Qucs-S: Simulation reference

Vadim Kuznetsov

February 25, 2024

Contents

List of Figures

List of Tables

1 Introduction

1.1 Common notes

Qucs-S was forked from the Qucs cross-platform circuit simulator in 2017. "S" letter indicates SPICE. The purpose of the Qucs-S subproject is to use free SPICE circuit simulation kernels with the Qucs GUI. It merges the power of SPICE and the simplicity of the Qucs GUI. Qucs intentionally uses its own SPICE incompatible simulation kernel Qucsator. It has advanced RF and AC domain simulation features, but most of the existing industrial SPICE models are incompatible with it. Qucs-S is not a simulator by itself, but requires to use a simulation backend with it. The schematic document format of Qucs and Qucs-S are fully compatible. Qucs-S allows to use the following simulation kernels with it:

- Ngspice is recommended to use. Ngspice is powerful mixed-level/mixed-signal circuit simulator. The most of industrial SPICE models are compatible with Ngspice. It has an excellent performance for time-domain simulation of switching circuits and powerful postprocessor.
- XYCE is a new SPICE-compatible circuit simulator written by Sandia from the scratch.
- SpiceOpus is developed by the Faculty of Electrical Engineering of the Ljubljana University. It based on the SPICE-3f5 code.
- Qucsator as backward compatible and for RF simulation with microwave devices and microstrip lines. Not recommended for general purpose circuits.

Qucs-S is a cross-platform software and supports a number of Linux distributions alongside with Windows™. The Linux packages are generated automatically with the Open Build Service (OBS) system. Check the official website to get the list of supported distributions. Please keep in mind that the installation packages doesn't provide the simulation kernel. It need to be installed separately. The Ngspice is recommended. For Debian and Ubuntu it is installed automatically as the dependency. Refer to Ngspice website for installation instructions for other platforms.

1.2 Qucs-S installation guide

2 Simulation with SPICE backend

2.1 Supported backends and quick simulator switching

2.2 DC operating point simulation

2.3 DC sweep simulation

2.4 Transient analysis

2.5 Transient analysis using initial conditions

2.6 AC analysis

2.7 Spectrum analysis (FFT)

2.8 Fourier analysis

2.9 Pole-zero analysis

2.10 Distortion analysis

2.11 Noise analysis

2.12 Parameter sweep

2.13 Using tuner mode

2.14 Nutmeg script simulation type

3 RF simulation

3.1 RF (S-parameter) simulation with Ngspice

3.2 RF (S-parameter) simulation with Qucsator

3.3 Harmonic balance simulation with XYCE

3.4 Harmonic balance simulation with Qucsator

4 Equations and postprocessing

4.1 Using equations with Ngspice

4.1.1 Parameters

4.1.2 Nutmeg equations

4.2 Using equations with Qucsator

5 Libraries and subcircuits

5.1 Library manager

5.2 Creating subcircuits

5.3 Creating libraries

5.4 Using SPICE models in schematics

5.4.1 SPICE modelcards .MODEL

5.4.2 SPICE subcircuits .SUBCKT

6 Compact modeling

6.1 Using Verilog-A models with Ngspice and OpenVAF

6.2 Verilog-A models synthesizer

7 Optimization

The optimization in the Qucs-S GUI is currently supported only using Qucsator.

7.1 Optimization with Ngspice

Ngspice in principle supports ASCO [1], but for Qucs-S optimization is possible only if you export netlist and run it from the CLI. The ASCO support for Qucs-S+Ngspice could be added in the future versions.

However, this section gives an example how to use ASCO with Ngspice. It is mandatory to install ASCO and to read the ASCO user's manual [2] and maybe [3] for supplementary information. The Ngspice manual [4] should be kept nearby in case of netlist debugging is necessary.

7.1.1 Example: optimization of a RLC circuit in the frequency domain and time domain simultaneously

Let's consider a RLC circuit to be optimized in the frequency domain and time domain simultaneously, as presented in Fig. 1.

