Hacking the radiofrequency spectrum: GNURadio as a signal processing prototyping tool

J.-M Friedt

FEMTO-ST Time & Frequency, Besançon, France
Contact: jmfriedt@femto-st.fr

All references available at http://jmfriedt.free.fr

August 1, 2013

Why digital ? Why software ?

Software provides **flexibility**, **reconfigurability**, **reproducibility** [1]

1. flexibility: use the same hardware for multiple purposes (analog/digital signal decoding) ⇒ no need for hardware modification
2. flexibility: update processing parameters depending on the environment or the conditions (flight/landing/mission)
3. reproducibility: no drift of processing result as a function of aging or environment (temperature ?)

⇒ shift from hardware to software

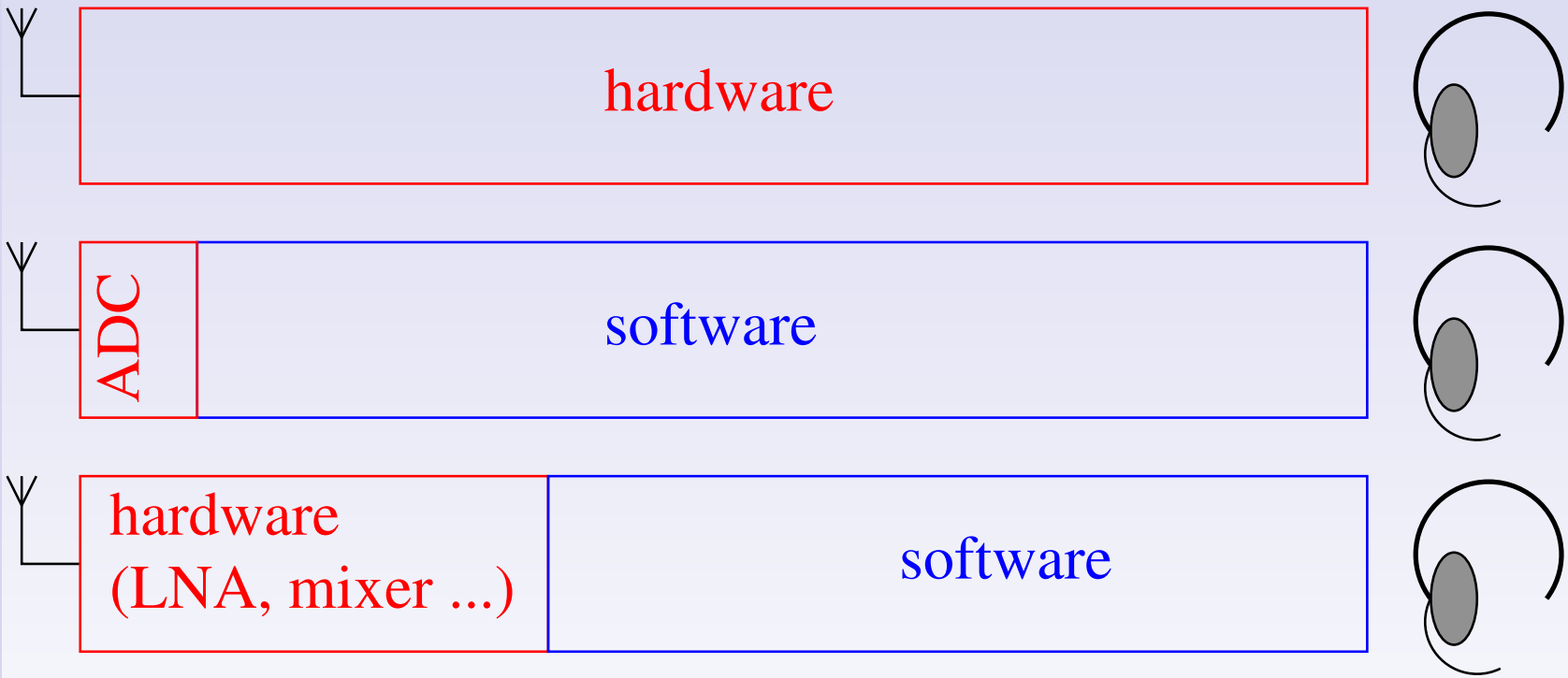
BUT limited bandwidth (cf SAW filters/correlators), and signal to noise/ratio + discretization ?

[1] D.A. Mindell, *Digital Apollo – Human and Machine in Spaceflight*, MIT Press (2008)
E.C. Hall, *Journey to the Moon – the history of the Apollo Guidance Computer*, American Institute of Aeronautics and Astronotics (1996)

Concepts of SDR

From all hardware receiver to a single front-end A/D converter (ADC) followed by software digital signal processing

$\rightarrow$ not applicable due to A/D bandwidth and memory usage [2]



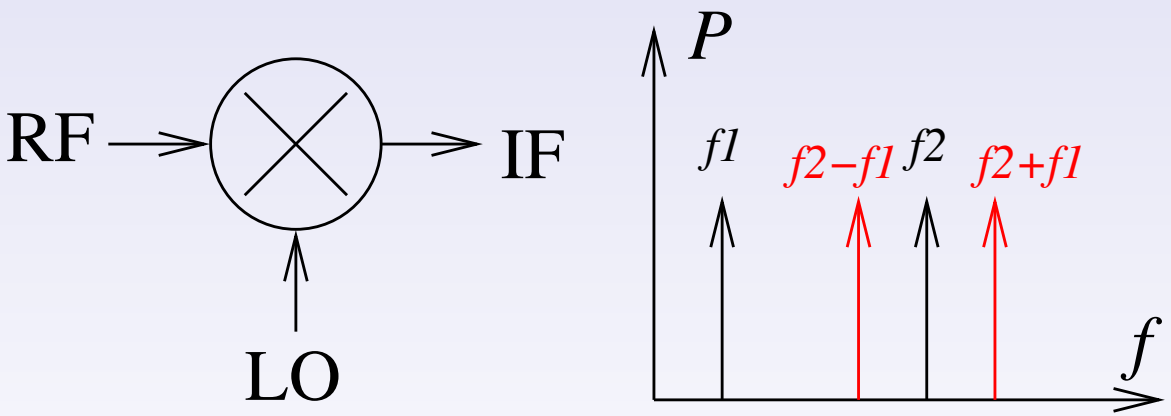
[2] K. Borre, D.M. Akos, N. Bertelsen, P. Rinder & S.H. Jensen, *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*, Birkhäuser Boston (2007) and slides at `http://kom.aau.dk/project/softgps/` and `http://kom.aau.dk/project/softgps/GNSS_SummerSchool_DGC.pdf`

Mixing sine waves

- Only **the bandwidth** of the signal matters in the digitization process, the carrier frequency is removed by mixing
- Core aspect of zero-IF receiver: frequency transposition

$$\cos(a)\cdot\cos(b)=\frac{1}{2}\left(\cos(a-b)+\cos(a+b)\right)$$

$$\sin(a)\cdot\cos(b)=\frac{1}{2}\left(\sin(a-b)+\sin(a+b)\right)$$



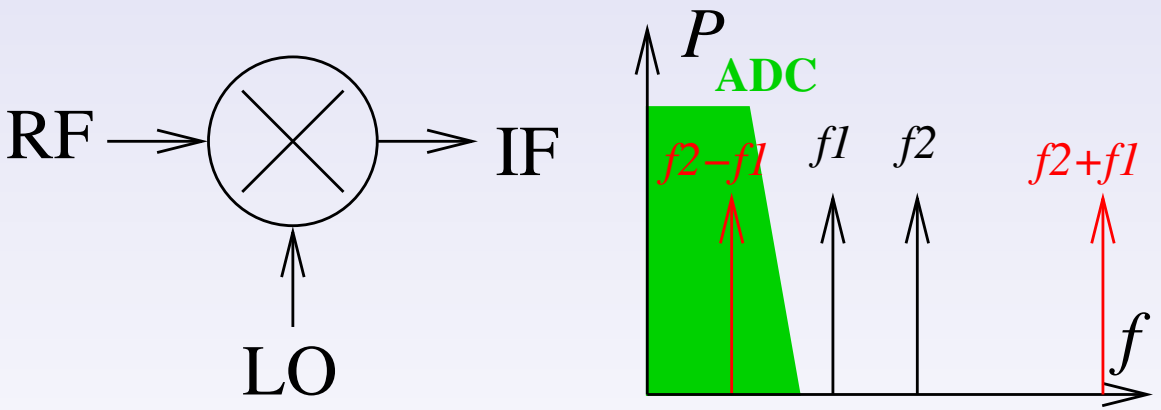
$$\cos(\omega_1 t)\cdot\cos(\omega_2 t)=\frac{1}{2}\left(\cos((\omega_1-\omega_2)t)+\cos((\omega_1+\omega_2)t)\right)$$

Mixing sine waves

- Only **the bandwidth** of the signal matters in the digitization process, the carrier frequency is removed by mixing
- Core aspect of zero-IF receiver: frequency transposition

$$\cos(a)\cdot\cos(b)=\frac{1}{2}\left(\cos(a-b)+\cos(a+b)\right)$$

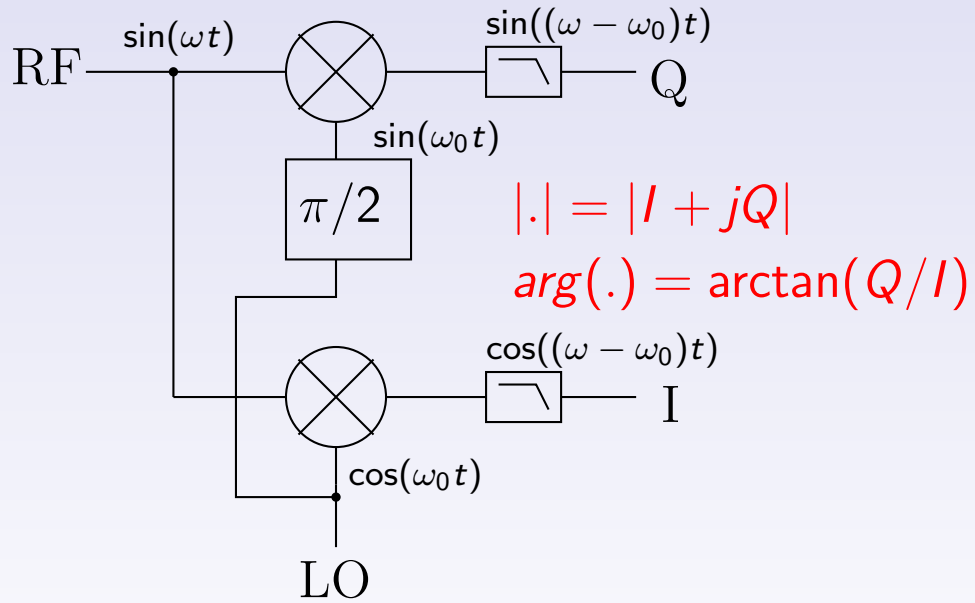
$$\sin(a)\cdot\cos(b)=\frac{1}{2}\left(\sin(a-b)+\sin(a+b)\right)$$



$$\cos(\omega_1 t)\cdot\cos(\omega_2 t)=\frac{1}{2}\left(\cos((\omega_1-\omega_2)t)+\cos((\omega_1+\omega_2)t)\right)$$

Consumer electronics for SDR

- Many sources of radiofrequeny A/D converters, in our examples Elonics E4000 + Realtek RTL2832U[3] and sound card for I/Q outputs[4] , but also radiomodems and DDS (USRP)
- sampling bandwidth up to 64 Msamples/s ⇒ zero-IF approach
- Raw information: stream of periodically sampled I and Q values (2.8 MS/s for E4k, 96 or 192 kS/s for sound card)



