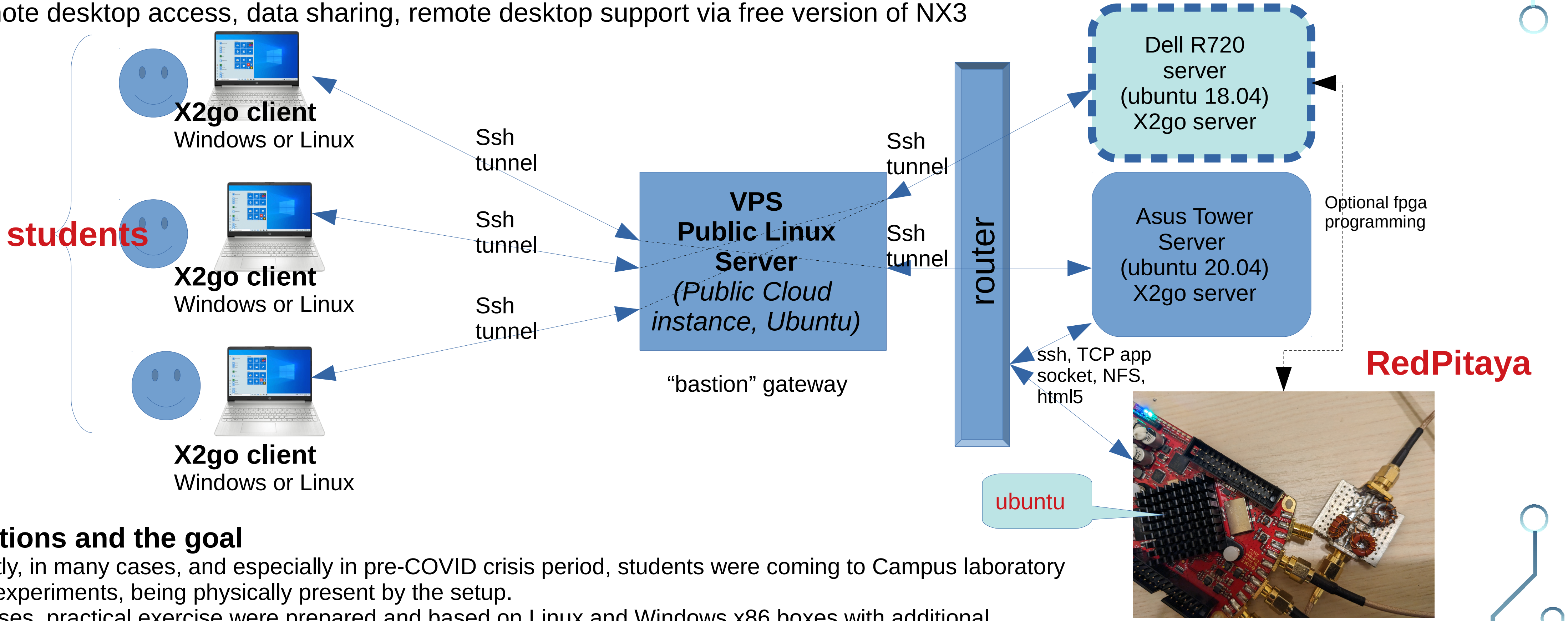
RELAB - Example of Remotely accesible Lab

Co-funded by the Erasmus+ Programme of the European Union

Implementation of Remote Lab setups

- Server side is Linux; Client can be Windows/Linux (single sw package need to be installed)
- without Public IP and behind a firewall/router, at your premises, based on reverse SSH tunnels
- Somewhat similar to AnyDesk, Teamviewer, but completely open source and under our control
- Remote desktop access, data sharing, remote desktop support via free version of NX3



Assumptions and the goal

Until recently, in many cases, and especially in pre-COVID crisis period, students were coming to Campus laboratory and doing experiments, being physically present by the setup.
In many cases, practical exercise were prepared and based on Linux and Windows x86 boxes with additional aparatus connected, e.g. using RPi3/4, Arduino framework, LabView, custom prepared scripts, applications.
We have selected **RedPitaya platform**, that can emulate 2 channel signal generator, and 2 channel osciloscope, based on Xilinx Zynq chip, with full embedded Linux support, I2C and SPI control oob, and FPGA programmability.
It can provide programmatically controlled signal input, and also acquisition of response for further analysis and comparison against simulation.

