GNURadio as a general purpose digital signal processing environment

G. Goavec-Mérou & J.-M Friedt

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

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

February 2, 2014

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 ?)

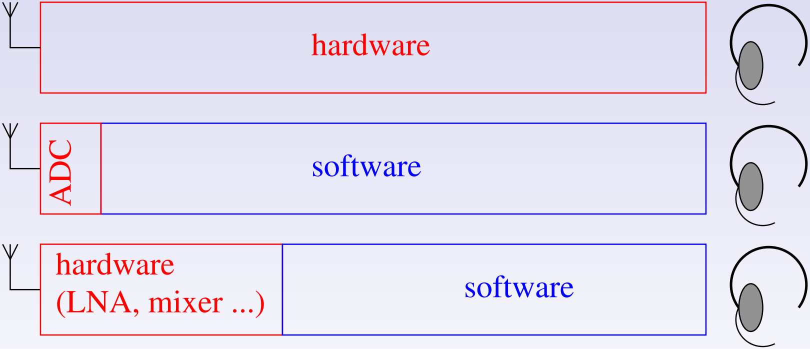
⇒ shift from hardware to software

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

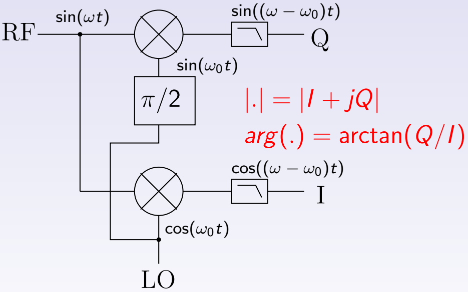
From all hardware receiver to a single front-end A/D converter (ADC) followed by software digital signal processing
→ not applicable due to A/D bandwidth and memory usage [2]



[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

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

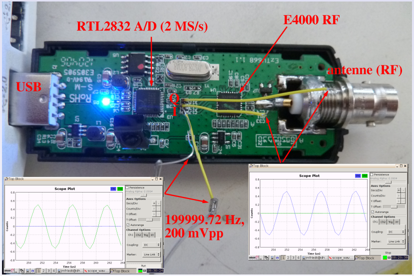
[4] Agilent, *Digital Modulation in Communications Systems – An Introduction*, Application Note 1298, or M. Steer, *Microwave and RF design – a systems approach*, SciTech Publishing, Inc (2010)

The GNURadio environment

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

- input (USRP, DVB receiver, sound card ...)
- 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×4⇒ 2800/92 ≃ 30=2.5 bits

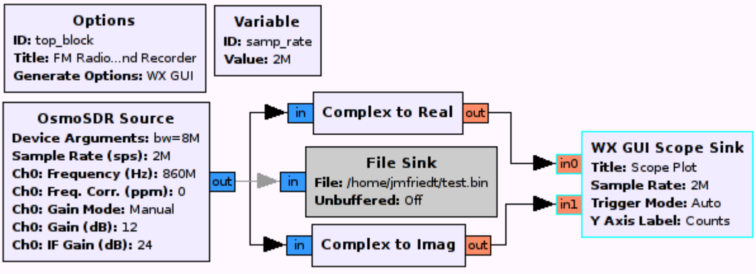
[5]Application Note AN2668, *Improving STM32F101xx and STM32F103xx ADC resolution by oversampling*, ST Microelectronics, 2008

The GNURadio environment

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

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

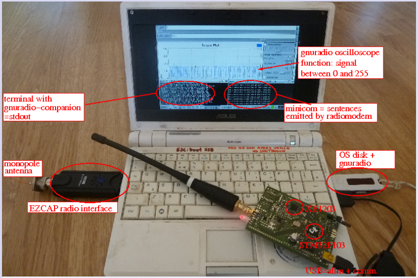
`gnuradio-companion`: GUI for assembling blocks and generator of Py

