Stuff you don’t see - every day

SDR and the GNU Radio Framework

Chaos Communication Camp 2011
by
Marius Ciepluch

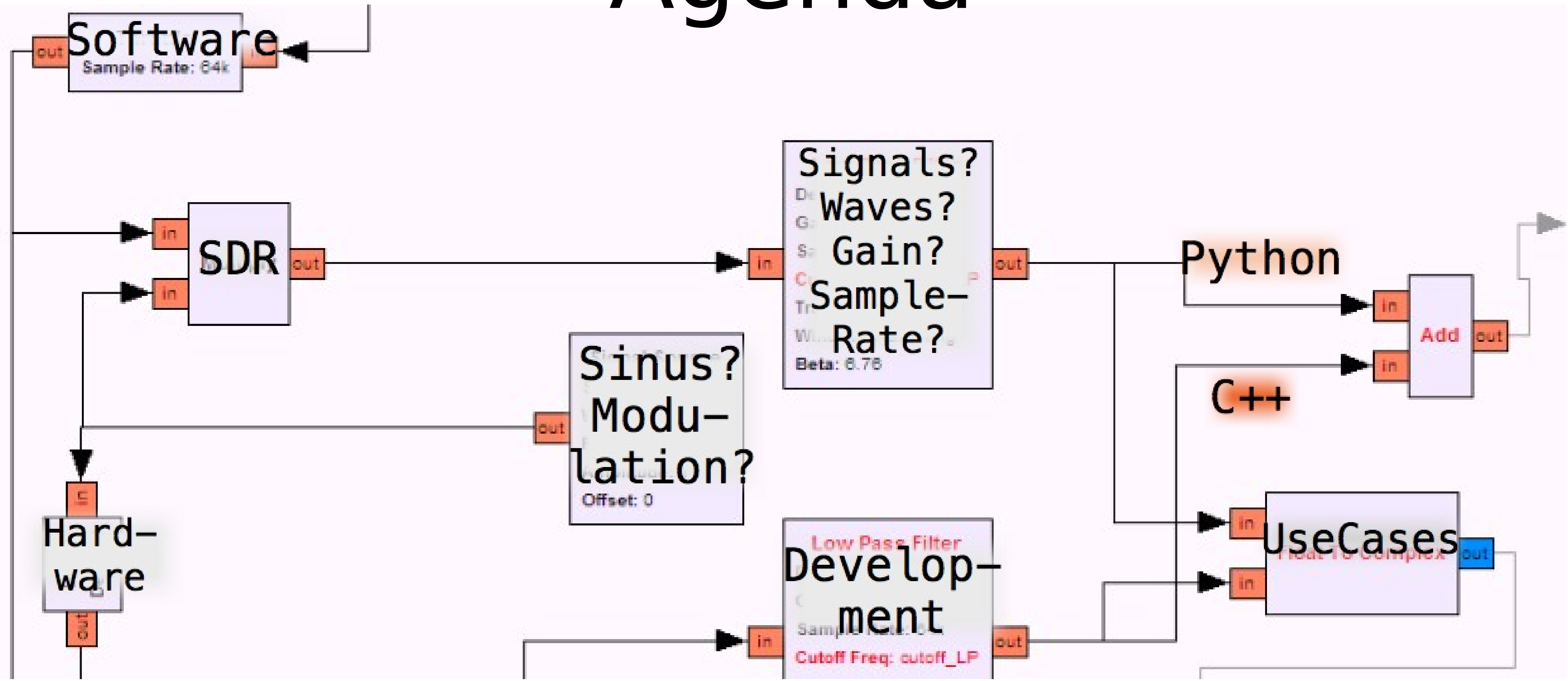
Public Speaking Protocol

```
Right hand: self.immediate_question(mesg);

Left hand: speaker.enhance_articulation();

Both hands: speaker.enhance_excitement();
```

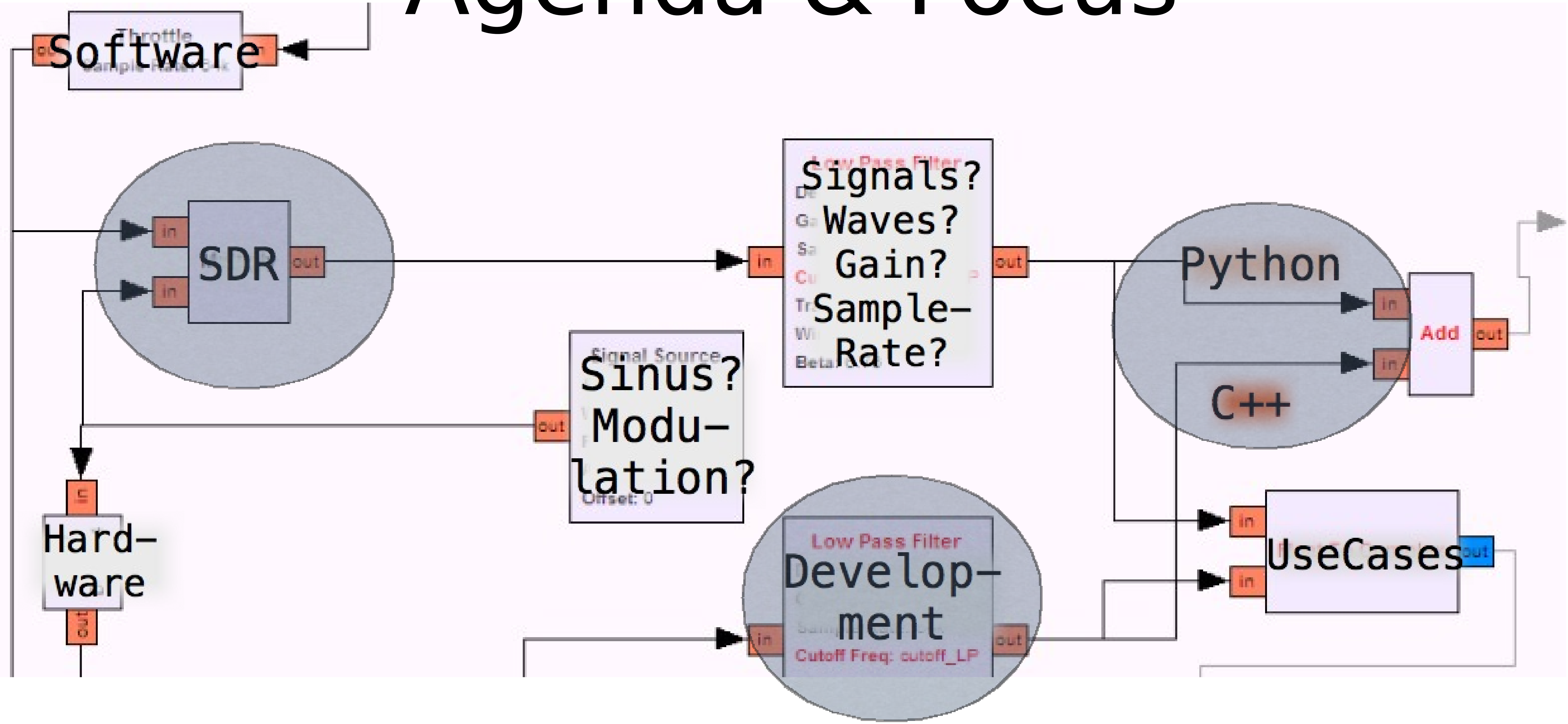
Disclaimer

- This document was prepared as a private “science” contribution to the Chaos Communication Camp 2011. Anything expressed within the documents and during the presentation is not associated with the author’s present or future clients or employees.

Agenda


Software
Sample Rate: 64k
SDR
Signals? Waves? Gain? Sample-Rate?
Beta: 6.76
Python
C++
Add
Sinus? Modu-lation?
Offset: 0
Hard-ware
Low Pass Filter
Develop-ment
Cutoff Freq: cutoff_LP
UseCases
in
out

Agenda & Focus



Who?

Marius Ciepluch (wishi)

Software Developer

Embedded Software Development

  Wireless Sensor Networks

  Industrial Automation

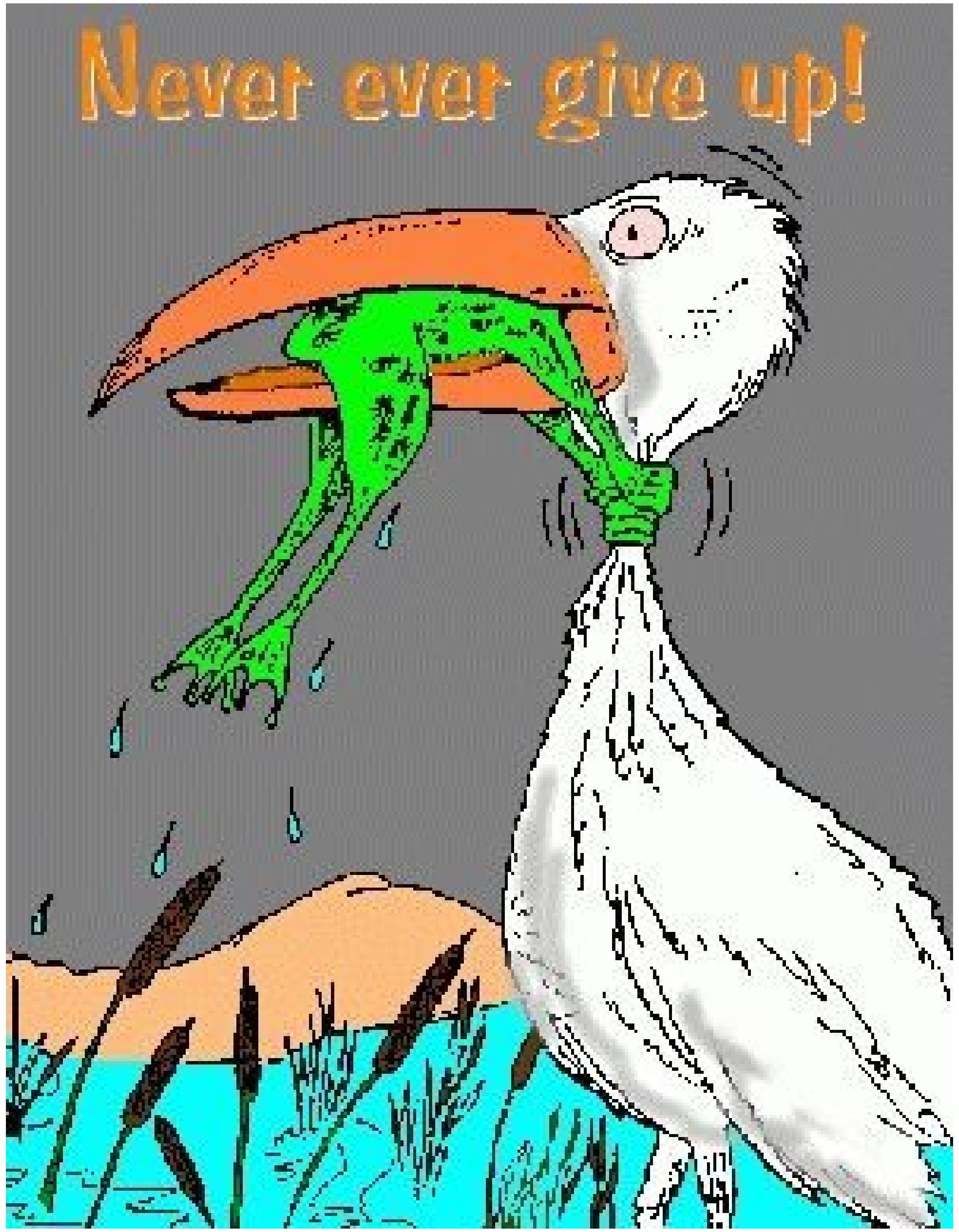
(Software Testing ||

Software Verification)

(Reverse Engineering >>

Security Research)

twitter: @wishinet, mc - at - sandokai.eu
web: http://crazylazy.info/blog

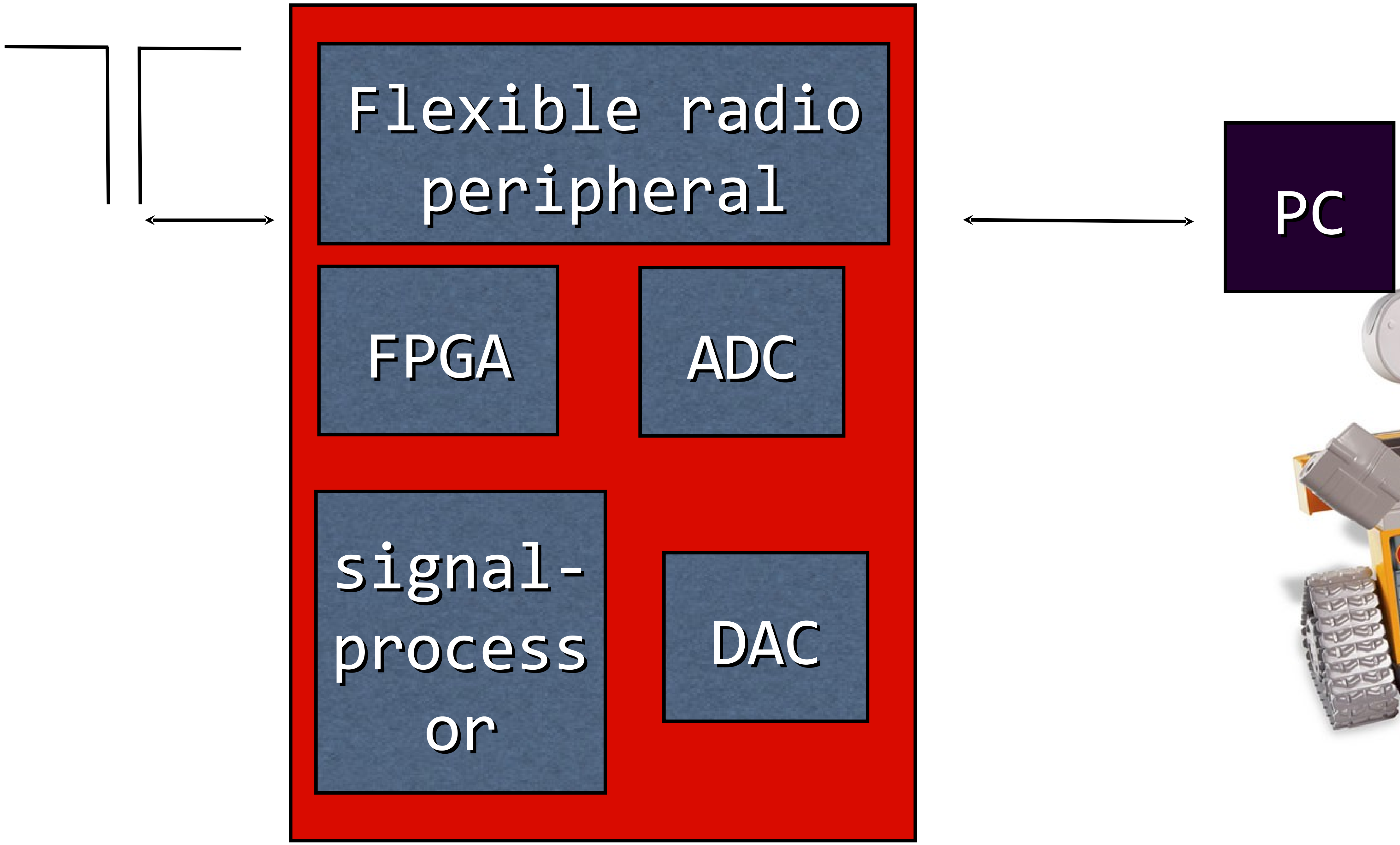


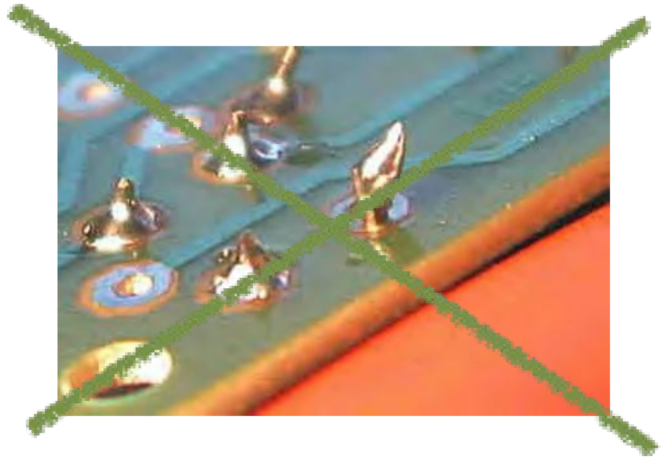
Motivation

How many wireless devices are here?
How many different wireless standards are here?