Detailed information on setup preparation on both client and server side are extension of guidelines provided in WebLab documents provided last year

Co-funded by the Erasmus+ Programme of the European Union

Digital twin model of some analog electrical circuits: Example of LC filters

Goal of the experiment:

- Interactively compare analog circuit simulation and physical setup behavior
- Identification of unknown model parameters by interactively expanding virtual model and measuring difference in circuit response.
    - Real world L and C components are represented with more complex network with unknown parameters (e.g. for L, series resistance & parallel, parasitic capacitor).

Method:

- Modify circuit structure, provide signal input and measure response.
    - Same stiumuli is applied on real-life model and virtual model of AC using LTSpice package.
    - Similarity metric of response of physical setup and simulated environment is also automatically calculated.
- Physical network parameters (R,L,C) are manually modified for the sake of simplicity
    - Two breadboards with different topologies and few different values of capacitors and toroid inductors offer variability in experiments
    - *We can also fully automate control side of the setup (parameter modification), over I2C (using circuits PCM9548/9534):*
        - *Relays, digital POT (MPT4017/18/19), capacitor (NCD2400M) over I2C, directly from RedPitaya (example schematic provided for reference)*

SW setup proxy bridge server:

- Linux, Ubuntu/Debian; SSH stack enabled; X2GO server
- User space packages: Octave; Wine package (for Windows emulation); Ltspice XVII installed on Linux;
- Python scripts to interact with LTSpice and WebApp.
- *Web App access is stretch goal and WIP*
    - *NGINX WebServer + Python Flask Web App for remote access,*
    - *Providing API to Python LTSpice scripts, and communication with RP over TCP sockets (optionally change parameters over I2C)*

SW setup lab server:

- RedPitaya C or Python code, to run acquisition and optionally I2C peripherals
- Access to RedPitaya via SSH or Web session.
    - RedPitaya generator/acquistion application communicates with backend over TCP socket; optionally sharing via NFS

Hardware requirements (BoM):

- Few toroids/inductors created with winding, capacitors and resistors (through hole), breadboard.
- LCR meter (optional)
- RedPitaya 125-14/122-16 (redpitaya.com)

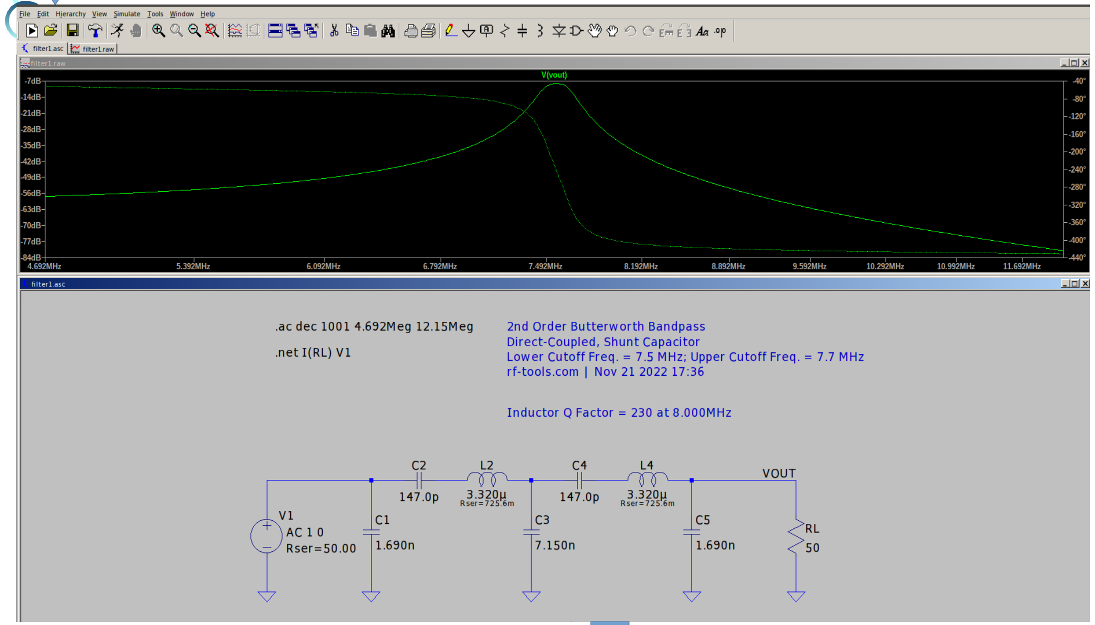
RELAB -
Analog circuits simulation digital twin
Experiment flow

