
Qucs-S Help Documentation

Release 0.0.19-S

Contributing Authors 2015, 2016

Oct 07, 2017

Contents

1 User Manual and Reference Material

3



User Manual and Reference Material

Authors Mike Brinson (mbrin72043@yahoo.co.uk) and Vadim Kusnetsov (ra3xdh@gmail.com)

Copyright 2015, 2016

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Contents:

Chapter 1. Introduction

Following the release of Qucs-0.0.18 in August 2014 the Qucs Development Team considered in detail a number of possible directions that future versions of the software could take. Spice4qucs is one of these routes. It addresses a number of problems observed with the current version of Qucs while attempting to combine some of the best features of other GPL circuit simulation packages. The project also aims to add additional model development tools to those currently available in Qucs-0.0.18. Qucs was originally written as an RF and microwave engineering design tool which provided features not found in SPICE, like S parameter simulation, two and multiport small signal AC circuit analysis and RF network synthesis. Since its first release under the General Public License (GPL) in 2003 Qucs has provided users with a relatively stable, flexible and reasonably functional circuit simulation package which is particularly suited to high frequency circuit simulation. In the years following 2003 the Qucs Development team added a number of additional simulation facilities, including for example, transient simulation, device parameter sweep capabilities and single tone Harmonic Balance simulation, making Qucs functionality comparable to SPICE at low frequencies and significantly extended at high frequencies. Considerable effort has also been made to improve the

device modelling tools distributed with Qucs. The recent versions of the software include code for algebraic equation manipulation, Equation-Defined Device (EDD) modelling, Radio Frequency Equation-Defined Device (RFEDD) simulation and Verilog-A synthesised model development plus a range of compact and behavioural device modelling and post simulation data analysis tools that have become central features in an open source software package of surprising power and utility.

One of the most often requested new Qucs features is “better documentation”, especially documentation outlining the use and limitations of the simulation and the modelling features built into Qucs. Qucs is a large and complex package which is very flexible in the way that it can be used as a circuit design aid. Hence, however much documentation is written describing its functionality there are always likely to be simulation and modelling examples that are missing from the Qucs documentation. In future Qucs releases will be accompanied by two or more basic Qucs documents. The first of these, simply called “Qucs-Help”, provides introductory information for beginners and indeed any other users, who require help in starting to use Qucs. The second Qucs document, called “Spice4qucs-Help”, introduces more advanced simulation and modelling topics. Both documents present a large number of typical circuit simulation and compact device modelling examples. In the text these are also linked to sets of Qucs reference material. Since 2003 the Qucs Development Team, and other authors, have published a body of work concerning Qucs and its applications. A bibliography of these publications can be found at the end of this document. Anyone interested in learning about Qucs is recommended to read these publications as they provide a wealth of information on basic and advanced Qucs topics. Both the “Qucs_Help” and “Spice4qucs-Help” documents are very much work in progress. Updated versions will be published with each new Qucs release. Moreover, to keep everyone up to date with Qucs current developments it is planned to update them during future Qucs development phases. The latest documentation can be found at <https://github.com/Qucs/qucs-help>.

1.1 Background

The current rate of Qucs downloads from <http://qucs.sourceforge.net/> is around 3000 per week with a total of roughly 1.5 million downloads since Qucs was first released in 2003. This figure does not however, imply that 1.5 million copies of Qucs are currently in regular use. It does perhaps give an indication of the popularity of the software, particularly amongst users interested in RF circuit design and simulation. For a high percentage of regular Qucs users the current distribution version (Qucs 0.0.18) provides a reasonably stable and accurate simulation tool equipped with more than enough facilities to meet their everyday needs. In reality Qucs is not perfect but includes a number of weaknesses and known limitations. The onward march of technology also makes it imperative that Qucs evolves from a traditional circuit simulator to a design tool that can handle modelling and simulation of circuits which include devices from new emerging technologies. Today Qucs includes an extensive range of component, device and circuit modelling tools, allowing it to model and simulate circuits built from standard components and new production devices.

Three of the primary aims of the work undertaken by the Qucs Development Team are firstly to remove software bugs and improve the overall performance of the package, secondly to address known weaknesses and limitations and thirdly to develop the package by adding features which increase its utility. Readers who are not familiar with open source software development may be unaware of how the development process works. By “Qucs Development Team” we mean a group of interested individuals who freely give both their time and expertise for the improvement of the GPL Qucs package. The Qucs Development Team is not a fixed group but is a dynamic organisation where different people contribute, simultaneously or at different times, to the same part or different parts of the software. The spice4qucs project group is one of a number of sub-groups within the overall Qucs Development Team. It was formed to address the known limitations of the previous Qucs releases and to take advantage of the work done by other GPL circuit simulation teams working on the Ngspice (<http://ngspice.sourceforge.net/>), Xyce (<https://xyce.sandia.gov/>) and SPICE OPUS (<http://www.spiceopus.si/>) circuit simulators.

The spice4qucs initiative is an ongoing project which attempts to:

1. Correct known weaknesses observed with the current Qucs analogue simulation engine “qucsator”. Qucsator is based on classical numerical mathematics routines for the solution of electrical network linear and non-linear real and complex algebraic equations and time domain algebraic and differential equations. For small circuits, qucsator works well in the DC and AC small signal domains. However, in the transient and Harmonic Balance simulation domains it can fail to converge to an acceptable solution. Its performance is also often below that

expected of a modern circuit simulator employing sparse matrix algorithms. However, qucsator works well for RF small signal AC simulation and will for some time remain the first choice for this simulation domain.

2. Provide Qucs users with a choice of simulation engine selected from qucsator, Ngspice, Xyce and SPICE OPUS. By selecting Ngspice, Xyce or SPICE OPUS as the Qucs simulation engine users may capitalise on all the features offered by the extensive SPICE developments which have taken place over the last forty years. Both Ngspice, Xyce and SPICE OPUS offer improved transient simulation convergence and speed, particularly for large non-linear circuits. Xyce brings an alternative implementation of single tone Harmonic Balance simulation to Qucs which offers much improved convergence properties for both linear and non linear components and devices. The latest version of Xyce, 6.5 at the time of writing, also offers multi-tone Harmonic Balance simulation. SPICE OPUS adds transient shooting methods for the steady state analysis of large signal AC simulation and optimization.
3. Extend Qucs subcircuit, Equation-Defined Device (EDD), Radio Frequency Equation-Defined Device (RFEDD) and Verilog-A device modelling capabilities. The latest spice4qucs release (Qucs-0.0.19-S) offers much improved component and device modelling features that work as interlinked tools, supporting model development as a continuous flow from physical concept to compiled C/C++ code. This feature is centred around a “turn-key” version of the XSPICE Code Model construction tools. If required the spice4qucs project can also use the Berkeley “Model and Algorithm Prototyping Platform” (MAPP <http://draco.eecs.berkeley.edu/dracotiki/tiki-index.php?page=MAPP>) for compact model construction. It is also possible to synthesise Ngspice, Xyce SPICE OPUS SPICE netlists from Qucs EDD and RFEDD models and to synthesise Verilog-A models from Qucs EDD and SPICE B components.
4. Offer Qucs users access to the additional simulation tools and extra component and device models provided by Ngspice, Xyce and SPICE OPUS. This includes much improved component library facilities which allow the use of device manufacturers SPICE models and XSPICE Code Models.
5. Offer for the first time with Qucs a true mixed-mode analogue-digital circuit simulation capability using Qucs/Ngspice/SPICE OPUS/XSPICE simulation.

The spice4qucs initiative is an on going project and must be considered as very much work in progress. In its early releases not all the features listed above will be available for public use. It is however, the intention of the Qucs Development Team to introduce them as quickly as possible. Other features not listed in the previous entries may also be introduced.

1.2 Qucs-0.0.18 Structure

A block diagram showing the main analogue modelling and simulation functions of the Qucs-0.0.18 package is illustrated in Figure 1.1. For convenience, particularly easy identification, blocks with similar modelling or similar simulation functions have been coded with identical colours, for example dark red indicates the GUI and qucsator analogue simulation engine and dark green major component and device modelling tools. The direction of the flow of data between blocks are also shown with directed arrows. Central to the operation of the Qucs-0.0.18 package is the Qucs graphical user interface (GUI), the qucsator simulation engine and a post simulation data processing feature (indicated by the yellow block in Figure 1.1) for the extraction of device and circuit parameters and the visualisation of simulated signal waveforms. Cyan blocks in Figure 1.1 identify the well known Octave numerical analysis package (<https://www.gnu.org/software/octave/>). Qucs employs Octave for additional post simulation data processing and waveform visualisation plus an experimental circuit simulation process where qucsator and Octave undertake cooperative transient circuit simulation (cyan coloured blocks). The single light brown block in Figure 1.1 represents the ASCO optimisation package which is used by Qucs for determining circuit component values and device parameters which result in specific circuit performance criteria.

Readers who are not familiar with the basic operation and use of the Qucs GUI, circuit simulator and output processing routines should consult the “Qucs-Help” document before proceeding further with this more advanced document.

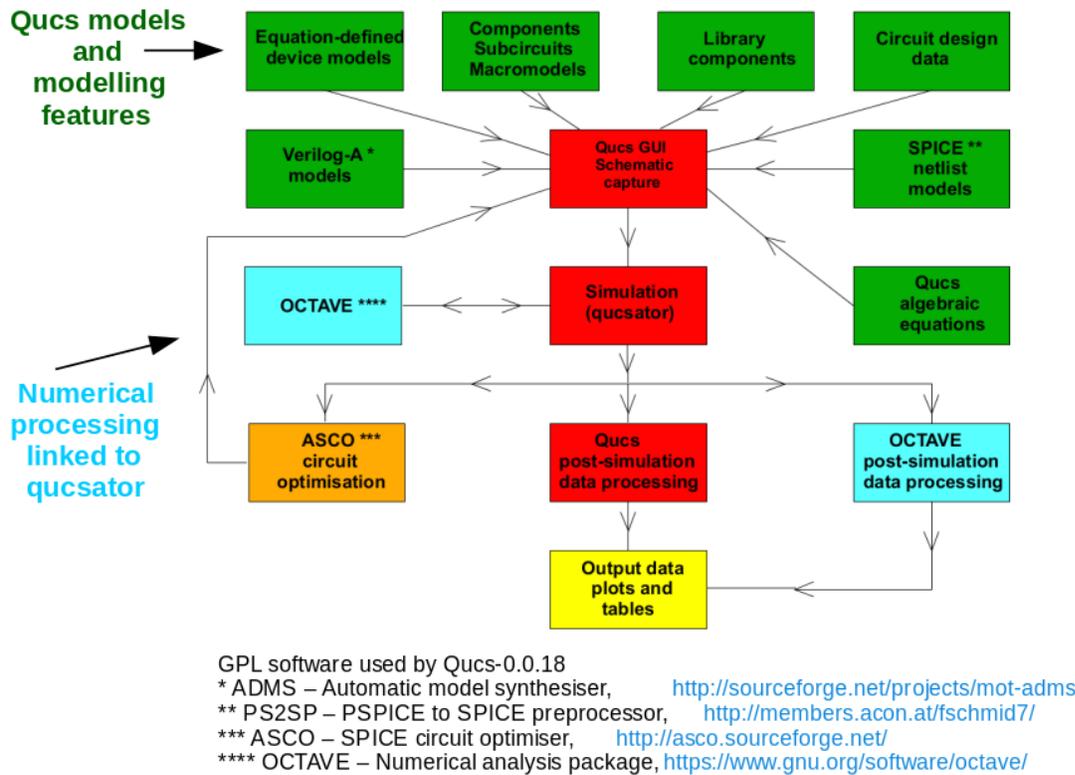


Figure 1.1. A block diagram showing the analogue modelling and simulation facilities provided by Qucs-0.0.18.

1.3 Qucs future capabilities

Figure 1.2. presents an extended version of the Qucs-0.0.18 functional diagram where the added blocks indicate areas chosen for current and future Qucs development. Two major extension to Qucs functionality are obvious, namely the addition of the Ngspice, Xyce and SPICE OPUS circuit simulators to the Qucs package and the increase in the Qucs device modelling capabilities through the addition of the XSPICE Code Modelling software. Figure 1.2. only gives a rough picture of the proposed changes to Qucs under development by the spice4qucs project. Much of the detail will become clearer later in the manual text and reference sections.

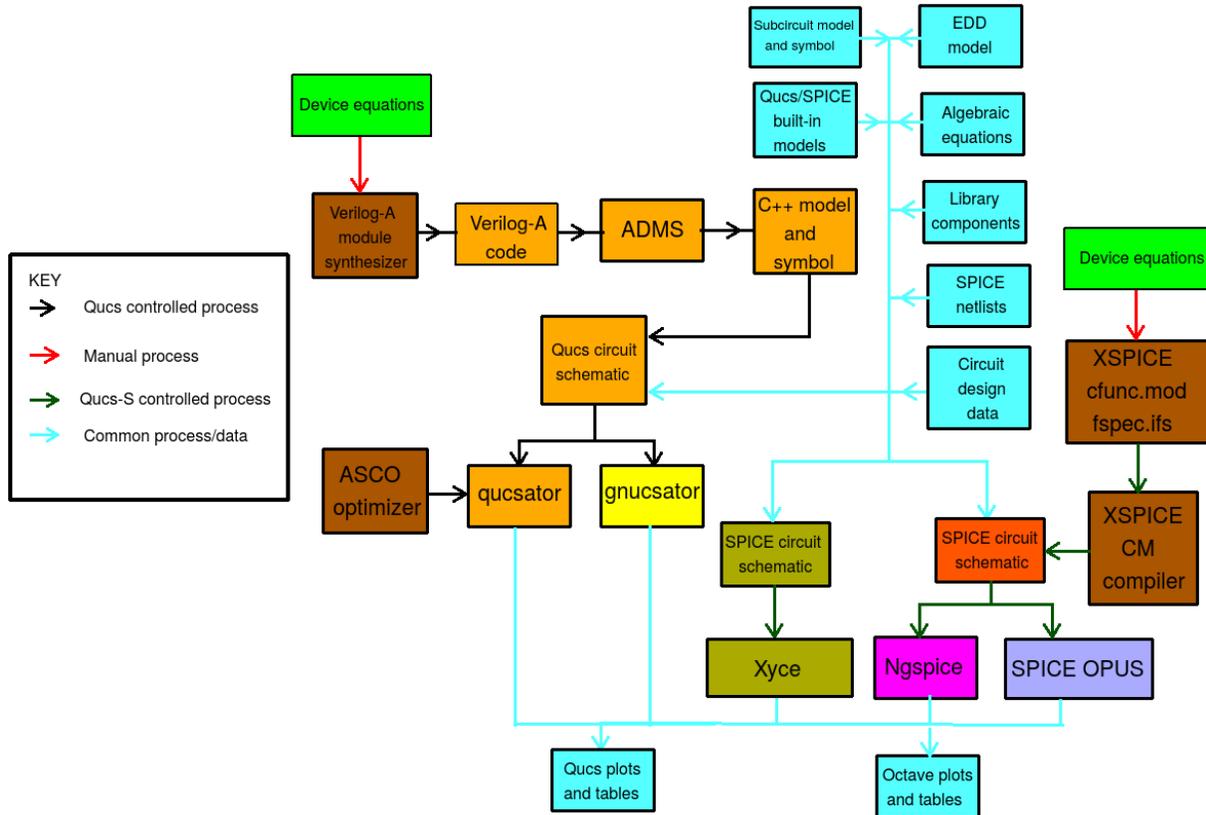


Figure 1.2. An block diagram outlining the extended Qucs-S simulation and modelling tools under development by the spice4qucs initiative.

