RFNoC: fosphor

How to apply RFNoC to RTSA display acceleration

Sylvain Munaut

FOSDEM 2015, February 1st, 2015

About the speaker

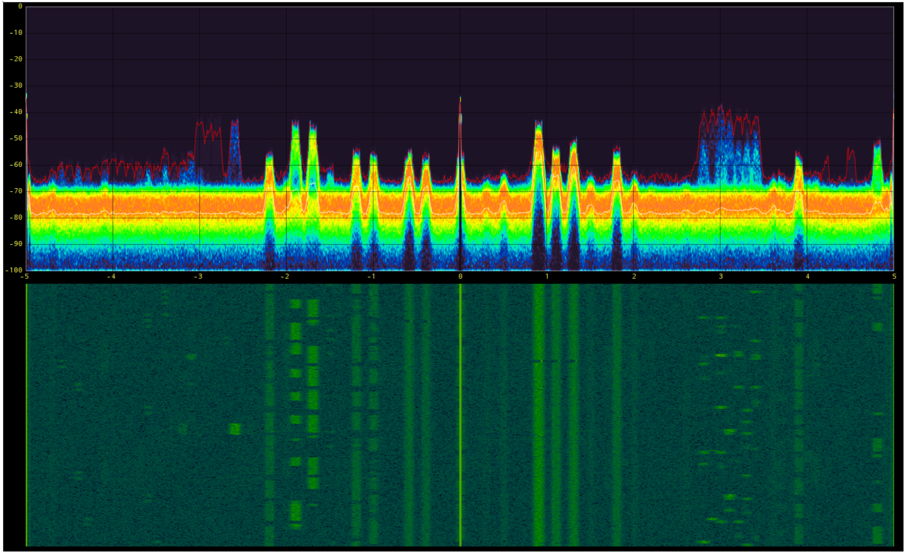
- Linux and free software enthusiast since 1999
- M.Sc. in C.S. + some E.E.
- General orientation towards low level
  - Embedded, Kernel, Drivers and such.
  - Hardware (Digital stuff, FPGA, RF, ...)
- Interest in RF telecom for about 5 years
  - GSM, GMR-1, TETRA, POCSAG, ...
  - Within the Osmocom project
  - Mostly in my spare time

Outline

gr-fosphor: Description

- First and foremost, it's *eye-candy* !
- GPU accelerated spectrum visualization
  - Uses a mix of OpenCL and OpenGL to achieve high speed (200+ Msps)
  - Can use software OpenCL implementation if no GP-GPU is available
  - Appears as a GR sink (WX & Qt & GLFW backends)
- All input samples are used
  - Perfect to see bursts / transients
- 3 main displayed elements:
  - Live spectrum lines (average + max-hold)
  - Histogram (statistical view of frequency/power distribution)
  - Waterfall / Spectrogram
- See the GRCon 2013 slides for all the dirty details

gr-fosphor: Action shot



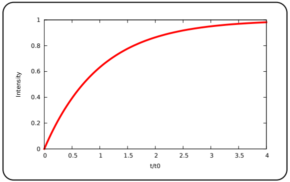
RFNoC fosphor: Scope and Motivation

- Accelerate fosphor using FPGA rather than GPU
- Focus solely on the histogram
  - Spectrum lines: Already doable in RFNoC today, will be integrated later
  - Waterfall: Not much can be done in FPGA
- `gr-fosphor` can be really fast, but :
  - Requires a pretty fast GPU for very high rates
  - Some systems (E310) don't have a GPU at all
  - Requires shipping all the RF samples to the host
- Architecture :
  - Re-use existing RFNoC FFT block
  - Compute all the statistics on the FPGA
  - Ship only display data to the host at 60 fps (or less)
    - Allows visualization of large bandwith with minimal host load

RFNoC fosphor: Histogram

Each texel is a capacitor

Charge / Rise:



$$y_r(t) = 1 - e^{-\frac{t}{t_{0r}}}$$

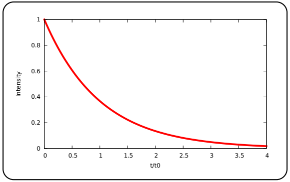
$$\frac{d}{dt} y_r(t) = \frac{1}{t_{0r}} \cdot e^{-\frac{t}{t_{0r}}}$$

$$= \frac{1}{t_{0r}} \cdot (1 - y_r(t))$$

$$y_r(t + \Delta t) \simeq y_r(t) + \Delta t \cdot \frac{d}{dt} y_r(t)$$

$$\simeq y_r(t) \cdot \left(1 - \frac{\Delta t}{t_{0r}}\right) + \frac{\Delta t}{t_{0r}}$$