omni-presence, visible math,
open approaches,
useful skills, practical approaches

Software Defined Radio





Origins of Software Defined Radio

- Expected to be the dominant technology in radio communications (Wireless Innovation Forum)
- Efforts since 1984, miliary/classified research
- Getting more affordable lately for amateurs
  - Some models use sound-cards as ADCs. Other approaches are with ADCs and FPGAs

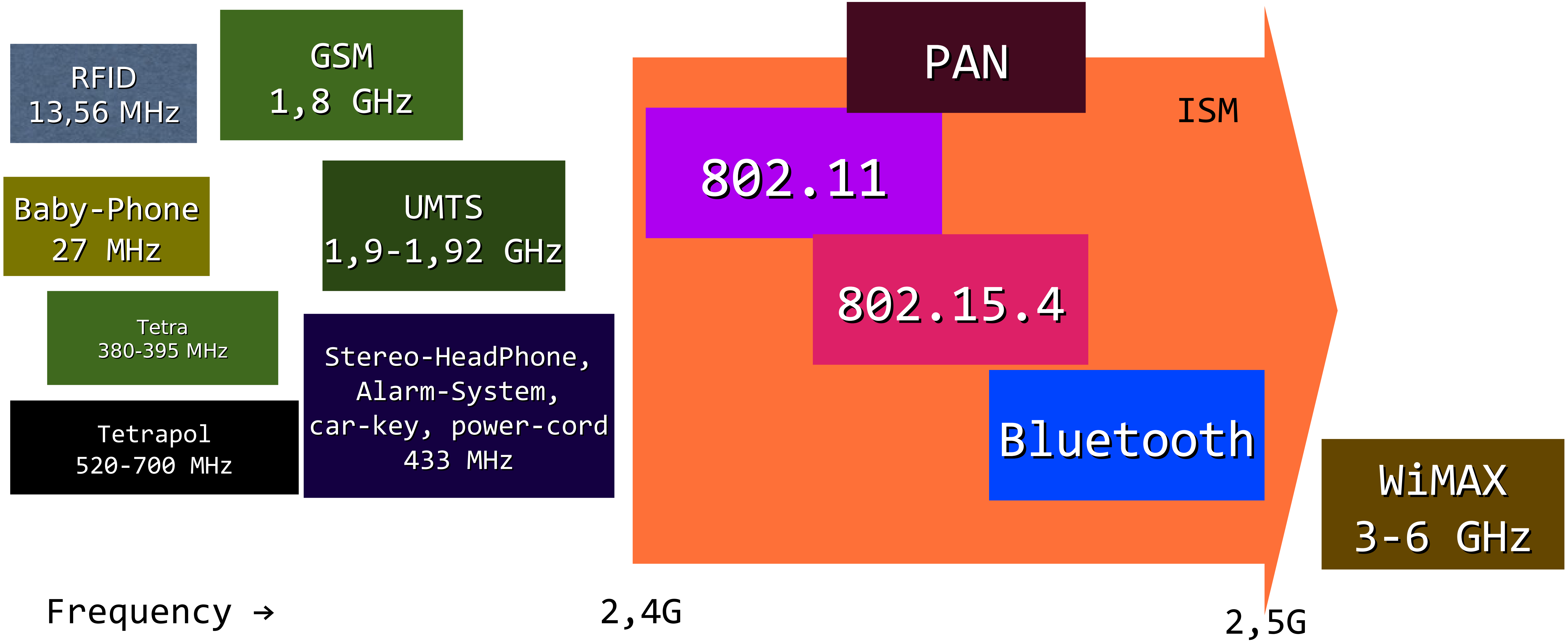
UseCases for SDR (with GNU Radio)

- 802.15.4, Bluetooth, DECT, GPS, GSM, Tetra, 802.11b, RFID, custom protocols/requirements (e.g. real-time) - commercial wireless
- Spectrum Sensing and Interference Studies - reliable communication design
- Active/Passive Radar, Sonar, SIGINT/COMINT, Field Smart Radios, Satellite Ground Stations - Security Research and Public Safety
- Multi-Mbps GMSK, PSK, OFDM, MIMO networking - development
- Radio Astronomy, Telemetry, Medical Imaging, Wild Life Tracking, Structural Analysis - science and engineering

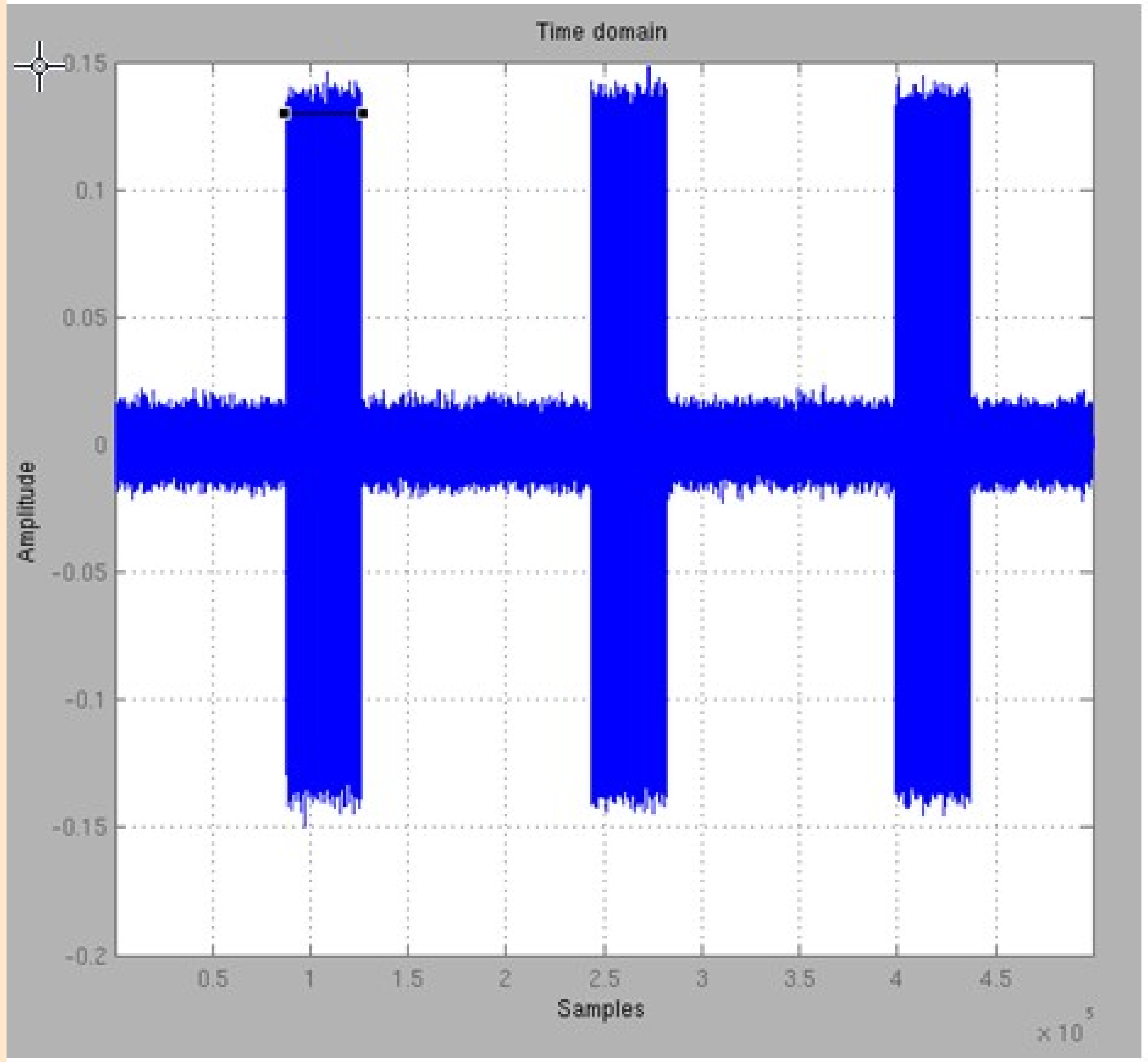
What is GNU Radio

- Software framework (GPL)
  - Develop transmission schemes
  - Many algorithms included
- Interesting architecture (C++ ↔ Python ↔ XML)
- Abstracts HW interaction with peripheral

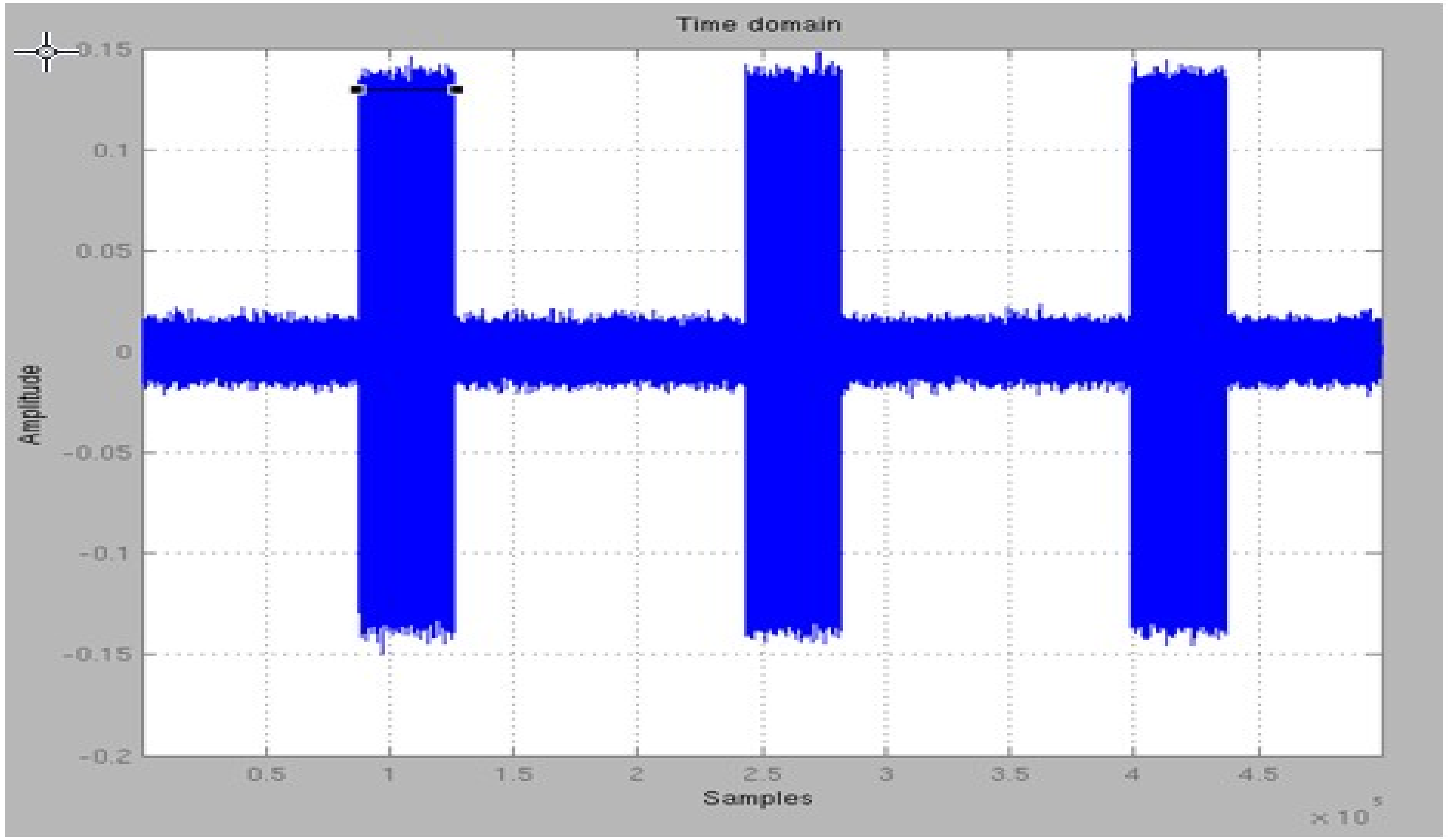
Wireless Germany – Stuff you don't see every day



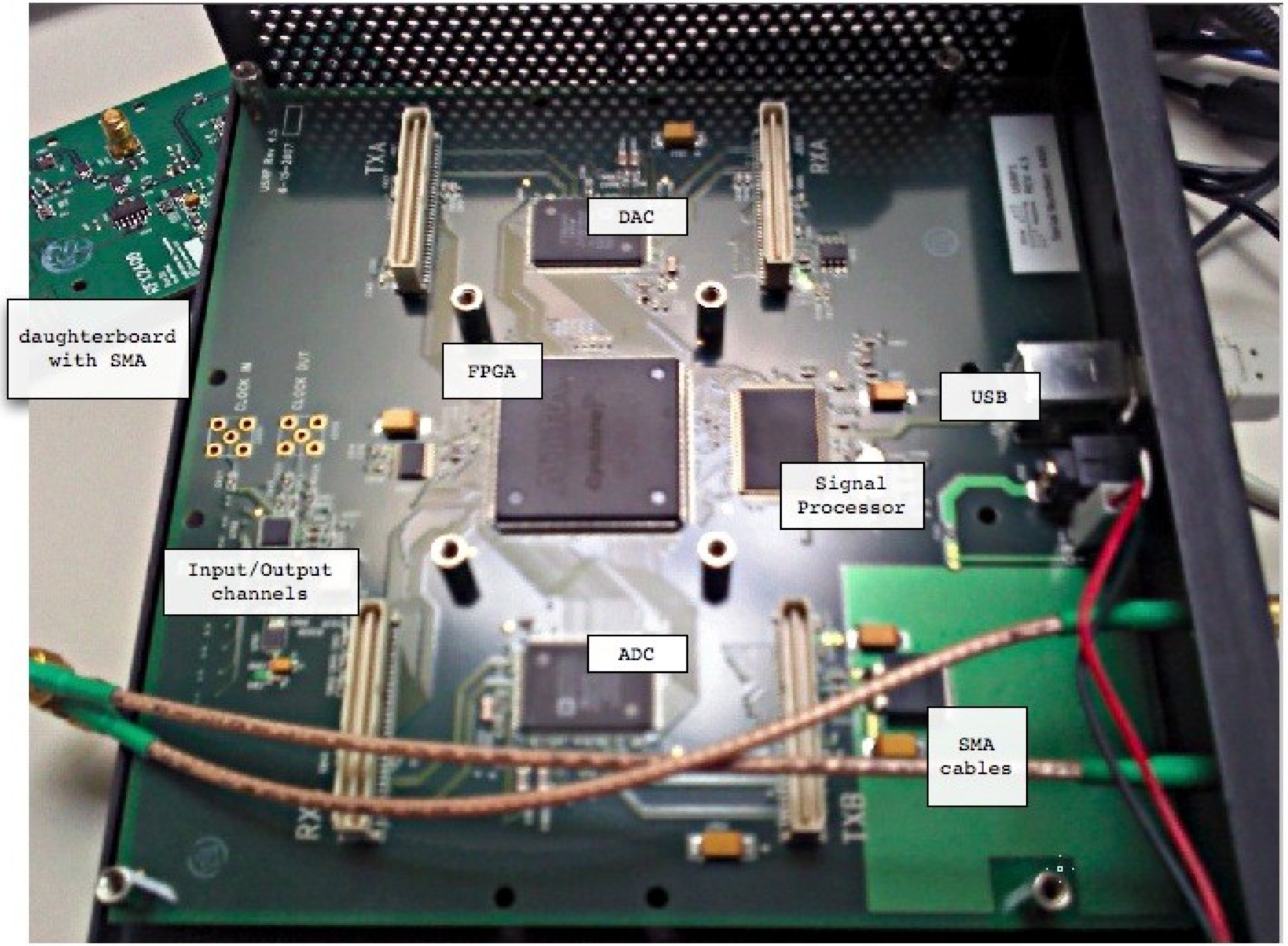
- sample-rate: samples/second - a real-time system
- gain: constant for PGA
- modulation: -> extra section
- wave: the “moving” curve
- pulse: positive short amplitude
- carrier: “mother wave”
- IF band: inner frequency band, RF band: received at Daughter Board (down-scales to ADC frequency)
- ADC/DAC: Analog Digital Converter / Digital Analog Converter