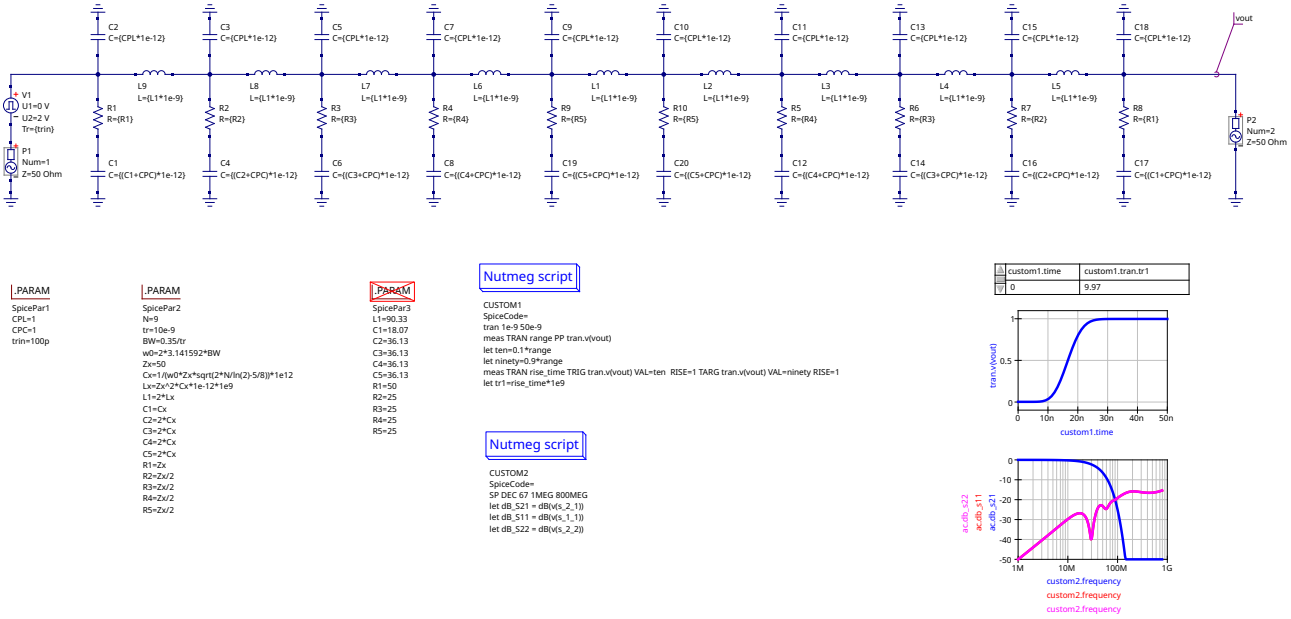

Figure 1: RLC circuit to be optimized in the frequency domain and time domain simultaneously

This circuit can be used as a 50 Ω low-pass filter at two additional boundary conditions [5], [6]:

1. defined step response (rise-time) in the time domain
2. 50 Ω matching of pass-band and stop-band in the frequency domain

The parameters `L1` [nH], `R1` ... `R5` [Ω], `[C1]` ... `C5` [pF] represent the low-pass filter core components. The parameters `CPL` [pF] and `CPC` [pF] represent printed circuit board parasitic capacitance's. `V1` is the pulse voltage source in the time domain set at 100 ps rise time using the parameter `trin`. `P1` and `P2` work as 50 Ω ports in the frequency domain (S-parameters) and time domain as well. Note, that the AC sources of `P1` and `P2` need to be disabled in the time

domain, as shown in Fig. 2 for P1, otherwise unwanted transient interference will occur in the time domain coming from the ports.

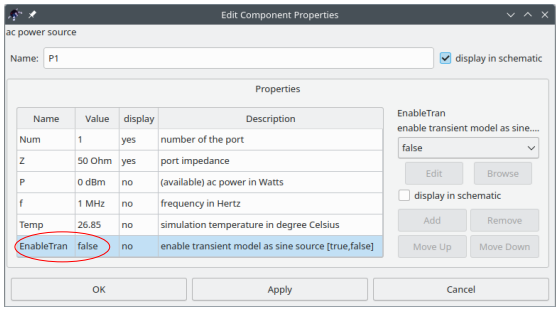

Figure 2: AC source of P1 and P2 need to be disabled in the time domain

Two separate Nutmeg scripts (`CUSTOM1` and `CUSTOM2` in Fig. 1) define the simulation and parameter extraction in the time domain and frequency domain, respectively. It is important to separate the frequency domain and time domain by two separate Nutmeg scripts. A mix of both domains in a single Nutmeg script would not work. The Nutmeg script `CUSTOM1` runs the transient simulation and extracts the rise-time of the output signal `vout`. The Nutmeg script `CUSTOM2` runs the S-parameter simulation and extracts the S-parameter S21, S11 and S22 in [dB]. For explanation of the Nutmeg syntax please read the Ngspice user's manual [4].