1.4 A first view of the extended spice4qucs device modelling and simulation features

At this point it seems appropriate to introduce a short example which demonstrates how much Qucs has evolved since the release of version 0.0.18. This example has been deliberately chosen to present an overview of the major new Qucs features either already developed by the spice4qucs project or planned for future releases. To provide readers with adequate information on how to make the best use of the new spice4qucs features they are described in detail in later chapters of this document.

Qucs version 0.0.18 is a surprisingly sophisticated program with quite a number of hidden features which are not obvious to most Qucs users. Given in Figure 1.3 is a Qucs schematic which demonstrates a little known application of the circuit simulator. Qucs is ideal for developing high level behavioural models of new components or devices which are not implemented in the distributed software. The schematic in Figure 1.3. introduces the physical equations and device parameters for a semiconductor tunnel diode. By using the Qucs parameter sweep and DC simulation operations it is possible to scan the diode bias voltage V_{pn} , calculate the tunnel diode bias current I_{pn} at each bias point and plot the device $i_d = f(v_d)$ characteristics. Note that in this introductory example the Qucs schematic does not include any electrical components. Moreover, the tunnel diode current is calculated directly from its physical **model_equations** and **model_parameters**.

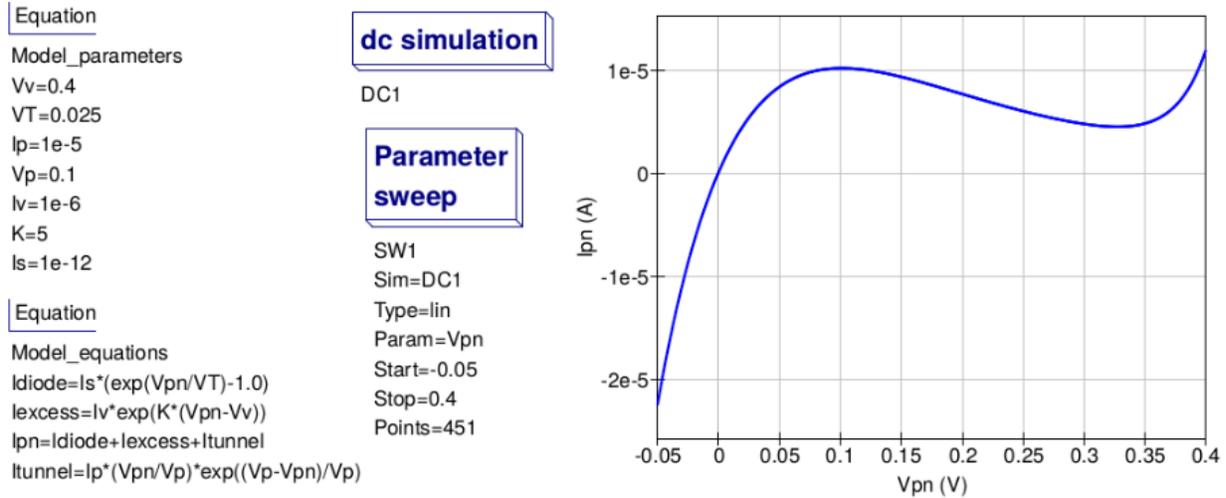


Figure 1.3. Mathematical representation of $I_d = f(V_d)$ for a semiconductor tunnel diode, including device model_parameters, model_equations and a Qucs DC scan test.

The Berkeley **Model and Algorithm Prototyping Platform** (MAPP) is a new GPL modelling and simulation tool. It is developed by the MAPP team at the Department of Electrical Engineering and Computer Science, University of California, Berkeley using a MATLAB (copyright) subset common to the Octave numerical analysis software. As part of the spice4qucs project the MAPP software has been interfaced with the Qucs GUI. Figure 1.4 introduces a MAPP behavioural model for the tunnel diode in Figure 1.3. Notice how similar the models in Figures 1.3 and 1.4 are. MAPP circuit simulation results in the diode characteristic plotted in Figure 1.4.

MAPP tunnel diode compact device model

```
function MOD = tunnelDiode_ModSpec_wrapper()
    MOD = ee_model();
    MOD = add_to_ee_model(MOD, 'external_nodes', {'p', 'n'});
    MOD = add_to_ee_model(MOD, 'explicit_outs', {'ipn'});
    MOD = add_to_ee_model(MOD, 'parms', {'ls', 1e-12, 'VT', 0.025});
    MOD = add_to_ee_model(MOD, 'parms', {'lp', 1e-5, 'Vp', 0.1});
    MOD = add_to_ee_model(MOD, 'parms', {'lv', 1e-6, 'Vv', 0.4, 'K', 5});
    MOD = add_to_ee_model(MOD, 'parms', {'C', 0});
    MOD = add_to_ee_model(MOD, 'f', @f);
    MOD = add_to_ee_model(MOD, 'q', @q);
    MOD = finish_ee_model(MOD);
end

function out = f(S)
    v2struct(S);
    I_diode = ls*(exp(vpn/VT)-1);
    I_excess = lv * exp(K * (vpn - Vv));
    I_tunnel = (lp/Vp) * vpn * exp(-1/Vp * (vpn - Vp));
    out = I_diode + I_tunnel + I_excess;
end

function out = q(S)
    v2struct(S);
    out = C*vpn;
end
```

Tunnel diode I/V curve

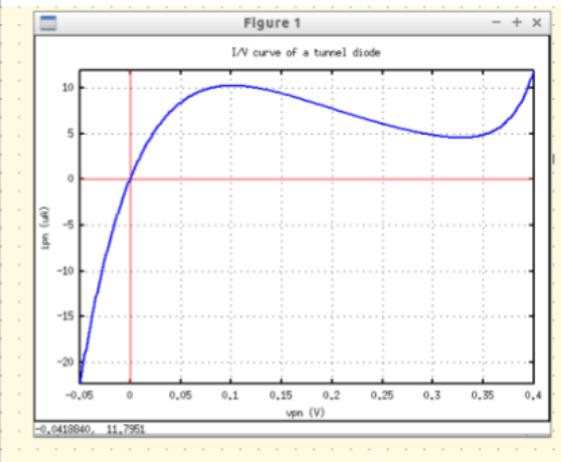


Figure 1.4. MAPP tunnel diode model and simulated diode current as a function of applied bias voltage.

The Qucs and MAPP modelling tools allow models represented by a set of mathematical equations based on the physical properties of a device to be tested and their correct operation confirmed prior to constructing a simulation model for inclusion in circuit schematics. Illustrated in Figure 1.5 is a third model for the tunnel diode plus a test circuit for simulating the device DC current versus voltage characteristics. This model will work with Qucs-0.0.18 and spice4qucs versions of the circuit simulator. It shows how a Qucs EDD model represents the physical model of

the tunnel diode and how this model can be represented with it's own symbol and tested by combining it with other components to form a DC characteristic test circuit. The Qucs EDD is not implemented in SPICE simulators. SPICE 3f5 and later simulators have instead other features like, for example, the B type sources.

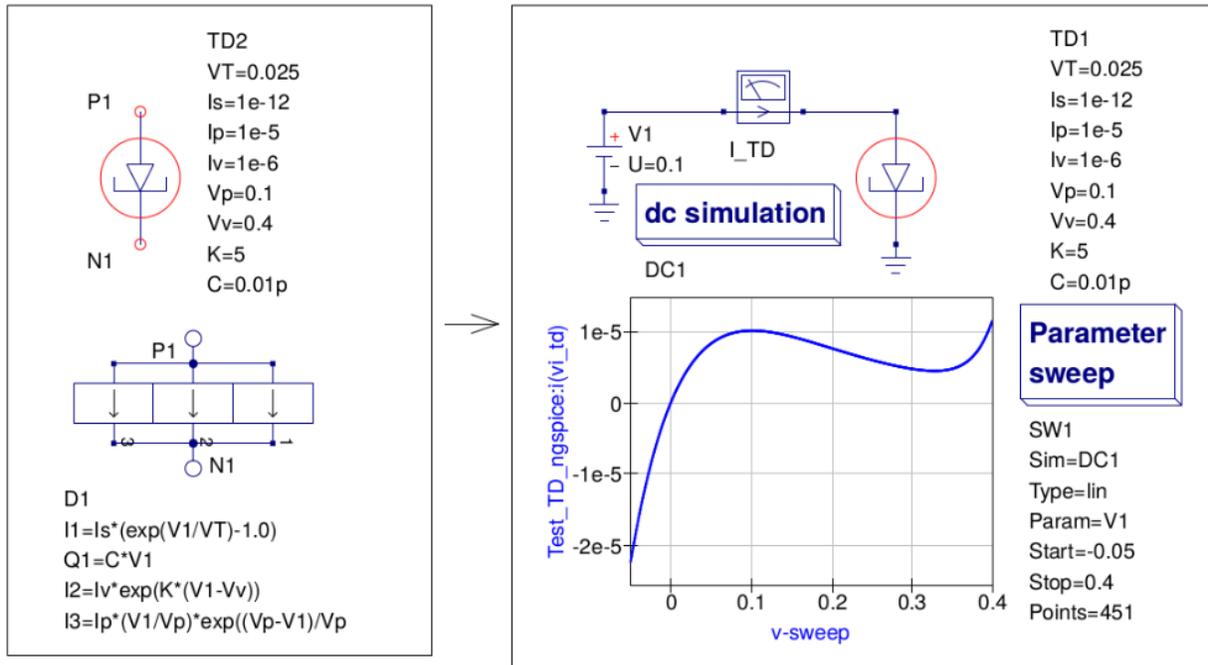


Figure 1.5. Qucs EDD behavioural model for the tunnel diode first introduced in Figure 1.3.

The Qucs EDD component has one feature which makes it particularly important for developing compact device simulation models, namely that its structure and modelling capabilities are similar to those available with the Verilog-A hardware description language. Hence, once an MAPP/Qucs EDD model is operating satisfactorily it can be transcribed into a Verilog-A compact model by inspection or by computer synthesis. Such a Verilog-A model and test circuit is shown in Figure 1.6.

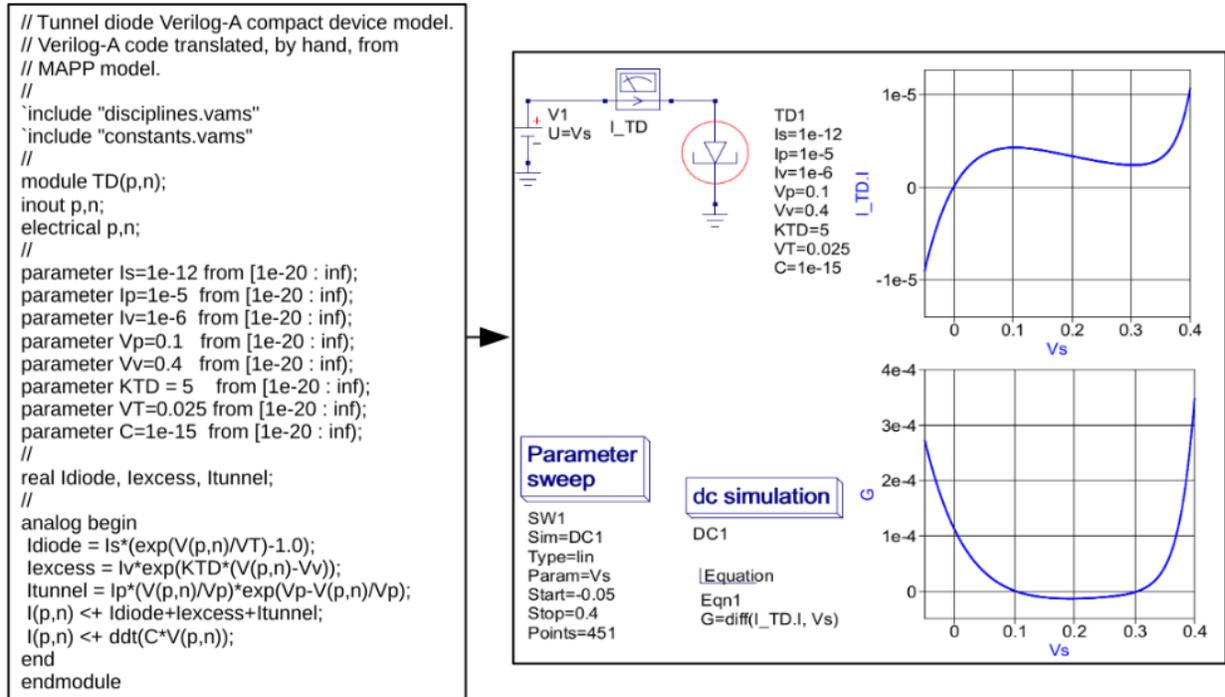


Figure 1.6. A Verilog-A compact tunnel diode model and test circuit.

One of the main aims of the spice4qucs initiative is both to improve the Qucs compact device modelling capabilities and to streamline the flow of information between each part of the modelling and simulation sequence. In all Qucs releases prior to the spice4qucs project a number of modelling tools were implemented in the distribution software but users had to translate manually each type of model format to other formats if they wished to use a model with a different simulator or modelling tool. One exception was the rudimentary translation tool called **qucsconv** for translating SPICE netlists to Qucs netlist format. It was for example not possible to simulate Qucs models encoded in the Qucs netlist format directly with a SPICE simulator or to generate a Verilog-A code model directly from a Qucs EDD model. This situation will change significantly as the spice4qucs project moves forward: in the medium to long term a number of synthesis-translation routines will be added to Qucs making the process of model translation transparent to the Qucs user. The first of these is the link between the Qucs netlist format and the Ngspice, Xyce and SPICE OPUS simulator netlist formats. Figure 1.5 shows a Qucs tunnel diode EDD model, a DC swept parameter test circuit and a set of Ngspice simulation results. Figure 1.7 lists an Ngspice netlist generated automatically by spice4qucs for the circuit shown in Figure 1.5. Notice that this netlist is not simply a list of SPICE component statements but includes an embedded Ngnutmeg script between the SPICE **.control andendc** statements.

```

1 * Qucs 0.0.19
2 * Qucs 0.0.19 TD.sch
3 .SUBCKT TD _net0 _net1 VT=0.025 Is=1e-12 Ip=1e-5 Iv=1e-6 Vp=0.1 Vv=0.4 K=5 C=0.01p
4 BD1I0 _net0 _net1 I=Is*(exp((V(_net0)-V(_net1))/VT)-1.0)
5 GD1Q0 _net0 _net1 nD1Q0 _net1 1.0
6 LD1Q0 nD1Q0 _net1 1.0
7 BD1Q0 nD1Q0 _net1 I=- (C*(V(_net0)-V(_net1)))
8 BD1I1 _net0 _net1 I=Iv*exp(K*((V(_net0)-V(_net1))-Vv))
9 BD1I2 _net0 _net1 I=Ip*((V(_net0)-V(_net1))/Vp)*exp((Vp-(V(_net0)-V(_net1)))/Vp)
10 .ENDS
11 XTD2 _net0 0 TD VT=0.025 Is=1E-12 Ip=1E-5 Iv=1E-6 Vp=0.1 Vv=0.4 K=5 C=0.01P
12 VI_TD1 _net1 _net0 DC 0 AC 0
13 V1 _net1 0 DC 0.1
14 .control
15 set filetype=ascii
    
```

```

16 DC V1 -0.05 0.4 0.000997783
17 write _dc.txt VI_TD1#branch
18 destroy all
19 exit
20 .endc
21 .END
    
```

Figure 1.7. A synthesized Ngspace netlist for the tunnel diode circuit shown in Figure 1.5.

