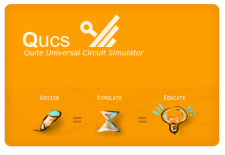
Building a second generation Qucs GPL circuit simulator: package structure, simulation features and compact device modelling capabilities

Mike Brinson*, Centre for Communications Technology, London Metropolitan University, UK, mbrin72043@yahoo.co.uk.
Richard Crozier, The University of Edinburgh, UK, richard.crozier@yahoo.co.uk.
Clemens Novak, Qucs Developer, clemens@familie-novak.net.
Bastien Roucaries, Laboratoire SATIE – CNRS UMR 8929, Université de Cergy-Pontoise, ENS Cachan, FR, bastien.roucaries@satie.ens-cauchan.fr.
Frans Schreuder, Nikhef, Amsterdam, NL, fransschreuder@gmail.com.
Guilherme Brondani Torri, imec / KU Leuven, BE, guitorri@gmail.com.
* Corresponding author.



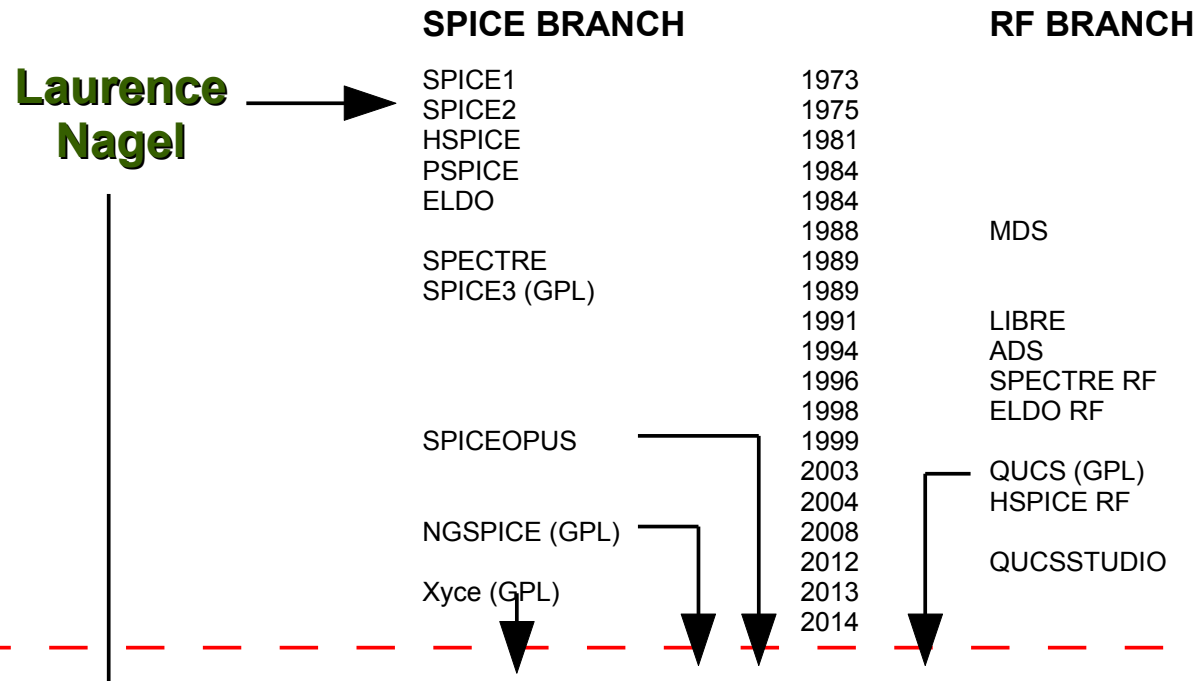
Presented at the MOS-AK Workshop on compact device modelling at London Metropolitan University on March 28 and 29, 2014, London, UK.

Background to presentation

May 2013 –- ten years of Qucs development as a GPL package supporting circuit simulation and compact device modelling

New development team starts work to take Qucs-0.0.16 to an improved level of performance with expanded simulation and modelling facilities

Circuit Simulator development time line

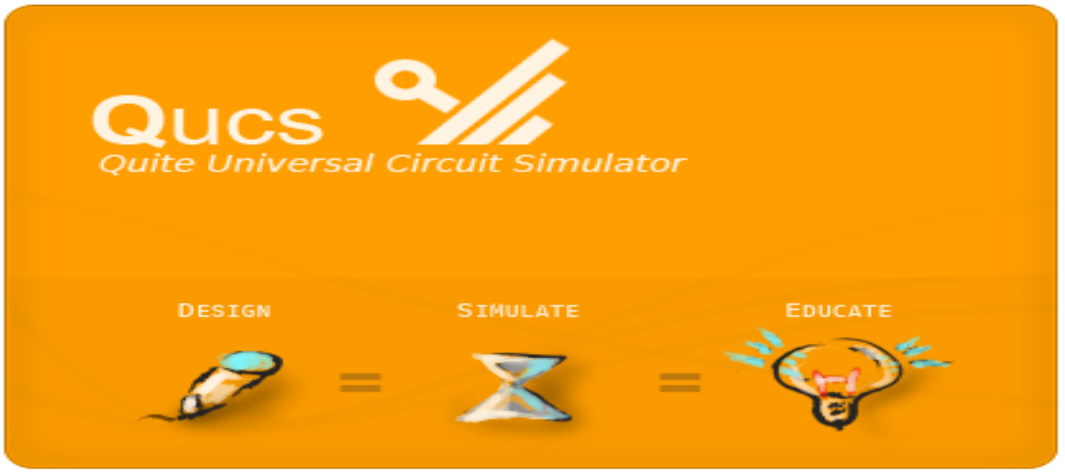


Wladek Grabinski, Mike Brinson, Paolo Nenzi, Francesio Lannutti, Nikolaos Makris, Angelos Antonopoulos and Matthius Bucher. "Open-source circuit simulation tools for RF compact semiconductor device modelling". International Journal of Numerical Modelling: Electrical Networks, Devices and Fields, 2014. Published online in Wiley Online Library. DOI:10.1002/jnm1973.

Next generation GPL circuit simulators

- **Open source tools**
- **Run on popular hardware – PC … Laptop ... Tablet**
- **Employ Verilog-A for emerging technology model development**
- **Include RF circuit simulation**
- **Include statistical circuit analysis**
- **Include thermal circuit analysis**
- **Include system simulation**

Qucs package : 1. Web resources



Revised Web site format and content

Latest News

Latest release: 0.0.17
Source Windows Ubuntu osx
Latest development snapshot: 0.0.18.130703
Windows tar.gz Ubuntu osx

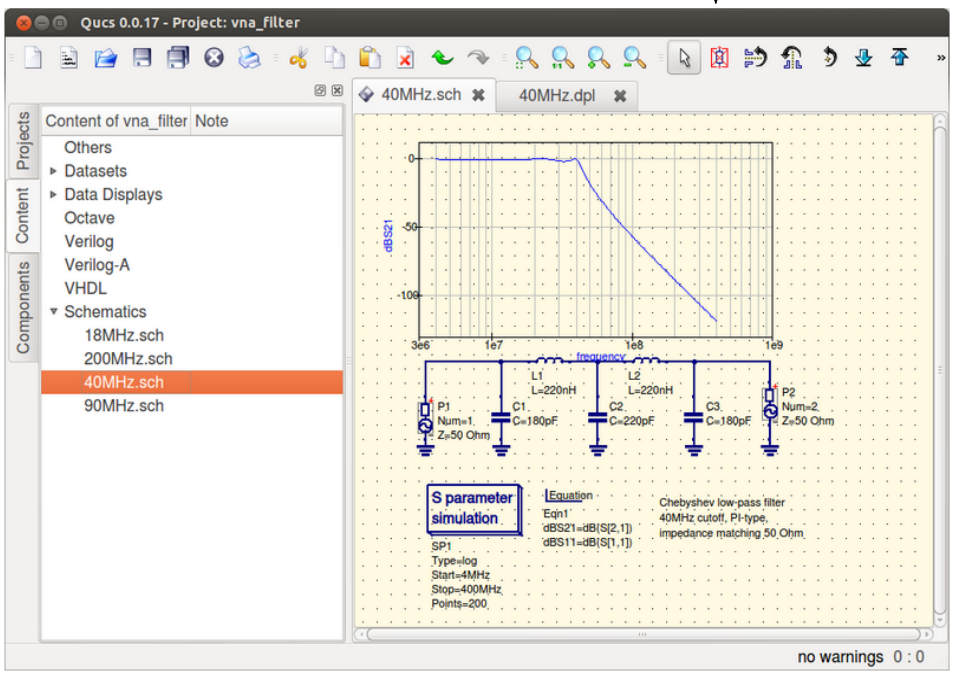
28 November	2013	Created GitHub repository for the Qucs website.
03 July	2013	New implemenation of matrix calculations using Libeigen3. Files are in branch local_complex_20130624
03 July	2013	Added option for changing the home directory and other paths using QSettings (~/.qucs/qucsrc is now obsolete)
23 June	2013	Released qucs 0.0.17!
14 June	2013	Added BSIM 4.30 nMOS and pMOS models
20 May	2013	Added BSIM 3.34 nMOS and pMOS models
26 April	2013	Added beginnings of m-code transient solver interface.

Tutorials
Technical papers
Publications
Build instructions
Install instructions
Contact addresses

Frequently asked questions and answers

Developers names and email addresses

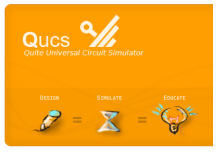
New component specifications



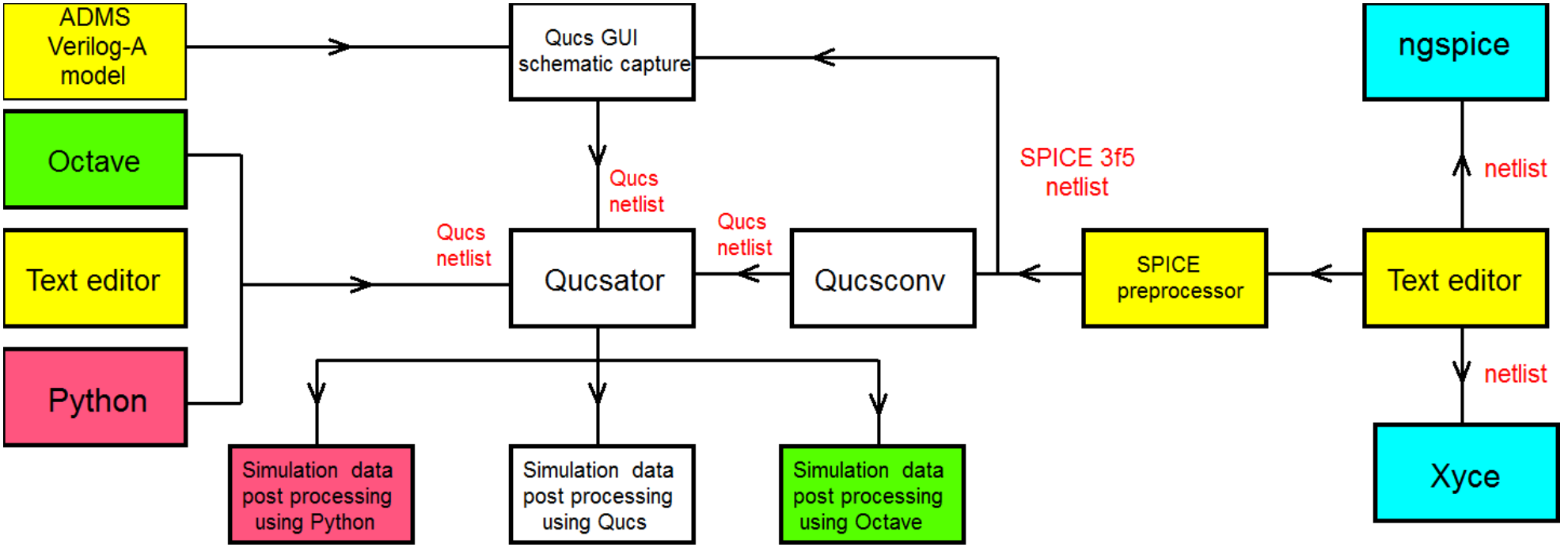
Source code
Official binaries
GIT respository
Unofficial Qucs packages

Regular Development Snapshots posted

Stage 1 - Setup a simple GUI and a simulator.
Stage 2 - Implementation of powerful circuit analysis tools.
Stage 3 - Support for more design- and synthesis tools.
Stage 4 - Implementation of industry standard device models.
Stage 5 - Design realization, production, verification.



Qucs package : 2. Analogue simulator Structure



Qucs package : 3. GUI

Major changes/work since Qucs version 0.0.16:

* Removed Qt3: an ongoing process. Initially, Qucs was a Qt3 application. Qt4 introduced API changes in some areas. To help porting Qt tool kit applications an official aid called qt3to4 was provided with the Qt4 software distribution. However, in the future the Qt3Support library will not be part of the next generation Qt5 tool kit. The porting process requires extensive work to replace Qt3Support with newer C++ classes. An important Qucs mid-term goal is the removal of all Qt3Support.

* The current implementation of the Qucs GUI uses rather low-level drawing routines operating within a complicated software architecture. Some of the current Qt C++ classes are not supported by Qt5, hence a long term issue is to make the Qucs GUI code future proof. One suggestion, currently being worked on, is to use the Qt4 Graphics View Framework.

* Extend QucsConv to work with current and future GPL versions of SPICE, including ngspice and Xyce.

*Added a Qucs Verilog-A dynamic loader: now in latter stages of development and testing. The dynamic loader reuses the symbol loading technique introduced in Qucs version 0.0.16 while introducing a new dynamic code loader. There is no need to recompile Qucs and Qucsator to add new Verilog-A models. Model symbols, and properties for Qucs, as well as compiled modules for qucssator are loaded dynamically during runtime.

* Many bug fixes undertaken in the Qucs GUI code and additions made for improving the quality of schematic drawings.

Qucs package : 4. Analogue simulation engine

Major changes/work since Qucs version 0.0.16:

* Much of the functionality of the Qucsator simulator has been moved to a shared library, with Qucsator now just a thin wrapper for this.

What can users do with the new simulation library?

* Users can load netlists with special circuit elements and perform transient simulations from within their own C++ code.

* Simulations can be performed in either a synchronous, or asynchronous mode. In the first case, transient step size control is done by the user, in the second the simulator solves multiple minor steps between major steps specified by the user.

* Data from probes etc. can be extracted at every time step

* Special circuit elements allow the user to set voltages at each time step, with other elements, such as a controllable current source, switches, resistances etc. are planned

* An interface to the library for the Matlab/Octave language has also been created. This works with the current development version of Octave.

Qucs simulation capabilities : 1. Analogue

Circuit Analysis

Schematic capture >>> Simulation >>> Graphics post processing

Circuit encoding

ngspice and Xyspice netlist

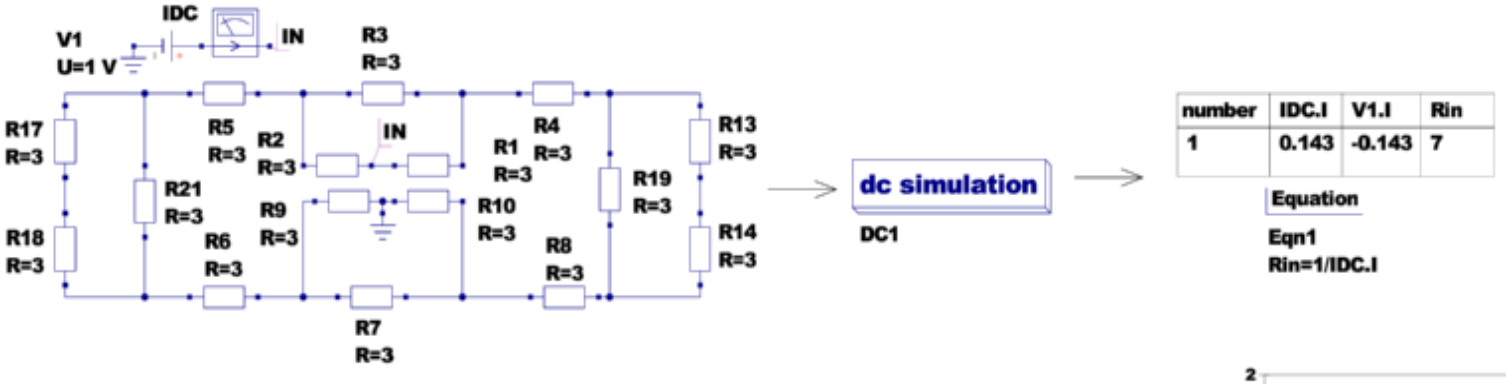
SPICE pre - processor

SPICE 2g6 And 3f5 netlist

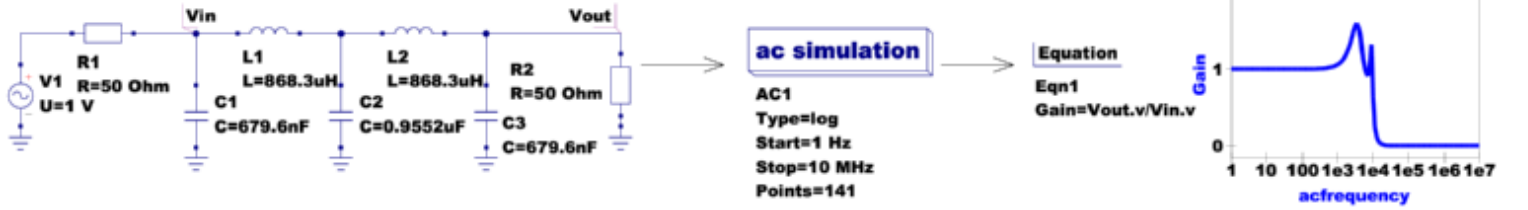
Qucs schematic diagram

Qucs netlist

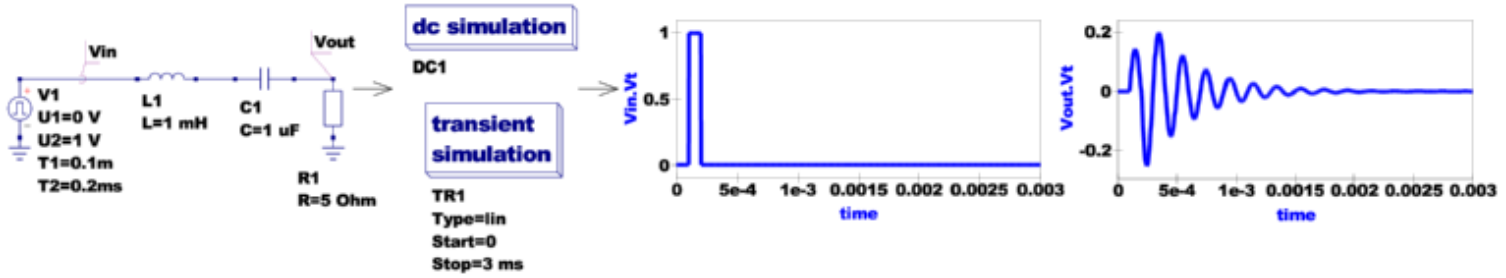
DC



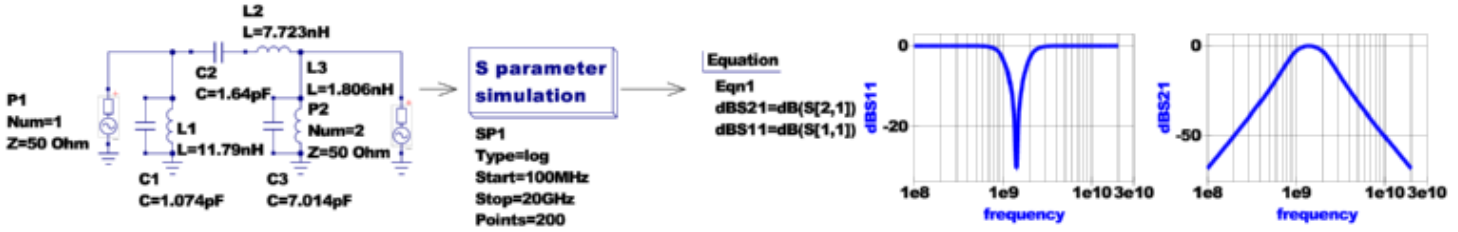
AC and small signal noise



TRAN

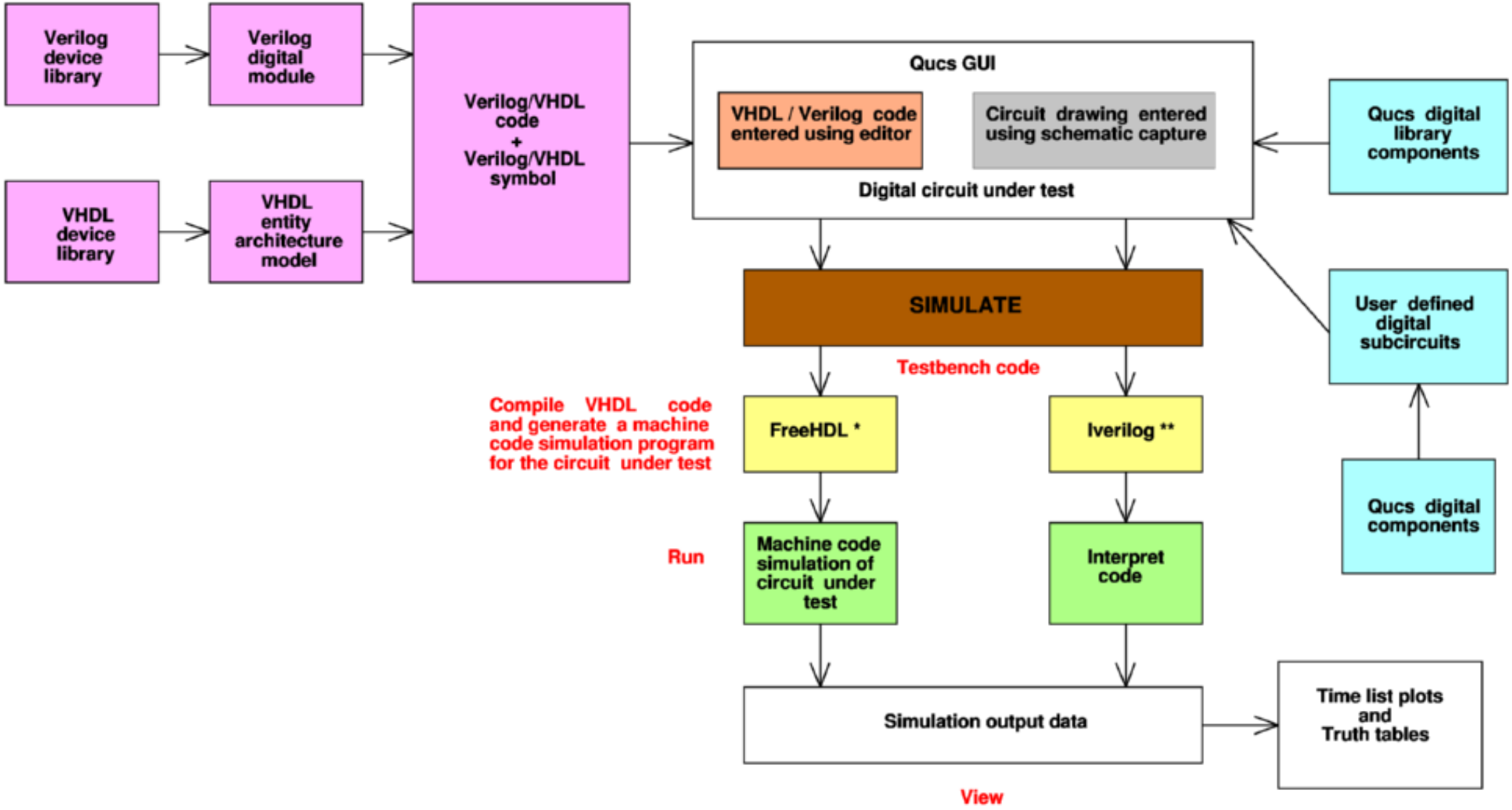


S-parameter* and small signal noise



*Implementation : Qucs built-in; SPICE via RCL networks

Qucs simulation capabilities : 2. Digital



* FreeHDL, http://freehdl.seul.org/

** Icarus Verilog, http://icarus.com/eda/verilog/

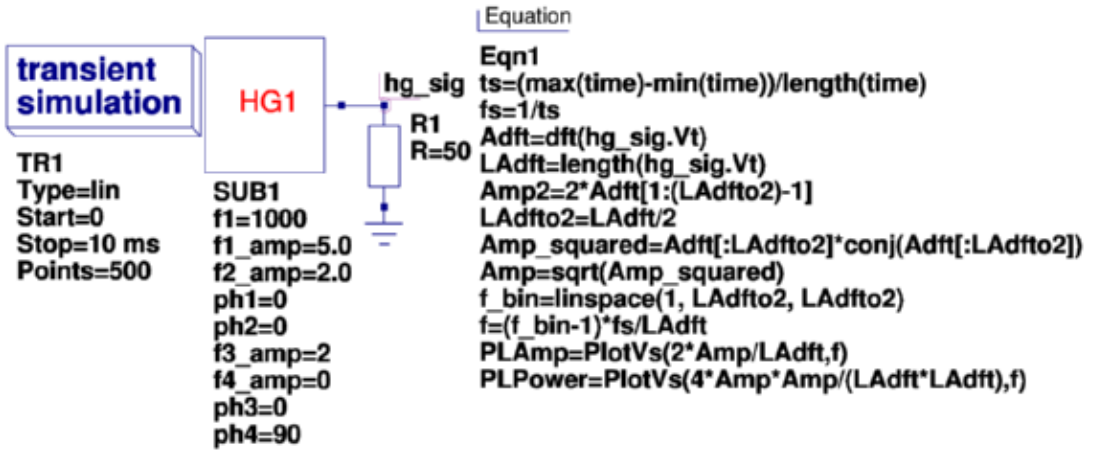
Post-simulation data processing : 1. using Qucs

Equation blocks + simulation data sets → **Data processing** → **Tables and plots**

