rtl-sdr
Turning USD 20 Realtek DVB-T receiver into a SDR

Harald Welte <laforge@gnumonks.org>

gnumonks.org
hmw-consulting.de
sysmocom GmbH

June 12, FreedomHEC, Taipei

Outline

About the speaker

- Using + toying with Linux since 1994
- Kernel / bootloader / driver / firmware development since 1999
- IT security expert, focus on network protocol security
- Former core developer of Linux packet filter netfilter/iptables
- Board-level Electrical Engineering
- Always looking for interesting protocols (RFID, DECT, GSM)
- OpenEXZ, OpenPCD, Openmoko, OpenBSC, OsmocomBB, OsmoSGSN

Disclaimer

- This talk is not about the Linux kernel
- This talk is not about consumer mass market
- It's about turning a single-purpose device into many more features
- ... and to illustrate the creativity unleashed when hardware / chipset makers don't lock their devices down

Traditional Radio

- uses hardware-based receiver + demodulator
- uses analog filtering with fixed filters for given application
- recovers either analog signal or digitizes demodulated bits
- has not changed much in close to 100 years of using radio waves for communications

Software Defined Radio (SDR)

- uses a more or less classic radio fronted / tuner to down-convert either to IF or to baseband
- uses a high-speed ADC to digitize that IF or baseband signal
- uses digital signal processing for filtering, equalization, demodulation, decoding

SDR advantages vs. traditional radio

- more flexibility in terms of communication system
- as long as tuner input frequency, ADC sampling rate and computing power are sufficient, any receiver can be implemented in pure software, without hardware changes
- this is used mostly by military (JTRS, SCA) and commercial infrastructure equipment (e.g UMTS NodeB / LTE eNodeB)

SDR technology in consumer electronics

- lots of consumer devices already implement SDR technology
    - GSM/UMTS/LTE baseband processor in mobile phones
    - WiFi, Bluetooth, GPS
- flexibility of such implementations is restricted to manufacturer, as low-level access to DSP code and/or raw samples is not intended / documented / activated
- user is locked out from real benefits and flexibility of SDR

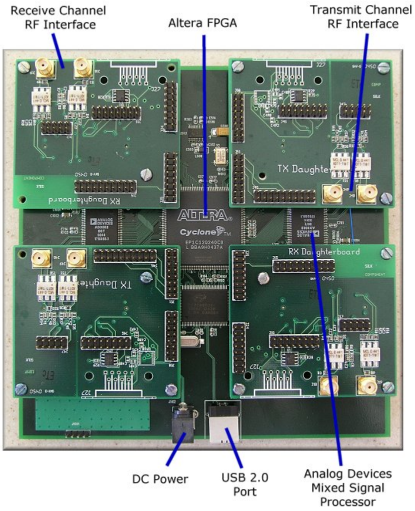
Existing SDR hardware marketed as SDR

- regular consumer-electronics SDR don't provide low-level access or documentation
- military / telco grade SDR device are way too expensive (five-digit USD per unit)
- Ettus developed the famous USRP family (four-digit USD per unit). Used a lot in education + research
- Even lower-cost devices now like Fun Cube Dongle Pro (FCDP) or OsmoSDR (three-digit USD per unit)

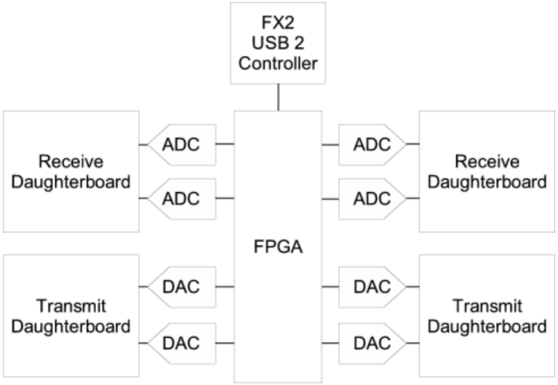
Universal Software Radio Peripheral

- A general-purpose open-source hardware SDR
  - Schematics and component placement information public
- Generally available to academia, professional users and individuals
- Modular concept
  - Mainboard contains Host PC interface and baseband codec
  - Daughter boards contain radio frontend with rf up/downconverter
- Big step forward in pricing, but still not affordable for everyone
  - USD 700 for mainboard
  - frontends from USD 75 to USD 250