`.PARAM SpicePar1` defines the printed circuit board parasitic capacitances and the rise-time of the pulsed input voltage source. `.PARAM SpicePar2` defines analytical equations for the evaluation of the core filter components. However, these equations do not provide best possible matching in transition band from pass-band to stop-band and in the stop-band. There could be an optimum solution to release the matching in the lower pass-band and to improve the matching in the transition and stop-band. This optimum solution we want to find by optimization. `.PARAM SpicePar3` is optional parameter setting in case the analytical solution `.PARAM SpicePar2` is disabled. As a next step we deactivate `.PARAM SpicePar2` and activate `.PARAM SpicePar3`, which represent the calculated values out of the equations `.PARAM SpicePar2` using Gnu Octave [7]. Now run the simulation (press F2) and save the netlist, as shown in the dialog Fig. 3.

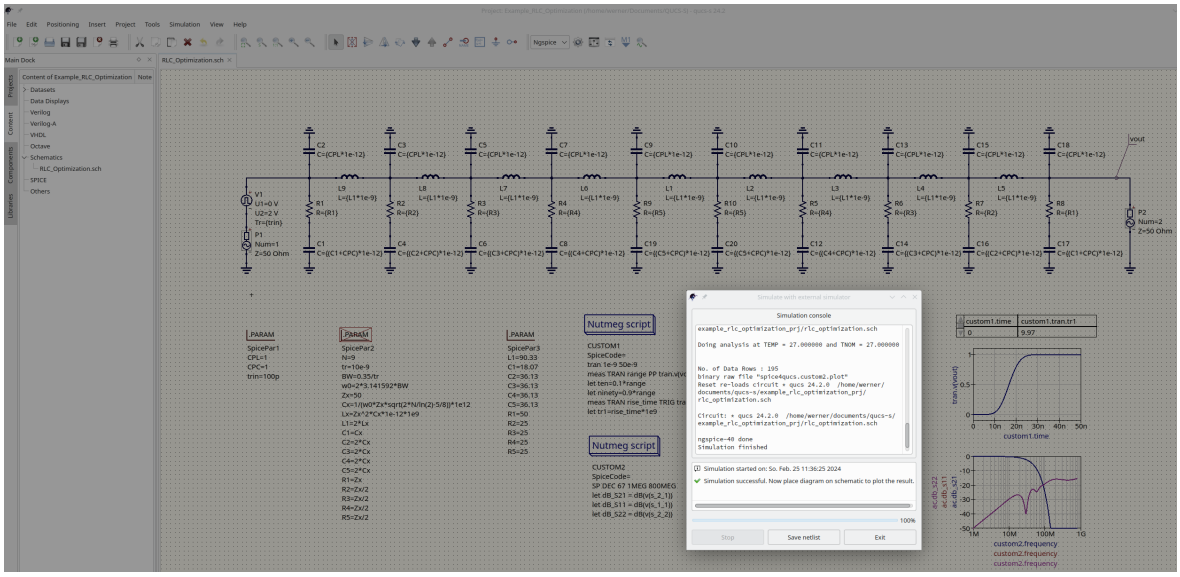

Figure 3: Save netlist after simulation

The previously saved netlist `netlist.cir` should look as follows:

```
* Qucs 24.2.0 /home/werner/Documents/QUCS-S/
   Example_RLC_Optimization_prj/RLC_Optimization.sch
.INCLUDE "/usr/local/share/qucs-s/xspice_cmlib/include/ngspice_mathfunc
   .inc"
.PARAM L1 = 90.33
.PARAM C1 = 18.07
.PARAM C2 = 36.13
.PARAM C3 = 36.13
.PARAM C4 = 36.13
.PARAM C5 = 36.13
.PARAM R1 = 50
.PARAM R2 = 25
.PARAM R3 = 25
.PARAM R4 = 25
.PARAM R5 = 25
.PARAM CPL = 1
.PARAM CPC = 1
.PARAM trin = 100p
R1 _net0 _net1 {R1} tc1=0.0 tc2=0.0
C1 0 _net0 {(C1+CPC)*1E-12}
C2 _net1 0 {CPL*1E-12}
C3 _net2 0 {CPL*1E-12}
R2 _net3 _net2 {R2} tc1=0.0 tc2=0.0
C4 0 _net3 {(C2+CPC)*1E-12}
C5 _net4 0 {CPL*1E-12}
R3 _net5 _net4 {R3} tc1=0.0 tc2=0.0
C6 0 _net5 {(C3+CPC)*1E-12}
C7 _net6 0 {CPL*1E-12}
R4 _net7 _net6 {R4} tc1=0.0 tc2=0.0
C8 0 _net7 {(C4+CPC)*1E-12}
C9 _net8 0 {CPL*1E-12}
L1 _net8 _net9 {L1*1E-9}
C10 _net9 0 {CPL*1E-12}
C11 _net10 0 {CPL*1E-12}
R5 _net11 _net10 {R4} tc1=0.0 tc2=0.0
C12 0 _net11 {(C4+CPC)*1E-12}