Figure 1.8, and the associated model code, introduce a user defined XSPICE Code Model for the tunnel diode example. A recent extension to the spice4qucs compact device modelling capabilities adds a “turn-key” feature which allows user defined XSPICE Code Models to be added to Qucs and automatically compiled to C code by the package. More on this topic and all the others introduced earlier in this chapter can be found in later sections of this document.

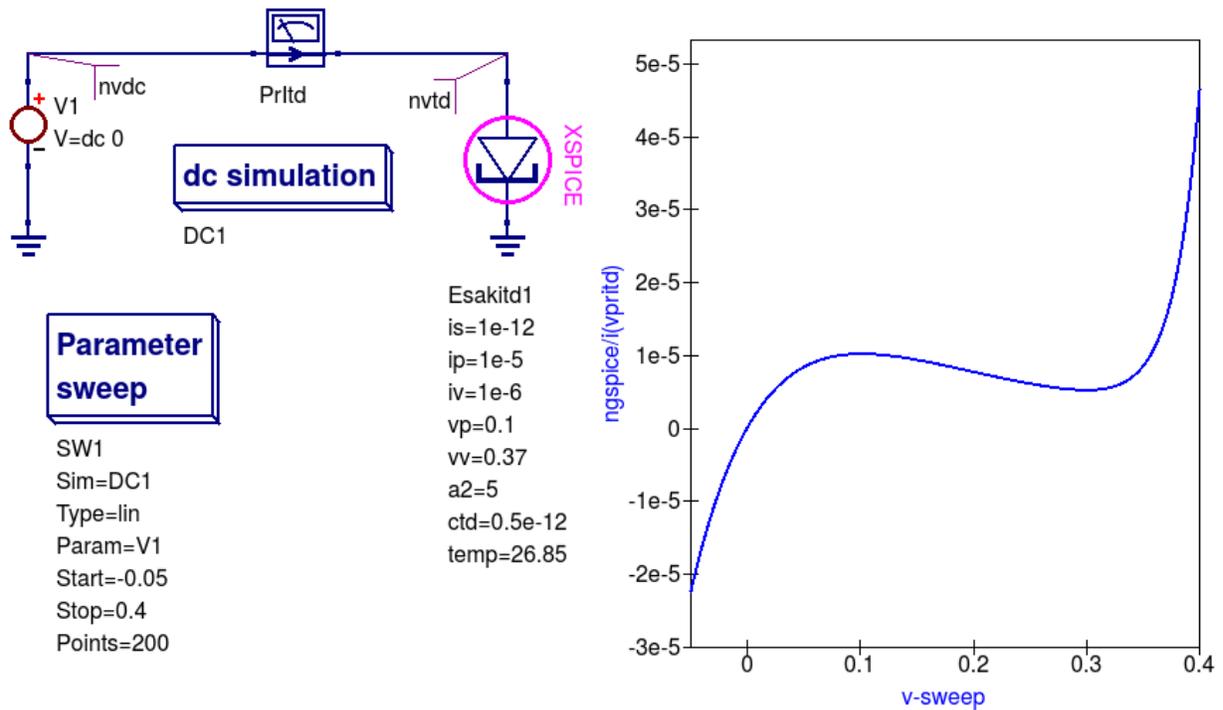


Figure 1.8. XSPICE Code Model tunnel diode model, test circuit and Ngspace simulation results.

```

1 /*
2   etd cm model.
3
4   2 April 2016   Mike Brinson
5
6
7   This is free software; you can redistribute it and/or modify
8   it under the terms of the GNU General Public License as published by
9   the Free Software Foundation; either version 2, or (at your option)
10  any later version.
11 */
12
13
14 #include <math.h>
15
    
```

```

16 void cm_etd(ARGS)
17 {
18     Complex_t ac_gain1;
19     static double PVP, PIP, PVV, PIV, PA2;
20     static double PIS, T2, con1, con2, con3, VT;
21     double ith, ix, it, dith, dix, ditu;
22     static double vtd, itd, diff;
23     double P_K, P_Q;
24
25     if (INIT) {
26         PVP = PARAM(vp);
27         PIP = PARAM(ip);
28         PVV = PARAM(vv);
29         PIV = PARAM(iv);
30         PA2 = PARAM(a2);
31         PIS = PARAM(is);
32
33     /* Constants */
34
35         P_K = 1.3806503e-23 ; /* Boltzmann's constant in J/K */
36         P_Q = 1.602176462e-19; /* Charge of an electron in C */
37         T2 = PARAM(temp)+237.15;
38         VT = P_K*T2/P_Q; /* Thermal voltage at Temp in volts */
39         con1 = PIV*PA2;
40         con2 = PIS/VT;
41         con3 = PIP/PVP;
42
43
44
45     }
46
47     if (ANALYSIS != AC) {
48         vtd = INPUT(ntd);
49         ith = PIS*(exp(vtd/VT) -1.0);
50         ix = PIV*exp(PA2*(vtd-PVV));
51         it = PIP*(vtd/PVP)*exp(1-vtd/PVP);
52         itd = ith+ix+it;
53         dith = con2*exp(vtd/VT);
54         dix = con1*exp(PA2*(vtd-PVV));
55         ditu = con3*(1-vtd/PVP)*exp(1-vtd/PVP);
56         diff = dith+dix+ditu;
57
58         OUTPUT(ntd) = itd;
59         PARTIAL(ntd, ntd) = diff;
60
61     }
62
63     else {
64         ac_gain1.real = diff;
65         ac_gain1.imag = 0.0;
66         AC_GAIN(ntd, ntd) = ac_gain1;
67
68     }
69
70 }

```

Chapter 2. Basic Ngspice, Xyce and SPICE OPUS simulation

2.1 Introduction

This section describes a number of fundamental methods for launching circuit simulations from the Qucs GUI using the Ngspice, Xyce and SPICE OPUS compatible simulator engines. `Spice4qucs` includes built-in support for SPICE via a subsystem specifically designed for this purpose. The Ngspice, Xyce and SPICE OPUS simulators are not embedded in Qucs but operate as independent external simulators. Before use they must be installed on the computer operating system that you are running Qucs.

2.2 Supported simulators

Ngspice is a mixed-level/mixed-signal circuit simulator implemented from three open source software packages: SPICE 3f5, Cider 1b1 and XSPICE. Ngspice is one of the most widely used and stable current generation open source SPICE simulators available. It implements the original SPICE3f5 simulation capabilities, including for example, DC, AC, and transient simulation, Fourier-analysis and sensitivity analysis, plus a significant number of extra simulation and device model extensions. Distributed with Ngspice is a data manipulation package called Ngnutmeg. This provides numerical analysis and visualisation routines for post processing Ngspice simulation data. Instructions for installing Ngspice can be found on the Ngspice website at <http://ngspice.sourceforge.net/download.html>, The Ngspice website also gives free access to all the distribution and development package code sources.

Xyce is an open source, SPICE-compatible, high-performance analogue circuit simulator, capable of solving extremely large circuit problems when installed on large-scale parallel computing platforms. It also supports serial execution on all common desktop platforms, and small-scale parallel execution on Unix-like systems. Xyce for Linux, Microsoft Windows, and MacOS can be downloaded from the official Xyce website at <https://xyce.sandia.gov/Xyce>. The Xyce parallel circuit simulator running on Linux requires installation of the openMPI libraries. `Spice4qucs` supports both Xyce-Serial and Xyce-Parallel (not currently available for the Microsoft Windows operating system).

SPICE OPUS is an improved version of SPICE based on the original SPICE 3f5 code with extensions for circuit and device performance optimization and a transient simulation shooting method for large signal steady state AC analysis. SPICE OPUS can be downloaded from its official website at <http://www.spiceopus.si/>.

Although Ngspice, Xyce and SPICE OPUS are all compatible SPICE simulators they also include extensions to the original SPICE 3f5 netlist syntax which are often incompatible and may not simulate but generate errors. The Qucs Team is aware of this limitation and are attempting to correct such problems as quickly as possible. Please note this may take some time. However, if you do identify a compatibility bug please inform us by sending in a bug report to the Qucs web site (with an example test schematic if possible) describing the problem you have identified.

2.3 General simulation methods

The starting point for understanding how the `spice4qucs` extensions are built into the Qucs GUI is to study the basic operations needed to simulate Qucs circuit schematics with external simulators. For this purpose consider the simple RCL circuit shown in Figure 2.1.

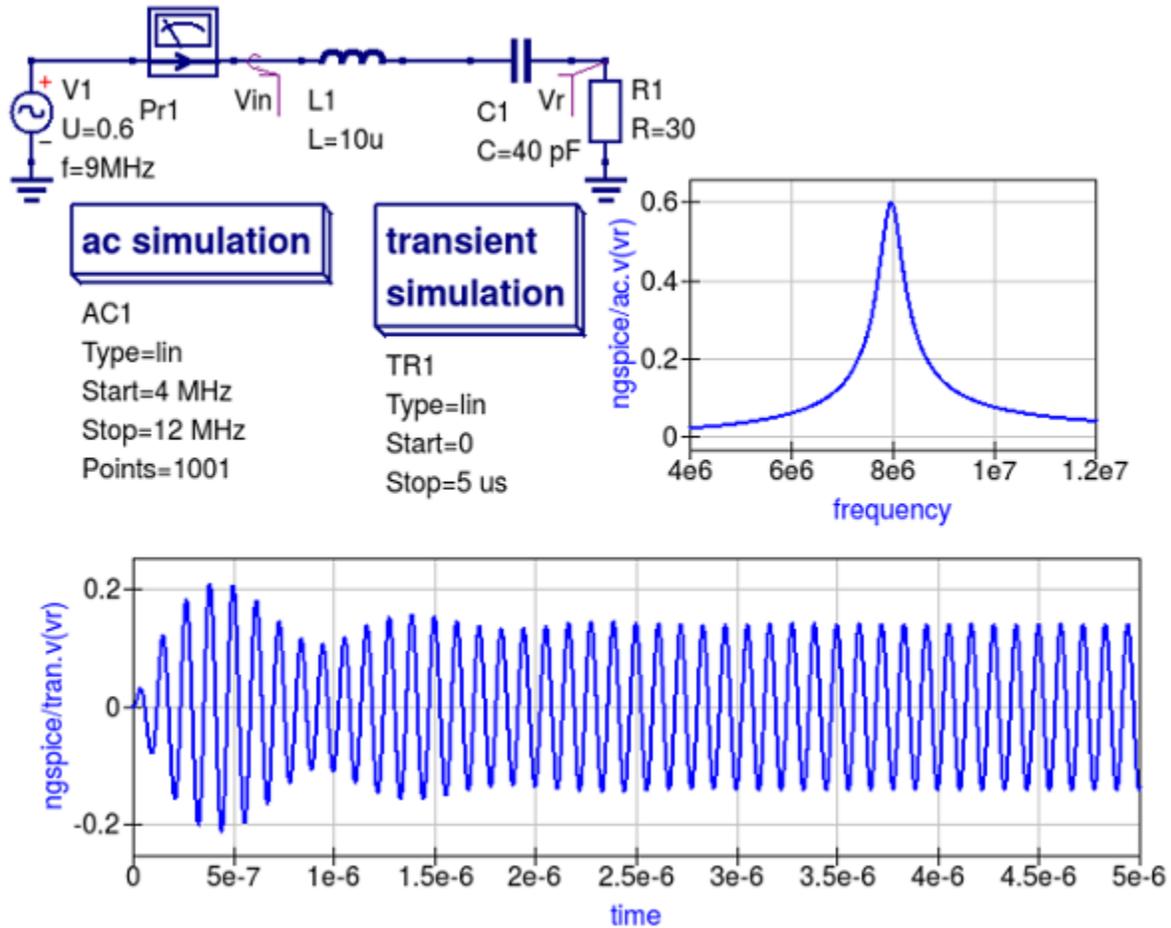


Figure 2.1. A simple RCL test circuit for demonstrating Ngspice, Xyce and SPICE OPUS simulation controlled from Qucs.

This schematic specifies two simulations:

- AC simulation from 4 MHz to 12 MHz.
- Transient simulation from 0 to 5 us;

Make sure the schematic in Figure 2.1 is drawn correctly then simulate it with Qucsator using the sequence *Simulation->Simulate*, or by pressing key F2. After Qucs finishes the the AC and transient simulations, plot the output data listed below:

- The voltage across R1 resistor in the frequency domain (given by the voltage at the Vr node),
- The input and output voltage waveforms (the voltages given by the Vin and Vr nodes) - your plots should be similar to those shown in Figure 2.1,
- The current in the frequency domain (Pr1 current probe),
- The transient current waveform sensed by the current probe Pr1.

Spice4qucs allows schematic component properties to be defined in the same way as Qucs. Component values and other icon properties are converted automatically into SPICE compatible netlist format. There is no need for manual adaptation by users. However, please note that not all the predefined Qucs components are available for simulation with Ngspice, Xyce or SPICE OPUS. A number of tables provided in later sections of the text list which components

can be used with which simulator. Following placement and wiring of components, plus the addition of one or more simulation icons, SPICE simulation is launched using the Qucs menu sequence *Simulation -> Simulate* or by pressing key F2. An *External simulator* dialogue then appears. This is illustrated in Figure 2.2.

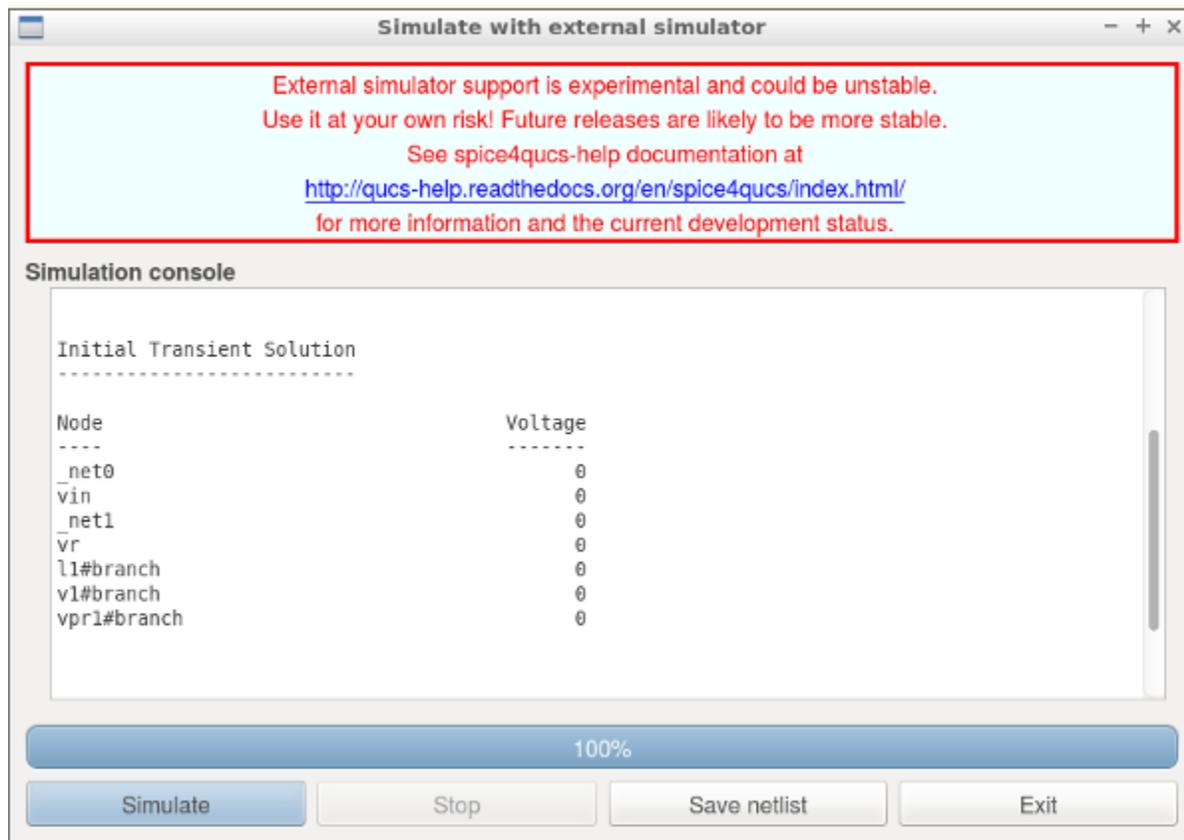


Figure 2.2. External simulator dialogue: where button *Simulate* launches a circuit simulation, button *Stop* causes a running simulation to finish; button *Save netlist* generates, and stores, the netlist of the circuit being simulated and button *Exit* closes the external simulator dialogue.