USRP Circuit Board Photograph


Receive Channel RF Interface
Altera FPGA
Transmit Channel RF Interface
DC Power
USB 2.0 Port
Analog Devices Mixed Signal Processor

USRP Block Diagram



USRP technical specs

- 4× 12 bit ADCs @ 64 MS/s (digitize band of up to 32 MHz)
- 4× 14 bit DACs @ 128 MS/s (useful output freq from DC...44 MHz)
- 64× Digital I/O ports, 16 to each daughter-board
- The USRP mainboard has 4 slots for daughter-boards (2 Rx, 2 Tx)
- transceiver frontends occupy 2 slots (1 Rx, 1 Tx)

The second generation USRP: USRP2

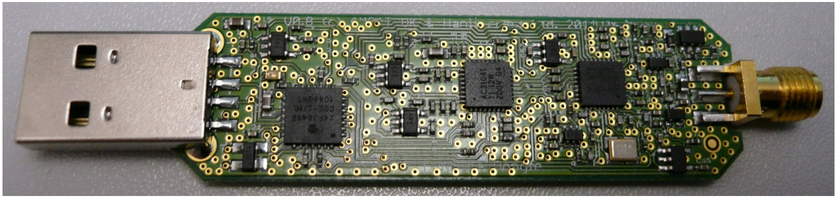
Main differences from USRP1

- Gigabit Ethernet replacing USB2 (full-duplex, 1 GBit/sec)
- 25 MHz of instantaneous RF bandwidth
- Xilinx Spartan 3-2000 FPGA
- $2\times$ 100 MHz 14 bit ADC
- $2\times$ 400 MHz 16 bit DAC
- 1 MByte high-speed SRAM
- Clock can be locked to external 10 MHz reference
- 1 pulse-per-second input (GPS clock conditioning)
- FPGA configuration can be stored on SD card
- Stand-alone operation without host PC
- Multiple systems can be connected for MIMO
- Daughterboards compatible with USRP(1)

Fun Cube Dongle Pro (2010)

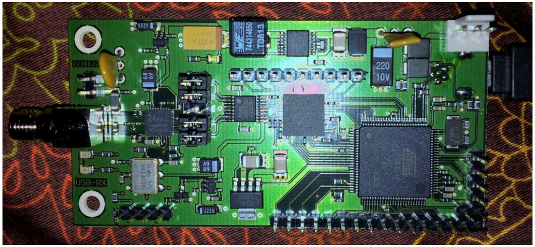
- 64 MHz to 1700 Mhz USB SDR receiver (193 USD)
- limited to 96 kHz I/Q baseband sampling
- great for amateur radio and TETRA, but most other communications systems (like GSM introduced in 1992) use wider band-widths
- great progress in terms of size and cost, but much more limited than USRP
- Hardware design and firmware sadly are proprietary

Fun Cube Dongle Pro (2010)



OsmoSDR (2012)

- small, low-power / low-cost USB SDR hardware (225 USD)
- higher bandwidth than FunCubeDonglePro (1.2 MHz / 14bit)
- much lower cost than USRP, but more expensive than FCDP
- Open Hardware (schematics), software (FPGA, firmware)



Gnuradio architecture

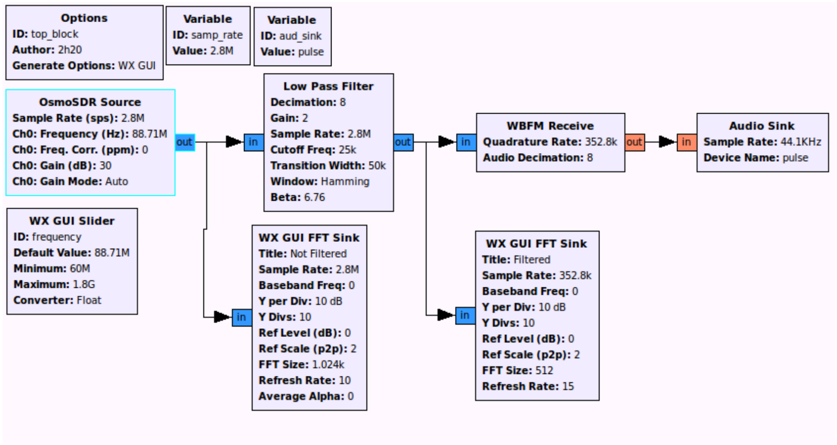
- Philosophy: Implement SDR not as hand-crafted special-case hand-optimized assembly code in some obscure DSP, but on a general purpose PC
    - with modern x86 systems at multi-GHz clock speeds and with many cores this becomes feasible
    - of course way too expensive for a mass-produced product, but very suitable for research, teaching and rapid prototyping
- Implement various signal processing elements in C++
    - assembly optimized libraries for low-level operations
    - provide python bindings for all blocks
- Python script to define interaction, relation, signal routing between blocks