C13 _net12 0 {CPL*1E-12}
R6 _net13 _net12 {R3} tc1=0.0 tc2=0.0
C14 0 _net13 {(C3+CPC)*1E-12}
C15 _net14 0 {CPL*1E-12}
R7 _net15 _net14 {R2} tc1=0.0 tc2=0.0
C16 0 _net15 {(C2+CPC)*1E-12}
R8 _net16 vout {R1} tc1=0.0 tc2=0.0
C17 0 _net16 {(C1+CPC)*1E-12}
C18 vout 0 {CPL*1E-12}
L2 _net9 _net10 {L1*1E-9}
L3 _net10 _net12 {L1*1E-9}
L4 _net12 _net14 {L1*1E-9}
L5 _net14 vout {L1*1E-9}
```

```
L6 _net6 _net8 {L1*1E-9}
L7 _net4 _net6 {L1*1E-9}
L8 _net2 _net4 {L1*1E-9}
L9 _net1 _net2 {L1*1E-9}
R9 _net17 _net8 {R5} tc1=0.0 tc2=0.0
C19 0 _net17 {(C5+CPC)*1E-12}
R10 _net18 _net9 {R5} tc1=0.0 tc2=0.0
C20 0 _net18 {(C5+CPC)*1E-12}
VP2 vout 0 dc 0 ac 0.316228 portnum 2 z0 50
V1 _net1 _net19 DC 0 PULSE(0 2 0 {TRIN} 0.1N 0.001 1e+06) AC 0
VP1 _net19 0 dc 0 ac 0.316228 portnum 1 z0 50

.control

tran 1e-9 50e-9
meas TRAN range PP tran.v(vout)
let ten=0.1*range
let ninety=0.9*range
meas TRAN rise_time TRIG tran.v(vout) VAL=ten RISE=1 TARG tran.v(vout)
   VAL=ninety RISE=1
let tr1=rise_time*1e9

write spice4qucs.custom1.plot V(vout) ten ninety tr1
destroy all
reset

SP DEC 67 1MEG 800MEG
let dB_S21 = dB(v(s_2_1))
let dB_S11 = dB(v(s_1_1))
let dB_S22 = dB(v(s_2_2))
write spice4qucs.custom2.plot V(vout) dB_S21 dB_S11 dB_S22
destroy all
reset

exit
.endc
.END
```

This netlist is now the basis for optimization and further processing with ASCO. The step-by-step optimization procedure is as follows:

1. Create a folder `/asco` anywhere on your computer. The name of this folder is not important. `asco` is just a name recommendation.

2. Copy the previous netlist `netlist.cir` and rename to `/asco/netlist.sp`.
   Now the folder `/asco` contains just one file: `netlist.sp`

3. Now we have to modify the netlist `/asco/netlist.sp` in order to remove unwanted lines. Remove the `.control` to `.endc` block. Mark parameters to be changed, e.g.: `#R1#`. `L1` was set from 99.33 nH to 100 nH, because of commercially available value from Coilcraft [8]. Note the unit of inductance in the netlist is [nH], unit of capacitance is [pF], as there

