From baseband to bitstream and back again: What security researchers really want to do with SDR

Andy Davis, Research Director NCC Group

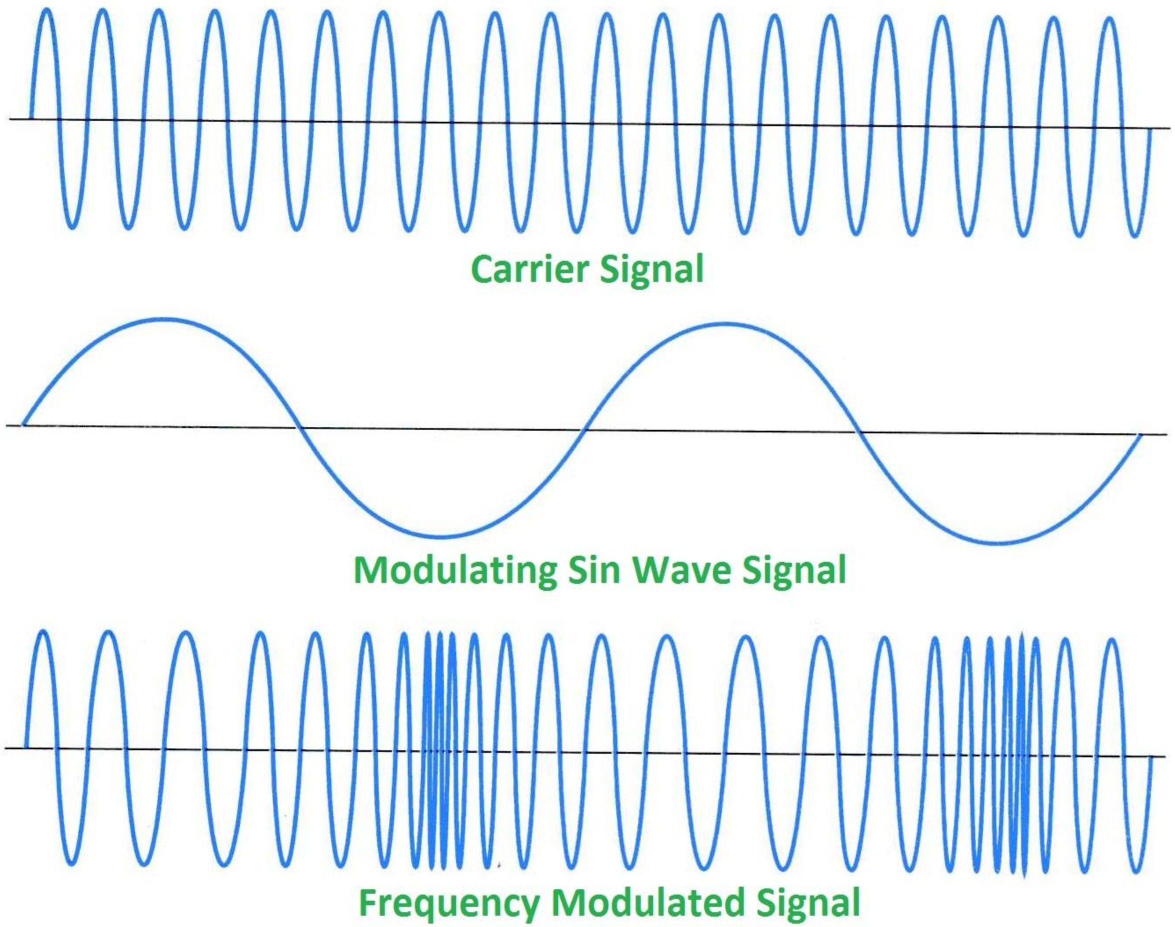
Agenda

- Signals basics
- Modulation schemes
- Information sources
- Receiving data (hardware and software)
- Developing a digital receiver step-by-step
- Transmitting data: Legal considerations
- Developing a digital transmitter step-by-step
- The RFTM

Acknowledgements

A big thanks to Michael Ossmann of Great Scott Gadgets for developing the HackRF and providing some excellent talks and tutorials which kick-started my journey through the SDR world

What is a radio signal?



Signals are waves measured in Hz (cycles per second)

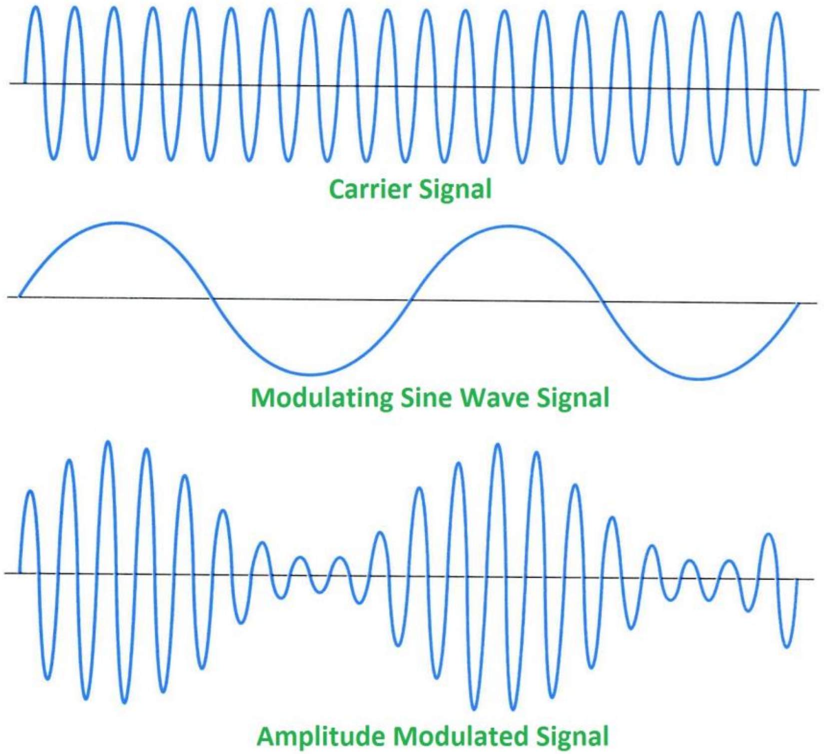
To transmit useful information we need to “modulate” a carrier signal

FM changes the frequency of the carrier proportionally to the information you wish to transmit

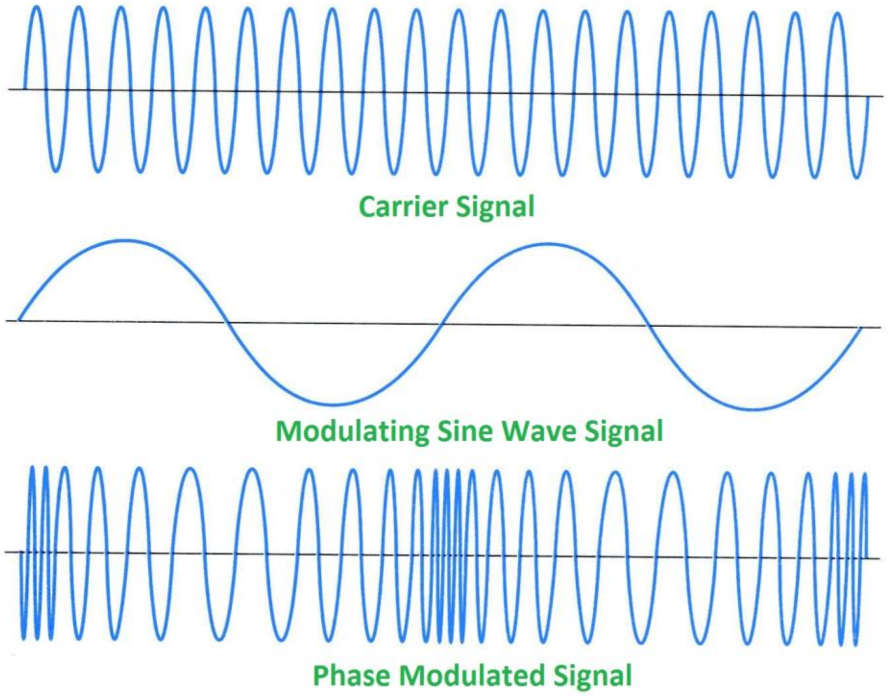
Images: ironbark.xtelco.com.au

Other types of modulation

Amplitude Modulation - AM



Phase Modulation - PM

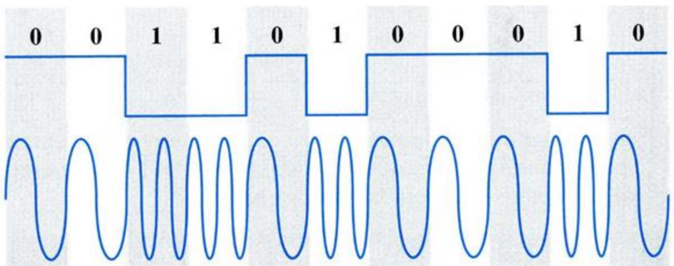


Images: ironbark.xtelco.com.au

Transmitting data instead of audio

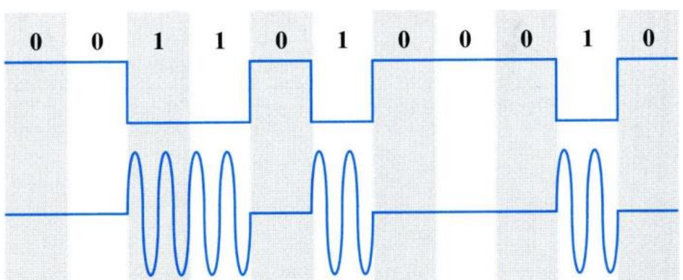
nccgroup
freedom from doubt

Frequency Shift Keying - FSK



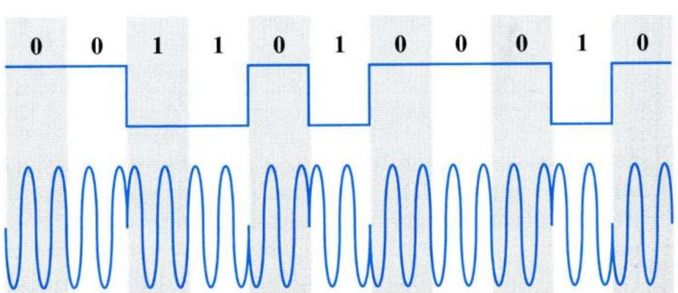
The frequency changes by a fixed “deviation” to represent either a “0” or a “1”

