fosphor

GPU accelerated visualization of spectrum

Sylvain Munaut

GRCon 2013, October 2nd, 2013

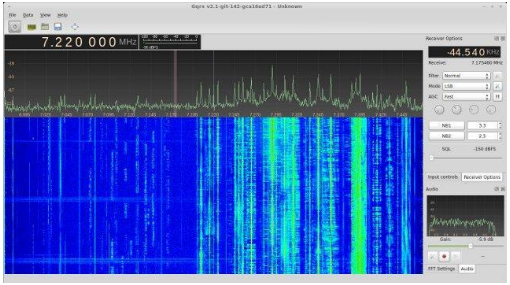
How to view spectrum
Motivation & Objectives

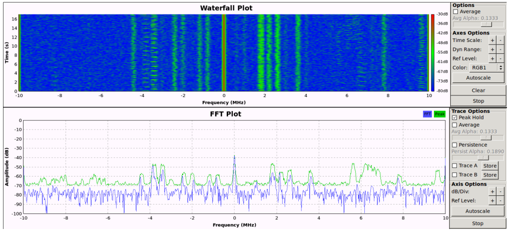
- Viewing spectrum is an essential instrumentation tool when working in radio
- Desirable properties:
  - Representative and Intuitive
    - Properly present the signal to see what's happening
    - Including short transients / burts
    - Easy to understand
  - Fast
    - CPU should be available to do "real" work (demod, ...)
  - Visually appealing
    - Not strictly required, but can't hurt

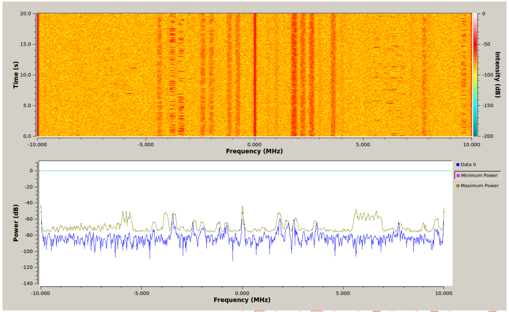
How to view spectrum

GNURadio

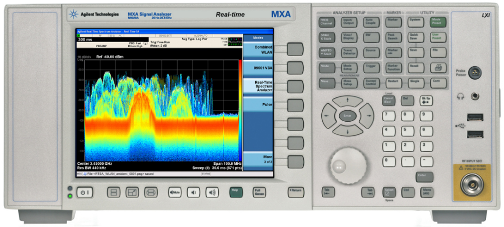
This is what's available currently

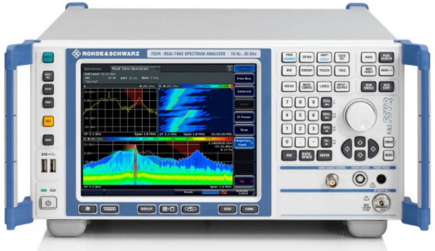


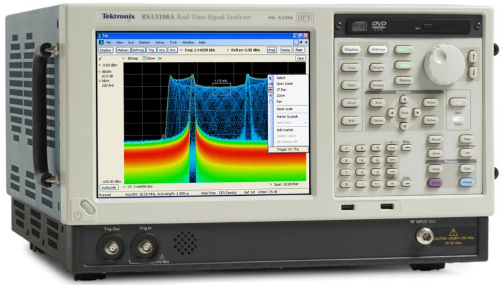




How to view spectrum
RTSA



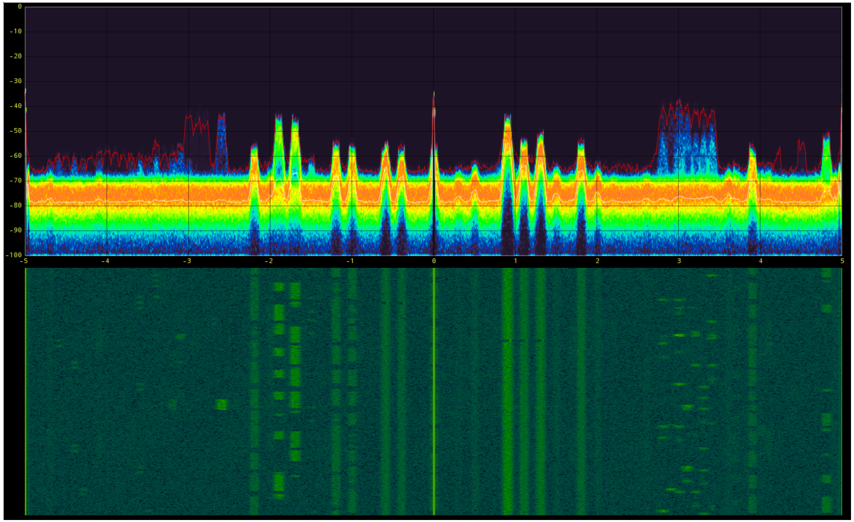




This is what I want

How to view spectrum

fosphor



What it is (1)