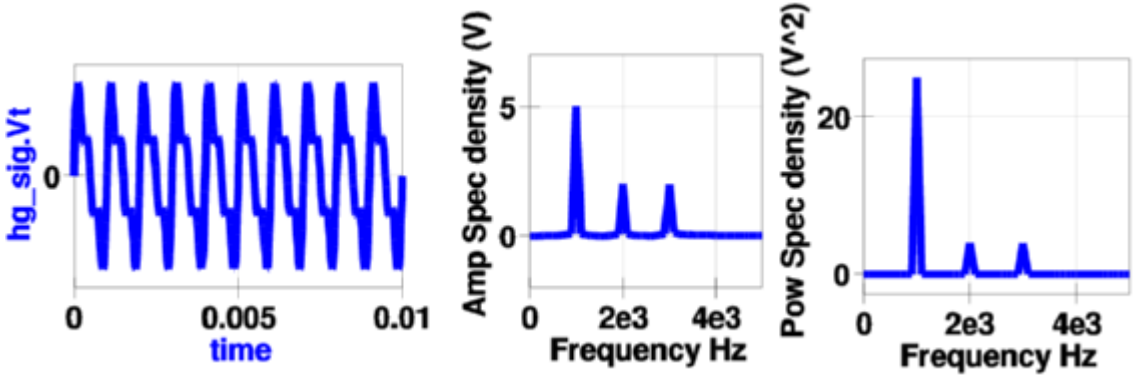
Constants: i, j, pi, e, kB, q
Immediate: 2.5, 1.4+j5.1, [1, 3, 4, 5, 7], [11, 12; 21, 22]
Ranges: Lo:Hi, :Hi, Lo:, :
Logical operators: !x, x&&y, x||y, x^^y, x?y:z, x==y, x!=y, x<y, x<=y,x>y, x>=y

Number suffixes: E, P, T, G, M, k, m, u, n, p, f, a
Matrices: M, M[2,3], M[:,3]
Arithmetic operators: +x, -x, x+y, x-y, x*y, x/y, x%y, x^y

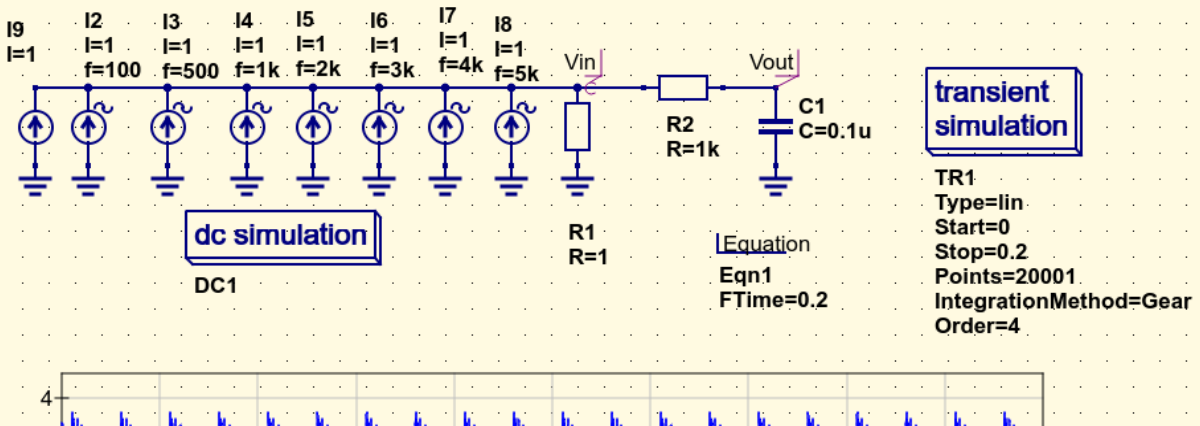
abs adjoint angle arccos arccosec arccot arcosech arcosh arcoth arcsec arcsin arctan arg arsech arsinh artanh avg besseli0 besselj bessely ceil conj cos cosec cosech cosh cot coth cumavg cumprod cumsum dB dbm dbm2w deg2rad det dft diff erf erfc erfcinv erfinv exp eye fft fix floor Freq2Time GaCircle GpCircle hypot idft ifft imag integrate interpolate inverse kbd limexp linspace ln log10 log2 logspace mag max min Mu Mu2 NoiseCircle norm phase PlotVs polar prod rad2deg random real rms Rollet round rtoswr rtoy rtoz runavg sec sech sign sin sinc sinh sqr sqrt srandom StabCircleL StabCircleS StabFactor StabMeasure stddev step stos stoy stoz sum tan tanh Time2Freq transpose twoport unwrap variance vt w2dbm xvalue ytor ytos ytoz yvalue ztor ztos ztoy

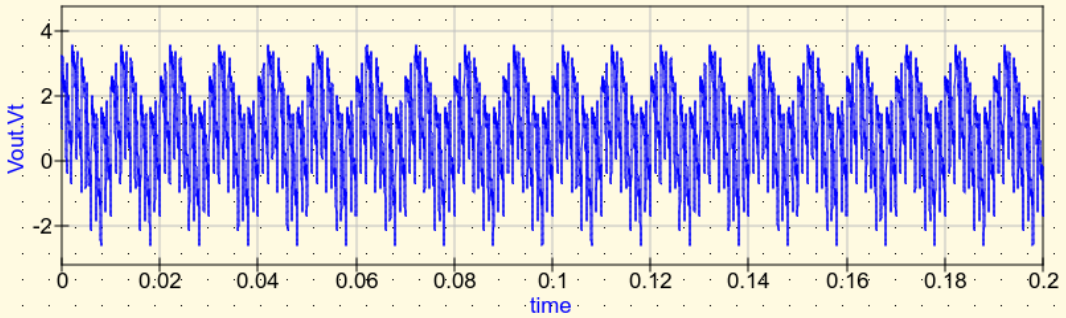


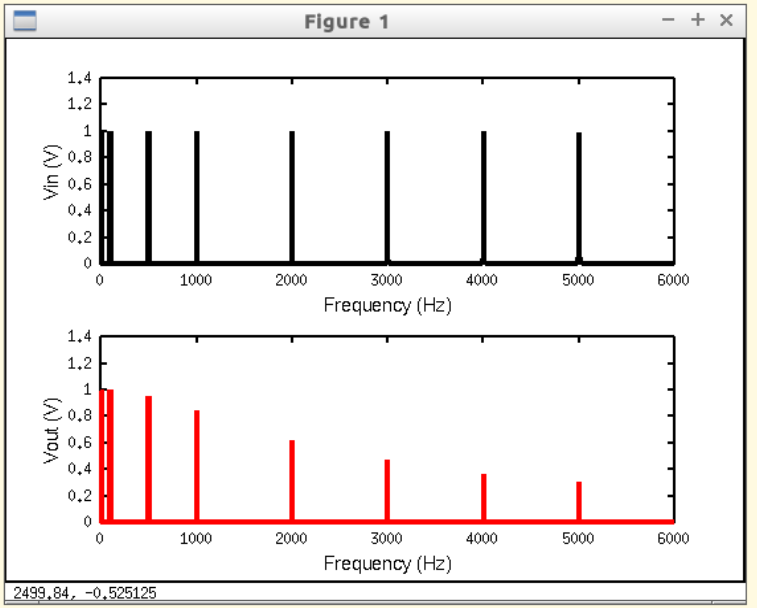
Limitations: NO user defined functions or control loops

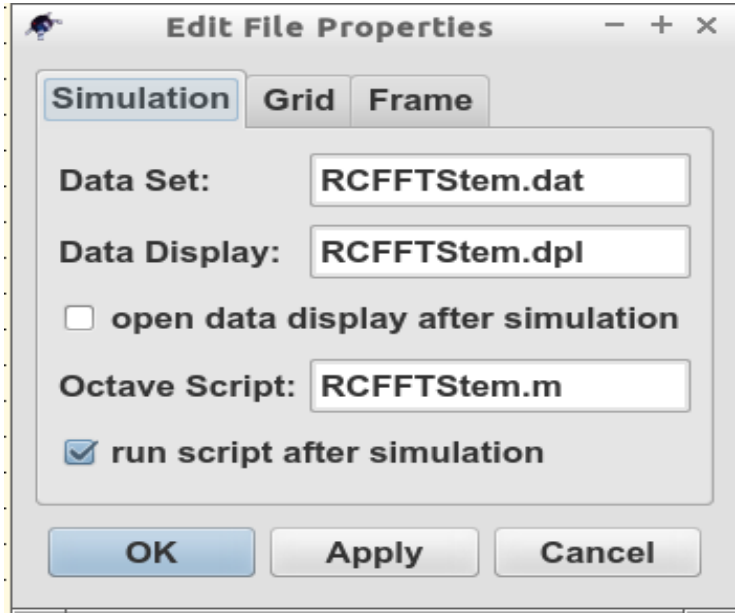


Post-simulation data processing : 2. using Octave









RCFFTStem.m

```
% test RC transient simulation plus FFT of output.
clear;
Data = "RCFFTStem.dat";
qdset = loadQucsDataSet(Data);
[time] = getQucsVariable(qdset,  "time");
[Vin_Vt] = getQucsVariable(qdset,  "Vin.Vt");
[Vout_Vt] = getQucsVariable(qdset,  "Vout.Vt");
[FTime] = getQucsVariable(qdset,  "FTime");
showQucsDataSet(qdset);
[Y1 , Y2, freq] = plotFFT2V( "Stem", "Frequency (Hz)",0,6000,
                             Vin_Vt, "Vin (V)", "black",
                                 Vout_Vt,"Vout (V)", "red", 4, FTime);
```

Post-simulation data processing : 3. using Python

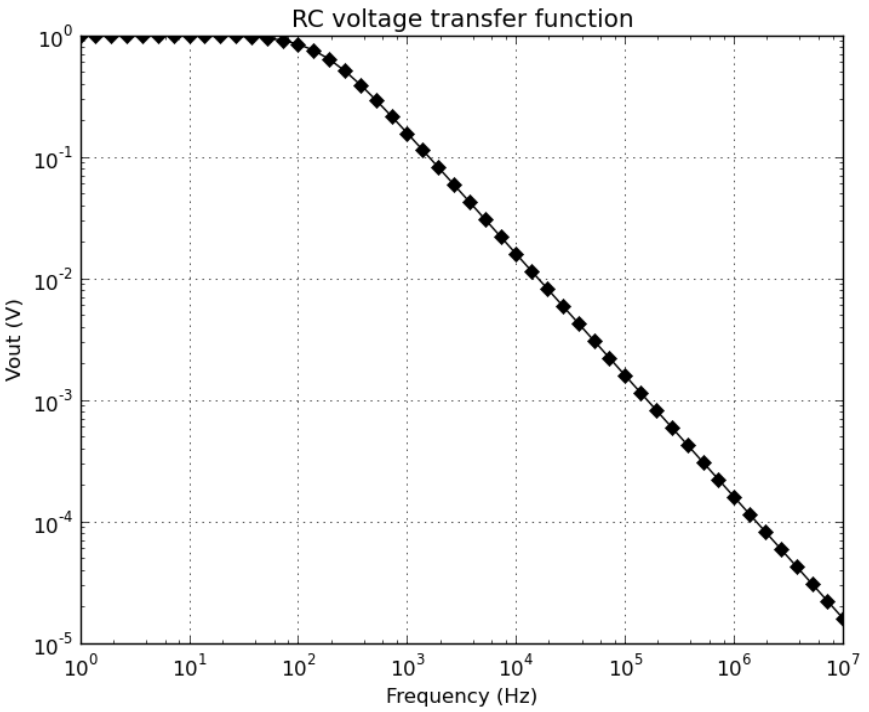
```
#  Basic Python script to demonstrate Qucs simulation with Qucsator
#
import subprocess
import parse_result as pr
import numpy as np
import matplotlib.pyplot as plt
#
from string import Template
#
def runSim():
  netfile = open('RC.net', 'r')
  outfile = open('sim_result.dat', 'w')
  process = subprocess.Popen('qucsator', stdin = netfile, stdout = outfile)
  process.wait()
  netfile.close()
  outfile.close()
#
# Undertake simulation and plot output results
#
runSim()
data = pr.parse_file('sim_result.dat')
x = data['acfrequency']
y = np.abs(data['out.v'])
plt.loglog(x, y, '-kD')
plt.grid()
plt.title('RC voltage transfer function')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Vout (V)')
plt.show()
```

Python script

Qucsator

Parse Qucsator output data in Python format

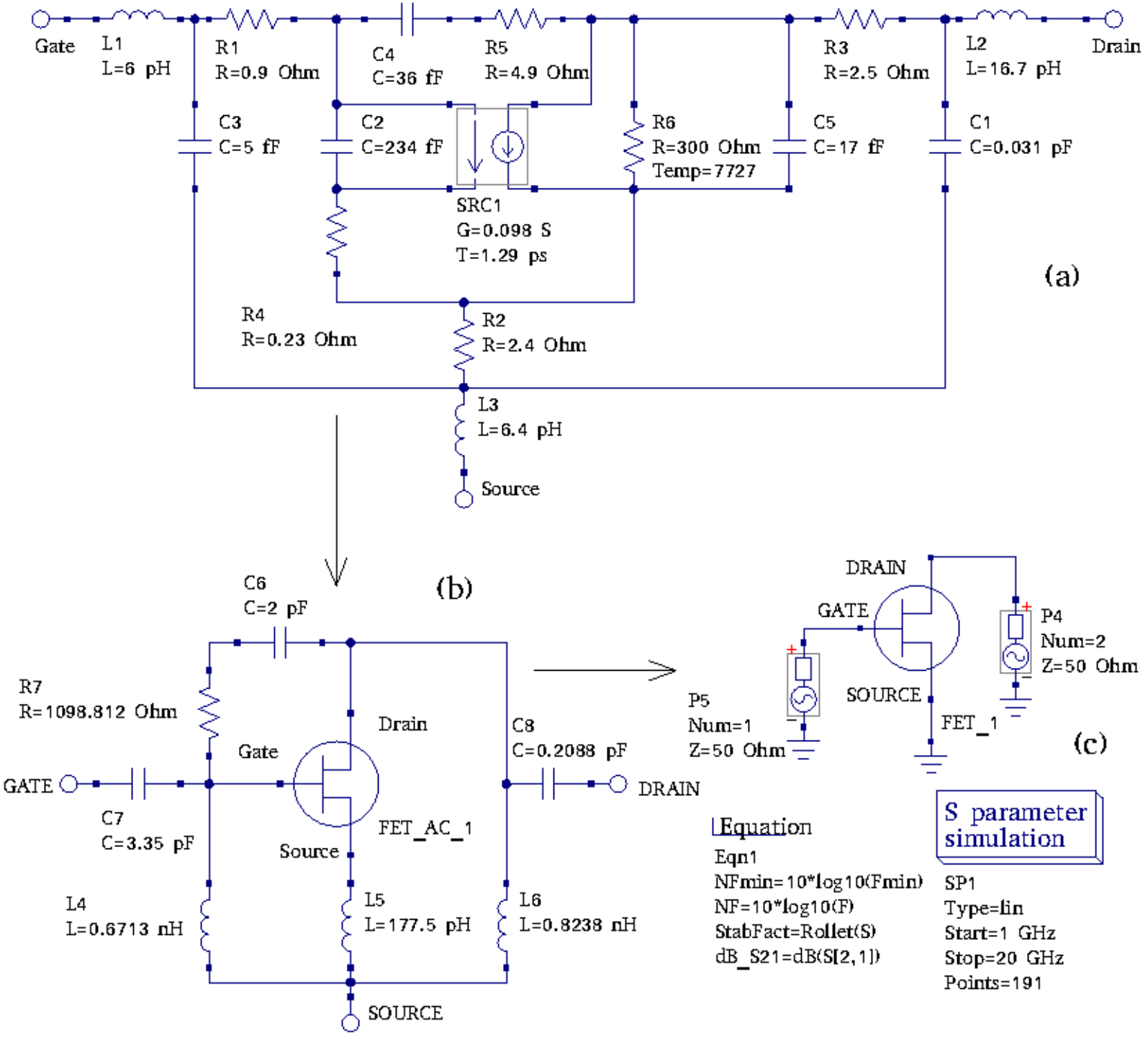
Plot with matplotlib



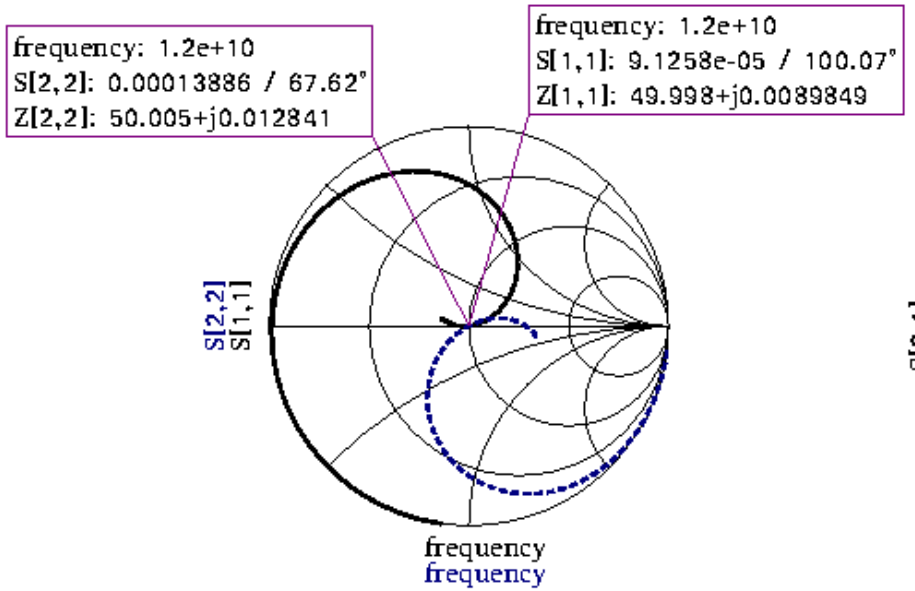

```
# File RC.net
#
Vac:V1 in gnd U = "1V"  f = "1 kHz"
R:R1 out in R = "1 k"
C:C1 out gnd C = "1 u"
.AC:AC1 Start = "1 Hz" Stop = "10 MHz" Points = "50" Type ="log"
```

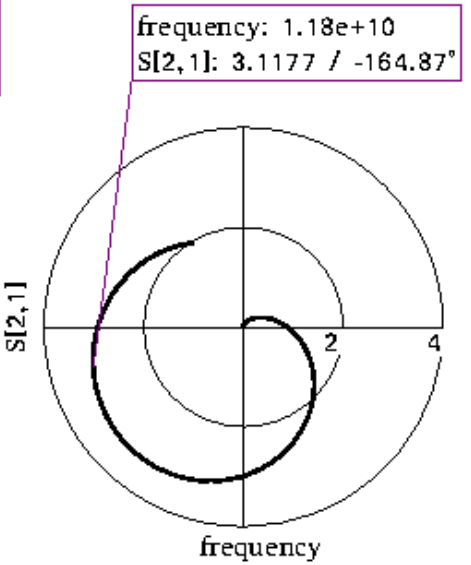
Qucs netlist

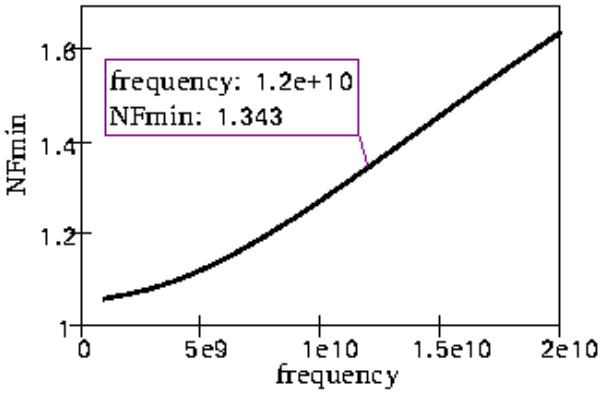
Qucs RF analogue simulation : slide 1

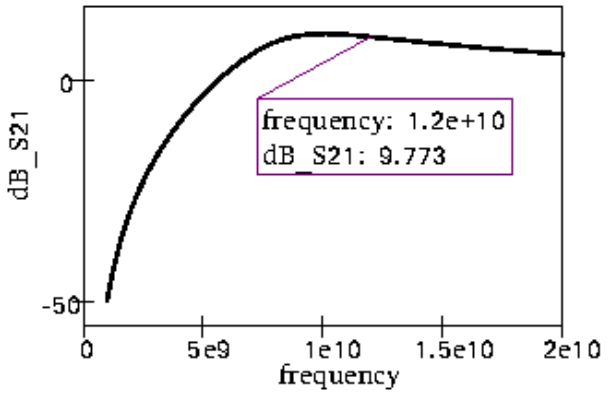


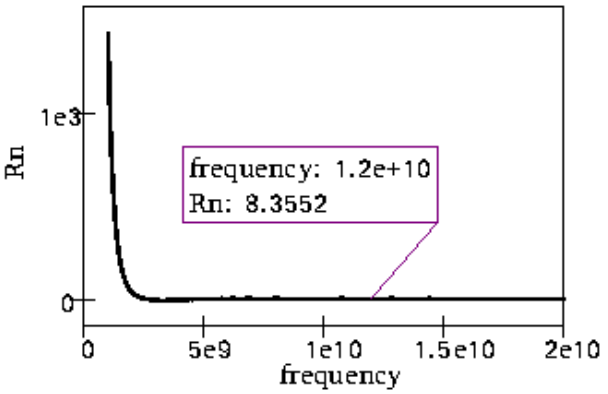
Qucs RF analogue simulation : slide 2

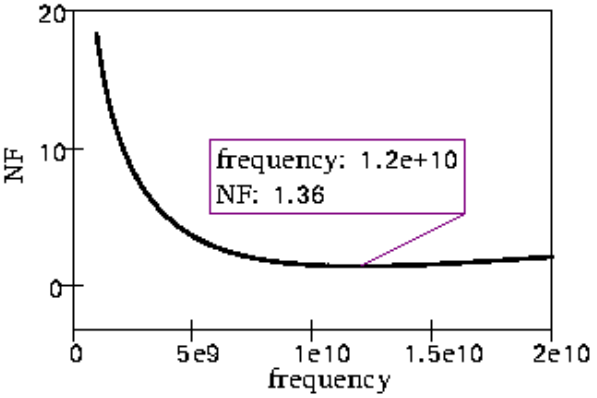






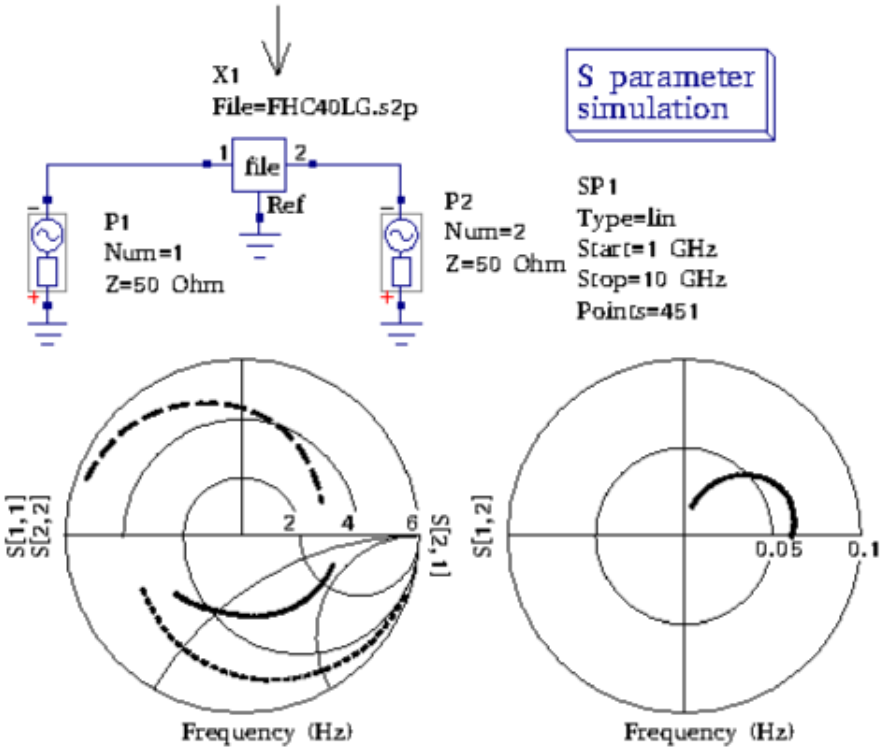




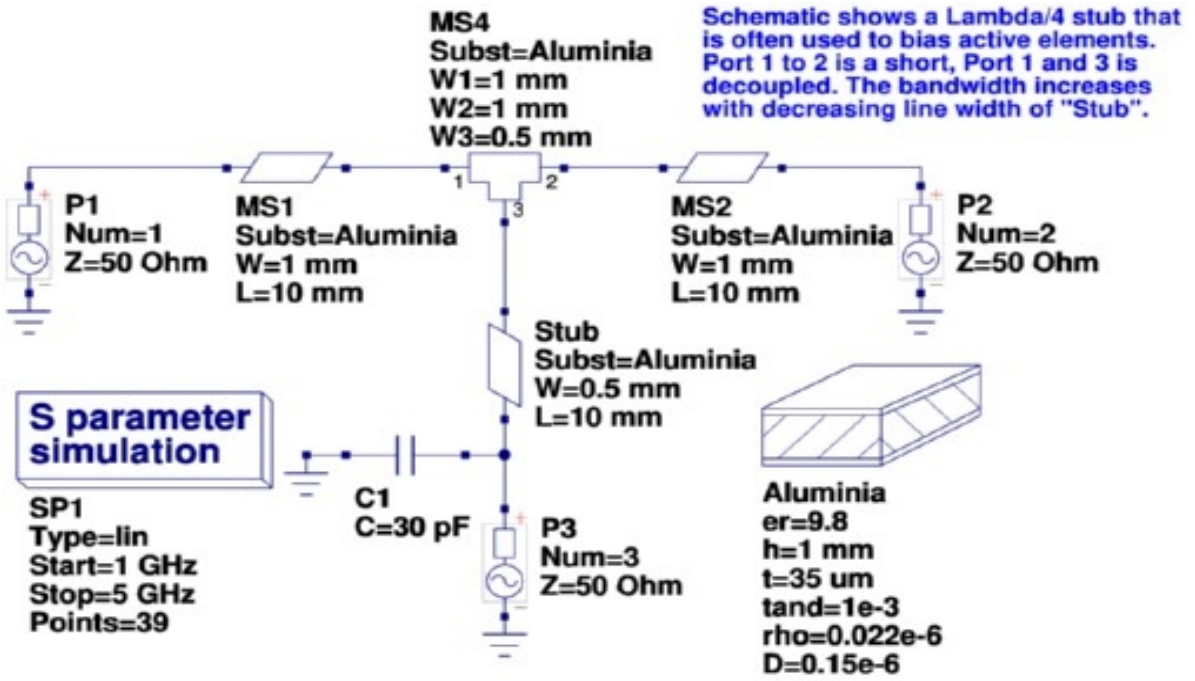


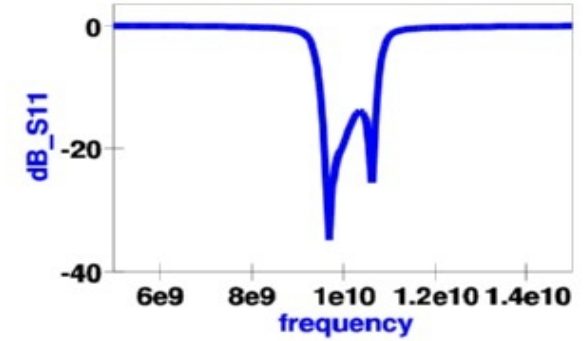
Qucs RF analogue simulation : slide 3

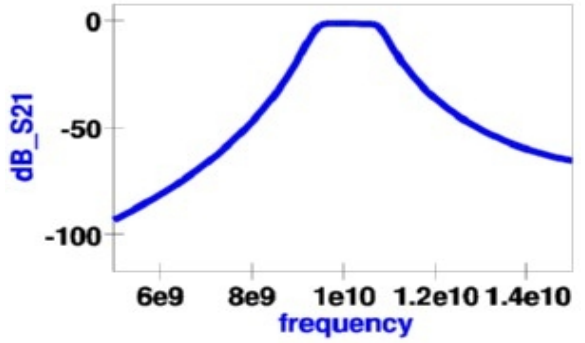
```
! s-parameters of Fujitsu FHC40LG, Vds=2V, Ids=10mA
# GHZ S MA R 50
1.0 0.980 -20.6 5.620 159.7 0.017 75.8 0.541 -17.8
2.0 0.942 -40.7 5.401 140.7 0.033 61.6 0.523 -35.0
3.0 0.887 -59.4 5.051 122.6 0.045 49.5 0.501 -51.2
4.0 0.838 -76.9 4.685 105.8 0.054 38.5 0.480 -66.6
5.0 0.786 -93.2 4.334 89.9 0.060 28.5 0.461 -81.3
6.0 0.742 -108.3 3.984 74.9 0.063 20.2 0.448 -95.4
7.0 0.705 -122.1 3.654 60.6 0.063 12.9 0.449 -108.9
8.0 0.672 -133.7 3.340 47.6 0.063 7.2 0.463 -120.3
9.0 0.651 -143.9 3.110 35.8 0.062 3.2 0.481 -130.1
10.0 0.633 -153.9 2.954 23.7 0.061 -0.2 0.498 -138.8
11.0 0.611 -164.1 2.786 11.8 0.059 -2.9 0.513 -147.6
12.0 0.595 -174.8 2.641 0.0 0.058 -5.1 0.535 -157.0
13.0 0.588 176.0 2.518 -11.6 0.057 -6.7 0.562 -165.3
14.0 0.579 167.6 2.412 -23.0 0.057 -7.9 0.597 -172.8
15.0 0.569 159.3 2.342 -34.6 0.057 -10.1 0.634 -179.7
16.0 0.555 150.5 2.290 -46.6 0.058 -12.9 0.667 173.6
17.0 0.536 140.3 2.272 -59.4 0.059 -17.0 0.697 166.4
18.0 0.525 129.9 2.233 -72.6 0.060 -22.4 0.727 158.8
```

