powerful (ie, pagers) or close enough to make the preamplifier behave non-linearly resulting in intermodulation. For this kind of RFI turning down the gain helps.

The tuners all have a certain amount of intrinsic noise too. keenerd had done tests with an R820T rtlsdr terminated to a resistor inside of a metal box. For these tests rtl_power gain was set to max (49.6dB) and a frequency sweep was done through the entire tuner range, r820t Background Noise. The 28.8 MHz spikes from the clock frequency can be seen among other abberations.

But not everything is a ghost from hardware design problems. Depending on your computer setup and local electronics there could be a lot of "real noise"; LCD monitors are a common culprit for VHF noise spikes distributed across wide ranges. It is best to shield and put ferrites on everything if you can.

To solve the commercial FM mixing problems an FM trap can be used. Commercial ones work fine typically. But for non-commercial FM RFI like emergency services and pagers custom filters must be made or ordered. Adam-9A4QV has a detailed write-up on making FM trap with a very high upper passband (all the way to 1.7 GHz) with links to design for other low VHF bands. tejeez shared his VHF bandstop design on IRC. Like Adam's it has the unique feature of not also wiping out harmonics of the FM band: fm-notch.jpg fm-notch_schematic.png. This means you can use it and still do wideband frequency hopping (unlike, say, a 1/4th wave coaxial stub). For more information on this general type of coaxial cable notch filter check out Ed Loranger's write up on VHF Notch filters (photo). For my powerful 461 MHz RFI that can be received without an antenna I use a custom 3 cavity notch filter from Par Electronics.

Acinonyx describes one way to doing this using a single strip of aluminum tape combined with a spring to connect it to the dongle ground. Akos Czermann at the sdrformariners blog made a somewhat confusing but definitely empirical comparison of noise levels compared to different hardware mods like disconnecting the USB ground from the rtlsdr ground. Quite a few people have had success with that and scotch tape around the USB connector works to test it.

Some others bond the enclosure to both the antenna and the USB shield and this works reliably and well.

Martin from g8jnj.net finds the most effective mod to reduce USB and DC-converter noise is shielding the antenna input area with metal soldered to the pcb ground, "The noise seems to be coupled directly between components on the topside of the PCB." You can find it if you scroll about halfway down the page linked.

8.4. RTL2832U Internal Switching Regulator

The RTL2832U integrates a switching regulator with input voltage 3.3V to output voltage 1.2V. Figure 6 shows the application circuit.

The ENSWREG pin default power is 3.3V. Applying 0V turns off the switching regulator.

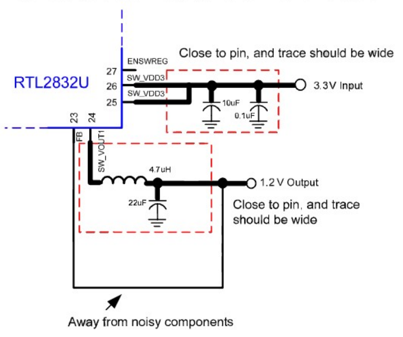

RTL2832U
27
26
25
23
24
ENSWREG
SW_VDD3
SW_VDD3
FB
SW_VOUT
Close to pin, and trace should be wide
10uF
0.1uF
3.3V Input
4.7uH
22uF
1.2 V Output
Close to pin, and trace should be wide
Away from noisy components

Figure 6. Internal Switching Regulator Layout

Additional noise comes from the switching power supply in the RTL2832U that runs at 1.024MHz. This drops the supplied 3.3v down to the 1.2v needed for internal use. ttrftech has successfully disconnected this switching supply replaced it with 3 diodes to drop the 5v line down to ~1.2v. In the example linked above ttrftech uses power form the far side of the board but the eeprom's power rail would also work. This decreases spurs in HF significantly. It will increase power usage though; something to watch out for when R820T dongles start out at ~300mA a piece.

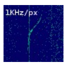
Laidukas's "Mods and performance of R820T2 based RTL SDR receiver covers replacing all the power rails with external linear regulators, increasing the amount of bypass capacitance on power lines, adding extra chip filtering for the USB 5v line, cutting off the IR receiver part of the PCB, wiring in a TCXO 28.8MHz oscillator, creating a shield with kapton tape and copper foil soldered extensively to the PCB ground, and a new heavy metal case and connectors.

To reduce signal loss over long distances and get away from computer RFI I like to run long USB *active* extension cable with hubs at the end and ferrites added instead of coaxial cable. Around this USB cable I clip on 5 or 6 ferrites at each end. Active extension/repeater USB2 cables of up to 25m in length can be used.

Antenna, but particularly broadband antenna

I've also written up a seperate, longer, page on the challenges and solutions when implementing broadband antenna.

When I want to do some scanning that takes advantage of the tuner's very wide ranges I use five types of antenna: discone, spiral, dual planar disks, vivaldi (tapered slot), and horns (TEM and pyramidal). Discone, dual planar disk, and archimedian spiral antenna can omnidirectionally cover almost the full range of the E4000 tuner but things get a bit too large to go all the way to the 24 Mhz of the R820T. You can refer to the seperate spiral antenna page for construction and technical details. To build my discone I followed Roklobsta's D.I.Y. Discone for RTLSDR (archive.org mirror). With just a discone and rtl_power it's possible to see lots of LEO satellite carrier frequencies doppler across the spectrum.



To get an idea of how much you can see with a discone here's a directory where I produce ~2 to 4 day long ~70 to 1000 MHz range 25KHz resolution 45k*10k pixel spectrograms. They each have a javascript zoomable interface to load small tiles progressively. An example. With just a discone and rtl_power it's possible to see lots of LEO satellite carrier frequencies doppler across the spectrum.

But with a band specific helix in a cone reflector (helicone) many more satellites can be picked up. The previous is a link to a zoomable spectrogram of ~2 days of the 1616-1626 MHz satellite band that Iridium satellites use. No LNA was used. There's plenty of RFI/EMI even through a 1 GHz high pass but the satellite doppler passes are clearly there in numbers if you zoom in far enough.

When using such broadband antenna, or even a band specific helix, it is possible to pick up powerful out of band signals due to overloading or incomplete mixer filtering. It's important to identify any extraordinarily powerful transmitters nearbye and filter them out. In my case I have a 50w transmitter at 461 MHz across the street *always* going full power. I bought a custom tuned 3 cavity notch filter from PAR Electronics. This limits the upper frequency range to 1GHz but does at least solve the RFI problem.

Usually the spectra are much cleaner when using directional and resonant antenna instead of wideband omnidirectionals. But many directional antenna like helix and log periodic dipoles have very large out of band sidebands on low frequencies not in the designed range.

Page Sections

RTL-SDR Links

- rtl-sdr wiki - osmocom
- Known applications - rtl-sdr wiki - osmocom
- rtlsdr.org wiki
- r/rtlsdr - reddit group
- Ultra Cheap SDR - google group
- RTL-SDR - yahoo group
- AB9IL's rtl2832 software defined radio overview

Warning: I'm learning as I go along. There are errors. Refer to the proper documentation and original sources first.

GNU Radio *and* RTL-SDR Setup

You don't need GNU Radio to use the rtlsdr dongles in sdr mode, but there *are* many useful apps that depend on it. patchvonbraun has made setting up and compiling GNU Radio and RTLSDR with all the right options very simple on Ubuntu and Fedora. It automates grabbing the latest of everything from git and compiling. It will also uninstall any packages providing GNU Radio already installed first. Simply run, http://www.sbrac.org/files/build-gnuradio, and it'll automate downloading and compiling of prequisites, libraries, correct git branches, udev settings, and more. I had no problems using Ubuntu 10.04, 12.04, or 14.04. These days (2015) pybombs is slowly taking over for build-gnuradio but for now this works best.

If you're thinking about trying this in a virtual machine: don't. If you do get it partially working it'll still suck.

As an aside: If you're an OSX user then you can use the MacPorts version of GNU Radio (including gqrx, etc) maintained by Michael Dickens.

```
mkdir gnuradio
cd gnuradio
wget http://www.sbrac.org/files/build-gnuradio
chmod a+x build-gnuradio
./build-gnuradio --verbose          # default is latest 3.7
*or*
./build-gnuradio -o --verbose       # install old 3.6.5.1
```

Install 3.7. Most gnu radio projects have been ported to it as default. Only a few old things will require 3.6.

An (re)install looks like this. It might be useful to save the log output for future reference. Then test it. The test output below is from a very old version of rtl_test with an E4K dongle. Newer versions, and R820T tuners will output slightly different text.

```
rtl_test -t
Found 1 device(s):
  0:  ezcap USB 2.0 DVB-T/DAB/FM dongle

Using device 0: ezcap USB 2.0 DVB-T/DAB/FM dongle
Found Elonics E4000 tuner
Benchmarking E4000 PLL...
[E4K] PLL not locked!
[E4K] PLL not locked!
[E4K] PLL not locked!
[E4K] PLL not locked!
E4K range: 52 to 2210 MHz
E4K L-band gap: 1106 to 1247 MHz
```

Once GNU Radio is installed the "Known Apps" list at the rtl-sdr wiki is a good place to start. Try running a third party receiver, a python file or start up GNU Radio Companion (gnuradio-companion) and load the GRC flowcharts. If you're having "Failed to open rtlsdr device #0" errors make sure something like /etc/udev/rules.d/15-rtl-sdr.rules exists and you've rebooted.

Updating

When updating you can just repeat the install instructions which is simple but long. The advantage to repeating the full process is mainly if there are major changes in the gr-osmosdr as well as rtl-sdr. It'll do things like ldconfig for you.

```
./build-gnuradio -e gnuradio_build
```

Just compile/installing rtl-sdr

If you don't have the patience for a full recompile and there haven't been major gnu radio or gr-osmosdr changes it's much faster just to recompile rtl-sdr by itself. The instructions to do so are at the osmosdr page. It'll only take a few minutes even on slow machines. Once you have the latest git clone it is like most cmake projects:

```
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr; mkdir build; cd build; cmake ../ ; make; sudo make install; sudo ldconfig
```

---

rtl-sdr supporting receivers, associated tools

- keenerd's rtl-sdr branch
  - This experimental branch contains a number of useful low processing power utilities, expansions of the original rtl tools, and improvements to the R820T driver re-tuning speed. A lot of them have already been merged into the librtlsdr master but rtl_fm and rtl_power fixes, features and bugs appear here first. rtl_fm is for scanning, listening, and decoding (and not just FM), rtl_adbs for plane watching with an external ads-b viewer, rtl_eeprom for checking and *setting* serial numbers and related data if your dongle has an eeprom. And as of 2013-09-20, rtl_power, a total power frequency scanner. These tools are very good for slow machines or when you want to do command line automation. Just build it like the osmocom rtlsdr page does for the vanilla install. Use these on the raspberry pi.

    ```
    git clone https://github.com/keenerd/rtl-sdr.git
    git clone https://github.com/keenerd/rtl-sdr-misc
    ```

    ```
    cd rtl-sdr; mkdir build; cd build; cmake ../ -DINSTALL_UDEV_RULES=ON; make; sudo make install-udev-rules; sudo ldconfig
    ```

    Most people use rtl_power for smaller total bandwidths (<200 MHz) and higher spectral resolution using the default FFT mode. This is visualized with keenerd's heatmap.py and can result in some really impressive plots when done with ~25% crop mode. Just refer to the -h "help" in rtl_power for instruction. There is also an rtl_power guide at keenerd's website.

    For RMS average power mode, which kicks in automatically for FFT bin sizes 1 MHz and larger, I do visualization of the resulting .csv file with gnuplot. Because the entire bandwidth is summed and saved as one value the the data rate to disk, and spectrogram dimensions are much lower than FFT mode.

    Example gnuplot visualization, annotated, and the gnuplot format, and colour palettes used to generate them.

    If you do a large number of frequency hops, (hundreds) then the time adds up. On my two computers the R820T tuner dongles average about 55 milliseconds per retune and sample cycle. I sometimes have dongles that'll fail to lock pll and go into a loop. The -e parameter sets a time limit for a run. Combining this time limit with a bash while loop results in pretty low downtime with resiliance to rtlsdr and USB failures.

    ```
    # 0.055 seconds *((1724-24) MHz/(1 MHz)) = 93.5 seconds = "-i".
    while true; do ./rtl_power -f 24M:1724M:1M -i 94 -g 20.7 -p 30 -e 1h >> 2013-12-02_totalpower_r820t_24-1724_1M_94s.csv; done
    ```

    To combine the results from multiple dongles just cat the files together. But on gnuplots end each new .csv filename requires you to manually edit the gnuplot format. Additionally you need to set the output spectrogram filename and a pixel width. I find for 1000 Mhz @ 1 MHz that approximately 1000px per 100 MB of file size is required to cover all gaps.

    ```
    gnuplot gnuplot-format-rms-mode.txt
    ```

    And that pops out a png.

    For rtl_fm stuff refer to keenerd's site's Rtl_fm Guide.

- Spektrum: an rtl_power GUI frontend:
  - There are a lot of rtl_power GUI frontends but the most useful for me is Spektrum. It uses a modified rtl_power with a Processing GUI front-end. It's available for linux and windows. One of it's best features is the "relative mode" for use in measuring changes in antennas and filters.

- patchvonbraun (Marcus Leech)'s multimode:
  - AM, FM, USB, LSB , WFM. TV-FM, PAL-FM. Very nice, easy to use (screenshots: main, scanning). It has an automated scanning and spectral zoom features with callbacks to click on the spectrogram or panorama to tune to the frequency of interest. There's a toggle for active gain control too. The way to get it is,

    ```
    svn co https://www.cgran.org/svn/projects/multimode
    ```

    then instead of using GRC, just run the multimode.py as is.

    ```
    make install
    python multimode.py
    ```

    If you run it outside of the svn created directory you might need to append ~/bin to pythonpath to find the helper script. If you used build-gnuradio it'll tell you what this is at the end of the install.

    Alternately set it in your ~/.bashrc. If you do the below make sure to reload in the terminal by "source ~/.bashrc")

    ```
    PYTHONPATH=/usr/local/lib/python2.6/dist-packages:~/bin
    export PYTHONPATH;
    ```

    When setting the sample rate it is rounded-down to a multiple of 200 Ksps so the decimation math works out.

    ```
    python multimode.py --srate=2.4M # use normal mode with 2.4 MHz bandwidth
    ./multimode.py --devinfo="rtl=0,direct_samp=1" # use direct sample mode
    python multimode.py --help
    ```

    If you have overruns like "OOOOoo..." then try reducing the sample rate or pausing the waterfall or spectrum displays.

    "The audio subsystem uses 'a' as the identifier, and UHD uses 'u'. With RTLSDR, it'll issue 'O' when it experiences an overrun. Which means that your machine isn't keeping up with the data stream. Sometimes buffering helps, but only if your machine is right on the edge of working properly. If it really can't, on average "keep up", no amount of buffering will help."

    If you have overruns like "aUaUaUaUa" or just "aaa" then the audio system is asking for samples at a higher rate than the DSP flow can provide (44vs48Khz, etc). Use "aplay -l" to get a list of the devices on your system.

    ```
    aplay -l
    ```

    The hw:X,Y comes from this mapping of your hardware -- in this case, X is the card number, while Y is the device number. Or you can use "pulse" for pulseaudio. Try specifying,

    ```
    python multimode.py--ahw hw:0,0
    ```

- gqrx:
  - Written by Alexandru Csete OZ9AEC "gqrx is an experimental AM, FM and SSB software defined receiver". The original version did not have librtlsdr support so changes were made by a number of others to add it. A couple weeks later Csete added gr-osmosdr support to the original. Dekar established a non-pulseaudio port of gqrx for Mac OSX. GNU Radio 3.7 has recently been released and it is not exactly backwards compatible. patchvonbraun's build-gnuradio.sh pulls ~~3.6.5~~ 3.7.x by default.

    As of August 9th 2013 Gqrx 2.2.0 has been released. This upgraded version can now be installed as binaries with all of it's dependencies pre-packaged on both Ubuntu linux (a custom PPA, no 10.04 packages) and Mac OS X! That includes all the GNU Radio stuff. So this is an all-in-one alternative to building GNU Radio from source.

    I think this person's guide is better than mine.

    ```
    git clone https://github.com/csete/gqrx.git
    cd gqrx
    # on Ubuntu/Debian, sudo apt-get install qtcreator , if you don't have it.
    qtcreator gqrx.pro      # press the build button (the hammer)
    # or avoid qtcreator and do it manually. if you have qt5 too, use "qmake-qt4"
    qmake
    make
    ```

  - rtlsdr w/Gqrx on N900 phones
    - xes provides pre-compiled packages of Gqrx and the GNU Radio dependencies for N900 linux cell phones.