[3] `http://sdr.osmocom.org/trac/wiki/rtl-sdr`

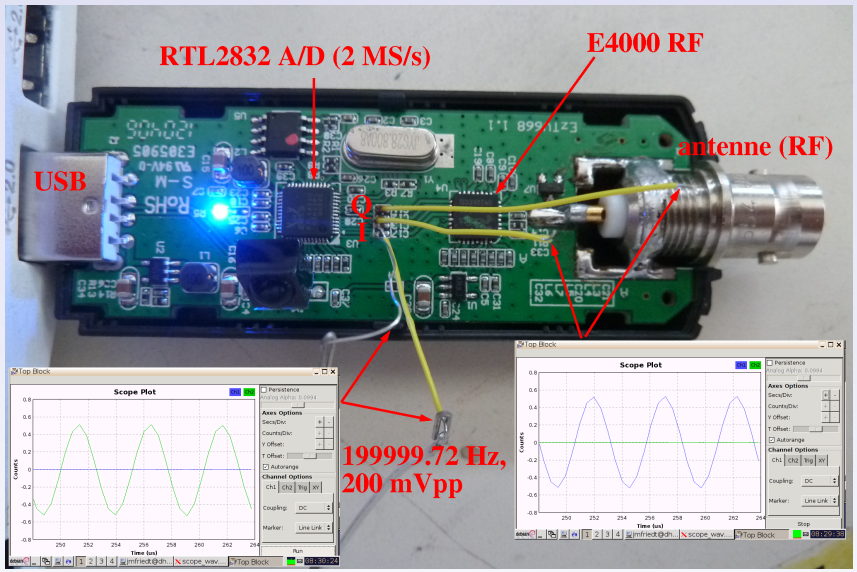
[4] Agilent, *Digital Modulation in Communications Systems – An Introduction*, Application Note 1298, or M. Steer, *Microwave and RF design – a systems approach*, SciTech Publishing, Inc (2010)

The GNURadio environment

Having obtained a stream of I/Q bytes, software processing blocks:

- input (USRP, DVB receiver, sound card, file ...)
- process
- output (file, audio stream, stdio, virtual oscilloscope/spectrum analyzer)

`gnuradio-companion`: GUI for assembling blocks and generator of Python file



- 8-bit ADC for high bandwidth (oversampling does not compensate for low resolution: [5]) 1 bit/(sampling rate$\times$4)$\Rightarrow$ 2800/92 $\simeq$ 30 $\Rightarrow$ 2.5 bits

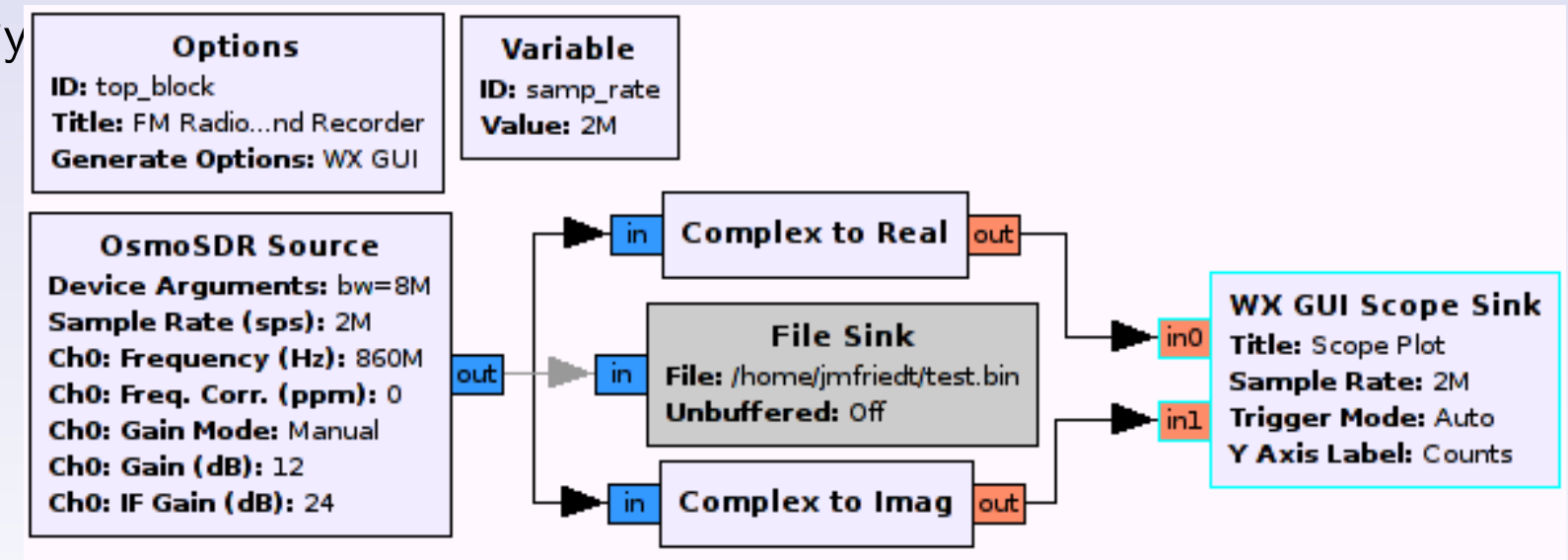
[5] Application note AN2668, *Improving STM32F101xx and STM32F103xx ADC resolution by oversampling*, ST Microelectronics, 2008

The GNURadio environment

Having obtained a stream of I/Q bytes, software processing blocks:

- input (USRP, DVB receiver, sound card, file ...)
- process
- output (file, audio stream, stdio, virtual oscilloscope/spectrum analyzer)

`gnuradio-companion`: GUI for assembling blocks and generator of Py

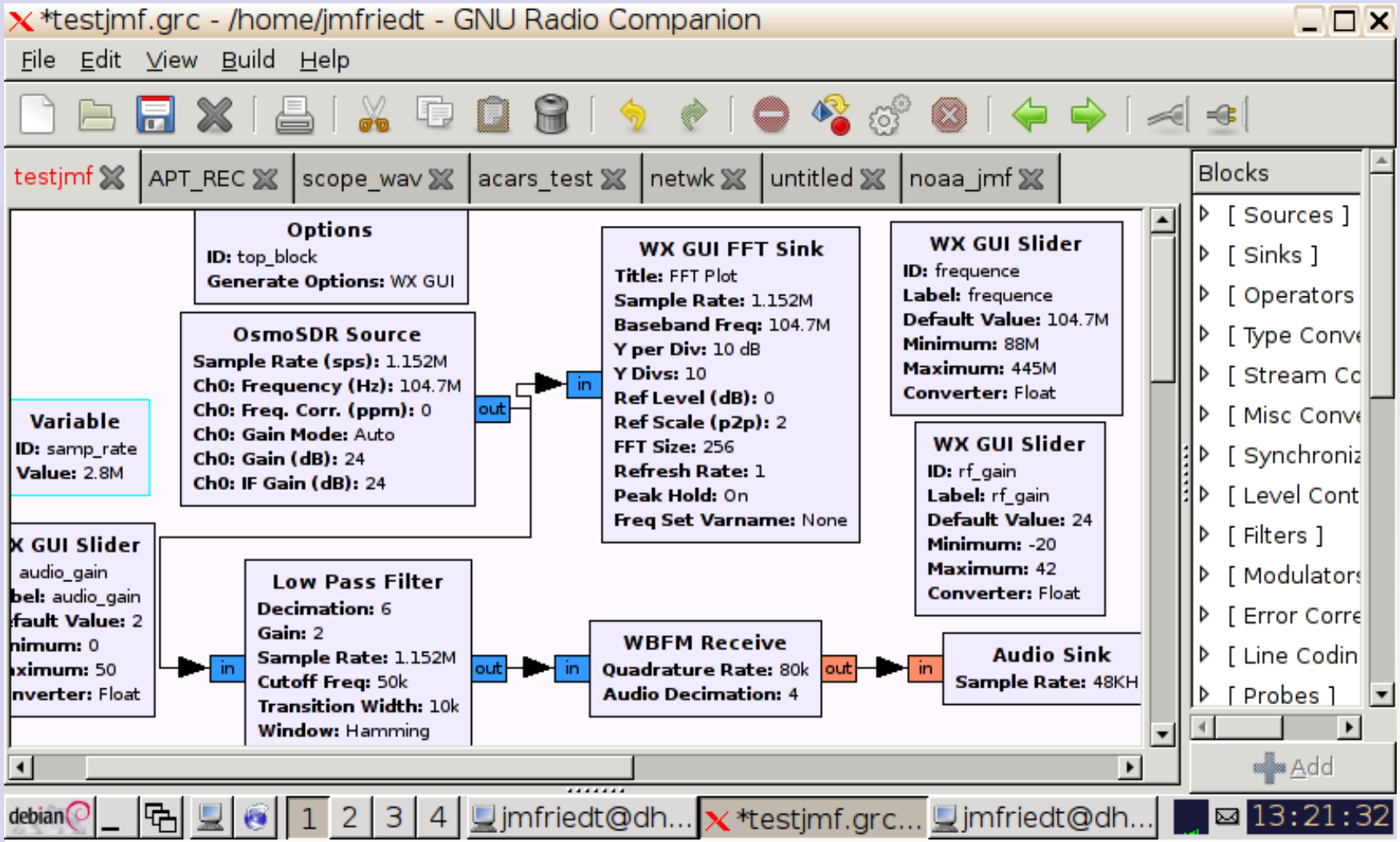


- 8-bit ADC for high bandwidth (oversampling does not compensate for low resolution: [5]) 1 bit/(sampling rate$\times$4)$\Rightarrow$ 2800/92 $\simeq$ 30 $\Rightarrow$ 2.5 bits

[5] Application note AN2668, *Improving STM32F101xx and STM32F103xx ADC resolution by oversampling*, ST Microelectronics, 2008