If the Ngspice, Xyce or SPICE OPUS installation directories are not included in the operating system shell `$PATH` statement the location of their executable code must be registered with `spice4qucs` before Ngspice Xyce or SPICE OPUS simulations will work. This step is necessary for all the operating systems used to run `spice4qucs`. To register external circuit simulator installation directories `spice4qucs` users need to launch the *Select default simulator*, from the *Simulate* dialogue. The resulting *Setup simulators executable simulator location* dialogue is illustrated in Figure 2.3. Using this dialogue enter the absolute address of the Ngspice, Xyce or SPICE OPUS executable program code from the keyboard or by pressing the appropriate *Open File Select* button.. In the case of the Xyce Parallel simulator the number of processors installed in your computer system, must also be entered from the keyboard or selected using the dialogue up-down arrow controls.

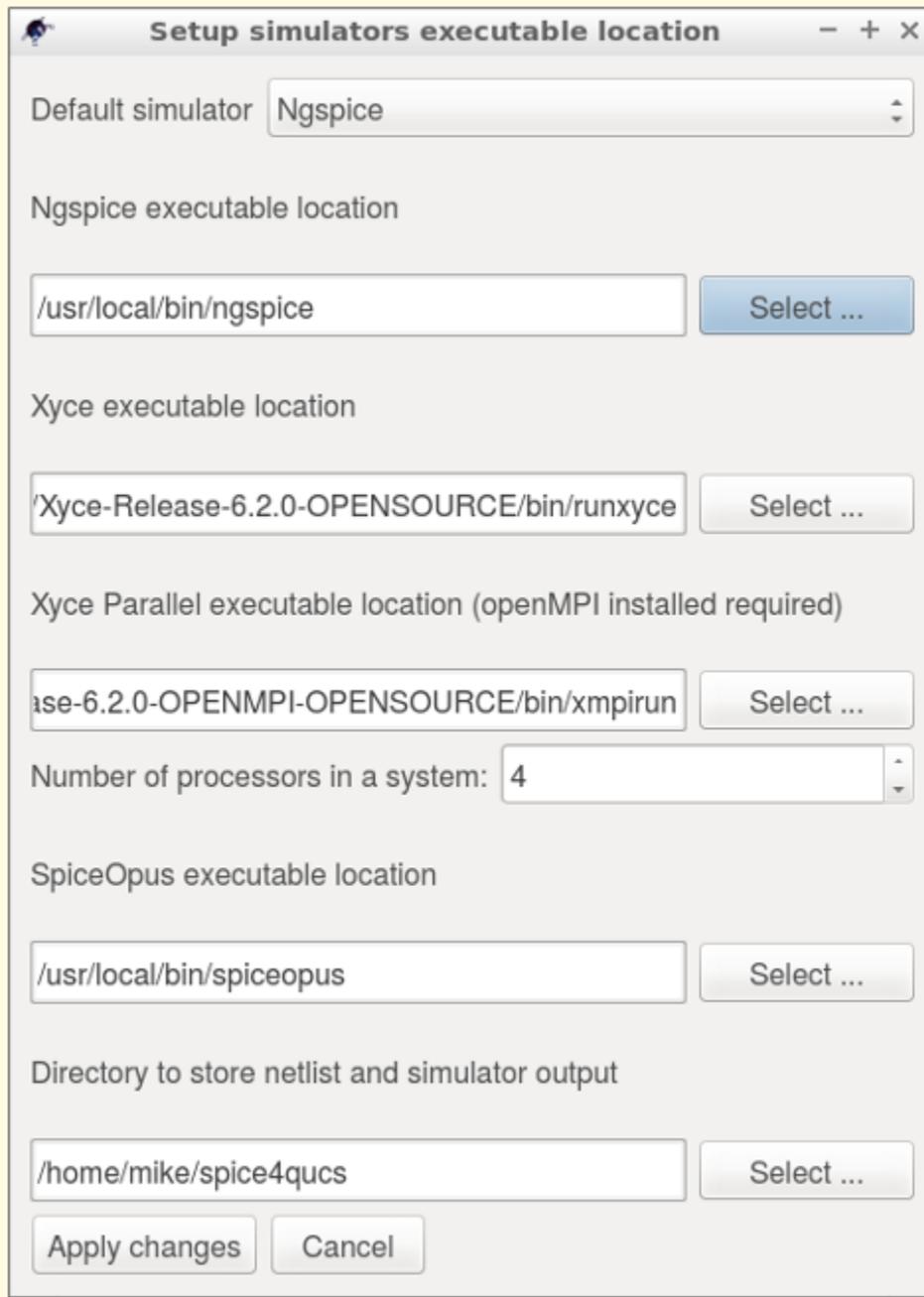


Figure 2.3. *Setup simulator executable locations* dialogue.

Please note the Xyce parallel command line for binary Xyce-Parallel package has the following format:

```
<Path_to_xyce_executable>/xmpirun -np %p
```

Where `spice4qucs` substitutes the number of processors for the `%p` wildcard entry.

Also please note that “user builds” of Xyce-Parallel have no `xmpirun` script, implying that the full script must be completed by users during the external simulators set up process, for example if `openMPI` is installed in directory `/usr/bin` and Xyce-Parallel installed in `/usr/local/Xyce_Parallel` the command line will be:

```
/usr/bin/mpirun -np %p /usr/local/Xyce_Parallel/bin/Xyce
```

Spice4qucs users can also define a directory where temporary simulator data and netlists are stored: this simulator working directory is by default assumed to be at `$HOME/.qucs/spice4qucs`.

To simulate a Qucs schematic with the Ngspice simulator, select simulator *Ngspice* and press the *Simulate* button shown in Figure 2.2. During simulation Ngspice produces a simulation log. This is displayed in the *External simulator* dialog window, see Figure 2.4. The Qucs Log text is also saved at Qucs system Log location `$HOME/.qucs/log.txt`. The Log text can be viewed and using the drop down menu sequence *Simulation->Show last messages* (or by pressing key F5). If the Ngspice simulation fails, any errors reported by Ngspice during simulation are listed in simulation Log window. Similarly, a successful completion of a Qucs/Ngspice simulation is reported.

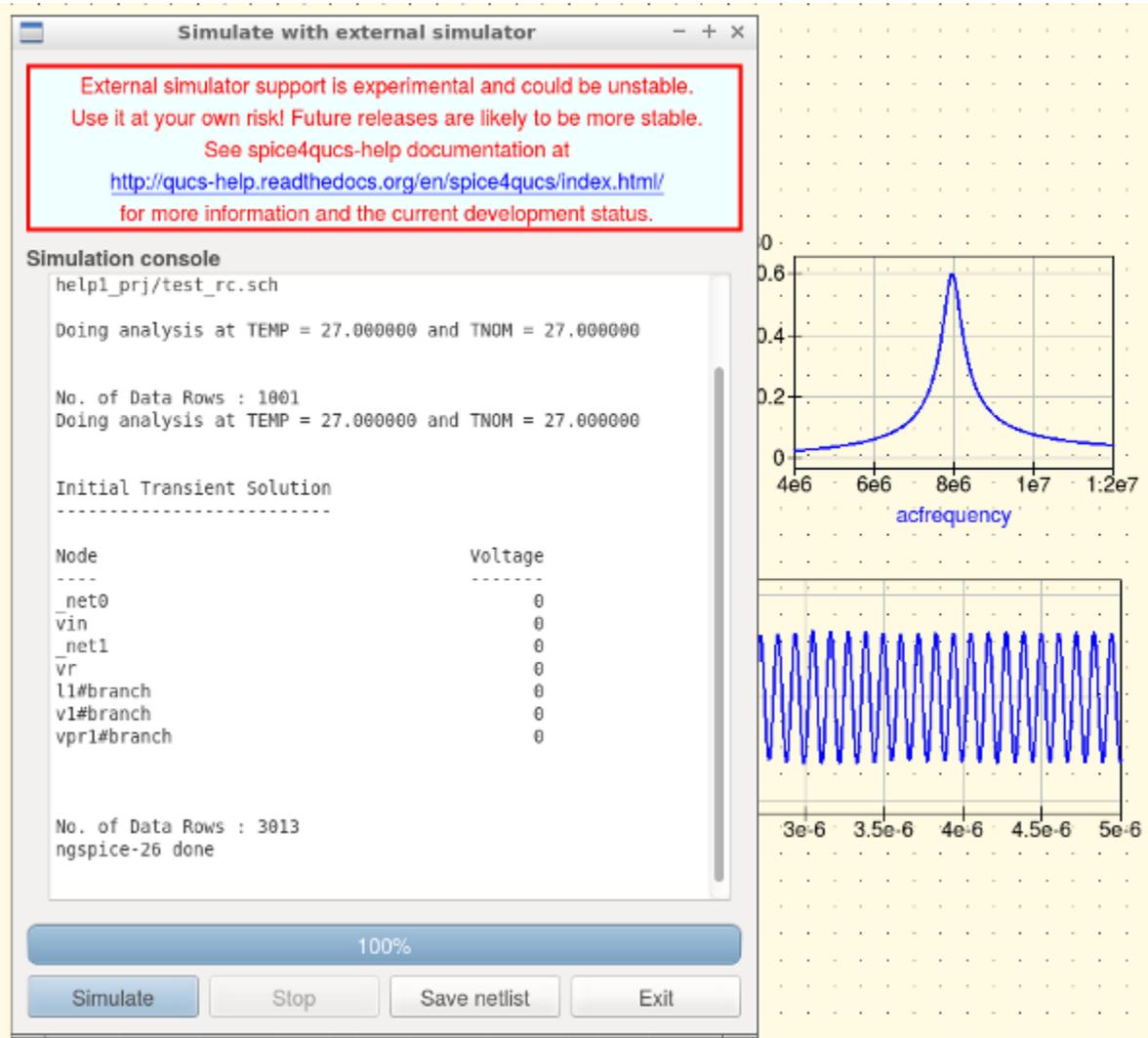


Figure 2.4. A section of an Ngspice execution Log displayed in the *Simulate with an external simulator* dialogue window.

An novel feature introduced by `spice4qucs` is its ability to generate and save SPICE netlist files from the information contained in a Qucs schematic. To save the SPICE netlist file for the current simulation press the *Save netlist* button shown in Figure 2.2. This process causes a SPICE netlist to be saved as file “netlist.cir” in the `~/qucs/spice4qucs` directory. Here is the generated netlist for the RCL test example:

```

1 * Qucs 0.0.19 /home/vvk/qucs/examples/ngspice/RCL.sch
2 V1 _net0 0 DC 0 SIN(0 0.6 7.5MEG 0 0) AC 0.6
3 VPr1 _net0 vIn DC 0 AC 0

```

```

4 L1 vIn _net1 10U
5 R1 0 vR 30
6 C1 _net1 vR 40P
7 .control
8 set filetype=ascii
9 AC LIN 1000 4MEG 12MEG
10 write RCL_ac.txt VPr1#branch v(vIn) v(vR)
11 destroy all
12 TRAN 4.97512e-09 1e-06 0
13 write RCL_tran.txt VPr1#branch v(vIn) v(vR)
14 destroy all
15 exit
16 .endc
17 .END

```

The simulation sequence introduced in the previous sections of the `spice4qucs-help` text also applies to the Xyce and SPICE OPUS simulators. However, the information displayed in the simulation log is likely to be different for different simulators and indeed operating systems.

After an Ngspice, Xyce or SPICE OPUS simulation has successfully completed close the *External simulation* dialogue by pressing the “Exit” button. The simulation data generated by a `spice4qucs` simulation is available for plotting using the normal Qucs visualisation routines: either drag a diagram icon, or table icon, onto the current Qucs schematic window or onto the associated Qucs display page. After a diagram or table is placed a *Diagram properties* dialog appears. On selecting the dataset for the current simulation the simulation output quantities become available for plotting or tabulating in a similar fashion to standard Qucs.

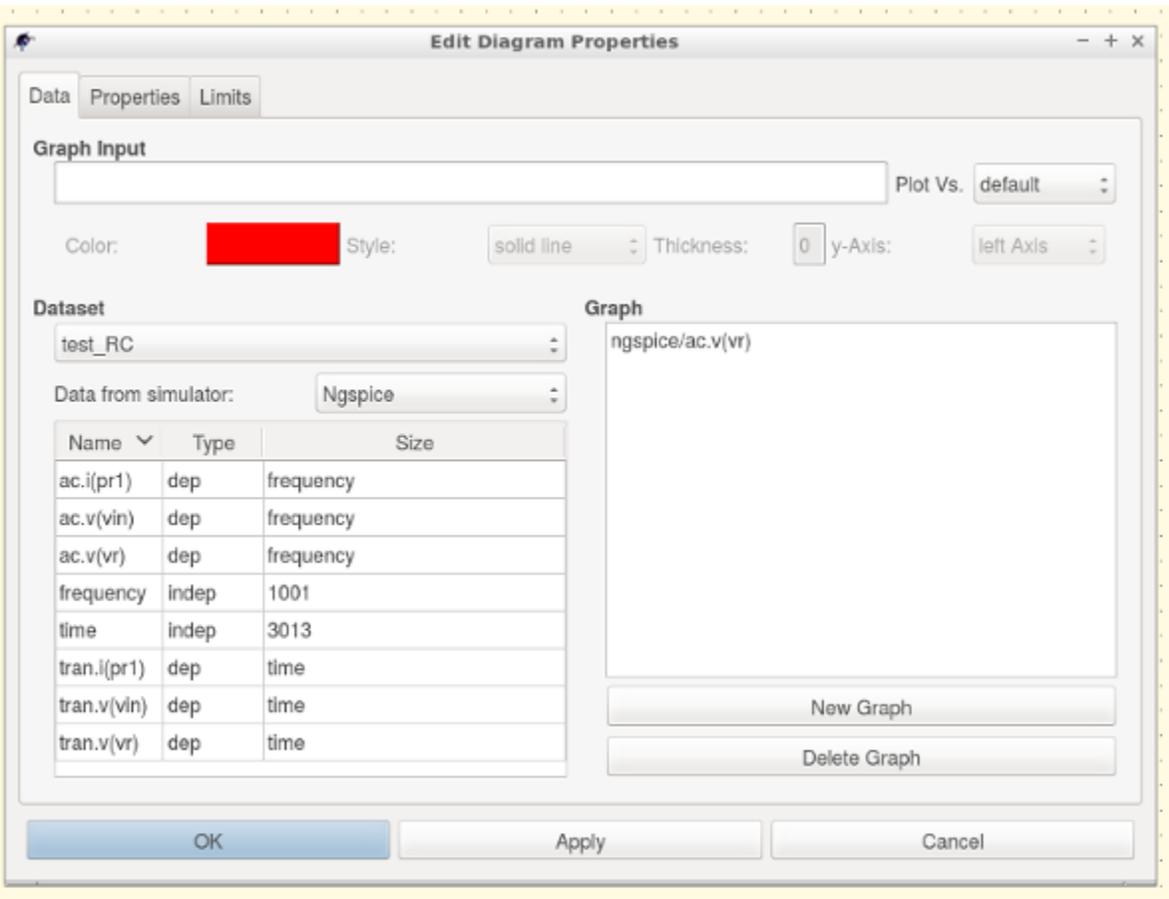


Figure 2.5. *Diagram properties* dialogue, listing the selected simulator and the available simulation data names.