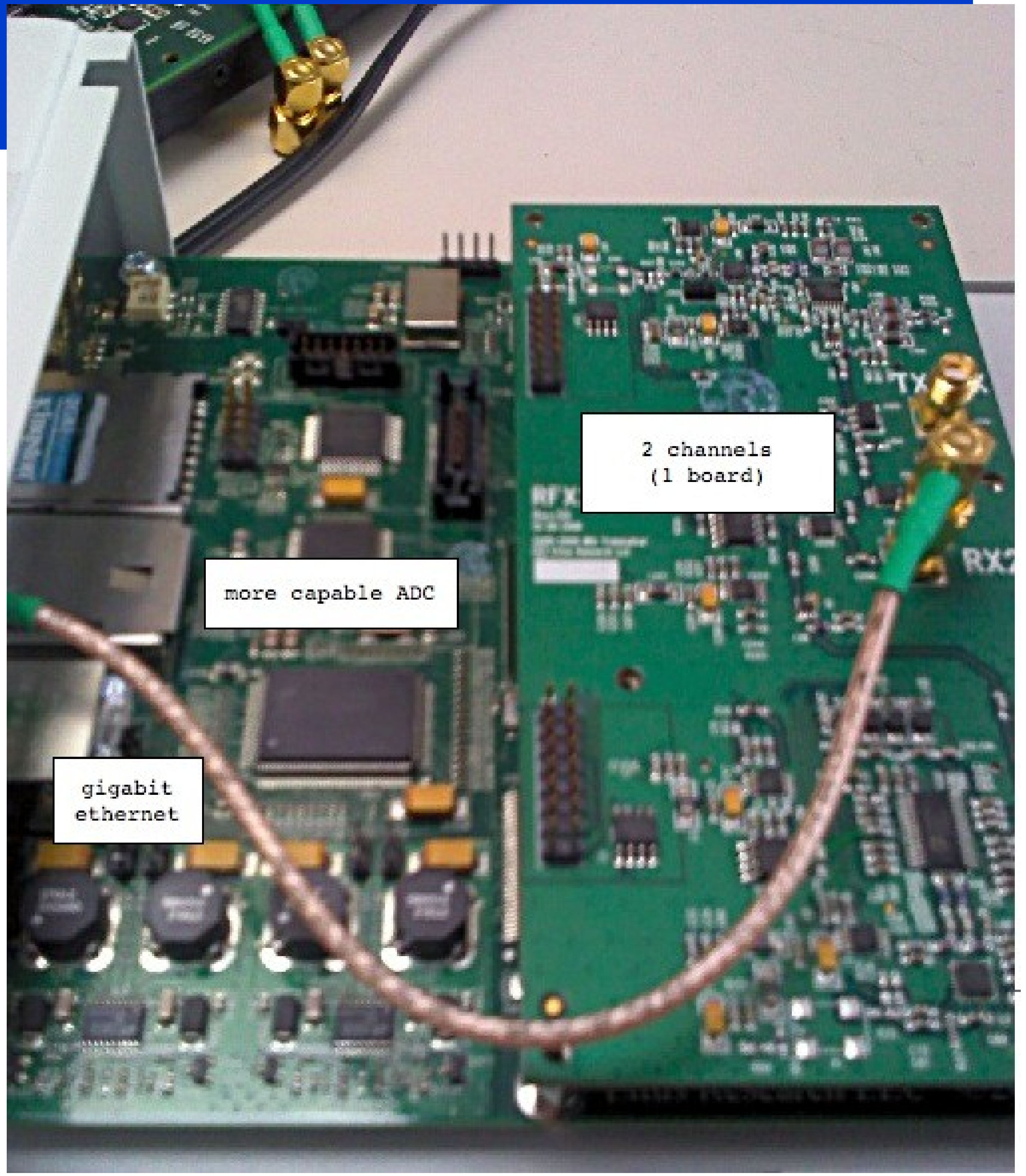


Demo: psk audio, 33KHz, 8bps

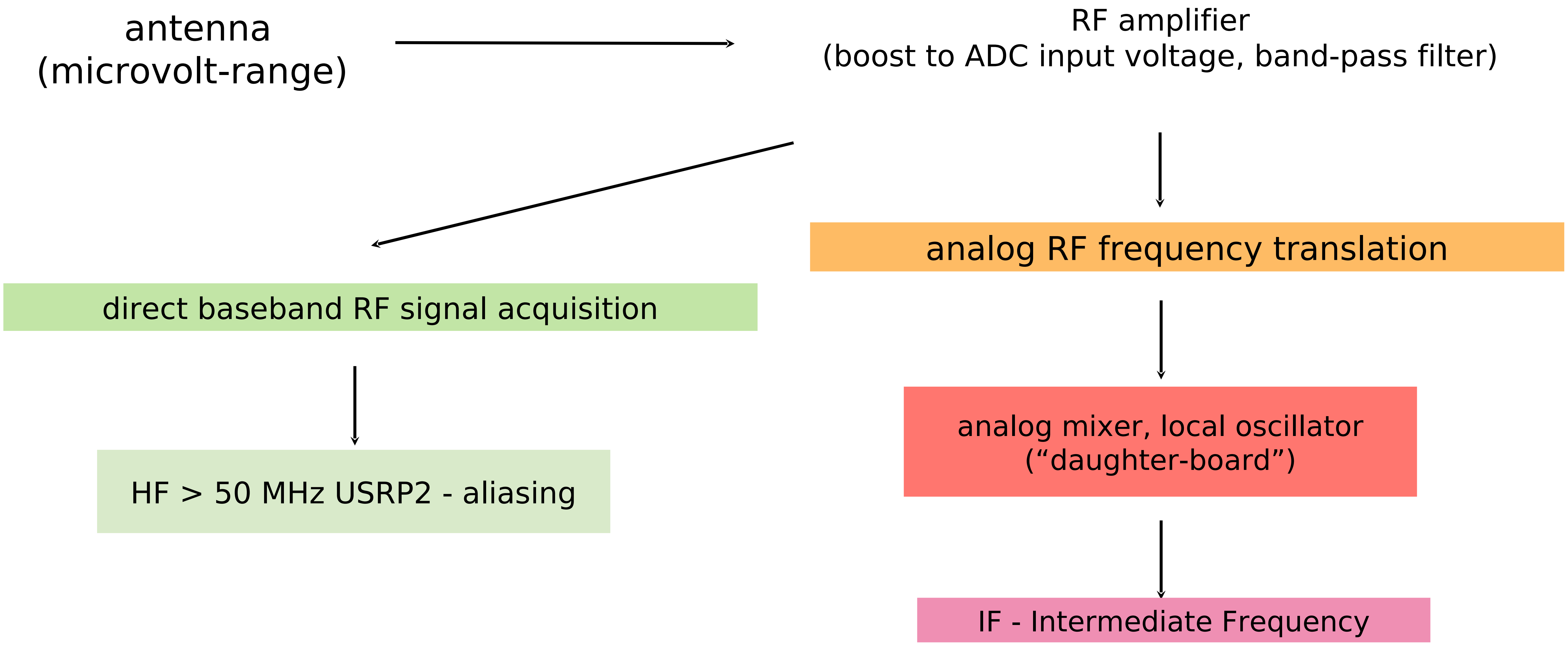


Hardware: USRPs





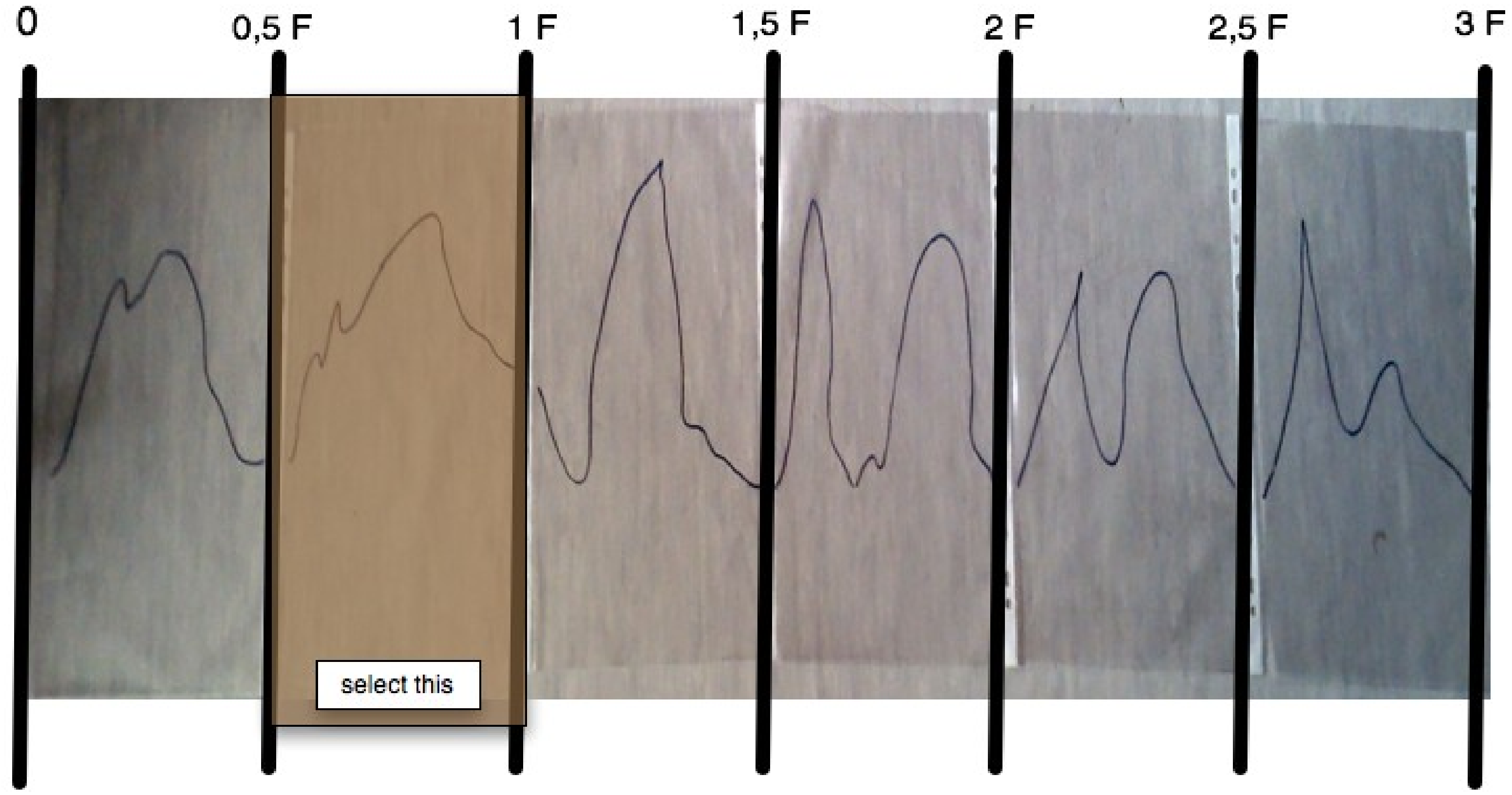
ADCs at the radio peripheral



At the receiver system: filtering

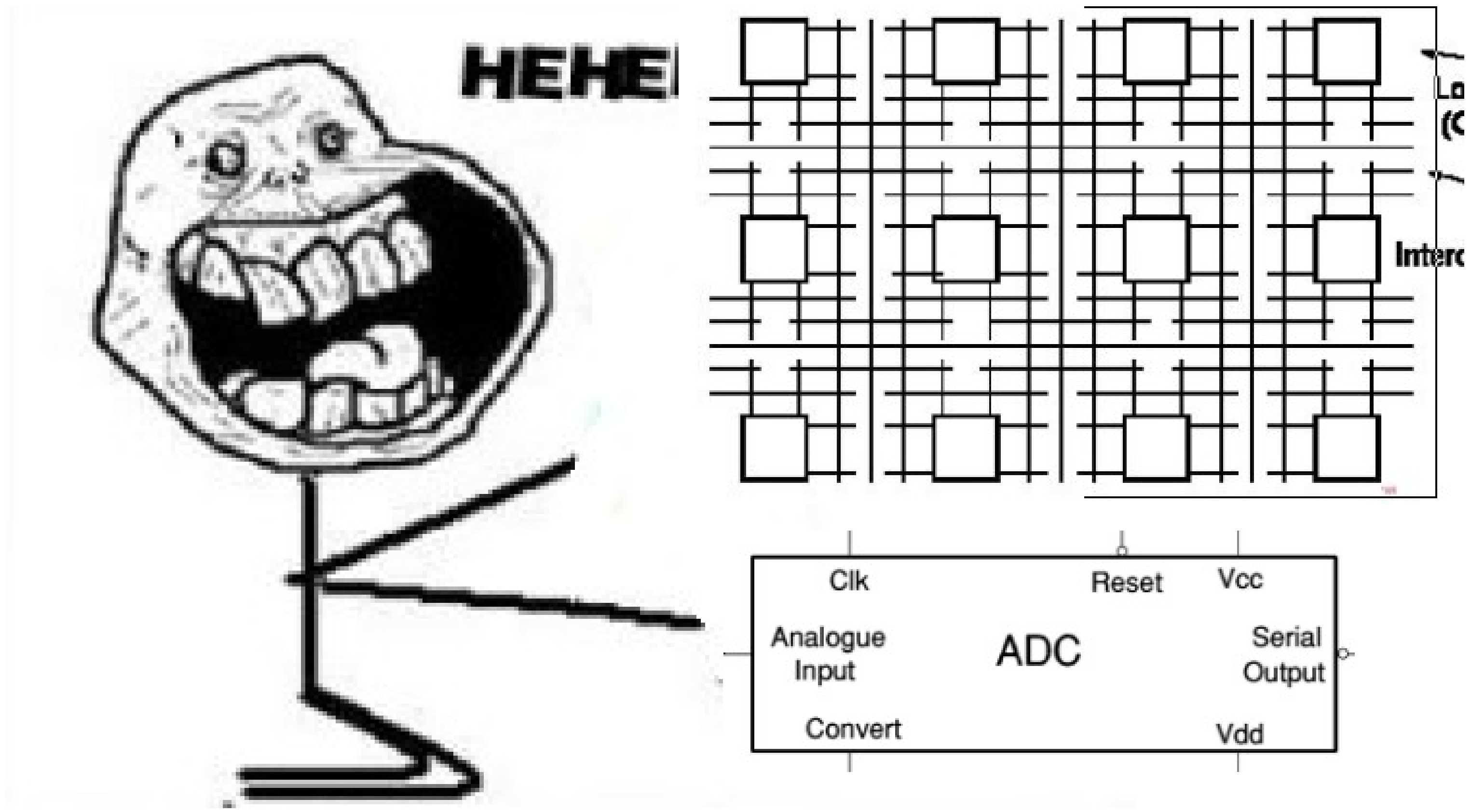
Low Pass Filter: wideband sampling

Band Pass Filter: baseband sampling



improve signal-to-noise ratio,
save dynamic ADC range

FPGAs at the radio peripheral



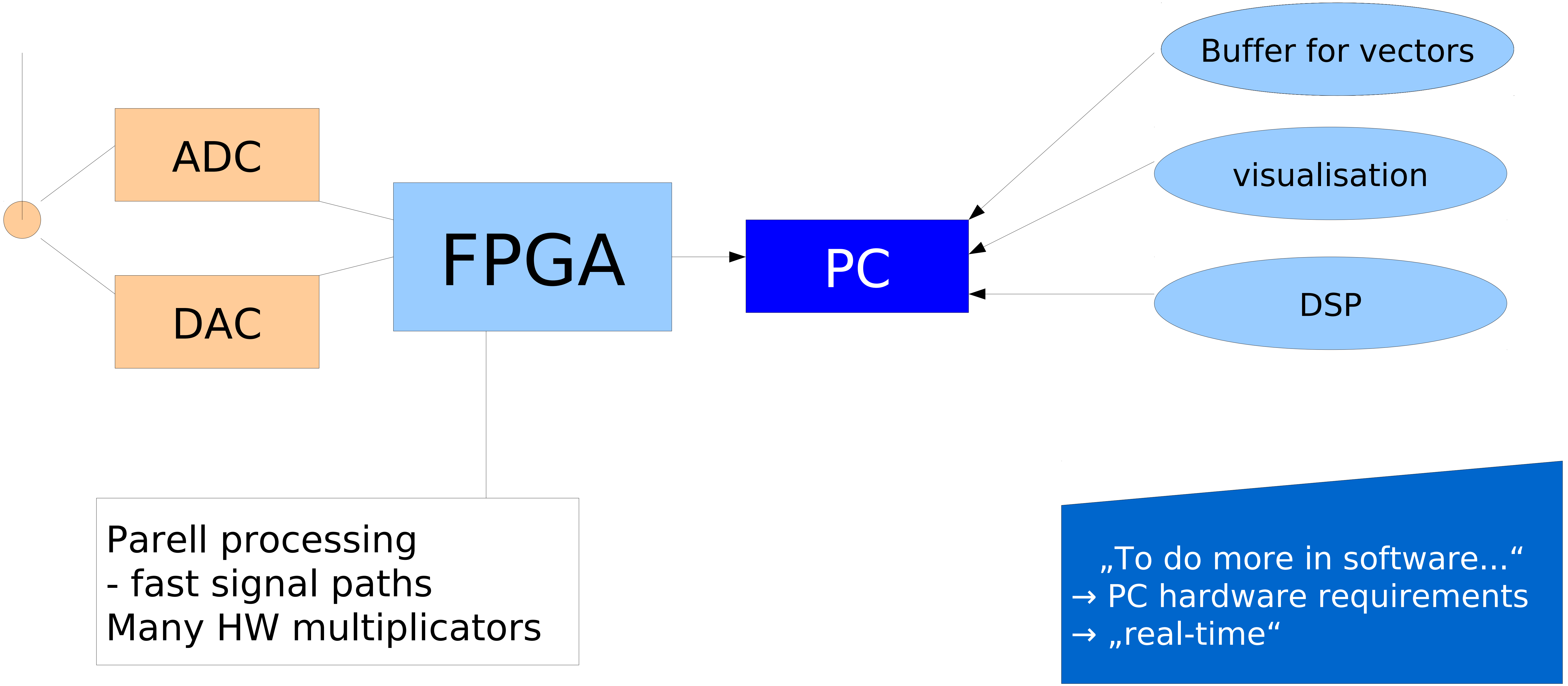
FPGA + ADCs = ♥