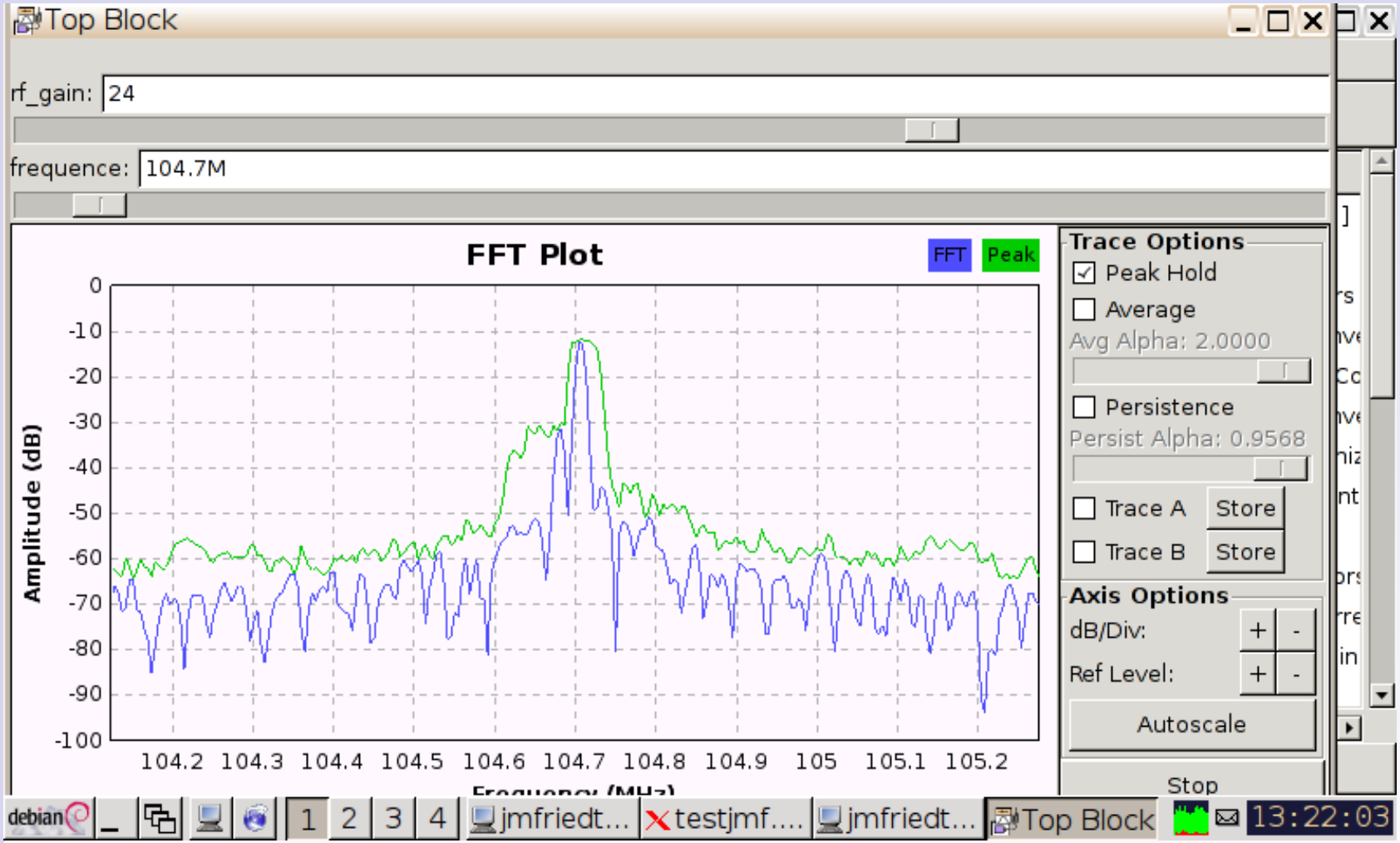
GNURadio basic use

WFM receiver !



GNURadio basic use

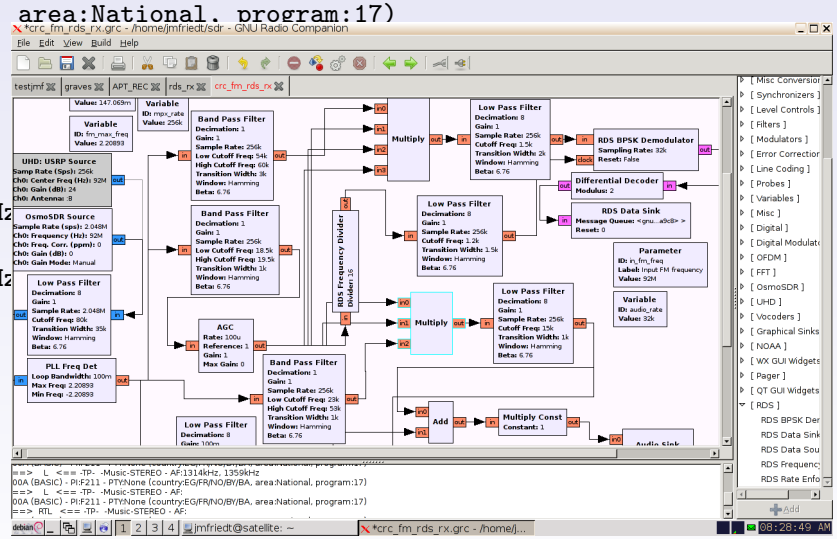
WFM receiver !



GNURadio basic use

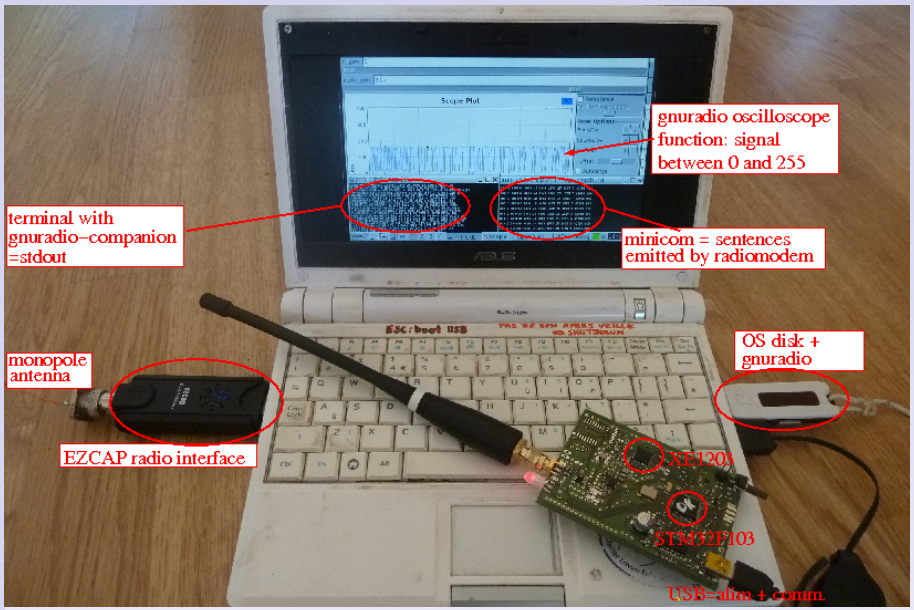
- Most common analog demodulation schemes (AM, WFM) and digital datastream encoding
- digital signal decoding: FSK is the digital version of FM,
- one example of mixed analog-digital signal processing: RDS decoding [6]

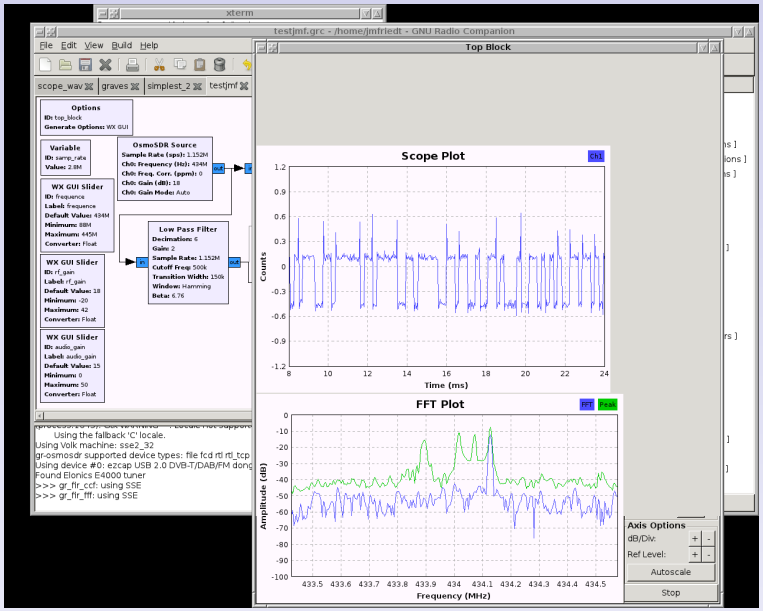
```
00A (BASIC) - PI:F211 - PTY:None (country:EG/FR/NO/BY/BA, area:National, program:17)
==>  RTL   <== -TP-  -Music-STEREO - AF:
00A (BASIC) - PI:F211 - PTY:None (country:EG/FR/NO/BY/BA, area:National, program:17)
==>  RTL   <== -TP-  -Music-STEREO - AF:
00A (BASIC) - PI:F211 - PTY:None (country:EG/FR/NO/BY/BA, area:National, program:17)
@@@@@ Lost Sync (Got 46 bad blocks on 50 total)
@@@@@ Sync State Detected
@@@@@ Lost Sync (Got 46 bad blocks on 50 total)
@@@@@ Sync State Detected
00A (BASIC) - PI:F219 - PTY:None (country:EG/FR/NO/BY/BA,
==>  IRL   <== -TP-  -Music-STEREO - AF:99.60MHz, 99.80MHz
00A (BASIC) - PI:F219 - PTY:None (country:EG/FR/NO/BY/BA,
==>  IRL   <== -TP-  -Music-STEREO - AF:99.60MHz, 99.80MHz
00A (BASIC) - PI:F219 - PTY:None (country:EG/FR/NO/BY/BA,
==> VIRL N <== -TP-  -Music-STEREO - AF:100.40MHz
```



[6] https://cgran.org/wiki/RDS

GNURadio basic use





FSK modulation: bits are visible as binary level on output of WFM demodulator

Knowing the baudrate of 4800 bauds:

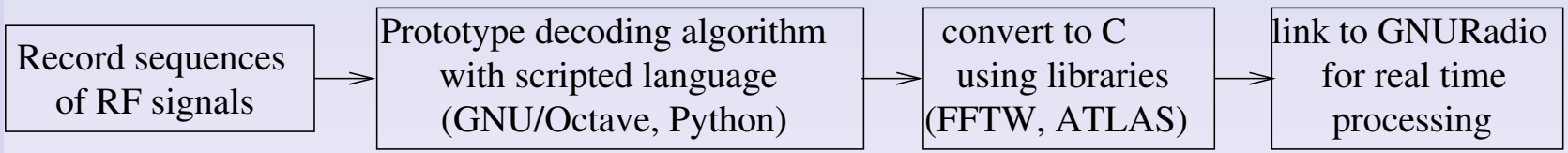
```
2 .0 0000 465 31 534 6 02109
2 .0 0000 558 31 500 7 02110
2 .0 0000 595 31 169 8 02110
2 .0 0000 637 31 426 9 02110
2 .0 0000 225 31 374 10 02110
2 .0 0000 333 31 324 12 02110
2 .0 0000 203 31 419 13 02111
2 .0 0000 367 31 246 14 02111
2 .0 0000 195 31 296 15 02111
2 .0 0000 284 31 305 17 02111
-> 2 .0 0000 201 31 325 18 02112
2 .0 0000 209 31 311 19 02112
2 .0 0000 336 31 341 20 02112
```

```
UUUU\bc\e7\fb\822 .0 \fc000 465 31 534 6002109
\c\f8\e0UUU\bc\e7\93\82\fe .0 00\fc0 558 31 500 7 02110
\fc\c0UUU\bc\e7\93\822 0 0000595 31 16\f9 8 02110
U\e0UUUU\bc\93\822 .0\fc0000 6\ff7 31 426 9 02110
\ff\b0E\e0UUUU\bc\e7\93\fe2 .0 0T00 225 31374 10 0210
\fc\f0UUUU\bc\e7\93\82\fe .0 000 333 31 24 12 02110
\ff\ff\e0\c1UUU\bc\e7\93\822 .0 0\bc00 203\bc31 419 13 02111
\ff\5\ff\f8\c0U5UU\bc\e7\93\822 .0 00\e40 367 \e71 246 14 02111
\f8\f0\e0UUU\bc\e7\93\822 .0 000\94 195 3\95 296 15 02111
\f8\f0UUUU\bc\e7\93\822 \fe0 0000\fc284 31 30\fd 17 02111
\ff\e0\5\f0\f0\e0UUUU\bc\e7\93\822 .0 0000\d02\f09 31 311 \859 02112
\ff\e0\f0\f0\e8\e0UUUU\bc\e7\93\822 .0 0000 336 31 \e741 20 021\e52
```