are multipliers 1E-9 and 1E-12 at the component statements, respectively. Finally, the netlist netlist.sp should look like this:

```
* Qucs 24.2.0 /home/werner/Documents/QUCS-S/Example_RLC_Optimization_prj/
   RLC_Optimization.sch
.INCLUDE "/usr/local/share/qucs-s/xspice_cmlib/include/ngspice_mathfunc.
   inc"
.PARAM L1 = 100
.PARAM C1 = #C1#
.PARAM C2 = #C2#
.PARAM C3 = #C3#
.PARAM C4 = #C4#
.PARAM C5 = #C5#
.PARAM R1 = #R1#
.PARAM R2 = #R2#
.PARAM R3 = #R3#
.PARAM R4 = #R4#
.PARAM R5 = #R5#
.PARAM CPL = 1
.PARAM CPC = 1
.PARAM trin = 100p
R1 _net0 _net1 {R1} tc1=0.0 tc2=0.0
C1 0 _net0 {(C1+CPC)*1E-12}
C2 _net1 0 {CPL*1E-12}
C3 _net2 0 {CPL*1E-12}
R2 _net3 _net2 {R2} tc1=0.0 tc2=0.0
C4 0 _net3 {(C2+CPC)*1E-12}
C5 _net4 0 {CPL*1E-12}
R3 _net5 _net4 {R3} tc1=0.0 tc2=0.0
C6 0 _net5 {(C3+CPC)*1E-12}
C7 _net6 0 {CPL*1E-12}
R4 _net7 _net6 {R4} tc1=0.0 tc2=0.0
C8 0 _net7 {(C4+CPC)*1E-12}
C9 _net8 0 {CPL*1E-12}
L1 _net8 _net9 {L1*1E-9}
C10 _net9 0 {CPL*1E-12}
C11 _net10 0 {CPL*1E-12}
R5 _net11 _net10 {R4} tc1=0.0 tc2=0.0
C12 0 _net11 {(C4+CPC)*1E-12}
C13 _net12 0 {CPL*1E-12}
R6 _net13 _net12 {R3} tc1=0.0 tc2=0.0
C14 0 _net13 {(C3+CPC)*1E-12}
C15 _net14 0 {CPL*1E-12}
R7 _net15 _net14 {R2} tc1=0.0 tc2=0.0
C16 0 _net15 {(C2+CPC)*1E-12}
R8 _net16 vout {R1} tc1=0.0 tc2=0.0
C17 0 _net16 {(C1+CPC)*1E-12}
C18 vout 0 {CPL*1E-12}
L2 _net9 _net10 {L1*1E-9}
L3 _net10 _net12 {L1*1E-9}
L4 _net12 _net14 {L1*1E-9}
```

```
L5 _net14 vout {L1*1E-9}
L6 _net6 _net8 {L1*1E-9}
L7 _net4 _net6 {L1*1E-9}
L8 _net2 _net4 {L1*1E-9}
L9 _net1 _net2 {L1*1E-9}
R9 _net17 _net8 {R5} tc1=0.0 tc2=0.0
C19 0 _net17 {(C5+CPC)*1E-12}
R10 _net18 _net9 {R5} tc1=0.0 tc2=0.0
C20 0 _net18 {(C5+CPC)*1E-12}
VP2 vout 0 dc 0 ac 0.316228 portnum 2 z0 50
V1 _net1 _net19 DC 0 PULSE(0 2 0 {TRIN} 0.1N 0.001 1e+06) AC 0
VP1 _net19 0 dc 0 ac 0.316228 portnum 1 z0 50
.END
```

4. So far, we have the one and only file `netlist.sp` inside the `/asco` folder. But we need another file `netlist.cfg`, which represents the configuration file for the ASCO optimizer. The file `netlist.cfg` should look like this:

```
#Optimization Flow#
Alter:no $do we want to do corner analysis?
MonteCarlo:no $do we want to do MonteCarlo analysis?
AlterMC cost:1.00 $point at which we want to start ALTER and/or
   MONTECARLO
ExecuteRF:no $execute or not the RF module to add RF parasitics?
SomethingElse: $
#


#DE#
choice of method:3
maximum no. of iterations:200
Output refresh cycle:2
No. of parents NP:100
Constant F:0.75
Crossover factor CR:1
Seed for pseudo random number generator:3
Minimum Cost Variance:1e-6
Cost objectives:10
Cost constraints:1000
#

# ALTER #
#

#Monte Carlo#
#

# Parameters #
C1:#C1#:18:5:100:LIN_DOUBLE:OPT
C2:#C2#:36:5:200:LIN_DOUBLE:OPT
C3:#C3#:36:5:200:LIN_DOUBLE:OPT
```

```
C4:#C4#:36:5:200:LIN_DOUBLE:OPT
C5:#C5#:36:5:200:LIN_DOUBLE:OPT
R1:#R1#:50:30:70:LIN_DOUBLE:OPT
R2:#R2#:25:5:100:LIN_DOUBLE:OPT
R3:#R3#:25:5:100:LIN_DOUBLE:OPT
R4:#R4#:25:5:100:LIN_DOUBLE:OPT
R5:#R5#:25:5:100:LIN_DOUBLE:OPT
#

# Measurements #
Rise_Time:vout:EQ:10e-9
S11_Max:---:LE:-30
#


# Post Processing #
#


#this is the last line
```