- SDR#:
  - Written by prog (Youssef) for Windows. It is probably the best general purpose software for rtlsdr devices. Mono is slow and ugly on linux but if you restrict the sample rate it works fine. It's probably the easiest program to use, has the most diverse plugin ecosystem (example: Vasili_ru's plugins), and has the best DSP and features for dealing with the quirks of the rtlsdr dongles.

    As of 2015-09-14 the changes to Mono 4 allow SDR# to be viable to run on linux again. Make sure you have the latest Mono 4 though. This still requires soft linking in your system rtlsdr and portaudio library to the sdrshape.exe dir like below,

    ```
    ln -s  /usr/local/lib/librtlsdr.so librtlsdr.dll
    ln -s /usr/lib/x86_64-linux-gnu/libportaudio.so.2 libportaudio.so
    ```

    Just make sure you link your actual system rtlsdr and libportaudio, not my example path above. On debian/ubuntu find it by using locate,

    ```
    locate portaudio.so
    ```

    Roklobsta's rtlsdr.org has a more detailed SDR# linux configuration guide if you're having trouble.

  - **Update: As of 2015-10-15 ADBS# is no more.** ADBS# is another easy to use application by prog, but specifically for plotting aviation transponders like gr-air-modes does. The distributed binaries also runs under linux with mono (or native in windows) and output virtualradar compatible data on 127.0.0.1:47806. If your antenna condition is crappy, try using filter = 1.

- gr-fosphor:
  - gr-fosphor is an amazingly fast and information dense spectrogram and waterfall visualization using OpenCL hardware acceleration. It surpasses the Wx widget elements in performance, and so usability, by far. With this visualization you can easily skip through 1 GHz of spectrum very quickly and actually notice transient signals as they pass. Right now it is not very configurable, just arrow keys for scale. But expect this to be the preferred visualization block in the future. I have written up an barebones guide to installing gr-fosphor on Ubuntu 12.04. Modern gr-fosphor requires OpenCl 2. If you only have OpenCl 1.2 installed use this commit.

- gr-air-modes:
  - A decoder of aviation transponder Mode S including ads-b reports near 1090 Mhz. It can be coupled to software to show plane positions in near real time (ex: VirtualRadar). This works under mono on Ubuntu 12.04 but not 10.04. Originally written by Nick Foster (bistromath) and adapted to rtlsdr devices first by Steve Markgraf (stevem), bistromath later added rtlsdr support. Here's an example of basic decoding done with the stock antenna on the early version by stevem. Nowdays it's better to use bistromaths'.

    As of July 23, 2013 there was a major update to gr-air-modes which now includes a nice google maps overlay and **works on gnu radio 3.7 branch only**.

    ```
    git clone https://github.com/bistromath/gr-air-modes.git
    ```

    Here's an example of install process and first run looks like.

    ```
    # -d stands for rtlsdr dongle, location is "North,East"
    uhd_modes.py -d --location "45,-90"
    ```

    To use with virtual radar output add the below -P switch. Then open up virtualradar with mono and go to tools->options->basestation and put in the IP of the computer running uhd_modes. There are not many compatible planes in the USA so far so even if you are seeing lots of Mode-S broadcast in uhd_modes you might not see anything in virtualradar. Sometimes my server is running at superkuh.com:81/VirtualRadar/GoogleMap.htm.

    ```
    uhd_modes.py -d --location "45,-90" -P
    mono VirtualRadar.exe
    ```

- Dump1090:

  - Dump 1090 is a Mode S decoder specifically designed for RTLSDR devices.

    Antirezs' ADS-B program is really slick. It does not depend on GNU Radio, has a number of interactive modes, and it even optionally runs it's own HTTP server with googlemaps overlay of discovered planes; no virtualradar needed. It uses very little CPU and has impressive error correction. This is your best choice to play with plane tracking quickly.

```
# run cli interactively and create a googlemaps server on localhost port 8080
./dump1090 --aggressive --interactive --net-http-port 8080
# submit plane data to a tracking server
./dump1090 --aggressive --raw | socat -u - TCP4:sdrsharp.com:47806
```

- linrad:
  - A guide on how to use linrad with rtl-sdr, and a modification to the the librtlsdr e4000 tuner code to disable digital active gain! reddit thread

    "I tried various bits blindly and found a setting that eliminates the AGC in the RTL2832 chip. That is a significant part of the performance improvement."

    As of 2012-07-07 this feature was added to the main (librtlsdr) driver as well.
- LTE Cell Scanner and LTE Tracker:
  - "This is an LTE cell searcher that scans a set of downlink frequencies and reports any LTE cells that were identified. A cell is considered identified if the MIB can be decoded and passes the CRC check."

    "LTE-Tracker is a program that continuously searchers for LTE cells on a particular frequency and then tracks, in realtime, all found cells. With the addition of a GPS receiver, this program can be used to obtain basic cellular coverage maps."

    The author had only tested it on Ubuntu 12.04 but with some frustrating work replacing cmake files and compiling dependencies I made it work on 10.04. Scanner is very useful to get your dongle's frequency offset reliably and Tracker is very pretty. Remember to let your rtlsdr dongle warm up to equilibrium temperature before checking frequency error.
- kalibrate-rtl:
  - "Kalibrate, or kal, can scan for GSM base stations in a given frequency band and can use those GSM base stations to calculate the local oscillator frequency offset."

    The code was written by Joshua Lackey and made rtlsdr accessible by stevem. There is also a windows build made by Hoernchen. Let your rtlsdr dongle warm up to equilibrium temperature before running the test. When you're using this to find your frequency error it's important to use the -e option to specify intial error. 270k of bandwidth is used for GSM reception and if the error of the dongle is too large the FCCH-peak is outside the range. I compiled some install process and example usage notes.
- Simple FM (Stereo) Receiver
  - simple_fm_rcv also by patchvonbraun is the best sounding and tuning commercial FM software in my opinion. He released a major update to his gnuradio creation at the end of October.

```
svn co https://www.cgran.org/svn/projects/simple_fm_rcv
cd simple_fm_rcv/
cd trunk
less README
make
make install
## it'll install to ~/bin/, so I use ~/superkuh/bin below
set PYTHONPATH=/usr/local/lib/python2.6/dist-packages:/home/superkuh/bin
# run the python script
python simple_fm_rcv.py
# or edit it
gnuradio-companion simple_fm_rcv.grc
```

- my DongleLogger:
  - I wrote these scripts do automatic generation of 1D spectrograms, per frequency time series plots of total power, and 2D spectral maps over arbitrary frequency ranges using multiple dongles at once. There is almost no DSP done and it is very simple but the wideband spectrograms and time series can be informative and fun regardless. It uses gnuplot for graphics generation. **Obsolete. Use rtl_power instead**.

    You can see a typical gallery output at, http://erewhon.superkuh.com/gnuradio/live/ but more useful are the gnuplot spectrograms it can make.
- simple_ra: simple radio astronomy
  - A simple, GRC-based tool for small-scale radio astronomy, providing both Total Power and Spectral modes. It has a graphical stripchart display, and a standard FFT display. It also records both total-power and spectral data using an external C program that records the data along with timestamps based on the Local Mean Sidereal Time.

    This is another incredible tool by patchvonbraun. It does all the heavy lifting of integration over time and signal processing to get an accurate measurement of absolute power over a range. With it he has managed to pick out the transit of the milky way at the neutral hydrogen frequency using rtlsdr sticks and a pair of yagi antenna. The log file format is text and fairly easy to parse with gnuplot but it comes with 'process_simple_tpdat' for cutting it into the bits you want and making total power or spectral component graphs. It'll make a directory called "simple_ra_data" in your home by default. Don't forget to set the --devid to rtl otherwise gnuradio won't find the gr-osmosdr source and it'll substitute a gaussian noise source.

```
git clone https://github.com/patchvonbraun/gr-ra_blocks
cd gr-ra_blocks/
cmake . ; make ; sudo make install ; sudo ldconfig
cd ..
git clone https://github.com/patchvonbraun/simple_ra
cd simple_ra;
make
make install

# Using a single E4k tuner
./simple_ra --devid rtl=0,offset_tune=1 --longitude -45
# Using a second dongle that just happens to have a R820T tuner
./simple_ra --devid rtl=1 --longitude -45
# Use direct sample mode @ 1.25 MHz, log once per second, set longitude
./simple_ra --devid rtl=0,direct_samp=1 --freq 1.25e6 --longitude -90 --lrate 1
# generate multi-day averaged gnuplots using sidereal time
process_simple_tpdat 02:00:00 2.0 -t "11 GHz Test" -f 11ghz.png ~/simple_ra_data/tp-20130329-*.dat ~/simple_ra_data/tp-20130328-*.dat ~/simple_ra_data/tp-20130327-*.dat
./simple_ra -h (README)
```

- RTLSDR-Scanner
  - Ear to Ear Oak made this wideband total power scanner that generates 1D spectrum plots over any tunable ranges with arbitrary integration times. It can update a matplotlib python plot GUI in real time and has the ability to output cvs values as well as an internal format. It's very useful for finding what's broadcasting in your area quickly. Using it's csv output and gnuplot I visualized a scan from 54-1100 MHz.

    If you want to use the data in gnuplot you have to sort it and make sure the header is commented out.

    ```
    sort -n 54-1100_500ms.csv > sorted_54-1100_500ms.csv
    ```

    Here are some example gnuplot formats for the data.

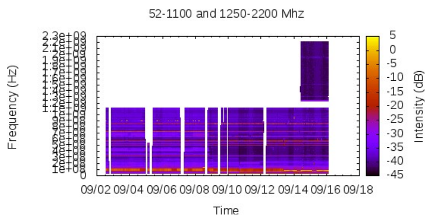
    You can comment out the header manually but I instead prefixed a hash to the log writing behavior at line 786,

    ```
    handle.write("# Frequency (MHz),Level (dB)\n")
    ```

Pager stuff

- Thomas Sailer's multimon, "Linux Radio Transmission Decoder" which I use to (try to) decode pager transmissions around 930Mhz. And more recently Dekar's multimonNG, a fork with improved error correction, more supported modes, and *nix/osx/windows support. Dekar also supplied a GRC receiver for pagers to decode pager transmissions in real-time using fifos. zarya has made rtl_flex.py, a gnuradio based flex decoder for pagers. It can be used to decode dutch p2000 messages, for example. This fills a gap in multimon-ng pager support.

DongleLogger: my pyrtlsdr lib based spectrogram and signal strength log and plotter



Purpose:

Obsolete. Use rtl_power instead. Automatic generation of and html gallery creation of wideband spectrograms using multiple rtlsdr dongles to divide up the spectrum. It also produces narrow band total charts, and other visualizations.

(not live): http://erewhon.superkuh.com/gnuradio/live/ - click the spectrograms for time series plot

These scripts cause the rtlsdr dongle to jump from frequency to frequency as fast as they can and take very rough total power measurement. This data is stored in human readable logs and later turned into wideband spectrograms by calling gnuplot. In order to further increase coverage of any given spectrum range multiple instances of the script can be run at once in the same directory adding to the same logs. Their combined output will be represented in the spectrogram.

Details:

I don't know much python but the python wrapper for librtlsdr pyrtlsdr was a bit easier to work with than gnu radio when I wanted to do *simple things without a need for precision or accuracy*. Actualy receivers with processing could be made with it too, but not by me. This is the gist of what it does,

```
power = 10 * math.log10(scan[freq]) = scan = self.scan(sdr, freq) = capture = sdr.read_samples(self.samples) = iq = self.packed_bytes_to_iq(raw_data) = raw_data = self.read_bytes(num_bytes)
```

The pyrtlsdr library can be downloaded by,

```
git clone https://github.com/roger-/pyrtlsdr.git
```

I have used the "test.py" matplotlib graphical spectrogram generator that came with pyrtlsdr as a seed from which to conglomerate my own program for spectrum observation and logging. Since I am not very good with python I had to pull a lot of the logic out into a perl script. So everything is modular. As of now the python script generates the spectrogram pngs and records signal strength (and metadata) in frequency named logs. It is passed lots of arguments.

These arguments can be made however you want, but I wrote a perl script to automate it along with a few other useful things. It can generate a simple html gallery of the most recent full spectral map and spectrograms with each linked to the log of past signal levels. Or it can additionally generate gnuplot time series pngs (example) and link those intead of the raw logs. It also calls LTE Cell Scanner and parses out the frequency offset for passing to graphfreq.py for correction. I no longer have it running because of the processor usage spikes which interrupt daily tasks. In the past I'd have rsync updating the public mirror with a big pipe every ~40 minutes.

Modifying pyrltsdr

As it is pyrtlsdr does not have the get/set functions for frequency correction even if I sent the PPM correct from the perl script. Since the hooks (?) were already in librtlsdr.py (line 60-66) but just not pythonized in rtlsdr.py they were easy to add to the library. These changes are required to use frequency correction and make the int variable "err_ppm" available. I have probably shown that I don't know anything about python with this description.

I forked roger-'s pyrtlsdr on github and added them there for review or use, https://github.com/superkuh/pyrtlsdr/commit/ffba3611cf0071dee7e1efec5c1a582e1e344c61. I apologize for cluttering up the pyrtlsdr namespace with such trivial changes but I'm new to this and github doesn't allow for private repositories.

What you should be using instead.

- rtl_power was recently (2013-08-20) released by keenerd. It does most of what my scripts do, except much better, faster, and easier. I highly recommend you try it first.
- RTLSDR Scanner by Ear To Ear Oak is awesome for generating 1D wideband spectrograms.
- Enoch Zembecowicz made a polished and useful sdr logging tool
- Panteltje's rtl_sdr_scan is another tool like RTLSDR Scanner for 1D total power scans. It is a good reference for using librtlsdr with C.
- If you are serious about measuring total power over one 2.5 Mhz range then simple_ra, or simple radio astronomy, is best.

fast version: see below

- donglelogger-faster.tar.gz - all needed files including pyrtlsdr

-
    - radioscan_faster.pl - pyrltsdr using script; frequency setting and incrimenting, sampling, and logging.
    - graphfreqs_faster5.py - option passing, log parsing, plot making, frequency corrections wrapper, html image gallery generation
    - graphfreqs_gnuplot.py - legacy functions

slow version:

- graphfreqs.py - pyrltsdr using script; sampling, and logging
- radioscan.pl - manages graphfreqs, parses logs, makes plots, gets frequency corrections, generates gallery

The faster version

Speed ups, Inline C usb reset, and avoiding dongle reinitialization... (less options)

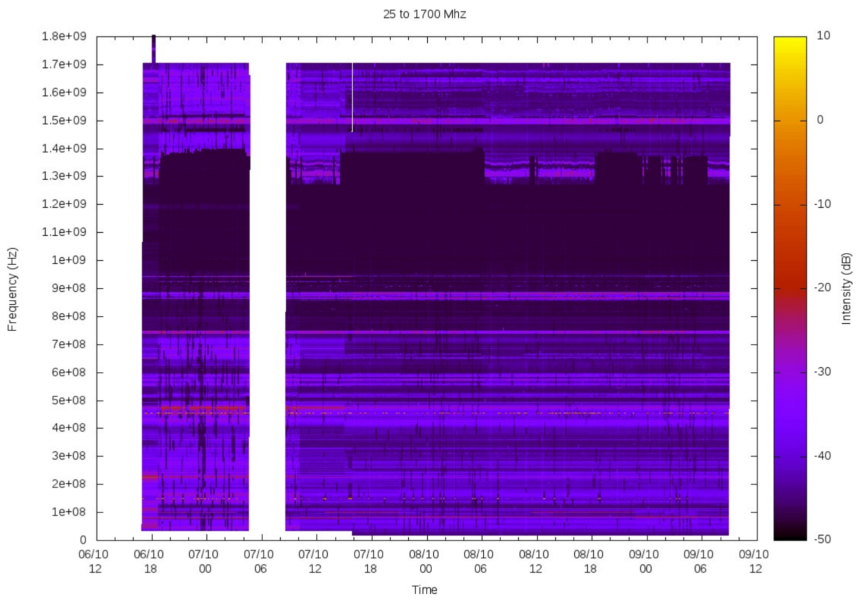
cli switches/options

```
-dev 1                  :: rtlsdr device ID (use to pick dongle)
-g 30                   :: gain
-s                      :: interval between center frequencies
-r 2400000              :: sample rate
-d2 /path/here          :: path to the directory to put logs, plots, gallery
-c 751                  :: LTC Cell scanner frequency offset correction, takes freq in Mhz of base cell
-w                      :: turn on web gallery generation
-p                      :: turn on gnuplot time series charts for every freq (don't use to maximize speed)
-m                      :: generate full range spectrogram using all.log (this is the most useful thing)
-mr "52-1108,1248-1400,1900-2200"       :: set of frequency ranges to plot as a another spectrogram
```

These two scripts do fast scans within python from x to y frequency. Enabled it with -fast and make sure to set start and stop frequency with -f1 and -f2. Do not use -flist with this option.

```
$ while true; do perl radioscan_faster.pl -d2 /tmp/faster -f1 25 -f2 1700 -fast -g 30 -r 2400000 -s 1.2 -m -p; sleep 1; done;
Running graphfreq in non-batch fast mode 25 to 1700 Mhz at 1.2 Mhz spacings.
 Spectrograms and text output disabled.
Using LTE Cell Scanner to find frequency offset from 751 Mhz station...Found Rafael Micro R820T tuner
-44k frequency offset. Correcting -58 PPM.
Generating spectral map.

python ./graphfreqs_faster.py 40000000 2400000 30 -58 /tmp/faster 1700000000
Found Rafael Micro R820T tuner
... (repeat many, many times)
```

This is an example output "spectral map" (a spectrogram with a silly name).