Ngspice, Xyce and SPICE OPUS simulation data output is in raw-binary SPICE 3f5 output format. Qucs converts the SPICE 3f5 style data into a Qucs dataset using routines provided in the `spice4qucs` Qucs subsystem. Results from different types of simulation, for example SPICE AC and TRAN, are combined into a single Qucs dataset. Qucs adds an appropriate suffix to each simulator dataset name in order to avoid name clashes and mixing up results from different types of simulation. In the RCL test example the Qucs schematic is named `RCL.sch`. Qucs qucsator simulation, Ngspice, Xyce and SPICE OPUS simulations result in three different datasets:

- `RCL.dat` — for Qucsator;
- `RCL.dat.ngspice` — for Ngspice;
- `RCL.dat.xyce` — for Xyce;
- `RCL.dat.spopus` — for SPICE OPUS;

All four datasets have an extension `dat` to signify that each set contains Qucs data for post simulation visualisation. The Ngspice, Xyce and SPICE OPUS datasets include second a extension to the file name to identify the name of the external Qucs simulator.

The Dataset selector (see Figure 2.5) shows only the base names of a dataset (for example `test_RC`). Users must also select the appropriate simulator from the *simulator name selector* drop-down list. This drop-down only gives existing simulator datasets which prevents users from selecting non-existent datasets by mistake.

Following the selection of a specific data set users must select the variables that are to be plotted. `Spice4qucs` preserves SPICE notation for **node voltage** names and **current probe** names. SPICE names are assumed to be case insensitive by `spice4qucs`, for example

- `v(out)` — Voltage at node `out`;
- `i(Pr1)` — Current recorded by current probe `Pr1`;

The Qucs `spice4qucs` extension also adds a simulation-dependent prefix to each variable name in order to differentiate output variables from different SPICE simulations, for example `ac.` for AC simulation, `tran.` for transient simulation, and `dc.` for DC-sweep.

There are also individual prefixes for each simulator:

- `ngspice/` — Ngspice simulator prefix;
- `xyce/` — Xyce simulator prefix;
- `spopus/` — SPICE OPUS prefix;

Hence for example, the full name of variable from an Ngspice simulation could be `ngspice/v(out)`. This naming system helps to avoid dataset name conflicts.

Individual items for plotting are selected by double clicking on a name in the variable list. As an example when double clicking on `ac.i(pr1)` its name is copied by Qucs into the right-hand plotting window. Like standard Qucs one or more variable items may be selected for plotting on the same 2D or 3D graph. Finally pressing the *Apply* button shown at the bottom of Figure 2.5. causes the selected variable items to be plotted. The plotted simulation results for the external Ngspice AC simulation of the RCL test circuit are shown in Figure 2.6.

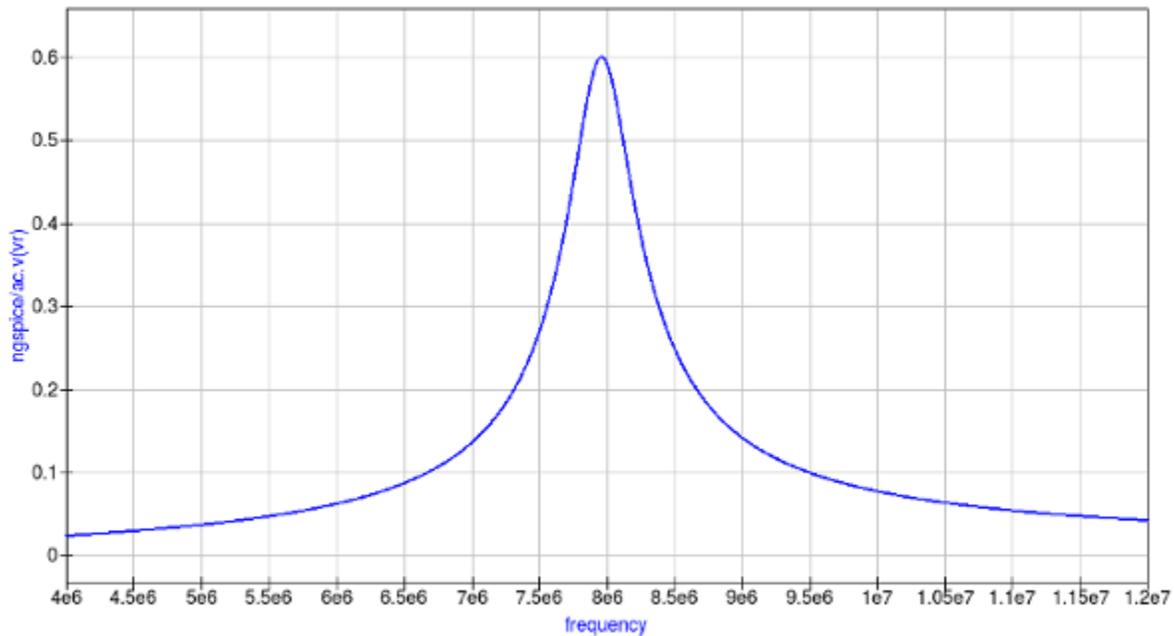


Figure 2.6. External SPICE AC simulation magnitude response for the current flowing in RCL circuit with a series resonant peak of roughly 8 MHz .

Plotting the transient simulation data for the RCL test example follows the same procedure as the sequence described for the AC simulation except that in the transient plot variables with `tran` in their name are selected, see Figure 2.7.

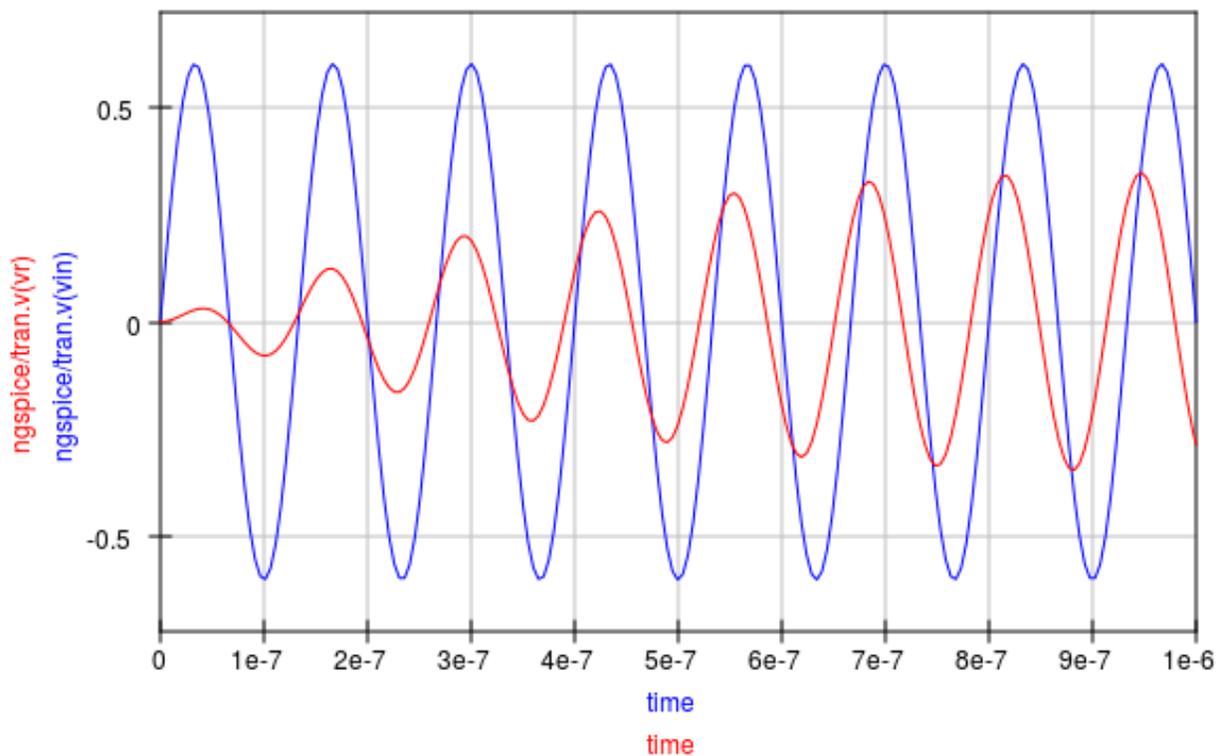


Figure 2.7. Transient simulation voltage waveforms at the input and output nodes of the RCL circuit.

A similar procedure is adopted for plotting simulation data generated with the Xyce and SPICE OPUS simulators. Readers should make sure they can simulate the example RCL circuit with both Xyce and SPICE opus, then plot the resulting simulation data. More advanced techniques for post processing, plotting and undertaking a range of different visualization processes using Qucs and Octave are outlined in chapter 7.

2.4 Variable names

As part of the `spice4qucs` extensions Ngspice and Xyce simulation variable names are converted from Qucs notation to SPICE notation and vice versa. Table 2.1 shows the correspondence between the two notations.

Table 2.1 Qucs and SPICE variable equivalences

Variable type	Qucs display notation	Spice display notation
DC node voltage	Node.V	V(node)
AC node voltage	Node.v	ac.v(node)
TRAN node voltage	Node.Vt	tran.v(node)
HB node voltage	Node.Vb	hb.v(node)
DC probe current	Pr1.I	i(pr1)
AC probe current	Pr1.i	ac.i(pr1)
TRAN probe current	Pr1.It	tran.i(pr1)

Also variable prefixes used to designate data from different simulators (Table 2.2)

Table 2.2 Qucs and SPICE variable name prefixes

Prefix	Explanation
Node.Vt	Qucs simulation, default dataset
dataset:Node.Vt	Qucs simulation, external dataset
ngspice/tran.v(node)	Ngspice simulation, default dataset
ngspice/dataset:tran.v(node)	Ngspice simulation, external dataset
xyce/tran.v(node)	Xyce simulation, default dataset
xyce/dataset:tran.v(node)	Xyce simulation, external dataset
spopus/tran.v(node)	SPICE OPUS simulation, default dataset
spopus/dataset:tran.v(node)	SPICE OPUS simulation, external dataset

2.5 DC simulation

Conventional SPICE 3f5 simulation commands OP and DC are not implemented by Qucs or indeed by `spice4qucs`. Instead more convenient versions of these simulation commands are implemented. These alternative forms of non-linear steady state DC circuit analysis are linked directly to circuit schematic capture, making them easy to use. Moreover, they provide `spice4qucs` users with a power full diagnostic and analysis tools for investigating basic circuit operation. The circuit shown in Figure 2.8 represents a simple resistive network with single voltage and current 1 V and 1 A sources respectively. Pressing key “F8” instigates a DC analysis and adds the DC node voltages, probe voltages and probe currents to the current schematic. This feature provides a practical method for scanning a circuit to see if the DC bias values are of the correct order of magnitude. The calculation of DC bias values via the F8 key applies to all the circuit simulators controlled by `spice4qucs`. Schematics which include the `spice4qucs` DC icon do not however, list a similar set of voltage and currents in the “*Simulate with an external Simulator” dialogue window. A DC voltage and current list is output when a schematic includes a transient simulation icon, see Figure 2.9.

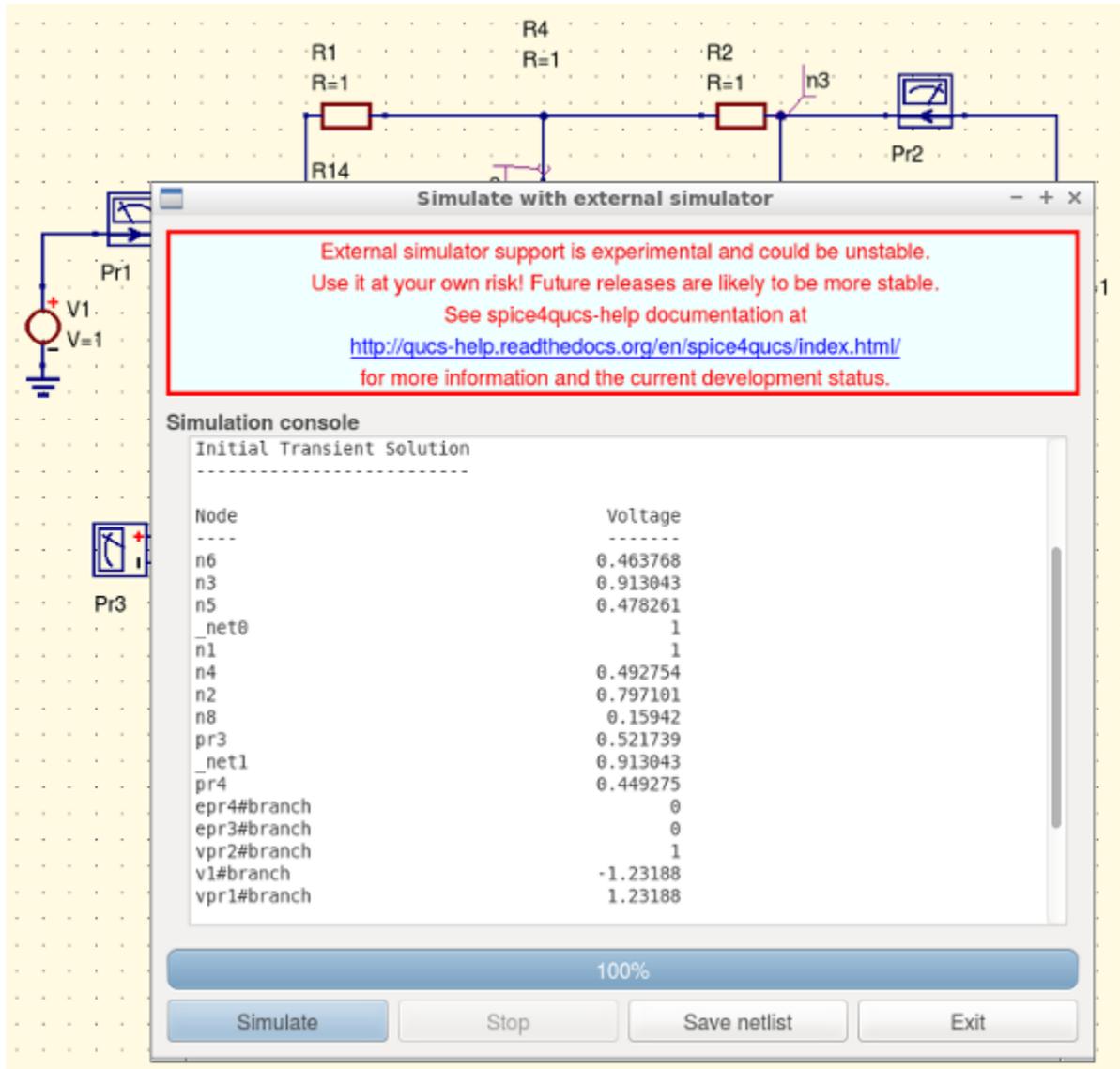


Figure 2.9 A screen dump showing transient simulation initial DC simulation voltage and current values in (V) and (A) respectively for the resistive circuit given in Figure 2.8: NOTE that the voltage and current variable names are output in SPICE style syntax.