Gnuradio blocks and flow graphs

block	represents one element of signal processing • filters, adders, transforms, decoders, hardware interfaces
flow graph	defines routing of signals and interconnection of blocks • Think of it as the *plumbing* between the blocks

Data passing between blocks can be of any C++ data type

GRC flow graph for Wideband FM



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

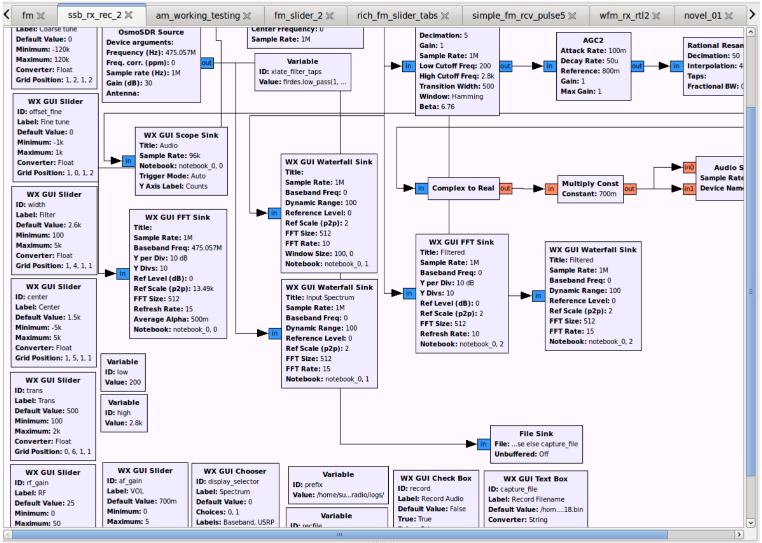
Audio Sink
Sample Rate: 44.1KHz
Device Name: pulse

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

GRC flow graph for SSB



Label: Coarse tune
Default Value: 0
Minimum: -120k
Maximum: 120k
Converter: Float
Grid Position: 1, 2, 1, 2

OsmoSDR Source
Device arguments:
Frequency (Hz): 475.057M
Freq. corr. (ppm): 0
Sample rate (Hz): 1M
Gain (dB): 30
Antenna:

Center Frequency: 0
Sample Rate: 1M

Variable
ID: xlate_filter_taps
Value: firdes.low_pass(1, ...

Decimation: 5
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: 200
High Cutoff Freq: 2.8k
Transition Width: 500
Window: Hamming
Beta: 6.76

AGC2
Attack Rate: 100m
Decay Rate: 50u
Reference: 800m
Gain: 1
Max Gain: 1

Rational Resan
Decimation: 50
Interpolation: 4
Taps:
Fractional BW: (

WX GUI Slider
ID: offset_fine
Label: Fine tune
Default Value: 0
Minimum: -1k
Maximum: 1k
Converter: Float
Grid Position: 1, 0, 1, 2

WX GUI Scope Sink
Title: Audio
Sample Rate: 96k
Notebook: notebook_0, 0
Trigger Mode: Auto
Y Axis Label: Counts

WX GUI Waterfall Sink
Title:
Sample Rate: 1M
Baseband Freq: 0
Dynamic Range: 100
Reference Level: 0
Ref Scale (p2p): 2
FFT Size: 512
FFT Rate: 10
Window Size: 100, 0
Notebook: notebook_0, 1

Complex to Real

Multiply Const
Constant: 700m

Audio S
Sample Rate
Device Name

WX GUI Slider
ID: width
Label: Filter
Default Value: 2.6k
Minimum: 100
Maximum: 5k
Converter: Float
Grid Position: 1, 4, 1, 1