Discharge / Decay:



$$y_d(t) = e^{-\frac{t}{t_{0d}}}$$

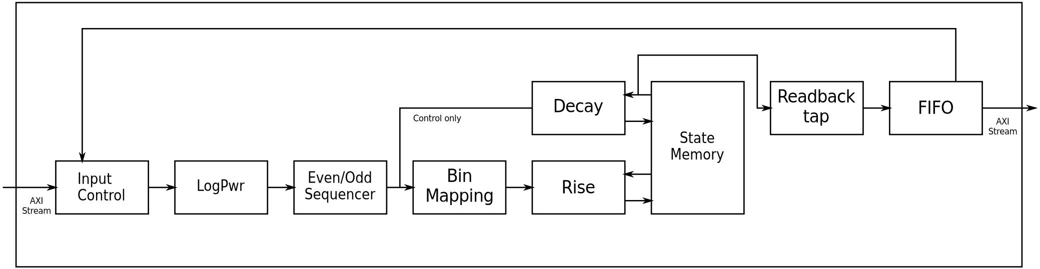
$$\frac{d}{dt} y_d(t) = -\frac{1}{t_{0d}} \cdot e^{-\frac{t}{t_{0d}}}$$

$$= -\frac{1}{t_{0d}} \cdot y_d(t)$$

$$y_d(t + \Delta t) \simeq y_d(t) + \Delta t \cdot \frac{d}{dt} y_d(t)$$

$$\simeq y_d(t) \cdot \left(1 - \frac{\Delta t}{t_{0d}}\right)$$

RFNoC fosphor: Architecture



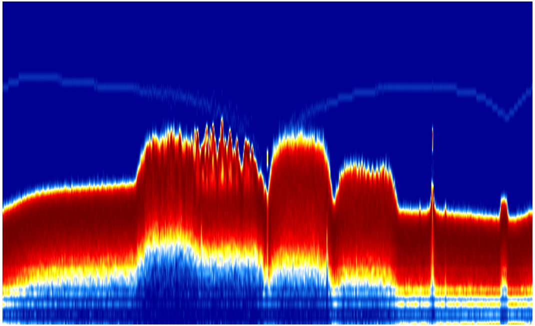
RFNoC fosphor: Numerical effects

- In `gr-fosphor`:
  - All floating point math
  - Decent amount of local me
  - "Unlimited" global memory with high bandwidth
- On a FPGA:
  - 16 bits complex fixed point
  - FFT output also quantized on 16 bits
    - `log` operation makes it more apparent at very low amplitude
    - Some power bins are 'skipped', yielding visible banding
  - Limited memory
    - Need to quantize internal state
    - Impacts accuracy of exponential rise/decay
- Use randomness to hide quantization effect
  - Use uniform RNG to avoid bias
  - For complex FFT output, do math on 18 bits with 2 random LSBs
  - For internal state, store 9 bits but do math on 9+N bits with N random LSBs

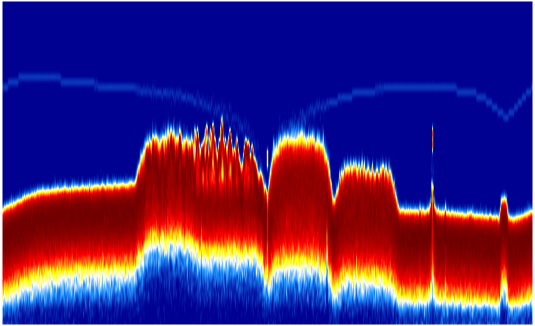
RFNoC fosphor: Numerical effects

Log(Power(x))