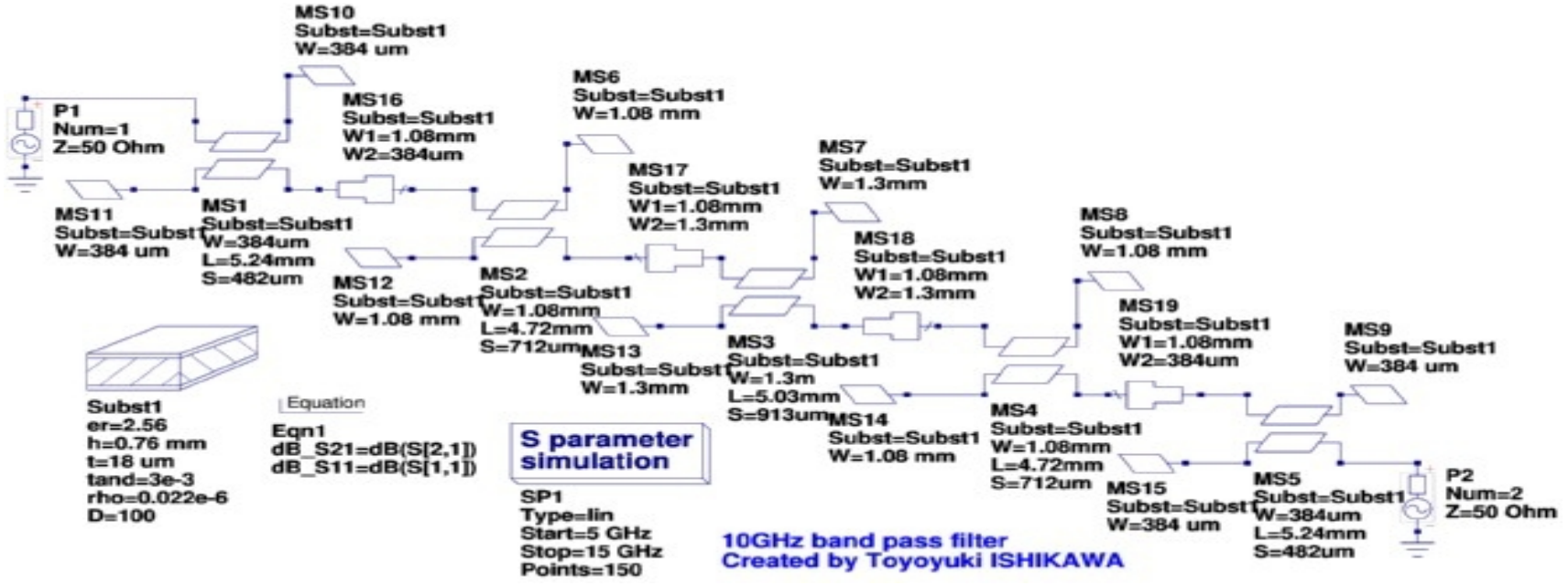


Qucs RF analogue simulation : slide 4





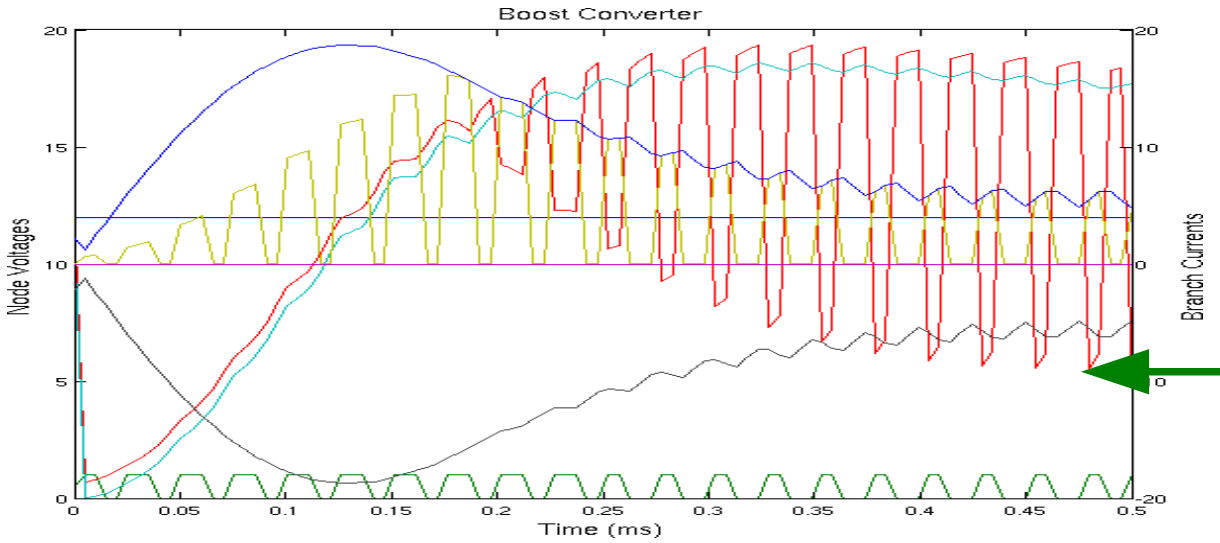




Qucs/Matlab synchronous simulation – an boostconverter example

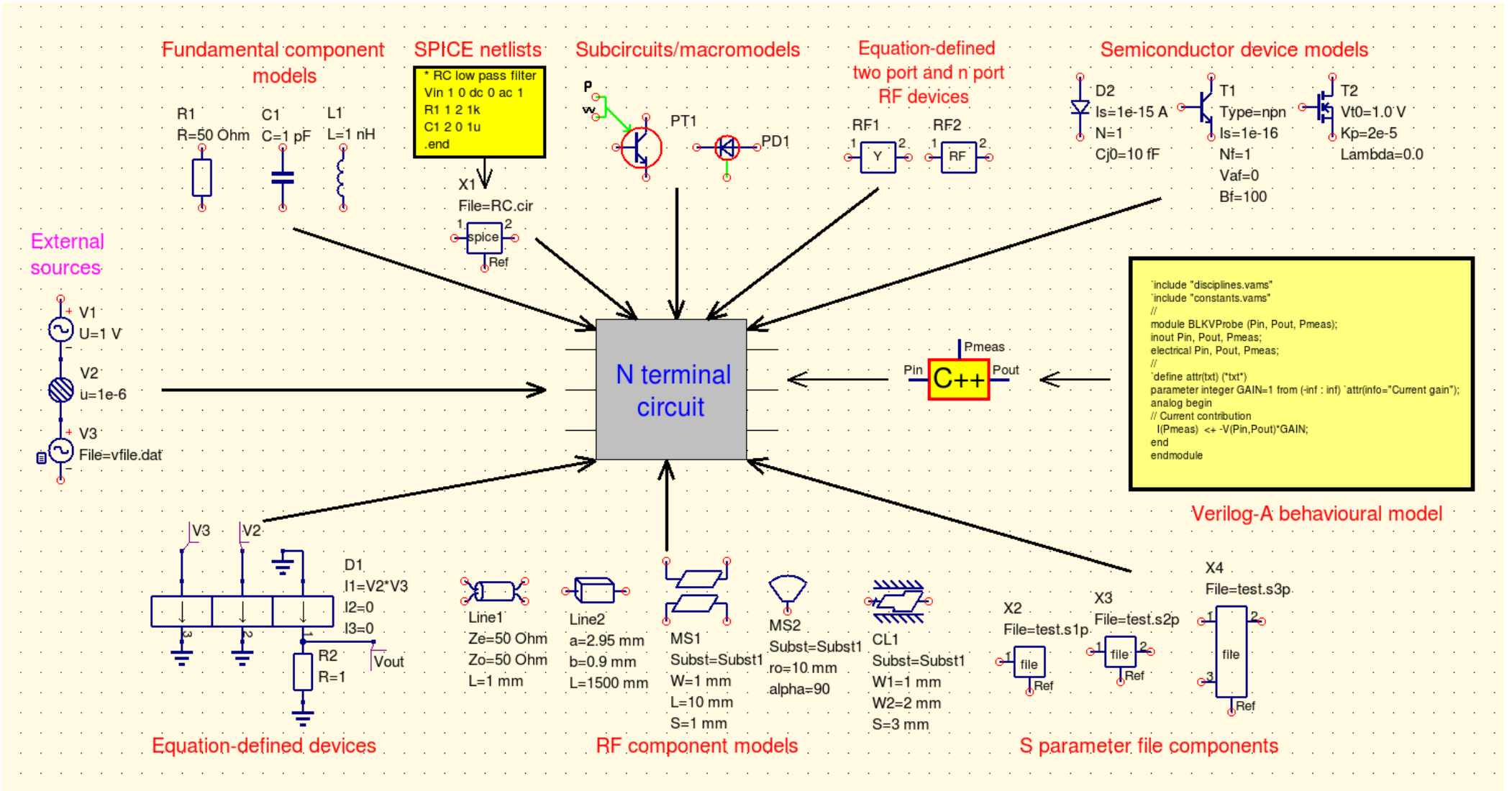
```
# Qucs 0.0.18  C:/Documents and Settings/s0237326/My Documents/Temp/boostconverter.sch
# boostconverter.net
L:L1 _net0 dio L="47uH" I="0"
Vdc:V2 _net0 gnd U="12V"
Eqn:Eqn1 Tmax="Bperiod*20" Tstep="Bperiod/1000" Export="yes"
.ETR:ETR1 IntegrationMethod="Trapezoidal" Order="2" InitialStep="1 ns" MinStep="Tstep"
    MaxIter="150" reltol="0.001" abstol="1 pA" vntol="1 uV" Temp="26.85" LTEreltol="1e-3"
    LTEabstol="1e-6" LTEfactor="1" Solver="CroutLU" relaxTSR="no" initialDC="yes" MaxStep="Tstep"
Eqn:Eqn2 Bfreq="40k" Bperiod="1/Bfreq" Bduty="50" Ton="Bperiod*Bduty/100" Toff="Bperiod-Ton"
Export="yes"
Relais:S1 ctrl dio gnd gnd Vt="0.5 V" Vh="0.1 V" Ron="1" Roff="1e12" Temp="26.85"
Vrect:V1 ctrl gnd U="1V" TH="Ton" TL="Toff" Tr="1 ns" Tf="1 ns" Td="0 ns"
Diode:D1 out dio Is="1e-12 A" N="1" Cj0="10 fF" M="0.5" Vj="0.7 V" Fc="0.5" Cp="0.0 fF" Isr="0.0" Nr="2.0"
   Rs="0.0 Ohm" Tt="0.0 ps" Ikf="0" Kf="0.0" Af="1.0" Ffe="1.0" Bv="0" Ibv="1 mA" Temp="26.85" Xti="3.0"
   Eg="1.11" Tbv="0.0" Trs="0.0" Ttt1="0.0" Ttt2="0.0" Tm1="0.0" Tm2="0.0" Tnom="26.85" Area="1.0"
C:C1 out gnd C="100u" V="0"
R:R1 gnd out R="5" Temp="26.85" Tc1="0.0" Tc2="0.0" Tnom="26.85"
```

Qucs netlist: boostconverter.net




```
% boost_converter_example.m
%
%
cd(fileparts(which('asynchronous_boost_converter_example.m')));
Tstart = 0; n = 100;  tend = 5e-4;
% fixed-step synchonous solver test
clear qtr_async1
% create a new asynchronous solver object from a netlist
qtr_async1 = asynctrcircuit('boostconverter.net');
% initialist the simulation
qtr_async1.init(tstart, (tend - tstart) / (10 * n));
% get the number of nodes
N = qtr_async1.getn;
% get the number of voltage sources
M = qtr_async1.getm;
% choose some time points
T1 = linspace(tstart, tend, n)';
% initialise storage for the solution
Y1 = zeros(numel(T1), M+N);
% get the initial solution
Y1(1, 1:(N+M)) = qtr_async1.getsolution();
% step through time solving the circuit
for ind = 2:numel(T1)
    % accept the step into the solution history
    qtr_async1.acceptstep(T1(ind));
    % get the node voltages and currents at the current time
    Y1(ind, 1:(N+M)) = qtr_async1.getsolution();
end
% plot the node voltages and branch currents
figure;
AX = plotyy(T1 * 1000, Y1(:,1:N), T1 * 1000, Y1(:,((N+1):(N+M))));
title ('Boost Converter', 'FontSize', 14)
xlabel('Time (ms)', 'FontSize', 14)
set(get(AX(1),'Ylabel'),'String','Node Voltages', 'FontSize', 14)
set(get(AX(2),'Ylabel'),'String','Branch Currents', 'FontSize', 14)
```

Qucs model catalogue



Qucs basic device and circuit macromodel modelling tools

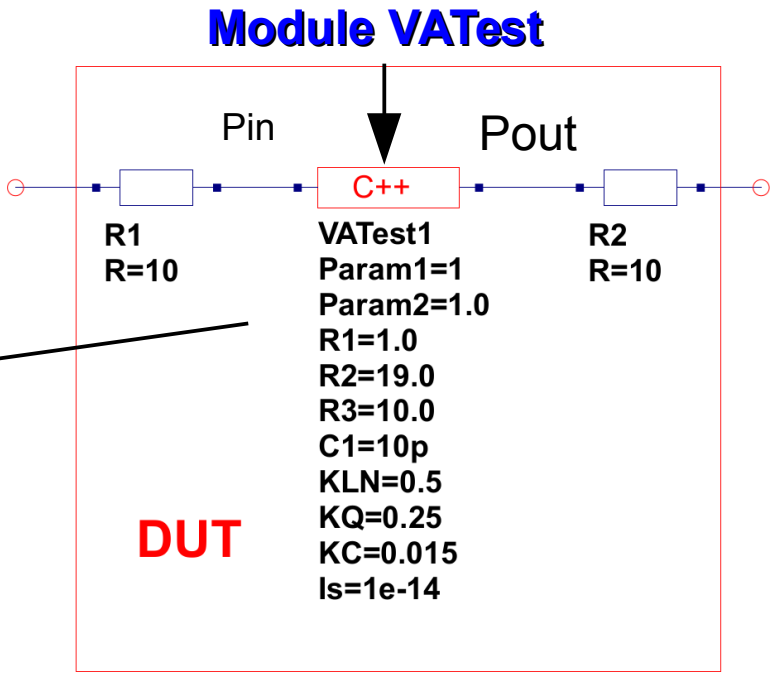
Simulator →	Qucs-0.0.18		ngspice-26		Xyce-6.0		SPICEOPUS-2.31
Features	EDD	Verilog-A[1]	B	Verilog-A[2]	B	Verilog-A[3]	B Verilog-A[4]
Operators	X	X	X	X	X	X	X
Arithmetic functions							
ABS	X	X	X	X	X	X	X
DDT	X	X		X	X	X	
DDX	X	X	X	X	X	X	
IF	X	X	X	X	X	X	
POW	X	X	X	X	X	X	X
RND	X		X		X	?	
SQRT	X	X	X	X	X	X	X
TABLE			X		X		
Exponential, logarithmic and trigonometric functions	X	X	X	X	X	X	X

1. DC, AC, AC noise, S parameters, TRAN and HB.
2. DC, AC and TRAN.
3. DC, AC, TRAN and HB.
4. Not implemented.

Qucs enhanced ADMS/Verilog-A modelling

Qucs/ADMS Verilog-A statement coverage: part 1

```
// VATest - Verilog-A statement test module.
//         This block module has a simple operational electrical circuit which
//         just connects signals between input Pin and output Pout.
//         The nominal series R should be 30 Ohm with small shunt capaciter to ground
//         via internal node n2 and a diode connected between node n1 and ground.
//         See the Verilog-A code for details of the component model and connections.
//         The main  purpose of the module is test the ADMS compiler Verilog-A statement
//         coverage and to identify the C++ code generated by each statement.
//         It should be useful as a test bench in the future when Qucs upgrades to
//         new versions of ADMS, beyond 2.30, and for testing the operation of the Qucs xml
//         files qucsVersion.xml, qucsMODULEcore.xml, qucsMODULEdef.xml, qucsMODULEgui.xml and
//         analogfunction.xml.
//
//   This is free software; you can redistribute it and/or modify
//   it under the terms of the GNU General Public License as published by
//   the Free Software Foundation; either version 2, or (at your option)
//   any later version.
//
//   Copyright (C), Mike Brinson, mbrin72043@yahoo.co.uk, November 2013.
//
// Use of 'include compiler directive.
`include "disciplines.vams"
`include "constants.vams"
// Use of `define compiler directive
`define GMIN 1e-12
//  Verilog-A module statement.
module VATest (Pin, Pout);
inout Pin, Pout;            // Module external interface nodes.
electrical Pin, Pout;
electrical n1, n2;          // Module internal nodes.
//   ground statement
//ground gnd;   // Ground (gnd) statement appears not to work.
// Typical parameter statements
`define attr(txt) (*txt*)
parameter integer Param1 = 1     from [1 : 10]       `attr(info="Integer parameter");
parameter real    Param2 = 1.0123456789   from (-inf : inf)   `attr(info="Real parameter");
parameter real    R1 = 1.0       from [1e-6 : 1e6]   `attr(info="Resistance in the range 1e-6 to 1e6 ohm" unit="Ohm");
parameter real    R2 = 19.0      from [1e-6 : 1e6]   `attr(info="Resistance in the range 1e-6 to 1e6 ohm" unit="Ohm");
parameter real    R3 = 10.0      from [1e-6 : 1e6]   `attr(info="Resistance in the range 1e-6 to 1e6 ohm" unit="Ohm");
parameter real    C1 = 1e-12     from [1e-15 : 1]    `attr(info="Capacitance in the range 1f to 1 F" unit = "F");
parameter real    KLN = 0.1      from (-inf : inf)   `attr(info="Capacitance linear coefficient" unit = "F/V");
parameter real    KQ  = 0.05     from (-inf : inf)   `attr(info="Capacitance quadratic coefficient" unit = "F/(V*V)");
parameter real    KC =  0.0025   from (-inf : inf)   `attr(info="Capacitance cubic coefficient" unit = "F/(V*V*V)");
parameter real    Is = 1e-14     from [1e-20 : 1e-2] `attr(info="Diode saturation current" unit="A");
parameter real    MAXEXP = 40.0  from [1.0  : 100]   `attr(info="Diode exponent maximum coefficient");
parameter integer RemoveZero = 1 from (-inf : inf) exclude 0 `attr(info="Show use of exclude statement");   // Tests Verilog-A exclude statement.
parameter real    KF  = 1e-12    from [0 : INF)      `attr(info="Flicker noise coefficient");
parameter real    AF  = 1.0      from [0 : INF)      `attr(info="Flicker noise exponent");
parameter real    FFE  = 1.0     from [0 : INF)      `attr(info="Flicker noise frequency exponent");
parameter real    Temp  = 26.85  from [-273.15 : 300] `attr(info="Simulation temperature" unit="Celsius");
```



Qucs enhanced ADMS/Verilog-A modelling

Qucs/ADMS Verilog-A statement coverage: part 2

```
//
// Variable definitions
//
real Real1, Real2, Real3;
real NumScaleE, NumScaleP, NumScaleT, NumscaleG, NumscaleM, Numscalek, NumScaleh, NumScaleD, NumScaled;
real NumScalec, NumScalem, NumScaleu, Numscalen, NumscaleA, Numscalep, NumScalef, NumScalea;
integer Int1, Int2, Int3, Pcount;
integer SwitchValue;
real Con1, Con2, Con3, Con4, Con5, Con6, Con7, Con8, Con9, Con10;
real Con11, Con12, Con13, Con14;
real Pcon1, Pcon2, Pcon3, Pcon4, Pcon5, Pcon6, Pcon7;
real TempK, DEL, P5, x, xdiff, Id;
real Q1, Ceff, Gdiode, Rh1, Ch1;
real P1, P2, P3, P4, z, y;
real Fourkt, TwoQ;
string Hash1, Hash2, Hash3, Hash4;
//
//  Definition of user defined sinc function
//
    analog function real sinc;
      input arg;
      real  arg;
      begin
       if (arg != 0.0)
          sinc = sin(arg)/arg;
       else
          sinc = 1.0;
      end
    endfunction

//
//  Define branches.
//
branch (n1) bn1;
branch (n1, n2) bn1n2;
//
// START OF ANALOG BLOCK
//
analog begin
//
// Use of `ifdef, `undif, `else and `endif compiler directives.
//
`define Switch 1
`ifdef Switch
     SwitchValue = 1;
`else
     SwitchValue = 0;
`endif
```

Qucs enhanced ADMS/Verilog-A modelling

Qucs/ADMS Verilog-A statement coverage: part 3

```
//
$strobe("=============================================================================================================");
$strobe("In analog block.");
$strobe("Test 3: SwitchValue = %d ", SwitchValue);

// Use of `undef compiler directive
//
`undef Switch
`ifdef Switch
    SwitchValue = 1;
`else
    SwitchValue = 0;
`endif
$strobe("Test 4: SwitchValue = %d ", SwitchValue);
$strobe("=============================================================================================================");

@(initial_model)
begin
//
//
//    $strobe statements output test results to simulation log.
//    Data is listed under "errors" section of log.
//
      $strobe("Verilog-A statement output test data follows.");              // Initialize variables using @(initial_model) statement.
//
//Set strings.
//
Hash1 = "#############################################################################################################";
Hash2 = ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>";
Hash3 = "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++";
Hash4 = "*************************************************************************************************************************************";
//
// Set reals - scientific notation.
//
   Real1 = 1.0; Real2 = 2.0; Real3 = 3.0;
//
// Set integers.
//
   Int1  = 1;   Int2  = 2;   Int3  = 3;
//
// Set reals - engineering notation.
//
    NumScaleE = 1E;  NumScaleP = 2P;    NumScaleT = 3P;  NumscaleG = 4G;  NumscaleM = 5M;
    Numscalek =  6k;   NumScaleh = 7h;    NumScaleD = 8D;  NumScaled = 9d;
    NumScalec = 10c; NumScalem = 11m; NumScaleu = 12u; Numscalen = 13n;
    NumscaleA = 14A; Numscalep =  15p;  NumScalef = 16p;  NumScalea = 17a;
```

Qucs enhanced ADMS/Verilog-A modelling

Qucs/ADMS Verilog-A statement coverage: part 4

```
// Set Verilog-A mathematical constants.
//
   Con1 =  `M_PI;    Con2  = `M_TWO_PI;  Con3  = `M_PI_2;     Con4  = `M_PI_4;
   Con5 =  `M_1_PI;  Con6  = `M_2_PI;    Con7  = `M_2_SQRTPI; Con8  = `M_E;
   Con9 =  `M_LOG2E; Con10 = `M_LOG10E;  Con11 = `M_LN2;      Con12 = `M_LN10;
   Con13 = `M_SQRT2; Con14 = `M_SQRT1_2;
   $strobe("%s", Hash1);
   $strobe("Start @(initial_model) statement.");
//
   $strobe("Test 1: `M_PI    = %g,  `M_TWO_PI  = %g , `M_PI_2     = %g ,      `M_PI_4 = %g ",  Con1, Con2,  Con3,  Con4);
   $strobe("        `M_1_PI  = %g,    `M_2_PI    = %g ,     `M_2_SQRTPI = %g , `M_E    = %g ",  Con5, Con6,  Con7,  Con8);
   $strobe("        `M_LOG2E = %g, `M_LOG10E  = %g , `M_LN2      = %g , `M_LN10 = %g ",  Con9, Con10, Con11, Con12);
   $strobe("        `M_SQRT2 = %g, `M_SQRT1_2 = %g ", Con1, Con2);
//
// Set Verilog-A physical constants.
//
    Pcon1 = `P_Q;     Pcon2 = `P_C;   Pcon3 = `P_K; Pcon4 = `P_H;
    Pcon5 = `P_EPS0;  Pcon6 = `P_U0;  Pcon7 = `P_CELSIUS0;
    $strobe("Test 2: `P_Q     = %g, `P_C  = %g , `P_K       = %g , `P_H = %g ",  Pcon1, Pcon2,  Pcon3,  Pcon4);
    $strobe("        `P_EPS0  = %g, `P_U0 = %g , `P_CELSIUS0 = %g ",              Pcon5, Pcon6, Pcon7);

//
// Diode model I/V and noise characteristic constants
//
    P5 = Is*exp(MAXEXP);
    TempK = Temp+`P_CELSIUS0;
    DEL  = `P_Q/(`P_K*TempK);
    Fourkt = 4.0*`P_K*TempK;
    TwoQ   = 2.0*`P_Q;
//
    $strobe("End @(initial_model) statement.");
    $strobe("%s", Hash1);
//
//  Test of case statement
//
    Pcount = 2;
    case (Pcount)
      0:  z = 0.0;
      1:  z = 1.0;
      2:  z = 2.0;
      3:  z = 3.0;
      4:  z = 4.0;
      default: z = 99.0;
    endcase
    $strobe("Case statement z = %10.6g", z);

end
```