Amplitude Shift Keying - ASK



The amplitude changes by a fixed “deviation” to represent either a “0” or a “1”

Phase Shift Keying - PSK



The phase changes by a fixed “deviation” to represent either a “0” or a “1”

Images: ironbark.xtelco.com.au

Information sources – data sheets

ANALOG DEVICES

io-homecontrol-Compliant RF Transceiver

ADF7022

FEATURES

Very low power, high performance, low IF transceiver
Fully integrated io-homecontrol compliant protocol covering
- Layer 1, Layer 2, and time critical elements of Layer 3
- Media access
- Master, slave, and beacon modes supported
- Automatic io-homecontrol channel scan
- Automatic CRC, preamble, start byte insertion/check
- UART data encoding as per io-homecontrol
- Smart preamble detect/packet sniffing
- Automatic address filtering
- Low power modes

Autonomous packet handling without intervention of host microprocessor thus significantly increasing battery life
1-way and 2-way communication supported
Automatic wake-up timer
32-bit hardware timer, 16-bit firmware timer (48 bits total)
Uses either
- External 32 kHz crystal
- Internal 32 kHz RC oscillator

Patented fast settling automatic frequency control (AFC)
Fully integrated image rejection calibration (patent pending)
Digital RSSI
Operating frequencies
- Channel 1: 868.25 MHz
- Channel 2: 868.95 MHz
- Channel 3: 869.85 MHz

Very low power consumption
- 12.8 mA in receive mode with AGC active
- 11.9 mA in receive mode with manual AGC, ADC off
- 24.1 mA in transmit mode (10 dBm output)
- 0.75 µA in RCO wake mode
- 1.25 µA in XTO wake mode (32 kHz oscillator active)
- 38.4 µA average current in low power mode

Receiver sensitivity (10^{-3} BER)
- −108.5 dBm at 38.4 kbps FSK, 20 kHz deviation

Output power programmable up to 13.5 dBm
Automatic PA ramping
Dual PAs offer Tx antenna diversity
Very few external components
- Integrated PLL loop filter
- Integrated Rx/Tx switch
- Integrated battery monitor
- On-chip 8-bit ADC and temperature sensor

Efficient and flexible SPI control interface
- 4 lines available for low cost microcontroller interface
- Flexible Tx and Rx data buffers
- Efficient burst mode register access

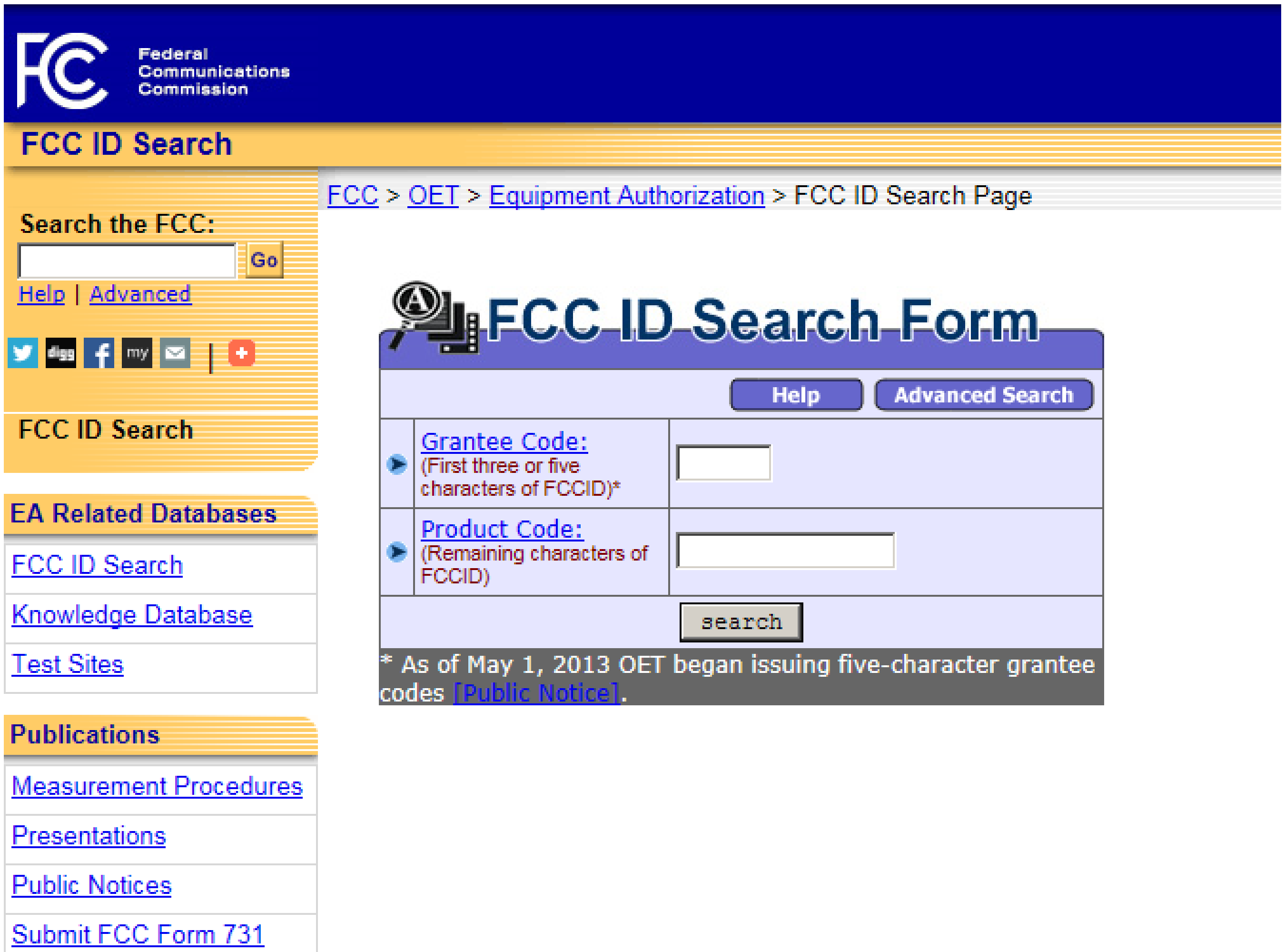
1.8 V to 3.6 V power supply
5 mm × 5 mm, 32-lead LFCSP package

APPLICATIONS

Home automation
Process and building control

Information sources – FCC database

http://transition.fcc.gov/oet/ea/fccid/



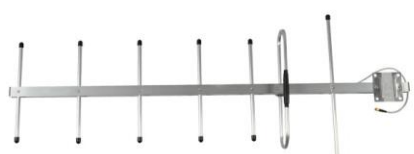
Receiving data

- Hardware: Software Defined Radio
  - RTL-SDR (Rx only)
  - FunCube (Rx only)
  - HackRF (Tx/Rx Half Duplex)
  - BladeRF (Tx/Rx Full Duplex)
  - USRP (Tx/Rx Full Duplex)