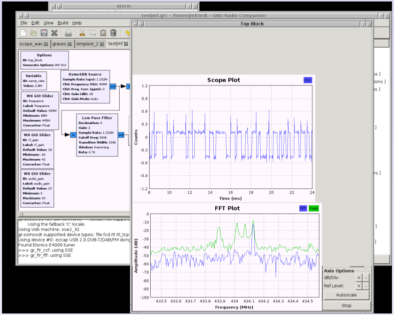


- 8-bit ADC for high bandwidth (oversampling does not compensate for low resolution: [5]) 1 bit/sampling rate$\times 4 \Rightarrow 2800/92 \simeq 30$=2.5 bits

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

GNURadio basic use

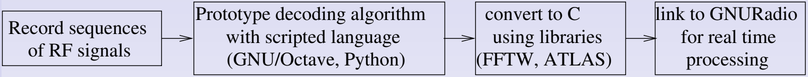




However, neither decoder for digital protocol I am interested in (ACARS), nor tools for time & frequency analysis ⇒ opensource tool, write your own if missing !

Write your own processing block

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



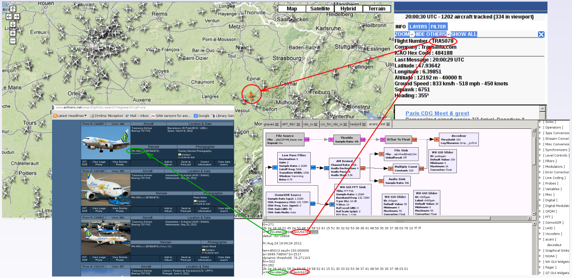
Development strategy:

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[6], used on VHF band (131.725 MHz in Europe):

1. encoding at 1200 (bit 0) and 2400 Hz (bit 1) [7]
2. data rate of 1200 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



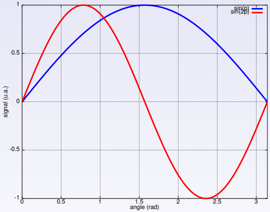
[6] http://files.radioscanner.ru/files/download/file4094/acars.pdf

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

Write your own processing block

Bit identification: many means to an end

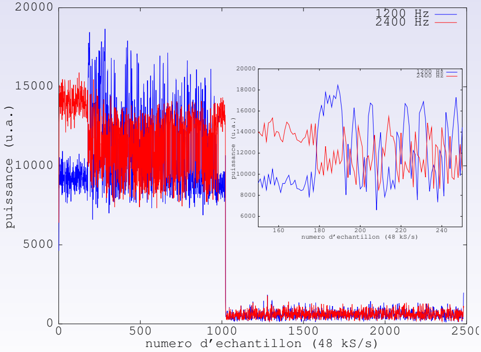
- time-domain band-pass filter (FIR) ... general purpose,
- convolution with the expected signals (1200 & 2400 Hz sine wave) $\Rightarrow$ frequeny domain (requires FFT),
- use at best the signal encoding properties
- $\int_0^1 \sin(2\pi t)\sin(\pi t)dt \propto \int_0^1 (\cos(3\pi t) - \cos(\pi t))\,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

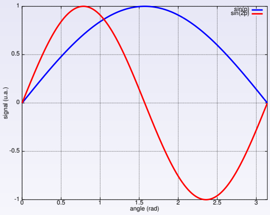


Write your own processing block

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) ⇒ frequeny domain (requires FFT),
- use at best the signal encoding properties



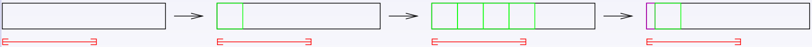


From GNU/Octave to C

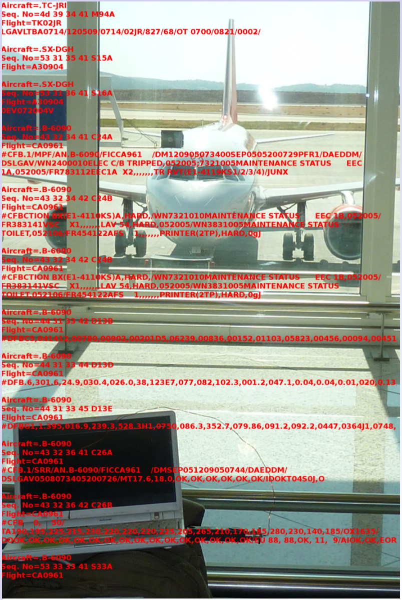
- Manual conversion. Could it be optimized ? (using Mathworks HDL Coder ?)
- FFT with different normalization convention ⇒ update threshold values
- From a complete (recorded) dataset to a stream of blocks of variable size

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.