Co-funded by the
Erasmus+ Programme
of the European Union

After installation setup, start LTSpice in wine:

```
exec wine '/home/username/.wine/drive_c/Program Files (x86)/LTC/LTspiceXVII/XVIIx86.exe' 'filter1.asc'
```

1) In LTSpice, modifying existing or create sch of electrical circuit to be simulated
2) After running simulation and modifying parameters, save in .asc format



```
.ac dec 1001 4.692Meg 12.15Meg

.net I(RL) V1
```

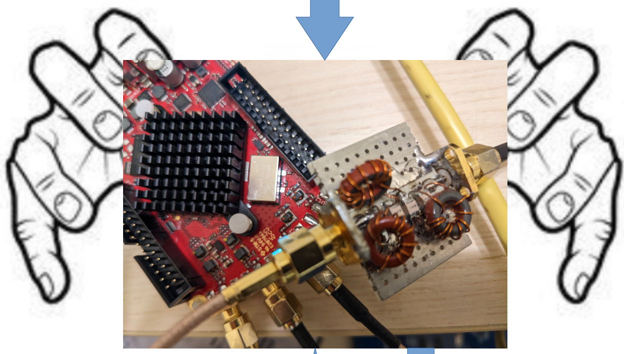
2nd Order Butterworth Bandpass
Direct-Coupled, Shunt Capacitor
Lower Cutoff Freq. = 7.5 MHz; Upper Cutoff Freq. = 7.7 MHz
rf-tools.com | Nov 21 2022 17:36

Inductor Q Factor = 230 at 8.000MHz

filter.asc

3) `source ltvenv/bin/activate; python3 ./myltr.py`, to simulate circuit in batch mode using PyLTSpice, plot simulated response
4) Initiate measurement of physical setup (communicate with RP)
5) Calculate similarity metrics with response from physical setup. If match within createria - break
6) Modify parameters (not topology) with sliders
7) Goto 5

- Design filter model or take one of few supplied examples
- Measure R, L, C values using LCR meter. If CR only is available estimate L based on number of turns.
- Do necessary assembly/solderingAttach to RedPitaya, OUT1->DUT->IN1



Raw measurements (DAC samples)

Raw measurements (ADC samples)

request

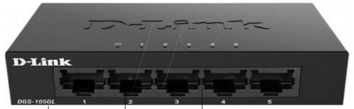
response

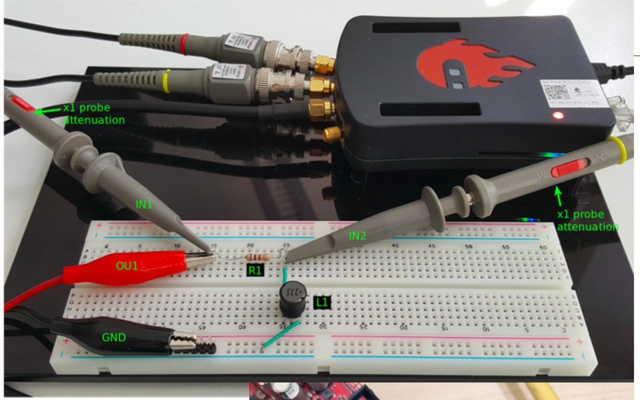
RP C-program:

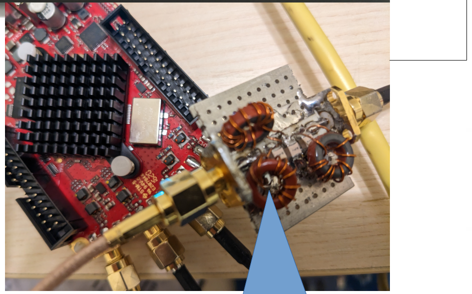
1) Wait for request
2) Initiate signal generation, at Fstart
3) Generate sine at Fn
4) Acquire signal samples (after passing through filter)
5) Increment Fn
6) Goto 3, until Fn riches Fend
7) Send RMS response back to requestor