Qucs enhanced ADMS/Verilog-A modelling

Qucs/ADMS Verilog-A statement coverage: part 5

```
//  Dummy @(initial_step) statement.
//
  @(initial_step) begin
    $strobe("%s", Hash2);
    $strobe("Start @(initial_step) statement.");
//
//  Test for statement.
    $strobe("Test 7: for loop. ");
    Rh1 = 10k; Ch1 = 1u;
    for (Pcount = 1; Pcount < 11; Pcount = Pcount+1) begin
      Rh1 = Rh1+ 1k;
      Ch1 = Ch1+ 1u;
      $strobe("Rh1 = %g, Ch1 = %g ", Rh1, Ch1);
      $strobe("Pcount = %d ", Pcount);
    end
    $strobe("_____________________________________________________________________________________________________");
//
//  Test while loop.
    $strobe("Test 8: while loop. ");
    Rh1 = 20k; Ch1 = 2u;
    Pcount = 1;
    while (Pcount < 11) begin
      Rh1 = Rh1+ 2k;
      Ch1 = Ch1+ 2u;
      $strobe("Rh1 = %g, Ch1 = %g ", Rh1, Ch1);
      $strobe("Pcount = %d ", Pcount);
      Pcount = Pcount+1;
    end
    $strobe("_____________________________________________________________________________________________________");
//
//  Test if statement, $given.
    $strobe("Test 10: if statement and $given system function");
    if ($given(Param1))
      begin
        $strobe("         Param1 confirmed by $given, value = %d ",Param1 );
      end
    else
      begin
        $strobe("         Param1 not found by $given system function");
      end
    $strobe("_____________________________________________________________________________________________________");
//
//  Test if statement, $param_given.
    $strobe("Test 11: if statement and $param_given system function");
    if ($param_given(Param2))
      begin
        $strobe("         Param2 confirmed by $param_given, value = %10.6g ", Param2);
      end
    else
      begin
        $strobe("         Param2 not found by $param_given system function");
      end
    $strobe("_____________________________________________________________________________________________________");
//
```

Qucs enhanced ADMS/Verilog-A modelling

Qucs/ADMS Verilog-A statement coverage: part 6

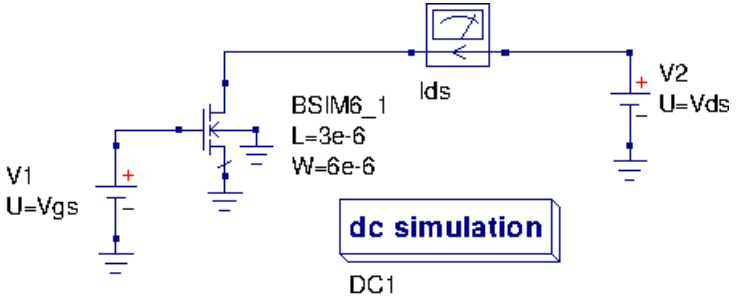
```
//
//  Test Verilog_a buit-in functions.
//
    P1 = ln(1.0);       P2 = log(1.0);      P3 = exp(1.0);  P4 = sqrt(2.0);
    $strobe("Test 5:  ln(1.0) = %10.6g,      log(1.0) = %10.6g,   exp(1.0) = %10.6g,  sqrt(2.0) = %10.6g ",P1, P2, P3, P4);
    P1 = min(1.0, 2.0); P2 = max(1.0, 2.0); P3 = abs(-2.0); P4 = floor(2.0);
    $strobe("   min(1.0, 2.0) = %10.6g, max(1.0, 2.0) = %10.6g,  abs(-2.0) = %10.6g, floor(2.0) = %10.6g ",P1, P2, P3, P4);
                    P2 = pow(2.0, 4.0); P3 = sin(1.0);  P4 = cos(1.0);                                  // Function ceil NOT implemented!
    $strobe("                          pow(2.0, 4.0) = %10.6g,   sin(1.0) = %10.6g,   cos(1.0) = %10.6g ", P2, P3, P4);
    P1 = tan(1.0);      P2 = asin(1.0);     P3 = acos(1.0); P4 = atan(2.0);
    $strobe("       tan(1.0) = %10.6g,     asin(1.0) = %10.6g,  acos(1.0) = %10.6g,  atan(2.0) = %10.6g ",P1, P2, P3, P4);
                    P2 = hypot(2.0, 2.0); P3 = sinh(1.0); P4 = cosh(2.0);                                  // Function atan2 NOT implemented!
    $strobe("                         hypot(2.0, 2.0) = %10.6g,  sinh(1.0) = %10.6g,  cosh(2.0) = %10.6g ", P2, P3, P4);
    P1 = tanh(1.0);         P2 = atanh(0.2);  P3 = $vt(27);  P4 = $temperature;
    $strobe("      tanh(1.0) = %10.6g,    atanh(0.2) = %10.6g,    $vt(27) = %10.6g $temperature = %10.6g",P1, P2, P3, P4);  // Functions asinh and acosh NOT implemented!
    $strobe("End @(initial_step) statement.");
    $strobe("%s", Hash2);

  end                                                                   // End of @(initial_step) statement.
//
//  Dummy @(initial_instance) statement.
//
  @(initial_instance) begin
    $strobe("%s", Hash3);
    $strobe("Start @(initial_instance) statement.");
//
//  Test user defined sinc function.
//
    $strobe("Test 9: Test of user defined sinc function. ");
    for (Pcount = 0; Pcount < 11; Pcount = Pcount+1)
      begin
        z = Pcount;
        y = sinc(z);
        $strobe("x = %10.6g , y = %10.6g ", z, y);
    end
    $strobe("End @(initial_instance) statement.");
    $strobe("%s", Hash3);
  end                                                                   // End of @(initial_instance) statement.
```

Qucs enhanced ADMS/Verilog-A modelling

Qucs/ADMS Verilog-A statement coverage: part 7

```
//
// Dummy @(final_step) statement.
//
  @(final_step) begin
    $strobe("%s", Hash4);
    $strobe("Test 7: Inside @(final_step) statement.");
    $strobe("%s", Hash4);
  end          // End of @(final_step) statement.
//
//
// Basic semiconductor diode I/V model, capacitance Ceff model and resistance current contributions,
// plus noise.
//
 x = DEL*V(n1);
 xdiff = x-MAXEXP;
 I(Pin, n1)    <+  V(Pin, n1)/R1;
 I(Pin, n1)    <+  white_noise(Fourkt/R1, "thermal");
 I(bn1)        <+  (x > MAXEXP) ? P5*(1.0+xdiff*(1.0+xdiff*(0.5+xdiff/6))) : Is*(exp(x)-1.0);
 I(bn1)        <+  white_noise(TwoQ*((x > MAXEXP) ? P5*(1.0+xdiff*(1.0+xdiff*(0.5+xdiff/6))) : Is*(exp(x)-1.0)), "shot");
 I(bn1)        <+  V(bn1)*`GMIN;    // Diode junction parallel R = 1/GMIN.
 I(bn1)        <+  flicker_noise(KF*pow((x > MAXEXP) ? P5*(1.0+xdiff*(1.0+xdiff*(0.5+xdiff/6))) : Is*(exp(x)-1.0), AF), FFE, "flicker");
 I(bn1n2)      <+  V(bn1n2)/R2;
 I(bn1n2)      <+  white_noise(Fourkt/R2, "thermal");
 I(n2)         <+  ddt((C1*(1.0+V(n2)*(KLN+V(n2)*(KQ+KC*V(n2)))))*V(n2));
 I(n2, Pout)   <+  V(n2, Pout)/R3;
 I(n2, Pout)   <+  white_noise(Fourkt/R3, "thermal");
//  Diode conductance.
 Gdiode  = ddx((x > MAXEXP) ? P5*(1.0+xdiff*(1.0+xdiff*(0.5+xdiff/6))) : Is*(exp(x)-1.0), V(n1));
// Ceff is the effective capacitance of component C1.
 Ceff = ddx((C1*(1.0+V(n2)*(KLN+V(n2)*(KQ+KC*V(n2)))))*V(n2), V(n2));
 $strobe("Test 6 : Effectve capacitance Ceff = %g, Diode conductance Gd = %g ", Ceff, Gdiode);
//
//  Test ground statement.
//
// Id  = Gdiode*V(n1,gnd);                    // Ground (gnd) statement appears not to work.
// $strobe("Id = %10.6g ", Id);
end
endmodule
```

Qucs enhanced ADMS/Verilog-A modelling: Part 2; an experimental implementation of BSIM6 Verilog-A code compiled with the Qucs version of ADMS 2.30

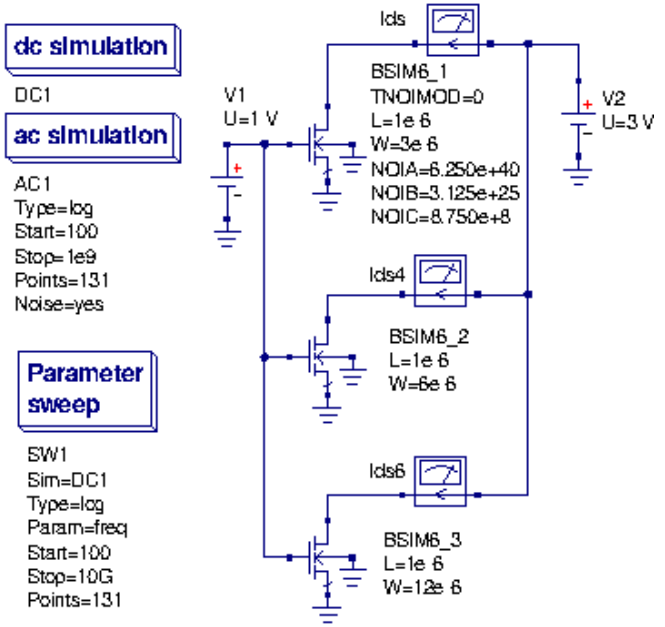


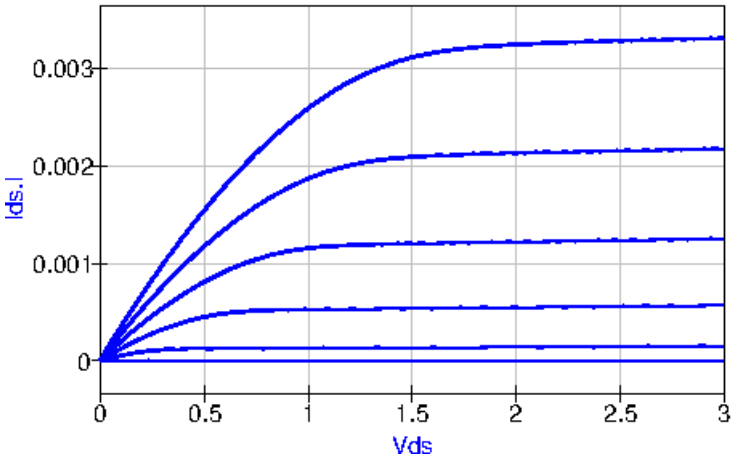
Parameter sweep

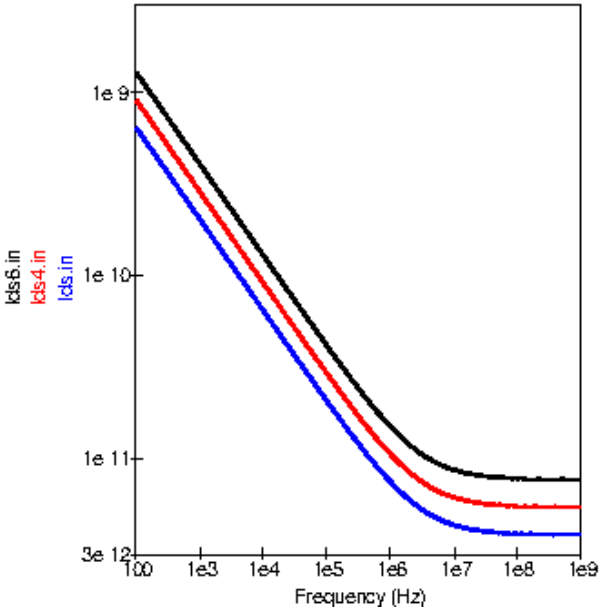
SW1
Sim=SW2
Type=lin
Param=Vgs
Start=0
Stop=3
Points=7

Parameter sweep

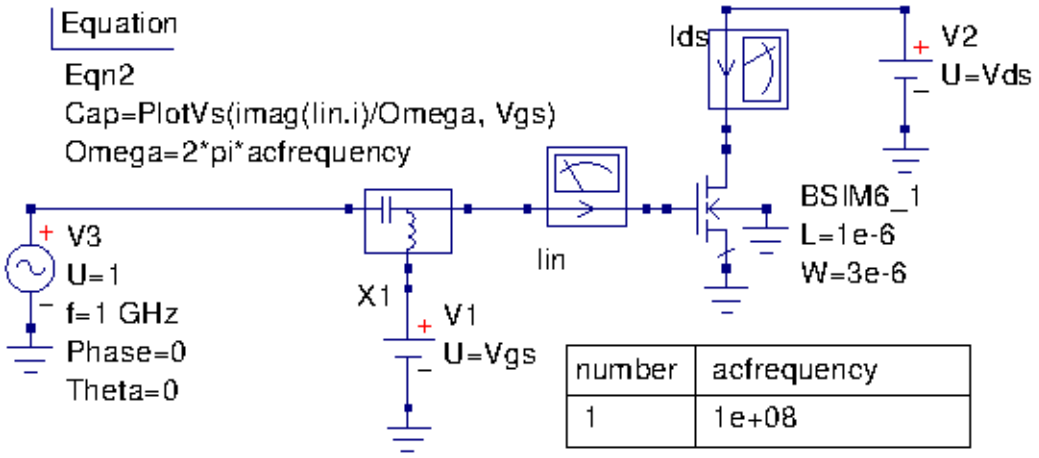
SW2
Sim=DC1
Type=lin
Param=Vds
Start=0
Stop=3
Points=101







Qucs enhanced ADMS/Verilog-A modelling: Part 3; BSIM6 capacitance extraction



number	acfrequency
1	1e+08

Parameter sweep

SW2
Sim=AC1
Type=lin
Param=Vgs
Start=-3
Stop=3
Points=201

Parameter sweep

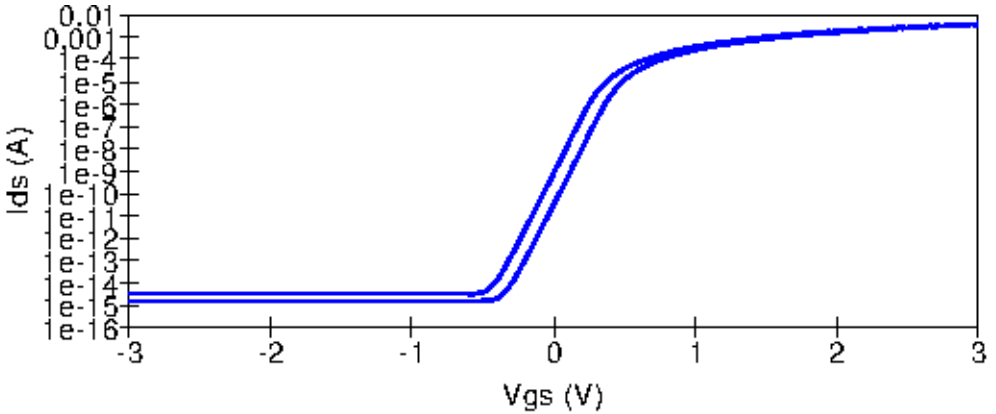
SW1
Sim=SW2
Type=lin
Param=Vds
Start=0
Stop=3
Points=3

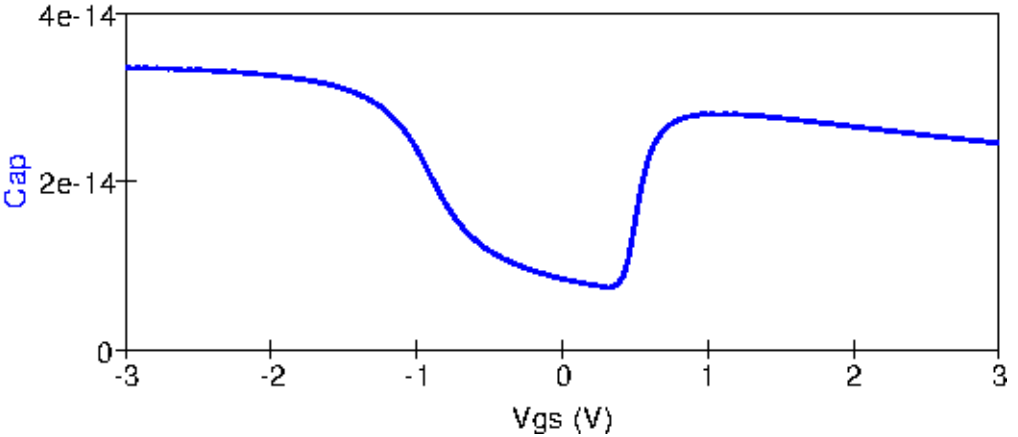
ac simulation

AC1
Type=const
Values=[1e8]

dc simulation

DC1





Qucs polynomial non-linear semiconductor device modelling and HB simulation: Part 1 Introduction

Modelling the non-linear diode Id/Vd equation for Harmonic Balance simulation

The diode equation Id = Is*(exp(Vd/Vt(Temp)-1.0) is subject to numerical overflow when Vd is large.

To ensure that numerical overflow does not occur the Verilog-A hardware description language represents exp(x) by limexp(x) = (x < 80.0) ? exp(x) : exp(80.0)*(1.0+(x-80.0)). Although limexp(x) largely prevents numerical overflow it does so by employing a linear approximation to the diode Id/Vd characteristic and an if statement to select the correct equation for each region of operation, allowing possible higher order discontinuities in the differential Id/Vd characteristics. Such discontinuities tend to cause RF Harmonic Balance simulation to either fail to converge or at best slowly converge to a satisfactory solution. Similar comments also apply to device dynamic charge equations.

Octave numerical curve fitting script

```
Vx =  [-2,-1.5-1,0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,
       1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2,2.3,
       2.4,2.5,2.6,2.7,2.8,2.9,3.0];
Vy =  [-2,-1.5,-1,0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,
        1.08,1.13,1.165,1.2,1.23,1.255,1.28,1.3,1.31,1.33,1.34,
        1.36,1.375,1.385,1.4,1.415,1.43,1.445,1.46,1.475];
%
n= 4;
p = polyfit(Vx, Vy,n)
```

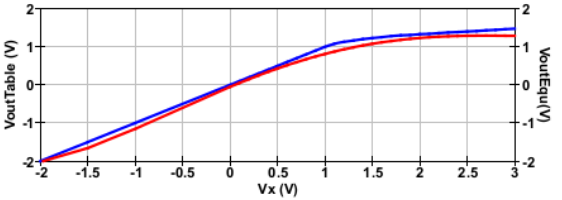
Voltage transfer function

$$Vs=0.01635*Vx^4-0.056794*Vx^3-0.1324*Vx^2+1.04041*Vx-0.051718$$

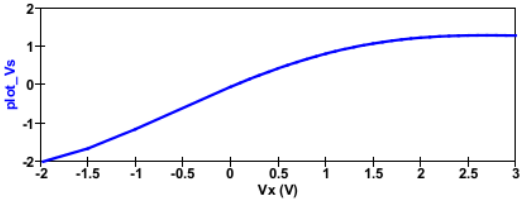
Continuous first and second order differentials

Equation
Eqn2
Vx=[-2,-1.5,-1,0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9,3.0]
Vy=[-2,-1.5,-1,0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.08,1.13,1.165,1.2,1.23,1.255,1.28,1.3,1.31,1.33,1.34,1.36,1.375,1.385,1.4,1.415,1.43,1.445,1.46,1.475]
Vs=(0.011635*Vx^4)-(0.056794*Vx^3)-(0.132402*Vx^2)+(1.04041*Vx)-0.051718

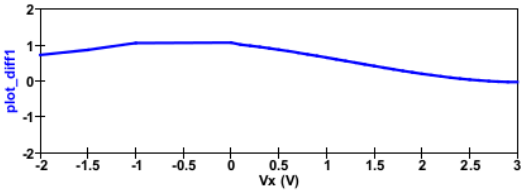
Equation
Eqn4
PlotIdEqu=PlotVs(real(Vs), Vx)
PlotVRatio=PlotVs(Vy, Vx)



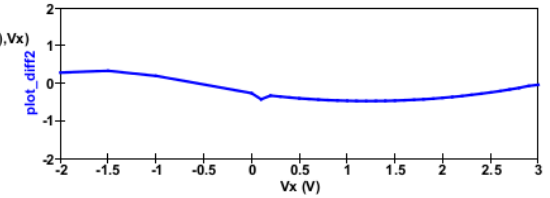
Equation
Eqn5
plot_Vs=PlotVs(real(Vs), Vx)



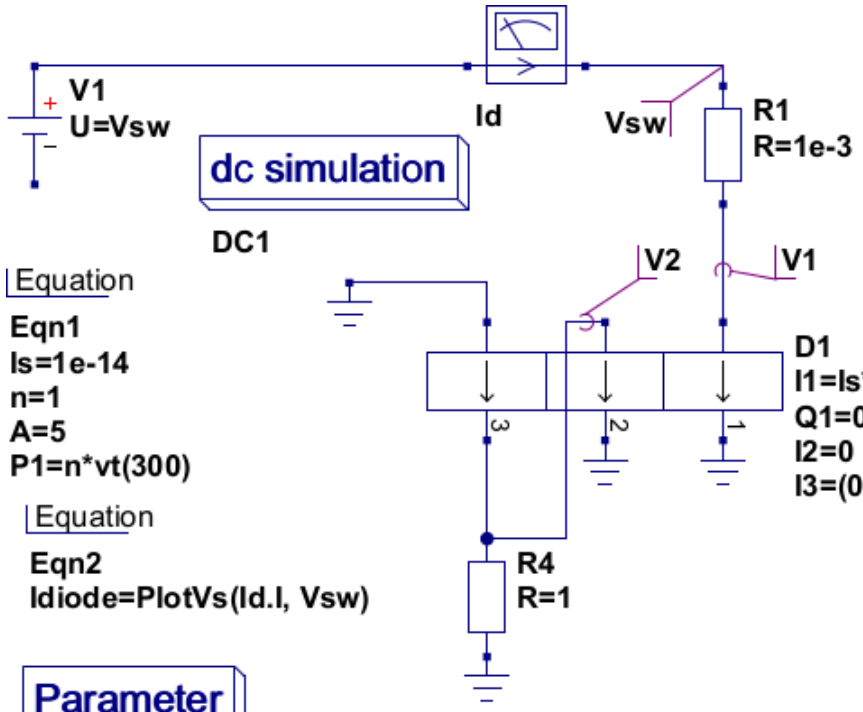
Equation
Eqn6
plot_diff1=PlotVs(diff(plot_Vs,Vx),Vx)

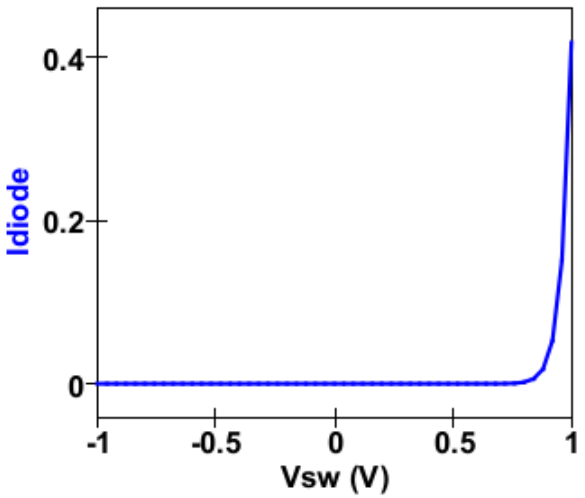


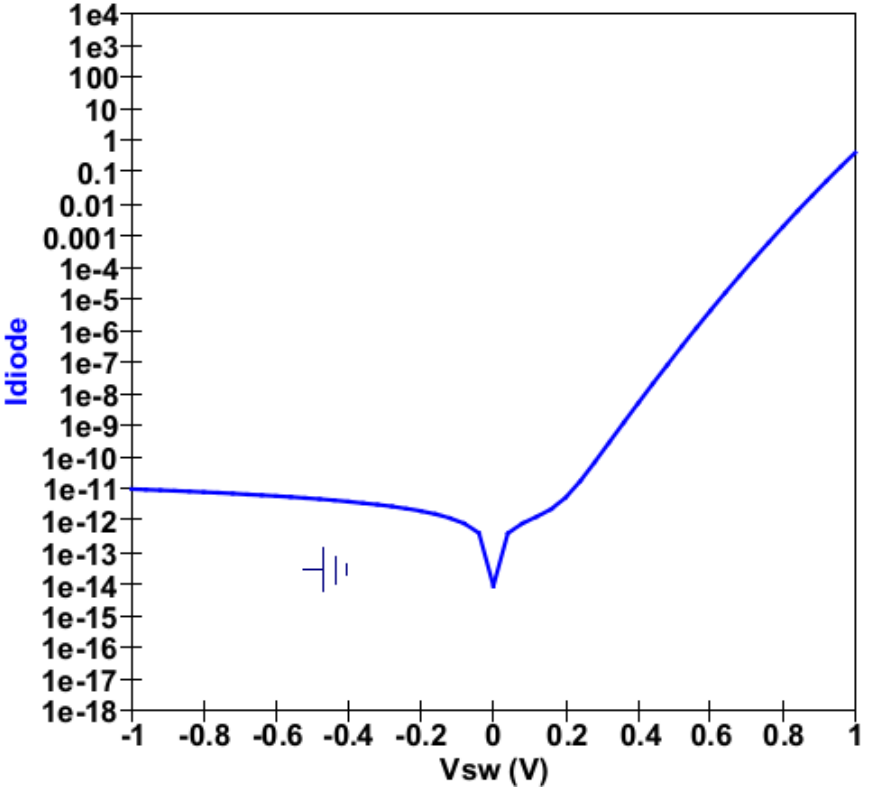
Equation
Eqn7
plot_diff2=PlotVs(diff(diff(real(Vs),Vx),Vx),Vx)