Write your own processing block

However, neither decoder for digital protocol I am interested in (ACARS), nor tools for time & frequency analysis

GNURadio is opensource ⇒ add the missing blocks by learning from other's source code



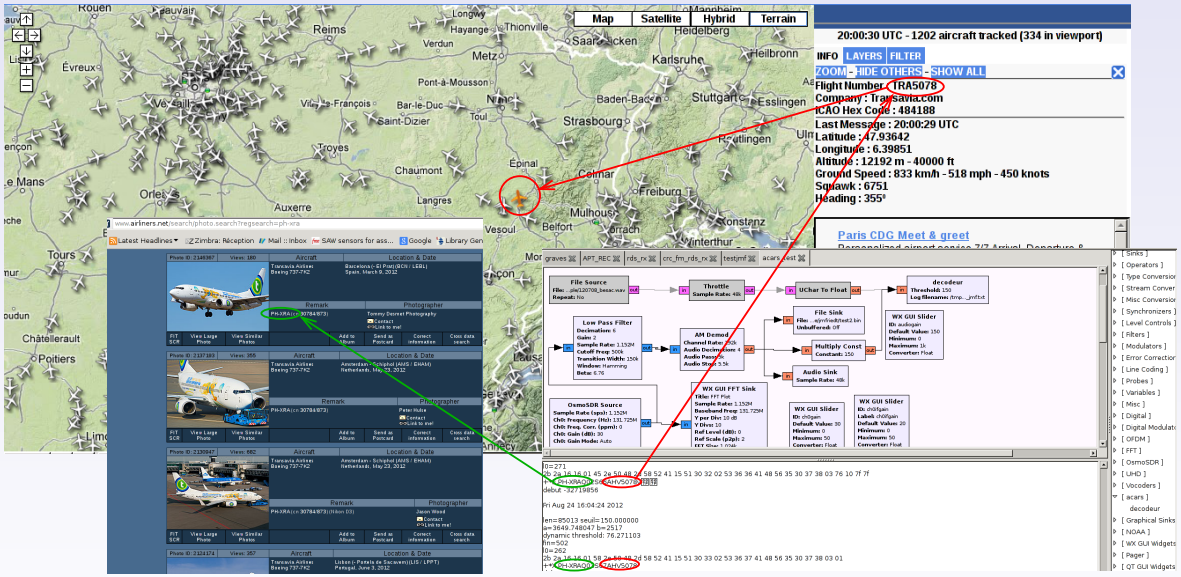
Development strategy:

1. prototyping using GNU/Octave (Matlab compatible) on recorded datasets,
2. convert to C(++) and test on the same recorded datasets,
3. comply with `gnuradio-companion` block description and test on recorded datasets but with chunks of unknown size,
4. apply to real time decoding.

Write your own processing block

Example of the ACARS protocol[7], used on VHF band (131.725 MHz in Europe), AM modulated:

1. encoding at **1200** (bit 0) and **2400** Hz (bit 1) [8]
2. data rate of **2400** bps
3. header to tune AGC of RF frontend: stream of 2400 Hz data ($\geq$13 periods)
4. data interpretation: 0 means the bit value changes, 1 means the bit value remains constant



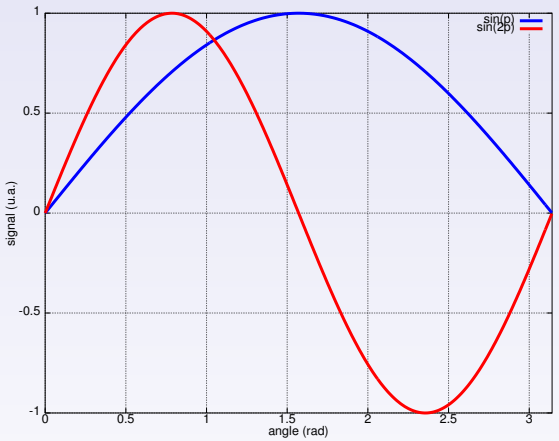
[7] http://files.radioscanner.ru/files/download/file4094/acars.pdf

[8] http://www.tapr.org/aprsdoc/ACARS.TXT

Write your own processing block

Bit identification: many means to an end

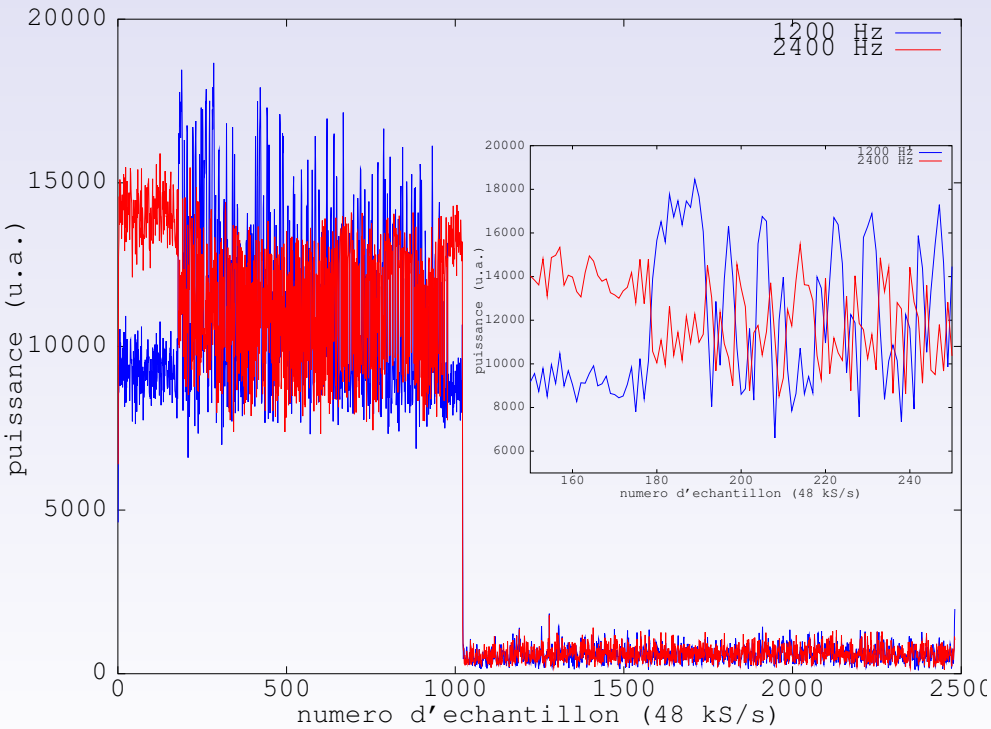
- time-domain band-pass filter (FIR) ... general purpose,
- convolution with the expected signals (1200 & 2400 Hz sine wave) $\Rightarrow$ frequeny domain (requires FFT),
- use at best the signal encoding properties
- $\int_0^1 \sin(2\pi t)\sin(\pi t)dt \propto$
  $\int_0^1 (\cos(3\pi t) - \cos(\pi t))\,dt =$
  $sin(3\pi) - sin(0) - (sin(\pi) - sin(0)) = 0$
- $\int_0^1 \sin(2\pi t)\sin(2\pi t)dt =$
  $1/2 \times \int_0^1 (\cos(4\pi t) - \cos(0))\,dt =$
  $1/2 \times (sin(4\pi) - sin(0) + 1) = 1/2$
- $\int_0^1 \sin(\pi t)\sin(\pi t)dt =$
  $1/2 \times \int_0^1 (\cos(2\pi t) - \cos(0))\,dt =$
  $1/2 \times (sin(4\pi) - sin(0) + 1) = 1/2$

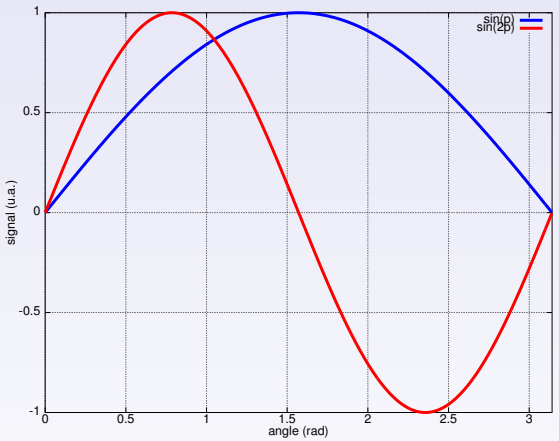


Write your own processing block

Bit identification: many means to an end

- time-domain band-pass filter (FIR) ... general purpose,
- convolution with the expected signals (1200 & 2400 Hz sine wave) $\Rightarrow$ frequeny domain (requires FFT),
- use at best the signal encoding properties



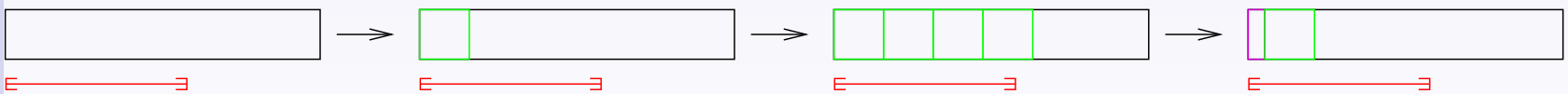


From GNU/Octave to C

- Manual conversion. Could it be optimized ? (proprietary Mathworks HDL Coder ?)
- Blocks can be written in Python ⇒ prototype using Scipy/Numpy rather than GNU/Octave
- FFT with different normalization convention ⇒ update threshold values
- From a complete (recorded) dataset to a stream of blocks of variable size

Solution:

1. fill buffer until the required datasize has been accumulated, and process a given number of data
2. Reinitialize the buffer with the remaining, unprocessed, data.



Complying with gnuradio-companion structure

Comply with gnuradio-companion block structure:

1. an XML configuration file describes input, output, callback functions (`.xml`),
2. a library file defines the implementation (`.cc`)
3. a header file defines variables (`.i`)

```
<?xml version="1.0"?>
<block>
  <name>decodeur</name>
  <key>acars_decodeur</key>
  <category>acars</category>
  <import>import acars</import>
  <make>acars.decodeur($seuil)</make>
  <param>
    <name>Threshold</name>
    <key>seuil</key>
    <type>real</type>
  </param>
  <sink>
    <name>in</name>
    <type>float</type>
  </sink>
</block>
```