Qucs does not define a separate analysis type which is equivalent to the original SPICE 2g6 “DC sweep” simulation or the extended SPICE 3f5 version which allows current and voltage source scans plus resistor value scans. In contrast to SPICE the Qucs equivalent “DC sweep” is just a specific case of the more general Qucs `Parameter sweep` capability. To emulate the original SPICE *DC sweep*, `spice4qucs` use a combination of Qucs DC simulation plus the `Parameter sweep` of an independent DC voltage or DC current source or of a resistor numerical value; when the `spice4qucs` Spice netlist builder finds these two linked types of simulation it synthesises them into a DC SPICE netlist entry. This procedure is demonstrated in Figure 2.10. where the test circuit consists of a diode DC bias network connected as a test bench for simulating the non-linear DC current-voltage characteristic of a 1N4148 diode. This example can be found in the Qucs examples directory tree listed as `examples\ngspice\diode.sch`.

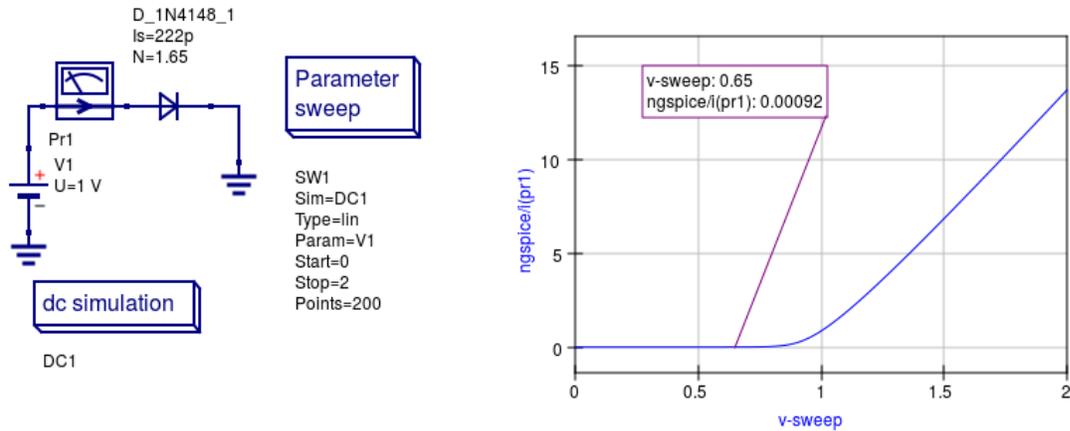


Figure 2.10. Test circuit and simulated DC current-voltage characteristics for a 1N4148 silicon diode.

Please note the following differences between SPICE and Qucs DC-sweep simulation:

- Specify a sweep source name or a resistor name **NOT** a source or resistor value; for example in Figure 2.10 V1.
- SPICE model parameters can be swept using the notation `Device.Param`, for example `T1.Bf` to sweep the Bf parameter of transistor T1.

2.6 AC simulation

Small signal AC simulation is fully supported by the `spice4qucs` subsystem. It doesn't require any special adaptation. Just simple place the AC simulation component icon on a schematic and execute an `Ngspice`, `Xyce` or `SPICE OPUS` simulation. Variable name conversions are listed in Table 2.1. The Qucs `spice4qucs` dataset builder adds the `ac.` prefix to all variables generated by an AC simulation.

`Ngspice`, `Xyce` and `SPICE OPUS` small signal frequency domain AC simulations use linear, decade or octave frequency scales. Adaptive frequency steps are not implemented.

2.7 TRANSient simulation

Transient simulation is also fully supported by the `spice4qucs` subsystem. Just place the `Transient simulation` component icon on a schematic and simulate it. There is a difference between the way the `qucsator`, `Ngspice`, `Xyce` and `SPICE OPUS` simulators implement transient simulation time steps.

`Qucsator` always uses a fixed time step. `Ngspice`, `Xyce` and `SPICE OPUS` use adaptive time steps. The number of simulation points output during a simulation will only be approximately equal to the number of simulation points specified in a `Transient simulation` properties list. For example, in an example test circuit 200 time points are specified on the schematic. However, due to the fact that the `SPICE` simulators use adaptive time steps, `Ngspice` employs 213 simulation points, and `Xyce` employs 799 time points. This difference should be taken into account during simulation data post processing and when comparing simulation results.

2.8 Other forms of simulation

In contrast to `SPICE 3f5`, the parameter sweep facility found in Qucs has also been implemented with `Ngspice`, `Xyce` and `SPICE OPUS` where the parameter sweep setup and control is organized by `spice4qucs`. The details of how this Parameter sweep feature works is the topic of section 5.8.

As well as the fundamental DC, AC and transient simulation types, Ngspice, Xyce and SPICE OPUS also support the additional forms of simulation listed in Table 2.3.

Table 2.3 `spice4qucs` simulation types additional to DC, AC and TRAN

Simulation Type	Ngspice	Xyce	SPICE OPUS	See section
Fourier	X	X	X	5.1
Distortion	X		X	5.2
Noise	X	X	X	5.3
Pole-zero	X		X	5.4
Sensitivity	X	X	X	5.5
Harmonic Balance		X		13.5
Tran shooting method			X	13.7
Custom simulation	X		X	8.0

Fourier, distortion pole-zero circuit simulation require special GUI icons. These can be found in the Qucs *Spice simulations* group. In contrast sensitivity, the SPICE OPUS tran shooting method is accessed by `spice4qucs` via the Custom simulation technique, see section 8.0.

2.9 `spice4qucs` circuit simulation components

Qucs is released with a good selection of passive and active component models. This selection includes both fundamental circuit components, like R, C and L and collections of specific components for a given circuit design sector, like the RF microstrip component models. All the original Qucs component and device models were written to work with Qucs and there is **NO Guarantee** that they will be work with Ngspice, Xyce or SPICE OPUS. For circuit simulation packages which take advantage of simulation multi-engines this can be a serious problem, particularly for the less experienced user. To help reduce problems to a minimum, `spice4qucs` uses a policy of “blacklisting” those models which do not work with the chosen circuit simulation engine. This policy works in the following way:- when a specific simulator is chosen by a Qucs `spice4qucs` user, on running the chosen simulator, **ONLY** those models which work with the selected simulator become available for drawing circuit schematics and simulation. The same approach applies to the components held in the `spice4qucs` libraries.

2.10 More basic simulation examples

2.10.1 DC Example 1: Calculating circuit input resistance and power dissipation in a resistor.

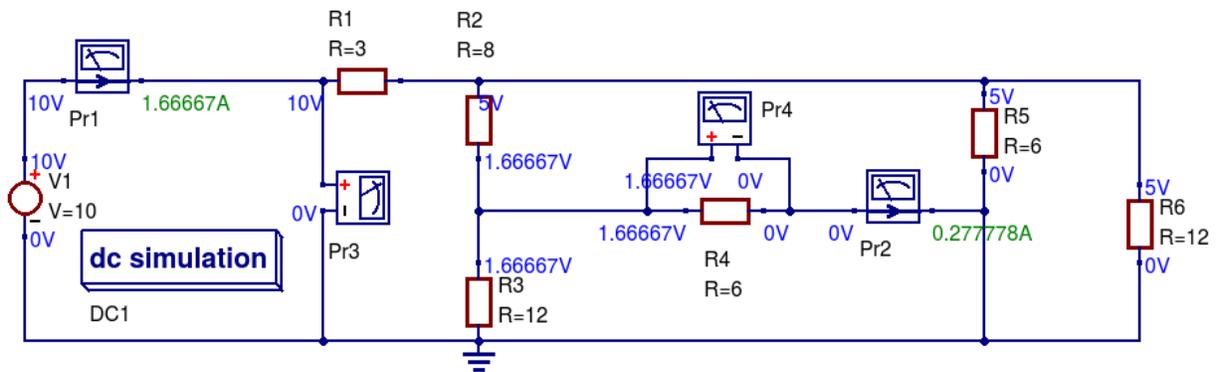


Figure 2.11 DC resistive test network.

- Draw the circuit diagram shown in Figure 2.11,

- Select simulator Ngspice,
- Press key F8
- Determine DC Rin = $V(\text{Pr3})/I(\text{Pr1}) = 10/1.66667 = 6 \text{ Ohm}$,
- Determine the power dissipated in R4 = $V(\text{Pr4}) * I(\text{Pr2}) = 1.66667 * 0.277778 = 0.463 \text{ W}$.

2.10.2 DC Example 2: Variation of power dissipation with varying DC input voltage.

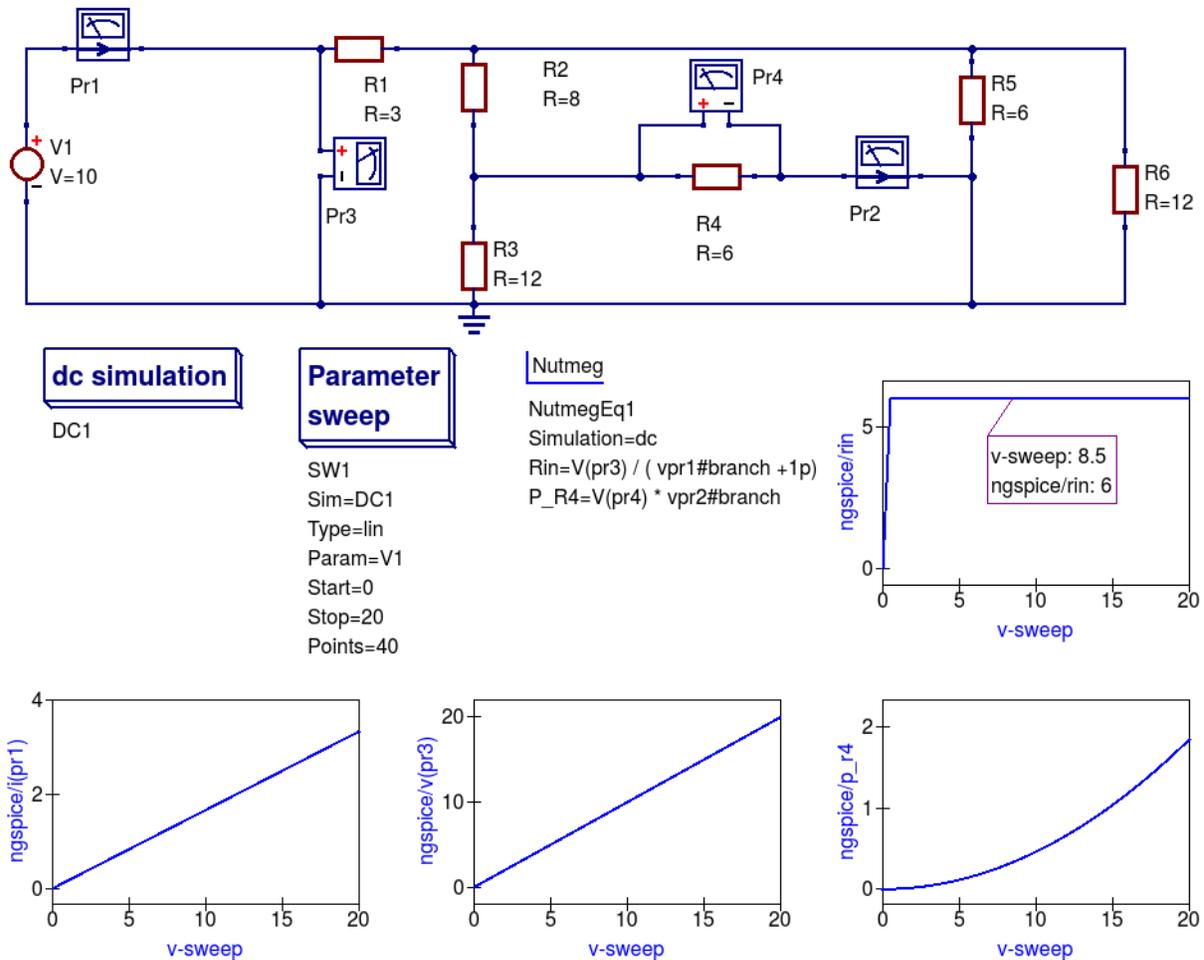


Figure 2.12 DC example 1 with varying DC input voltage: demonstrating the use of a DC sweep simulation.

- Draw the circuit diagram shown in Figure 2.11,
- Select simulator Ngspice,
- Add the dc simulation, Parameter sweep and Nutmeg component icons to the drawn schematic,
- Complete the Parameter sweep and Nutmeg component data entries so that they are the same as given in Figure 2.11,
- Press the F2 to simulate the circuit,
- Plot the graphs illustrated in Figure 2.11,

- Check that your results are the same - if not or the simulation fails check your schematic for errors and re-simulate.

Notes:

- Current probe values are represented by the SPICE 3f5 notation: vpr1#branch and vpr2#branch.
- There is a discontinuity in R_{in} when the vpr1#branch current is zero Amperes; hence the need for the dummy 1pA in the Nutmeg equation for R_{in} .

back to the top

Chapter 3. Spice4qucs subcircuits, macromodels and device libraries

3.1 Spice4qucs Subcircuits: with and without parameters

Subcircuits are a concept that has been part of established circuit simulation practice since the early days of computer aided circuit design. Today, all practical circuit simulators have subcircuits as part of their basic device compliment. This is not surprising because they form a natural way of breaking an electronic system down into a number of smaller self contained functional blocks. Spice4qucs supports all the features available with Qucs subcircuits. In a similar fashion to Qucs, the Ngspice, Xyce and SPICE OPUS circuit simulators allow subcircuits with or without parameters. However, an optional part of the Ngspice, Xyce and SPICE OPUS subcircuit netlist syntax that signifies that a parameter extension is present is not allowed (see section 3.1.2). As a starting point spice4qucs subcircuits without parameters are considered first. This introduction is followed by a detailed description of the structure, and netlist syntax, of subcircuits with one or more parameters.

3.1.1 Spice4qucs subcircuits without parameters

Figure 3.1 shows a Qucs subcircuit model for a 15MHz centre frequency band pass passive filter. Note that the three distinct parts of a subcircuit model without parameters are: (1) a circuit representing the model body with one or more input (Pin) and output (Pout) pins plus connected components selected from Qucs pre-defined components and user designed subcircuits (there are no user defined subcircuits present in Figure 3.1), (2) a subcircuit symbol, and (3) a Qucs netlist giving a list of the internal components, their connection nodes and a wrapper which defines the subcircuit. The syntax of the subcircuit netlist listed in Figure 3.1 is only understood by Qucs and cannot be read without error by external SPICE simulators.