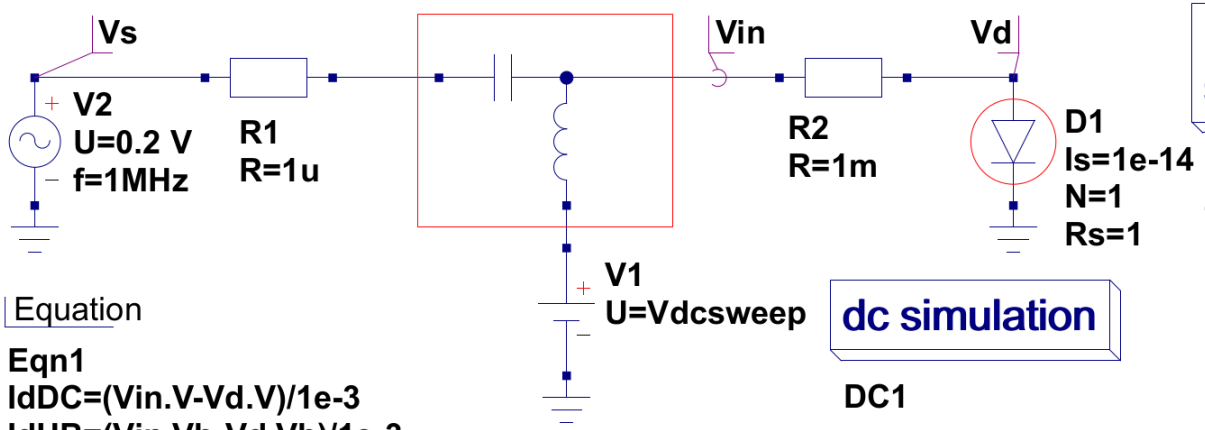
Qucs polynomial non-linear semiconductor device modelling and HB simulation: Part 2 EDD diode Id/Vd model







Qucs polynomial non-linear semiconductor device modelling and HB simulation: Part 3 swept Harmonic Balance simulation



Equation

Eqn1
IdDC=(Vin.V-Vd.V)/1e-3
IdHB=(Vin.Vb-Vd.Vb)/1e-3
PlotIdDC=PlotVs(IdDC, Vd.V)

dc simulation

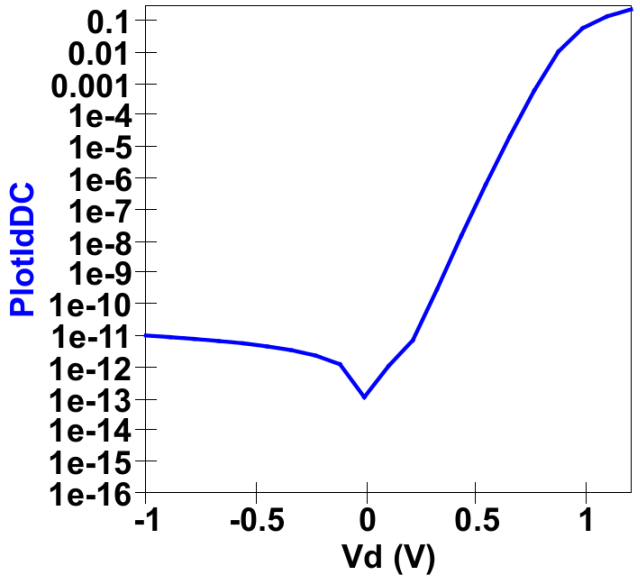
DC1

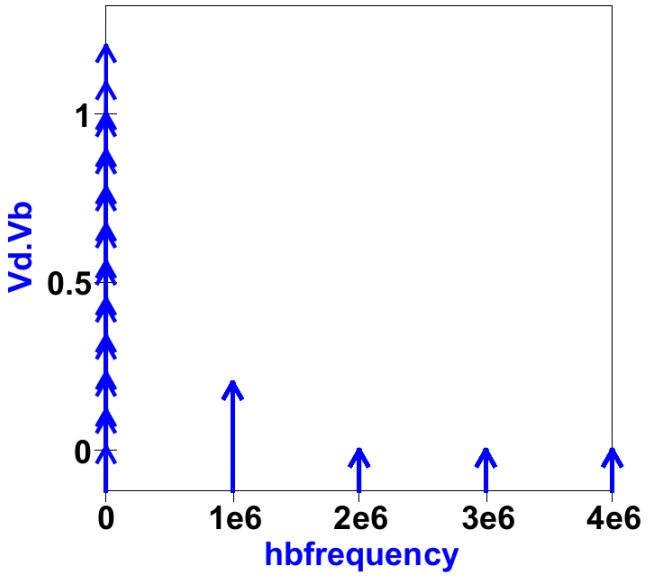
Harmonic balance simulation

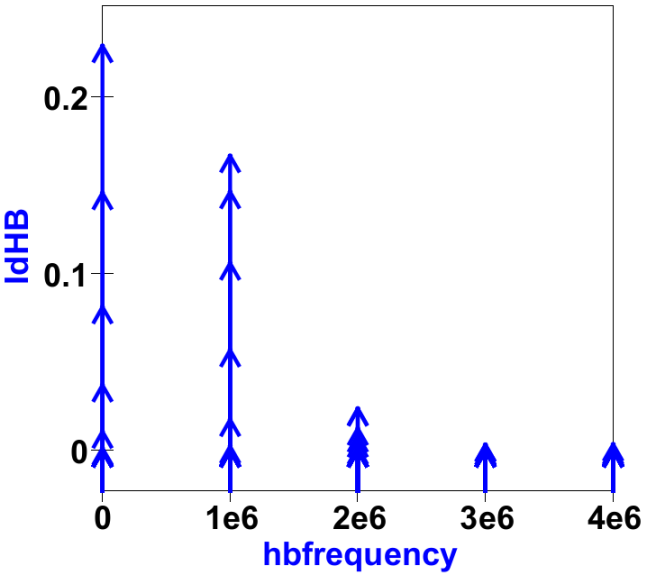
HB1
f=1e6Hz
n=4

Parameter sweep

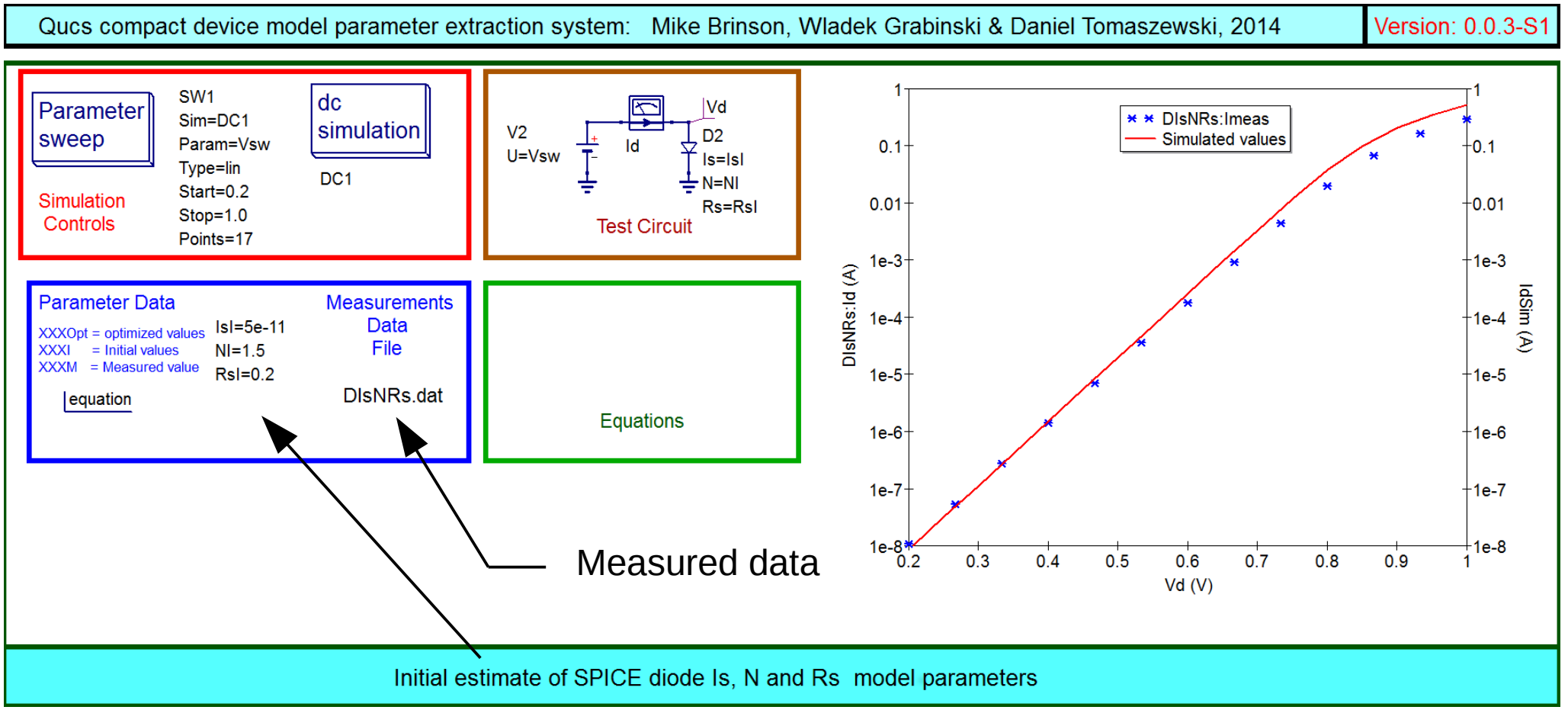
SW1
Sim=HB1
Type=lin
Param=Vdcsweep
Start=-1
Stop=1.2
Points=21





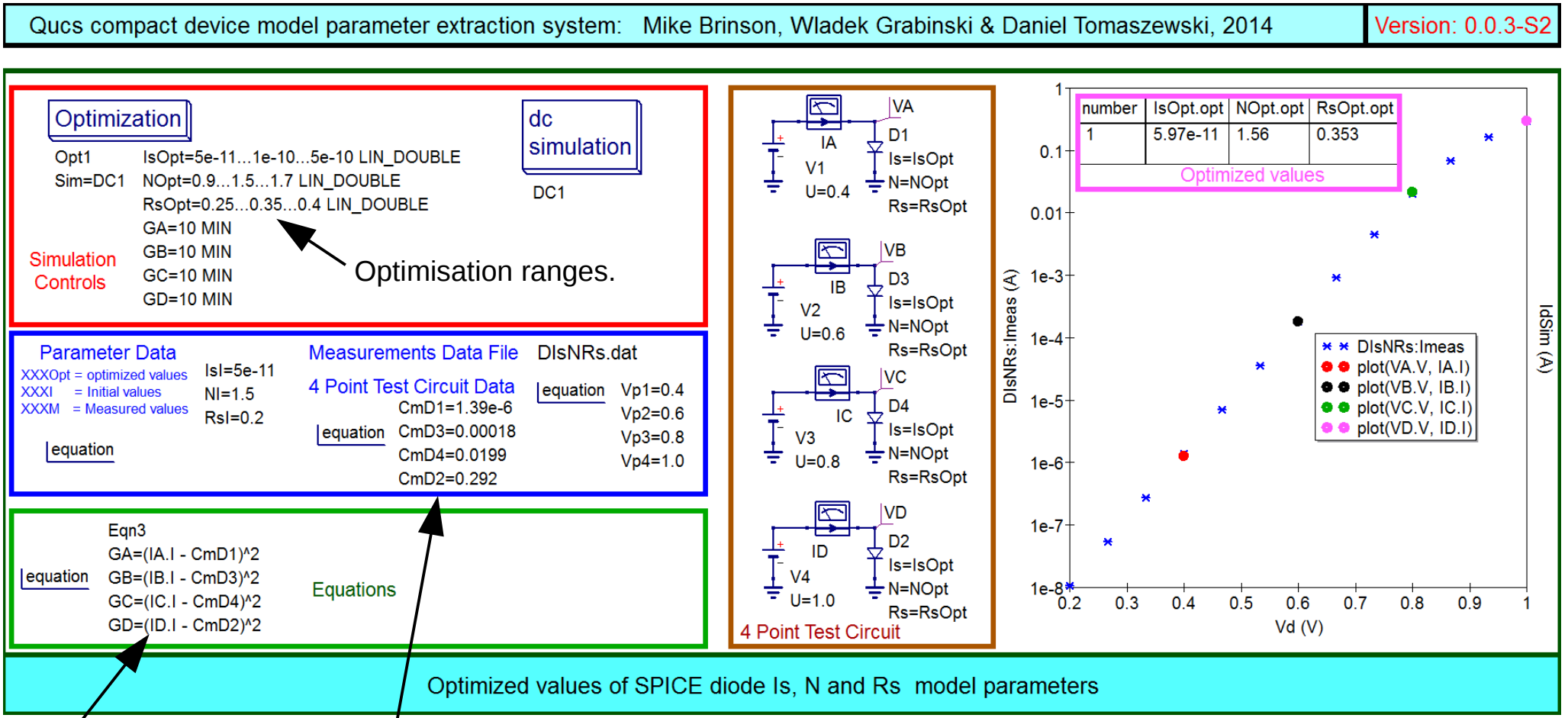


Using Qucs and optimisation for model parameter extraction: Part 1; Manual diode model parameter estimation



Change "Parameter Data" values Isl, Nl and Rsl by hand and press key F2 to simulate circuit performance. Observe the effects of parameter changes from data plot.

Using Qucs and optimisation for model parameter extraction: Part 2; Automated diode parameter estimation using ASCO



Measured values for a four point test circuit.

Optimisation error functions.

Set values for IsI, NI and RsI then press key F2, yields optimised values as output.

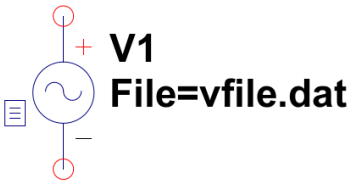
ASCO (A SPICE Circuit Optimizer): available from asco.sourceforge.net

Qucs large signal noise modelling and simulation in the transient domain: Part 1; An Octave script for generating random noise sequences


```
% Experimental Octave XXX.dat script generation program.
% This script generates noise voltage and current Qucs 'dat' files
% for use with file based voltage and current generators.
% Input  ----- Enter data from keyboard as requested.
% Output ----- XXX.dat text file written to disk in working directory.
filename = input('Enter name of dat file [format ''file1.dat''] > ');
Version  = input('Enter Qucs version number [format ''x.x.xx''] > ');
Npoints  = input('Enter the number of data points in noise sequence > ');
Ftime    = input('Enter the finish time of the noise sequence  [in seconds] > ');
IorV     = input('Enter current or voltage generator [format ''I'' or ''V'' ] > ');
Type     = input('Enter type of noise required [format ''n'' or ''f''] > ');
% n = normally distributed pseudo-random elements having zero mean and varience one.
% f = 1/(f^alpha) power law pseudo-random elements.
if (Type == 'n')
   x = randn(1,Npoints);
elseif (Type == 'f')
  alpha = input('Enter power law coefficient > ');
  x = powernoise(alpha, Npoints);
endif
Ts = Ftime/Npoints;
n = 0:length(x)-1;
t=Ts*n;
fid = fopen(filename, "w");
fprintf(fid, "<Qucs Dataset %s>\n", Version);
fputs(fid, "<indep time ");
fprintf(fid, "%d>\n", Npoints);
for i= 1 : length(t)
 fprintf(fid, "%12.3e\n", t(i));
endfor
fputs(fid, "</indep>\n");
if (IorV == 'I')
   fputs(fid, "<dep IOctave.It time>\n");
else
   fputs(fid, "<dep VOctave.Vt time>\n");
endif
for i= 1 : length(t)
 fprintf(fid, "%11.4e\n", x(i));
endfor
fputs(fid, "</dep>\n");
fclose(fid);
```

TranNoiseSourceGen.m

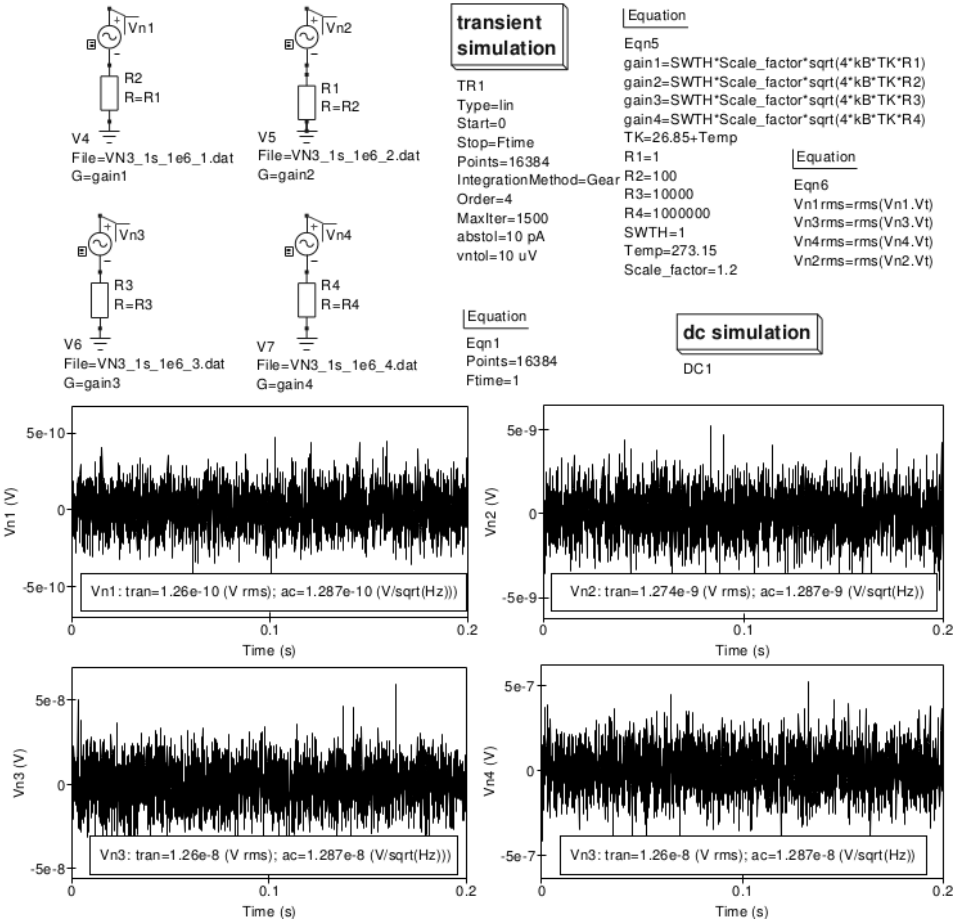
OR



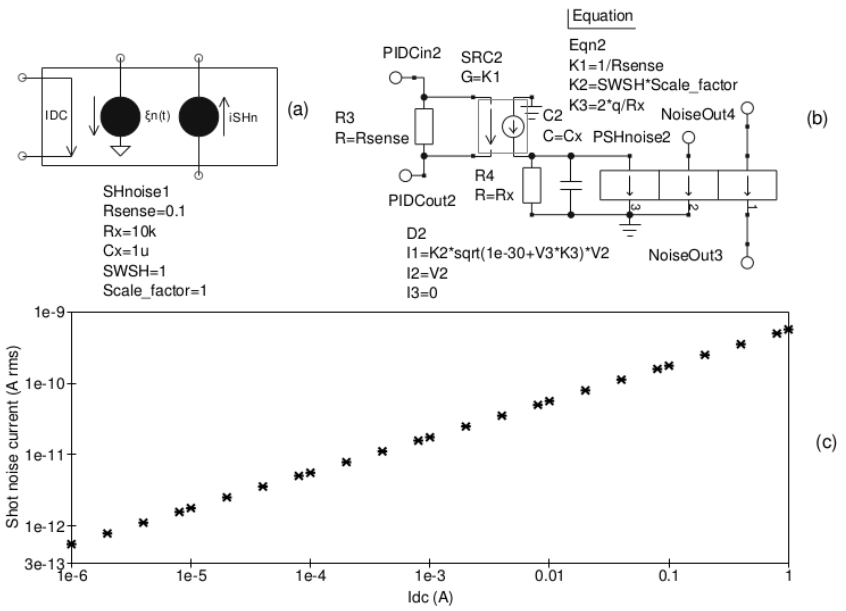

```
function x = powernoise(alpha, N, varargin)
% Generate samples of power law noise. The power spectrum
% of the signal scales as f^(-alpha).
%% Useage:
%  x = powernoise(alpha, N)
%  x = powernoise(alpha, N, 'option1', 'option2', ...)
% Inputs:
%  alpha - power law scaling exponent
%  N     - number of samples to generate
%% Output:
%  x     - N x 1 vector of power law samples
%% With no option strings specified, the power spectrum is
% deterministic, and the phases are uniformly distributed in the range
% -pi to +pi. The power law extends all the way down to 0Hz (DC)
% component. By specifying the 'randpower' option string however, the
% power spectrum will be stochastic with Chi-square distribution. The
% 'normalize' option string forces scaling of the output to the range
% [-1, 1], consequently the power law will not necessarily extend
% right down to 0Hz.
% (cc) Max Little, 2008. This software is licensed under the
% Attribution-Share Alike 2.5 Generic Creative Commons license:
% http://creativecommons.org/licenses/by-sa/2.5/
% If you use this work, please cite:
% Little MA et al. (2007), "Exploiting nonlinear recurrence and fractal
% scaling properties for voice disorder detection",
% Biomed Eng Online, 6:23
% As of 20080323 markup
% If you use this work, consider saying hi on comp.dsp
% Dale B. Dalrymple
opt_randpow = false;
opt_normal = false;
for j = 1:(nargin-2)
    switch varargin{j}
        case 'normalize', opt_normal = true;
        case 'randpower', opt_randpow = true;
    end
end
N2 = floor(N/2)-1;
f = (2:(N2+1))';
A2 = 1./(f.^(alpha/2));
if (~opt_randpow)
    p2 = (rand(N2,1)-0.5)*2*pi;
    d2 = A2.*exp(i*p2);
else
    % 20080323
    p2 = randn(N2,1) + i * randn(N2,1);
    d2 = A2.*p2;
end
d = [1; d2; 1/((N2+2)^alpha); flipud(conj(d2))];
x = real(ifft(d));
if (opt_normal)
    x = ((x - min(x))/(max(x) - min(x)) - 0.5) * 2;
end
```

powernoise.m

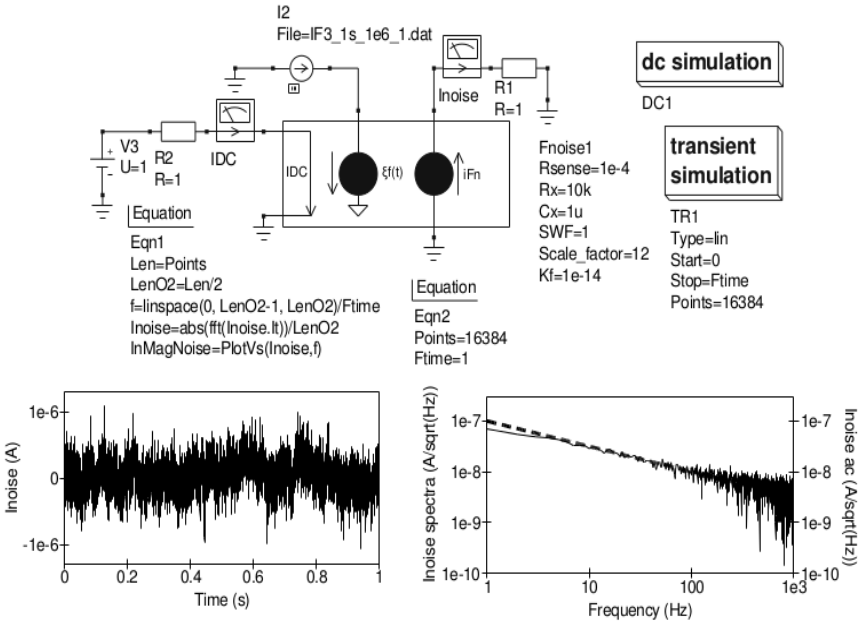
Qucs large signal noise modelling and simulation in the transient domain: Part 2; Thermal, shot and 1/f noise models



Thermal

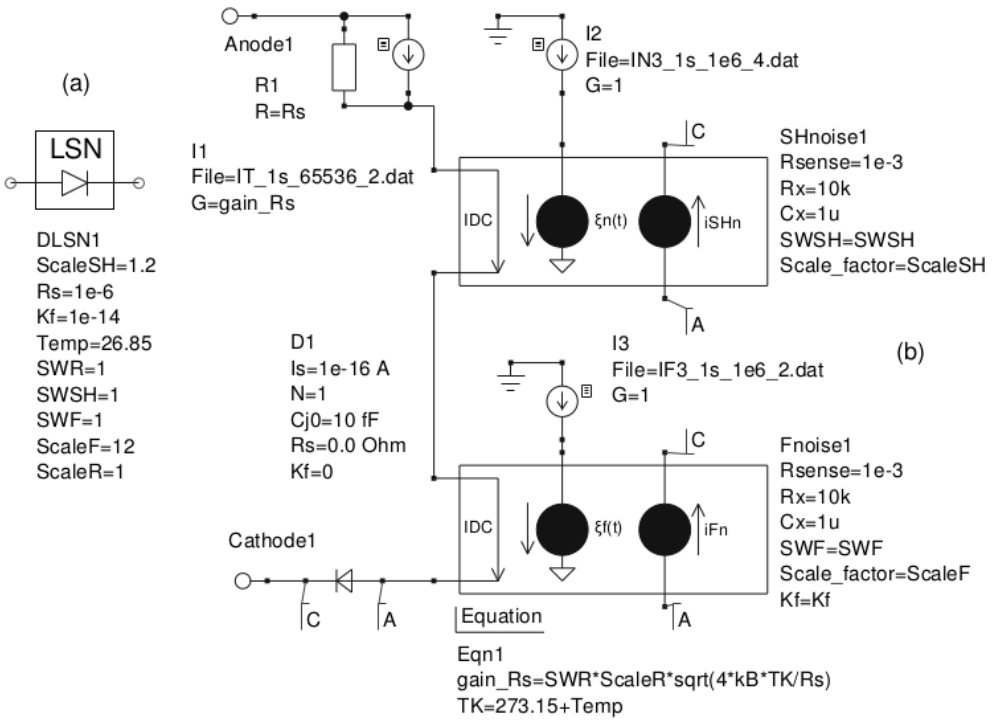


Shot

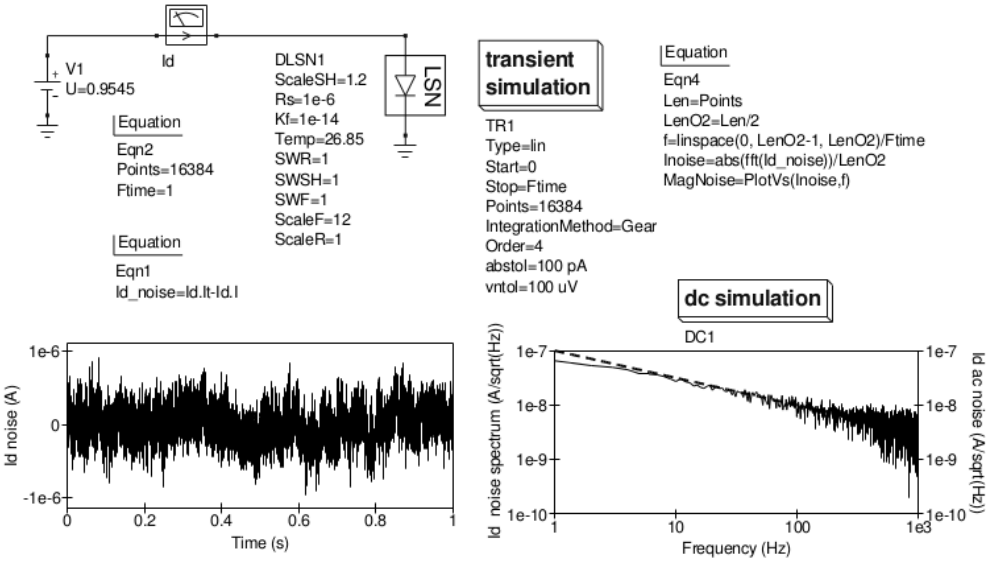


1/f

Qucs large signal noise modelling and simulation in the transient domain: Part 3; A large signal semiconductor noise model