Take care about the `#DE#` section:

- set `maximum no. of iterations:200`
- set `No. of parents NP:100`, which means about 10 times the number of parameters to be optimize (we have 10)

The `# Parameters #` section defines optimization variables and ranges. `C1:#C1#:18:5:100:LIN_DOUBLE:OPT` means: enable `C1` for optimization with start value 18 in the range from 5 to 100, as linear double (float). `OPT` means enable for optimization.

Finally, the `# Measurements #` section defines the goals for optimization. `Rise_Time:vout:EQ:10e-9` means: optimize the circuit for `Rise_Time` at node `vout` is equal to `10e-9`, which denotes 10 ns. `S11_Max:---:LE:-30` means: let `S11_Max` less or equal to -30 dB.

As we have defined 2 goals, we need also to goal files for ASCO. These 2 files have to have same file name: `Rise_Time` and `S11_Max`, which must be placed in a sub-folder named `/asco/extract`.

5. The first goal file `/asco/extract/Rise_Time` looks like this:

```
# Info #
Name:Rise_Time
Symbol:ZRise_Time
Unit:S
Analysis type:TRAN
Definition: Rise Time
Note:
#

# Commands #
.control
```

```
TRAN 1e-9 50e-9
MEAS TRAN range PP tran.v(#NODE#)
let ten=0.1*range
let ninety=0.9*range
MEAS TRAN #SYMBOL# TRIG tran.v(#NODE#) VAL=ten RISE=1 TARG tran.v(#NODE#)
    VAL=ninety RISE=1
print #SYMBOL#
.endc
#
```

The `# Commands #` section defines the controls of the transient analysis and the parameter extraction. `#NODE#` will be replaced by `vout`, as given in the `# Measurements #` section of the `netlist.cfg`. `#SYMBOL#`, the extracted rise time, will be printed as variable with a name `ZRise_Time` as defined in the `# Info #` section above.

6. The second goal file `/asco/exract/S11_Max` looks as follow:

```
# Info #
Name:S11_Max
Symbol:ZS11_Max
Unit:DB
Analysis type:SP
Definition: S11
Note:
#

# Commands #
.control
SP DEC 67 10e6 200e6
let s11xdb=maximum(dB(v(s_1_1)))
print s11xdb
.endc
#


# Post Processing #
MEASURE_VAR: #SYMBOL#: SEARCH_FOR:'s11xdb = '
#
```

It is important, that the `print s11xdb` statement fits to the `MEASURE_VAR: #SYMBOL#: SEARCH_FOR:'s11xdb = '`. Otherwise ASCO would not find the extracted values of $\max(S11)$.

7. At the end, now we have created 4 files:
   (a) `/asco/netlist.sp`
   (b) `/asco/netlist.cfg`
   (c) `/asco/extract/Rise_Time`
   (d) `/asco/extract/S11_Max`
8. Now it is time to open a terminal in the `/asco` folder and run ASCO:
   `asco -ngspice netlist.sp`