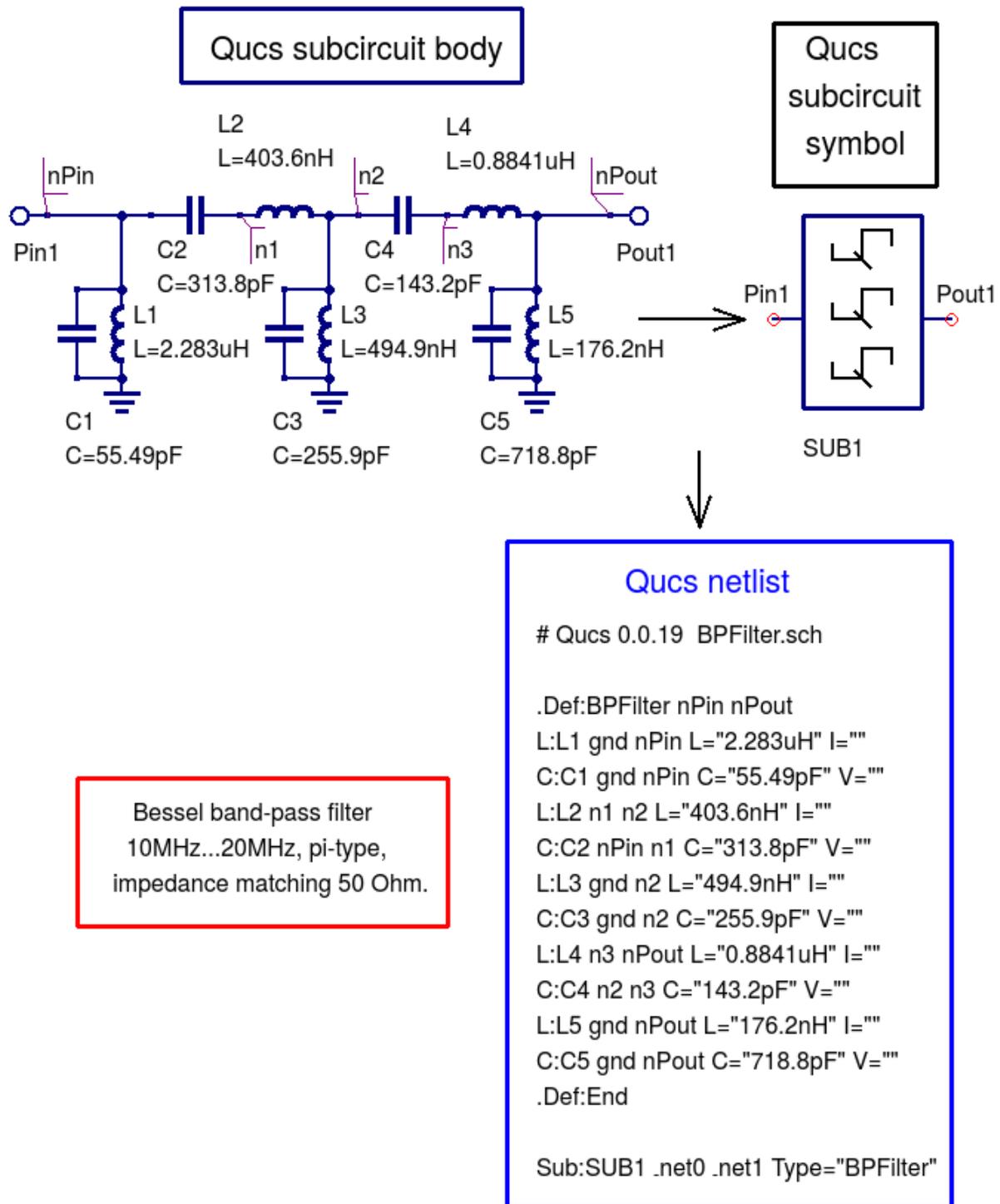


Figure 3.1 Qucs 15MHz centre frequency band pass passive filter subcircuit without parameters

A test bench circuit for simulating the band pass filter circuit shown in Figure 3.1 is given in Figure 3.2. This figure includes a plot of the small signal AC output voltage for a filter with 50 Ohm input and output matching resistors. Note the use of a node voltage probe and the signal name allocated by Qucs. Also note that the individual capacitor voltage and inductor current initial conditions are not set as they are not needed due to fact that the filter subcircuit is not DC biased. As a consequence the DC simulation icon shown in Figure 3.2 is not strictly necessary. However, its

a good idea to add it automatically to AC simulations because circuits with semiconductor devices or other non-linear components must have their small signal AC properties calculated, at their DC bias conditions, prior to small signal AC simulation.

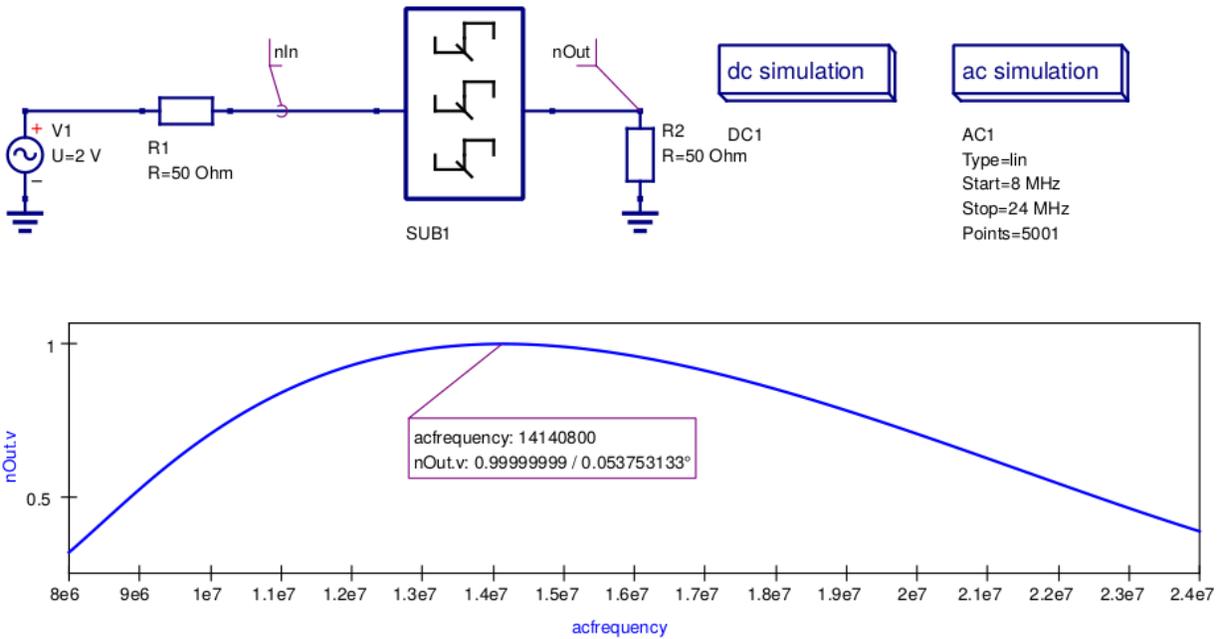


Figure 3.2 Qucs 15MHz centre frequency band pass passive filter test bench with 50 Ohm source and load matching

Figure 3.3 to Figure 3.5 present AC simulation results for the band pass filter generated with the Ngspice, Xyce and SPICEOPUS circuit simulators.

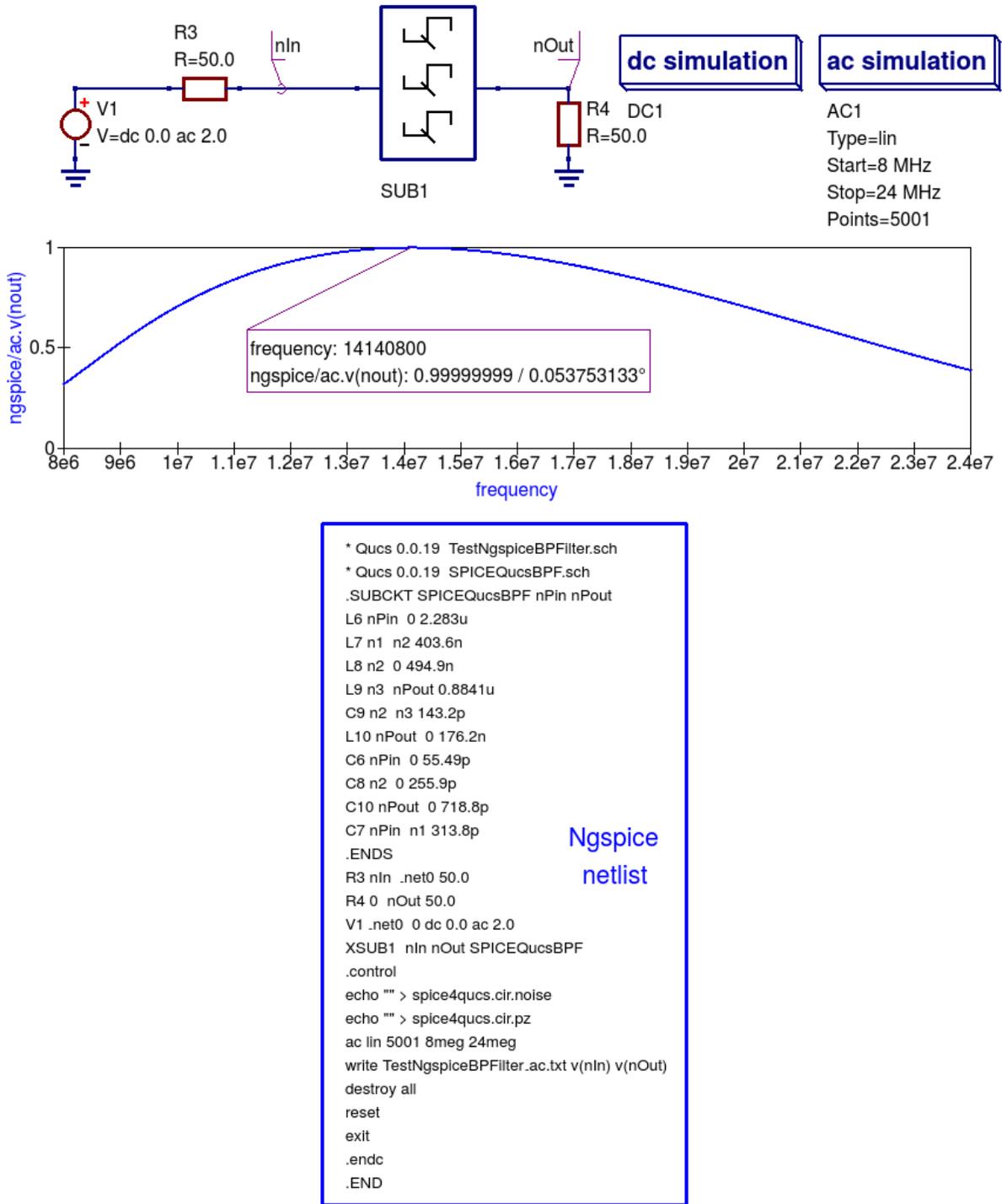


Figure 3.3 Band pass filter Ngspice test results and SPICE netlist for test bench circuit.

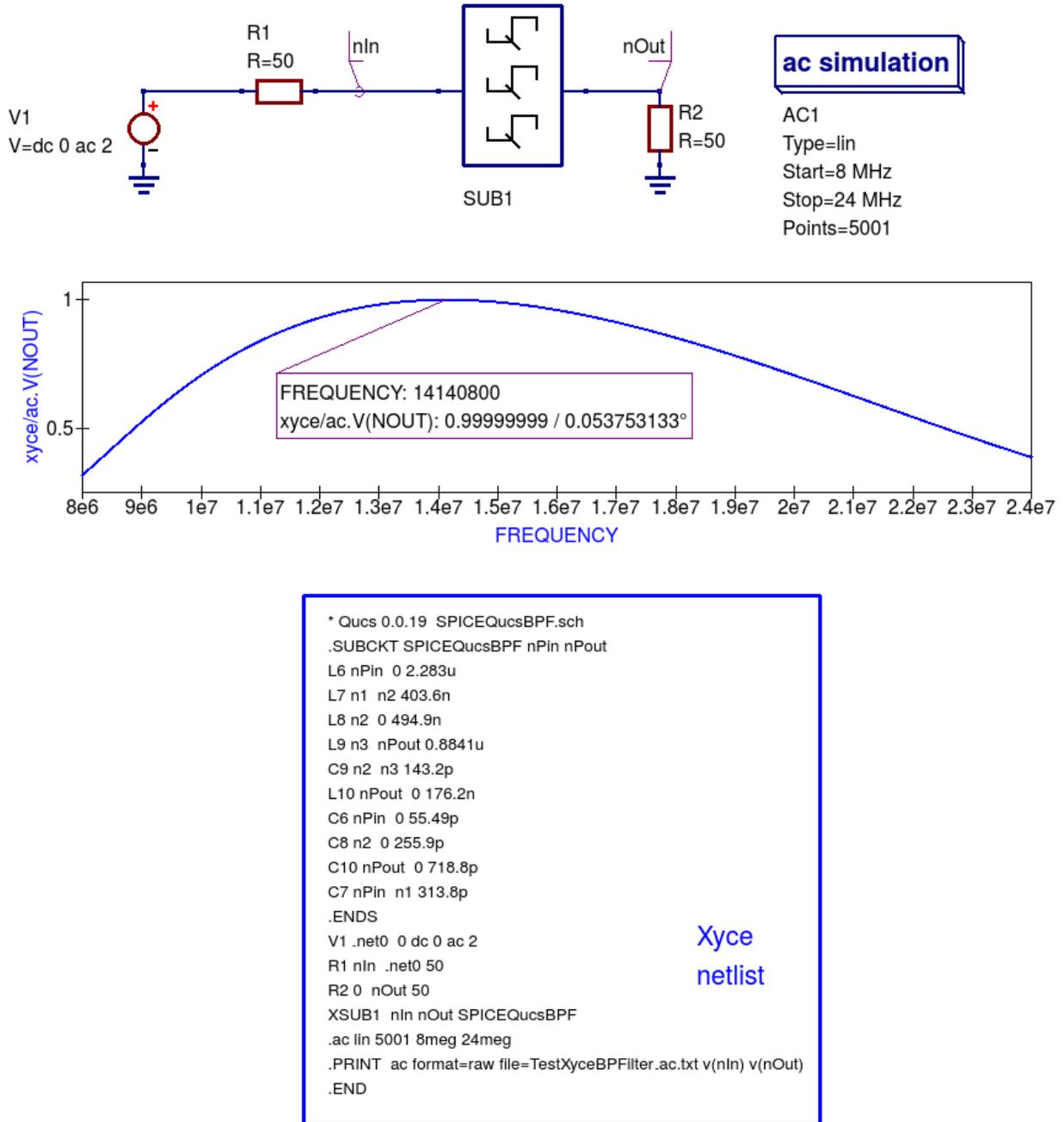
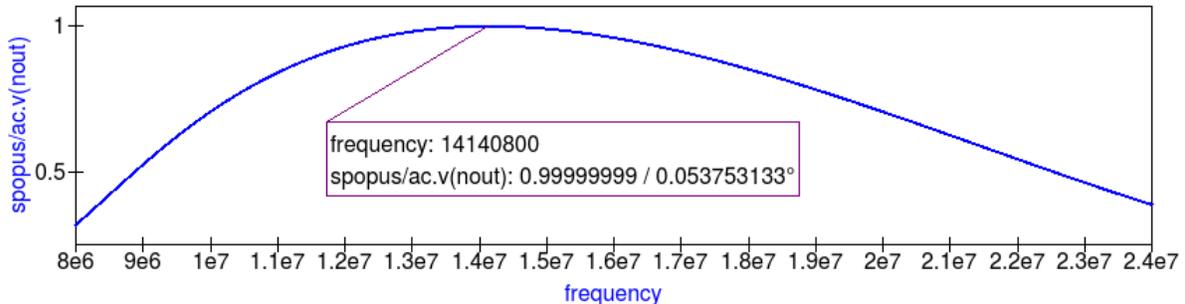
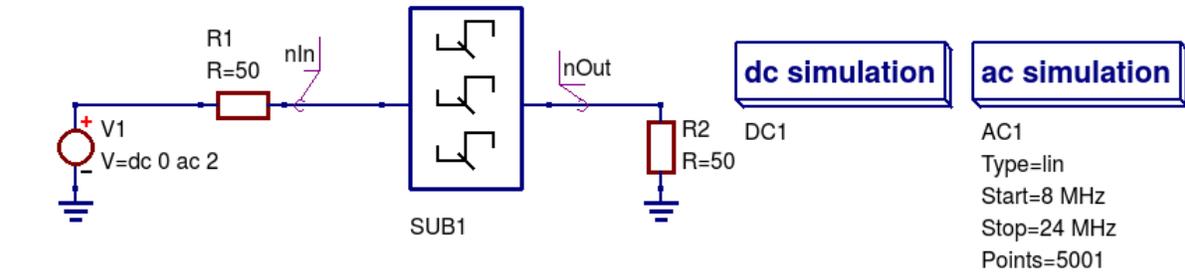