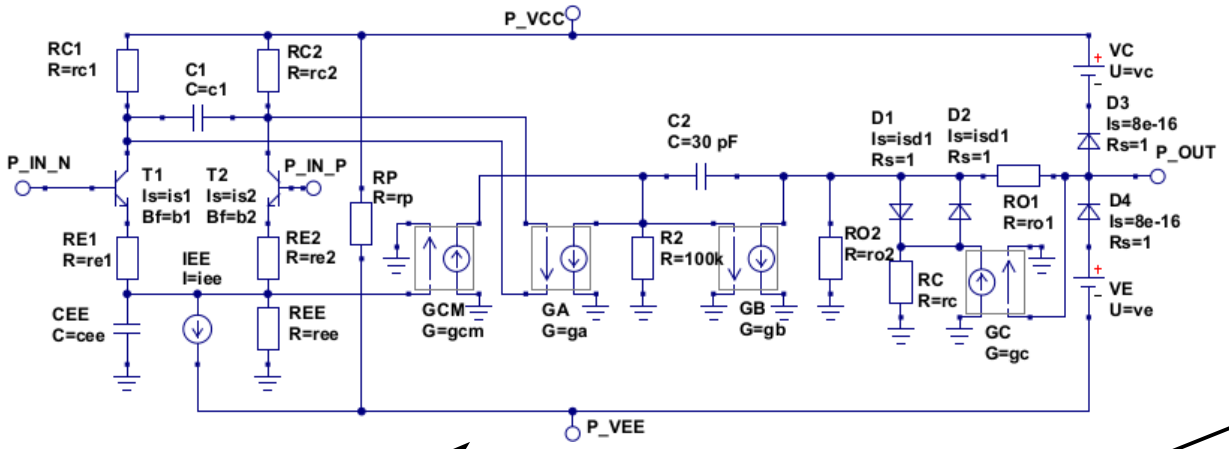


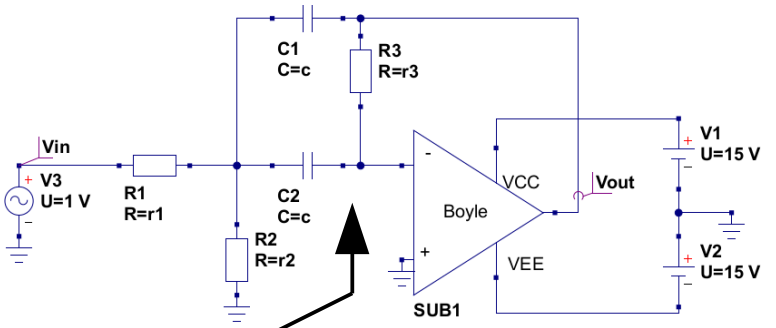
Diode large signal noise model



Diode noise test circuit

Merging circuit design with Qucs simulation





Equation
Eqn1
Q=f0/bw
r1=r3/(2*h0)
c=1/(pi*r3*bw)
r2=r3/(4*Q^2-2*h0)

Equation
Eqn2
gain=dB(Vout.v)
phase=phase(Vout.v)

Equation
Eqn3
bw=bandwidth
r3=100e3
f0=1e3
h0=1

dc simulation
DC1

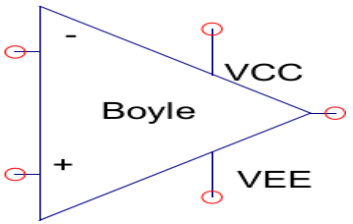
ac simulation
AC1
Type=lin
Start=800Hz
Stop=1200Hz
Points=21

Parameter sweep
SW1
Sim=AC1
Type=lin
Param=bandwidth
Start=100
Stop=400
Points=10

Circuit design equations

Equation
Eqn1
is1=is
ic1=0.5*c2*pswrt
ic2=ic1
VT=vt(300)
is2=is1*(1+vos/VT)
ib1=ib-0.5*ios
ib2=ib+0.5*ios
b1=ic1/ib1
b2=ic2/ib2
iee=ic1/(((b1+1)/b1)+((b2+1)/b2))
gm1=ic1/VT
rc1=1/(2*pi*gbp*c2)
rc2=rc1
re1=((b1+b2)/(2+b1+b2))*(rc1-1/gm1)
re2=re1
ree=va/iee
cee=(2*ic1/nswrt)-c2
dphi=90-pm
c1=0.5*c2*tan(dphi*pi/180)
gcm=1/(cmrr*rc1)
ga=1/rc1
gb=(avol*rc1)/(r2*ro2)
ix=2*ic1*r2*gb-is1
temp1=-(ro1*is1/VT)
isd1=ix*exp(temp1)+1e-32
temp2=ix/isd1
rc=VT*ln(temp2)/(100*ix)
gc=1/rc
vc=abs(vcc)-vsp+VT*ln(iscp/is1)
ve=abs(vee)-vsn+VT*ln(iscn/is1)
rp=(vcc-vee)*(vcc-vee)/pd

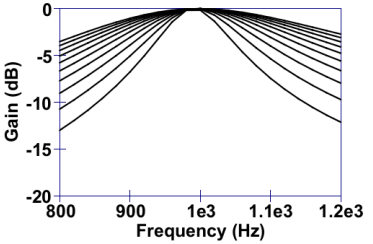
Macromodel design equations

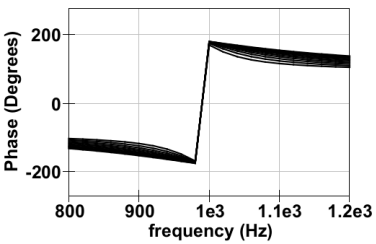


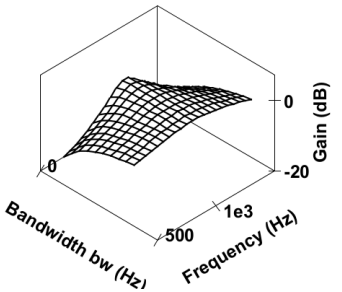
SUB1
is=8e-16
vos=0.7e-3
c2=30e-12
r2=100e3
pswrt=0.625e6
nswrt=0.5e6
ios=20e-9
ib=80e-9
gbp=1e6
va=200
pm=70
cmrr=31622.8
avol=200e3
vsp=14.2
vsn=-13.5
vcc=15
vee=-15
iscp=25e-3
iscn=25e-3

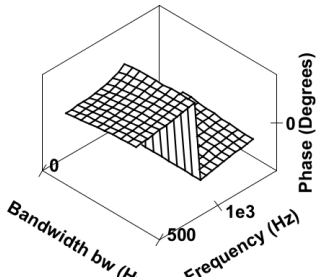
Macromodel parameters

Filter specification



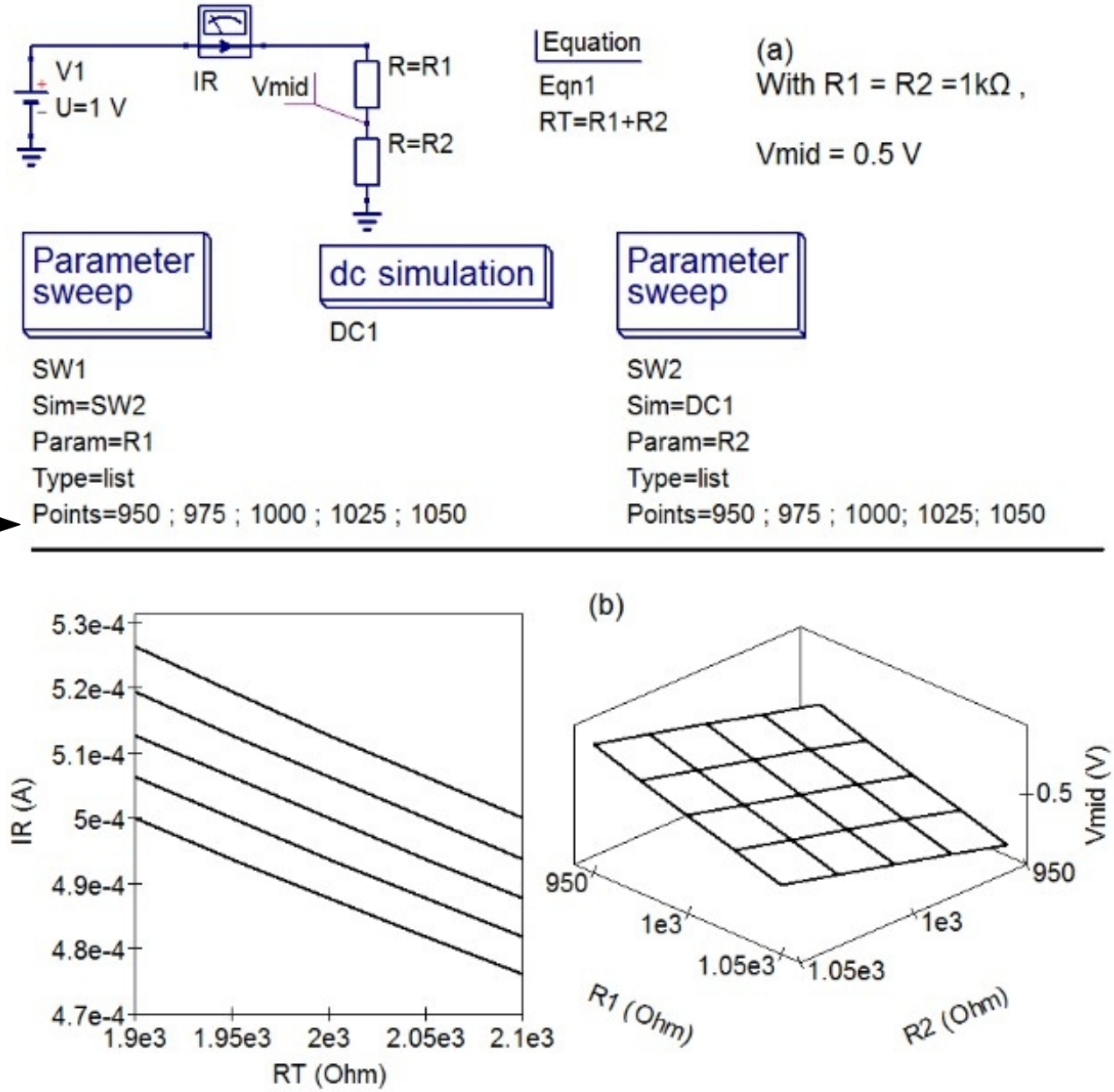






M.E. Brinson and H. Nabijou, Adaptive subcircuits and compact Verilog-A macromodels as integrated design and analysis blocks in Qucs circuit simulation, International Journal of Electronics, Vol. 98, No. 5, June 2011 , 631-645.

Qucs statistical circuit simulation using Octave or Python: Part 1; Sensitivity analysis of a two resistor voltage divider



Qucs statistical circuit simulation using Octave or Python: Part 2; Text form of “Parameter sweep” icon

SW1
Sim=DC1
Param=p1
Type=list
Points=9.917e+002; 1.042e+0023; 9.714e+002; 1.028e+003; 9.972e+002.....

```
<QucsStudio Schematic 1.5.0>
<Properties>
  <View=500,131,1323,504,1,0,0>
  <Grid=10,10,1>
  <DataSet=slide6.dat>
  <DataDisplay=slide6.dpl>
  <OpenDisplay=1>
  <showFrame=0>
  <FrameText0=Title>
  <FrameText1=Drawn By:>
  <FrameText2=Date:>
  <FrameText3=Revision:>
</Properties>
<Symbol>
</Symbol>
<Components>
  <.SW SW1 1 550 180 0 89 0 0 "DC1" 1 "p1" 1 "list" 1 "5 Ohm" 0 "50 Ohm" 0 "9.917e+002; 1.042e+0023; 9.714e+002; 1.028e+003; 9.972e+002....." 1>
</Components>
<Wires>
</Wires>
<Diagrams>
</Diagrams>
<Paintings>
</Paintings>
```

Qucs statistical circuit simulation using Octave or Python: Part 3; Using Octave to generate Parameter sweep lists and icon code

```
% QucsStudio tabular normal distribution file generator.
%
% INPUT:  Enter data from keyboard as requested.
% OUTPUT: Script writes file XXXX.sch containing
%         a QucsStudio swept parameter icon with
%         a list of "npoints" items normally distributed
%         around a specified mean and standard deviation.
%
% Script written with Octave 3.2.4
% Copyright 2012 by Mike Brinson
% Published under GNU General Public License (GPL V2).
% No warranty at all.
%
filename = input("Enter name of Tabular Distribution File > ", "s");
version  = input("Enter QucsStudio version number > ", "s");
npoints  = input("Enter the number of data points in the tabular sequence > ");
mean     = input("Enter mean value > ");
tol      = input("Enter tolerance [ in percentage ] > ");
std = (tol.*mean)./300.0;
n = normrnd(mean, std, 1, npoints);
fname = strcat(filename,".sch");
%swname = strcat("SW","1");
fid  =  fopen(fname, "w");
fprintf(fid, "<QucsStudio Schematic %s>\n", version);
fputs(fid, "<Properties>\n");
fputs(fid, "   <View=0,0,3256,800,1,0,0>\n");
fputs(fid, "   <Grid=10,10,1>\n");
fprintf(fid, "   <DataSet=%s.dat>\n", filename);
fprintf(fid, "   <DataDisplay=%s.dpl>\n", filename);
fputs(fid, "   <OpenDisplay=1>\n");
fputs(fid, "   <showFrame=0>\n");
fputs(fid, "   <FrameText0=Title>\n");
fputs(fid, "   <FrameText1=Drawn By:>\n");
fputs(fid, "   <FrameText2=Date:>\n");
fputs(fid, "   <FrameText3=revision:>\n");
fputs(fid, "</Properties>\n");
fputs(fid, "<Symbol>\n");
fputs(fid, "</Symbol>\n");
fputs(fid, "<Components>\n");
fputs(fid, "<.SW SW1 1 90 120 0 83 0 0 ");
fputs(fid, "\"DC1\" 1  \"p1\" 1 \"list\" 1  \"   \" 0  \"   \" 0 \" ");
for i =1:npoints-1
  fprintf(fid," %12.3e ;", n(i));
endfor
fprintf(fid, " %12.3e \" 1>\n", n(npoints));
fputs(fid, "</Components>\n");
fputs(fid, "<Wires>\n");
fputs(fid, "</Wires>\n");
fputs(fid, "<Diagrams>\n");
fputs(fid, "</Diagrams>\n");
fputs(fid, "<Paintings>\n");
fputs(fid, "</Painting>\n");
fclose(fid);
```

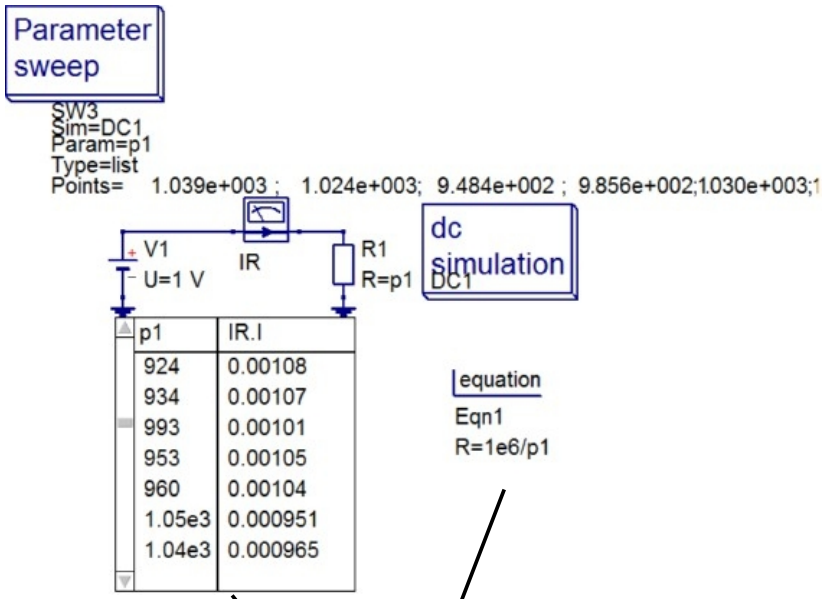
Input Data:

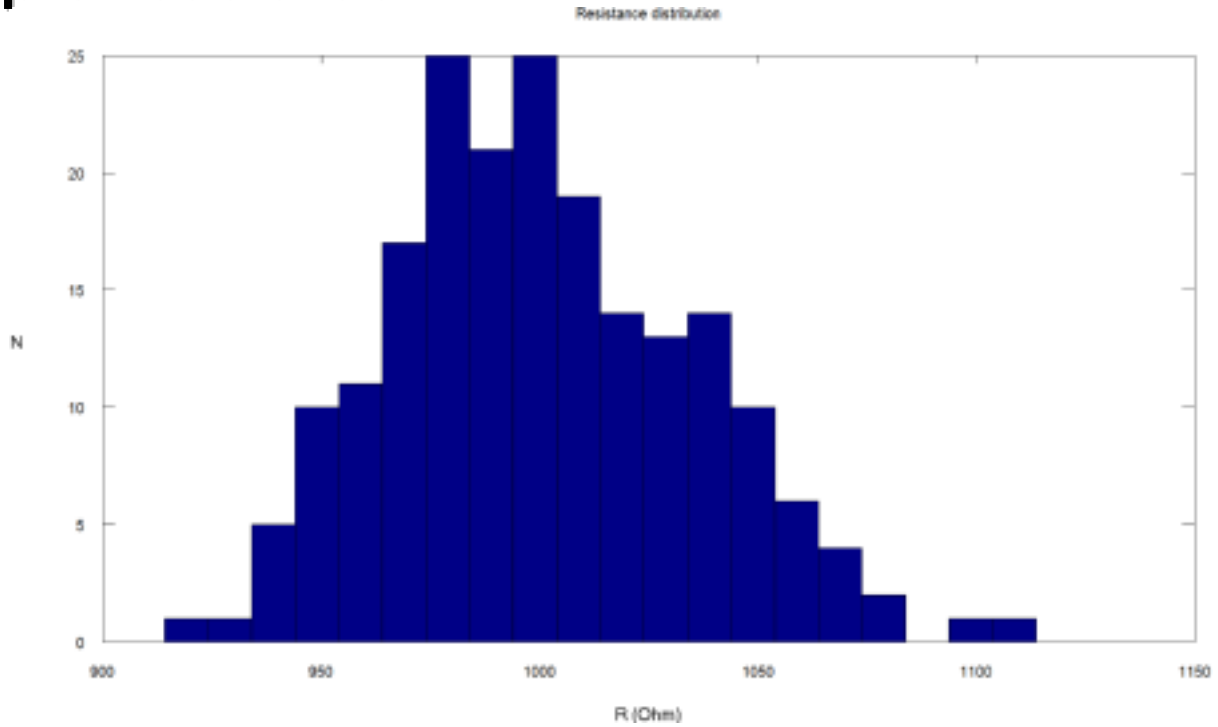
1. **Parameter sweep icon file name.**
2. **Version number.**
3. **Number of data points in list.**
4. **Mean value of data.**
5. **Data tolerance (in %)**

Octave can generate random numbers from a large number of distributions, for example

Uniform	**unifrnd**
Binomial	**binrnd**
Exponential	**exprnd**
Normal	**normrnd**
Weibull	**wblrnd**

Qucs statistical circuit simulation using Octave or Python: Part 4; An example : Distribution of resistance component values generated using the Octave script presented in the previous slide






```
% Example component value distribution plot using Octave
clear;
qucsFilename = "TestFig4.dat";
loadQucsDataset;
whos;
PlotHistogram(R, 20,  " R (Ohm) ", "N", "Resistance distribution");
```

```
% Parameters
% Y                                   plot data variable name.
% Xlabel                X axis label (string).
% Ylabel                Y axis label (string).
% Title                 Title at top of histogram (string).
function PlotHistogram(Y, NBins, Xlabel, Ylabel, Title)
 if(nargin < 5)
   error("Too few input parameters.")
 return
 endif
    hist(Y,NBins);
          xlabel(Xlabel, "fontsize", 14, "fontname", "Arial", "fontweight", "bold");
    ylabel(Ylabel, "fontsize", 14, "fontname", "Arial", "fontweight", "bold");
    title(Title)
endfunction
```

Brinson M.E., A Qucs/QucsStudio swept parameter technique for statistical circuit simulation, International Journal of Microelectronics and Computer Science, 2013, Volume 4, Number 3, pp. 92-97.

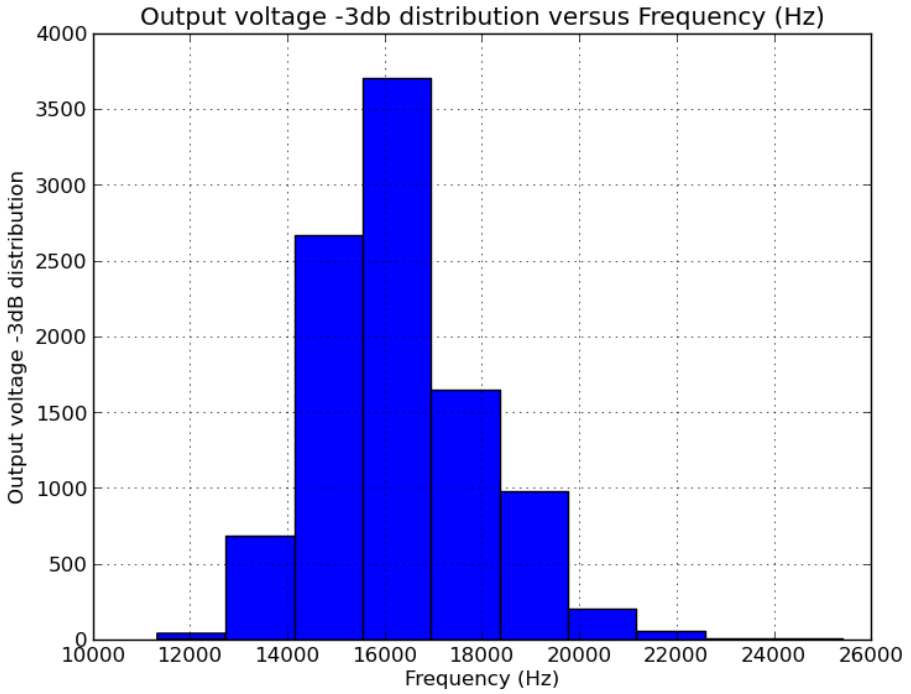
Qucs statistical circuit simulation using Octave or Python: Part 5; A basic Python statistical example; -3dB output voltage distribution for a RC filter

Determine -3dB frequency

Generate Qucs netlist from rc_ac.net

Run Qucs

```
# -*- coding: utf-8 -*-
"""
Qucs/Python sensitivity script sensitivity.py
This is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2, or (at your option)
any later version.
Copyright (C), C. Novak, January 2014: Modified  Mike Brinson, February 2014
"""
import subprocess
import parse_result as pr
import numpy as np
import numpy.random as rnd
import matplotlib.pyplot as plt

from string import Template


def minus3dB(x, y):
    for (ind, xval) in enumerate(x):
        if y[ind] < 0.71:
            return x[ind]

def create_NetList(netlist_filename, values):
    templ_file = open(netlist_filename, 'r')
    templ = Template(templ_file.read())
    netlist = templ.substitute(values)
    f = open('temp.net', 'w')
    f.write(netlist)
    f.close()

def runSim():
    Fin = 'temp.net'
    Fout = 'sim_result.dat'
    netfile = open(Fin, 'r')
    outfile = open(Fout, 'w')
    process = subprocess.Popen('./qucsator', stdin=netfile, stdout=outfile)
    process.wait()
    netfile.close()
    outfile.close()

N = 10000
netlist_filename = 'rc_ac.net'
freq_array = []
for i in range(N):
    point = {}
    point['Rval'] = 1e3 + 100 * rnd.randn()
    point ['Cval'] = 10e-9
    create_NetList(netlist_filename, point)
    runSim()
    data = pr.parse_file('sim_result.dat')
    x = data['acfrequency']
    y = np.abs(data['out.v'])
    freq = minus3dB(x, y)
    freq_array.append(freq)

plt.hist(freq_array, 10)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Output voltage -3dB distribution')
plt.title('Output voltage -3db distribution versus Frequency (Hz)')
plt.grid()
plt.show()
```




```
Vac:V1 in gnd U="1 V" f="1 kHz"
.AC:AC1 Start="1 Hz" Stop="10 MHz" Points="500" Type="log"

R:R1 out in R="$Rval Ohm"
C:C1 out gnd C="$Cval F"
```

rc_ac.net

Histogram plotted using Python hist() function

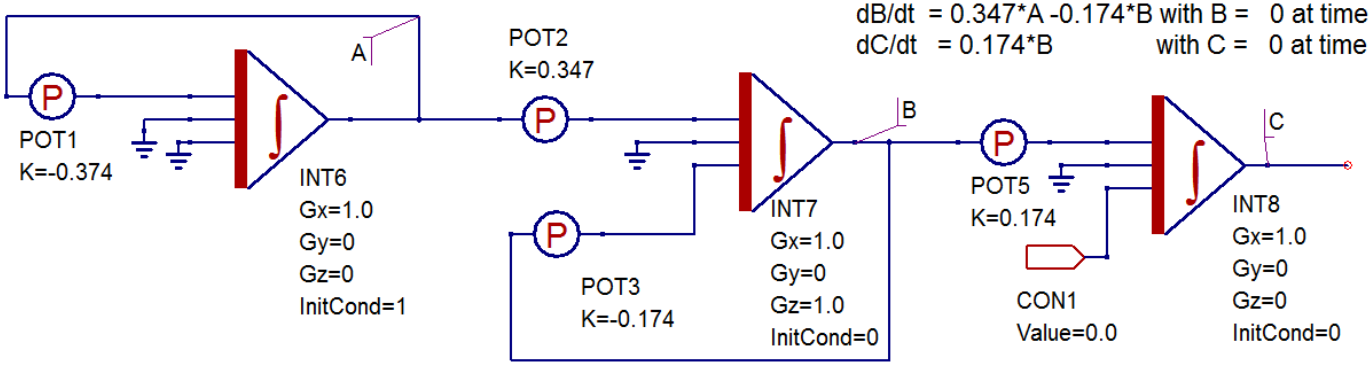
C. Novak, "Qucs report: Using Qucs in Text mode", 2013. Available from http://qucs.sourceforge.net/docs.html.