- Hardware: Other stuff
  - Connectors



  - Antennas

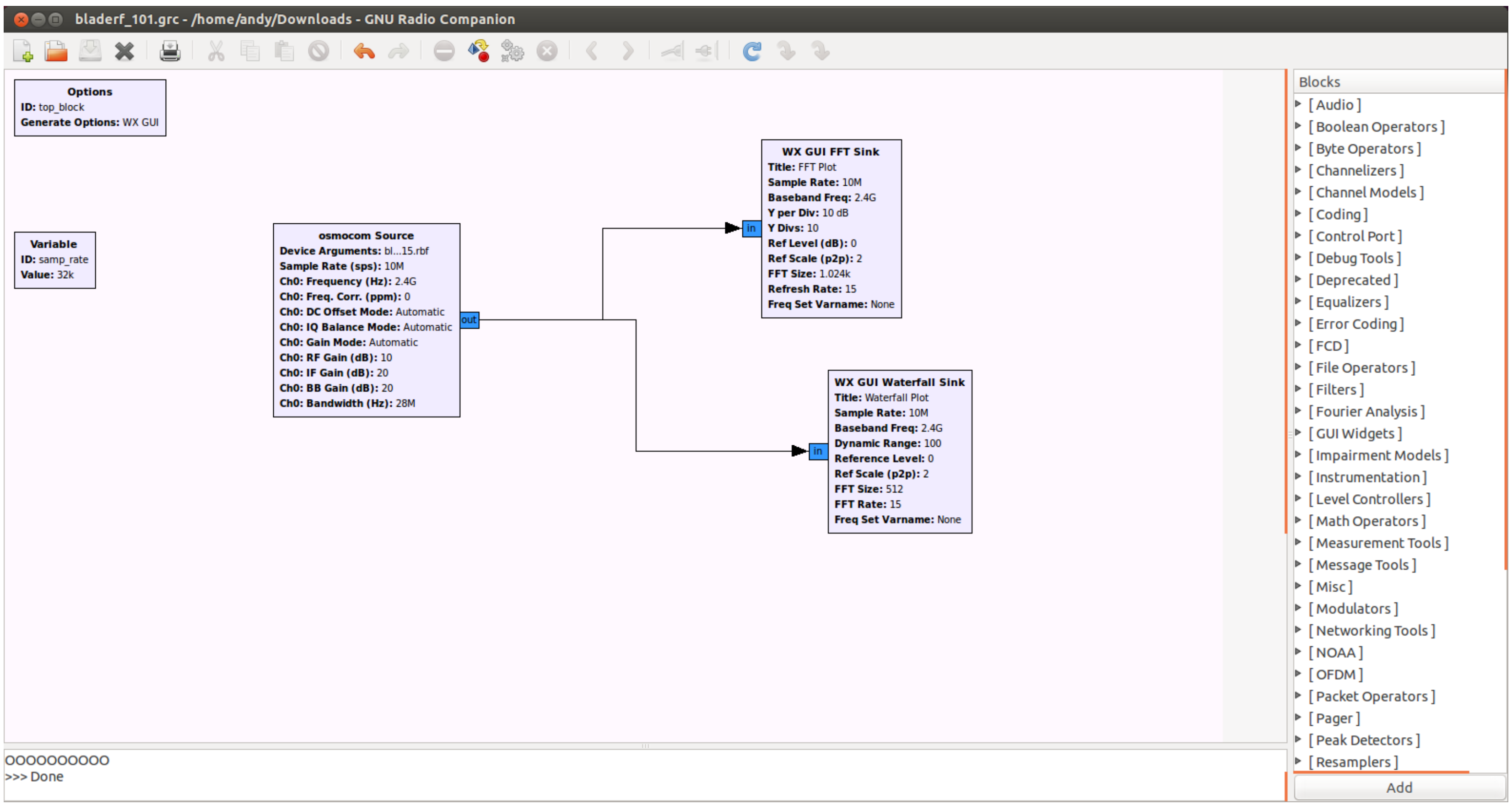

  - Amplifiers

Software: GNU Radio

nccgroup
freedom from doubt

GNU Radio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

Download software or LiveDVD from: gnuradio.org

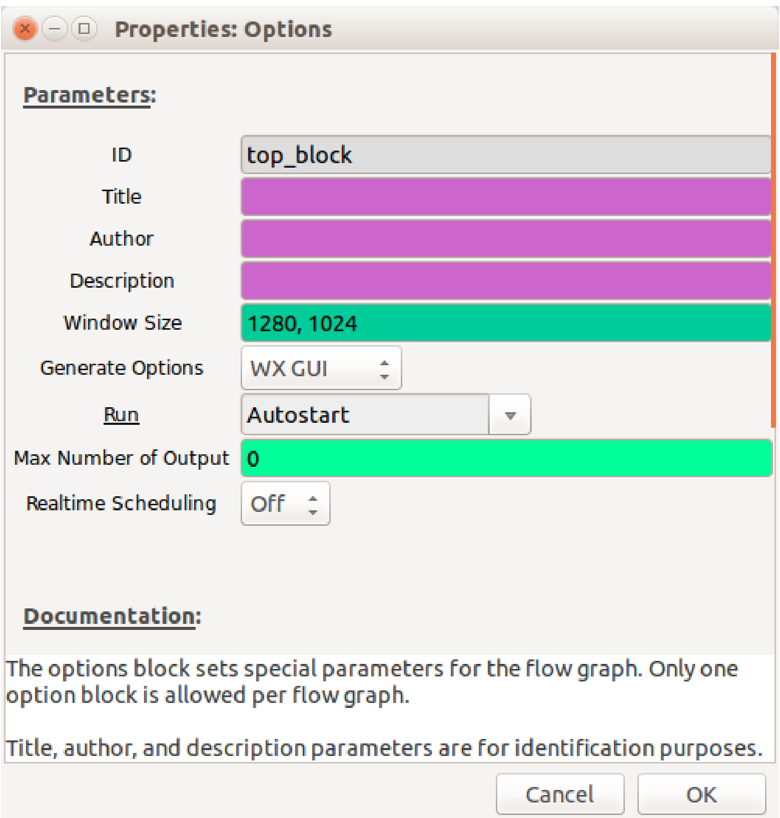


Developing a receiver

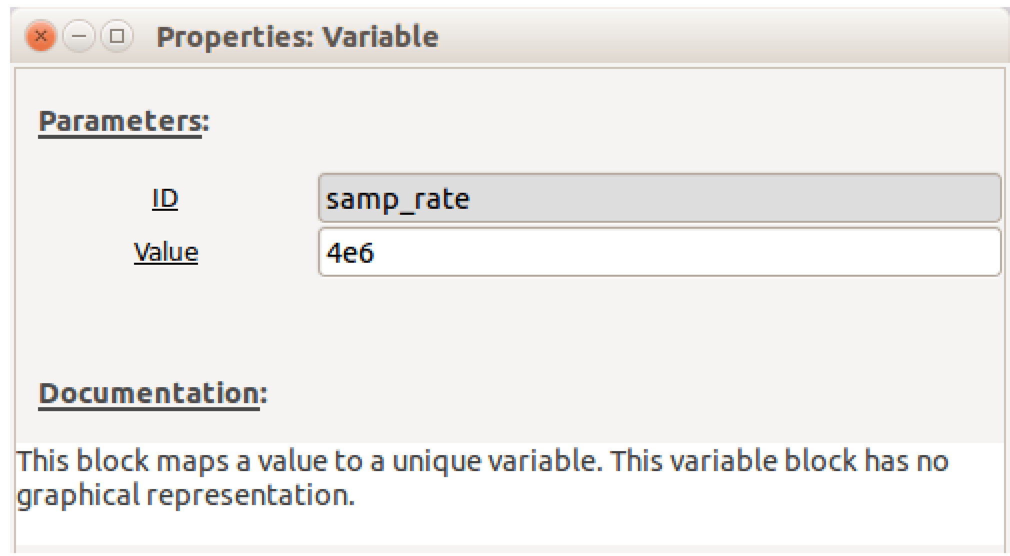
Developing a digital receiver step-by-step

- Configure the GNU Radio environment
- Configure the receiver
- Visualising the signal
- Signal identification
- Centring the signal
- Filtering
- Demodulating
- Visualising the demodulated signal
- Line coding
- Further filtering and clock recovery
- Data recovery