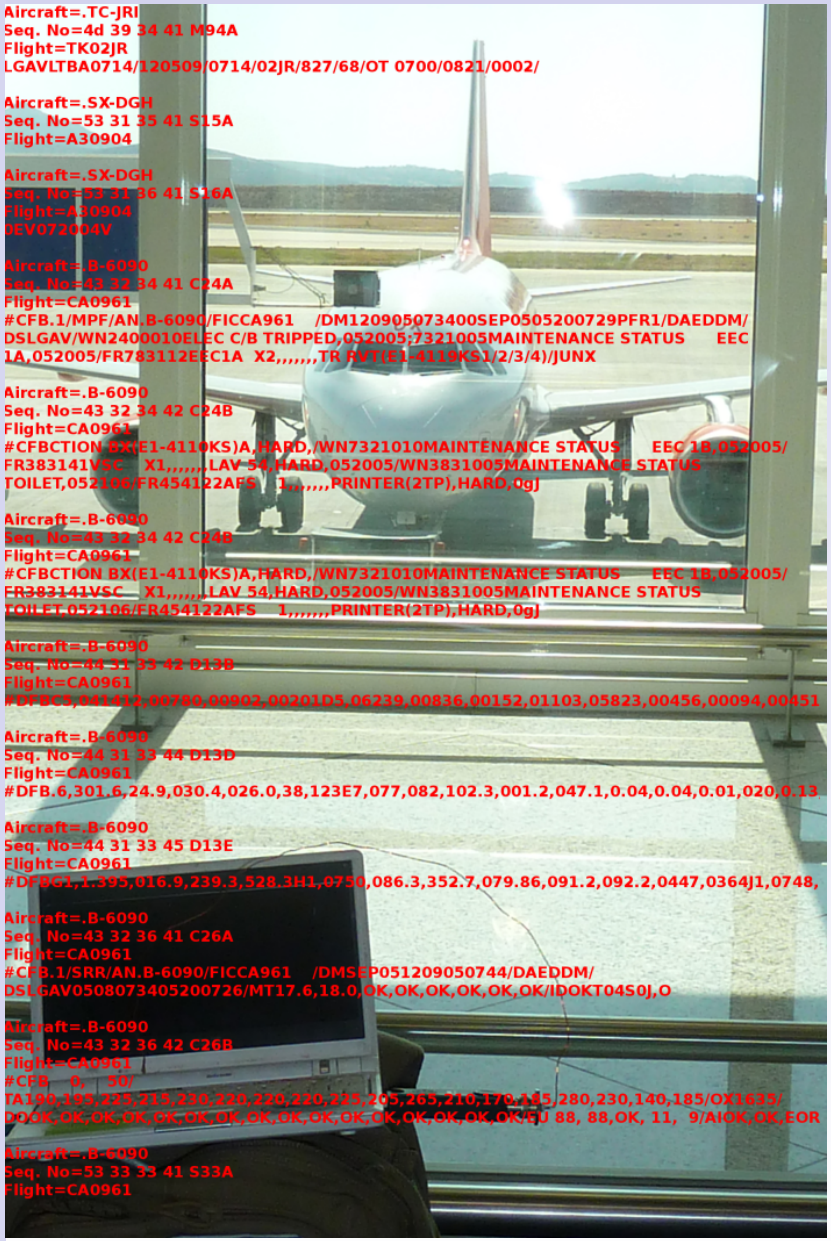
Complying with gnuradio-companion structure

```
int counters_counters::general_work (int noutput_items,
                                     gr_vector_int &ninput_items,
                                     gr_vector_const_void_star &input_items,
                                     gr_vector_void_star &output_items)
{
  const float *in = (const float *) input_items[0];
  float *out = (float *) output_items[0];
  float min=500.,max=-500.;
  int k,N,cpt,debut,fin;

  N=noutput_items;
  for (k=-Ntot;k<-Ntot+N;k++) {dm[k]=in[k--Ntot];}
  -Ntot+=N;
  if (-Ntot>-tgate) // active compteur
     {printf("tgate=%d Ntot=%d N=%d ",-tgate,-Ntot,N);
      // compteur direct
      cpt=0;
      for (k=0;k<-tgate-1;k++)
          if ((-dm[k]>=(-seuil)) && (-dm[k+1]<(-seuil))) cpt++;
      printf("freq=%f cpt=%d ",-freq,cpt);
      // compteur reciproque
      cpt=0; k=-1;
      do {k++;} while (!((-dm[k]>=(-seuil)) && (-dm[k+1]<(-seuil))));
      debut=k; k=debut+-tgate;
      do {k++;} while (!((-dm[k]>=(-seuil)) && (-dm[k+1]<(-seuil))));
      fin=k;
      for (k=debut+1;k<=fin;k++)
          if ((-dm[k]>=(-seuil)) && (-dm[k+1]<(-seuil))) cpt++;
      for (k=fin-1;k<-Ntot;k++) -dm[k-(fin-1)]=-dm[k];
      printf(" cpt=%d fin-deb=%d f=%f\n",cpt,fin-debut,(float)-samp_rate/(float)(fin-debut)*(→
          ↪float)cpt);
      -Ntot-=(fin-1);
     }
  consume_each (noutput_items);
  return noutput_items;
}
```

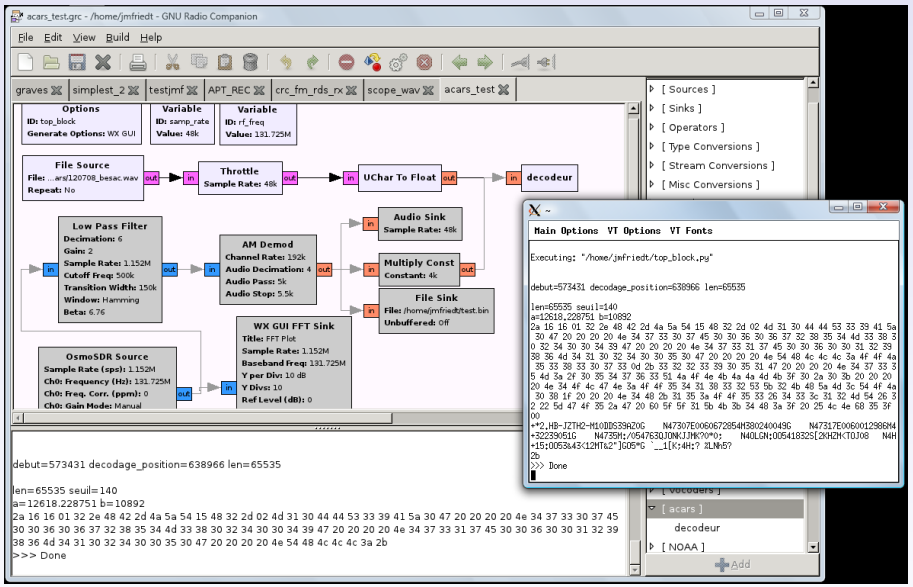
Processing sequence

1. for each chunk of data, remove average and convolve with 13 periods of 2400 Hz sine wave ($FFT(a \otimes b) = FFT(a) \cdot FFT(b)$)
2. threshold to detect whether a message is started
3. if so, wait to accumulate enough data to be sure one full message is recorded (max 300 characters)
4. convolution with one period of 2400 and 1200 sine waves, synchronized on initial message rising front (**very sensitive** to synchronization alignement)
5. identify most probable bit value (2400>?1200 Hz)
6. from the bit sequence, extract message (0 means bit change, 1 means bit propagate)
7. check header 0x2B 0x2A 0x16 0x16 0x01: if so, interpret sequence of bits as bytes (plane identifier, flight identifier, message sequence, message content)

Result: decoding a digital protocol



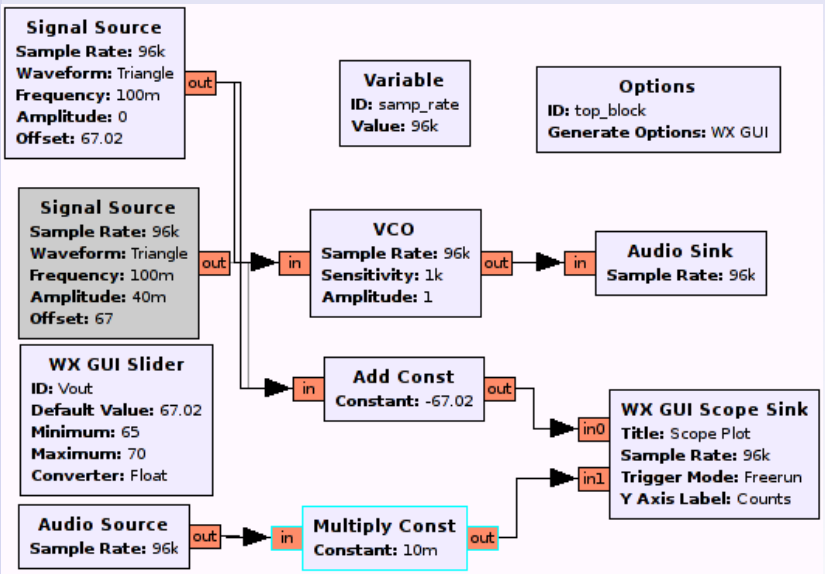

```
binaire=fft_decod('acars_orleans.wav',101001+2.15e6,
101001+2.15e6+40000,7000);

2b 2a 16 16 01 58 2e 47 2d 45 55 55 47 15 48 31
39 02 43 30 33 41 42 41 39 32 31 36 23 43 46 42
57 52 4e 2f 57 4e 31 32 30 36 33 30 30 38 33 33
30 30 33 34 30 30 30 30 30 36 4e 41 56 20 49 4c
53 20 32 20 46 41 55 4c 54 20 20 20 20 20 20 20
20 20 0d 03 6d 1d 7f 7f 7f 7f 7f 7f 7f
m*X.G-EUUGH19C03ABA9216#CFBWRN/WN12063008330034
000006NAV ILS 2 FAULT
CRC:00000000000000000000000000000000000000000000
00000000000000000000000000000000000000-100-1-1-1-1-1-1
```

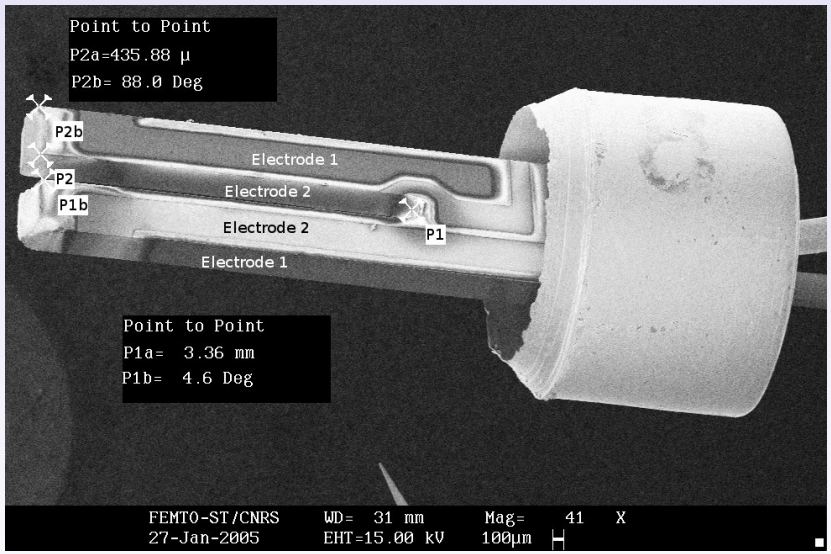


https://www.cgran.org/wiki/ACARS

Network analyzer and tuning fork characterization

- GNURadio: playground for implementing and testing protocols compatible with real time processing of data streams
- Current testing approach (getting started): use of the sound card as emitter and receiver. Full duplex ⇒ audio network analyzer
- Application to the quartz tuning fork (32768 Hz)

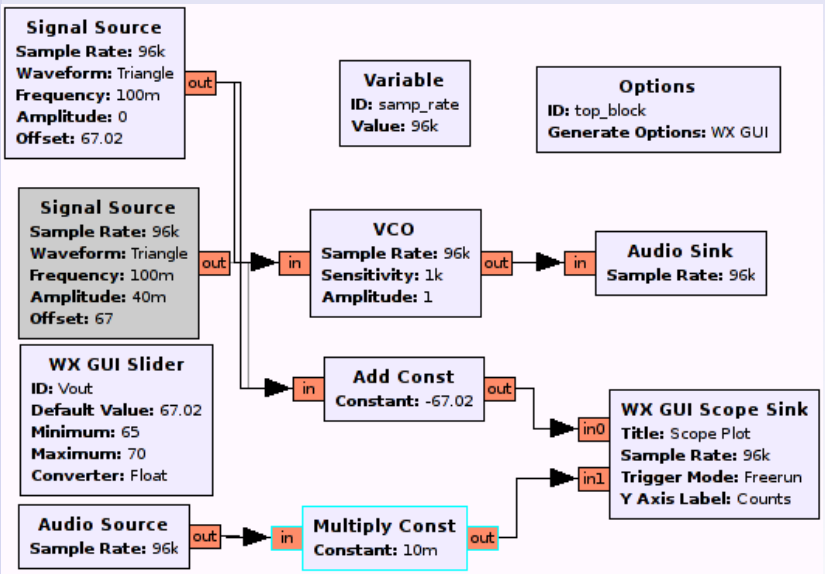




- Narrowband device (32768±10 Hz) ⇒ comparator to convert sine to square & use 3rd overtone + amplitude detector (diode+LPF)