9. On the running ASCO optimizer, have a look on the runing value of `cost-variance`. It should go down < 1 pretty much soon. If not, there is a problem in the results handover procedure from ngspice to ASCO. After about 5 minutes the optimizer has finished the maximum number of iterations. The results file has the hostname of your computer with the extension `sp`. In my case the file looks as follows:

```
* Qucs 24.2.0 /home/werner/Documents/QUCS-S/Example_RLC_Optimization_prj/
   RLC_Optimization.sch
.INCLUDE "/usr/local/share/qucs-s/xspice_cmlib/include/ngspice_mathfunc.
   inc"
.PARAM L1 = 100
.PARAM C1 = 3.433807E+01
.PARAM C2 = 2.598532E+01
.PARAM C3 = 4.131560E+01
.PARAM C4 = 4.284842E+01
.PARAM C5 = 3.658031E+01
.PARAM R1 = 5.124539E+01
.PARAM R2 = 2.432580E+01
.PARAM R3 = 8.690387E+00
.PARAM R4 = 2.140148E+01
.PARAM R5 = 1.722261E+01
.PARAM CPL = 1
.PARAM CPC = 1
.PARAM trin = 100p
R1 _net0 _net1 {R1} tc1=0.0 tc2=0.0
C1 0 _net0 {(C1+CPC)*1E-12}
C2 _net1 0 {CPL*1E-12}
C3 _net2 0 {CPL*1E-12}
R2 _net3 _net2 {R2} tc1=0.0 tc2=0.0
C4 0 _net3 {(C2+CPC)*1E-12}
C5 _net4 0 {CPL*1E-12}
R3 _net5 _net4 {R3} tc1=0.0 tc2=0.0
C6 0 _net5 {(C3+CPC)*1E-12}
C7 _net6 0 {CPL*1E-12}
R4 _net7 _net6 {R4} tc1=0.0 tc2=0.0
C8 0 _net7 {(C4+CPC)*1E-12}
C9 _net8 0 {CPL*1E-12}
L1 _net8 _net9 {L1*1E-9}
C10 _net9 0 {CPL*1E-12}
C11 _net10 0 {CPL*1E-12}
R5 _net11 _net10 {R4} tc1=0.0 tc2=0.0
C12 0 _net11 {(C4+CPC)*1E-12}
C13 _net12 0 {CPL*1E-12}
R6 _net13 _net12 {R3} tc1=0.0 tc2=0.0
C14 0 _net13 {(C3+CPC)*1E-12}
C15 _net14 0 {CPL*1E-12}
R7 _net15 _net14 {R2} tc1=0.0 tc2=0.0
C16 0 _net15 {(C2+CPC)*1E-12}
R8 _net16 vout {R1} tc1=0.0 tc2=0.0
C17 0 _net16 {(C1+CPC)*1E-12}
C18 vout 0 {CPL*1E-12}
```

```
L2 _net9 _net10 {L1*1E-9}
L3 _net10 _net12 {L1*1E-9}
L4 _net12 _net14 {L1*1E-9}
L5 _net14 vout {L1*1E-9}
L6 _net6 _net8 {L1*1E-9}
L7 _net4 _net6 {L1*1E-9}
L8 _net2 _net4 {L1*1E-9}
L9 _net1 _net2 {L1*1E-9}
R9 _net17 _net8 {R5} tc1=0.0 tc2=0.0
C19 0 _net17 {(C5+CPC)*1E-12}
R10 _net18 _net9 {R5} tc1=0.0 tc2=0.0
C20 0 _net18 {(C5+CPC)*1E-12}
VP2 vout 0 dc 0 ac 0.316228 portnum 2 z0 50
V1 _net1 _net19 DC 0 PULSE(0 2 0 {TRIN} 0.1N 0.001 1e+06) AC 0
VP1 _net19 0 dc 0 ac 0.316228 portnum 1 z0 50
.control
TRAN 1e-9 50e-9
MEAS TRAN range PP tran.v(vout)
let ten=0.1*range
let ninety=0.9*range
MEAS TRAN ZRise_Time0 TRIG tran.v(vout) VAL=ten RISE=1 TARG tran.v(vout)
   VAL=ninety RISE=1
print ZRise_Time0
.endc
.control
SP DEC 67 10e6 300e6
let s11xdb=maximum(dB(v(s_1_1)))
print s11xdb
.endc
.end
```

On top in the results netlist above you can see the optimized results of the parameters.

As a result the optimized parameters can be checked using the Qucs-S schematic with the `.PARAM SpicePar4` block, as shown in Fig. 4. It can be seen in the results, that the rise time results to 10.1 ns and the maximum S11 is in the range of −25 dB or less, in the frequency range up to 300 MHz, which is a significant improvement over the parameters shown in Fig. 1.

Conclusions: Overall, the tricky thing is mainly the handover of extracted results from the ngspice simulation to the ASCO optimizer executable. The optimizer control (weighting function) of this example could be improved. However, the example gives a basic idea how to optimize a circuit in the time domain and frequency domain simultaneously using ASCO.