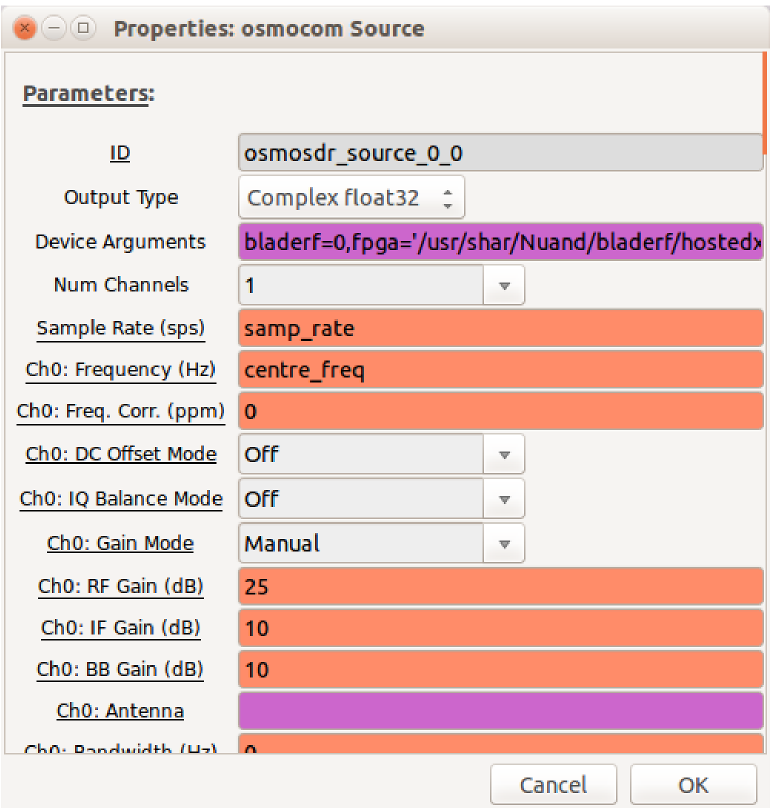
Configure the GNU Radio environment



Set the `samp_rate` variable to 4000000



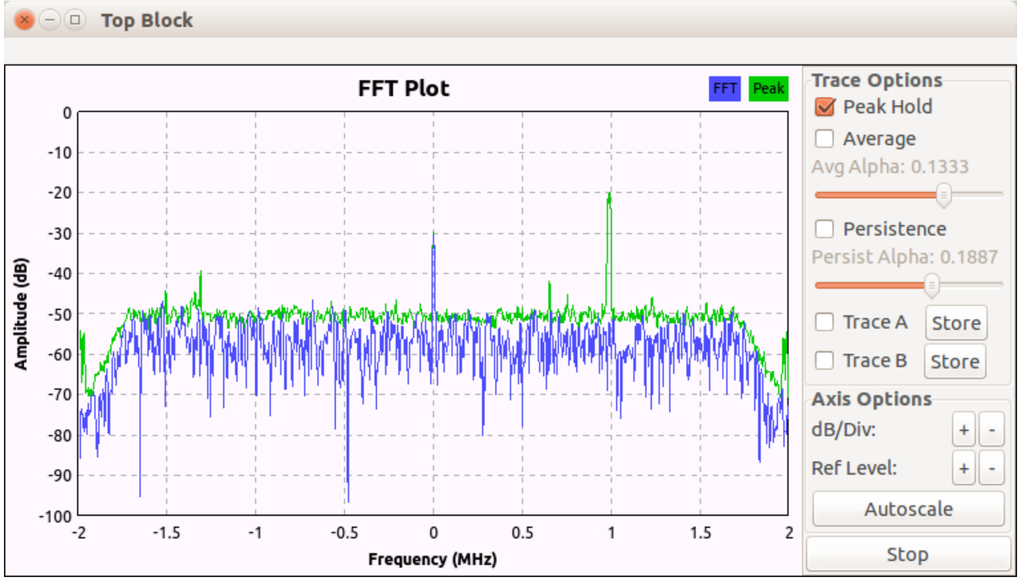
Configure the receiver



Sources:

- RTL-SDR
- FCD
- UHD
- osmocom

Visualising the signal

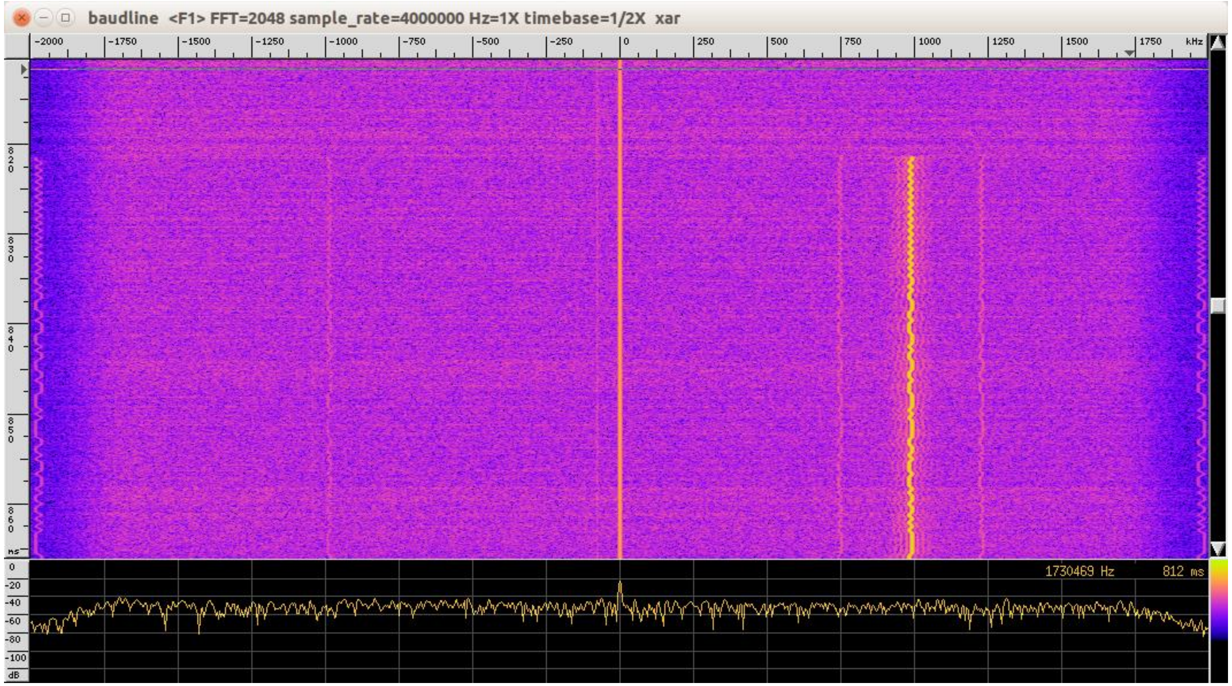


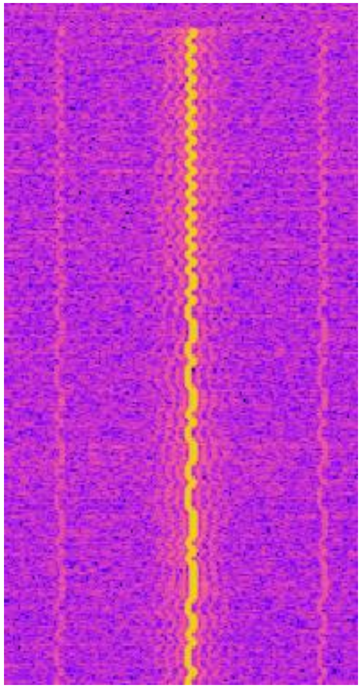
Wx GUI FFT sink:

- Visualise signals in the frequency domain

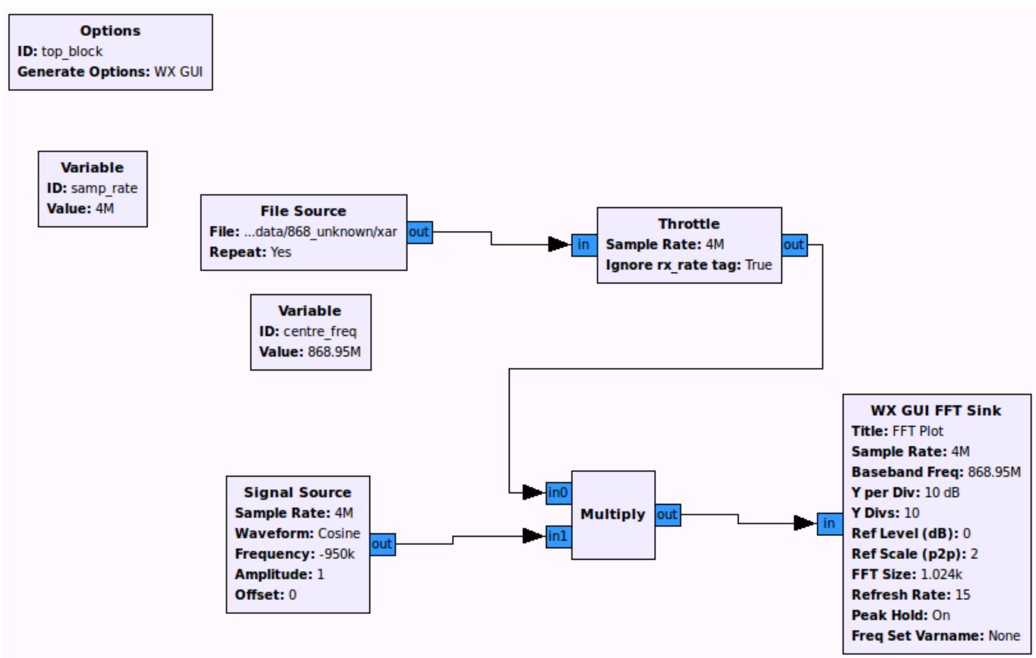
Signal identification with Baudline

Download from: baudline.com

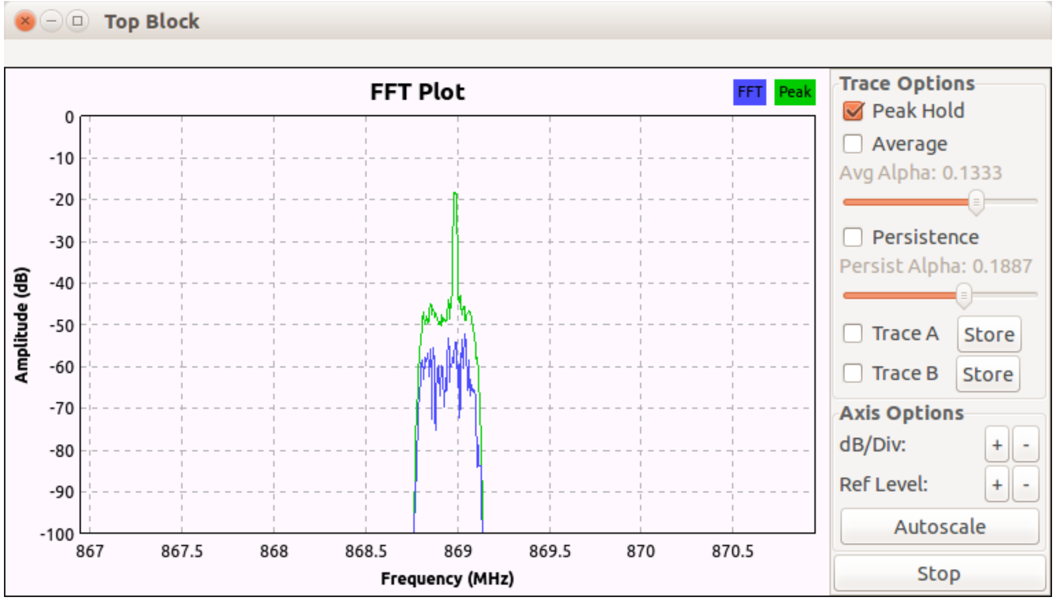




Centering the signal



Filtering



Low Pass Filter:

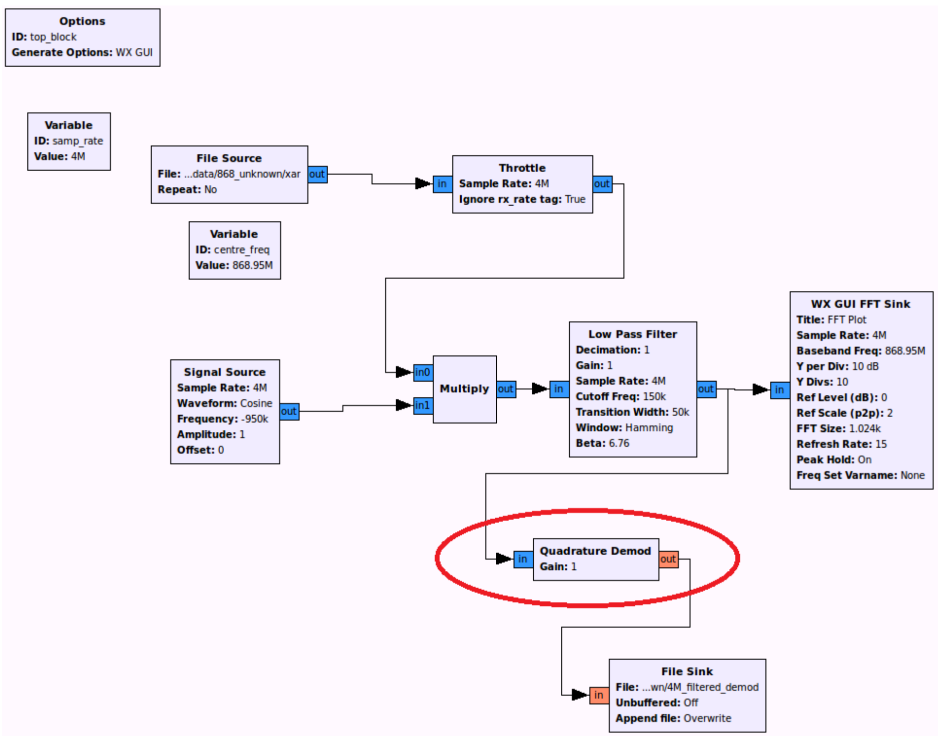
- Cut-off frequency
- Transition width

Rule of thumb:

Cut-off frequency = Baud rate

Transition width = Baud rate / 2

Demodulating



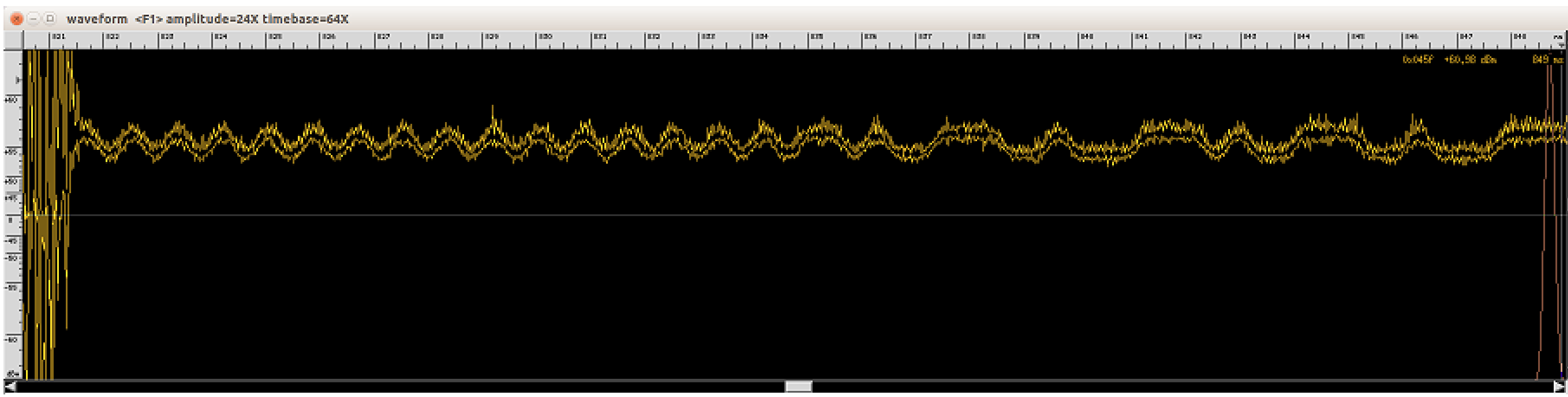
Frequency Shift Keying:

- Quadrature demod

Output to a file sink so we can view the demodulated signal

Visualising the demodulated signal

You can see the preamble then the data



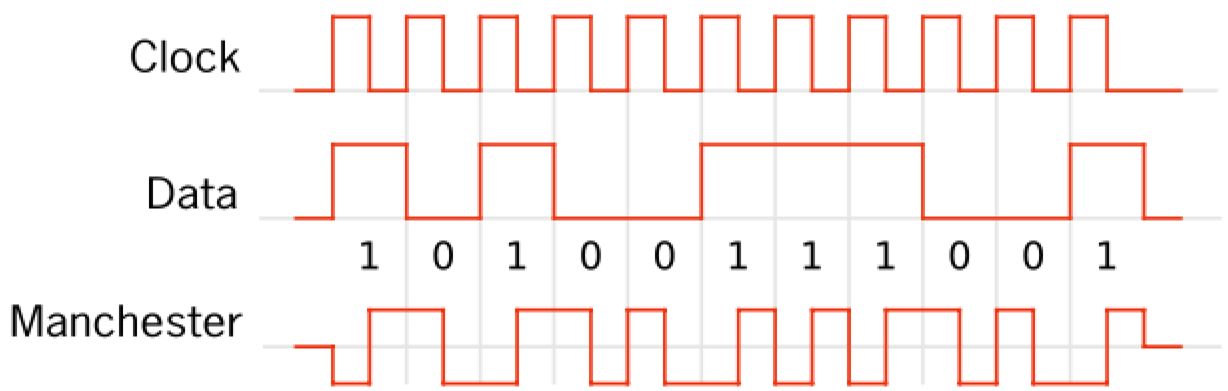
But the signal is quite fuzzy…

Line coding

Manchester encoding is what is known as a "line code”

- Ensures frequent voltage transitions, directly proportional to the clock rate,
- Helps clock recovery.

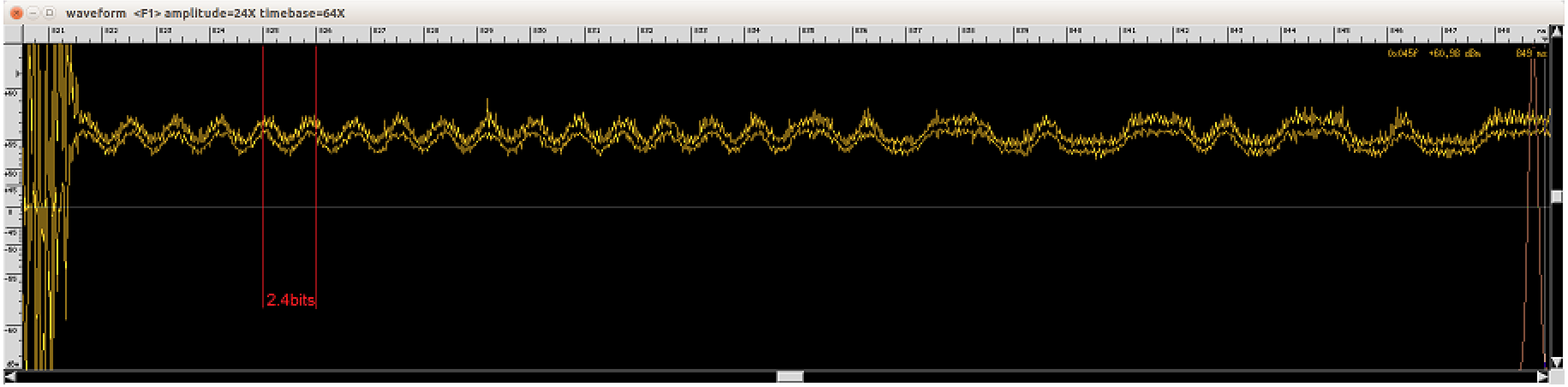
"0" is represented by a transition to low
"1" is represented by a transition to high



Images: wikipedia.com

Further filtering and clock recovery

Using the rulers to calculate the Baud rate:



2.4 symbols (bits) in 1 millisecond = Baud rate of 2400

Further filtering and clock recovery (2)

Add another Low Pass Filter to clean up the signal:

Cut-off frequency = 2400, Transition width = 1200

Further filtering and clock recovery (3)

Why is everything running so slowly?...