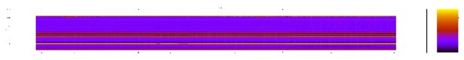
This example output above shows the overloading effects of using a wideband discone that picks up off-band noise. Each column is made up of small squares colored by intensity of the signal. Since the scripts start at the low frequency and sweep to high there is a small time delay between the bottom and top (see it more clearly zoomed in). And this is represented as the slant of the row. Sometimes strong signals will swamp out others resulting in discontinuities displaying as small dark vertical bands.

Or fast (-fast) scan a smaller range with smaller range (-f1,-f2: 24-80Mhz), with smaller samplerate (-r: 250 Khz) at smaller intervals (-s: 400Khz steps) with a gain of ~30. Only output a large spectrogram of all frequencies to the directory specified with -d2 as spectral-map.png. This example does not use frequency offset correct (-c) for even faster speeds.

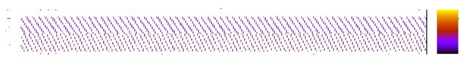
```
while true; do perl radioscan_faster.pl -d2 /home/superkuh/radio/2012-02-02_R820T_Discone_lowfreq -f1 24 -f2 80 -fast -g 30 -r 250000 -s 0.4 -m -p; sleep 1; done;
Running graphfreq in non-batch fast mode 24 to 80 Mhz at 0.4 Mhz spacings.
 Spectrograms and text output disabled.
Generating spectral map.

python ./graphfreqs_faster5.py 24000000 250000 30 0 /home/superkuh/radio/2012-02-02_R820T_Discone_lowfreq 80000000 400000
Found Rafael Micro R820T tuner
Exact sample rate is: 250000.000414 Hz
```

Combining multiple rtlsdr devices for greater speed

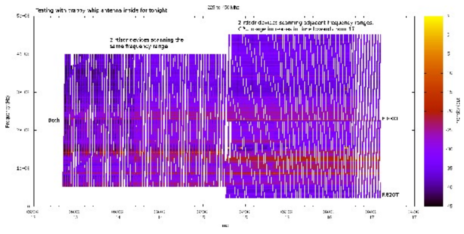
```
Full
```



```
Zoom
```



By splitting up the spectrum into multiple smaller slices and giving them to multiple dongles the time required for one scan pass can be greatly improved. The above spectrogram is made with 2 dongles, one for the lower half and one for the upper. It is from ***ryannathans who also contributed the code for for specifying device ID***. This is as simple as running the script twice but giving each instance a different "-dev" argument to specify device ID. You can run as many rtlsdr devices with my scripts as you wish (up to the USB and CPU limits). If they are using the same directory (-d2) their log data will be combined automagically for better coverage.



```
while true; do perl radioscan_faster.pl -d2 /home/superkuh/radio/2013-06-09_multidongle -f1 25 -f2 525 -fast -g 40 -r 2400000 -s 1.2 -m -dev 0; sleep 1; done;
...
Found Rafael Micro R820T tuner

while true; do perl radioscan_faster.pl -d2 /home/superkuh/radio/2013-06-09_multidongle -f1 525 -f2 1025 1100 -fast -g 29 -r 2400000 -s 1.2 -m -dev 1; sleep 1; done;
...
Found Elonics E4000 tuner
```

Outlier signals skewing your color map scale?

Sometimes I get corrupt samples that show a signal level of +60dB. These skew the scale of the output spectrograms. If I notice that they have occurred during a long run I'll use grep to find them and remove them manually. I replace the signal level with the level of the previous non-corrupt sample. In the future I'll build this kind of outlier removal in to the scripts, or sanity check before writing them.

```
grep -rinP " (3|4|5|6)\d+\.\d+" *.log
```

All the incremental improvements in speed I've made above are okay but not very easy to maintain with multiple script types (bash/perl/python). I'm slowly putting together an Inline C based perl wrapper for exposing librtlsdr's functions within a perl script to write this as a standalone in perl. This is slow work because I've never done anything like it before.

rtlsdrperl - what if there were a perl wrapper for librtlsdr?

Well, there never will be. But here's some example code anyway.

```
#!/usr/bin/perl
use Inline C => DATA => LIBS => '-L/usr/local/lib/ -lrtlsdr -lusb';
use warnings;
use strict;

my $fuckingtest = get_device_name(0);
print "Device name: $fuckingtest\n";
my $fuckingtest2 = get_device_count();
print "# Devices: $fuckingtest2\n";

__END__
__C__
const char* rtlsdr_get_device_name(uint32_t index);
char * get_device_name(int count) {
        char* res = rtlsdr_get_device_name(count);
        return res;
}

uint32_t rtlsdr_get_device_count(void);
int get_device_count() {
        int hem = rtlsdr_get_device_count();
        return hem;
}
```

Older version

graphfreqs.py

You have to have the modified pyrtlsdr with the get/set functions for frequency correction. LTE Cell Scanner should also be installed so the "CellSearch" binary is available. Then download the two scripts above and put them in the same directory. For large bandwidths sampled this feature, ppm error correction, has an unnoticably small effect but I wanted to add it anyway.

To call the spectrogram/log generator by itself for 431.2 Mhz at 2.4MS/s with a gain of 30 and frequency correction of 58 PPM use it like,

```
python graphfreqs.py 431200000 2400000 30 58
```

I've disabled the matplotlib (python) per frequency spectrogram plots for frequencies over 1 Ghz because there's not much going on up there. Also, the x-axis ticks and labels become inaccurate for some reason.

Logs and format

The signal strength logs, named by frequency (e.g. 53200000.log), use unix time and are comma seperated with newlines after each entry. In order of columns it is: unix time , relative signal level , gain in dB, PPM correction.

```
1345667680.28 , -34.65 , 29 , 57
1345667955.59 , -34.67 , 29 , 57
1345668004.37 , -34.55 , 29 , 57
1345668110.06 , -33.88 , 29 , 57
```

It also generates a log file with all frequencies for use with gnuplot, all.log. This file has unixtime first, then frequency, then gain and ppm error.

```
1347532002.5 52000000 -14.84 29.0 58
1347532004.88 53200000 -17.84 29.0 58
1347532007.04 54400000 -17.98 29.0 58
1347532009.04 55600000 -19.78 29.0 58
1347532011.02 56800000 -24.04 29.0 58
1347532012.98 58000000 -26.21 29.0 58
1347532014.92 59200000 -25.10 29.0 58
```

radioscan.pl

The radioscan.pl script is used to automate calling graphfreqs in arbitrary steps. To generate plots and signal strength for 52 Mhz to 1108 Mhz with a gain of 30, sample rate of 2.4MS/s, and an interval between center frequencies of 1.2 Mhz, call it like,

```
$ perl radioscan.pl -flist "52-1108,1248-2200" -g 30 -r 2400000 -s 1.2
```

cli switches/options

```
-flist "52-1108,1248-2200"  :: sets of frequency ranges to scan.
-g 30                   :: gain
-s                      :: interval between center frequencies
-r 2400000              :: sample rate
-d1 /path/here          :: path to where the scripts are if now pwd
-d2 /path/here          :: path to the directory to put logs, plots, gallery
-c 751                  :: LTC Cell scanner frequency offset correction, takes freq in Mhz of base cell
-w                      :: turn on web gallery generation
-p                      :: turn on gnuplot time series charts for every freq
-m                      :: generate full range spectral map using all.log
-mr "52-1108,1248-1400,1900-2200"       :: set of frequency ranges to plot as a another spectral map
```

Because I can use the default directories I keep it running like the below, but anyone else should make sure to set -d2.

```
$ while true; do perl radioscan.pl -flist "52-1108,1248-2200" -g 30 -r 2400000 -s 1.2 -w -c 751 -p -m -mr "52-1108"; sleep 1; done;

Running pyrtl graphfreq batch job 52 to 1108 Mhz at 1.2 Mhz spacings.
Using LTE Cell Scanner to find frequency offset from 751 Mhz station...Found Elonics E4000 tuner
42.6k frequency offset. Correcting 56 PPM.
Generating spectral map.
Generating another spectral map over only 52-1108.

python ./graphfreqs_offset.py 52000000 2400000 30 56
Found Elonics E4000 tuner
python ./graphfreqs_offset.py 53200000 2400000 30 56
Found Elonics E4000 tuner
python ./graphfreqs_offset.py 54400000 2400000 30 56
Found Elonics E4000 tuner
...
python ./graphfreqs_gnuplot.py 316000000 2400000 30 56
Found Elonics E4000 tuner
Dongle froze, reseting it's USB device...
Resetting USB device /dev/bus/usb/001/017
Reset successful
python ./graphfreqs_gnuplot.py 317200000 2400000 30 56
Found Elonics E4000 tuner
..
Generating page, moving images.

starting rsync...
```

Tuner/USB freeze solution with unplugging

edit: as of Jan 5th 2013, librtlsdr has added soft reset functionality

Since graphfreqs.py's initializing and calling of rtl-sdr happens so frequently there are sometimes freezes. To fix these the USB device has to be reset. In the past I would accomplish this by un and re-plugging the cord manually. But that meant lots of downtime when I was away or sleeping. So, I've added in a small C program to the perl script using Inline::C that exposes a function, resetusb(). It is used if the eval loop around the graphfreqs call takes more than 10 seconds. *This means you need Inline::C to run this script*. To look at the original C version with a good explanation of how to use it click here.

```
sub donglefrozen {
        my $usbreset;
        my @devices = split("\n",`lsusb`);
        foreach my $line (@devices) {
                if ($line =~ /\w+\s(\d+)\s\w+\s(\d+):.+Realtek Semiconductor Corp\./) {
                        $usbreset = "/dev/bus/usb/$1/$2";
                        resetusb($usbreset);
}}}

__END__
__C__
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <linux/usbdevice_fs.h>

int resetusb(char *dongleaddress)
{
        const char *filename;
        int fd;
        int rc;
        filename = dongleaddress;
        fd = open(filename, O_WRONLY);
        if (fd < 0) {
                perror("Error opening output file");
                return 1;
        }
        printf("Resetting USB device %s\n", filename);
        rc = ioctl(fd, USBDEVFS_RESET, 0);
        if (rc < 0) {
                perror("Error in ioctl");
                return 1;
        }
        printf("Reset successful\n");
        close(fd);
        return 0;
}
```

My rtlsdr receiver + w/gnuradio implementation of the 11 GHz VSRT solar interferometer

As far as I understand it, the VSRT design is a subset of intensity interferometer that uses the frequency error between multiple 11 GHz satellite TV "low noise downconverter block" (LNBF) clocks to create a beat frequency in the total power integrated. I am basically copying the MIT Haystack Very Small Radio Telescope (VSRT) but replacing the discrete component integrator and USB video input device with an rtlsdr dongle. The idea is to spend as little on hardware as possible.

With modern LNBF the error between same model parts is about 30 ppm which results in beat frequencies of ~100 KHz at the 10 GHz of the mixers. With this kind of front-end there are no nulls but the fringe modulation can still be read out as variations in count of histogram bins that contain the beat frequency (in the total power fft). This intensity measurement proxy traces out the the envelope of the fringes and varies as a sinc function of distance between antenna. Knowing this and the distance can give you high angular diameter and position measurements of very bright radio sources.

```
$75 2x 18" satellite dishes w/mounts shipped
$10 2x Ku LNFB (PLL321 S-2, ~30ppm error, RDA3560 w/27Mhz xtal.)
$10 rtlsdr receiver (r820t or e4k, ~30ppm error)
$10 power combiner (cheaper ones work too)
$5 coaxial power injector (LPI 2200)
$20 coaxial power supply (LPI 188PS) + diodes
$20 100ft RG6 quadshield + F connectors
$130 Two Dish Position Motors (HH90)
$60 DVB-S PCI card (Skystar 2, DiSEqC 1.2)
$10 DiSEqC 1.2 switch
$80 PVC, metal stock, drill bits
```

Historical and other context.

For a detailed mathematical explanation of VSRT see MIT Haystack's VSRT Introduction. There is also a thread on the Society for Amateur Radio Astronomers list discussing the VSRT design. The more general concept of intensity interferometry, where you correlate total power instead of frequency, was originally developed by Hanbury-Brown & Twiss. Roger Jennison was around too. "The Early Years of Radio Astronomy: Reflections Fifty Years after Janskys Discovery" by W T Sullivan (2005) is an excellent source about Hanbury Brown and Twiss's side of it. The chapter "The Invention and Early Devlopment of The Intensity Interferometer" (pdf) is fascinating. Also see "The Development of Michelson and Intensity Long Baseline Interferometry" (pdf). It covers not only the technical concepts but also historical context, detailed hands-on implementations, and other personal anectdotes. And check out Jennison's book "Radio Astronomy" (1966)) as he invented the process of phase closure which uses a third antenna signal combined mathematically to recover some of the missing phase information. Arranged in a triangle of projected baselines the phase errors cause equal but opposite phase shifts in ajoining baselines, canceling out in the "closure phase". The MIT Haystack groups managed to resolve individual sunspots groups moving across the solar disk using with the technique with the VSRTs.

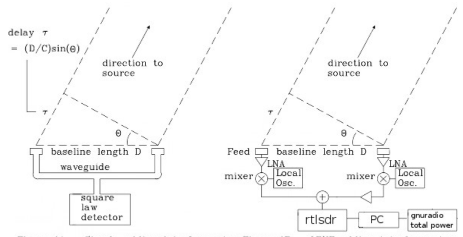


Figure 1A – Simple adding interferometer Figure 1B – LBNF adding interferometer

---

"An interferometer is an instrument that combines two signals (normally from two detectors) in a manner that the signals interfere to produce a resultant signal. The resultant signal is usually the vector sum of the two signals, but in some cases it is the product or some other mix. The traditional interferometer, usually studied and analyzed in physics courses, combines the two signals in a way that both amplitude and phase information are used. By varying the positions of the two detectors, it is possible to synthesize an effective aperture that is equivalent to the separation of the detectors and to reconstruct the impinging wavefront, thus providing significant information about the extent and structure of the signal source. The traditional phase-sensitive interferometer requires retention of the signal phase at each detector – the phase-sensitive interferometry technique will not be discussed in detail here."

"A special case of the interferometer is the intensity interferometer, which performs an intensity correlation of signals from the two detectors. Although in the intensity interferometer the phase information from the two antennas is discarded, the correlation of the two signals remains useful. Aperture synthesis is not practical, but some important source characteristics may be determined."

I think the VSRT is a special case of intensity interferometer where you don't try to align samples by time after recording. Instead you just look for the baseline distance sinc pattern in total power at the beat frequency of the unsynchronized clocks.

Implementation so far.

So far I've only done it with manual pointing screwed to a board. The interferometry correlation is done with a satellite tv market stripline power combiner at the intermediate frequency (IF, ~950-1950 MHz) and then an rtlsdr dongle is used to measure the total power of a 2.4 MHz bandwidth of the intermediate frequency range. I use a gnuradio-companion flowgraph to take the total power and then do a fourier transform of the total power. In this fourier transform the fringes show up as a modulation of the count in the FFT bins which correspond to the difference in frequency between the two downconverters. In my case this is about ~100 KHz.