WX GUI FFT Sink
Title:
Sample Rate: 1M
Baseband Freq: 475.057M
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 13.49k
FFT Size: 512
Refresh Rate: 15
Average Alpha: 500m
Notebook: notebook_0, 0

WX GUI FFT Sink
Title: Filtered
Sample Rate: 1M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 512
Refresh Rate: 10
Notebook: notebook_0, 2

WX GUI Waterfall Sink
Title: Filtered
Sample Rate: 1M
Baseband Freq: 0
Dynamic Range: 100
Reference Level: 0
Ref Scale (p2p): 2
FFT Size: 512
FFT Rate: 15
Notebook: notebook_0, 2

WX GUI Slider
ID: center
Label: Center
Default Value: 1.5k
Minimum: -5k
Maximum: 5k
Converter: Float
Grid Position: 1, 5, 1, 1

WX GUI Waterfall Sink
Title: Input Spectrum
Sample Rate: 1M
Baseband Freq: 0
Dynamic Range: 100
Reference Level: 0
Ref Scale (p2p): 2
FFT Size: 512
FFT Rate: 15
Notebook: notebook_0, 1

Variable
ID: low
Value: 200

Variable
ID: high
Value: 2.8k

WX GUI Slider
ID: trans
Label: Trans
Default Value: 500
Minimum: 100
Maximum: 2k
Converter: Float
Grid Position: 0, 6, 1, 1

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

WX GUI Slider
ID: rf_gain
Label: RF
Default Value: 25
Minimum: 0
Maximum: 50

WX GUI Slider
ID: af_gain
Label: VOL
Default Value: 700m
Minimum: 0
Maximum: 5

WX GUI Chooser
ID: display_selector
Label: Spectrum
Default Value: 0
Choices: 0, 1
Labels: Baseband, USRP

Variable
ID: prefix
Value: /home/su...radio/logs/

Variable
ID: recfile

WX GUI Check Box
ID: record
Label: Record Audio
Default Value: False
True: True

WX GUI Text Box
ID: capture_file
Label: Record Filename
Default Value: /hom....18.bin
Converter: String

Gnuradio sinks and sources

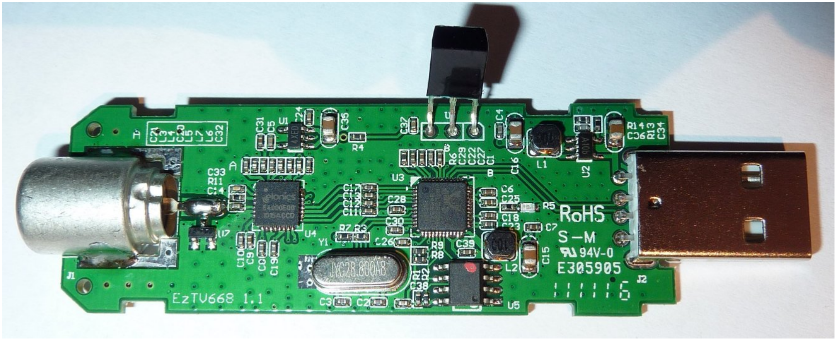
sink	special block that consumes data
hardware sinks	USRP, Sound card, COMEDI
software sinks	Scope UI, UDP port, Video card
source	special block that sources data
hardware sources	USRP, Sound card, COMEDI
software sources	Noise source, File, UDP port

Every flow graph needs at least one sink and one source!

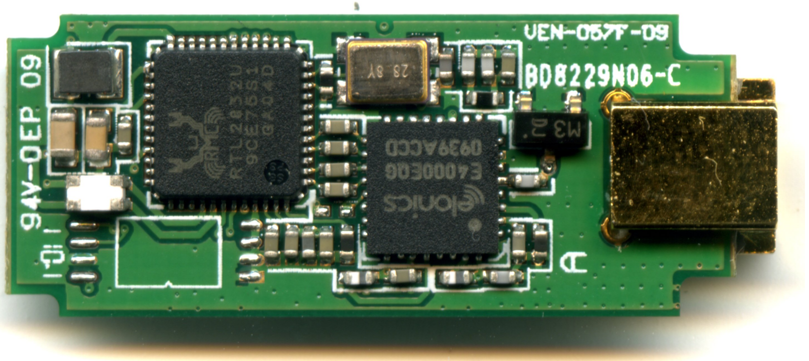
Realtek RTL2832U based DVB-T receivers

- Realtek RTL2832U based DVB-T receivers are cheaply available on the market (USD 20)
- RTL2832U implements ADC, DVB-T demodulator and high-speed USB device
- Normal mode of operation includes full DVB-T receiver inside RTL2832U hardware and only sends MPEG2-TS via USB
- Realtek released GPL-licensed Linux kernel driver for watching TV (not mainline style, but at least GPL)
- Realtek released limited manual to V4L developers