real-time DSP (vendor DSP initiatives) - dedicated HW multipliers

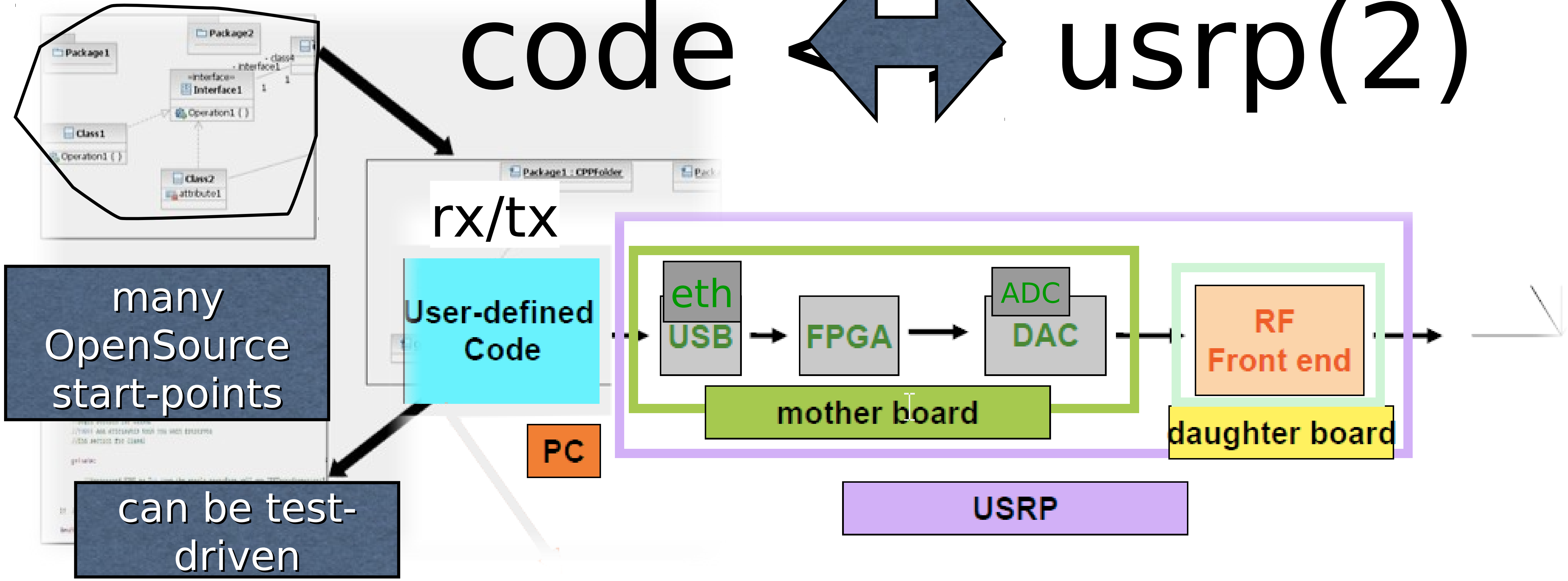
I/O pins: gigabit serial transceivers (BGA and flip-chip packages)

keep power and heat down (low voltage compared to CPU/GPU)

FPGA conclusions

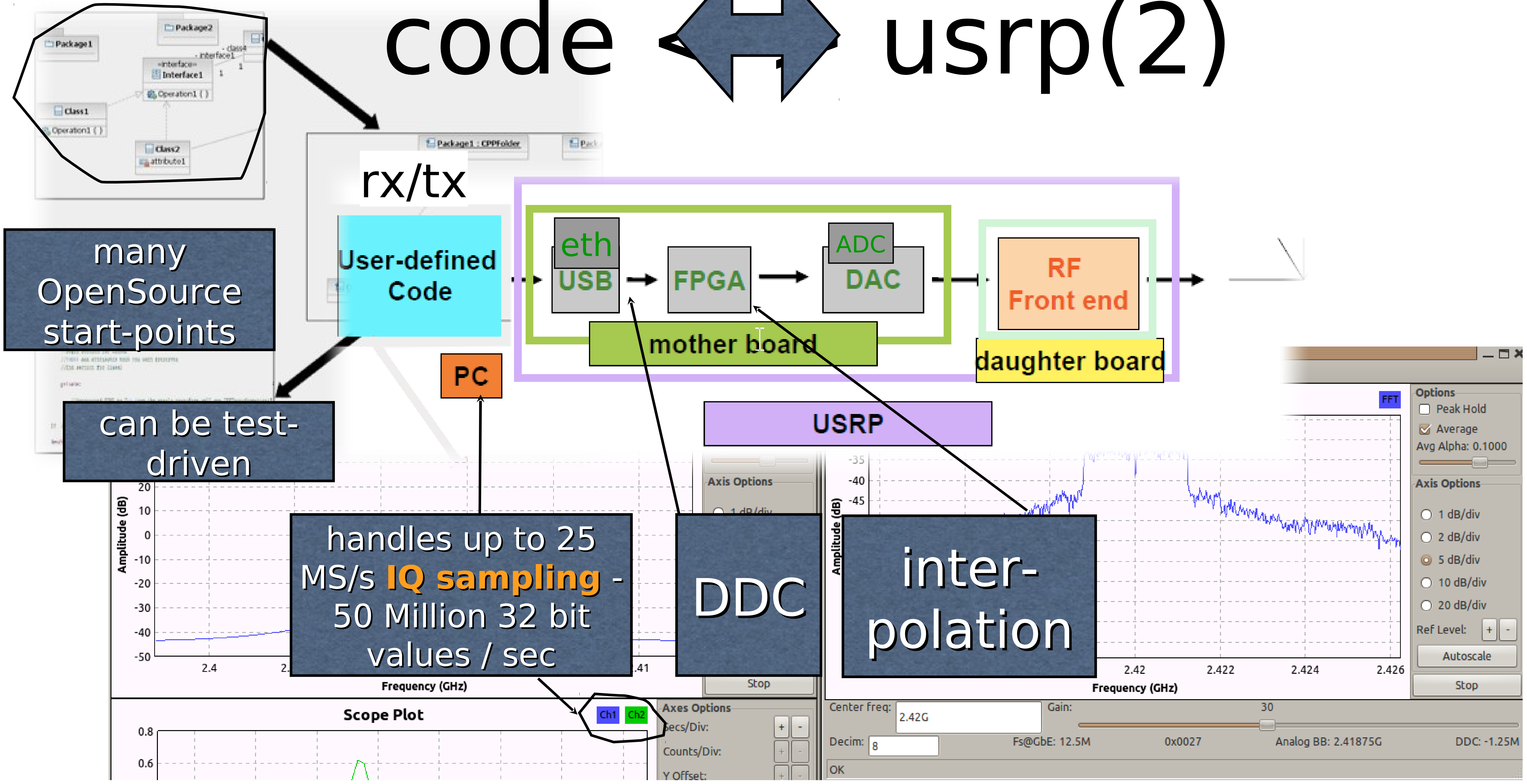


code ⟷ usrp(2)

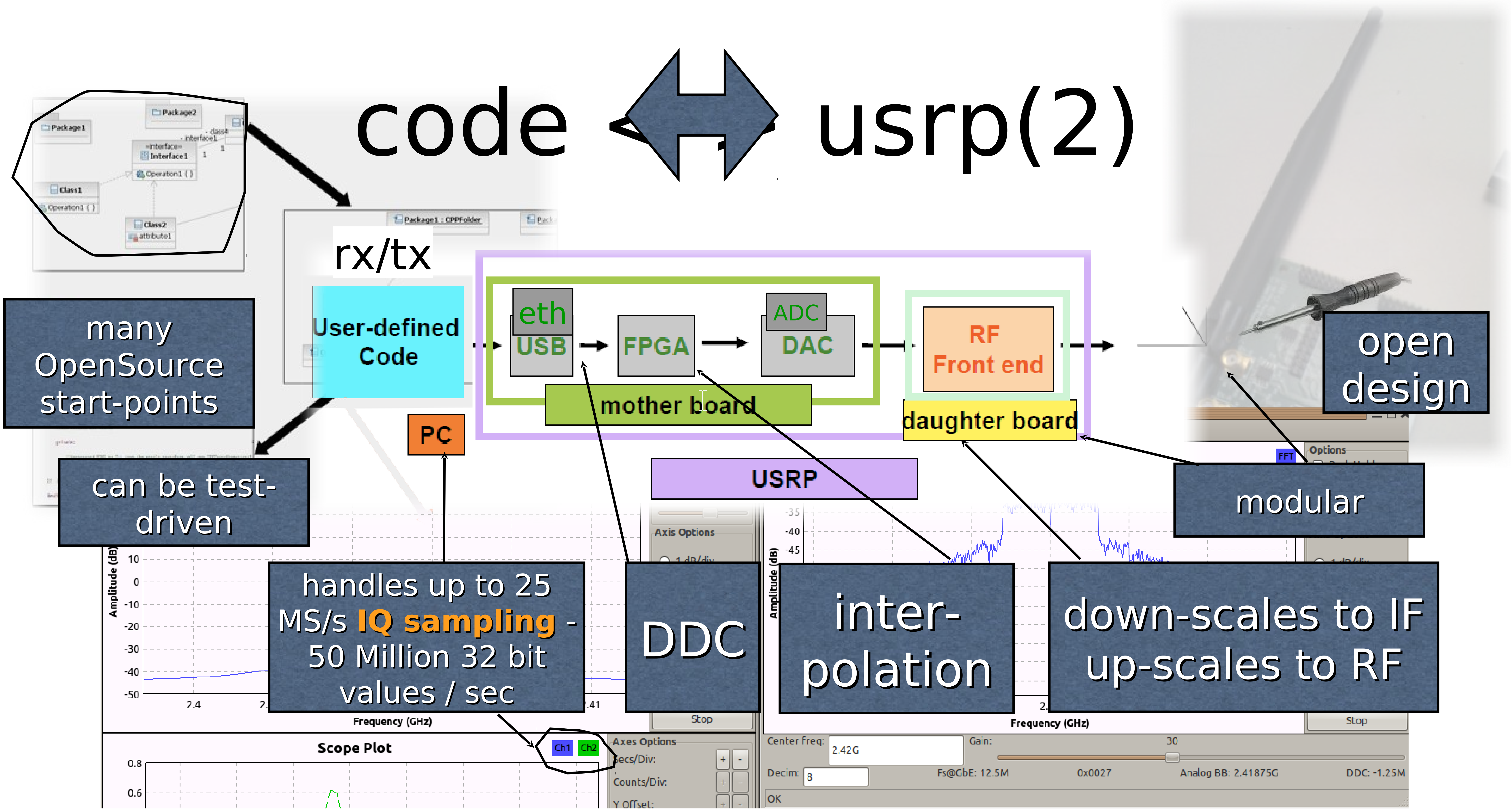


many OpenSource start-points
can be test-driven
rx/tx
User-defined Code
PC
eth
USB
FPGA
ADC
DAC
mother board
RF Front end
daughter board
USRP

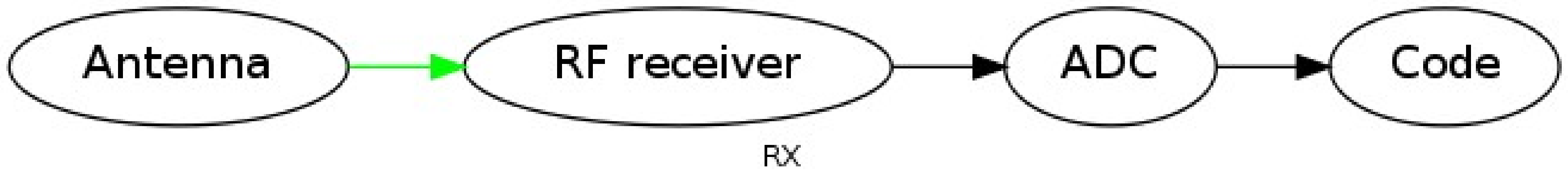
code ⟷ usrp(2)