In the Haystack VSRT memos a line drop amplifier, or two, are sometimes put behind the respective LNBF IF coax outputs or the power combiner. With the rtlsdr dongle and relative short (<10m) baselines of RG6 this isn't required.

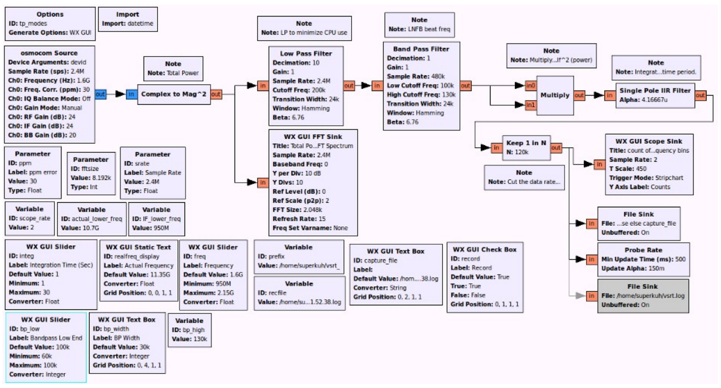
The GUI allows for setting the exact 2.4 MHz bandwidth of the IF range to sample and the total power FFT bin bandpass to where and what the LNBF beat frequency is. The file name is autogenerated to the format,

```
prefix + datetime.now().strftime("%Y.%m.%d.%H.%M.%S") + ".log"
```

The time embedded in the filename is later used by a perl script, vsrt_log_timeplot.pl, which converts and metadata tags the binary records to gnuplot useable text csv format for making PNG plots.

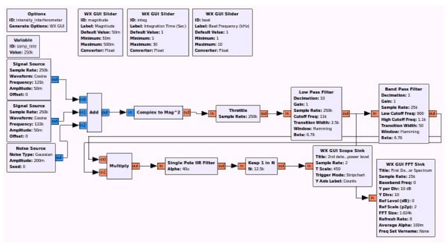
Download

- total power modes (tp-modes.grc)
- vsrt_log_timeplot.pl



```
Options
ID: tp_modes
Generate Options: WX GUI

Import
Import: datetime

Note
Note: LP to minimize CPU use

Note
Note: LNFB beat freq

osmocom Source
Device Arguments: devid
Sample Rate (sps): 2.4M
Ch0: Frequency (Hz): 1.6G
Ch0: Freq. Corr. (ppm): 30
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 24
Ch0: IF Gain (dB): 24
Ch0: BB Gain (dB): 20

Note
Note: Total Power

Complex to Mag^2

Low Pass Filter
Decimation: 10
Gain: 1
Sample Rate: 2.4M
Cutoff Freq: 200k
Transition Width: 24k
Window: Hamming
Beta: 6.76

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 480k
Low Cutoff Freq: 100k
High Cutoff Freq: 130k
Transition Width: 24k
Window: Hamming
Beta: 6.76

Note
Note: Multiply...If^2 (power)

Multiply

Note
Note: Integrat...time period.

Single Pole IIR Filter
Alpha: 4.16667u

Parameter
ID: ppm
Label: ppm error
Value: 30
Type: Float

Parameter
ID: fftsize
Value: 8.192k
Type: Int

Parameter
ID: srate
Label: Sample Rate
Value: 2.4M
Type: Float

WX GUI FFT Sink
Title: Total Po...FT Spectrum
Sample Rate: 2.4M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 2.048k
Refresh Rate: 15
Freq Set Varname: None

Keep 1 in N
N: 120k

Note
Note: Cut the data rate...

WX GUI Scope Sink
Title: count of...quency bins
Sample Rate: 2
T Scale: 450
Trigger Mode: Stripchart
Y Axis Label: Counts

Variable
ID: scope_rate
Value: 2

Variable
ID: actual_lower_freq
Value: 10.7G

Variable
ID: IF_lower_freq
Value: 950M

File Sink
File: ...se else capture_file
Unbuffered: On

WX GUI Slider
ID: integ
Label: Integration Time (Sec)
Default Value: 1
Minimum: 1
Maximum: 30
Converter: Float

WX GUI Static Text
ID: realfreq_display
Label: Actual Frequency
Default Value: 11.35G
Converter: Float
Grid Position: 0, 0, 1, 1

WX GUI Slider
ID: freq
Label: Frequency
Default Value: 1.6G
Minimum: 950M
Maximum: 2.15G
Converter: Float

Variable
ID: prefix
Value: /home/superkuh/vsrt_

Variable
ID: recfile
Value: /home/su...1.52.38.log

WX GUI Text Box
ID: capture_file
Label:
Default Value: /hom....38.log
Converter: String
Grid Position: 0, 2, 1, 1

WX GUI Check Box
ID: record
Label: Record
Default Value: True
True: True
False: False
Grid Position: 0, 1, 1, 1

Probe Rate
Min Update Time (ms): 500
Update Alpha: 150m

File Sink
File: /home/superkuh/vsrt.log
Unbuffered: On

WX GUI Slider
ID: bp_low
Label: Bandpass Low End
Default Value: 100k
Minimum: 60k
Maximum: 100k
Converter: Integer

WX GUI Text Box
ID: bp_width
Label: BP Width
Default Value: 30k
Converter: Integer
Grid Position: 0, 4, 1, 1

Variable
ID: bp_high
Value: 130k
```

Who else helped


Options
ID: intensity_interferometer
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 250k

WX GUI Slider
ID: magnitude
Label: Magnitude
Default Value: 50m
Minimum: 50m
Maximum: 500m
Converter: Float

WX GUI Slider
ID: integ
Label: Integration Time (Sec)
Default Value: 1
Minimum: 1
Maximum: 30
Converter: Float

WX GUI Slider
ID: beat
Label: Beat Frequency (kHz)
Default Value: 1
Minimum: 1
Maximum: 10
Converter: Float

Signal Source
Sample Rate: 250k
Waveform: Cosine
Frequency: 121k
Amplitude: 50m
Offset: 0

Signal Source
Sample Rate: 250k
Waveform: Cosine
Frequency: 122k
Amplitude: 50m
Offset: 0

Noise Source
Noise Type: Gaussian
Amplitude: 200m
Seed: 0

Add

Complex to Mag^2

Throttle
Sample Rate: 250k

Low Pass Filter
Decimation: 10
Gain: 1
Sample Rate: 250k
Cutoff Freq: 11k
Transition Width: 2.5k
Window: Hamming
Beta: 6.76

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 25k
Low Cutoff Freq: 900
High Cutoff Freq: 1.1k
Transition Width: 50
Window: Hamming
Beta: 6.76

Multiply

Single Pole IIR Filter
Alpha: 40u

Keep 1 in N
N: 12.5k

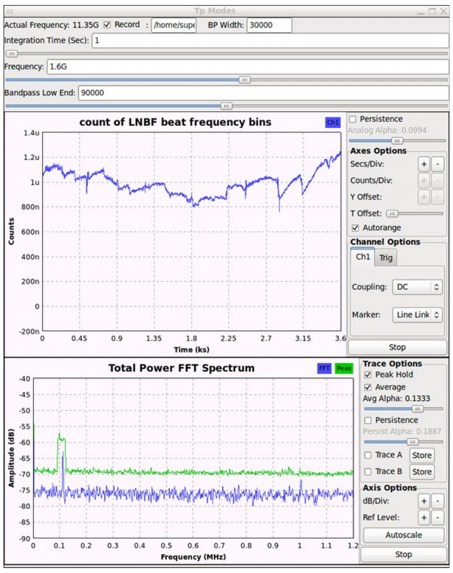
WX GUI Scope Sink
Title: 2nd dete...power level
Sample Rate: 2
T Scale: 450
Trigger Mode: Stripchart
Y Axis Label: Counts

WX GUI FFT Sink
Title: First De...or Spectrum
Sample Rate: 25k
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 8
Average Alpha: 100m
Freq Set Varname: None

I consulted with patchvonbraun a lot for the software/gnuradio side. He gave me an example of how to use the WX GUI Stripchart and I would not have guessed I needed to square the values from the beat frequency bins *after* the first squaring for taking total power. He made a generic simulator for dual free running clocks LNBF intensity interferometers. You don't even need to have an rtlsdr device to run it; only an up to date install of gnuradio. It is an easy way to understand how to do interferometry without a distributed clock signal.

patchvonbraun's: simulated-intensity-interferometer.grc

Physical


Tp Modes

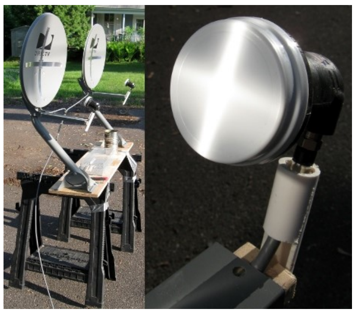
Actual Frequency: 11.35G ✓ Record : /home/supe BP Width: 30000

Integration Time (Sec): 1

Frequency: 1.6G

Bandpass Low End: 90000

With this setup on a 1 meter baseline and a intermediate tuning frequency of 1.6 GHz IF (10700 MHz+(1600 MHz−950 MHz)= 11350 MHz) the main beamwidth would be about 70*(c/11GHz)/1m), or 1.9 degrees. This does not resolve the solar disk (~0.5 deg) during drift scans. I have been told that the magnitude goes down in a SINC pattern as you widen the baseline and approach resolving the source but I will not resolve the sun initially. In the VSRT Memos "Development of a solar imaging array of Very Small Radio Telescopes" a computationally complex way to resolve individual action regions is done with a 3rd dish providing "phase closure" in the array on a slanted north-south baseline in addition to the existing east-west baseline. I try to point my dishes so that the Earth is passing the sun through the beam at ~12:09pm (noon) each day. To aid in pointing a cross of reflective aluminum tape is applied center of the dish. This creates a cross of light on the LNBF feed when it is in the dish focal plane and the dish is pointed at the sun. The picture below is from later in the day, the one of the left shows the sun drifting out of the beam as it sets. I made my LNBF holders out of small pieces of wood compression fit in the dish arm. There are grooves for the RG6 coax to fit ground out with a rotary tool. The PVC collars have slots cut in the back with screws going into the wood to set the angle.

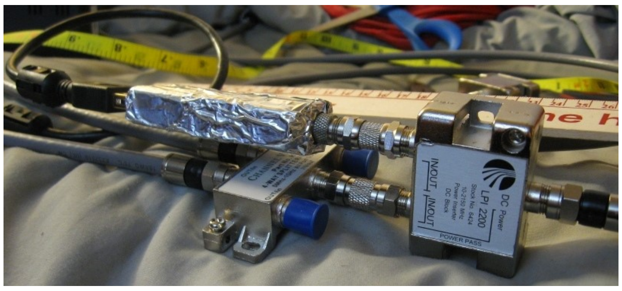


The screenshot shows a short run near sunset on an otherwise cloudy day. The discontinuities are me running outside and manually re-pointing the dishes. But it does highlight how the beat frequency of the 2 LNBF varies as they warm up when turned on. It starts down at ~90 KHz but within 10 minutes it rises to ~115 KHz. After it reaches equilibrium the variation is ~ -+1 KHz. I could change the existing 80-120 KHz bandpass to a 110-120 KHz bandpass and have better sensitivity. But that bandwidth is something that has to be found empirically with each LNBF pair and set manually within the GUI for now.

patchvonbraun said it was feasible to identify the frequency bins with the most counts and that there was an example within the simpla_ra code,

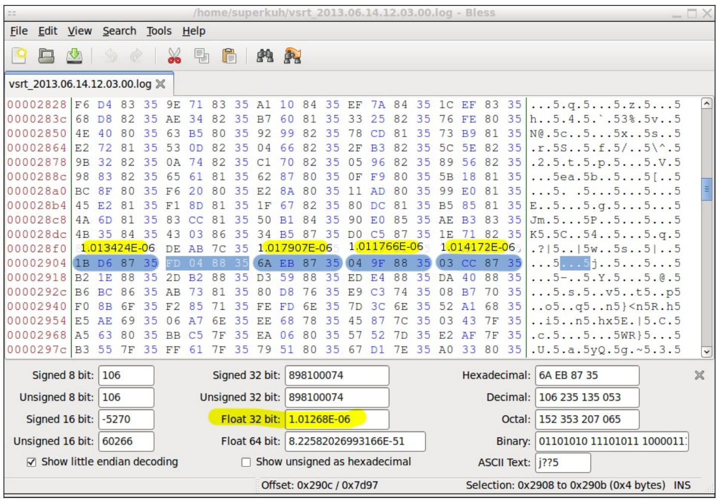
"You could even have a little helper function, based on a vector probe, that finds your bin range, and tunes the filter appropriately."

The below close up of indoor testing showing how everything is connected on the rtlsdr side showing the power injector, e4k based rtlsdr (wrapped in aluminum tape), and the stripline based satellite power combiner for correlation. The two rg6 quadshield coaxial lines going from the power combiner to the ku band LNBF are as close to the same length as I could trim them. I use a 1 amp 18v power supply and coaxial power injector to supply power to the LNB and any amplifiers. This voltage controls linear polarization (horiztonal/vertical) and it can be changed by putting a few 1 amp 1N4007 in series with the power line to drop the voltage.



Accessory scripts.

tp-modes.grc produces binary logs that are pretty simple. The count of the LNBF beat frequency bins in the bandpass are saved as floats represented as 4 pairs of hexadecimal. When the integration time is set to the default 1 second then one 4 byte data point is written to the log every 0.5 seconds. I highly recommend not changing this for now. There is no metadata or padding. Here's a screenshot of a run using the utility "bless",


/home/superkuh/vsrt_2013.06.14.12.03.00.log - Bless
File Edit View Search Tools Help
vsrt_2013.06.14.12.03.00.log

```
00002828 F6 D4 83 35 9E 71 83 35 A1 10 84 35 EF 7A 84 35 1C EF 83 35 ...5.q.5...5.z.5...5
0000283c 68 D8 82 35 AE 34 82 35 B7 60 81 35 33 25 82 35 76 FE 80 35 h..5.4.5.`.53%.5v..5
00002850 4E 40 80 35 63 B5 80 35 92 99 82 35 78 CD 81 35 73 B9 81 35 N@.5c..5...5x..5s..5
00002864 E2 72 81 35 53 0D 82 35 04 66 82 35 2F B3 82 35 5C 5E 82 35 .r.5S..5.f.5/..5\^.5
00002878 9B 32 82 35 0A 74 82 35 C1 70 82 35 05 96 82 35 89 56 82 35 .2.5.t.5.p.5...5.V.5
0000288c 98 83 82 35 65 61 81 35 62 87 80 35 0F F9 80 35 5B 18 81 35 ...5ea.5b..5...5[..5
000028a0 BC 8F 80 35 F6 20 80 35 E2 8A 80 35 11 AD 80 35 99 E0 81 35 ...5. .5...5...5...5
000028b4 45 E2 81 35 F1 8D 81 35 1F 67 82 35 80 DC 81 35 B5 85 81 35 E..5...5.g.5...5...5
000028c8 4A 6D 81 35 83 CC 81 35 50 B1 84 35 90 E0 85 35 AE B3 83 35 Jm.5...5P..5...5...5
000028dc 4B 35 84 35 43 03 86 35 34 B5 87 35 D0 C5 87 35 1E 71 82 35 K5.5C..54..5...5.q.5
000028f0 1.013424E-06 DE AB 7C 35 1.017907E-06 1.011766E-06 1.014172E-06 .?|5..|5w..5s..5|..5
00002904 1B D6 87 35 FD 04 88 35 6A EB 87 35 04 9F 88 35 03 CC 87 35 ...5...5j..5...5...5
00002918 B2 1E 88 35 2D B2 88 35 D3 59 88 35 ED E4 88 35 DA 40 88 35 ...5-..5.Y.5...5.@.5
0000292c B6 BC 86 35 AB 73 81 35 80 D8 76 35 E9 C3 74 35 08 B7 70 35 ...5.s.5..v5..t5..p5
00002940 F0 8B 6F 35 F2 85 71 35 FE FD 6E 35 7D 3C 6E 35 52 A1 68 35 ..o5..q5..n5}<n5R.h5
00002954 E5 AE 69 35 06 A7 6E 35 EE 68 78 35 45 87 7C 35 03 43 7F 35 ..i5..n5.hx5E.|5.C.5
00002968 A5 63 80 35 BB C5 7F 35 EA 06 80 35 57 52 7D 35 E2 AF 7F 35 .c.5...5...5WR}5...5
0000297c B3 55 7F 35 FF 61 7F 35 79 51 80 35 67 D1 7E 35 A0 33 80 35 .U.5.a.5yQ.5g.~5.3.5
```