RELAB - Setup for Analog circuits simulation digital twin

Co-funded by the Erasmus+ Programme of the European Union

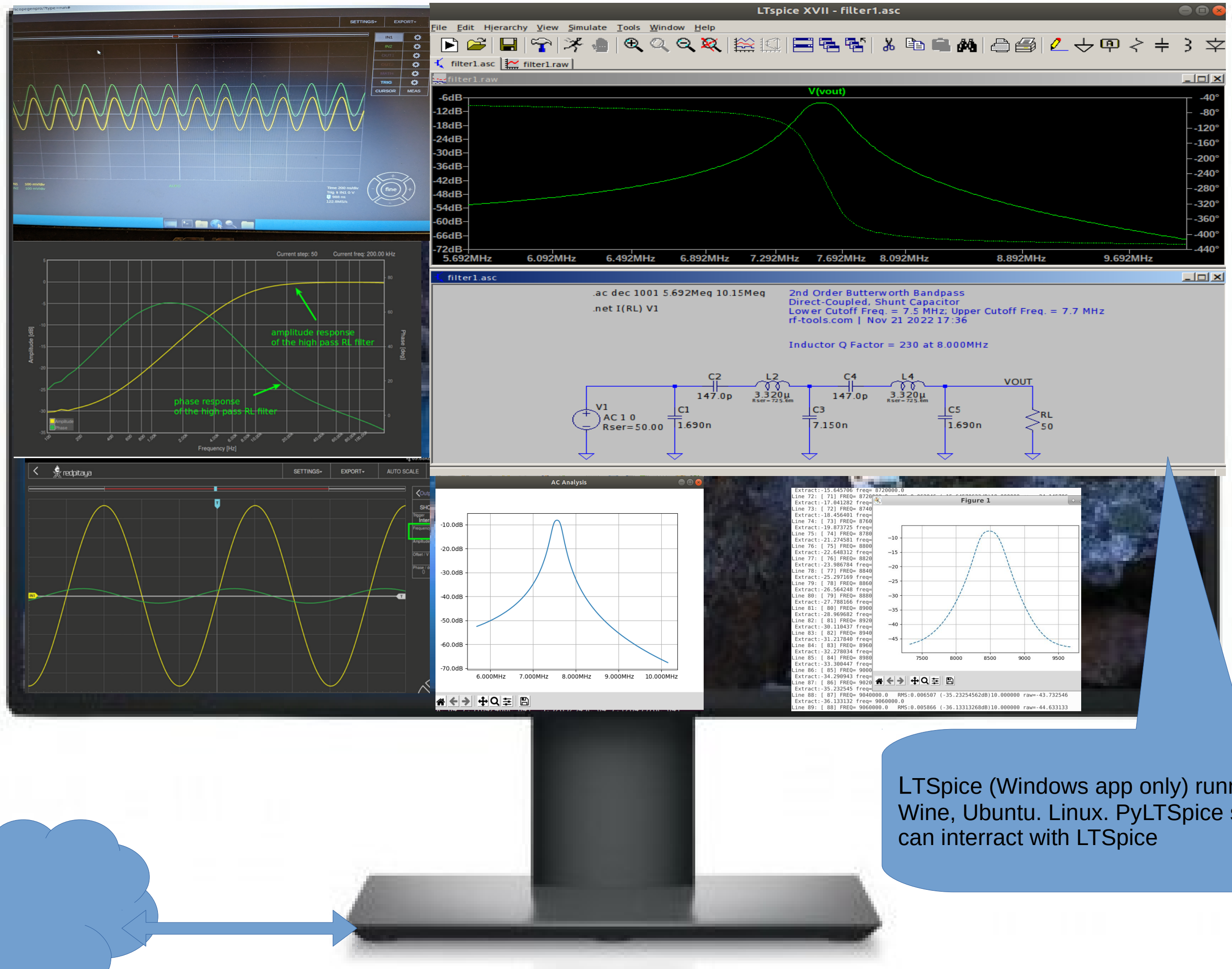






SERVER:

- Ubuntu
- X2go
- octave
- Wine
- LTSpice XVII
- Custom PyScripts
- For WebApp:
  - Nginx
  - *Flask WebApp*