16 bits I/Q direct from FFT

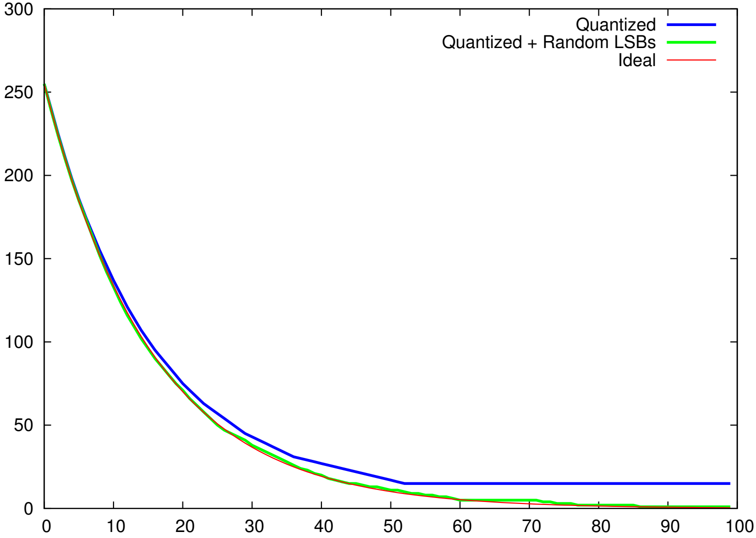


18 bits I/Q with 2 random LSBs



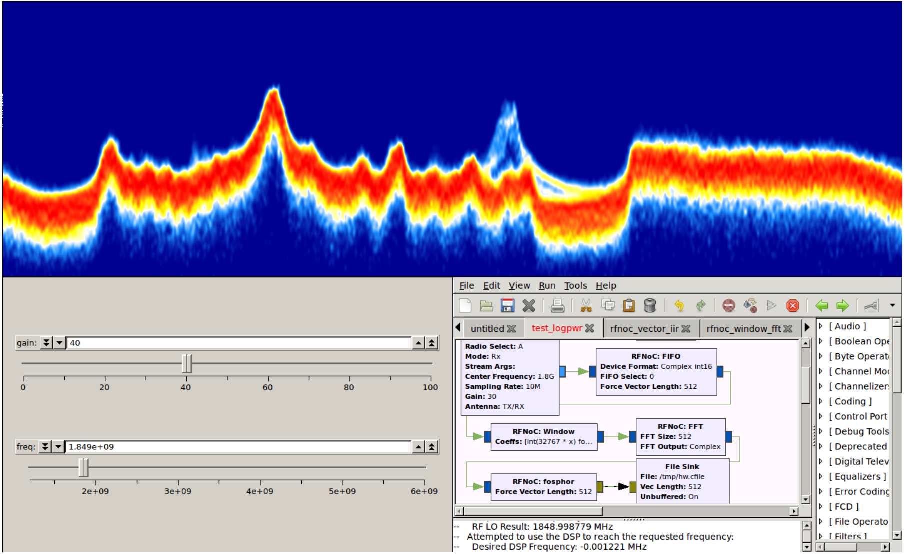
RFNoC fosphor: Numerical effects

Exponential Rise/Decay with quantized state



RFNoC fosphor: Current state

First live output



RFNoC fosphor: Current state

- Very early
  - First image out barely 4 days ago
- No decimation yet
  - Sends way too much info to the host, useful for debug
- Lots of hardcoded constants
  - 1024 FFT bins, 64 power bins
  - Need to be made parametric, both build-time and runtime
- Resources
  - 250 slices core, 1000 slices RFNoC block (X310: 63k, E310: 13k)
  - 16 RAMB36 (X310: 795, E310: 140)
  - 5 DSP48 (X310: 1540, E310: 220)

RFNoC fosphor: Future

- Proper GNURadio block
- FPGA:
  - Proper
  - Cleanup and Integration
  - Runtime configurability
  - Multi-channel support
  - Higher internal clocking
- Alternative uses:
  - Should be usable for eye-pattern easily
    - Remove logpower
    - Feed pre-synched/triggered time data
  - Not quite so easy for constellation plots

Thanks

Thanks to :

- Ettus Research for RFNoC and sponsoring this project
- GNURadio community for a great framework

Thank you for your attention !

Questions ?

Any questions ?

Resources

- **fosphor**
  https://sdr.osmocom.org/trac/wiki/fosphor
- **GRcon 13 fosphor**
  http://gnuradio.squarespace.com/storage/grcon13_presentations/grcon13_munaut_fosphor.pdf
- **GRcon 14 RFNoC**
  http://gnuradio.squarespace.com/storage/grcon14/presentations/Sep17_04_Ettus_rfnoc.pdf
- **RFNoC**
  https://github.com/EttusResearch/uhd/wiki/RFNoC:-Getting-Started