Signed 8 bit: 106
Unsigned 8 bit: 106
Signed 16 bit: -5270
Unsigned 16 bit: 60266
Signed 32 bit: 898100074
Unsigned 32 bit: 898100074
Float 32 bit: 1.01268E-06
Float 64 bit: 8.22582026993166E-51
Hexadecimal: 6A EB 87 35
Decimal: 106 235 135 053
Octal: 152 353 207 065
Binary: 01101010 11101011 1000011
ASCII Text: j??5
[✓] Show little endian decoding
[ ] Show unsigned as hexadecimal
Offset: 0x290c / 0x7d97
Selection: 0x2908 to 0x290b (0x4 bytes) INS

In order to convert the binary logs of 4 byte records into something gnuplot can parse I use a simple perl script,

```
#!/usr/bin/perl
use warnings;
use strict;

my $data = '/home/superkuh/vsrt_2013.06.13.12.26.47.log';
my $bytelength = 4;
my $format = "f"; # floats (little endian)
my $num_records;

if ($ARGV[0]) {
        $data = $ARGV[0];
} else {
        print "you need to pass the log file path as an argument.";
        exit;
}

open(LOG,"$data") or die "Can't open log.\n$!";
binmode(LOG);

my $i = 0;
until ( eof(LOG) ) {
        my $record;
        my $decimal;
        read(LOG, $record, $bytelength) == $bytelength
                or die "short read\n";
        $decimal = unpack($format, $record);
        printf("$i,\t$decimal\n", $decimal);
        $i++;
}
```

Now I have the filename which gives the time the gnuradio-companion grc file started running. This is *not* the time I hit the record button and started logging. The offset is a second or two. Ignoring that, it is possible to use the start time encoded in the log file name to figure out when a particular measurement was taken. To do that I have to know the interval between entries saved to the binary log.

```
$ date && ls -l /home/superkuh/vsrt_2013.06.14.12.03.00.log && sleep 60 && date && ls -l /home/superkuh/vsrt_2013.06.14.12.03.00.log
Fri Jun 14 13:05:24 CDT 2013
-rw-r--r-- 1 superkuh superkuh 29644 2013-06-14 13:05 /home/superkuh/vsrt_2013.06.14.12.03.00.log
Fri Jun 14 13:06:24 CDT 2013
-rw-r--r-- 1 superkuh superkuh 30124 2013-06-14 13:06 /home/superkuh/vsrt_2013.06.14.12.03.00.log

((30124-29644)/4)/60 = 2

$ date && ls -l /home/superkuh/vsrt_null.log && sleep 60 && date && ls -l /home/superkuh/vsrt_null.logFri Jun 14 13:44:36 CDT 2013
-rw-r--r-- 1 superkuh superkuh 8 2013-06-14 13:44 /home/superkuh/vsrt_null.log
Fri Jun 14 13:45:36 CDT 2013
-rw-r--r-- 1 superkuh superkuh 488 2013-06-14 13:45 /home/superkuh/vsrt_null.log

((488-8)/4)/60 = 2
```

To know what time a log record corresponds to, take the time from the filename and then add 0.5 seconds * the index of the 4 byte entry in the binary log. This should be possible to write into the until loop so it outputs time instead of just index $i. The below example is a hacky version of my log parser that does just this. Here's an example output.

```
# UTC Epoch     # Beat Freq Bins
1371229380.0,   1.38292284646013e-06
1371229380.5,   1.37606230055098e-06
1371229381.0,   1.374015937472e-06
1371229381.5,   1.366425294691e-06
1371229382.0,   1.35845414206415e-06
1371229382.5,   1.36476899115223e-06
1371229383.0,   1.36480070977996e-06
1371229383.5,   1.36444589315943e-06
1371229384.0,   1.35775212584122e-06
1371229384.5,   1.36395499339415e-06
1371229385.0,   1.35322613914468e-06
1371229385.5,   1.36412847950851e-06
1371229386.0,   1.36531491534697e-06
1371229386.5,   1.3664910056832e-06
1371229387.0,   1.36144888074341e-06
1371229387.5,   1.35596496875223e-06
1371229388.0,   1.35830066483322e-06
1371229388.5,   1.3654090480486e-06
1371229389.0,   1.358990175504e-06
1371229389.5,   1.37098015784431e-06
1371229390.0,   1.387945303577e-06
1371229390.5,   1.38286770834384e-06
1371229391.0,   1.36734763600543e-06
1371229391.5,   1.36036248932214e-06
...
```

```
#!/usr/bin/perl
use DateTime;
use warnings;
use strict;

# The simplest possible gnuplot plot using this program's output.
# ./vsrt_log_timeplot.pl /home/superkuh/vsrt_2013.06.14.12.03.00.log > whee2.log
# gnuplot> plot "./whee2.log" using 1:2 title "VSRT Test" with lines

my $data = '/home/superkuh/vsrt_2013.06.13.12.26.47.log';
my $bytelength = 4;
#my $format = "V"; # oops, not this unsigned 32 bit (little endian)
my $format = "f"; # float
my $num_records;

if ($ARGV[0]) {
        $data = $ARGV[0];
} else {
        print "you need to pass the log file path as an argument.";
        exit;
}

my $dt; # declare datetime variable globally
extracttime($data); # $dt now has date object.

open(LOG,"$data") or die "Can't open log.\n$!";
binmode(LOG);

my $i = 0;
until ( eof(LOG) ) {
        my $record;
        my $decimal;
        read(LOG, $record, $bytelength) == $bytelength
                or die "short read\n";

        $decimal = unpack($format, $record);

        # This is a stupid/fragile way to deal with datetime
        # not having enough precision. It only works if the
        # record to record interval is always 0.5 seconds.
        my $recordtime = $dt->epoch();
        if (0 == $i % 2) {
                printf("$recordtime.0,\t$decimal\n", $decimal);
        } else {
                printf("$recordtime.5,\t$decimal\n", $decimal);
        }

        $dt->add( nanoseconds => 500000000 );
        $i++;
}

sub extracttime {
        my $timestring = shift;
        # /home/superkuh/vsrt_2013.06.13.12.26.47.log
        $timestring =~ /(\d{4}\.\d{2}\.\d{2})\.(\d\d\.\d\d\.\d\d)/;
        my $year_month_day = $1;
        my $time = $2;
        my ($year,$month,$day) = split(/\./, $year_month_day);
        $time =~ s/\./:/g;
        my ($hour,$minute,$second) = split(/:/, $time);

        $dt = DateTime->new(
                year       => $year,
                month      => $month,
                day        => $day,
                hour       => $hour,
                minute     => $minute,
                second     => $second,
                time_zone  => 'America/Chicago',
        );

        $dt->set_time_zone('UTC');
        return 1;
}
```

Now I just have to make up a good gnuplot format and integrate the calls into the perl script.

Computer controlled pointing, mechanical and software differences

MIT Haystack's memo #9 photo of their dish pointing system

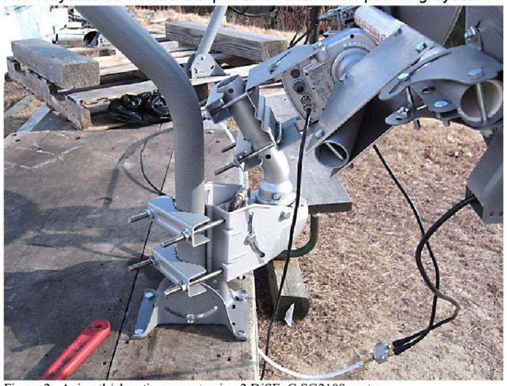
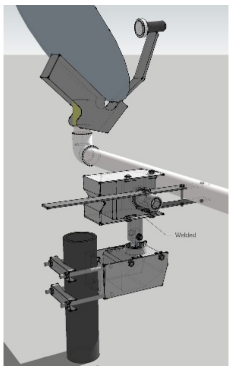


Figure 2. Azimuth/elevation mount using 2 DiSEqC SG2100 motors

Manually repositioning the dishes swamps out the signal of interest as the target leaves the beamwidth. For any decent measurements I need computer controlled pointing. This means the Haystack idea of two coupled Diseqc 1.2 compatible motor positioners mounted one on the other. In their design both dishes are mounted on a single PVC tube hooked to one of the positioners with a metal extension. My satellite dish mounts can't rotate like theirs so I'll have to modify this design a bit. They use a serial relay to "push" the buttons on a physical Diseqc 1.2 motor controller remote. That seemed a bit convoluted to me. I bought a SkyStar2 DVB-S pci card and under linux send raw Diseqc commands out by calling xdipo which accesses the linux DVB interface. It has both a GUI and cli interface. Unfortunately xdipo cannot send through Diseqc switches. I had to add manual motor commands to tune-s2 which did support switches but not manual motor commands. This version which supports manual stepping mode is available at https://github.com/superkuh/tune-s2-stepping.

Another alternative Diseqc motor controller I didn't persue would be using a 192 KHz USB soundcard and the DiSEqC Audio Generator software from Juras-Projects. The documentation for the hardware side of the audio generator is 404 now, but Juras responded to an email of mine with the schematics attached.

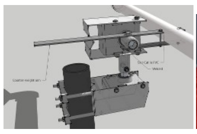


Since the bent motor shafts that came will my motors looked really difficult to drill through I thought I'd use straight hex holed shafts to make everything mechanically simpler. I found http://www.reidsupply.com/sku/HHS-18/ and ordered a couple. Unfortunately my measurements of the dish motor shaft flat-to-flate size were off. The Reid hex holed shaft hole is just a tiny bit too large. This was easily fixed by wrapping a couple turns of masking tape around the shaft to increase the diameter. This is often how fishing rod handles are made. I also encountered this construction technique on Jarrod Kinsey's CO2 laser pages.

The hardest part of all this is drilling an 8mm hole precisely normal to the curved outside surface of the hex hole shaft. The first step is to flatten the area with a hand file. This took me about 10 minutes. I had previously ordered and received two carbide drill bits, one small to sub-drill the intial hole and then one 8mm for the final hole. A drill press and small vice are quired to actually drill the holes. And even then it's really tricky. My first two attempts resulted in holes not quite normal to the surface of the hex flat. I could only use roll or taper pins to secure the shaft. Luckily I bought 2x shafts just in case.

I also had to drill 4 additional 8mm holes in the 2x satellite dish motor mounts to make holes for level mounting instead of at a tilt. The VRST guys got lucky with their sat motor mounts having a long slot.

The diameter of easily available PVC is slightly to small for the dish mounting clamp. This is remedied like the motor shafts; by wrapping wide masking tape to size and optionally epoxy coating/sanding it.



The dish motors used in the VSRT project were Stab HH90. These have come down in cost since the VSRT memos were written and are still widely available.

```
 065    Stab HH90 dish pointing motor #1
 065    Stab HH90 dish pointing motor #2
$130 total
```

In order to control these motors a system to send DISEqC 1.2 commands is needed. The first option would be to faithfully replicate the VSRT implementation. They do it in a rather roundabout way but at least it is tested and known to work with their software. Unfortunately the specific hardware used has become rare, is mostly shipped from overseas, or is expensive.

```
 045    STAB MP01 Positioner Control #1
 045    STAB MP01 Positioner Control #2
 080    WTSSR-M Serial port Solid State Relay
 000    VSRT Java software (windows)

 030    Two 5" x 9" x 0.125" thick steel plates
 020    Two 15" x 1" x 0.125" thick aluminum counterweight arms
 020    Shipping for the metal parts.
```

My chosen method of HH90 motor control is a single DVB-S card under linux with DVB API 5.x w/ my modified tune-s2 and optionally xdipo. This can be combined with a DiSEqC switch to scale to control of multiple motors relatively cheaply. I do sun alt-az position calculation by using a small pysolar python script. I have not yet completed the scripts to turn alt-az positions of the sun at my location into motor step commands. Hopefully I can use some of the USAL fuctions in tune-s2 for that.

```
 066    Skystar 2 HD pci card
 010    DiSEqC switch
 000    tune-s2 and xdipo DVB control software (linux)

 020    .753" Hex-Holed Sleeve
 010    5" wide masking tape
 020    Two 15" x 1" x 0.125" aluminum counterweight arms
```

Both require PVC pipe, tools like drills, 8mm drill bits and smaller sub-drill bit, hand saws, files, and potentially a welder (though liberal J-B Weld would probably work).

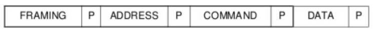
Diseqc switches problems and solutions.

It turns out that xdipo alone cannot deal with motors behind Diseqc switches. This means it can only control one Diseqc motor at once. Controlling two would require 2x Skystar 2 pci cards. Luckily there are other options. CrazyCat's tune-s2 supports Diseqc switches and addressing. It normally only provides for motor commands using the USAL system which isn't too helpful. But I was able to modify the code to support manual motor position commands while retaining the switch support. xdipo could still be used in theory by calling tune-s2 to set the Diseqc switch to the appropriate port/motor and then calling xdipo as normal. But it is easier to just use the modified tune-s2 for everything.

This gutted version of tune-s2 for manual motor commands is available at:

https://github.com/superkuh/tune-s2-stepping

The functions I added are basically just look up arrays with Diseqc bus commands for different steps in the clockwise or counter-clockwise directions. In Diseqc the packets have 4 sections. Check out the Diseqc Bus Functional Specification (pdf) for a better explanation with more detail.



The first, "Framing" byte represent if the command is from the receiver or diseqc device and wether it needs a reply. For my table these are all just "EO" which means it's a packet from the receiver with no response required. Most commands are EO but it goes up to E7.

The second, "Address" specifies which types of Diseqc devices should listen (ex: LNB, switch, motor, polarizer). For motors this is "32"

The third is "Command". This is a huge list of values of which only "68" and "69" are relevant. They are "Drive Motor East" and "Drive Motor West" respectively. The "Command" byte is only relevant to their specific devices specified via the "Address" byte.

The remaining bytes of the packet are "Data" and how they're interpreted depends on the "Command" bytes specifying a specific type of command. For motor movement there are three options. "00" makes the motor turn until a Diseqc stop command is sent. The second mode is positive values for the bytes, "01" to "7F". They represent an amount of time to turn the motor. Or by specifying negative byte values "80" to "FF" the motor is rotated a number of steps. This last is best and detailed in the Positioner Application Note (pdf) with an excerpt below,

The number of steps to make is given by the additional count needed to make the parameter byte reach zero (or overflow to zero if the byte is considered as unsigned). Thus the byte 'FF' (hexadecimal) requests only one step, 'FE' two steps, and for example 'F9' requests 7 steps.