RTL2832U based devices: EzTV 668



RTL2832U based devices: Hama nano1



RTL2832U based devices

- more than 20 different devices from various vendors
- Brand names include ezcap, Hama, Terratec, Compro, GTek, Lifeview, Twintech, Dexatek, Genius, Gigabyte, Dikom, Peak, Sveon
- all based on the identical RTL2832U reference design
- only major difference is mechanical shape/size and silicon tuner used. Best tuner we know is Elonics E4000 (high frequency range)

RTL2832U FM and DAB radio

- Some people realized certain windows drivers for RTL2832U based products implement FM Radio, others even DAB
- Linux driver had no FM radio or DAB support
- Realtek-disclosed limited data sheet didn't mention it either
- Sniffing USB protocol on Windows revealed that raw samples are passed from ADC over USB, FM or DAB demodulation happens in x86 software.
- Realtek didn't provide documentation or source code for this on request

RTL2832U towards rtl-sdr

- Reverse engineering the USB protocol and replaying certain commands from custom libusb based code was able to trigger the raw sample transmission
- remaining Realtek driver provided information on how to use the I2C controller to control the tuner frontend
- Harald already developed Elonics E4000 driver for osmo-sdr, which could be re-cycled
- Tuning to arbitrary frequencies allows digitizing spectrum of any communications system within the tuner range

RTL2832U towards rtl-sdr

- *librtlsdr* contains the major part of the driver
- *rtl_sdr* command line capture tool
- *gr-osmosdr* gnuradio source block
- Homepage: http://sdr.osmocom.org/trac/wiki/rtl-sdr
- libusb is portable, there even are pre-built windows binaries

rtl-sdr software support

- gnuradio (of course), using gr-osmosdr
- gr-pocsag (POCSAG pagers)
- simple_fm_rcv (FM receiver)
- python-librtlsdr / pyrtlsdr (generic python bindings)
- QtRadio
- qgrx
- rtl_fm
- SDR#
- gr-air-modes
- tetra_demod_fft (TETRA radio)
- airprobe (GSM receiver)

Free Software SDR Receivers

Full FOSS receivers/demodulators/decoders available for

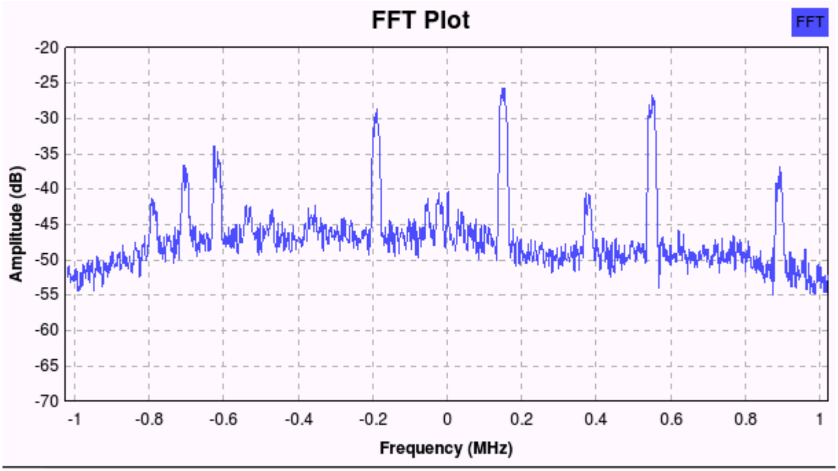
- Old analog modes like AM/FM/WFM/SSB
- DAB (Digital Audio Broadcasting)
- GSM downlink + uplink (airprobe)
- TETRA downlink (OsmocomTETRA)
- ETSI GMR / Thuraya (OsmocomGMR)
- P25 (OsmocomOP25)
- AIS (Maritime transponders)
- Gen2 UHF RFID
- DECT (Digital European Cordless Telephony)

Who needs all of this?