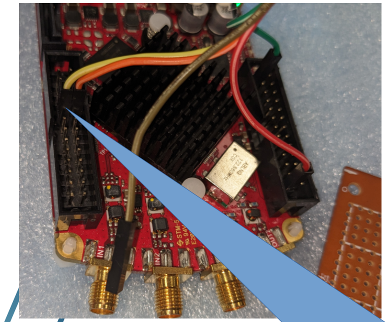
filter1.asc

```
.ac dec 1001 5.692Meg 10.15Meg
.net I(RL) V1
```

2nd Order Butterworth Bandpass
Direct-Coupled, Shunt Capacitor
Lower Cutoff Freq. = 7.5 MHz; Upper Cutoff Freq. = 7.7 MHz
rf-tools.com | Nov 21 2022 17:36

Inductor Q Factor = 230 at 8.000MHz

LTSpice (Windows app only) running in Wine, Ubuntu. Linux. PyLTSpice scripts can interract with LTSpice



DUT
LC filters of up to 5th order:

- Change filter order by adding / removing L/C
- Change number of turns for L change
- By stretching and squeezing turns change L
- Solder different capacitors

Optional I2C wiring for custom PCB board with digitally controlable C and R elements.

Students client station:
Windows or Linux

Repository of Open Educational Resources for Laboratory Support in Engineering and Natural Science

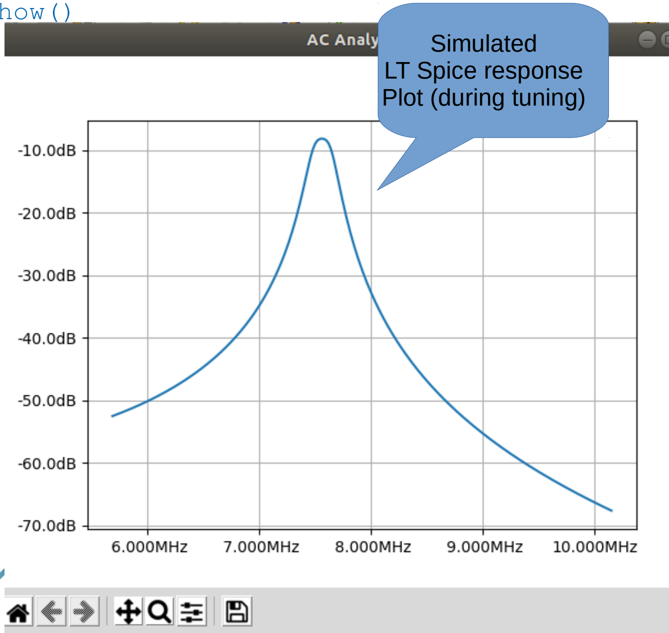
RELAB - Code snippets for Digital twin of Analog circuits simulation

Co-funded by the
Erasmus+ Programme
of the European Union

Code snippets:

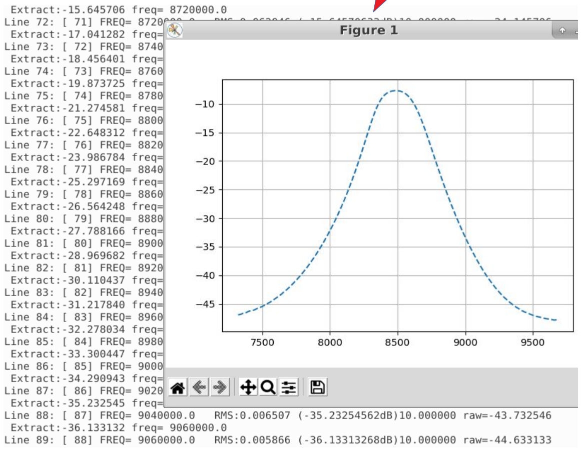
- LTSpice python driven simulation (x86)
  - Start from circuit entered in Ltspice and saved as filter1.asc
  - Change parameters in LTspice, save filter1.asc, run python script
  - Integrated Flask WebApp is exposing sliders for changing parameters inside selected few topologies
- Red Pitaya signal generator and acquisition

```
from PyLTSpice.LTSpice_RawRead import LTSpiceRawRead
from matplotlib import pyplot as plt
import numpy as np
import os
import subprocess

ltnetlist = './filter1.asc'
cmd = ['wine', '/home/relab/.wine/drive_c/Program Files (x86)/LTC/LTspiceXVII/XVIIx86.exe', '-Run -b -netlist']
cmd.append(ltnetlist)
subprocess.call(cmd, shell=False)

LTR = LTSpiceRawRead("filter1.raw")

print(LTR.get_trace_names())
print(LTR.get_raw_property())

TRF = LTR.get_trace("V(vout)")
x   = LTR.get_trace("frequency")
fig, ax = plt.subplots()
ax.plot(np.real(x) / 1000000, 20.0*np.log10(np.abs(TRF)))
ax.xaxis.set_major_formatter(plt.FormatStrFormatter('%6.3fMHz'))
ax.yaxis.set_major_formatter(plt.FormatStrFormatter('%4.1fdB'))
fig.canvas.manager.set_window_title('AC Analysis')
plt.grid()

plt.show()
```