With my motors each step corresponds to about 0.1 deg. Using this information I made up a table of Diseqc packets for each rotation direction.

```
struct dvb_diseqc_master_cmd step_east[] =
{
        { { 0xe0, 0x31, 0x68, 0xFF, 0x00, 0x00 }, 4 },  // Drive Motor West 1 step
        { { 0xe0, 0x31, 0x68, 0xFE, 0x00, 0x00 }, 4 },  // Drive Motor West 2 step
        { { 0xe0, 0x31, 0x68, 0xFD, 0x00, 0x00 }, 4 },  // Drive Motor West 3 step
        { { 0xe0, 0x31, 0x68, 0xFC, 0x00, 0x00 }, 4 },  // Drive Motor West 4 step
        { { 0xe0, 0x31, 0x68, 0xFB, 0x00, 0x00 }, 4 },  // Drive Motor West 5 step
        { { 0xe0, 0x31, 0x68, 0xF6, 0x00, 0x00 }, 4 },  // Drive Motor West 10 step
        { { 0xe0, 0x31, 0x68, 0xEC, 0x00, 0x00 }, 4 },  // Drive Motor West 20 step
        { { 0xe0, 0x31, 0x68, 0xE2, 0x00, 0x00 }, 4 },  // Drive Motor West 30 step
        { { 0xe0, 0x31, 0x68, 0xD8, 0x00, 0x00 }, 4 },  // Drive Motor West 40 step
        { { 0xe0, 0x31, 0x68, 0xCE, 0x00, 0x00 }, 4 },  // Drive Motor West 50 step
        { { 0xe0, 0x31, 0x68, 0x9C, 0x00, 0x00 }, 4 }   // Drive Motor West 100 step
};

struct dvb_diseqc_master_cmd step_west[] =
{
        { { 0xe0, 0x31, 0x69, 0xFF, 0x00, 0x00 }, 4 },  // Drive Motor West 1 step
        { { 0xe0, 0x31, 0x69, 0xFE, 0x00, 0x00 }, 4 },  // Drive Motor West 2 step
        { { 0xe0, 0x31, 0x69, 0xFD, 0x00, 0x00 }, 4 },  // Drive Motor West 3 step
        { { 0xe0, 0x31, 0x69, 0xFC, 0x00, 0x00 }, 4 },  // Drive Motor West 4 step
        { { 0xe0, 0x31, 0x69, 0xFB, 0x00, 0x00 }, 4 },  // Drive Motor West 5 step
        { { 0xe0, 0x31, 0x69, 0xF6, 0x00, 0x00 }, 4 },  // Drive Motor West 10 step
        { { 0xe0, 0x31, 0x69, 0xEC, 0x00, 0x00 }, 4 },  // Drive Motor West 20 step
        { { 0xe0, 0x31, 0x69, 0xE2, 0x00, 0x00 }, 4 },  // Drive Motor West 30 step
        { { 0xe0, 0x31, 0x69, 0xD8, 0x00, 0x00 }, 4 },  // Drive Motor West 40 step
        { { 0xe0, 0x31, 0x69, 0xCE, 0x00, 0x00 }, 4 },  // Drive Motor West 50 step
        { { 0xe0, 0x31, 0x69, 0x9C, 0x00, 0x00 }, 4 }   // Drive Motor West 100 step
};
```

For addressing specific ports of the Diseqc switch tune-s2's normal functions are used. They are called before the motor position commands are sent. Usage of the modified tune-s2 is pretty simple. The only differences are two new cli switches and not needing to give it tuning parameters.

```
-step-east
-step-west
```

They each take any value from 0 to 10 like,

```
./tune-s2 -step-west 0 -committed 1
```

This would cause the satellite dish motor on port 1 of the Diseqc switch to step 1 position counter-clockwise. To send the same command of stepping 1 position counter-clockwise to the other motor,

```
./tune-s2 -step-west 0 -committed 2
```

The stepping argument values 0 through 10 are mapped on a fairly arbitrary set of actual steps. This results from just doing array index look ups in the above packet tables,

```
0->1
1->2
2->3
3->4
4->5
5->10
6->20
7->30
8->40
9->50
10->100
```

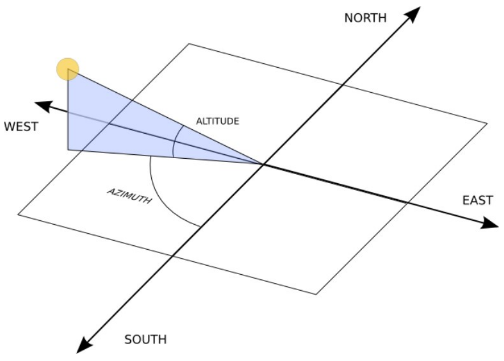
```
http://www.satnigmo.com/2254/how-to-configure-emp-centauri-diseqc-switches/
```

Calculating solar position and using that to decide how many steps to step per axis

Figuring out where the sun is in the sky in terms of an alt-az format is made simple by pysolar. Figuring out how to turn that position into sequences of steps on the motors is much, much harder.

```
#! /usr/bin/env python
import Pysolar
import datetime
d = datetime.datetime.utcnow()
lat = 40.0
long = -90.0
sol_alt = Pysolar.GetAltitude(lat, long, d)
sol_long = Pysolar.GetAzimuth(lat, long, d)
print sol_alt
print sol_long
```

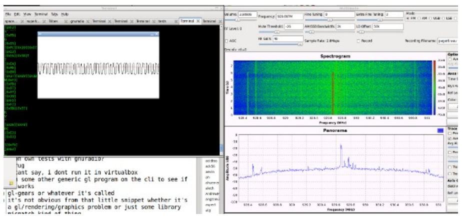
These values are relative to the pysolar reference frame which is given by their diagram,



```
$ ./solpos.py
41.4925424732
-35.8472087363
```

My setup is pointed directly south. So for this example time that means I need to calculate the number of steps required to turn the (top) altitude motor 41.5 *up* from level and the (bottom) azimuth motor 35.8 degrees to the *left* (east).

Decoding Pager Data with multimon and/or gnu radio receivers



```
n own tests with gnuradio?
ug
ant say, i dont run it in virtualbox
 some other generic gl program on the cli to see if
works
gl-gears or whatever it's called
it's not obvious from that little snippet whether it's
a gl/rendering/graphics problem or just some library
```

The hardest part of this is figuring out what kind of pager system you have. I spent a long time trying to decode the local FLEX pager system with decoders that did not support it.

Written by Thomas Sailer, HB9JNX/AE4WA, multimon (multimon.tar.bz2) supports decoding a large number of pager modulations. FLEX is not one of them. Scroll down for FLEX.

On June 29th 2012 dekar told me about his updated fork of multimon, multimonNG, with better error correction and more modulations supported. As of right this instant those on 64bit linux should just use the existing makefile and *not* qmake or qt-creator to compile it. For the windows users (or anyone wanting more info) there's a precompiled version and blog post. Make sure to disable all the demodulators you don't need. I think especially ZVEI is quite spammy. This and this is what pocsag sounds like if you're wondering.

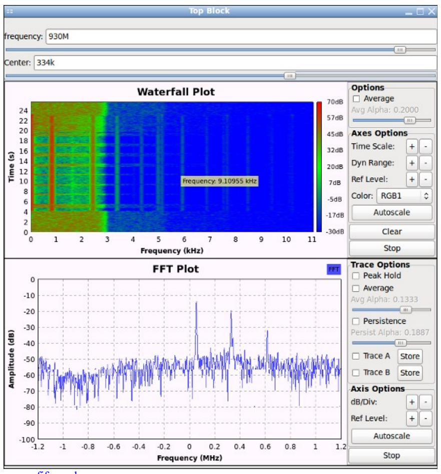
When I originally started playing and wrote this there were only a couple options for rtlsdr receivers to use with the multimon decoder. I used patchvonbraun's multimode to save .wavs and dekar's pager example GRC I modified for OsmoSDR sources linked below for raw, real time decoding. Lately (as of late 2012/13) a large number of receivers have been released that don't depend on GNU Radio. rtl_fm is one and there's an example usage below.

real time decoding rtl_fm

```
rtl_fm -f 930.353e6 -g 100 -s 22050 -l 310 - |multimon -t raw -a POCSAG512 -a POCSAG1200 -a POCSAG2400 -f alpha /dev/stdin
```

real time decoding w/dekar's pager_fifo

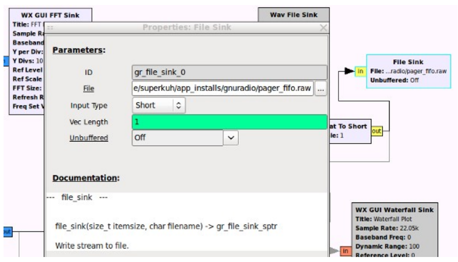
Dekar's multimonNG, a fork with improved error correction, more supported modes, and *nix/osx/windows support. In the screenshots below the signal is not pocsag. I thought it might be zwei but now I'm not so sure it's even pager data. Test samples of pocsag that Dekar links on his blog decode just fine.



pager_fifo_web.grc

```
mkfifo /tmp/pager_fifo.raw
./multimonNG -t raw /tmp/pager_fifo.raw
gnuradio-companion pager_fifo_web.grc
```

In order to decode the pager data in real time you should use a first-in first-out file (fifo). Dekar's pager_fifo is designed to do that but you'll need to set the correct file paths for the File Sink yourself. In the copy downloadable here the File Sink's path is set to "/tmp/pager_fifo.raw". You should be able to run it without editing once you've made that fifo. Make sure to start multimon reading the fifo before you begin GRC and execute the receiver.



Properties: File Sink

Parameters:

ID gr_file_sink_0
File e/superkuh/app_installs/gnuradio/pager_fifo.raw
Input Type Short
Vec Length 1
Unbuffered Off

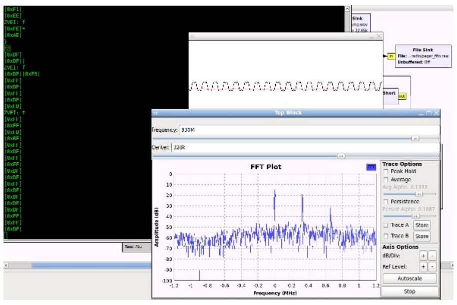
Documentation:

--- file_sink ---

file_sink(size_t itemsize, char filename) -> gr_file_sink_sptr

Write stream to file.

In my personal copy of dekar's pager_fifo the file and audio sinks are enabled while the waterfall, wav, and other sinks are disabled. To enable the disabled (grey) block select them and press 'e' ('d to disable). The audio sink is set to pulseaudio ("pulse").



FLEX Pagers

Unfortunately it turned out my local pagers were all using FLEX, and so not supported by any of the above software. But the procedures might still be useful for someone. Decoding FLEX can be done with the software PDW, but it is windows only. In GNU Radio there is additionally gr-pager, which is supposed to support flex, but many implementation scripts for it are GNU Radio 3.6.5 or older and getting stuff to work with 3.7 requires namespace changes. mothran's flex_hackrf is one of these. Since the rtlsdr receivers can but shouldn't do 3.125 MS/s, like flex_hackrf of uhd_flux, what they use natively for the bandwidth, and so decimation, and pretty much everything else have to be re-written too. I've attempted to start this and you can see a copy here.

A couple days after I wrote the above paragraph zarya came on ##rtlsdr on freenode and mentioned his rtlsdr supprting FLEX decoder written months before. It is easy to use and works great! This script runs at a 250 KS/s sample rate and decodes 12.5 KHz channel only. Internally it uses gnuradio's optfir to generate low pass taps that wide to use witih a frequency xlating FIR filter. It then passes that to gr-pager's flex_demod.

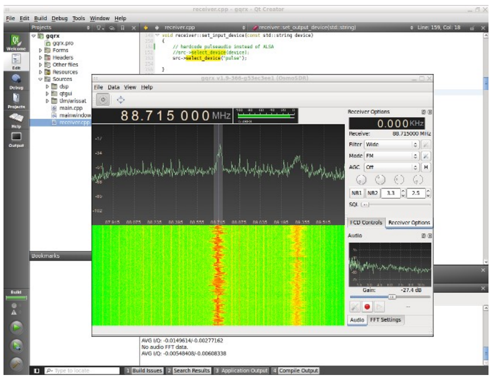
later: argilo (Clayton Smith) has also put together an osmosdr source based gr.pager flex decoder for his GNU Radio tutorial series.

The below output is heavily censored and edited to avoid disclosing or reproducing sensitive information but it gives you an idea of the type of messages.

```
git clone https://github.com/zarya/sdr
cd sdr/receivers/flex/
./rtl_flex_noX.py -f 929.56M --rx-gain=37.2
linux; GNU C++ version 4.6.3; Boost_104800; UHD_003.005.004-149-gc357a16e

No database support
gr-osmosdr v0.1.0-33-g8facbbcc (0.1.1git) gnuradio 3.7.2git-123-g0ded5889
built-in source types: file fcd rtl rtl_tcp uhd hackrf netsdr
Using device #0 Generic RTL2832U SN: 77771111153705700
Found Rafael Micro R820T tuner
Exact sample rate is: 250000.000414 Hz

Setting gain to 20.700000 (from [0.000000, 49.600000])
Using Volk machine: avx_64_mmx_orc
0 929.560|        55|SPN|2900
0 929.560|   5555555|ALN|osoft JDBC type 4 driver for MS SQL Server 2005:FreePoolSize = 24  [03]
0 929.560|       555|ALN|MSN 020 hello Message from NOC PCB. 129
0 929.560|   5555555|ALN| Task: 3322391 Net: 0 Pressman: REDACTED, REDACTED Click here to view the curre [28]
0 929.560| 555555555|ALN|4.Q!T .(-RS(7+*# ~A-!lu+3L:REDACTED:YSGO>T JL*qN>][WOA"$(0?"$8QB, 33*dM:YSG.>][WO<]\(0BK [1
0 929.560|   5555555|ALN|From: REDACTED@REDACTED.com Subject: REDACTED scheduled report: "05:00 Medication Devices" - Scheduled report "05:00 Medication Devices" was sent by REDACTED. <<00medicationdevices-11-23-135-00am.pdf>>  [43]
0 929.560|   5555555|ALN|re| 7291 W 190th St| FD o/s calling a 2nd alarm for a structure fire, Setting up water supply| REDACTED008| 05:02 . .   [97]
0 929.560|   5555555|ALN|Fr/m: 7th.Floor-.REDACTED REDACTED: +Blood Cultures; G+ cocci from yesterday's clinic draw. On Cefepime. Afebrile. New orders? Draw BCs in am? [31]
0 929.560|   5555555|SPN|766087U410 5[[[
0 929.560|   5555555|ALN|REDACTED, REDACTED 9105-2 FYI: Notified by lab stool sample came back positive of C.diff Liculda, RN REDACTED c9q [74]..
```

I tried for over a year before successfully decoding local pager signals. Now that I have I think it is a bad idea. There is far much too much private information in cleartext. I don't plan to try again.

(old) gqrx install notes



```
void receiver::set_input_device(const std::string device)
{
    // hardcode pulseaudio instead of ALSA
    //src->select_device(device);
    src->select_device("pulse");

}
```

Receive: 88.715000 MHz
Filter: Wide
Mode: FM
AGC: Off
NR1 NR2 3.3 2.5
SQL
Gain: -27.4 dB

```
AVG I/Q: -0.0149614/-0.00277162
No audio FFT data.
AVG I/Q: -0.00548408/-0.00608338
```

Read this person's guide instead.

When I wrote this up the original version by csete didn't support the hardware yet but mathis_, phirsch, Hoernchen, and perhaps others I've missed from ##rtlsdr on freenode had added librtlsdr support to gqrx; their repos are still listed by commented out. These days csete has added in rtlsdr support so you can use his original repository.

```
git clone https://github.com/csete/gqrx.git
cd gqrx
# on Ubuntu, sudo apt-get install qtcreator , if you don't have it.
qtcreator gqrx.pro      # press the build button (the hammer)
# Avoid qtcreator doing it manually.
qmake
make

./gqrx
```

Use with Ubuntu 10.04 and distros with old Qt < 4.7

You will almost certainly not get this error. But, someone might, so I'm leaving it here to be indexed.

If you're like me and run an older distribution then your Qt libraries will be out of date and lack a function required for generating the name of the files to be saved when recording.

```
/home/superkuh/app_installs/gnuradio/gqrx/gqrx/qtgui/dockaudio.cpp:100: error: 'currentDateTimeUtc' is not a member of 'QDateTime'
```

Initially I thought it was a qtcreator thing so I tried to get more information by doing it manually,

qmake
make
g++ -c -pipe -O2 -I/usr/local/include/gnuradio -I/usr/local/include -I/usr/local/include -I/usr/local/include/gnuradio -D_REENTRANT -D_REENTRANT -I/usr/include/libusb-1.0 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_NO_DEBUG_OUTPUT -DVERSION="\"0.0\"" -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -I. -o dockaudio.o qtgui/dockaudio.cpp
qtgui/dockaudio.cpp: In member function 'void DockAudio::on_audioRecButton_clicked(bool)':
qtgui/dockaudio.cpp:100: error: 'currentDateTimeUtc' is not a member of 'QDateTime'

make: *** [dockaudio.o] Error 1

To get it to compile on these systems you'll have to do the below. (edit: This little change is now added into phirsch's.)

Ubuntu 10.04 has old Qt libs and gqrx uses a function call not in them. So, while I was waiting for Qt 4.74 to compile I decided to try a hack. I removed that function call with a static string of text. [edit] I later found comparable functions for Qt 4.6 and older.

If you are using qtcreator like the docs suggest you can double click on the error and go to the line. If not, it was in ./Sources/qtgui/dockaudio.cpp replace,

```
void DockAudio::on_audioRecButton_clicked(bool checked)
{
    if (checked) {
        // FIXME: option to use local time
        lastAudio = QDateTime::currentDateTimeUtc().toString("gqrx-yyyyMMdd-hhmmss.'wav'");
```

With something like this.

```
void DockAudio::on_audioRecButton_clicked(bool checked)
{
    if (checked) {
        // FIXME: option to use local time
        // use functions compatible with older versions of Qt.
        lastAudio = QDateTime::currentDateTime().toUTC().toString("gqrx-yyyyMMdd-hhmmss.'wav'");
```

And it'll compile and run correctly on my specific machine.