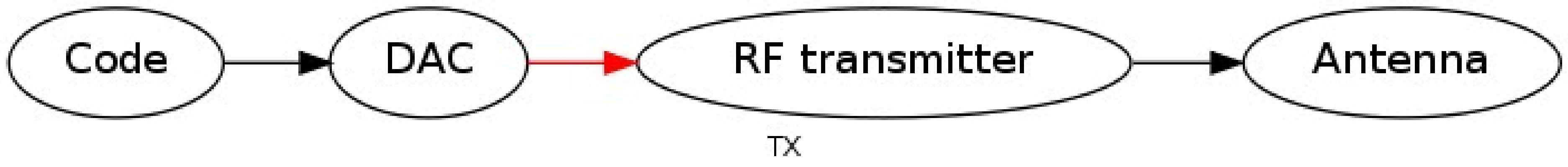
rx/tx
User-defined Code
eth
USB
FPGA
ADC
DAC
RF Front end
mother board
daughter board
PC
USRP
many OpenSource start-points
can be test-driven
handles up to 25 MS/s IQ sampling - 50 Million 32 bit values / sec
DDC
inter-polation
Package1
Package2
Interface1
Operation1 ( )
Class1
Class2
attribute1
Package1 : CPPFolder
Amplitude (dB)
Frequency (GHz)
Scope Plot
Ch1
Ch2
Axis Options
Axes Options
Secs/Div:
Counts/Div:
Y Offset:
Stop
FFT
Options
Peak Hold
Average
Avg Alpha: 0.1000
1 dB/div
2 dB/div
5 dB/div
10 dB/div
20 dB/div
Ref Level:
Autoscale
Center freq:
2.42G
Gain:
30
Decim:
8
Fs@GbE: 12.5M
0x0027
Analog BB: 2.41875G
DDC: -1.25M
OK

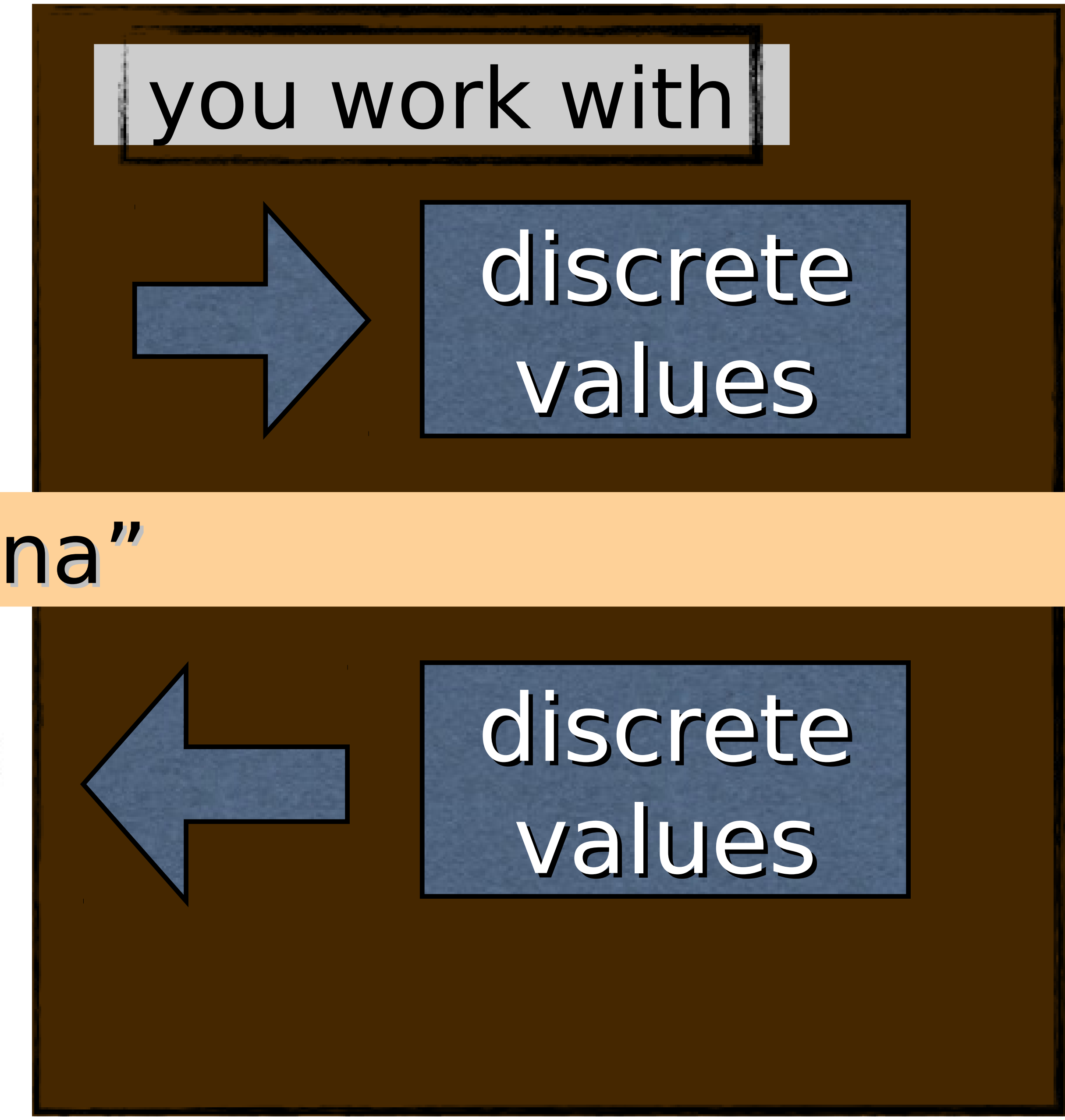

code ⟷ usrp(2)
rx/tx
User-defined Code
eth
USB
FPGA
ADC
DAC
RF Front end
mother board
daughter board
PC
USRP
many OpenSource start-points
can be test-driven
open design
modular
handles up to 25 MS/s IQ sampling - 50 Million 32 bit values / sec
DDC
inter-polation
down-scales to IF up-scales to RF
Package1
Package2
Interface1
Operation1 ( )
Class1
Class2
attribute1
Package1 : CPPFolder
Amplitude (dB)
Frequency (GHz)
Scope Plot
Ch1
Ch2
Axis Options
Axes Options
Secs/Div:
Counts/Div:
Y Offset:
Options
FFT
Stop
Center freq:
2.42G
Gain:
30
Decim:
8
Fs@GbE: 12.5M
0x0027
Analog BB: 2.41875G
DDC: -1.25M
OK

RX and TX Paths on USRPs



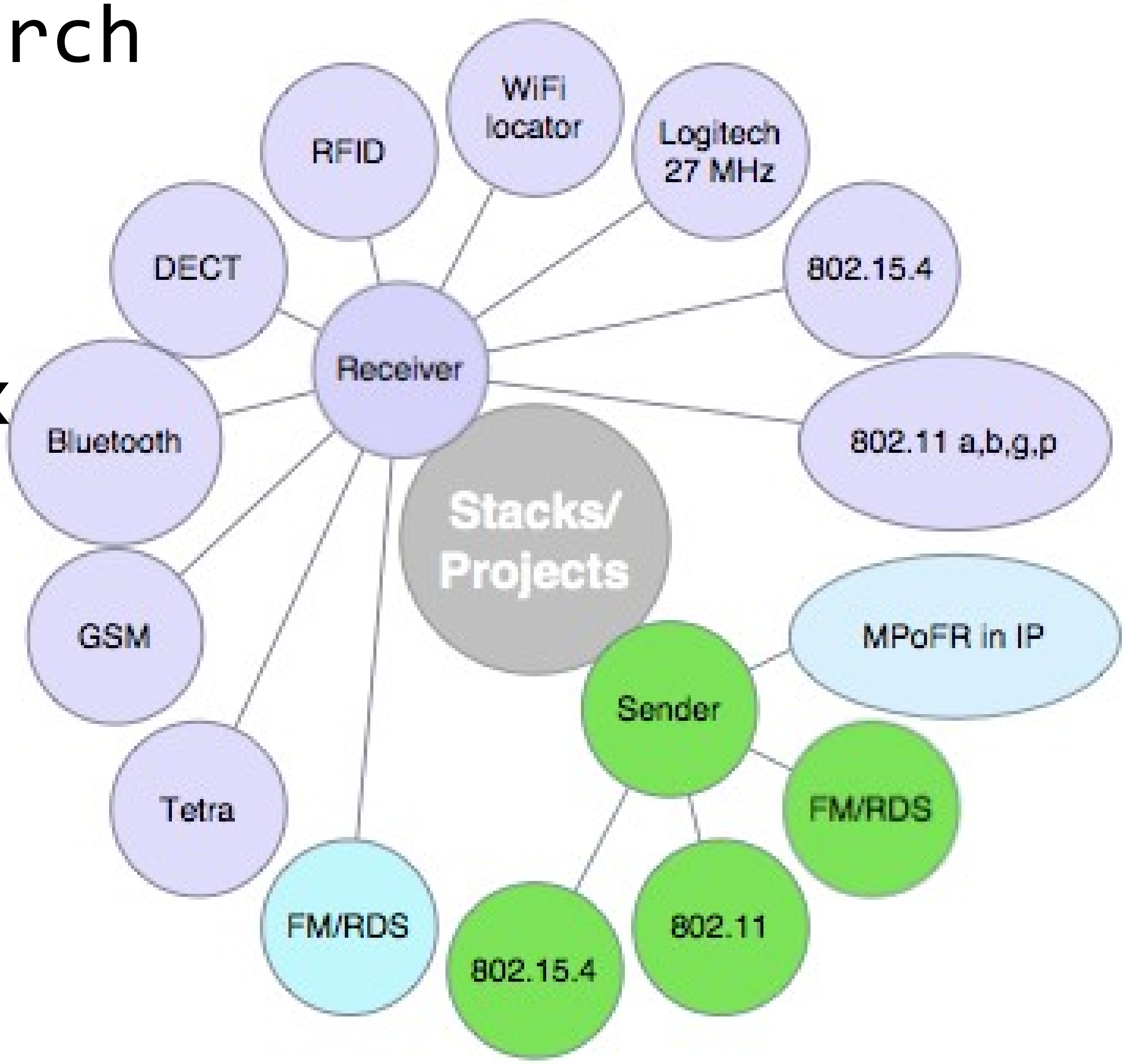
“code near to the antenna”



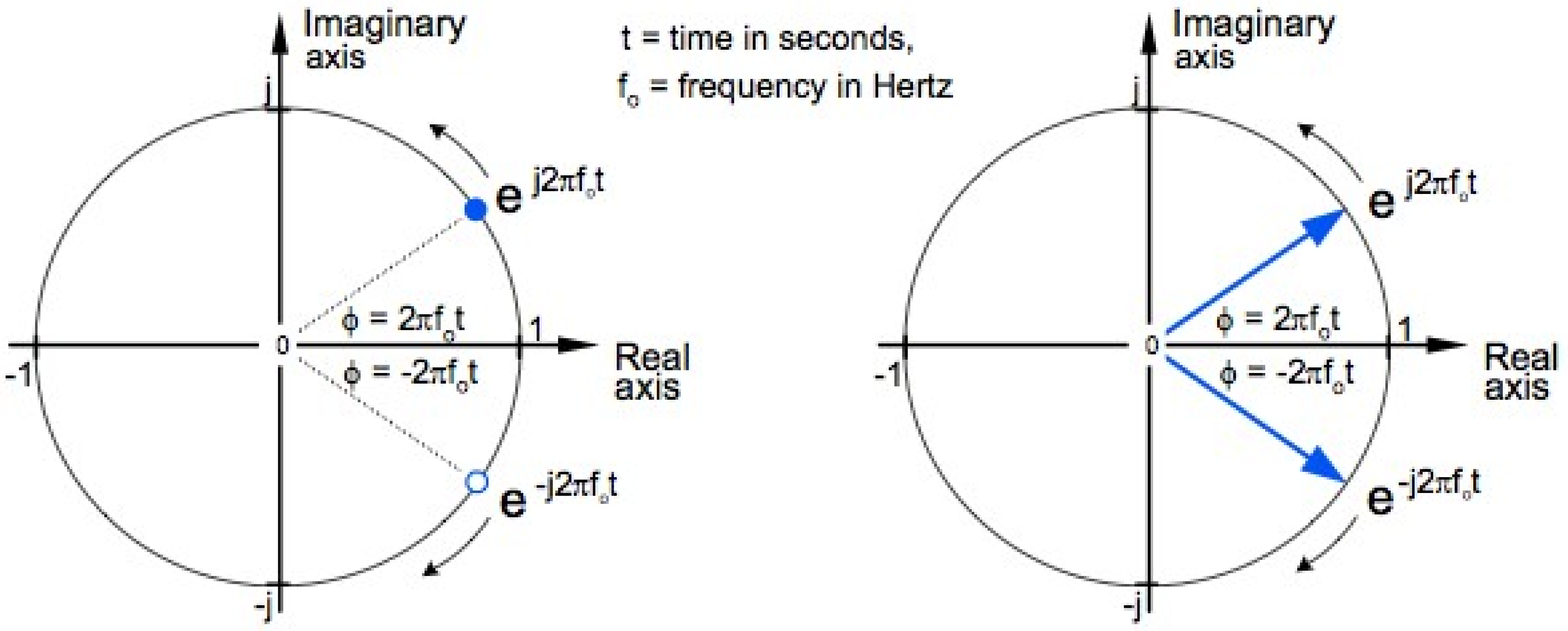


publicly available GR stacks

- communication technology research
  - custom protocol analysis
  - not every stack is full rx/tx
    - algorithms not in GRC
- so why is this possible?
  - one way to rule them all?