Simulated
LT Spice response
Plot (during tuning)

Plot of measured
RMS data



```
 Extract:-15.645706 freq= 8720000.0
Line 72: [ 71] FREQ= 8720
 Extract:-17.041282 freq=
Line 73: [ 72] FREQ= 8740
 Extract:-18.456401 freq=
Line 74: [ 73] FREQ= 8760
 Extract:-19.873725 freq=
Line 75: [ 74] FREQ= 8780
 Extract:-21.274581 freq=
Line 76: [ 75] FREQ= 8800
 Extract:-22.648312 freq=
Line 77: [ 76] FREQ= 8820
 Extract:-23.986784 freq=
Line 78: [ 77] FREQ= 8840
 Extract:-25.297169 freq=
Line 79: [ 78] FREQ= 8860
 Extract:-26.564248 freq=
Line 80: [ 79] FREQ= 8880
 Extract:-27.788166 freq=
Line 81: [ 80] FREQ= 8900
 Extract:-28.969682 freq=
Line 82: [ 81] FREQ= 8920
 Extract:-30.110437 freq=
Line 83: [ 82] FREQ= 8940
 Extract:-31.217840 freq=
Line 84: [ 83] FREQ= 8960
 Extract:-32.278034 freq=
Line 85: [ 84] FREQ= 8980
 Extract:-33.300447 freq=
Line 86: [ 85] FREQ= 9000
 Extract:-34.290943 freq=
Line 87: [ 86] FREQ= 9020
 Extract:-35.232545 freq=
Line 88: [ 87] FREQ= 9040000.0   RMS:0.006507 (-35.23254562dB)10.000000 raw=-43.732546
 Extract:-36.133132 freq= 9060000.0
Line 89: [ 88] FREQ= 9060000.0   RMS:0.005866 (-36.13313268dB)10.000000 raw=-44.633133
```

```
/* Red Pitaya C API example Acquiring a signal from a buffer
 * This application acquires a signal on a specific channel */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include "rp.h"

int main(int argc, char **argv){
        float freq = 5500000;
        uint32_t buff_size = 16384;
        float *buff = (float *)malloc(buff_size * sizeof(float));

        /* Print error, if rp_Init() function failed */
        if(rp_Init() != RP_OK){
                fprintf(stderr, "Rp api init failed!\n");
        }

        for(int a = 0; a < 150; a++)
        {
          printf("\n[%2d] FREQ=%10.1f  ", a, freq);
          memset(buff, 0, buff_size * sizeof(float));

          rp_GenReset();
          rp_GenFreq(RP_CH_1, freq);
          rp_GenAmp(RP_CH_1, 0.03);
          rp_GenWaveform(RP_CH_1, RP_WAVEFORM_SINE);
          rp_GenOutEnable(RP_CH_1);

          freq += 20000;

          rp_AcqReset();
          rp_AcqSetDecimation(RP_DEC_1);
          rp_AcqSetTriggerDelay(0);

          rp_AcqStart();
          …
          rp_AcqGetOldestDataV(RP_CH_1, &buff_size, buff);
          printf(" Calc RMS:"); fflush(stdout);
          int i;
          double rms_tot = 0.0;
          for(i = 0; i < buff_size; i++){
            rms_tot += (double)buff[i] * (double)buff[i];
          }
          rms_tot = sqrt(rms_tot / (double)buff_size);
```

- Generate sine at different frequencies
- 150 steps
- Acquire
- Calculate RMS
- Store to NFS disk/send over TCP