Compiling LTE Cell Scanner and LTE Tracker on Ubuntu 10.04

Before starting make sure to have a fortran compiler, FFTW, BLAS, and LAPACK libraries installed from the repositories.

```
sudo apt-get install automake autoconf libtool libfftw3-3 libfftw3-dev gfortran libblas3gf libblas-dev liblapack3gf liblapack-dev libatlas-base-dev
```

If you're using 12.04 just follow the instructions on the github page and everything is trivial. For 10.04 (lucid) users the the initial hurdle is cmake. LTE Cell Scanner requires cmake 2.8.8 and Ubuntu 10.04 only has 2.8 the finding of BLAS and LAPACK libraries will fail like,

```
CMake Error at CMakeLists.txt:1 (CMAKE_MINIMUM_REQUIRED):
  CMake 2.8.4 or higher is required.  You are running version 2.8.0.
```

Until you open CMakeList.txt and change the version number on first line to 2.8.0. After fixing that the BLAS and LAPACK issues come in,

```
cmake ..
-- Found ITPP: /usr/lib64/libitpp.so
CMake Error at /usr/share/cmake-2.8/Modules/FindBLAS.cmake:45 (message):
  FindBLAS is Fortran-only so Fortran must be enabled.
Call Stack (most recent call first):
  CMakeLists.txt:29 (FIND_PACKAGE)
```

You can see my installation notes before I figured it out. To fix it I searched for people complaining of similar problems on other projects and then replaced my *system* files with theirs, FindBLAS.cmake.

```
sudo cp /usr/share/cmake-2.8/Modules/FindBLAS.cmake /usr/share/cmake-2.8/Modules/FindBLAS.cmake.bak
sudo cp FindBLAS.cmake /usr/share/cmake-2.8/Modules/
```

LAPACK will also fail this way. I used this arbitray cmake file, http://code.google.com/p/qmcpack/source/browse/trunk/CMake/FindLapack.cmake?r=5383. And this is a local backup in case that disappears.

```
sudo cp /usr/share/cmake-2.8/Modules/FindLAPACK.cmake /usr/share/cmake-2.8/Modules/FindLAPACK.cmake.bak
sudo FindLAPACK.cmake /usr/share/cmake-2.8/Modules/FindLAPACK.cmake
```

After fixing the cmake issues compile and install the latest IT++ (ITPP 4.2). Make sure to completely remove the old ITPP 4.0.7 libraries from the Ubuntu repository. When LTE Cell scanner compiles you can go back and restore the .bak cmake files. The rate of scan is about 0.1 Mhz per 10 seconds.

```
./CellSearch -v -s 751e6 -e 751e6
LTE CellSearch v0.1.0 (release) beginning
  Search frequency: 751 MHz
  PPM: 100
  correction: 1
Found Elonics E4000 tuner
Waiting for AGC to converge...
Examining center frequency 751 MHz ...
Capturing live data
  Calculating PSS correlations
  Searching for and examining correlation peaks...
  Detected a cell!
    cell ID: 414
    RX power level: -17.0733 dB
    residual frequency offset: 43592.8 Hz
  Detected a cell!
    cell ID: 415
    RX power level: -20.8041 dB
    residual frequency offset: 43592.3 Hz
  Detected a cell!
    cell ID: 209
    RX power level: -28.8524 dB
    residual frequency offset: 43581.2 Hz
Detected the following cells:
C: CP type ; P: PHICH duration ; PR: PHICH resource type
CID      fc   foff RXPWR C nRB P  PR CrystalCorrectionFactor
414    751M  43.6k -17.1 N  50 N one 1.0000580496943698439
415    751M  43.6k -20.8 N  50 N one 1.0000580490797574829
209    751M  43.6k -28.9 N  50 N one 1.0000580342133056355
```

Both positive and negative frequency offsets happen, but rarely in the same dongle.

LTE Tracker I haven't used as much yet (recently released) but it is included in the github repository cloned initially and should be compiled as well if you did the above. Check out the authors site for videos of it's use since an ascii paste of the ncurses like interface wouldn't tell you much. But... the start looks like this,

```
./LTE-Tracker -f 751e6
LTE Tracker v1.0.0 (release) beginning
  Search frequency: 751 MHz
  PPM: 120
  correction: 1
Found Rafael Micro R820T tuner
Calibrating local oscillator.
Calibration succeeded!
   Residual frequency offset: -48937.5 Hz
   New correction factor: 0.99993484110674779597
Searcher process has been launched.
```

Slightly altered GNU Radio Companion flowcharts

"The GUI stuff in Gnu Radio was rather an afterthought. Nobody really expected that you'd use it to build actual applications, but rather just use it as a way of making "test jigs" for your signal flows."

This section is my notes on how I made basic examples work, and how I edited those examples in very simple and often broken ways. Also, since gqrx, multimode, and other intergrated receivers came out I don't see any need to update these as things change. **Most of this is very old.**

While there are links to the originals in the summaries, these descriptions are of the versions modified by me; usually just sample rate and GUI stuff. While the sample rate or tuner width I set may be some large number, it'll become obvious what the limits of each other as you scan about and see the signal folding or mirroring. **Using sample rates above 2.4 MS/s with rtlsdr is not recommended. It *does* create aliases all over.** If you're using GNU Radio 3.7 don't even bother trying with any .grc files hosted here.

- FM:
  - patchvonbraun's simple (stereo) fm receiver - harder setup, best reception, best sound, 2.048 MS/s, +-600Khz fine tune
  - lindi's fm receiver easy setup, good reception, good sound, 3.2 MS/s, +-600Khz fine tune
  - superkuh's offset fm receiver - easy setup, okay reception, okay sound, 2.8 MS/s, +-900Khz tune, +-50Khz fine.
  - 2h20's beginner fm (mono) receiver - easy to understand, easy setup, okay sound, 2.8 MS/s, no fine tune
- SSB:
  - OZ9AEC's SSB Receiver - SSB rx and record to disk, seperate playback script. 1 MS/s, +-1k fine tune.

Tips

If it comes with a python file, try that first before generating one from the GRC file. When tuning, make sure to hit enter again if it doesn't work the first time or tunes to the wrong frequency. Always hit autoscale to start, and for FFT displays try using the "average" settings. I have set all audio sinks to "pulse" (pulseaudio) instead of say, "hw:0,0" (ALSA). You might have to change that. To get a list of hardwareuse "aplay -l". That'll show the various cards and devices. Use the format, "hw:X,Y" where "hw:CARD=X,DEV=Y". Some flowcharts have variables for it, others put it directly in the Audio Sink element. If you hear something interesting you can try comparing it to indentified samples from http://www.kb9ukd.com/digital/ or http://hfradio.org.uk/html/digital_modes.html. or the windows program, Signals Analyzer. Check http://www.radioreference.com/ or http://wireless2.fcc.gov/UlsApp/UlsSearch/searchAdvanced.jsp to see what's in the USA area at a given frequency.

Multiple Dongles

There are two ways to specify the use of multiple dongles. The first, correct, way is to set the "Num Channels" in the OsmoSDR Source block to "2" and then specify the device IDs in "Device Arguments" like, "rtl=0 rtl=1". Each specified device is seperate with a space from the previous one.

The not so correct but still working way is to use multiple OsmoSDR Source blocks with "Num Channels" set to "1" and each with it's respective "Device Arguments" field set to "rtl=0" or "rtl=1", or so on.

The OsmoSDR Source block has extensive help files at the bottom of it's properties if you scroll down.

Editing

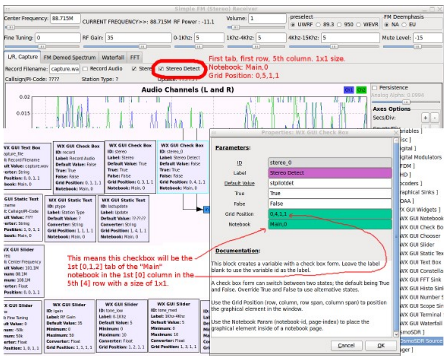
To enable a block, select it and press 'e'. To disable a block select it and press 'd'. When disabled blocks will appear darker gray.

If you open a .grc file and it looks like there are blocks missing (red error highlights and no connections between them) then it is likely the name of the block changed during some GNU Radio update. If your install is more than a month or two old this often happens. Update GNU Radio.

It's easier to type in 1e6 than 1000000 so use scientific notation when you can in variable fields. If you double click on an element in a flowchart it usually includes a helpful "Documentation:" of most the variables to be set at the bottom. The GUI element grid position is a set of two pairs of numbers: "y,x,a,b" where the first pair "y,x," is position (y row, x column) and "a,b" is the span of the box. If you enter a Grid Position and it overlaps with another element it'll turn red and report the error and where the origin is of the element it overlaps with.

```
Use the Grid Position (row, column, row span, column span) to position the graphical element in the window.
```

The tab effect is done with notebooks.



Parameters:

ID stereo_0
Label Stereo Detect
Default Value stpilotdet
True True
False False
Grid Position 0,4,1,1
Notebook Main,0

Documentation:

This block creates a variable with a check box form. Leave the label blank to use the variable id as the label.

A check box form can switch between two states; the default being True and False. Override True and False to use alternative states.

Use the Grid Position (row, column, row span, column span) to position the graphical element in the window.

Use the Notebook Param (notebook-id, page-index) to place the graphical element inside of a notebook page.

For RTL2832 Source the minimum sample rate is ~800KS/s, it's gr-baz(?) and generally not updated. Use OsmoSDR source. It's under "OsmoSDR", not "Sources" on the right panel). It has a 1MS/s minimum sample rate. It's not recommended to use sample rates above 2.4M.

In older versions of gr-osmosdr and rtl-sdr I think automagic gain control (AGC) was on all the time so you didn't have to set the gain explicitly in the source in GRC. New versions require that and also require setting the chan 0. freq to something.

The dongles seem to have noise at their 0Hz center frequency so the best performance is from selecting a band 100-200Khz offset from the center (depending on signal type). patchvonbraun's simple_fm_rcv is a great example of that.

---

patchvonbraun's simple_fm_rcv

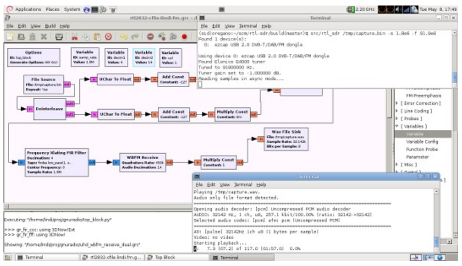
(this summary is outdated) The best sounding software I've found for listening to FM is patchvonbraun's Simple FM (Stereo) Receiver. I don't think it is very simple; it includes many advanced FM specific features like extraction of the 19k (pilot) tone next to some commercial FM broadcasts. It used to do RDS, I hear, and older versions checked into CGRAN still have it, but it is removed for simplicity in this version.

```
svn co https://www.cgran.org/svn/projects/simple_fm_rcv
cd simple_fm_rcv/
cd trunk
less README
make
make install
## it'll install to ~/bin/, so I use ~/superkuh/bin below
set PYTHONPATH=/usr/local/lib/python2.6/dist-packages:/home/superkuh/bin
# run the python script
python simple_fm_rcv.py
# or edit it
gnuradio-companion simple_fm_rcv.grc
```

---

lindi's FM receiver

Original: http://lindi.iki.fi/lindi/gnuradio/rtl2832-cfile-lindi-fm.grc , this was an example posted to ##rtlsdr by lindi. It used a file source which was decoded to wav and saved to disk. Seen in the screenshot above.

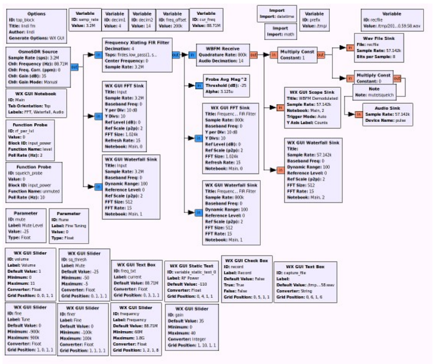


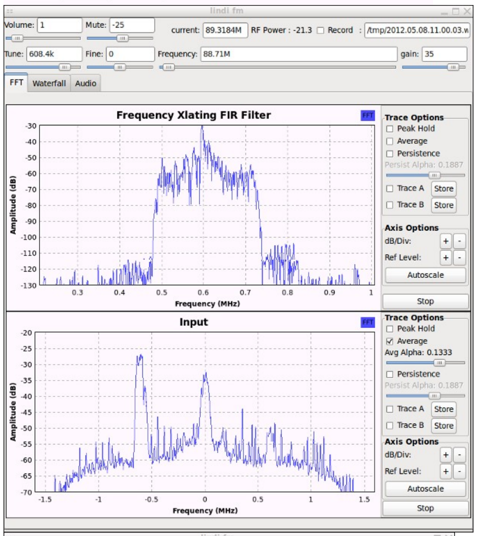
```
(sid)oregano:~/scm/rtl-sdr/build(master)$ src/rtl_sdr /tmp/capture.bin -s 1.8e6 -f 91.9e6
Found 1 device(s):
  0:  ezcap USB 2.0 DVB-T/DAB/FM dongle

Using device 0: ezcap USB 2.0 DVB-T/DAB/FM dongle
Found Elonics E4000 tuner
Tuned to 91900000 Hz.
Tuner gain set to -1.000000 dB.
Reading samples in async mode...
```

```
Playing /tmp/capture.wav.
Audio only file format detected.
==========================================================================
Opening audio decoder: [pcm] Uncompressed PCM audio decoder
AUDIO: 32142 Hz, 1 ch, u8, 257.1 kbit/100.00% (ratio: 32142->32142)
Selected audio codec: [pcm] afm: pcm (Uncompressed PCM)
==========================================================================
AO: [pulse] 32142Hz 1ch u8 (1 bytes per sample)
Video: no video
Starting playback...
A:   7.3 (07.2) of 117.0 (01:57.0)  0.0%
```

Modified: http://superkuh.com/rtl2832-cfile-lindi-fm_edit.grc , had a frontend GUI and an increased sample rate. Right now the rate of the audio files saved out is... not very useful. But it sounds fine. Seen below.

3.2 MS/s field of view, tune +-900Khz



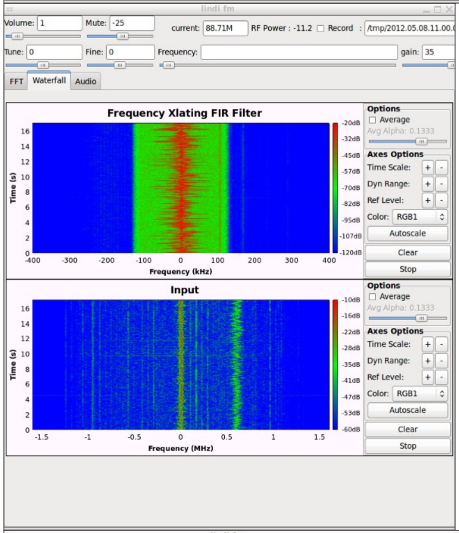


lindi fm

Volume: 1 Mute: -25 current: 89.3184M RF Power : -21.3 Record : /tmp/2012.05.08.11.00.03.w

Tune: 608.4k Fine: 0 Frequency: 88.71M gain: 35

FFT Waterfall Audio

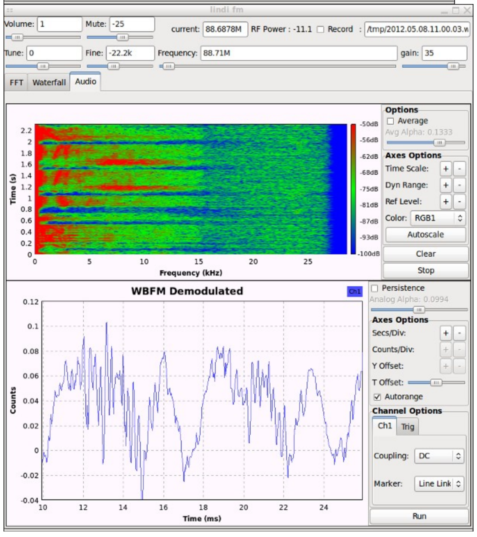


lindi fm

Volume: 1 Mute: -25 current: 88.71M RF Power : -11.2 Record : /tmp/2012.05.08.11.00.0

Tune: 0 Fine: 0 Frequency: gain: 35

FFT Waterfall Audio



lindi fm

Volume: 1 Mute: -25 current: 88.6878M RF Power : -11.1 Record : /tmp/2012.05.08.11.00.03.w

Tune: 0 Fine: -22.2k Frequency: 88.71M gain: 35

FFT Waterfall Audio

2h20's simple fm receiver

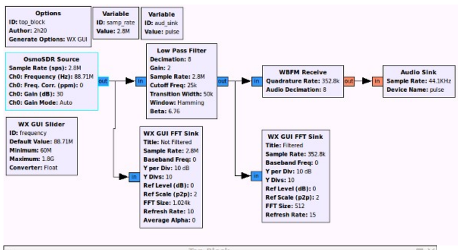
2.8 MS/s field of view, no fine tuning.