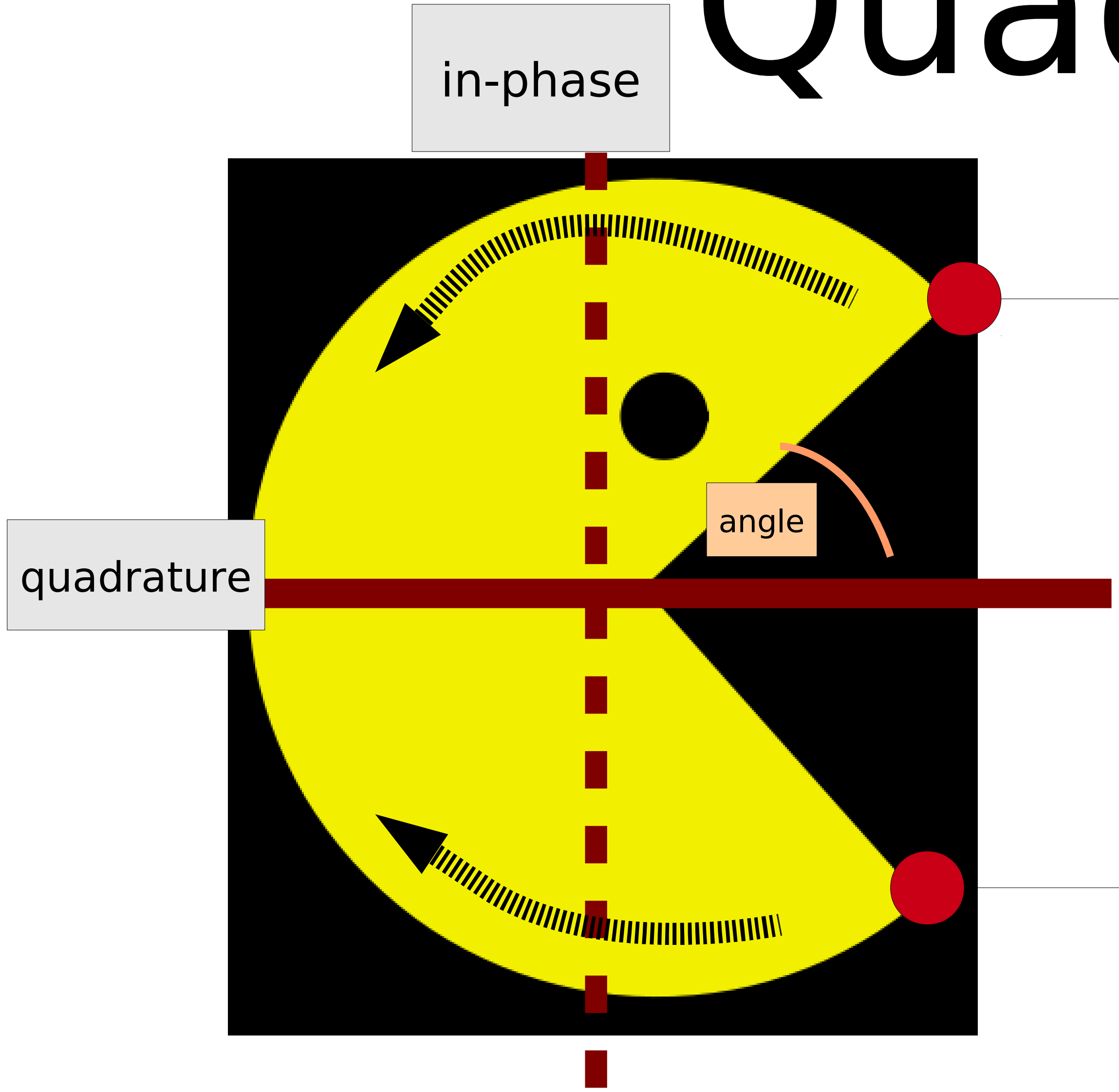


Quadrature Signals



See last slide for image references

Quadrature signals



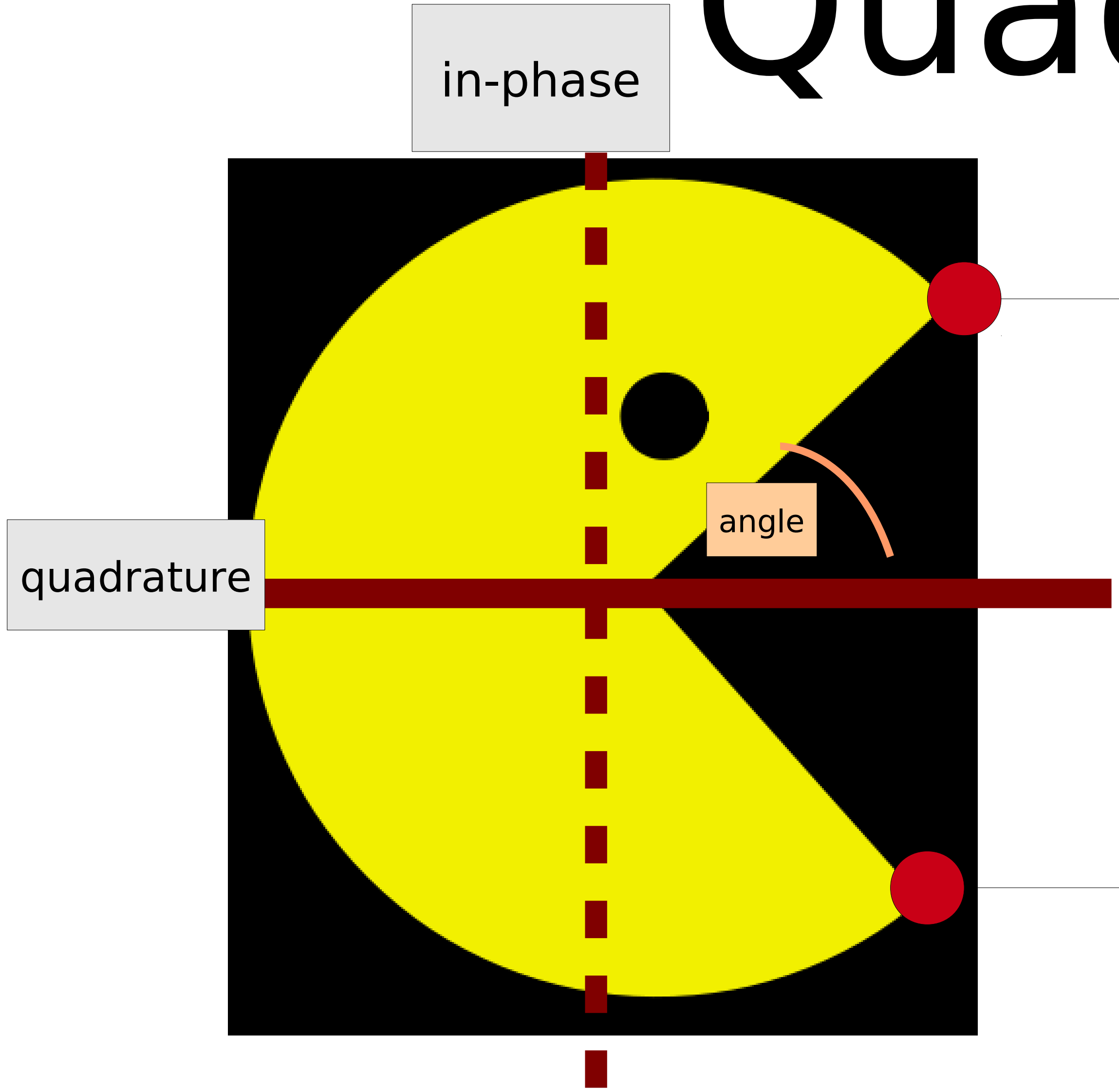
e^(in-phase * 2 Pi)*freq. * time

e^(-in-phase * 2 Pi)*freq. * time

Pacman perimeter:
2 Pi * r,
r=1

- Practically in every communication system
- angle=2Pi * freq. * time

Quadrature signals



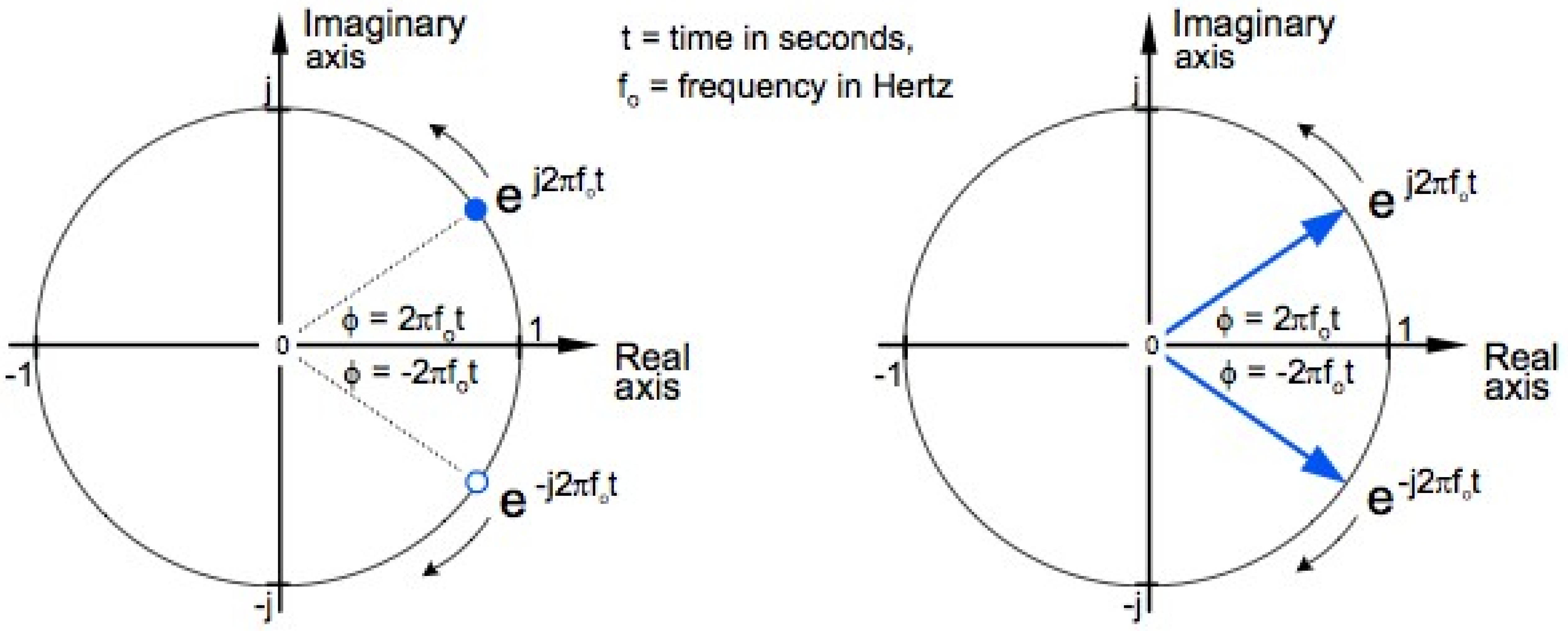
e^(in-phase * 2 Pi)*freq. * time

e^(-in-phase * 2 Pi)*freq. * time

Pacman perimeter:
2 Pi * r,
r=1

- This Pacman is **very hungy (eats itself)**: red dot rotates with the frequency
- It symbolizes how the unity circle can be used to understand Quadrature signals

Quadrature Signals (wait)



See last slide for image references

Used in IEEE 802.15.4 (incl. Q Delay)

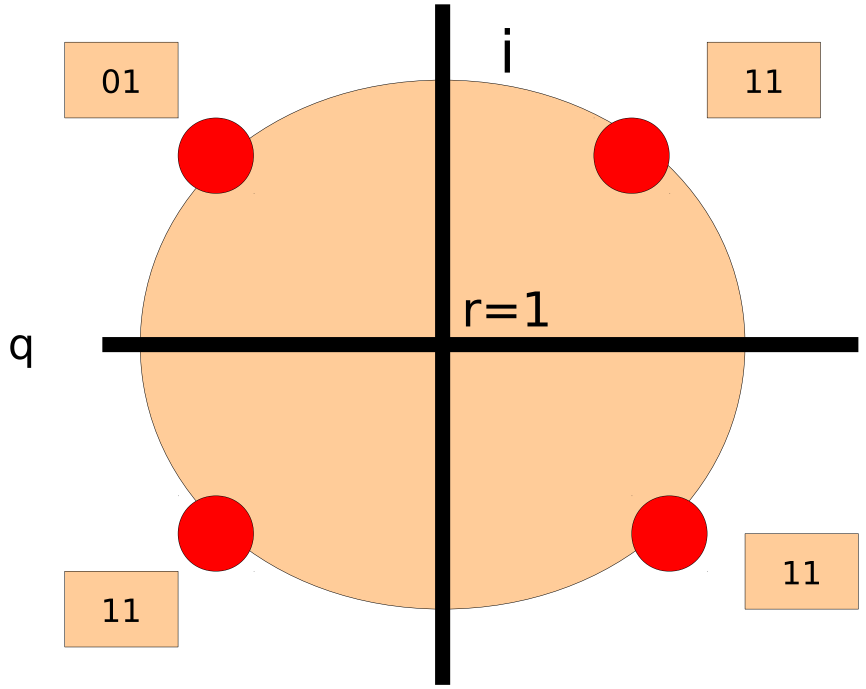
GR “Dive in”: QPSK

```
for (int i = 0; i < noutput_items / SAMPLES_PER_SYMBOL; i++){
    float iphase = real(in[i]);
    float qphase = imag(in[i]);

    *out++ = gr_complex(0.0, 0.0);
    *out++ = gr_complex(iphase * 0.70710678, qphase * 0.70710678);
    *out++ = gr_complex(iphase, qphase);
    *out++ = gr_complex(iphase * 0.70710678, qphase * 0.70710678);
  }
```

Iteration within infinite vector stream from radio peripheral – ring buffer

Quadrature

Sinus as carrier



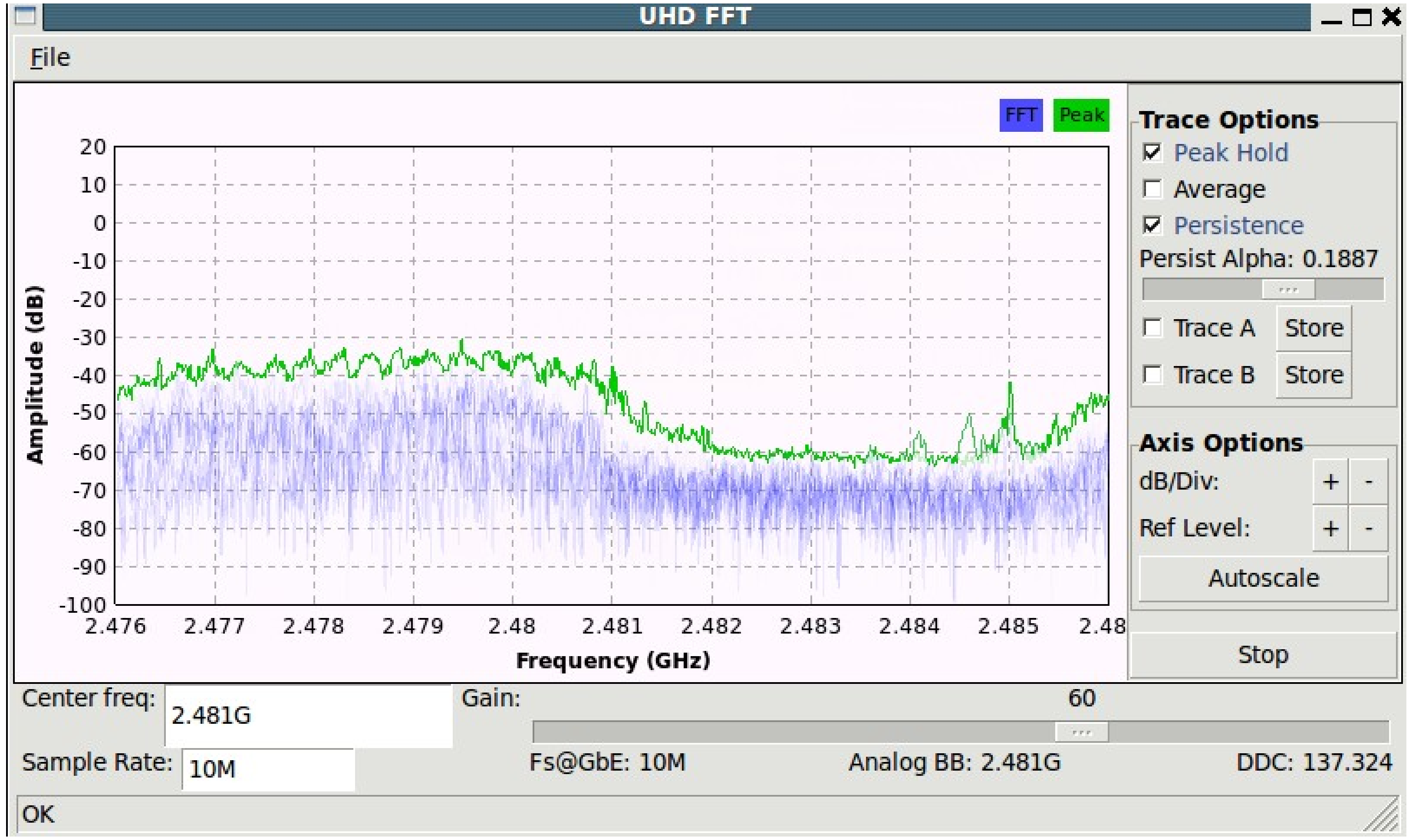
Modulates phase changes in 4 phases (red dots)
2 bits per symbol,
Phase angle changes by multiplication