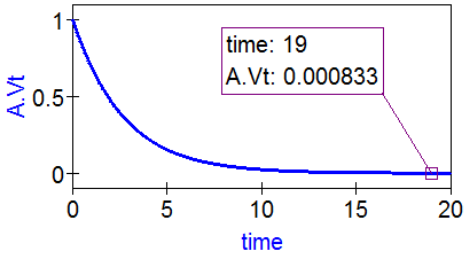
Qucs system simulation: Part 1 continuous systems

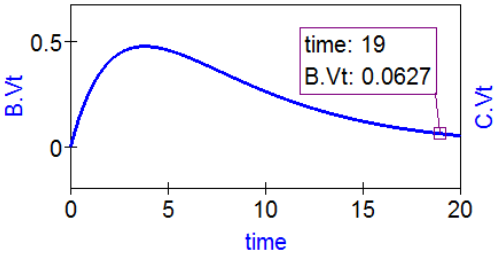
Radio-active decay simulation in the time domain

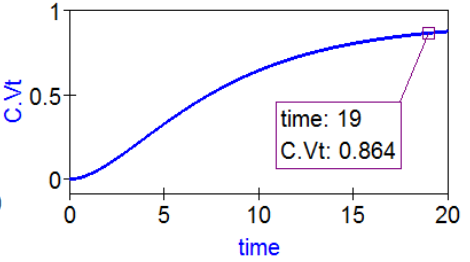
Radio-active decay model : where the radioactive material decays at a rate proportional to the amount of material present; A -> B -> C and

dA/dt = -0.347*A, with A = 1 at time = 0 s.
dB/dt = 0.347*A -0.174*B with B = 0 at time = 0 s.
dC/dt = 0.174*B with C = 0 at time = 0 s.





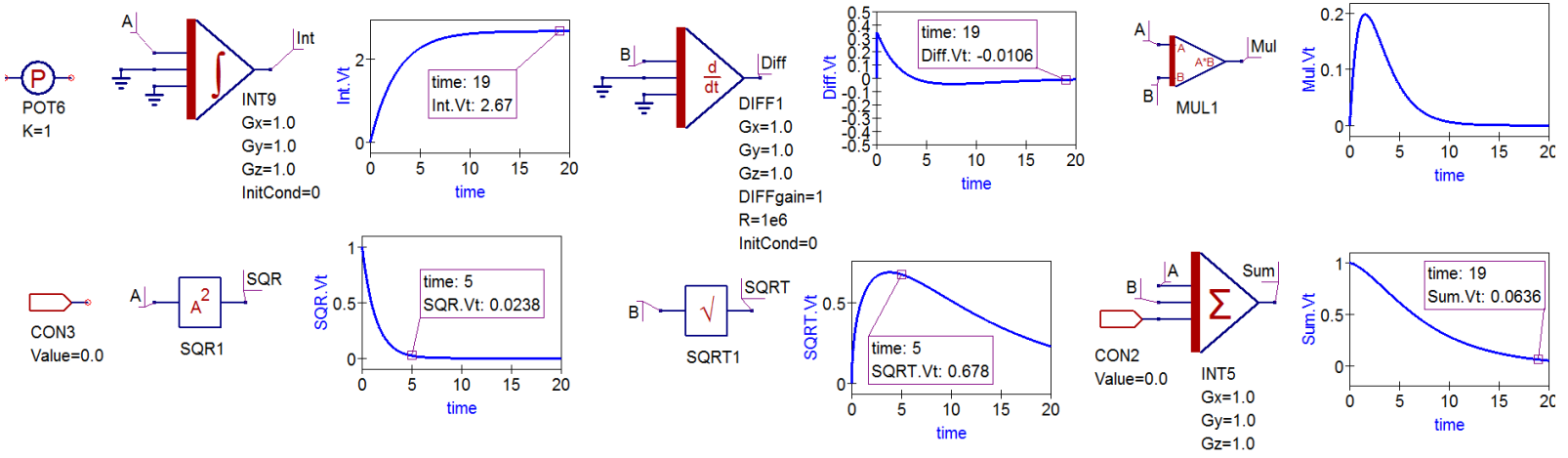




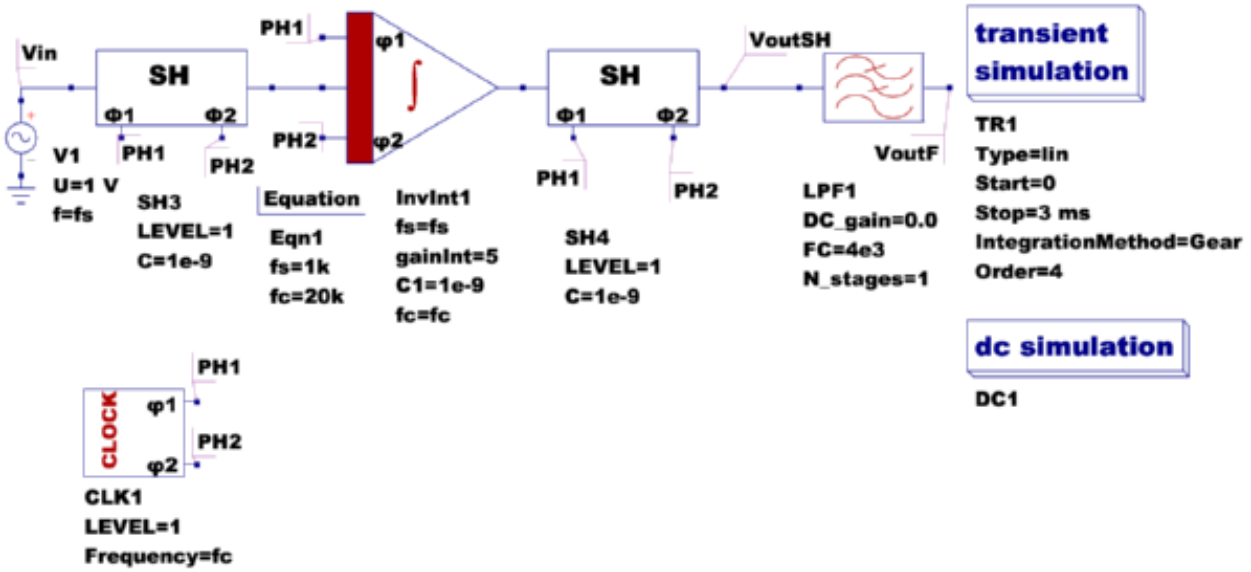
transient simulation

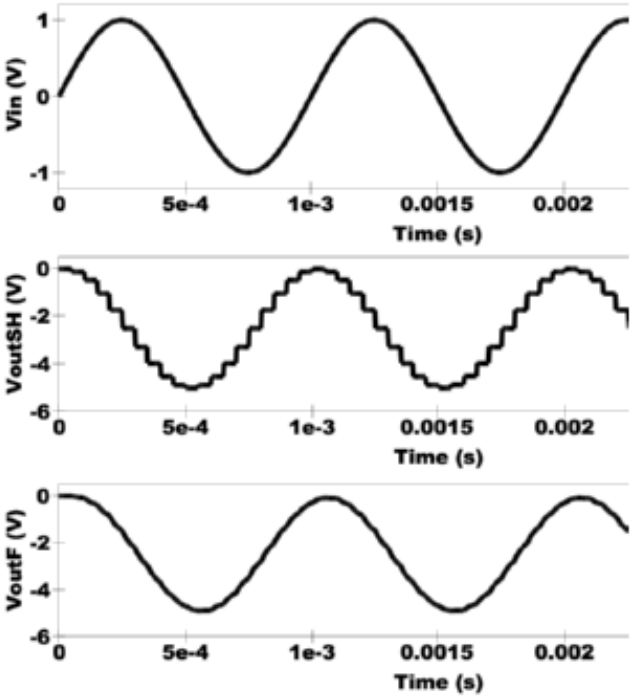
```
TR1
Type=lin
Start=0
Stop=20
Points=2001
method=Gear4
MinStep=1e-16
MaxIter=300
reltol=0.001
abstol=1 µA
initialDC=no
```

Example continuous system building blocks

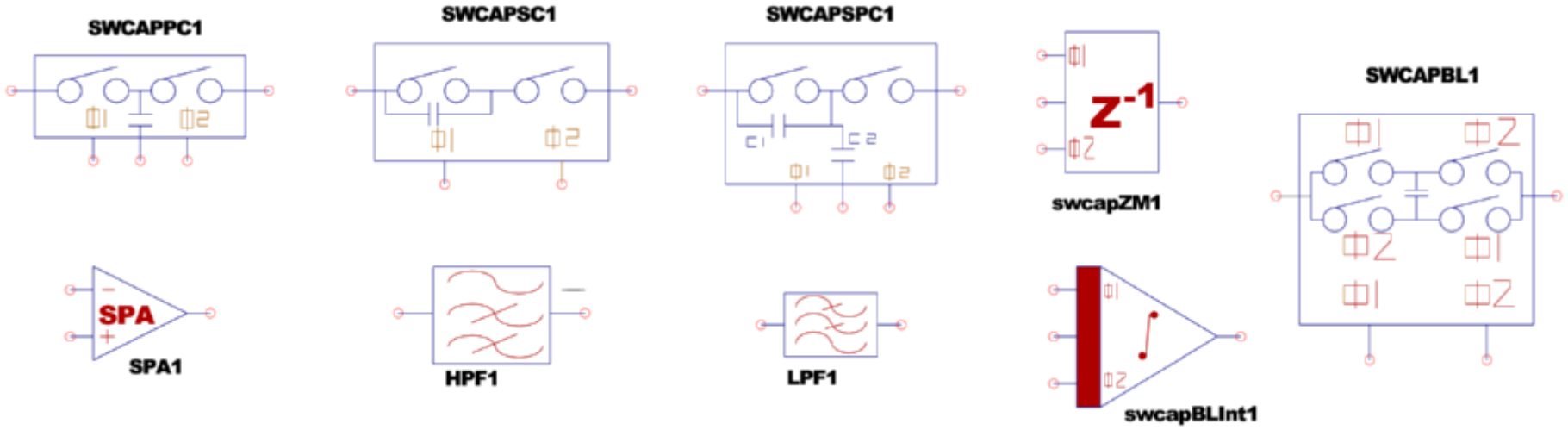


Qucs system simulation: Part 2 sampled data systems

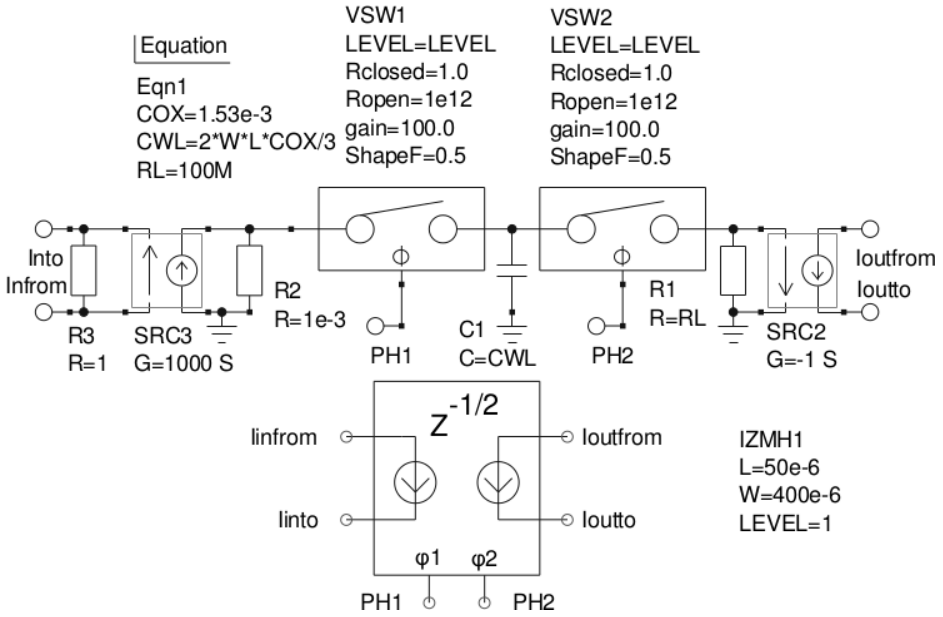


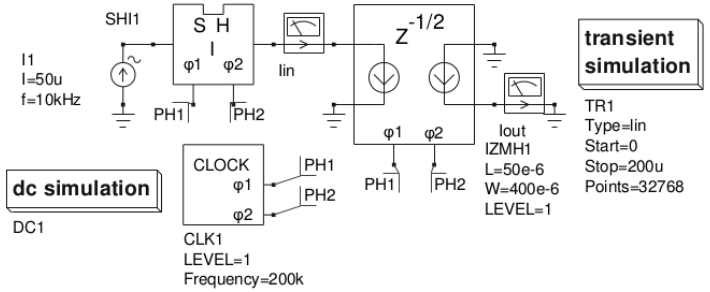


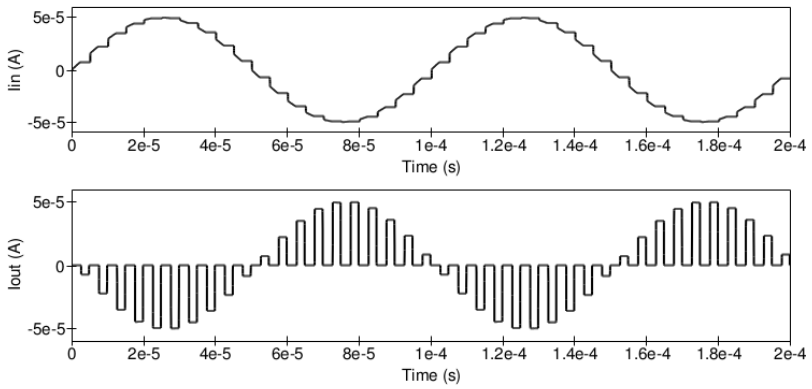
Example sampled data system building blocks



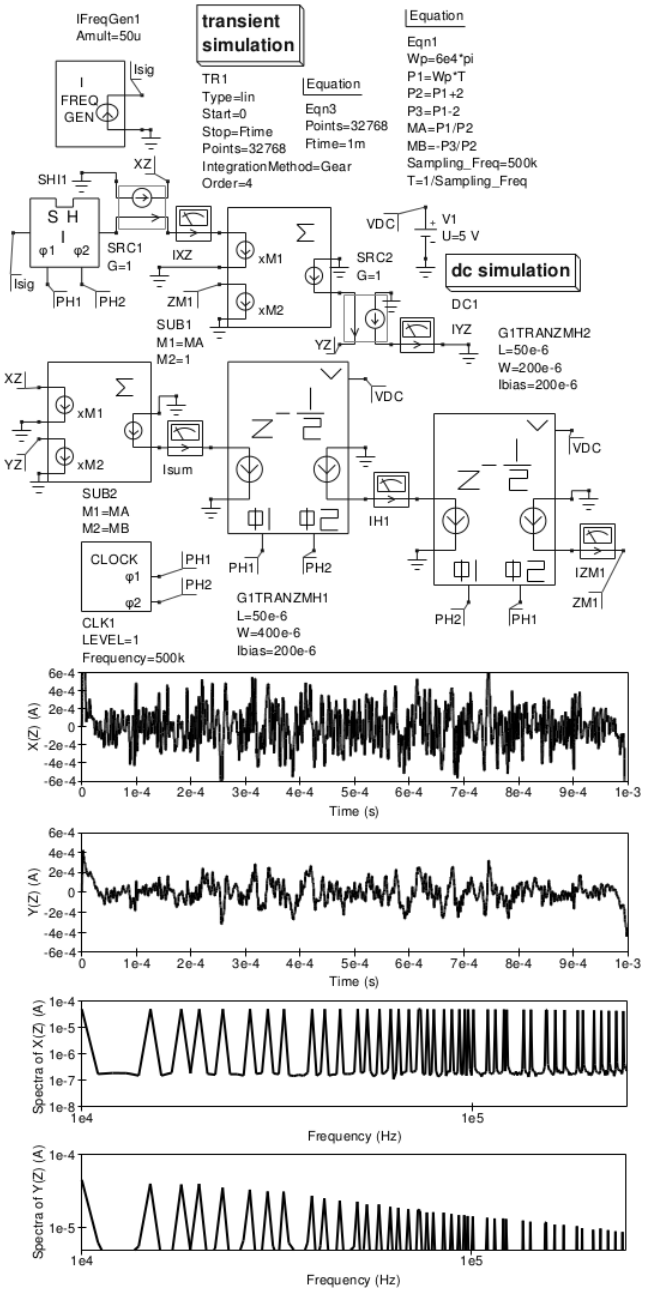
Qucs system simulation: Part 3 switched current analogue systems



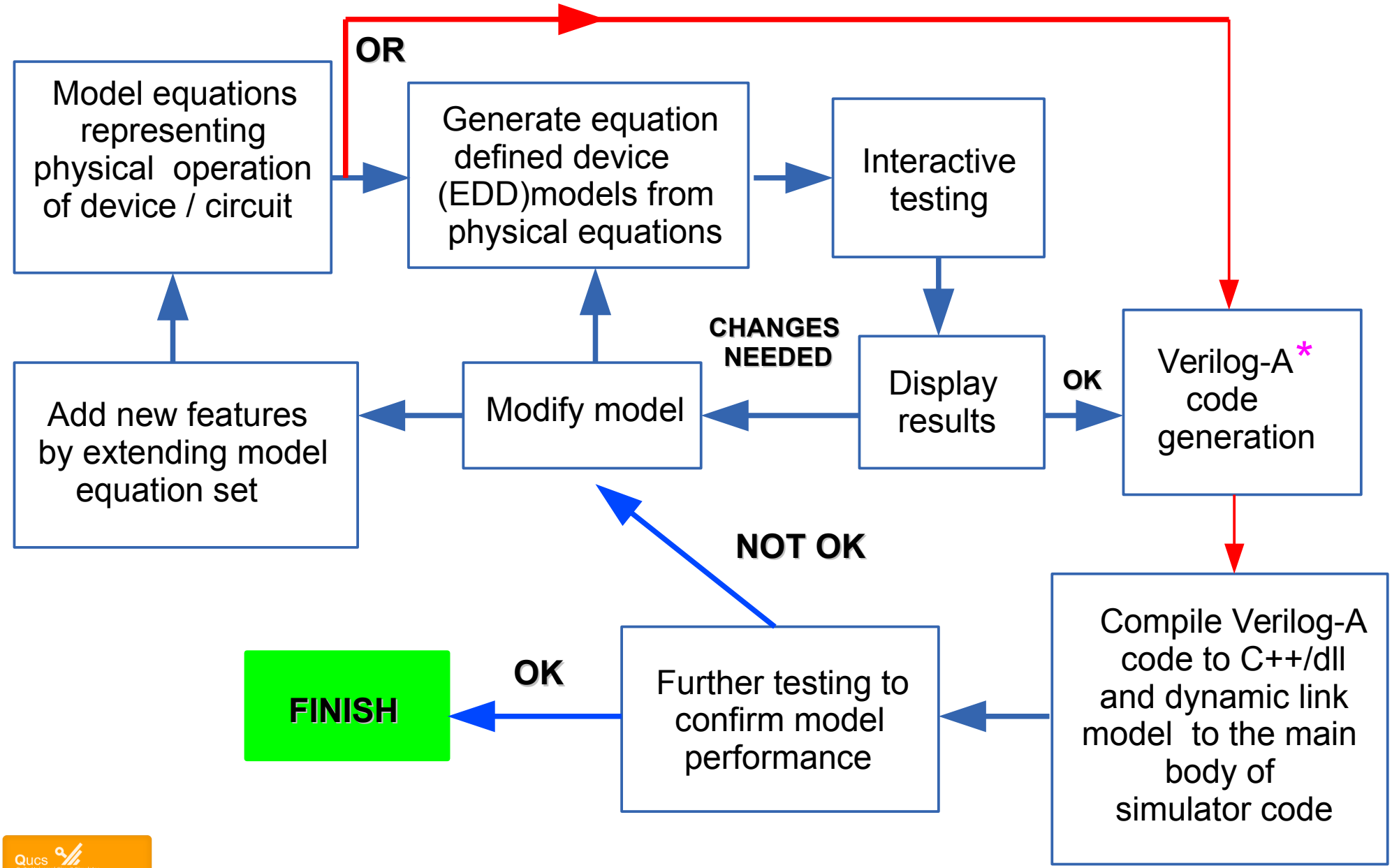




Single pole low pass filter with fp=20kHz



Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 1; model development flow chart



* Only the subset of Verilog-A presented in previous slides allowed

Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 2; simplified EPFL-EKV v.26 long channel equations

Symbol

$$Vg = V(gate) - V(bulk),\quad Vs = V(source) - V(bulk),\quad Vd = V(drain) - V(bulk)$$

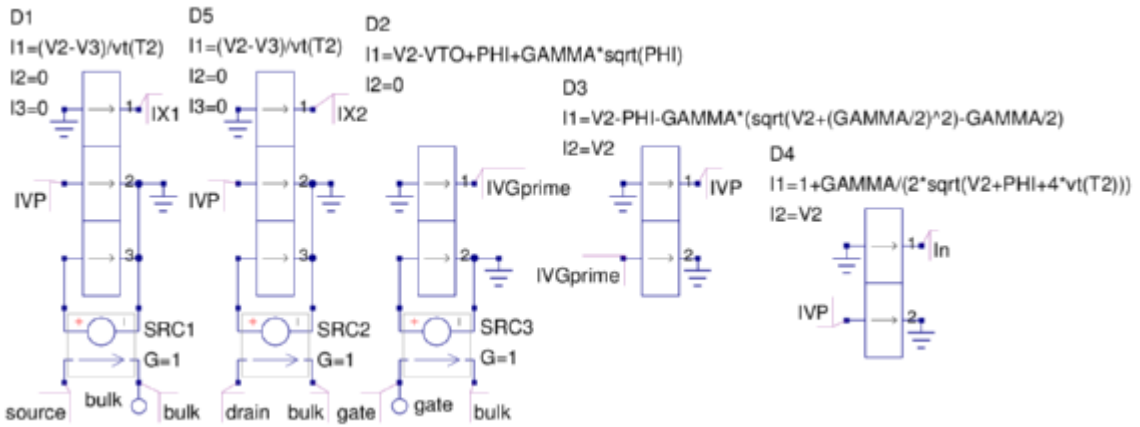
$$VGprime = Vg - VTO + PHI + GAMMA \cdot \left[\sqrt{VGprime + \left(\frac{GAMMA}{2}\right)^2} - \frac{GAMMA}{2} \right]$$

$$n = 1 + \frac{GAMMA}{2} \cdot \sqrt{VP + PHI + 4 \cdot Vt},\quad BETA = KP \cdot \frac{W}{L} \cdot \frac{1}{1 + THETA \cdot VP}$$

$$X1 = \frac{Vp - Vs}{Vt},\quad If = \left[\ln(1 + limexp(X1/2))\right]^2,\quad X2 = \frac{Vp - Vd}{Vt},\quad Ir = \left[\ln(1 + limexp(X2/2))\right]^2$$

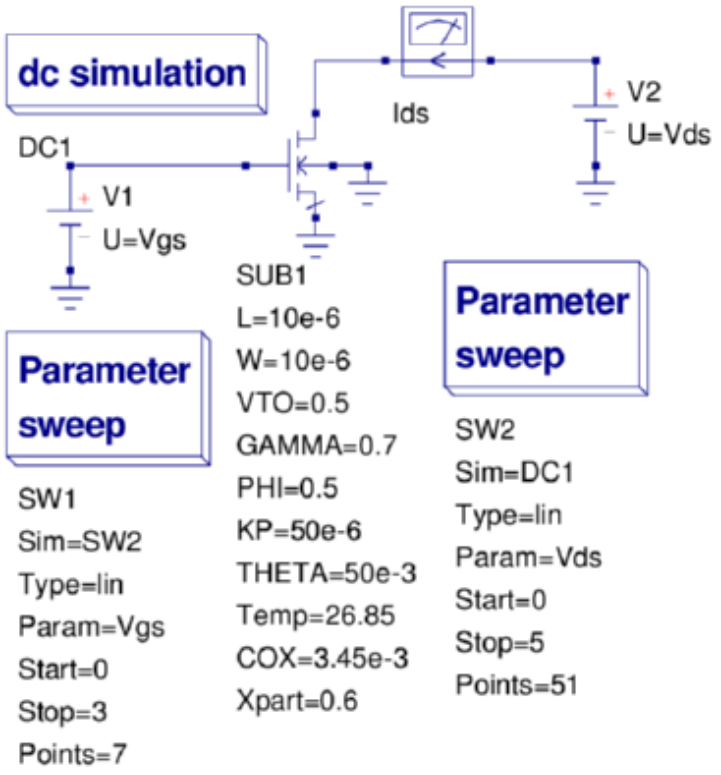
$$Ispecific = 2 \cdot n \cdot BETA \cdot VT^2,\quad Ids = Ispecific \cdot (If - Ir)$$