2h20 made available, with thorough explaination, a bare bones FM (mono) receiver to learn how to use GNU Radio. This was the first one I managed to get to work. Because the original h202's uses the RTL2832 Source and not the OsmoSDR Source you might experience tuner crashes if you scan too quickly. Make sure to un/replug in the dongle after these. It's best just to enter the frequency as a number.

[Be aware this section of this page was written many months ago when rtl-sdr was different and I had little idea of what I was doing. xzero has since manually added signal seeking to this example.]

My edit of 2h20's simple receiver does not add much, but I did replace the RTL2832 source with an OsmoSDR source to avoid tuner crashes. I also increased the sample rate to 2.8MS/s (to see more spectrum) and then increased the decimation in the filter from 4 to 8 to compensate so everything still decodes/sounds right. I also remove the superfluous throttle block.

- Modified 2h20's Mono FM Receiver (.grc)



Options
ID: top_block
Author: 2h20
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 2.8M

Variable
ID: aud_sink
Value: pulse

OsmoSDR Source
Sample Rate (sps): 2.8M
Ch0: Frequency (Hz): 88.71M
Ch0: Freq. Corr. (ppm): 0
Ch0: Gain (dB): 30
Ch0: Gain Mode: Auto

Low Pass Filter
Decimation: 8
Gain: 2
Sample Rate: 2.8M
Cutoff Freq: 25k
Transition Width: 50k
Window: Hamming
Beta: 6.76

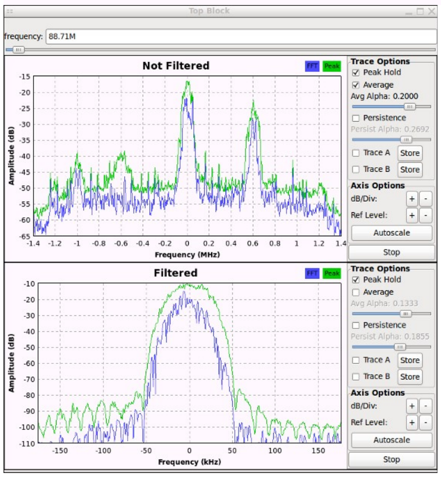
WBFM Receive
Quadrature Rate: 352.8k
Audio Decimation: 8

Audio Sink
Sample Rate: 44.1KHz
Device Name: pulse

WX GUI Slider
ID: frequency
Default Value: 88.71M
Minimum: 60M
Maximum: 1.8G
Converter: Float

WX GUI FFT Sink
Title: Not Filtered
Sample Rate: 2.8M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 10
Average Alpha: 0

WX GUI FFT Sink
Title: Filtered
Sample Rate: 352.8k
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 512
Refresh Rate: 15


Top Block
frequency: 88.71M

Not Filtered — Trace Options: Peak Hold [✓], Average [✓], Avg Alpha: 0.2000, Persistence [ ], Persist Alpha: 0.2692, Trace A [ ], Trace B [ ]

Filtered — Trace Options: Peak Hold [✓], Average [ ], Avg Alpha: 0.1333, Persistence [ ], Persist Alpha: 0.1855, Trace A [ ], Trace B [ ]

---

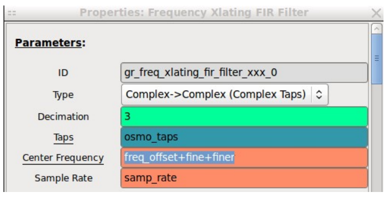
my offset tuning + recording example

2.8 MS/s field of view, +-900Khz tuning, +-50Khz fine.

This takes parts from a bunch of the other example receivers and repurposes them in presumably incorrect but seemingly working ways. It is a basic example of how to offset the tuner 200khz away from the center to avoid the noise there. I started with 2h20's simple tuner's GUI framework and removed almost all of the content. I copied, with inaccurate trial and error changes of sample rate and filter offset, sections of the offset tuning and other advanced bits from simple_fm_rcv and wfm_rx.grc. The tuner is tuned +200Khz. The freq_xlating filter is tuned +200Khz. The the bandpass filter is specified in a variable,

```
firdes.complex_band_pass(1.0,1.024e6,-95e3,95e3,45e3,firdes.WIN_HAMMING,6.76)
```

The net result is that the frequency of interest comes out of the tuner 200Khz below DC, and the freq_xlater "lifts it up" by 200Khz, and then it's bandpassed.


Properties: Frequency Xlating FIR Filter

Parameters:

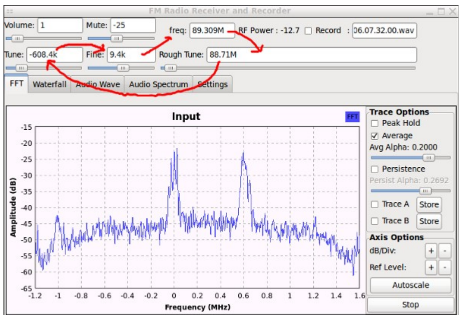
ID	gr_freq_xlating_fir_filter_xxx_0
Type	Complex->Complex (Complex Taps)
Decimation	3
Taps	osmo_taps
Center Frequency	freq_offset+fine+finer
Sample Rate	samp_rate

I also blindly copied the RF power display, a toggle for saving the audio files out to disk, and a +-900khz tuning slider from other receivers. I added a second 'fine' tune +-50Khz. This is done by setting the frequency of the Xlating FIR filter to,

```
freq_offset+fine+finer
```

where freq_offset is the frequency offset from center (200Khz in this case), fine is the ID of a wx gui slider for regular tuning, and finer is the same for fine tuning. In order for the frequency display to show the proper value it was correspondingly set to a variable ID cur_freq,

```
frequency-fine-finer
```


FM Radio Receiver and Recorder
Volume: 1 Mute: -25 freq: 89.309M RF Power : -12.7 Record [ ] : 36.07.32.00.wav
Tune: -608.4k Fine: 9.4k Rough Tune: 88.71M

Input — Trace Options: Peak Hold [ ], Average [✓], Avg Alpha: 0.2000, Persistence [ ], Persist Alpha: 0.2692, Trace A [ ], Trace B [ ]

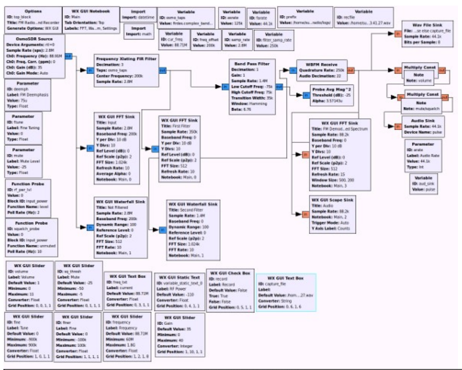
I also made the current frequency display a editable text field so you can tune, copy, and paste. There are good examples of how notebook positioning works and includes simple scripting examples for the file field. This flowchart is simple enough to learn from but includes many elements pulled out of the very complex simple_fm_rcv from patchvonbraun. Without his explanation of the offset process I wouldn't have figured it out. All blocks are layed out by type and GUI elements in the same order as they appear when run. This should help you figure out Grid and Notebook positioning.

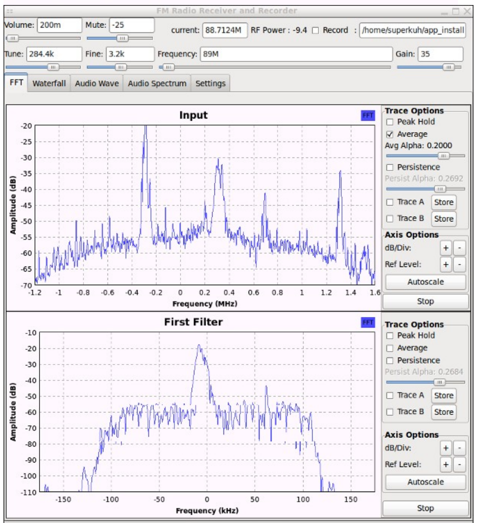
The sound is only "okay". I think the signal is being clipped off at the edges a little bit. I am not sure if it is required to install patchvonbraun's simple_fm_rcv to use this, I do use some of his custom filter stuff.

Usage

Use the Waterfall for scanning through channels. Once located, look at the offset from 0 on the bottom display. Use that to set the tuning (and fine) slider and wiggle it till you get the signal crossing the band in the "Second Filter" top display. Switch to FFT view and look at the bottom "First Filter" display, use tuning and fine tuning to center the peak on the "First Filter" display. Or the other way around. It's personal preference. Ignore the noise you see at higher frequencies (900Mhz) at +0.2Mhz baseband. Although sometimes it gets folded in depending on tuning.

- superkuh's FM w/offset tuning, fine tuning, and recording (.grc)

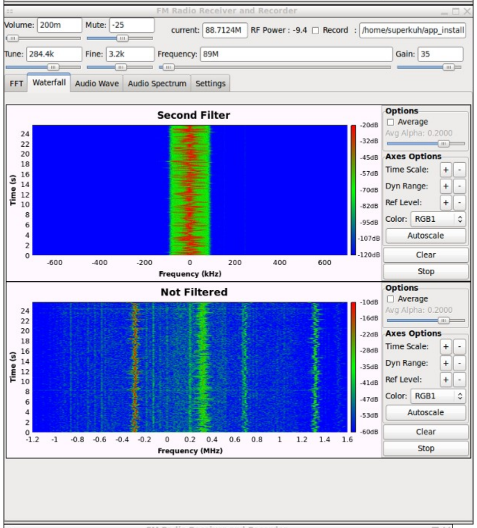



FM Radio Receiver and Recorder

Volume: 200m Mute: -25 current: 88.7124M RF Power : -9.4 ☐ Record : /home/superkuh/app_install

Tune: 284.4k Fine: 3.2k Frequency: 89M Gain: 35

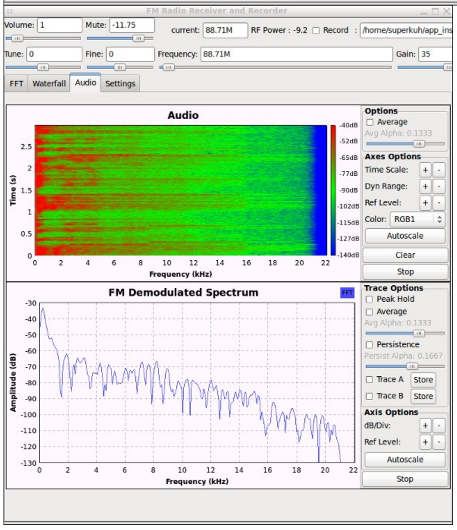
FFT | Waterfall | Audio Wave | Audio Spectrum | Settings


FM Radio Receiver and Recorder

Volume: 200m Mute: -25 current: 88.7124M RF Power : -9.4 ☐ Record : /home/superkuh/app_install

Tune: 284.4k Fine: 3.2k Frequency: 89M Gain: 35

FFT | Waterfall | Audio Wave | Audio Spectrum | Settings


FM Radio Receiver and Recorder

Volume: 1 Mute: -11.75 current: 88.71M RF Power : -9.2 ☐ Record : /home/superkuh/app_ins

Tune: 0 Fine: 0 Frequency: 88.71M Gain: 35

FFT | Waterfall | Audio | Settings

SSB Receiver and data Recorder

Created by Alexandru Csete OZ9AEC the notes say, "Simple SSB receiver prototype". This comes from the GNU Radio GRC examples repository over at https://github.com/csete/gnuradio-grc-examples/tree/master/receiver

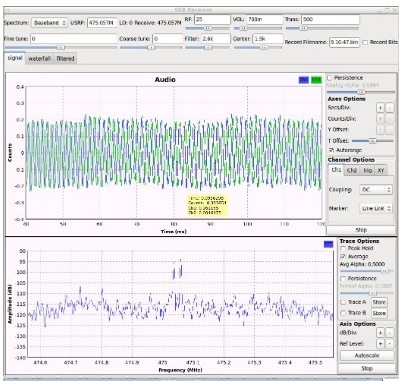
```
git clone https://github.com/csete/gnuradio-grc-examples.git
```

I changed the way it saves samples for the sister decoder program by adding automatic generation of file names and an on/off tickbox toggle for recording. You might want to change the default directory by editing the variable "prefix". The key was

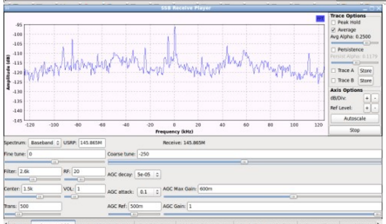
```
"/dev/null" if record == False else capture_file
```

in the File Sink 'file' field. I also changed the GUI so it was easier to find signals. Use the saved .bin files with ssb_rx_play to hear. Jumping around in frequency is a lot smoother when reading from disk instead of the dongle.

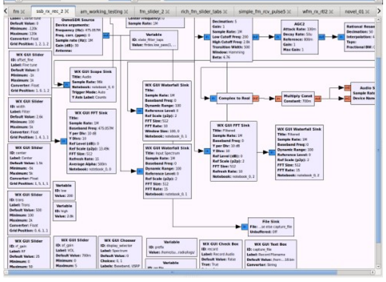
- Modified OZ9AEC SSB Receiver (.grc)
- ssb_rx_play (.grc)



Spectrum: Baseband USRP: 475.057M LO: 0 Receive: 475.057M RF: 25 VOL: 700m Trans: 500
Fine tune: 0 Coarse tune: 0 Filter: 2.6k Center: 1.5k Record Filename: 9.10.47.bin Record Bits



Spectrum: Baseband USRP: 145.865M Receive: 145.865M
Fine tune: 0 Coarse tune: -250
Filter: 2.6k RF: 20 AGC decay: 5e-05
Center: 1.5k VOL: 1 AGC attack: 0.1 AGC Max Gain: 600m
Trans: 500 AGC Ref: 500m AGC Gain: 1



How to use rtlsdr and hackrf on a *fresh* odroid-u3 with ubuntu

```
# rtlsdr on odroid-u3
sudo /usr/local/bin/root-utility.sh # resize partition first
sudo apt-get update
sudo apt-get upgrade
sudo apt-get -f dist-upgrade
# at this point you'll get a kernel that won't work with lots of modules (like NFS)
# use root-utility.sh again to update the kernel and it'll work.
sudo /usr/local/bin/root-utility.sh
# Install the required programs and libs
sudo apt-get install git cmake libusb-1.0-0 libusb-1.0-0-dev
# For some reason libusb.h didn't get installed so I did it manually
sudo apt-get install unp # to unpack the .deb
apt-get download libusb-1.0-0-dev
unp data.tar.xz
unp libusb-1.0-0-dev_2%3a1.0.17-1ubuntu2_armhf.deb
sudo cp ./usr/include/libusb-1.0/libusb.h /usr/include/libusb-1.0/libusb.h
# Then I could start compilation of rtlsdr
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr/
mkdir build
cd build
# The lib dir is for odroid-u3 weirdness. The _FILE_OFFSET_BITS is for writing >2GB files without pipes.
cmake ../ -DCMAKE_INSTALL_PREFIX=/usr -DINSTALL_UDEV_RULES=ON -DLIB_DIR=/usr/lib/arm-linux-gnueabihf -D_FILE_OFFSET_BITS=64
make
sudo make install
sudo su -
ldconfig

#hackrf one on odroid-u3, do all the non-rtlsdr stuff above then,
git clone git://github.com/mossmann/hackrf.git
cd hackrf/host
mkdir build
cd build
# The lib dir is for odroid-u3 weirdness. The _FILE_OFFSET_BITS is for writing >2GB files without pipes.
cmake ../ -DINSTALL_UDEV_RULES=ON -DLIB_DIR=/usr/lib/arm-linux-gnueabihf -D_FILE_OFFSET_BITS=64
make
sudo make install
sudo su -
ldconfig
```

2020-10-26: This change is just to trigger change detection bots. Bork bork bork.

[comment on this post] Append "/**@say**/your message here" to the URL in the location bar and hit enter.