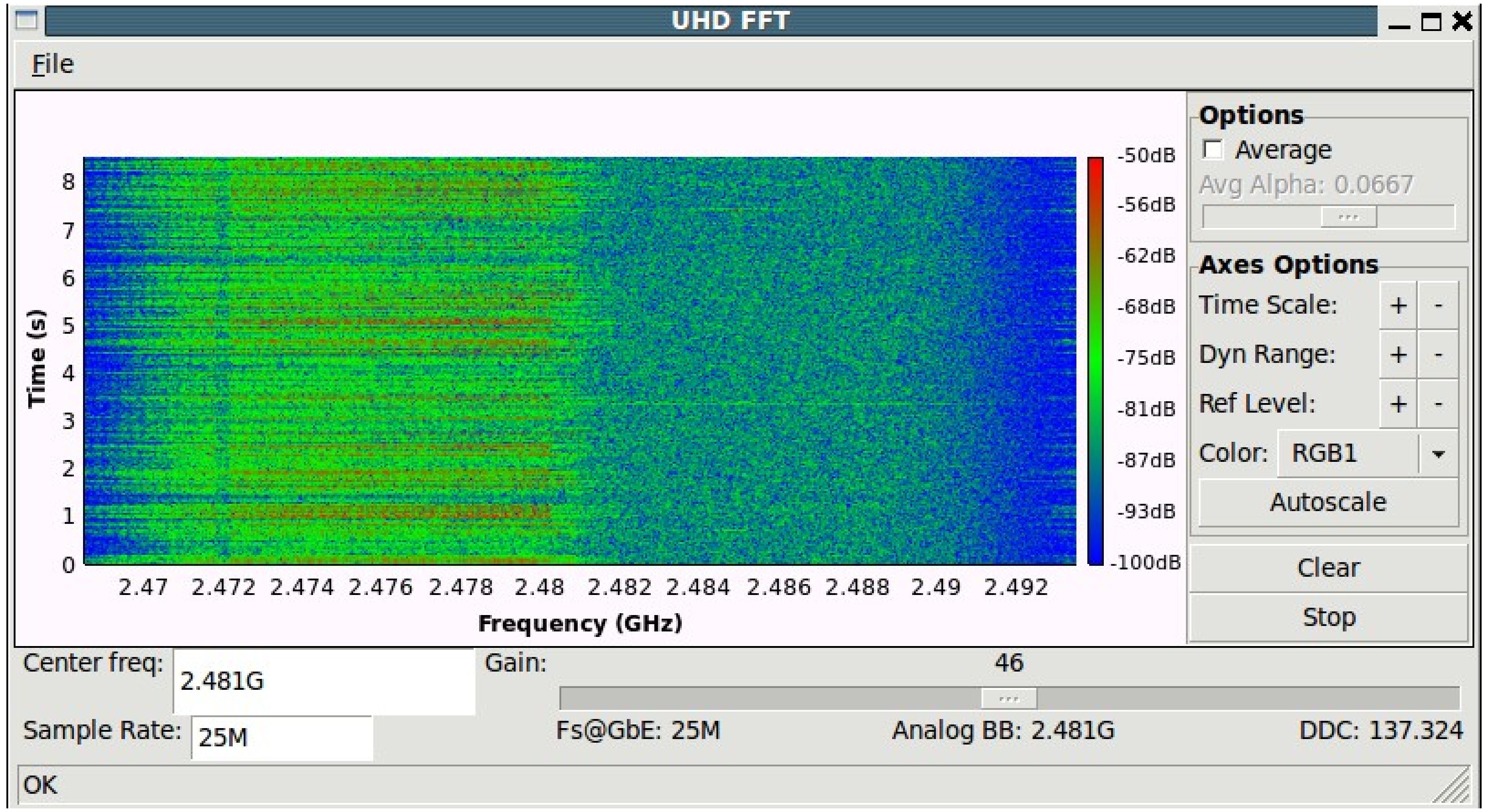
Interim summary

- Thinking + Images → DSP insights
  - Quadrature Signals
- Radio Peripheral → device insights, HW requirements
  - Sample Rate, Down Conversion, ADC/FPGA
- Checked out C++ → why Quadrature Signals
  - an Implementation of QPSK

FFT (demo)
visualize Badge radio
2.4 GHz – 2.4835 Ghz
one channel 1 Mhz
2.481 GHz

FFT – spectrum scope



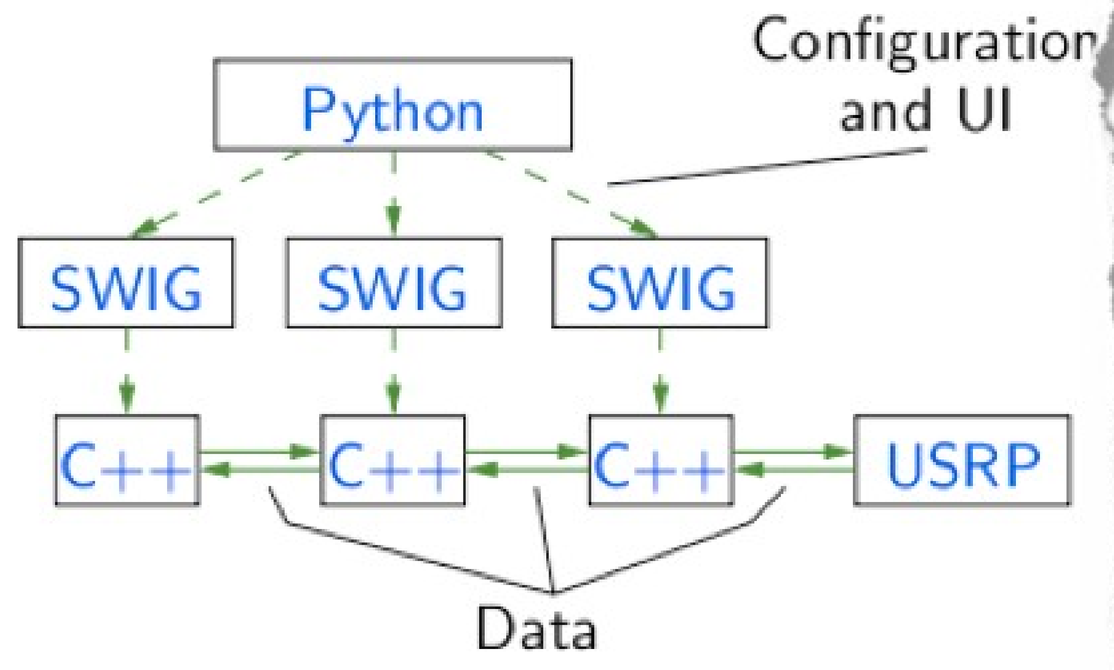


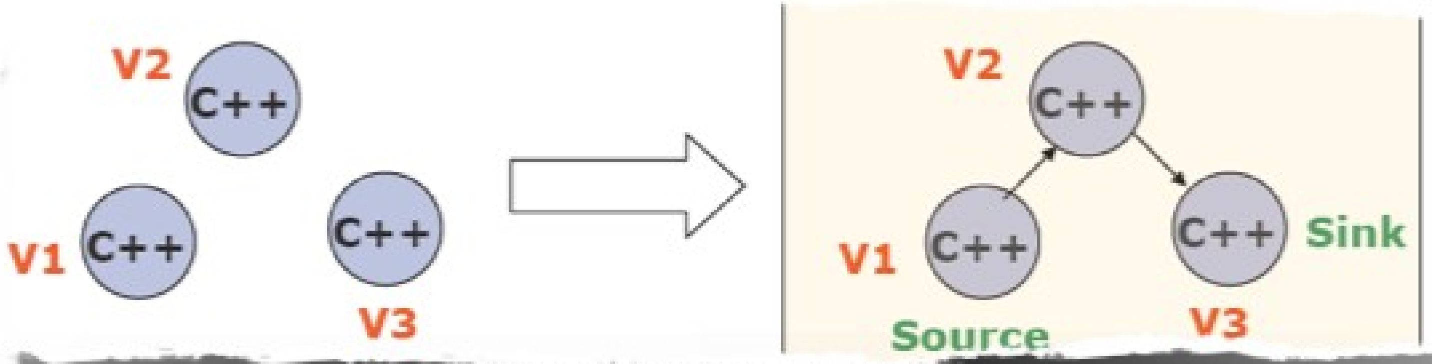
Left: FFT search for badge at the campside: no chance
Right: signals in 25 MHz spectrum in Waterfall-FFT

“GR Dive In” - there’s no documentation?

- GR lacks documentation and introductorily efforts
  - just Doxygen – due to active development
  - few direct literature, few presentations
- the GUI (GNU Radio companion) just covers parts
- some academic research, rarely in Software Engineering ;)

Software: GNU Radio





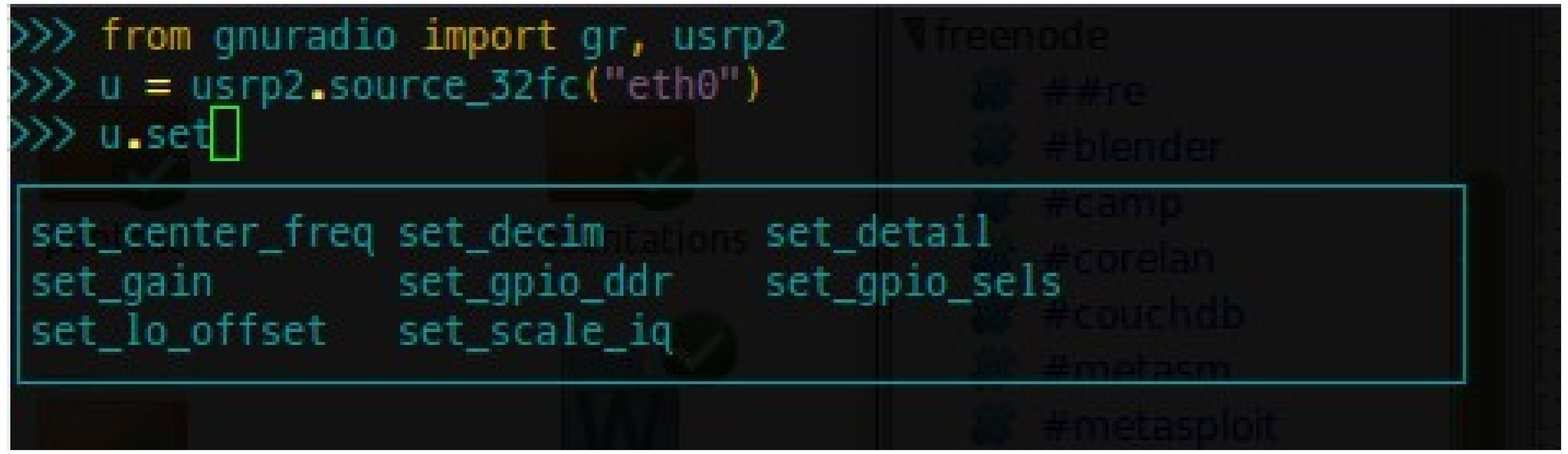
Python: orchestration and visualization
C++: wave-form transformation - "vector" handling

Swig: language interoperability - interface generator

no hw/sw buffering (GNU Radio Scheduler does that),
OOP (reuse everything!!!) - use the framework

price: software development know how:
C++ inheritance - DSP algorithms
Python OOP - connections, GUI (QT/WX +opengl) - real time plots leverage GPU

Python: parameter

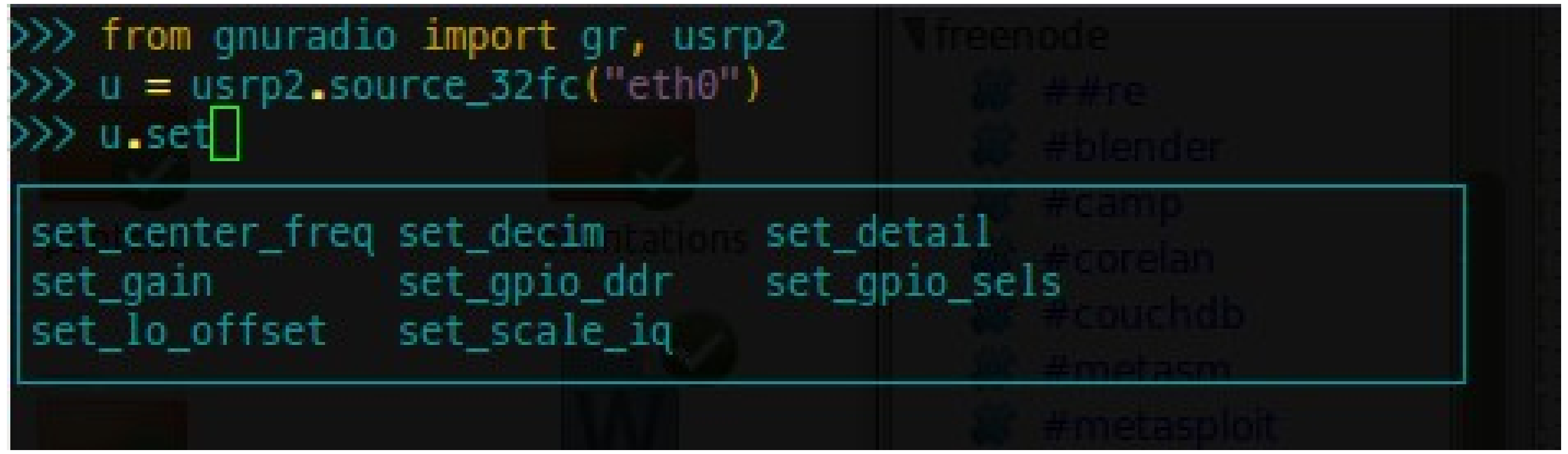


control parameter,
connections: sink -> ... -> source
source -> ... -> sink

parameter	effect
center-frequency	<- value ->
decimation	100 MS/s / value = sample-rate
gain	hw, pga

channel bandwidth depends

Python: parameter

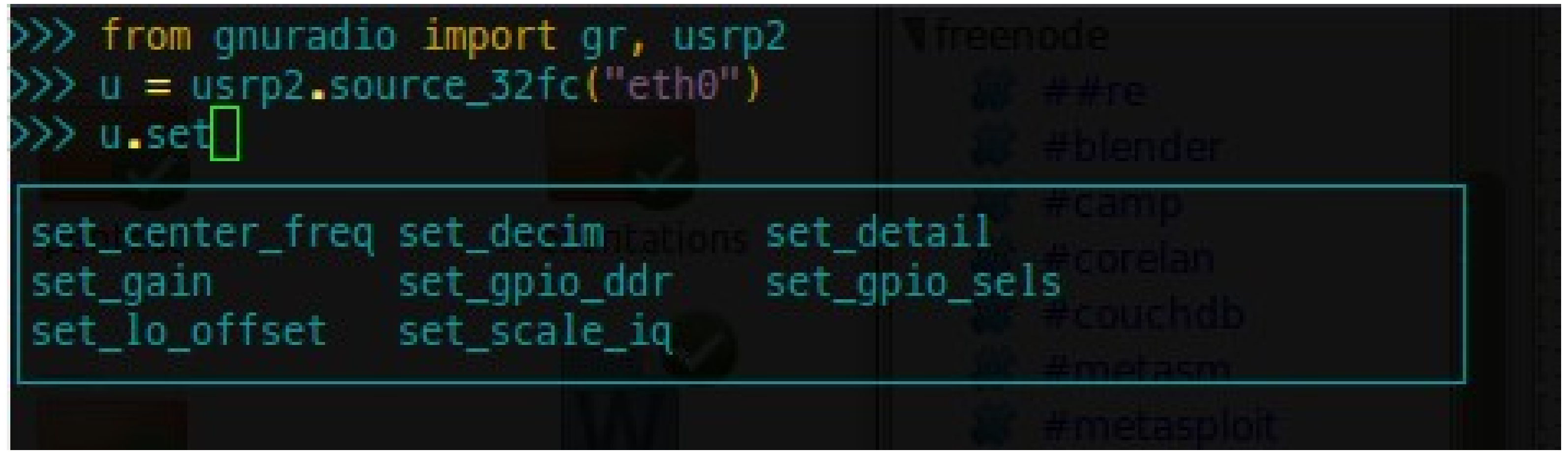


control parameter,
connections: sink -> ... -> source
source -> ... -> sink

parameter	effect
center-frequency	<- value ->
decimation	100 MS/s / value = sample-rate
gain	hw, pga

Nyquist Theorem

Python: parameter



control parameter,
connections: sink -> ... -> source
source -> ... -> sink

parameter	effect
center-frequency	<- value ->
decimation	100 MS/s / value = sample-rate
gain	hw, pga

too much?