Network analyzer and tuning fork characterization

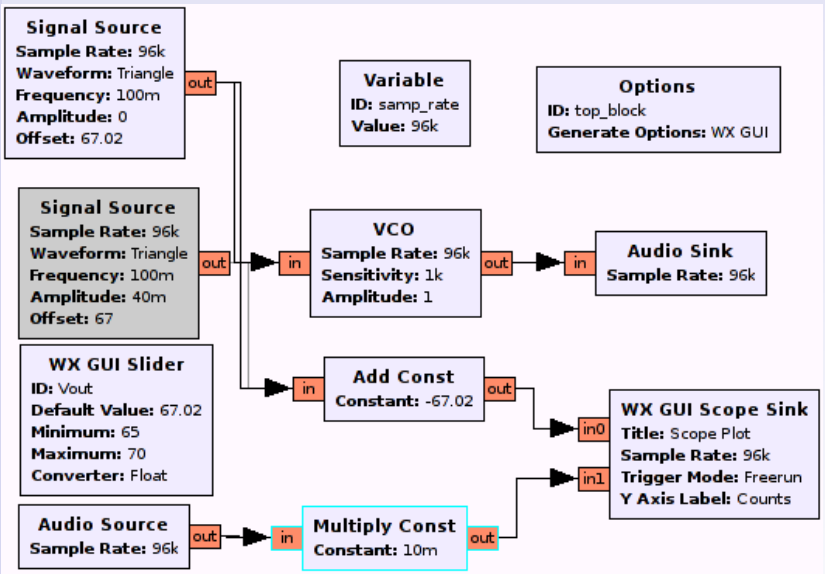
- GNURadio: playground for implementing and testing protocols compatible with real time processing of data streams
- Current testing approach (getting started): use of the sound card as emitter and receiver. Full duplex ⇒ audio network analyzer
- Application to the quartz tuning fork (32768 Hz)

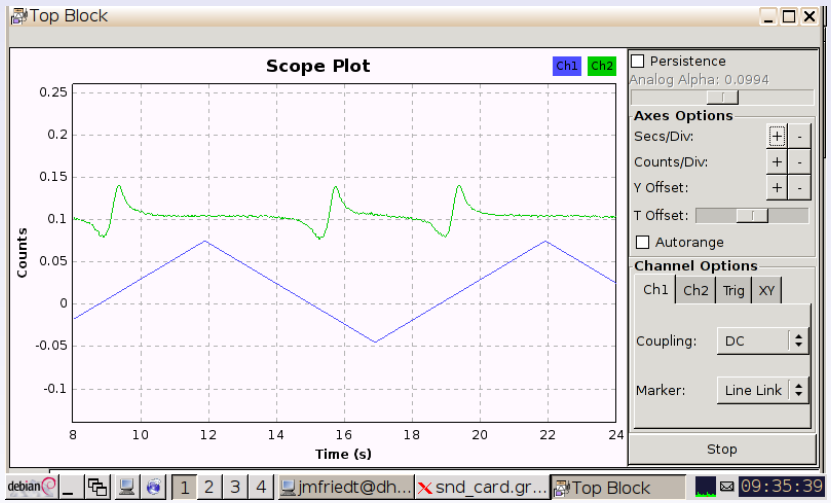


- Narrowband device (32768±10 Hz) ⇒ comparator to convert sine to square & use 3rd overtone + amplitude detector (diode+LPF)

Network analyzer and tuning fork characterization

- GNURadio: playground for implementing and testing protocols compatible with real time processing of data streams
- Current testing approach (getting started): use of the sound card as emitter and receiver. Full duplex ⇒ audio network analyzer
- Application to the quartz tuning fork (32768 Hz)

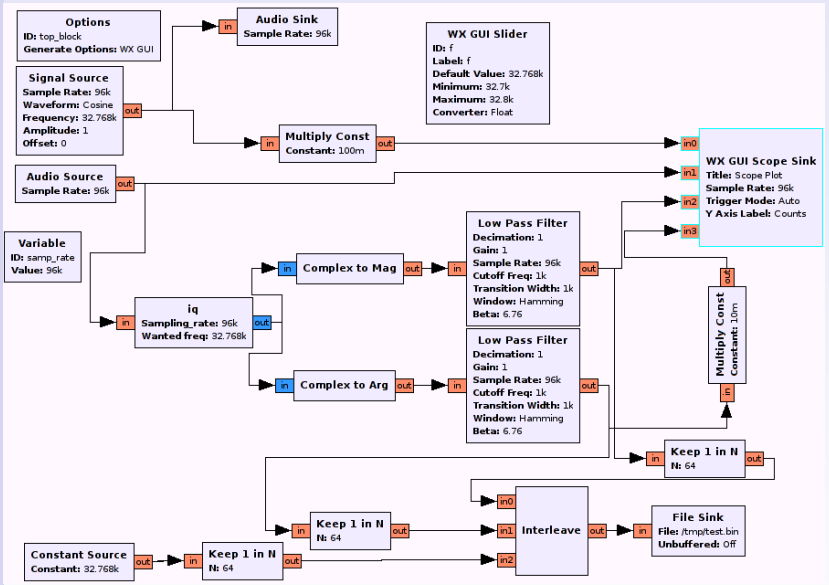


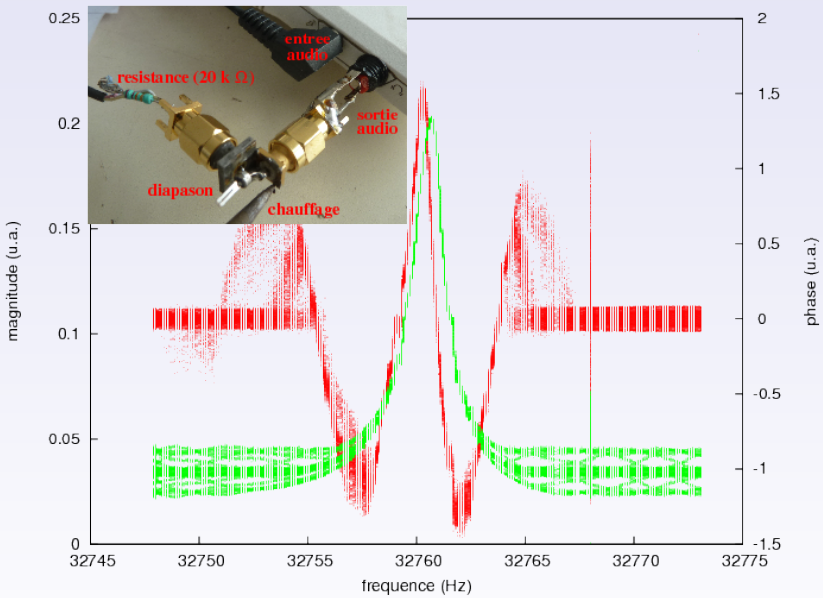


- Narrowband device (32768±10 Hz) ⇒ comparator to convert sine to square & use 3rd overtone + amplitude detector (diode+LPF)

Audio-frequency network analyzer

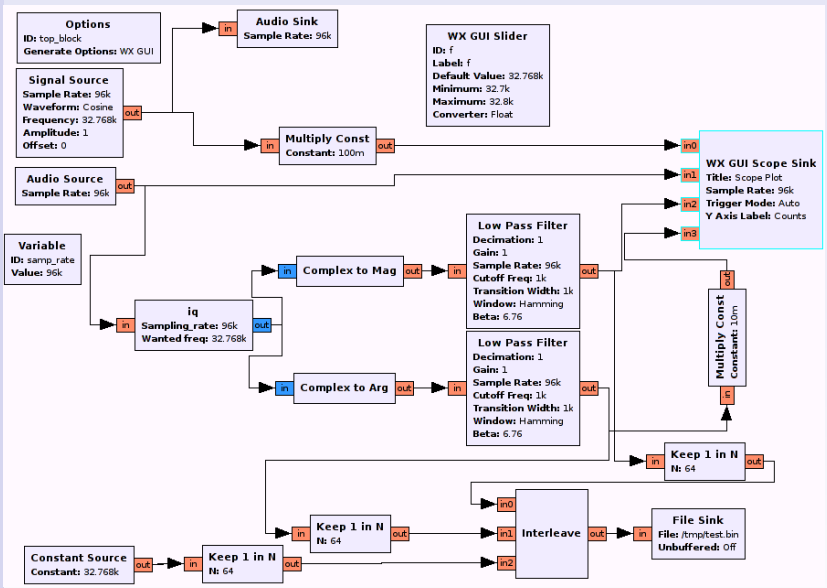
- Digital implementation of the I/Q demodulator: $\cos(\omega t + \varphi(t)) \times \cos(\omega_0 t)$ and $\cos(\omega t + \varphi(t)) \times \sin(\omega_0 t)$ followed by sliding average to extract $\tan(\varphi(t)) = \frac{Q}{I}$
- If $\omega \neq \omega_0$, linear fit to remove the $\omega - \omega_0$ phase drift.
- DO NOT use $\cos(\omega_0 t)$: quickly diverges as $t \uparrow$. See `gnuradio-core/src/lib/general/gr_nco.h`: $\varphi+ = 2\pi f / f_s [2\pi]$