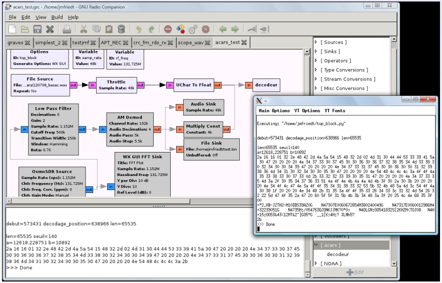


Result: decoding a digital protocol



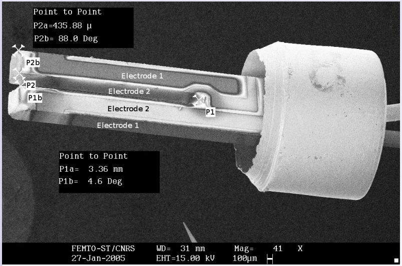

```
binaire=fft_decod('acars_orleans.wav',101001+2.15e6,
101001+2.15e6+40000,7000);

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:000000000000000000000000000000000000000000
00000000000000000000000000000000000000-100-1-1-1-1-1-1
```



Application to time & frequency

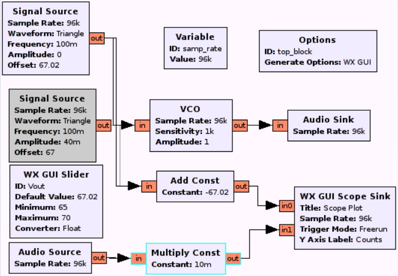
- GNURadio: **playground** for implementing and testing protocols compatible with real time processing of data streams
- beyond wireless data transmission: physics and feedback control
- Current testing approach (getting started): use of the sound card as emitter and receiver. Full duplex ⇒ audio network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

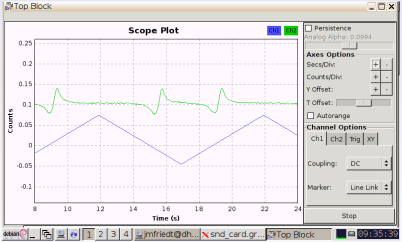


- Comparator to convert sine to square and use 3rd overtone + amplitude detector

Application to time & frequency

- GNURadio: **playground** for implementing and testing protocols compatible with real time processing of data streams
- beyond wireless data transmission: physics and feedback control
- 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)

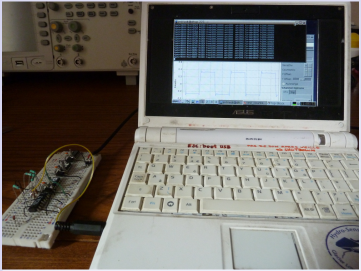


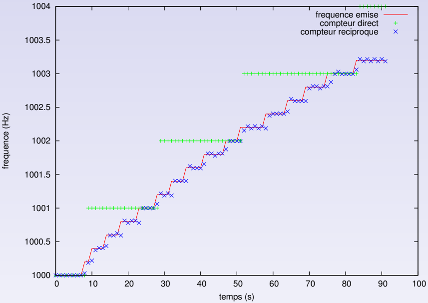


- Comparator to convert sine to square and use 3rd overtone + amplitude detector

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

New source to GNURadio

Example of the Semtech **SX1255** radiofrequency frontend on a Armadeus Systems **APF51**:

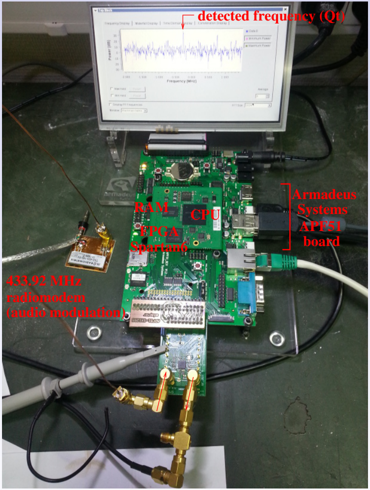
- emitter and receiver front ends operating in 400-510 MHz range
- I/Q input flow (modulator) and I/Q output flow (demodulator)
- I^2S data format (one channel I, one channel Q)
- variable decimation factor ($\Sigma\Delta$ output at 36 MHz, and decimates to get I/Q resolution)

→ grab I/Q data flow of the receiver and store (FPGA)

→ transfer from FPGA to CPU

→ transfer from kernel module to user space

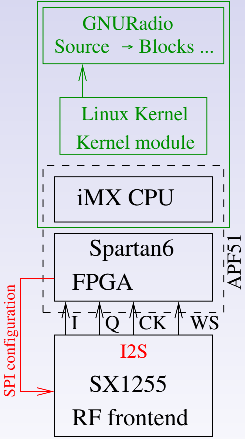
→ provide a GNURadio-compatible data source



New source to GNURadio

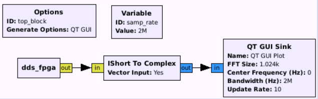
Development strategy:

1. source reads file from userspace (`throttle` block needed)
2. source reads from kernel module fed by FPGA running a DDS (kernel module communication to FPGA validated previously, independently of GNURadio)
3. FPGA reads from SX1255 (clock source) and stores in RAM before transfering to CPU (clock domain crossing)
4. CPU requests data from FPGA (FPGA RAM filled generates interrupt after a request for data from the CPU): one thread in the GNURadio source permanently acquires data from FPGA

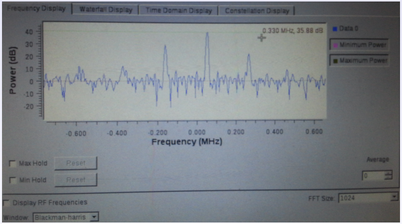


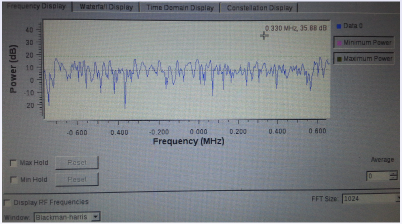
New source to GNURadio

- A source is a processing block with 0 input (item vector is the unit information size of `noutput_items`)
- I and Q interleaved output (two shorts): `ishort` GNURadio format



- `ishort` → `complex` processing block (vector input=yes)
- 36 MHz/9=4 MS/s at 9 bit I and Q resolution





Problem: continuous dataflow ?

TODO: dual buffers in the FPGA and the GNURadio source

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

Literature (PDF available at `http://jmfriedt.free.fr`):

- J.-M Friedt, G. Goavec-Mrou La rception radiofrquence définie par logiciel (Software Defined Radio – SDR), GNU/Linux Magazine France **153** (Oct. 2012), pp.4-33 [French, English translation at `jmfriedt.free.fr/en_sdr.pdf`]
- J. Marc, C. Canard, A. Vailly, V. Pichery, J.-M. Friedt Le diapason quartz comme capteur : utilisation de la carte son de PC pour l'instrumentation Bulletin de l'Union des Physiciens **107** (958), pp.1051-1076 (2013) [in French]
- J.-M Friedt Hacking the radiofrequency spectrum: GNURadio as a signal processing prototyping tool OHM 2013 (Observe, Hack, Make) (The Netherlands, 31 Jul-04 Aug. 2013)
- `https://www.cgran.org/wiki/ACARS` (tested on GNURadio 3.7)

Next step: port to Zynq platform

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 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)

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;
}
```