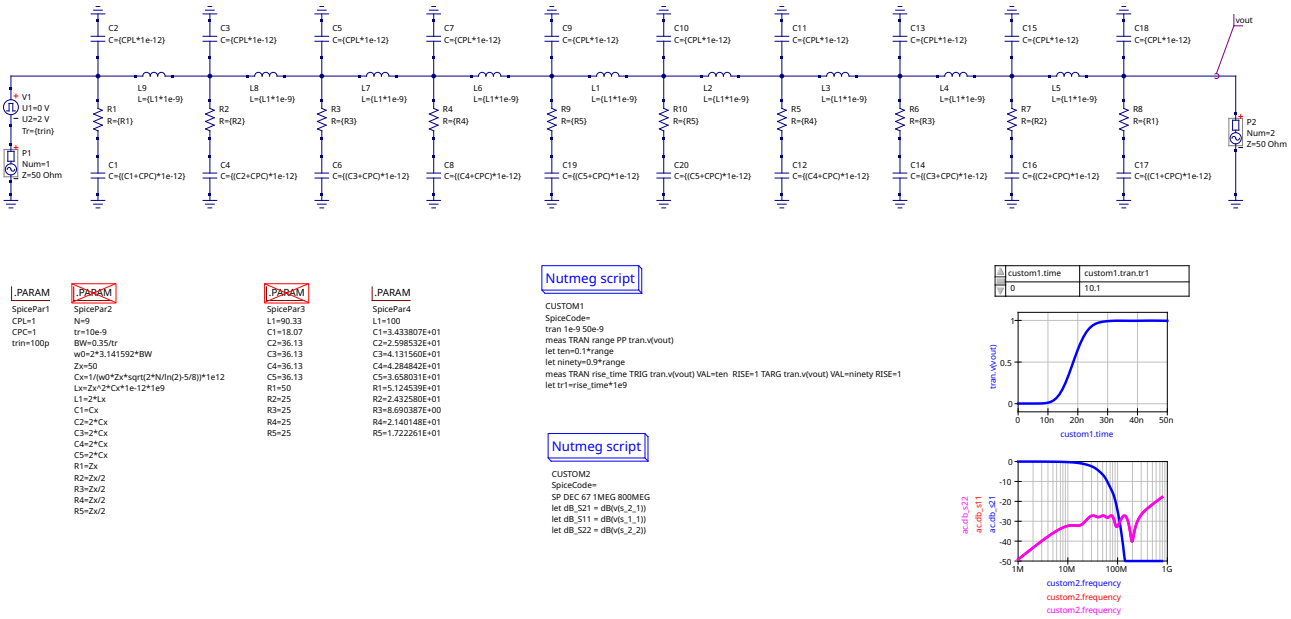

Figure 4: Simulation of the optimized parameters

References

[1] J. Ramos. "ASCO (A SPICE Circuit Optimizer)." (Jan. 19, 2022), [Online]. Available: https://asco.sourceforge.net/.

[2] J. Ramos. "ASCO (A SPICE Circuit Optimizer)," Manual. (), [Online]. Available: https://asco.sourceforge.net/doc/ASCO.pdf.

[3] J. Ramos. "CMOS Operational and RF Power Amplifiers for Mobile Communications," PhD thesis. (Mar. 2005), [Online]. Available: https://asco.sourceforge.net/doc/phd_jramos.pdf.

[4] "Ngspice Documentation," User's manual. (), [Online]. Available: https://ngspice.sourceforge.io/docs.html.

[5] A. Djordjevic, A. Zajic, A. Stekovic, M. Nikolic, Z. Maricevic, and M. Schemmann, "On a class of low-reflection transmission-line quasi-gaussian low-pass filters and their lumped-element approximations," *IEEE Transactions on Microwave Theory and Techniques*, vol. 51, no. 7, pp. 1871–1877, 2003. DOI: 10.1109/TMTT.2003.814310.

[6] J. Breitbarth and D. Schmelzer, "Absorptive near-gaussian low pass filter design with applications in the time and frequency domain," in *2004 IEEE MTT-S International Microwave Symposium Digest (IEEE Cat. No.04CH37535)*, vol. 3, 2004, 1303–1306 Vol.3. DOI: 10.1109/MWSYM.2004.1338805.

[7] "GNU Octave, Scientific Programming Language." (), [Online]. Available: https://octave.org/.

[8] Coilcraft. "Power Inductors, RF Inductors, Transformers, EMI CHokes/Wirewound Ferrite Beads." (), [Online]. Available: https://www.coilcraft.com/.