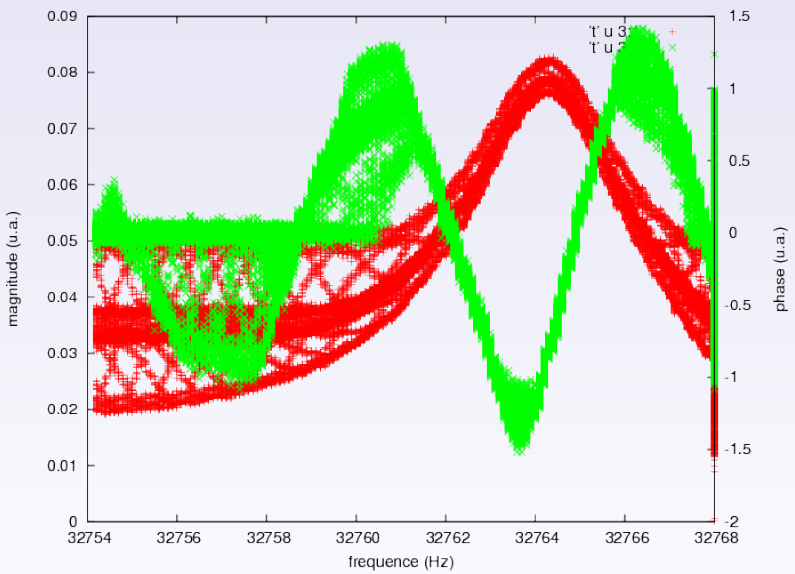




Audio-frequency network analyzer

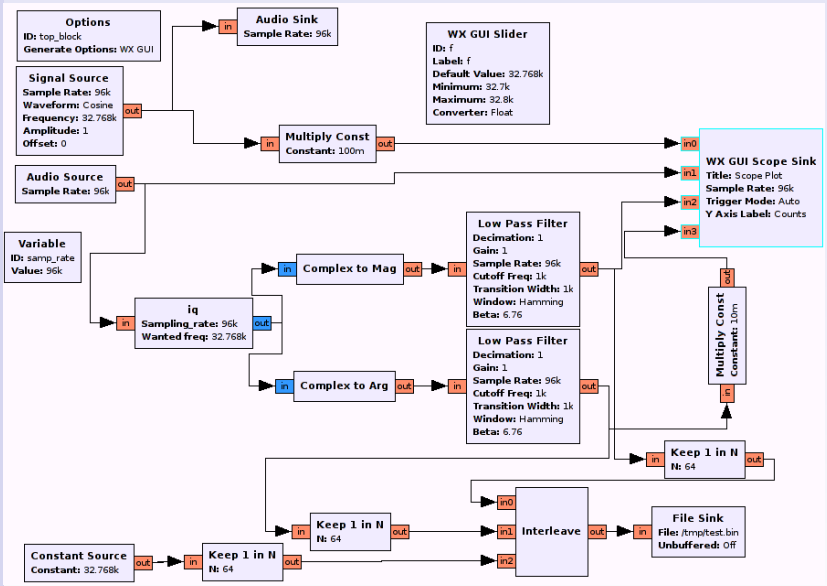
- Digital implementation of the I/Q demodulator: $\cos(\omega t + \varphi(t)) \times \cos(\omega_0 t)$ and $\cos(\omega t + \varphi(t)) \times \sin(\omega_0 t)$ followed by sliding average to extract $\tan(\varphi(t)) = \frac{Q}{I}$
- If $\omega \neq \omega_0$, linear fit to remove the $\omega - \omega_0$ phase drift.
- DO NOT use $\cos(\omega_0 t)$: quickly diverges as $t \uparrow$. See `gnuradio-core/src/lib/general/gr_nco.h`: $\varphi+ = 2\pi f / f_s[2\pi]$

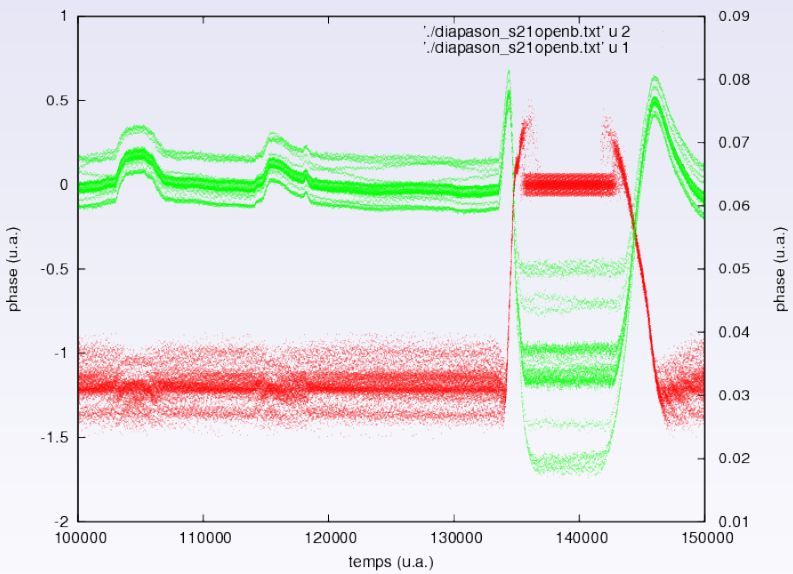




Audio-frequency network analyzer

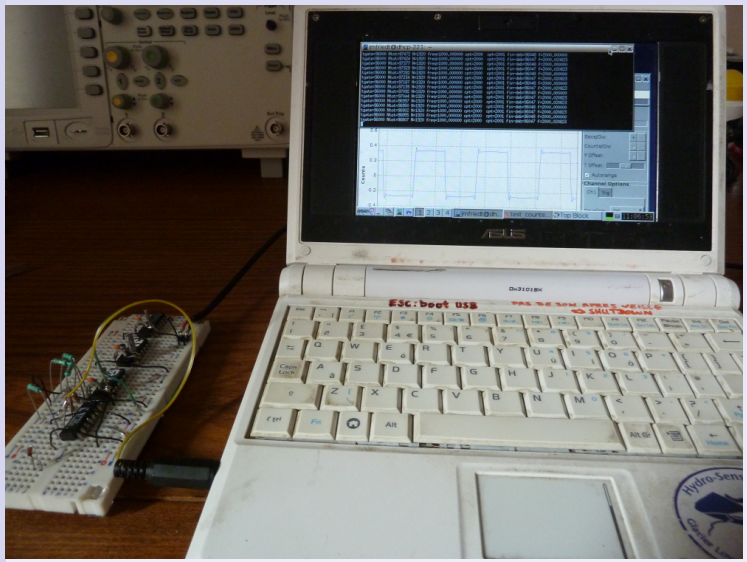
- Digital implementation of the I/Q demodulator: $\cos(\omega t + \varphi(t)) \times \cos(\omega_0 t)$ and $\cos(\omega t + \varphi(t)) \times \sin(\omega_0 t)$ followed by sliding average to extract $\tan(\varphi(t)) = \frac{Q}{I}$
- If $\omega \neq \omega_0$, linear fit to remove the $\omega - \omega_0$ phase drift.
- DO NOT use $\cos(\omega_0 t)$: quickly diverges as $t \uparrow$. See `gnuradio-core/src/lib/general/gr_nco.h`: $\varphi+ = 2\pi f / f_s[2\pi]$

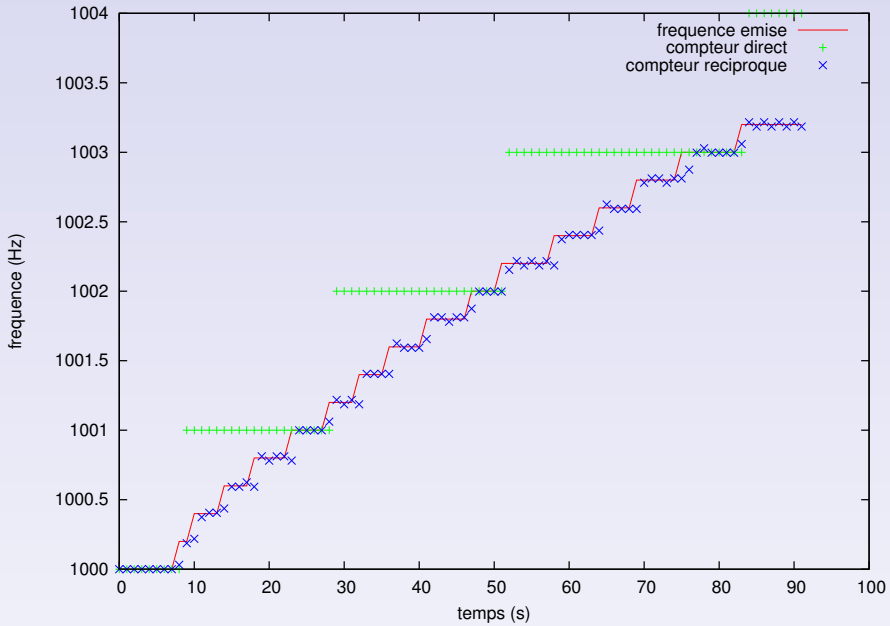




Example of the frequency counter

Implementing the direct and reciprocal frequency counters (96 kHz sampling rate on the sound card)





Left: quartz tuning fork experimental setup Right: synthesized signal

$\Delta f_i = 1/T_{gate} \rightarrow \Delta f_i = \frac{f_i}{f_r \times T_{gate}}$: resolution gain is f_r/f_i, or 48-fold if $f_r = 96$ kHz and $f_i \simeq 2$ kHz,

$\Rightarrow$ basic tool to assess the sampling rate to measured signal frequency influence on measurement resolution

FMCW RADAR

RADAR systems:

1. pulsed RADAR: **high power**, short, pulse travels until it reaches a target, reflects and a **high bandwidth** (fast) receiver records the returned electromagnetic signal
2. Frequency-Modulated Continuous Wave (FMCW) RADAR: continuous emission of a frequency-swept signal, and mixing between LO and RF provides audio-frequency output [9] [10]

[9] G.L. Charvat, J.H. Williams, A.J. Fenn, S.Kogon, & J.S. Herd. RES.LL-003 *Build a Small Radar System Capable of Sensing Range, Doppler, and Synthetic Aperture Radar Imaging*, January IAP 2011. (Massachusetts Institute of Technology: MIT OpenCourseWare), `http://ocw.mit.edu` (Accessed 28 Jul, 2013).

[10] G.L. Charvat, A.J. Fenn, & B.T. Perry, *The MIT IAP radar course: Build a small radar system capable of sensing range, Doppler, and synthetic aperture (SAR) imaging*, IEEE Radar Conference (RADAR), 2012, pp. 0138–0144

FMCW RADAR: principle

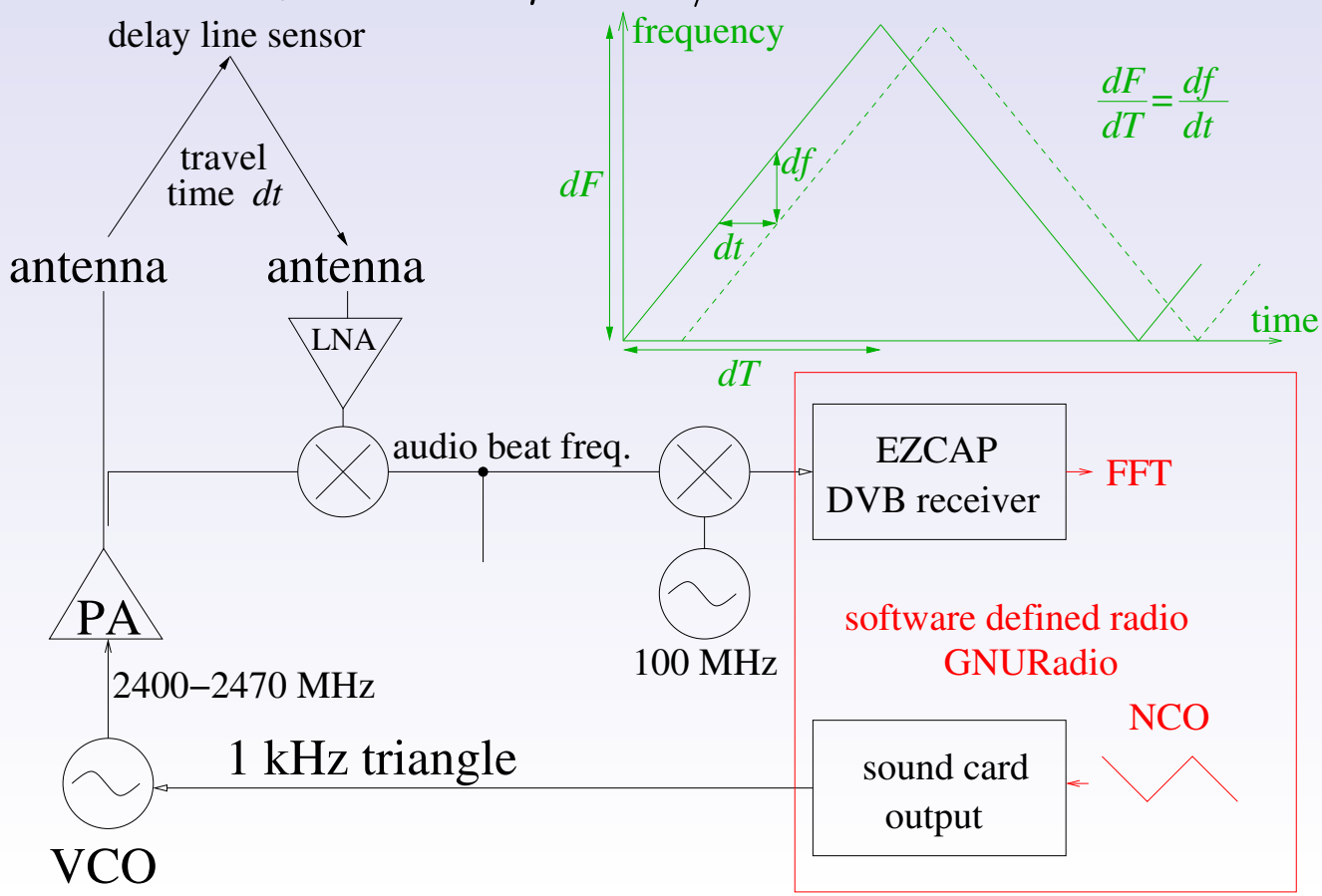
Degrees of freedom: frequency range dF and sweep rate dT