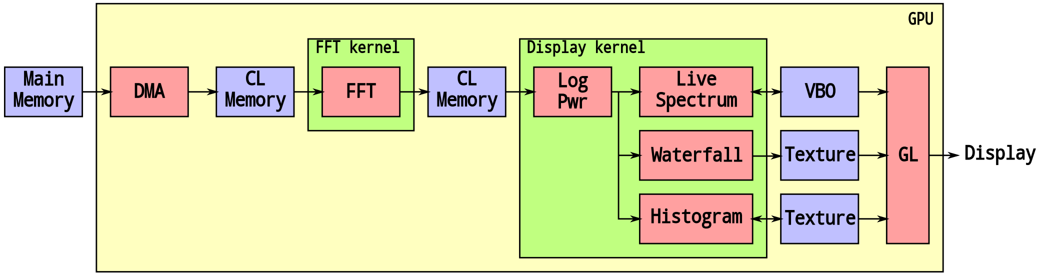
- First and foremost, it's *eye-candy* !
  - GPU accelerated re-implementation of `SDRangeLove` main display effect
  - Similar to Agilent/Tek/R&S DPX on their RTSA
- It's fast
  - That was kind of the whole point ...
  - `fosphor`: On my laptop (NVidia G96M) $\approx$ 30 Msps. On a ATI HD6850 $\approx$ 250 Msps
  - More CPU hungry than I would have liked
- OpenCL / OpenGL mix
  - You need working CL/GL interop card and drivers

What it is (2)

4 main parts :

- FFT
  - Process every samples at the input in at least one FFT window
  - Ideally in several, using overlapping windows (future)
- Live spectrum
  - IIR average of all computed spectra
  - So many of them that noise is smoothed while keeping responsiveness
- Waterfall
  - Again, all spectra used
  - 1:1 currently, in the future 1:2^n using aggregation (min/max/avg)
- Histogram
  - Statistical view of spectra
  - Perfect to see bursts, glitches or any transients

How it works
Architecture



How it works
FFT

- DFT:
  $X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$
  - Way too slow
- Use CooleyTukey FFT algorithm:
  - Divide and Conquer, easy to use in //
  - using radix-2,-4,-8 as bases
  - eg for 1024: 4 passes, 128x8, 128x8, 128x8, 512x2
- Currently limited to what fits in OpenCL local memory (16k min, 1024 FFT). Can be extended using:
  - Split real/imag store/load (double space)
  - Newer GPU with 32k / 64k local memory
  - Global memory shuffle (won't be implemented)
- Batch processing
  - 2D OpenCL job: FFT size x batch size. Workgroup = FFT size x 1.
  - Helps data transfer and increase # threads (better GPU usage)

How it works
Live spectrum

- Fills an OpenGL VBO (Vertex Buffer Object)
  - x = FFT bin number, y = FFT bin power
- FFT bin power:
  $x_n = \log_{10} \sqrt{Real(w_n)^2 + Imag(w_n)^2}$
  - No need for proper scaling to dB, can be done using OpenGL
  - Even the square root is not needed, but GPU have dedicated hardware for `hypot`
- Drawn using `glDrawArrays(GL_LINE_STRIP, 0, FFT_LEN);`
  - Scaling, shifting done dynamically with OpenGL geometry transforms
- IIR filtered:
  $y_n = (1 - \alpha) \cdot y_{n-1} + \alpha \cdot x_n$
- Batch processing:
  $y_n = (1 - \alpha)^N \cdot y_{n-N} + \alpha \cdot \sum_{i=0}^{N-1} \left[(1 - \alpha)^i \cdot x_{n-i}\right]$

How it works
Spectrogram / Waterfall

- Fills an OpenGL `FLOAT` texture
  - 1:1 mapping with the FFT kernel output buffer
  - In the future 1:2^n with various aggregation functions (min/max/avg)
- Drawn on textured quad
  - Use texture coordinates for centering, zooming, sliding, ...
- Dynamically mapped (shift/scale) to a palette using a pixel shader
  - Can change range / scale without recompute
  - Optional tri-linear filtering shader (smooth zoom)

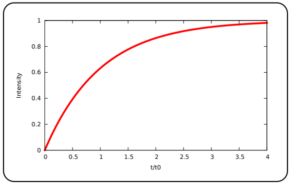
How it works

Histogram (1)

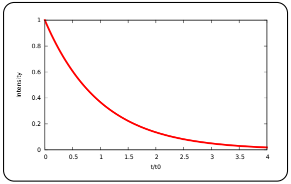
- Emulates phosphor persistence of the spectrum
  - Exact function based on charge / discharge of a capacitor
- Fills an OpenGL `FLOAT` texture
  - Each pixel represent how often a given FFT bin is at a given power level
  - Contains a normalized intensity between 0.0f and 1.0f
- Drawn on textured quad
  - And, again, using OpenGL shader for mapping intensity range to a palette
- Requires OpenCL 1.1 atomic increments
  - But since my own laptop doesn't support those, there is a hw specific "hack" for NVidia's SM11 based GPUs.