Python: “Top-Block” concept: connections

GNU Radio Companion: Flow-graph, Python codegen from XML

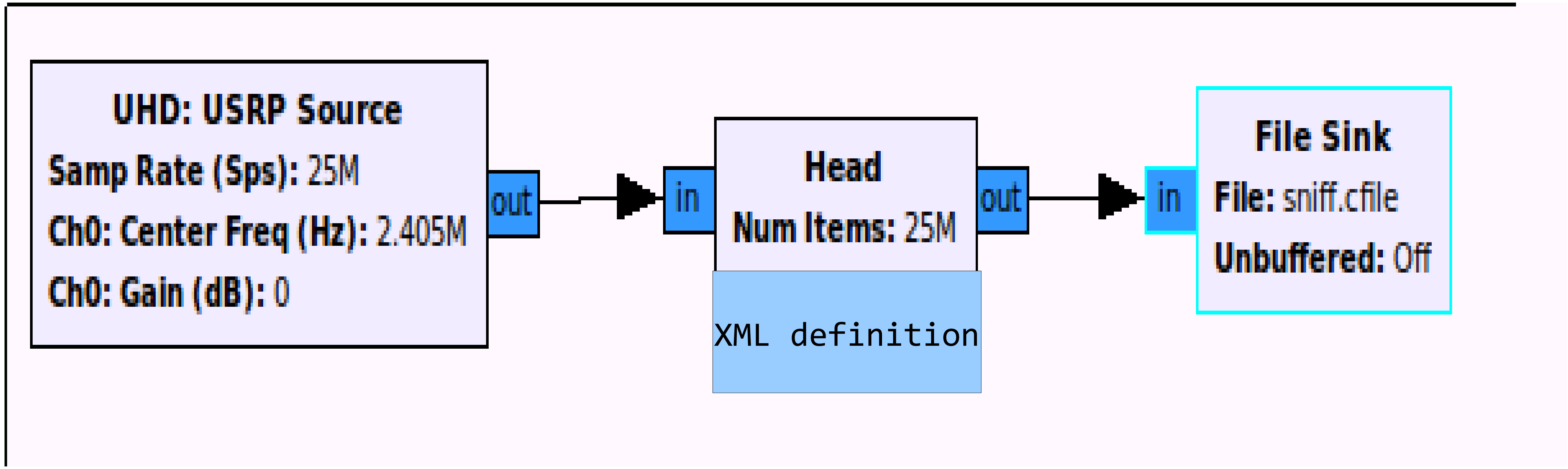
Top-Block hierarchy

layers encapsulate “packages” - reuse

global variables define parameters

(graphical) sink

Top-Block: Capture signals



USRP2 Source – 25 MS/s
Head – just capture 25 MS
File Sink – save as cfile (IEEE single-precision 4 Byte Floats)

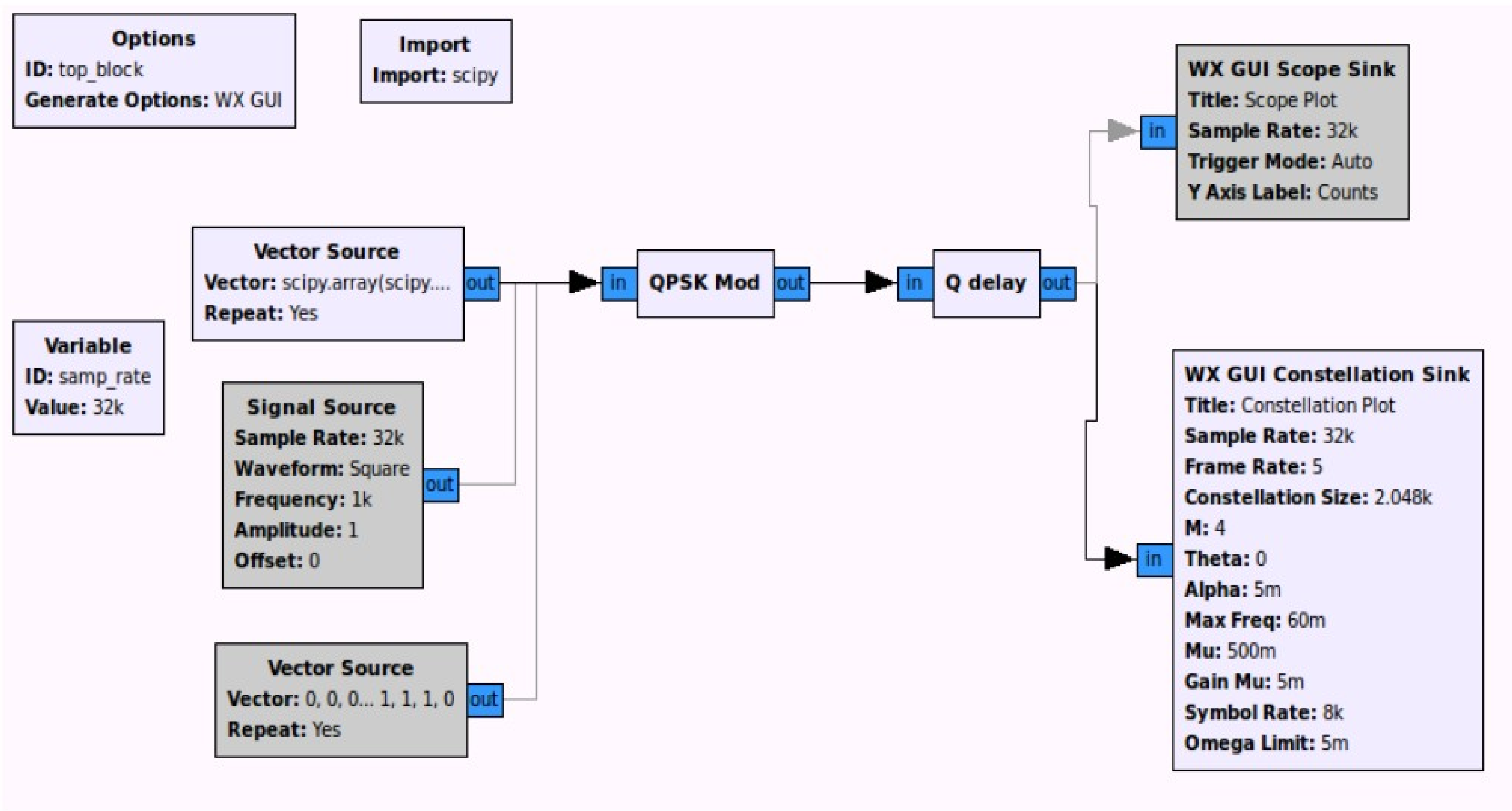
Python: GRC definitions

```xml
<?xml version="1.0"?>
<block>
  <name>QPSK Mod</name>
  <key>ucla_qpsk_modulator_cc</key>
  <category>802_15_4</category>
  <import>from gnuradio import ucla</import>
  <make>ucla.qpsk_modulator_cc()</make>

  <sink>
    <name>in</name>
    <type>complex</type>
  </sink>
  <source>
    <name>out</name>
    <type>complex</type>
  </source>
  <doc>
Generate a QPSK signal from a +/- 1 float stream.
For each two input symbols we output 4 complex symbols with a half-
sine
pulse shape.
  </doc>
</block>
```

Imports (Python)

Sink & Source

Use QPSK Modulation



Q-Phase gets delayed by half a symbol.

QPSK → oQPSK

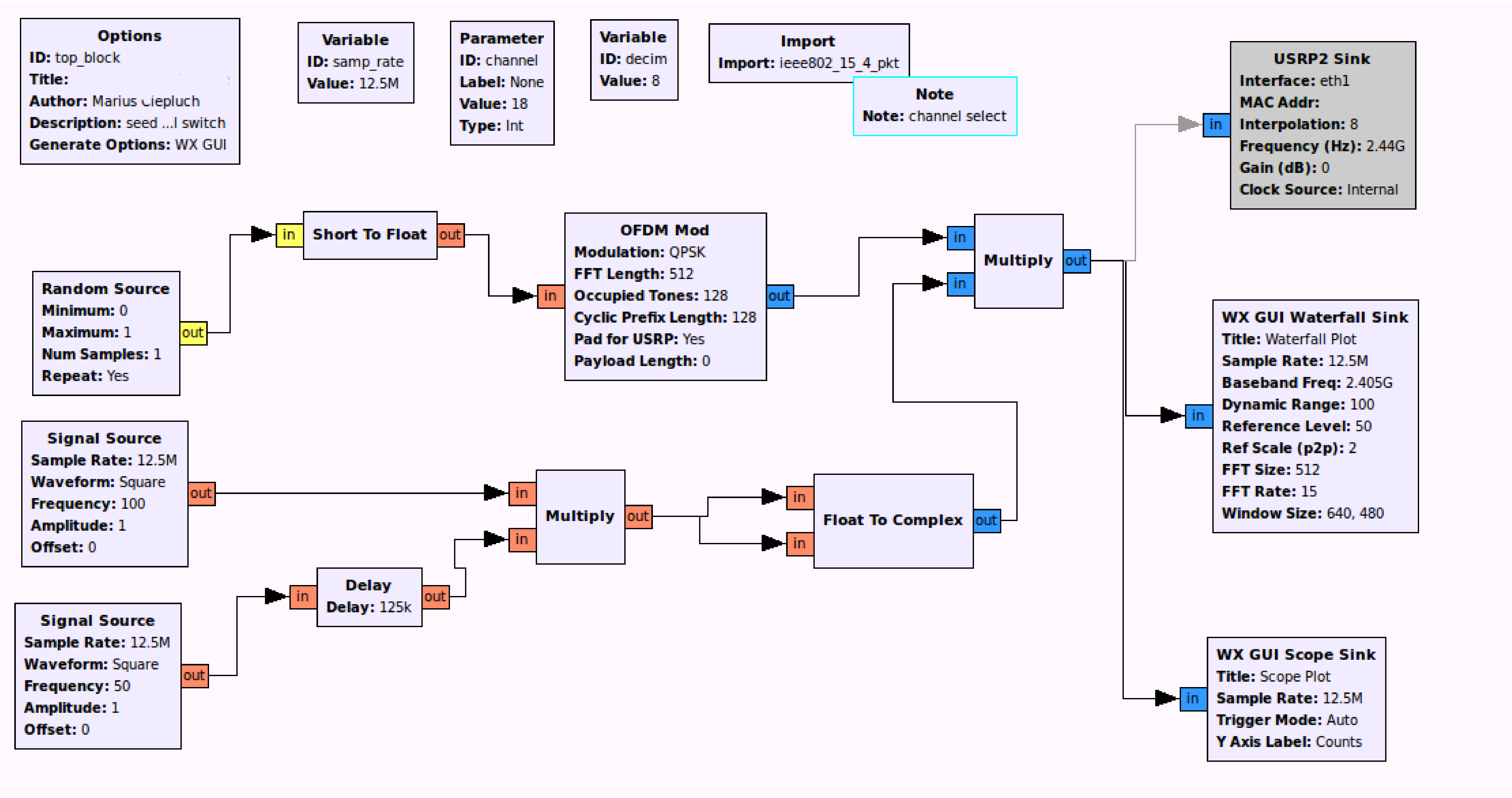
Q-Phase Delay Block

```
int
ucla_delay_cc::work (int noutput_items,
             gr_vector_const_void_star &input_items,
             gr_vector_void_star &output_items)
{
  gr_complex *in = (gr_complex *) input_items[0];
  gr_complex *out = (gr_complex *) output_items[0];

  for (int j = 0; j < noutput_items; j++)
    out[j] = gr_complex (real(in[j+d_delay]), imag(in[j]));

  return noutput_items;
}
```

Pointer in ring-buffer

+ d_delay → in-phase forward

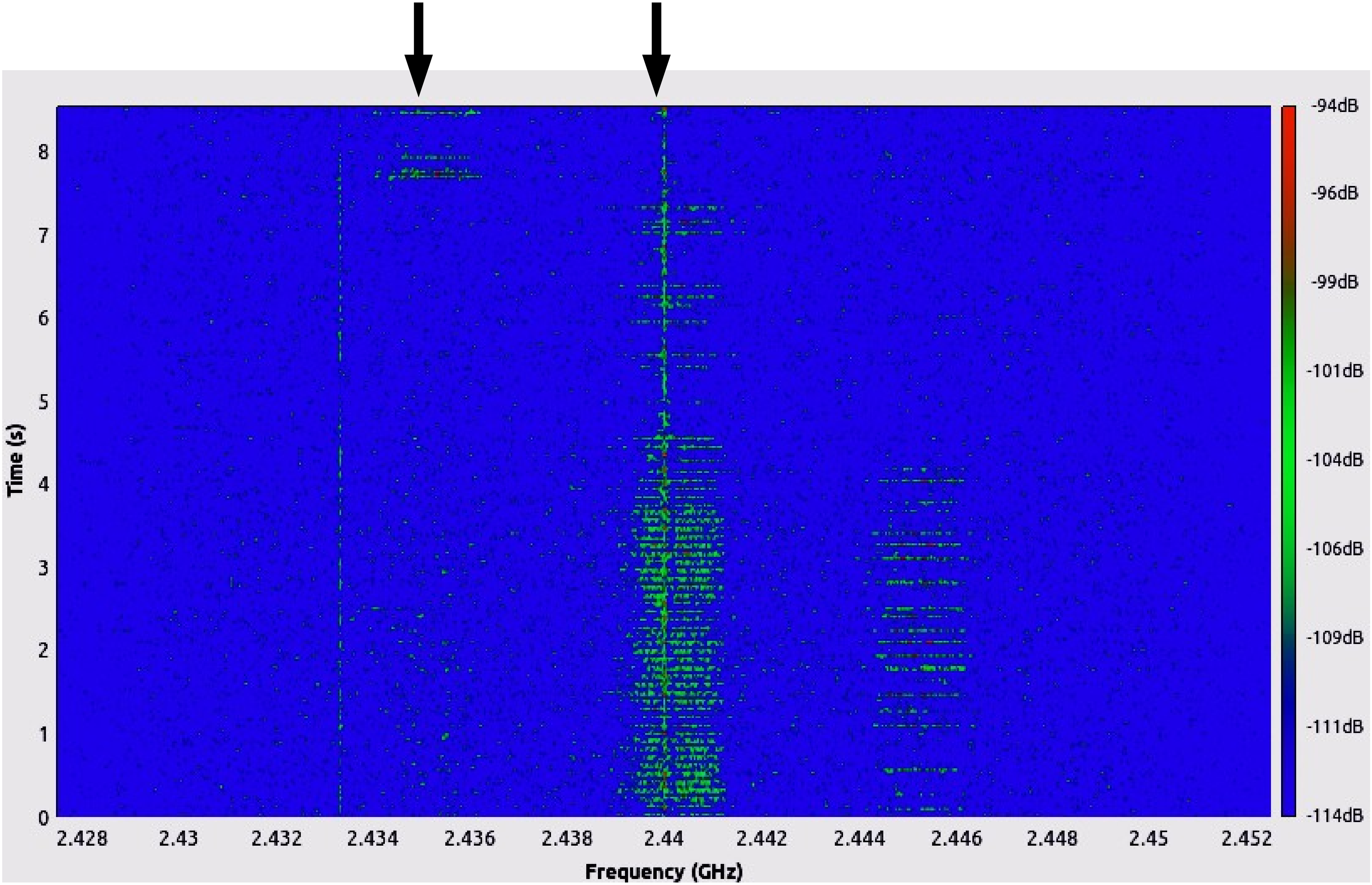
Python: GRC codegen



Interim summary

- GRC generates Python source (easy to change)
  - Paramaters - to control Top Blocks via scripts
- OFDM – spread interference over range (QPSK) – 4MHz
  - Verify real-time capabilities on IEEE 802.15.4 protocols e.g. - protocol specific (time on air)
- Hierarchical blocks will be integrated after restart

FFT Waterfall: channel hopping - time



waterfall uses
usrp.source_32fc()
IQ - each 32bit

limit on instantaneous
bandwidth
decimation minimally 4 –
USRP2. 25 MHz

Future of GNU Radio

- hopefully more GUI, GRC blocks and shared Flow-Graphs
  - better performance at GUI sinks (I/O exhaustion at X11 sucks - software may lose samples)
- real user documentation
  - more compatible peripheral radios - not „just“ USRPs
  - wider industry adaption and code contribution

Summary: stuff we see...

- ✔ Software Defined Radio with FOSS + modular HW
- ✔ GNU Radio Architecture
- ✔ Digital Signal Processing – one inch deep
- ✔ Implementations: C++, Python, XML
- ✔ Radio peripheral design: FPGA, ADC...

Sources

- http://dspguru.com/sites/dspguru/files/QuadSignals.pdf
  - slide, 25 – © pictures of Quadrature Signals
- http://wiesel.ece.utah.edu/redmine/projects/gr-ieee802-15
  - some source, GPL