Sample rate = 4000000 samples per second

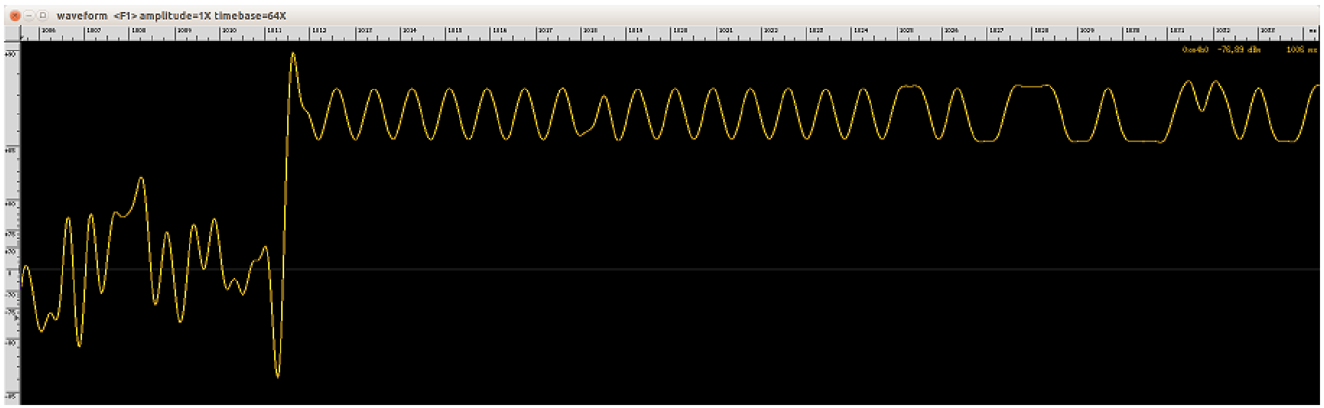
Baud rate = 2400 symbols per second

Samples per symbol = 1666.66!

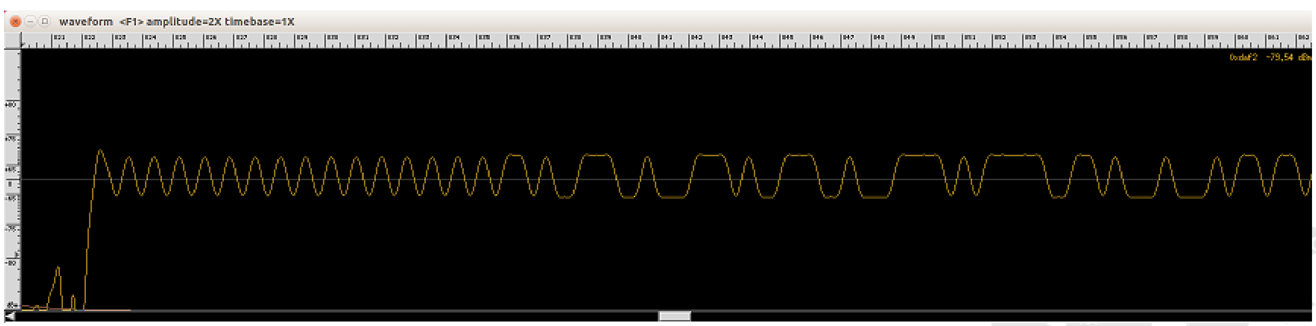
Within the Low Pass Filter, decimate the signal by a factor of 100

Samples per symbol now = 16.66

Signal offset



Add a "Add Const" block in between the second "Low Pass Filter" and the "File Sink" and determine the value through trial-and-error:



Data recovery

Clock Recovery MM:

Gain Omega: 0.01
Mu: 0
Gain Mu: 0.1
Omega Relative Limit: 0.01

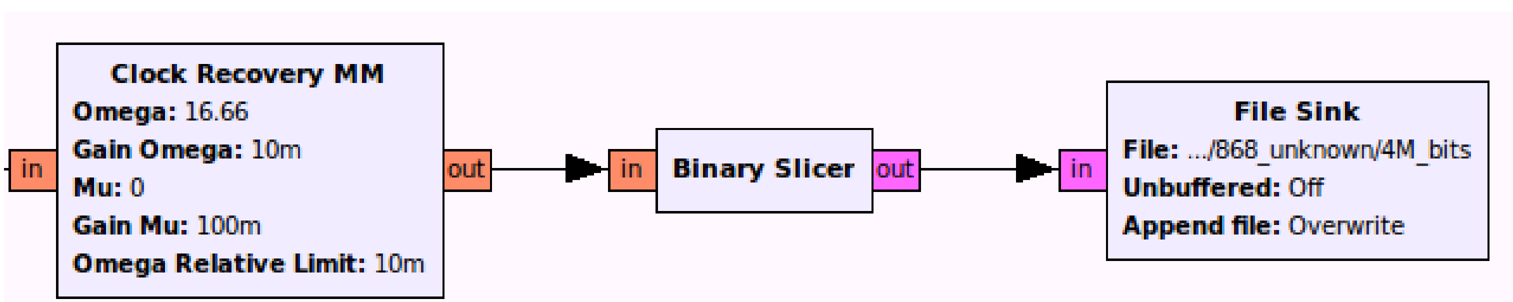
Omega: 16.66
(samples per symbol)

Binary Slicer:

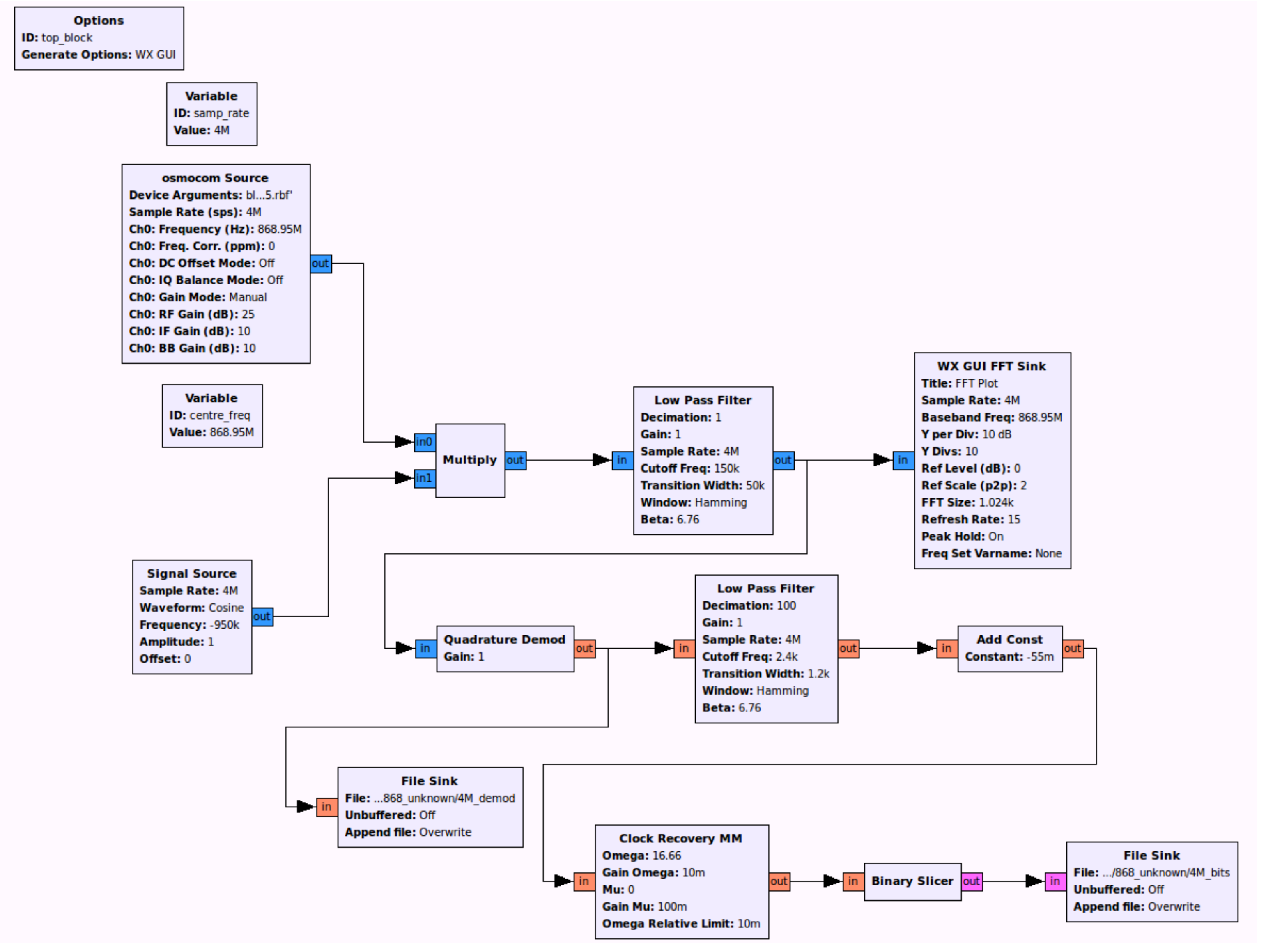
To recover the binary bits

File sink:

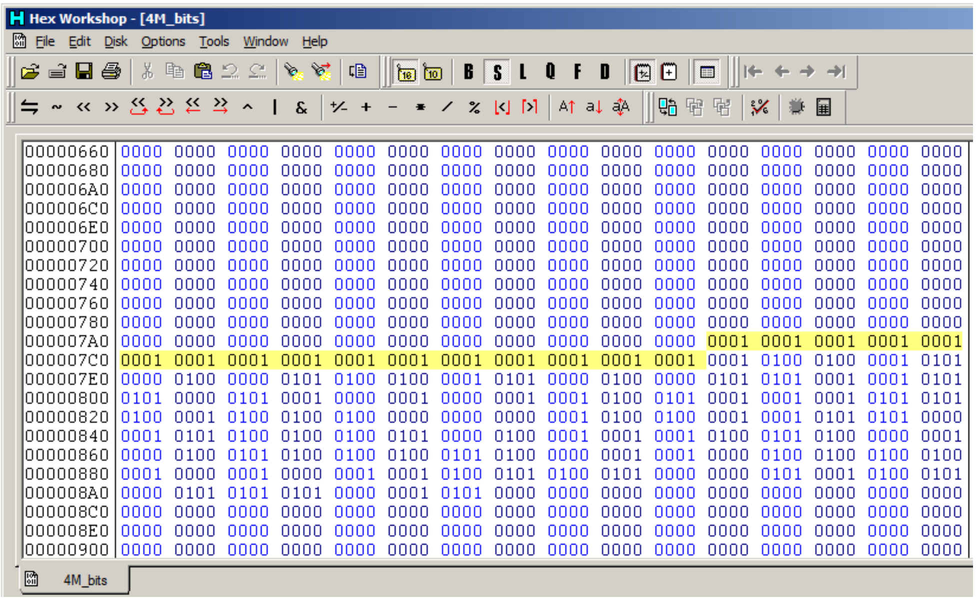
To receive binary data



Complete receiver flow graph



Data recovery (2)



Remember the preamble of "0101010101..."?

- "0" bit is represented by the byte "00"
- "1" bit is represented by the byte "01"

Use a TCP Sink block instead of a File Sink to send demodulated data to a network socket

Transmitting data

Legal considerations

- Do not transmit to air without an appropriate license
- Only transmit on frequencies and at power levels you’re authorised to
- If in any doubt consult the appropriate governing body:
    - Ofcom – UK
    - FCC – USA
    - CRTC - Canada

Images: linkedin.com

RF shielding and using attenuators

There are of course alternatives to broadcasting your transmission:

Shielded enclosures

Attenuators

Images: ramseytest.com, mercateo.co.uk

Developing a digital transmitter step-by-step

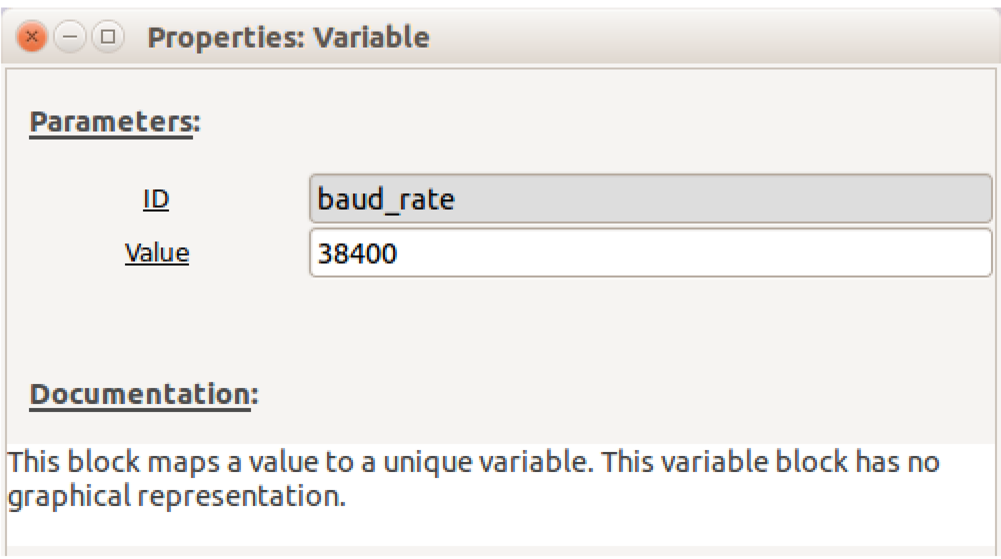
- Data source
- Set the Baud rate
- Modulation
- Resampling
- Adjust the signal level
- Configure the transmitter

Data source

- This transmitter is assumed to use FSK modulation (this example is actually a transmitter developed for the io-homecontrol protocol)
- The data source is a file containing bits captured using the process described
- Create `samp_rate_tx` variable = 4000000

Set the Baud rate

The io-homecontrol protocol uses a Baud rate of 38400:

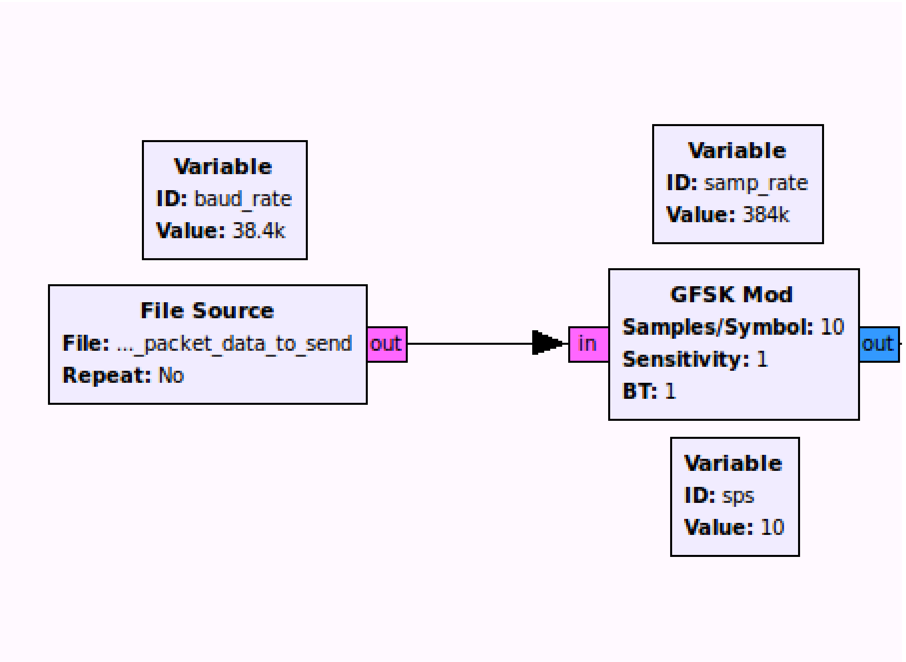


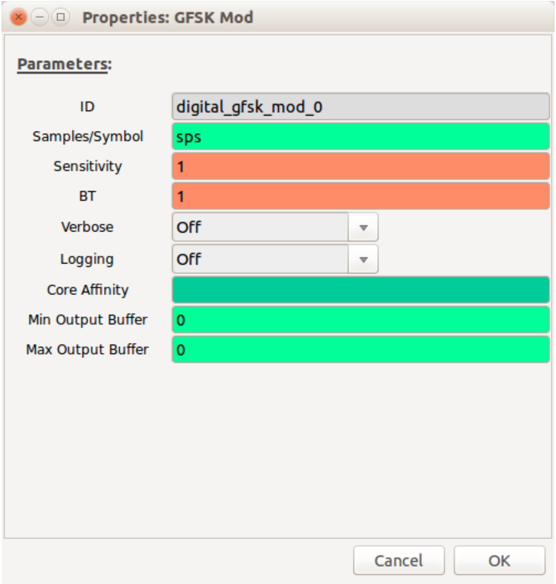
Modulation

Create `sps` (samples per symbol) variable = 10

Create `samp_rate` variable = `baud_rate` * `sps`

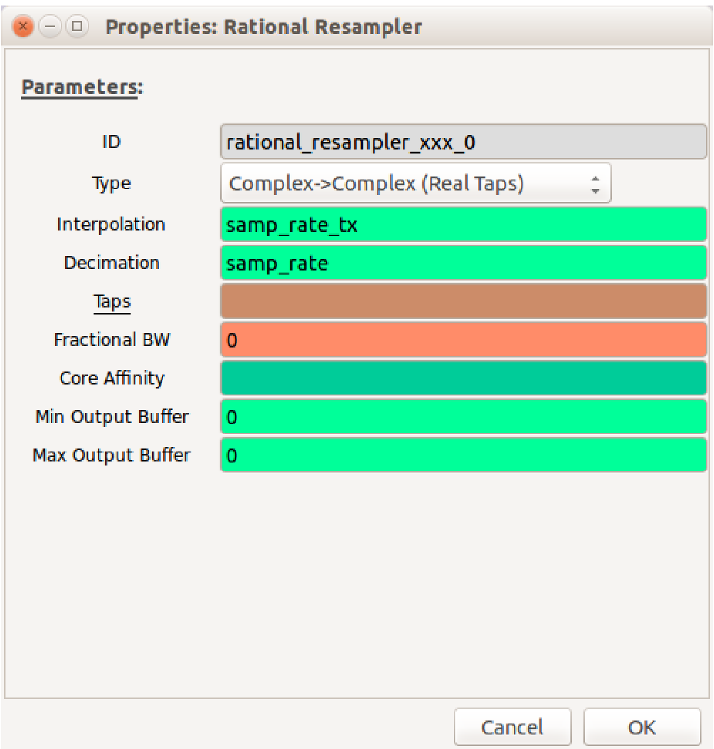
Add a GFSK Mod





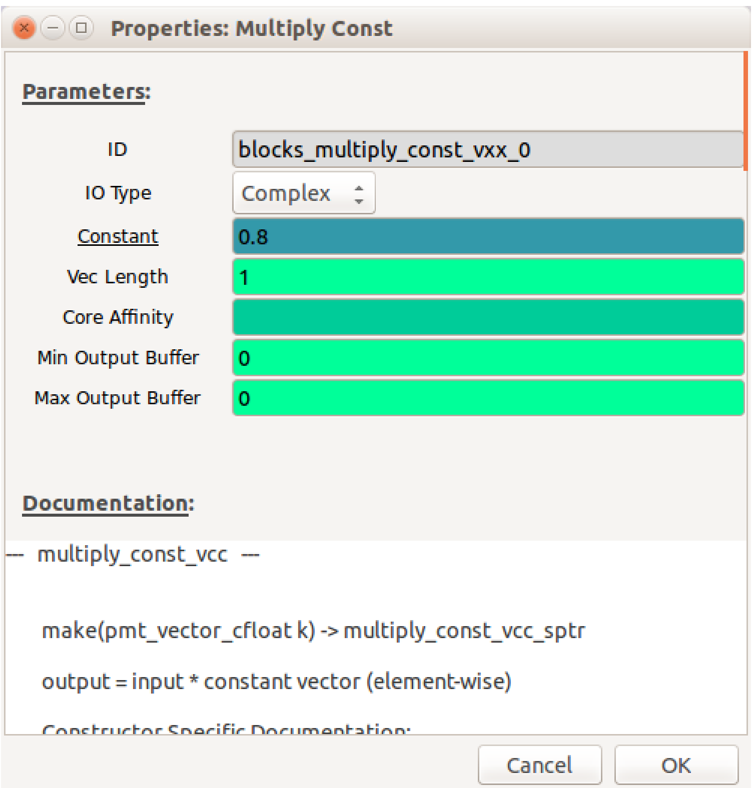
Resampling

Need to match the sample rate of the data with the transmitter (as it is the final sample rate of the transmitter that will actually determine the rate that the data is transmitted).



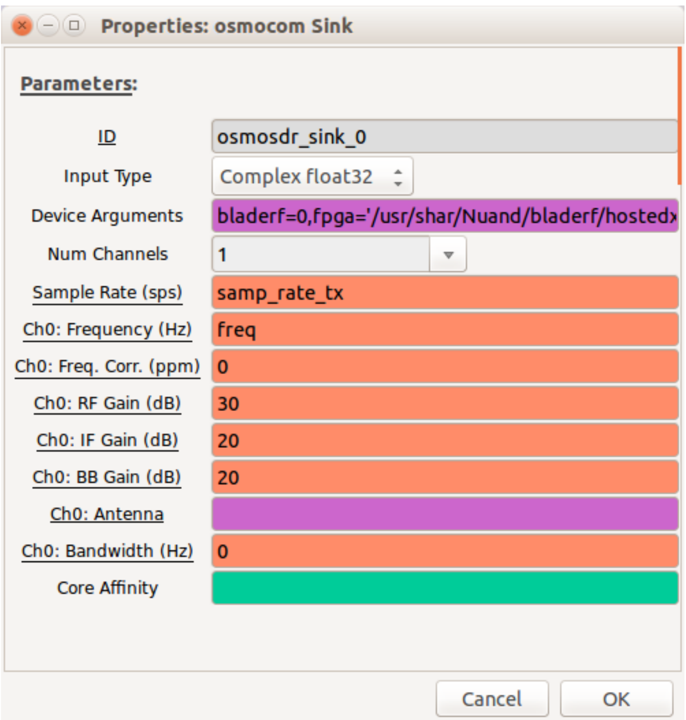
Adjust the signal level

We don't want to overload the input to the transmitter so we need to attenuate (reduce) the signal level.

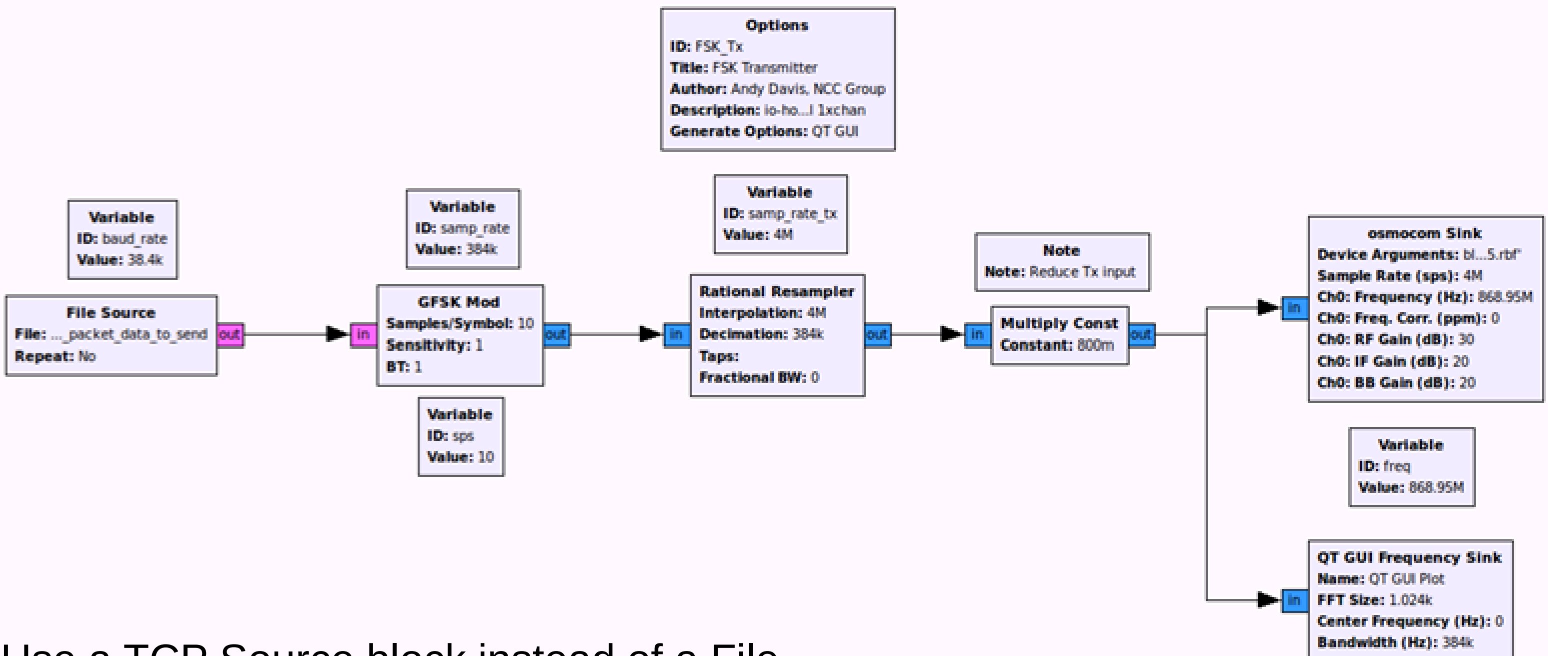


Configure the transmitter

Create `freq` variable = 868950000
(transmit carrier frequency in Hz)



Complete transmitter flow graph



Use a TCP Source block instead of a File Source to send data to be transmitted to a network socket

The Radio Frequency Testing Methodology

- Plain English and minimal mathematics!
- Community collaborative resource – please contribute if you can
- Hosted on Github pages
- The site is up now

Until we can find a suitable domain, this can be found at: http://nccgroup.github.io/RFTM/


./ RF Testing Methodology

www.nccgroup.com

Download as .zip | Download as .tar.gz | View on GitHub

The RFTM is an Open Source, collaborative testing methodology. It is specifically written in a straightforward way, avoiding mathematics where possible and focussed on providing the information that security researchers and consultants need to know in order to effectively test systems that employ RF technologies.

If you would like to contribute to the RFTM content, please submit your additions via GitHub.

Signals and Modulation

Information Sources

Receiving Signals

Developing an FSK receiver step-by-step

Transmitting Data

Developing an FSK transmitter step-by-step

Signals identification

Examples and Tutorials

Questions?

UK Offices

Manchester - Head Office

Cheltenham

Edinburgh

Leatherhead

London

Milton Keynes

European Offices

Amsterdam - Netherlands

Munich – Germany

Zurich - Switzerland

North American Offices

San Francisco

Atlanta

New York

Seattle

Australian Offices

Sydney