How it works
Histogram (2)

Charge / Rise:



Discharge / Decay:



$$\begin{aligned}
y_r(t) &= 1 - e^{-\frac{t}{t_{0r}}} \\
\frac{d}{dt} y_r(t) &= \frac{1}{t_{0r}} \cdot e^{-\frac{t}{t_{0r}}} \\
&= \frac{1}{t_{0r}} \cdot (1 - y_r(t)) \\
y_r(t + \Delta t) &\simeq y_r(t) + \Delta t \cdot \frac{d}{dt} y_r(t) \\
&\simeq y_r(t) \cdot \left(1 - \frac{\Delta t}{t_{0r}}\right) + \frac{\Delta t}{t_{0r}}
\end{aligned}$$

$$\begin{aligned}
y_d(t) &= e^{-\frac{t}{t_{0d}}} \\
\frac{d}{dt} y_d(t) &= -\frac{1}{t_{0d}} \cdot e^{-\frac{t}{t_{0d}}} \\
&= -\frac{1}{t_{0d}} \cdot y_d(t) \\
y_d(t + \Delta t) &\simeq y_d(t) + \Delta t \cdot \frac{d}{dt} y_d(t) \\
&\simeq y_d(t) \cdot \left(1 - \frac{\Delta t}{t_{0d}}\right)
\end{aligned}$$

How it works
Histogram (3)

Combined and adapted for discrete steps : ($x_n \in \{0, 1\}$ represents a histogram "hit")

$$y_n = y_{n-1} + \underbrace{\frac{1}{t_{0d}} \cdot y_{n-1}}_{\text{Decay}} + \underbrace{\frac{1 - y_{n-1}}{t_{0r}} \cdot x_n}_{\text{Rise}} = y_{n-1} \cdot \left[1 - \frac{1}{t_{0d}} - \frac{x_n}{t_{0r}}\right] + \frac{x_n}{t_{0r}}$$

And for batch processing, let h_c be the number of histogram "hits" within the batch and we approximate that those are uniformly distributed within the batch.

$$y_n \simeq y_{n-1} \cdot \underbrace{\left[1 - \frac{1}{t_{0d}} - \frac{h_c}{t_{0r}}\right]}_{A} + \underbrace{\frac{h_c}{t_{0r}}}_{B} \qquad \text{with} \qquad h_c = \frac{1}{N} \cdot \sum_{i=0}^{N-1} x_{n-i}$$

$$\simeq y_{n-N} \cdot A^N + B \cdot \sum_{i=0}^{N-1} A^i$$

$$\simeq y_{n-N} \cdot A^N + B \cdot \left[\frac{A^N - 1}{A - 1}\right]$$

Future

- Testing
  - OS: Linux / OSX / Win32
  - GPU: NVidia / Radeon / Intel
  - APU: AMD
  - And all combinations thereof
- Better GNURadio integration
  - Proper QT & WX blocks
  - Expose more to GRC
- Expose settings
  - FFT size, timing constants, batch size, color palette ...
  - Some at init time, some dynamically
- Function improvements
  - FFT length (512,2048,4096)
  - FFT Overlap
  - Waterfall spectra aggregation
  - Cursors

Thanks

Thanks to anyone contributing to the various Open Source SDR projects.
For this one in particular :

- Christian "*Hopscotch*" Daniel
- Dimitri "*horiz0n*" Stolnikov
- Eric "*Hoernchen*" Wild
- Steve "*steve-m*" Markgraf

Resources

- fosphor
  - `http://cgit.osmocom.org/gr-fosphor/`
  - `git://git.osmocom.org/gr-fosphor.git`
- SDRangelove
  - `http://sdrangelove.org`
  - `http://cgit.osmocom.org/sdrangelove/`
  - `git://git.osmocom.org/sdrangelove.git`
- "Implementation of Real-Time Spectrum Analysis", 2011, Dr. Florian Ramian
  - `http://cdn.rohde-schwarz.com/dl_downloads/dl_application/application_notes/1ef77/1EF77_0E.pdf`

Questions ?

Questions ?