$$\frac{dF}{dT} = \frac{df}{dt}$$

when echo is located dt away

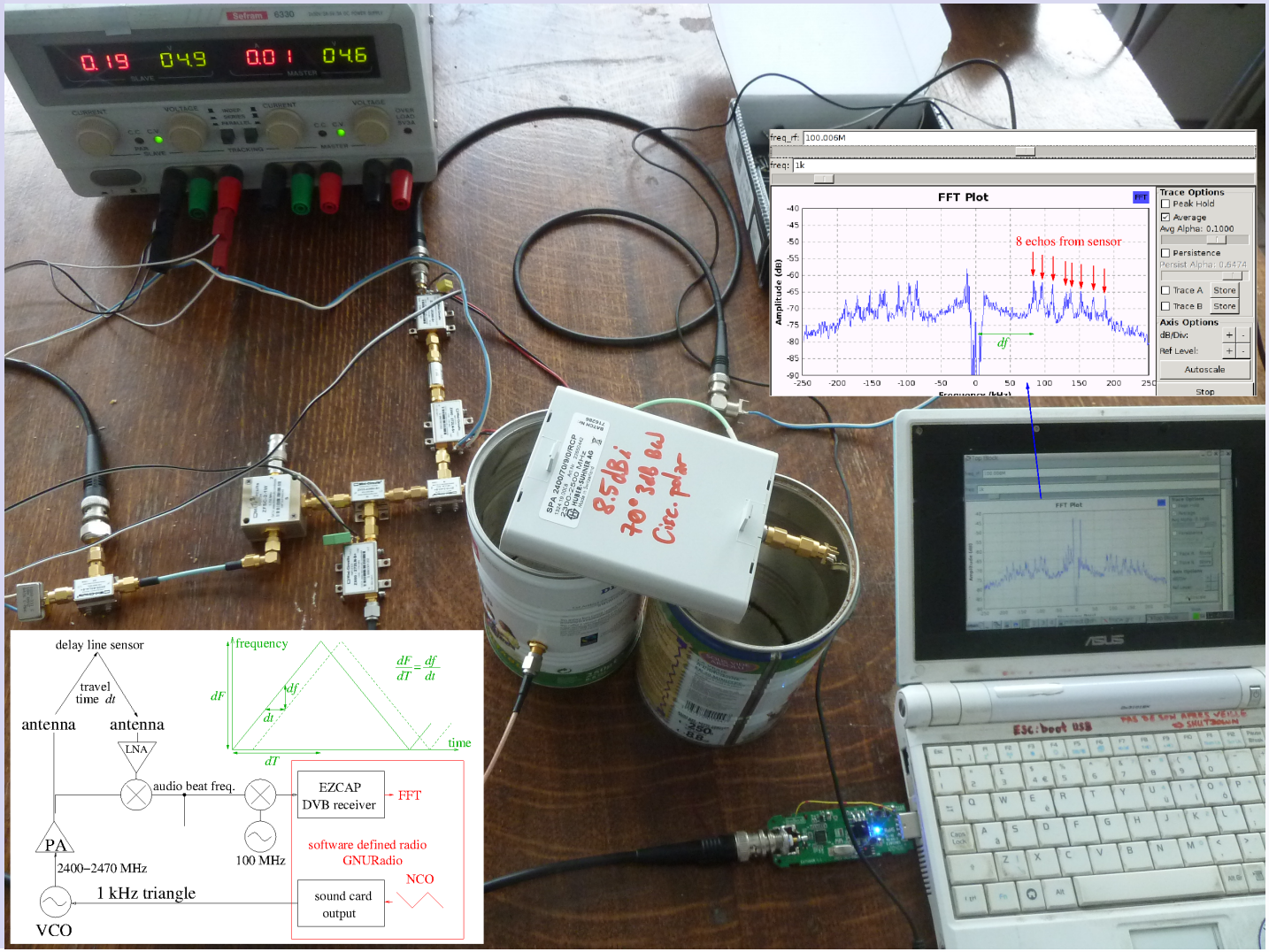
Numerical application: $df < 96$ kHz for $dt < (2 \times 15 \text{ m})/(3 \cdot 10^8 \text{ m/s})$

$\Rightarrow$ if $dF = 50$ MHz, $dT > 52$ μs or $1/dT < 19200$ Hz



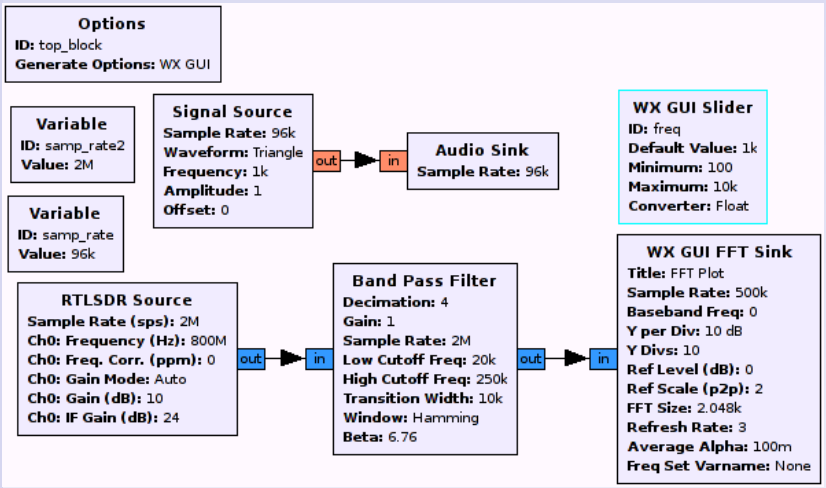
FMCW RADAR

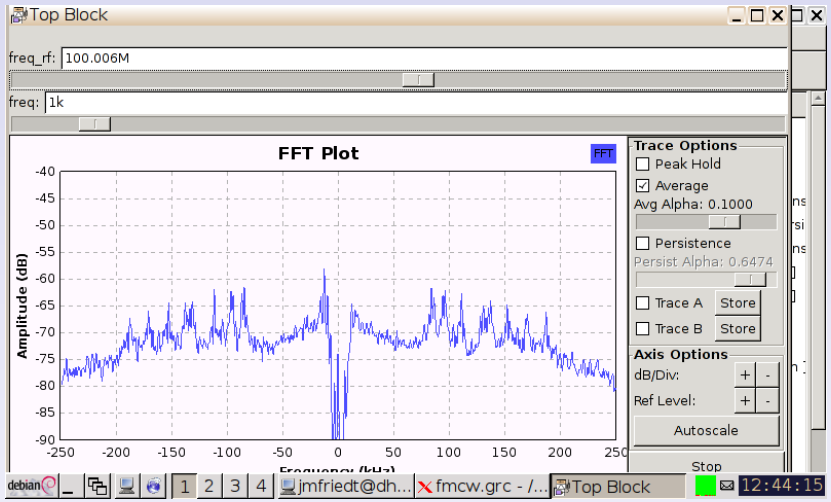
- Audio out: adapt wave shape to linearize the VCO output (freq. vs voltage)
- **Real time** FFT for displaying echo distance



FMCW RADAR

- Audio out: adapt wave shape to linearize the VCO output (freq. vs voltage)
- **Real time** FFT for displaying echo distance





- 8-echoes are identified, located 1-4 μs (simulated echo generator using a *surface acoustic wave* delay line)
- increased sweep rate (>400 Hz) for optimum sound card operating range
- DVB receiver for increased bandwidth
  (4 μs$\times$50 MHz$\times$400 Hz=80 kHz $\gg$ 48 kHz accessible with a sound card)

Conclusion

Results:

- SDR: digital processing for improved **stability** & **flexibility** (software prototyping)
- **reusable** software through multiple sources
- part of an active **opensource** project
- **existing** basic processing blocks
- initial graphical user interface or Python programming
- **workflow** from decoder prototyping (offline) to GNURadio compatible block (inline processing)

Further activities:

- NOAA weather satellite reception[11]
- GPS reception [12]
- add your own interfaces: DDS & radiomodem sinks, ADC and FTDI USB source (issue of data rate) – `gnuradio-core/src/lib/io/gr_file_source.cc`

[11] `http://www.oz9aec.net/index.php/gnu-radio/gnu-radio-blog/477-noaa-apt-reception-with-gqrx-and-rtlsdr`

[12] `http://www.gnss-sdr.org/` and specifically `http://www.gnss-sdr.org/documentation/gnss-sdr-operation-realtek-rtl2832u-usb-dongle-dvb-t-receiver`

Selected bibliography

D.A. Mindell, *Digital Apollo: Human and Machine in Spaceflight*, The MIT Press, 2011

J. Hamkins & M.K. Simon, *Autonomous Software-Defined Radio Receivers for Deep Space Applications*, Deep Space Communications and Navigation Series Vol. 9, at http://descanso.jpl.nasa.gov/Monograph/series9/Descanso9_Full_rev2.pdf

P.B. Kenington *RF and baseband techniques for software defined radio* Artech House (2005)

D.J. Mudgway, *Uplink-Downlink – A History of the Deep Space Network, 1957–1997*, NASA SP-2001-4227, The NASA History Series (2001), at history.nasa.gov/SP-4227/Uplink-Downlink.pdf

M.K. Simon, *Bandwidth-Efficient Digital Modulation with Application to Deep-Space Communications*, Deep Space Communications and Navigation Series Vol. 3, at http://descanso.jpl.nasa.gov/Monograph/series3/complete1.pdf

N. Foster, *Tracking Aircraft With GNU Radio*, GNU Radio Conference (2011), at http://gnuradio.org/redmine/attachments/download/246/06-foster-adsb.pdf

T. McDermott, *Wireless Digital Communications: Design and Theory 2nd Ed.*, Tucson Amateur Packet Radio Corporation – TAPR (1998)

R.H.L. Stroop, *Enhancing GNU Radio for Run-Time Assembly of FPGA-Based Accelerators*, master thesis, Faculty of the Virginia Polytechnic Institute and State University (2012)

A.V. Oppenheim, *Discrete Time Signal Processing (3rd Ed.)*, Prentice Hall (2009), lectures available at http://ocw.mit.edu/resources/res-6-007-signals-and-systems-spring-2011/video-lectures/

J.G. Proakis, D.K. Manolakis, *Digital signal processing (4th Ed.)*, Prentice Hall (2006)

SENSEOR

femto-st SCIENCES & TECHNOLOGIES

English: http://jmfriedt.free.fr/en_sdr.pdf

French: http://jmfriedt.free.fr/lm_sdr.pdf