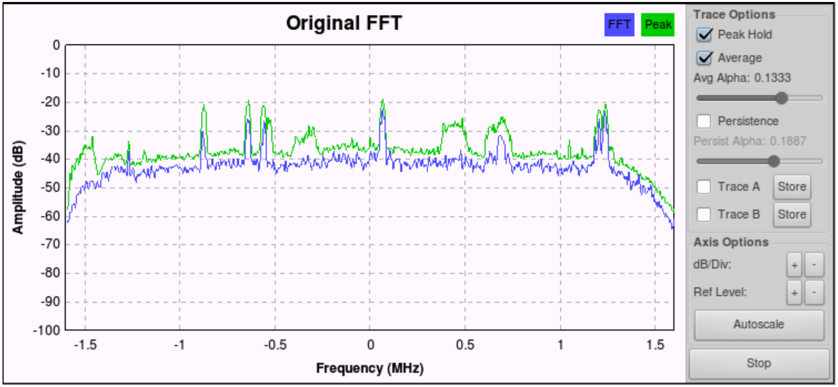
- Students learning about digital signal processing
- Radio Amateurs
- Communications (security) resarchers
- Anyone interested in building their own software radio receivers

This is of course not the 100k / million quantity consumer mass market. But nonetheless, definitely thousands to tens of thousands

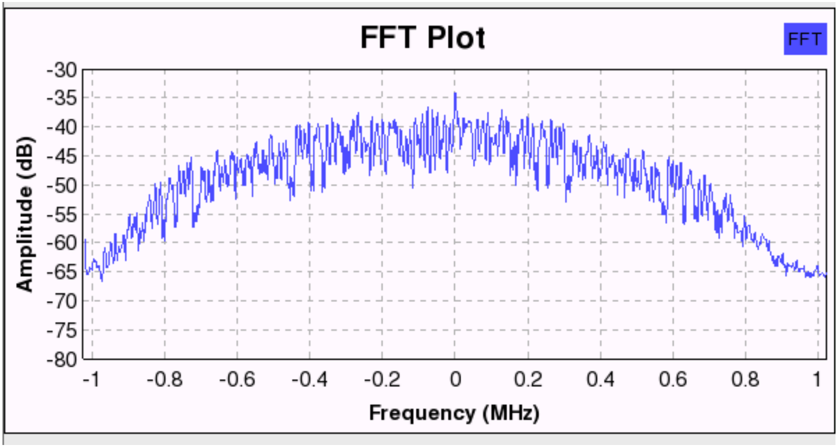
rtl-sdr: Multi-Carrier TETRA



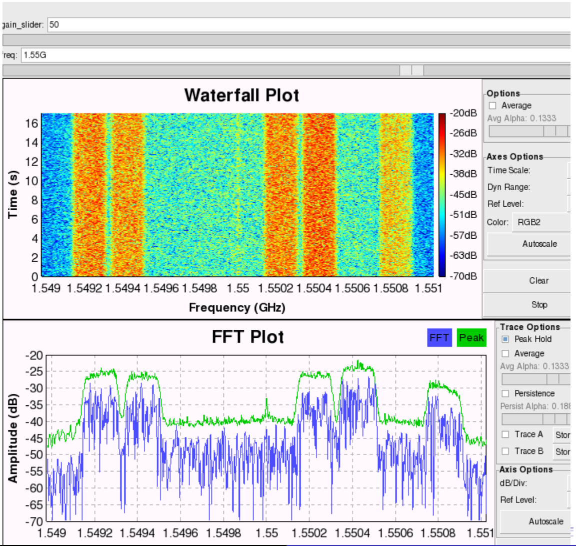
rtl-sdr: ETSI GMR (Thuraya Satphone)



rtl-sdr: GPS (after filter / LNA)



rtl-sdr: inmarsat (after LNA)


gain_slider: 50
freq: 1.55G

Thanks

I'd like to thank the many Osmocom developers and contributors, especially

- Steve Markgraf
- Dimitri Stolnikov
- Hoernchen
- Sylvain Munaut

also, Realtek for providing at least some (DVB oriented) documentation and for releasing GPL licensed code for their hardware in the first place.

Thanks

Thanks for your attention. I hope we have time for Q&A.