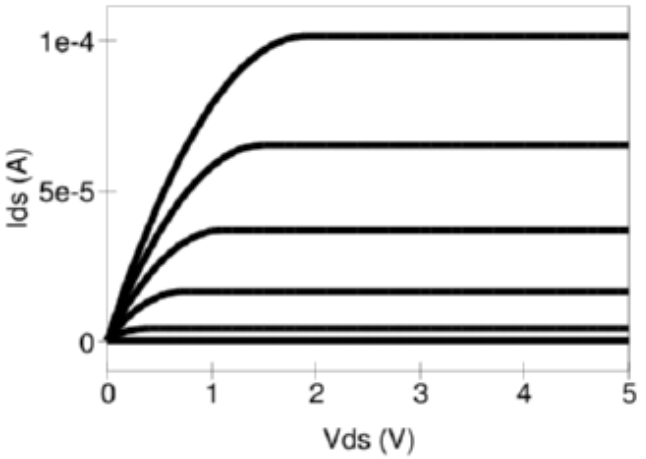
EDD Model



Non-linear I/V model

Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 3; EPFL-EKV v.26 long channel Ids/Vds EDD model interactive testing

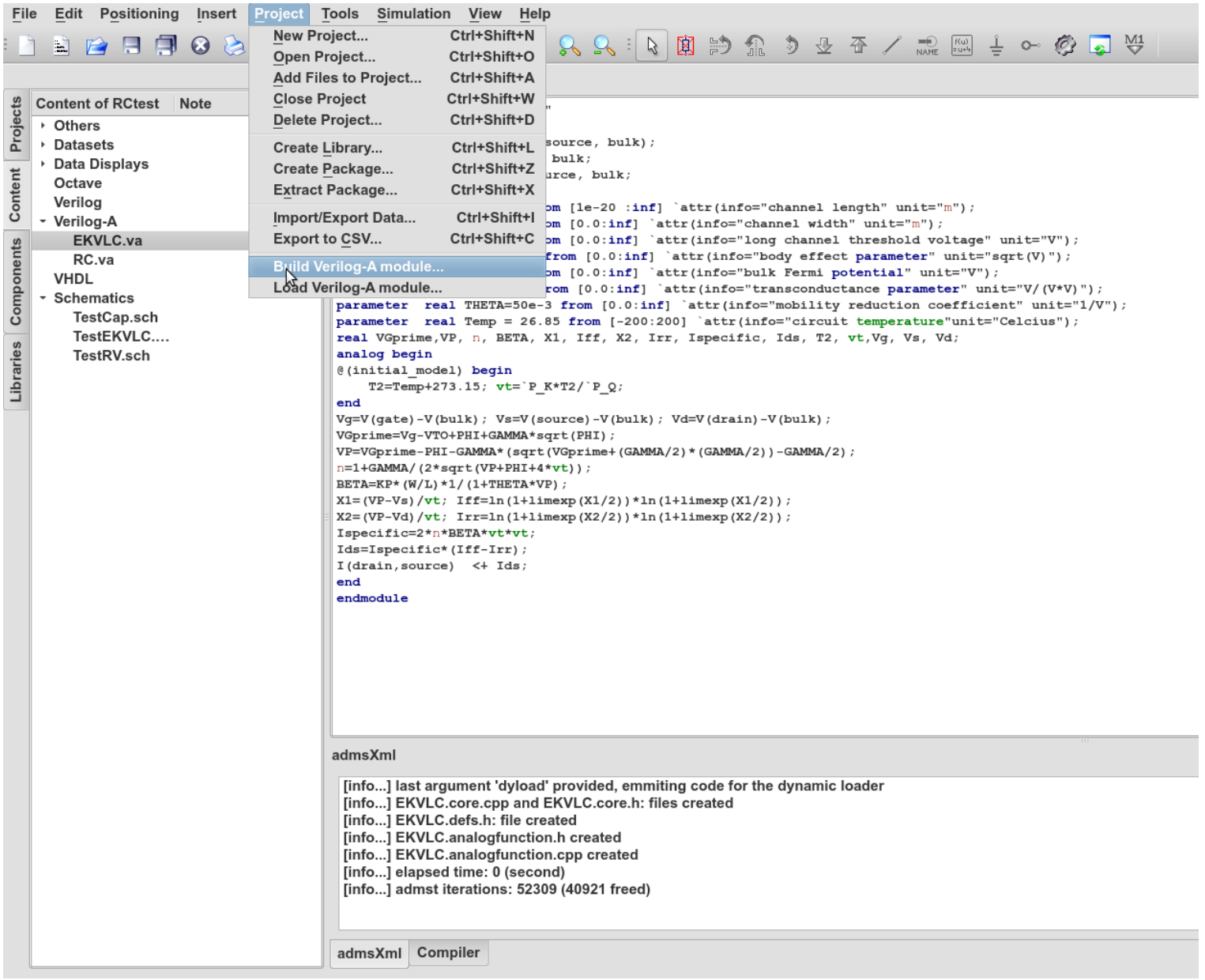




Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 4a; develop EPFL-EKV v.26 long channel Ids/Vds Verilog-A code model and enter it into Qucs with the built in text editor

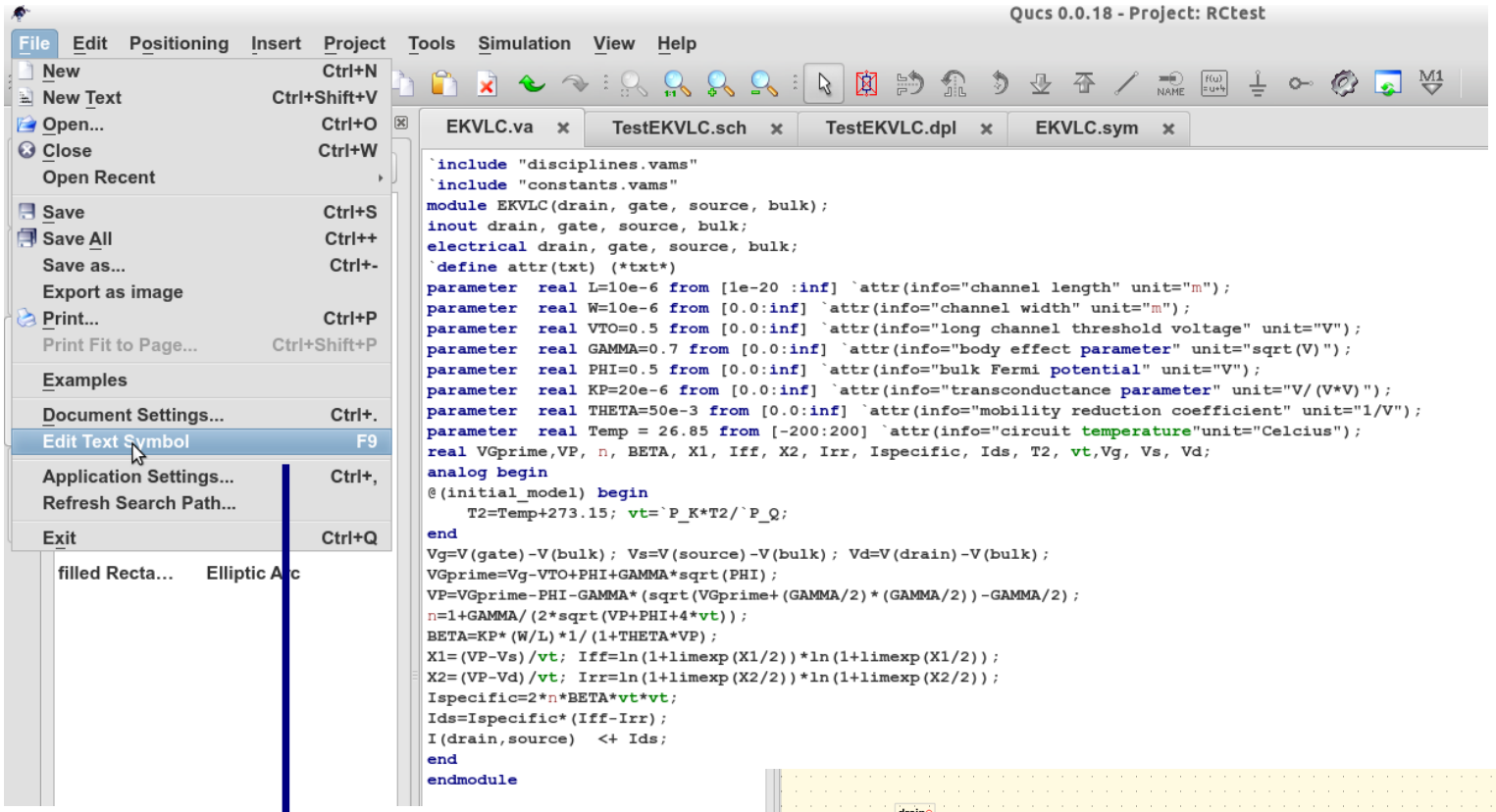
```
`include "disciplines.vams"
`include "constants.vams"
module EKVLC(drain, gate, source, bulk);
  inout drain, gate, source, bulk;
  electrical drain, gate, source, bulk;
  `define attr(txt) (*txt*)
  parameter  real L=10e-6 from [1e-20 :inf] `attr(info="channel length" unit="m");
  parameter  real W=10e-6 from [0.0:inf] `attr(info="channel width" unit="m");
  parameter  real VTO=0.5 from [0.0:inf] `attr(info="long channel threshold voltage" unit="V");
  parameter  real GAMMA=0.7 from [0.0:inf] `attr(info="body effect parameter" unit="sqrt(V)");
  parameter  real PHI=0.5 from [0.0:inf] `attr(info="bulk Fermi potential" unit="V");
  parameter  real KP=20e-6 from [0.0:inf] `attr(info="transconductance parameter" unit="V/(V*V)");
  parameter  real THETA=50e-3 from [0.0:inf] `attr(info="mobility reduction coefficient" unit="1/V");
  parameter  real Temp = 26.85 from [-200:200] `attr(info="circuit temperature"unit="Celcius");
  real VGprime,VP, n, BETA, X1, Iff, X2, Irr, Ispecific, Ids, T2, vt,Vg, Vs, Vd;
  analog begin
     @(initial_model) begin
         T2=Temp+273.15; vt=`P_K*T2/`P_Q;
     end
     Vg=V(gate)-V(bulk); Vs=V(source)-V(bulk); Vd=V(drain)-V(bulk);
     VGprime=Vg-VTO+PHI+GAMMA*sqrt(PHI);
     VP=VGprime-PHI-GAMMA*(sqrt(VGprime+(GAMMA/2)*(GAMMA/2))-GAMMA/2);
     n=1+GAMMA/(2*sqrt(VP+PHI+4*vt));
     BETA=KP*(W/L)*1/(1+THETA*VP);
     X1=(VP-Vs)/vt; Iff=ln(1+limexp(X1/2))*ln(1+limexp(X1/2));
     X2=(VP-Vd)/vt; Irr=ln(1+limexp(X2/2))*ln(1+limexp(X2/2));
    Ispecific=2*n*BETA*vt*vt;
    Ids=Ispecific*(Iff-Irr);
    I(drain,source)  <+ Ids;
  end
endmodule
```

Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 4b; build EPFL-EKV v.26 long channel Ids/Vds Verilog-A code model

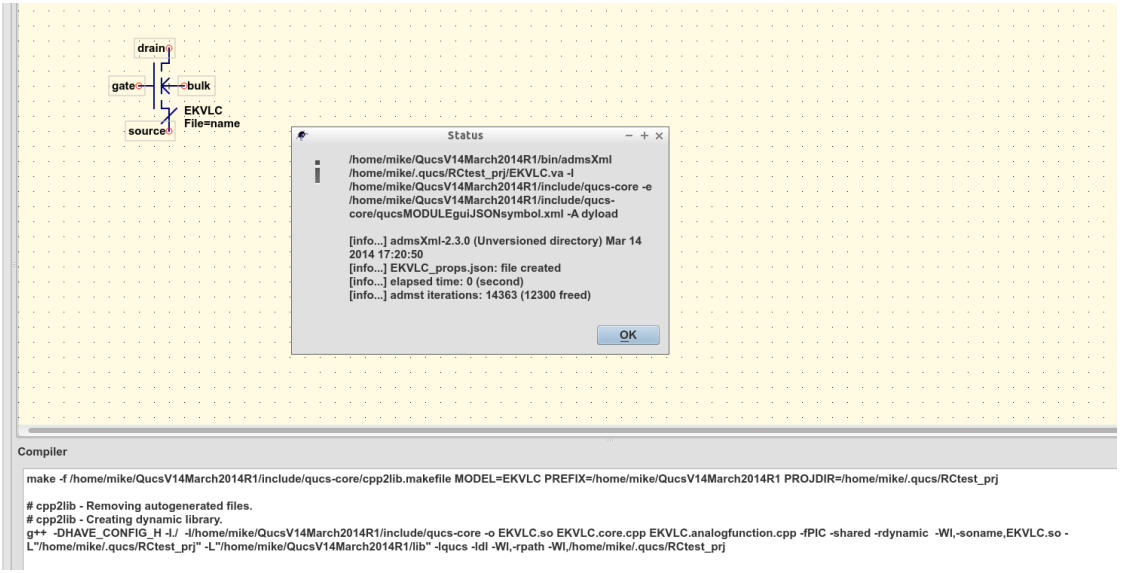


Section of build log

Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 4c; generate EPFL-EKV v 2.6 long channel Ids/Vds Verilog-A model symbol

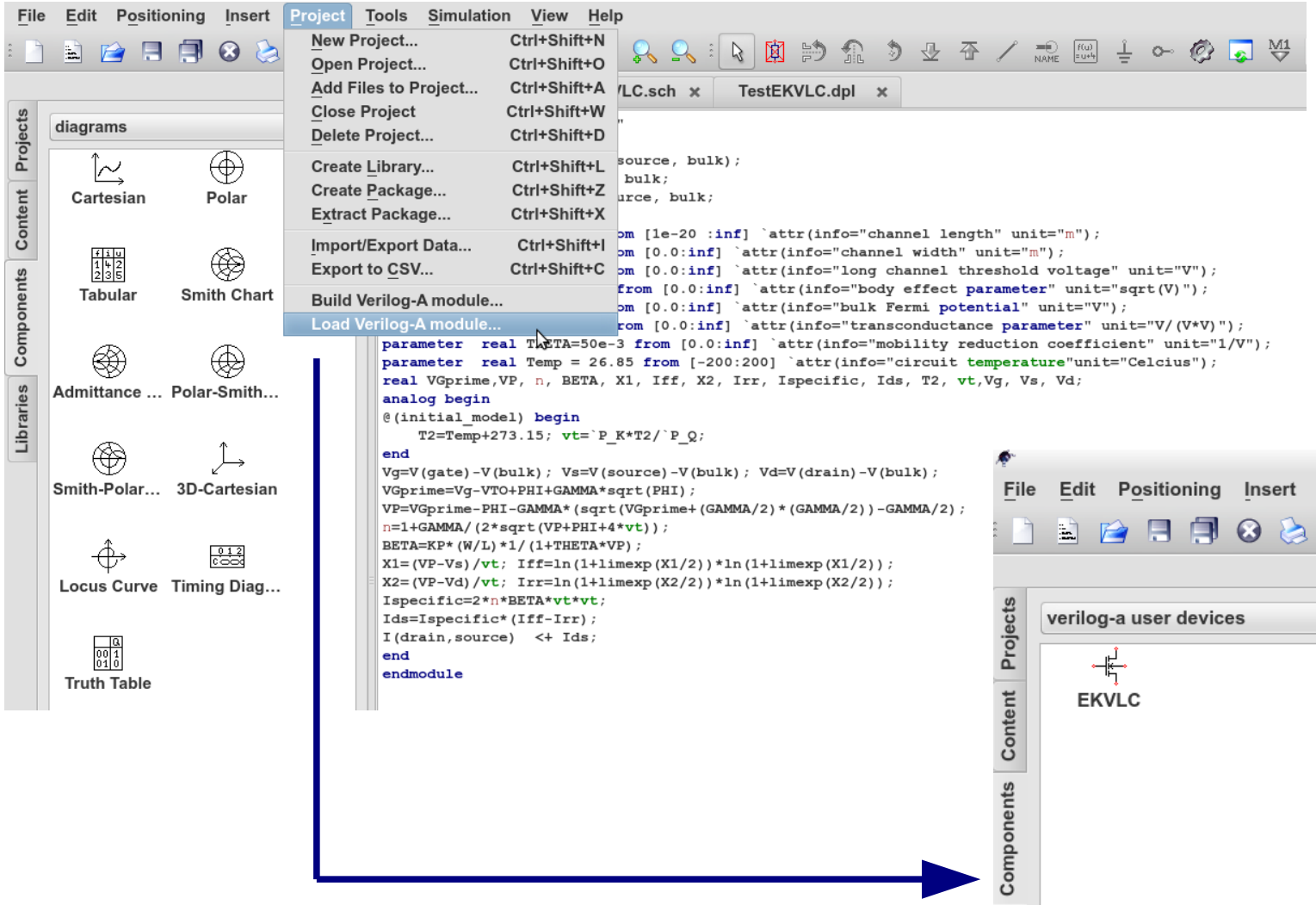


Generate schematic symbol for xxx.va



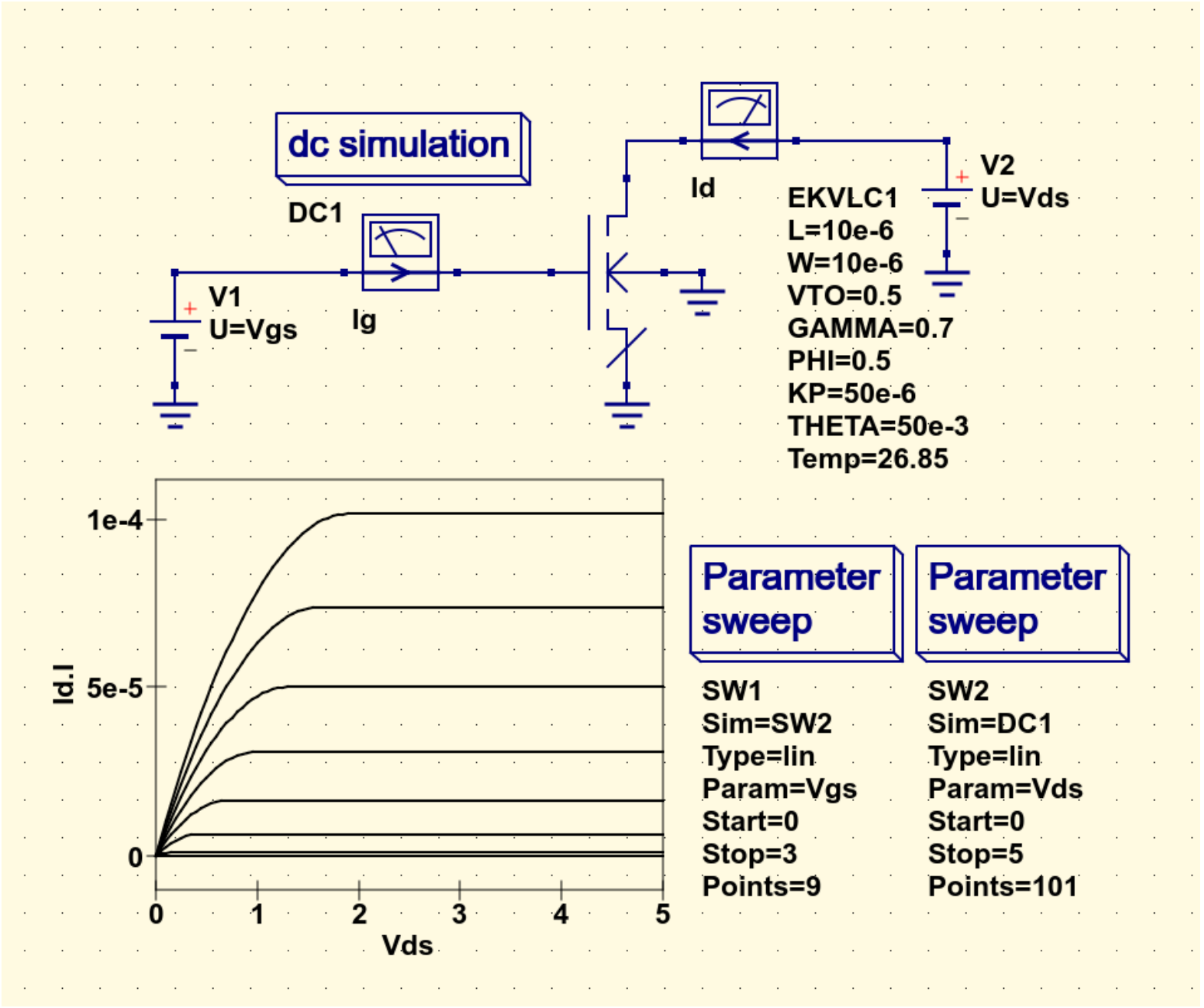
Same procedure as a sucircuit

Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 4d; load EPFL-EKV v.26 long channel Verilog-A Ids/Vds model

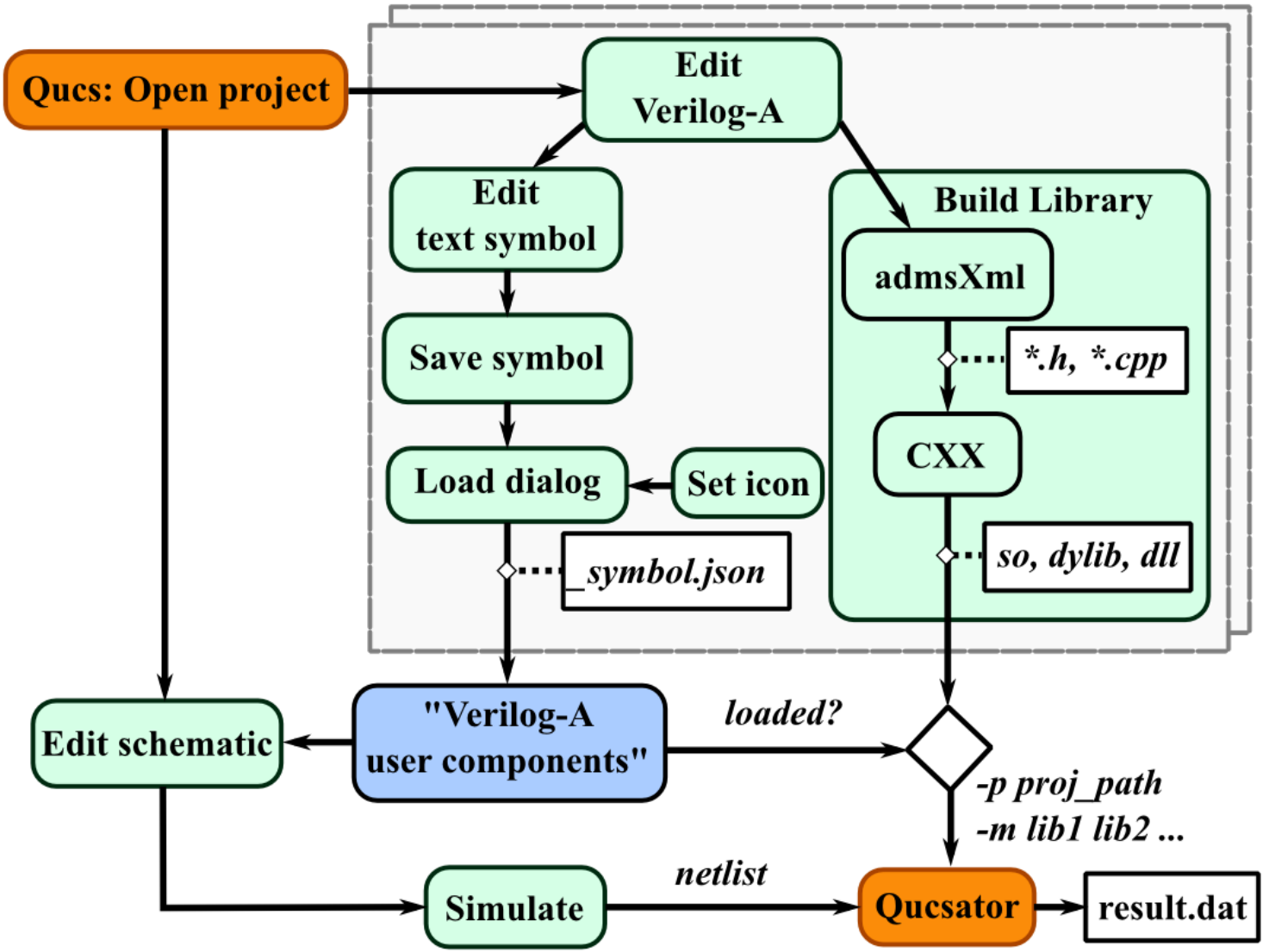


One, or more, Verilog-A modules loaded into the "Verilog-A user devices" window

Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 5; EPFL-EKV v.26 long channel Verilog-A Ids/Vds model interactive testing



Introduction to compact device modelling with the Qucs EDD blocks, ADMS Verilog-A code and Qucs dynamic linking loader: Part 6; Qucs Verilog-A model generation flow chart



Future directions

The next release of Qucs for the Linux, MAC OS and PC Windows is targeted for early summer 2014

Short Term

- Finish Qucs GUI Qt3 to Qt4 move then convert to Qt5
- New implementation of matrix calculations using Libeigen3
- Expand and unify the number of Qucs component and device models
- Improve Qucs analogue simulation engine
- Improve ADMS/Verilog-A model synthesis via dynamic model loading
- Increase the coverage of Verilog-A(MS) statements
- Stabilise Qucs/Octave simulation link
- Improve Qucs/Python link
- Improve Qucs documentation

Long Term

- Introduce some sort of parallelism into Qucs-core
- Add PCB facilities to Qucs
- Rewrite Qucs GUI to improve overall performance
- Add animation to Qucs
- Enable mixed-signal co-simulation (tight coupling of Qucs/ Qucsator with Icarus Verilog or Verilator)
- Extend Qucsconv to work with later SPICE versions, for example Xyce and ngspice
- Add circuit design routines to Qucs component netlists
- Move to Graphics View Framework
- Re-factor Qucs code base to improve readability and
- maintainability

Summary

- Qucs is a freely available circuit simulators distributed as open source software under the GNU General Public Licence (GPL).

- This presentation has attempted to outline the history and the fundamental features of the package, the available equation-defined components, built in modelling aids, analysis types and post-simulation data analysis and visualisation capabilities.

- A series of slides reviewed the current position of the ADMS Verilog-A model synthesiser, describing its implementation in the current Qucs release. The talk also introduced how ADMS can be used to develop equation-defined component models of established and emerging technology devices.

- An outline of a number of proposed future developments was presented at the end of the talk.

Qucs is freely available under the open source General Public Licence.
Download from: Qucs version 0.0.18, http://qucs.sourceforge.net

The current development version of Qucs which includes the new Verilog-A/ADMS compact modelling system, including the Qucs dynamic-loader source code can be downloaded from http://sourceforge.net/p/qucs/git/ci/dynamic-loader-rebase140211/tree/