Figure 3.4 Band pass filter Xyce test results and SPICE netlist for test bench circuit.



```

* Qucs 0.0.19 SPICEQucsBPF.sch
.SUBCKT SPICEQucsBPF nPin nPout
L6 nPin 0 2.283u
L7 n1 n2 403.6n
L8 n2 0 494.9n
L9 n3 nPout 0.8841u
C9 n2 n3 143.2p
L10 nPout 0 176.2n
C6 nPin 0 55.49p
C8 n2 0 255.9p
C10 nPout 0 718.8p
C7 nPin n1 313.8p
.ENDS
V1 .net0 0 dc 0 ac 2
R2 0 nOut 50
R1 nIn .net0 50
XSUB1 nIn nOut SPICEQucsBPF
.control
echo "" > spice4qucs.cir.noise
echo "" > spice4qucs.cir.pz
ac lin 5001 8meg 24meg
write TSPICEOPUSBPF.ac.txt v(nIn) v(nOut)
destroy all
reset
exit
.endc
.END
    
```

SPICEOPUS
netlist

Figure 3.5 Band pass filter SPICEOPUS test results and SPICE netlist for test bench circuit.

Most readers will probably have noticed that the SPICE netlists for the Ngspice and SPICEOPUS band pass filter test benches are identical except for file names. This is because Ngspice and SPICEOPUS both include implementations of the SPICE 3f5 **Nutmeg** post simulation data processing package that is distributed with their SPICE engines.

These are not the same however, mainly because the SPICE OPUS development team have modified the original SPICE 3f5 Nutmeg package to firstly remove errors/bugs and secondly to improve its syntax. The extent to how this will affect the operation of spice4qucs is at this time unclear. If problems/bugs, due to implementation differences, surface in the future the spice4qucs Development Team will attempt to correct them as quickly as possible. The Xyce circuit simulator does not include a version of Nutmeg in its distribution package. This implies that Nutmeg style post simulation data processing is not possible with Xyce. However, to minimise the effects of this omission an extended form of the SPICE .PRINT statement has been implemented in Xyce, allowing algebraic expressions for data processing to be embedded in .PRINT statements. This topic and other aspects of Xyce post simulation data processing are covered in later sections of this help manual.

3.1.2 Spice4qucs subcircuits with parameters

Subcircuits which have component or physical parameter values set by a list of names and values attached to a schematic symbol add a significant “value added” feature to the subcircuit concept. This form of subcircuit can, for example, be used to represent manufacturers product variations which have identical circuits but require component values or device parameter values of differing value. Unfortunately, SPICE 3f5 only implements subcircuits without parameters. Recent generations of open-source GPL circuit simulators, including Ngspice, Xyce and SPICE OPUS, have been extended by their Development Teams to allow subcircuits with parameters. One consequence of this is that over time divergence of the SPICE subcircuit statement syntax has occurred amongst different circuit simulators. Spice4qucs implements a common subset of the published extended SPICE subcircuit syntax. This works well, but does have one disadvantage however, in that some published subcircuit netlists may require a small amount of editing before they will simulate with Spice4qucs. One code word often found in the SPICE extended subcircuit syntax is the term **PARAMS:**. This can occur in an **X** subcircuit call to signify a subcircuit with parameters. As this is optional in Ngspice, and indeed in other SPICE derived circuit simulators, it is not implemented in Spice4qucs.

Qucsator, Ngspice, Xyce and SPICEOPUS all allow parameters to be attached to subcircuit symbols and to be used in design equation calculations. As an introductory example Figure 3.6 illustrates a circuit schematic and user generated symbol for a simple Qucs harmonic generator composed of a fundamental AC signal and three sinusoidal harmonic components. Parameters $f1$ to $f4$ set the frequencies of the harmonics. The Qucs Equation block, at the subcircuit internal circuit level, is used to calculate the individual harmonic frequencies. In a similar fashion $ph1$ to $ph4$ represent the phases of the signal harmonics.

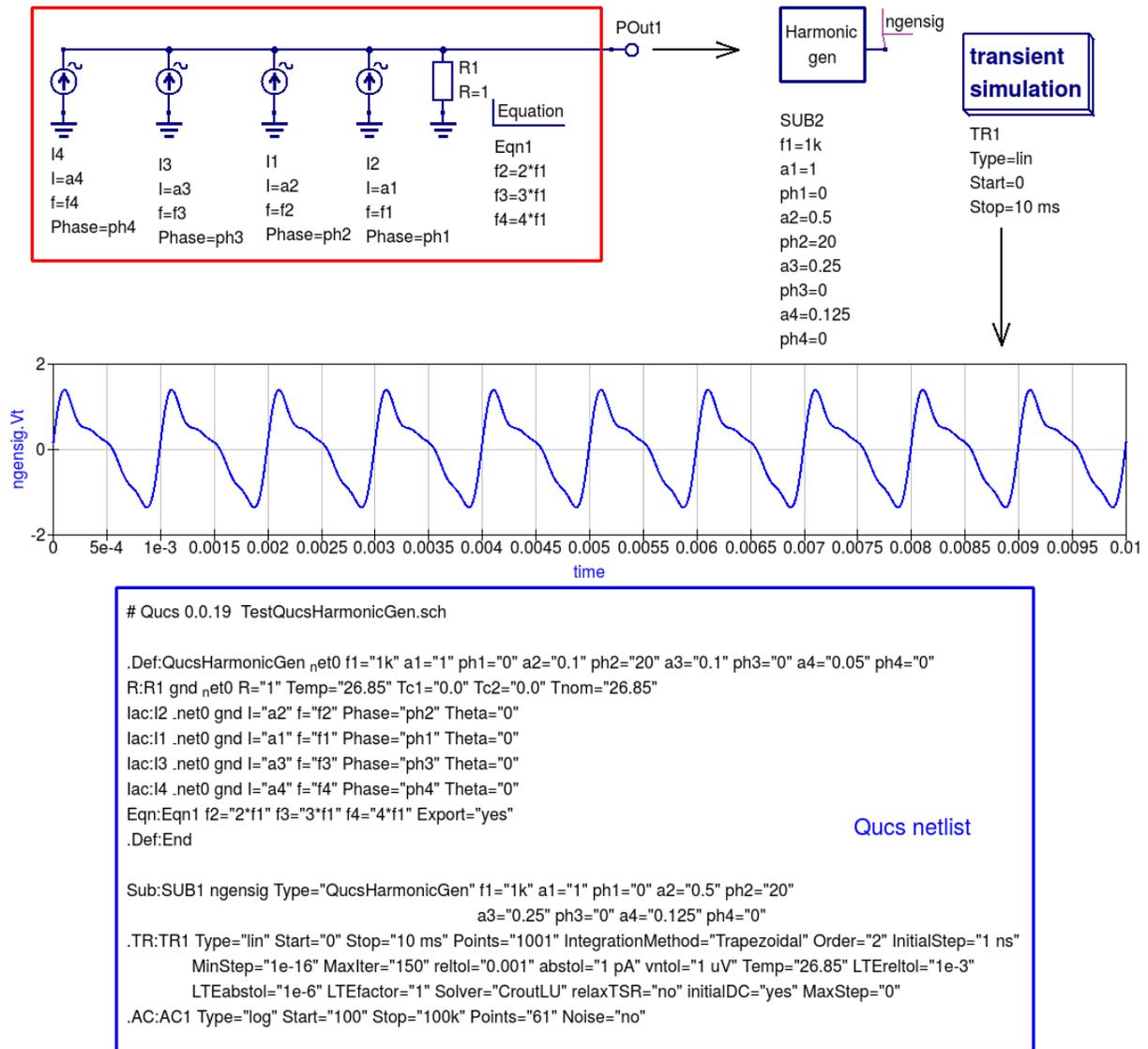


Figure 3.6 Qucs subcircuit sinusoidal harmonic signal generator: f_1 is the fundamental frequency and f_2 to f_4 the higher order harmonics; ph_1 to ph_4 the phases of the fundamental signal and its harmonics. For clarity long Qucs netlist lines have been spread over more than one line.

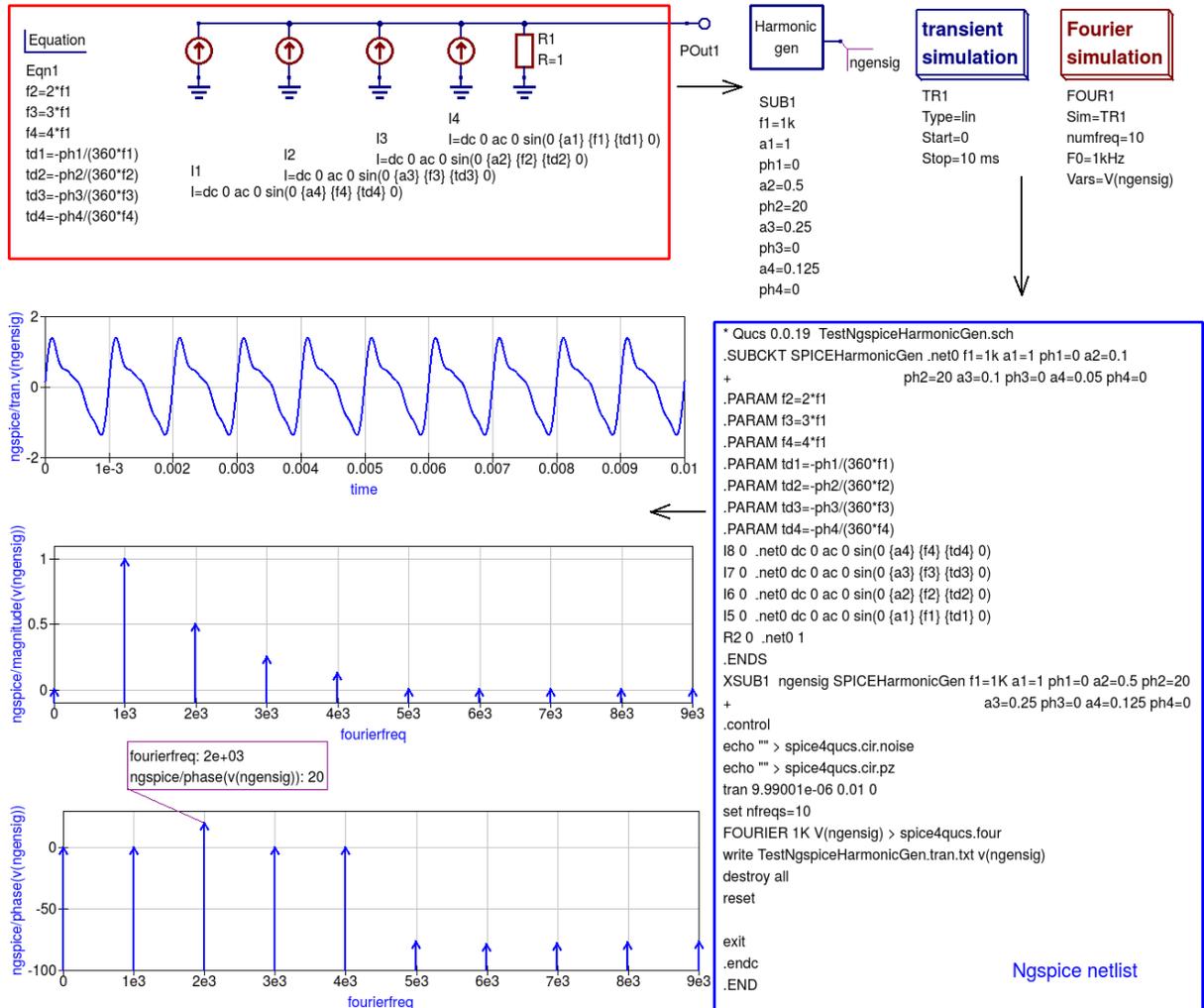


Figure 3.7 Ngspace subcircuit sinusoidal harmonic signal generator.

Figure 3.7 shows an Ngspace version of the Qucs sinusoidal harmonic generator illustrated in Figure 3.6. A casual look at these two subcircuit diagrams shows that they are not dissimilar. However, there are a number of subtle changes apparent from the diagrams. First it is important to realise that the Qucs and SPICE sinusoidal (sin) signal generator specifications are different; Qucs requires the signal phase and SPICE the signal delay to be specified as parameters. In Figure 3.7 extra equations to convert phase to time delay are added to Equation block Eqn1 inside subcircuit SPICEHarmonicGen. To ensure that Eqn1 variables, for example frequency $f2$, are passed to the subcircuit component values as numerical values SPICE curly delimitator brackets, {...}, are placed round equation variable names. Finally, it is important to realize that the order of the variables in Equation blocks are important. Qucs allows them to be in any order because it arranges all entries into a sequence which ensures each variable can be allocated a numerical value before it is used in other equations. However, SPICE does not do the same but assumes that all variables included in the right hand side of an equation have been allocated a numerical value prior to being used in the calculation of the variable named on the left hand side of the same equation. To check that the Ngspace generated waveform is correctly generated a Fourier analysis of signal $V(ngnsig)$ is displayed on Figure 3.7. At frequencies above $f4$ the phase values have no meaning. The simulated signal waveform obtained with SPICE OPUS was found to be similar to that obtained with NGSPICE, see Figure 3.8. Try simulating the sinusoidal harmonic generator waveform with SPICE OPUS to check this statement for your self.

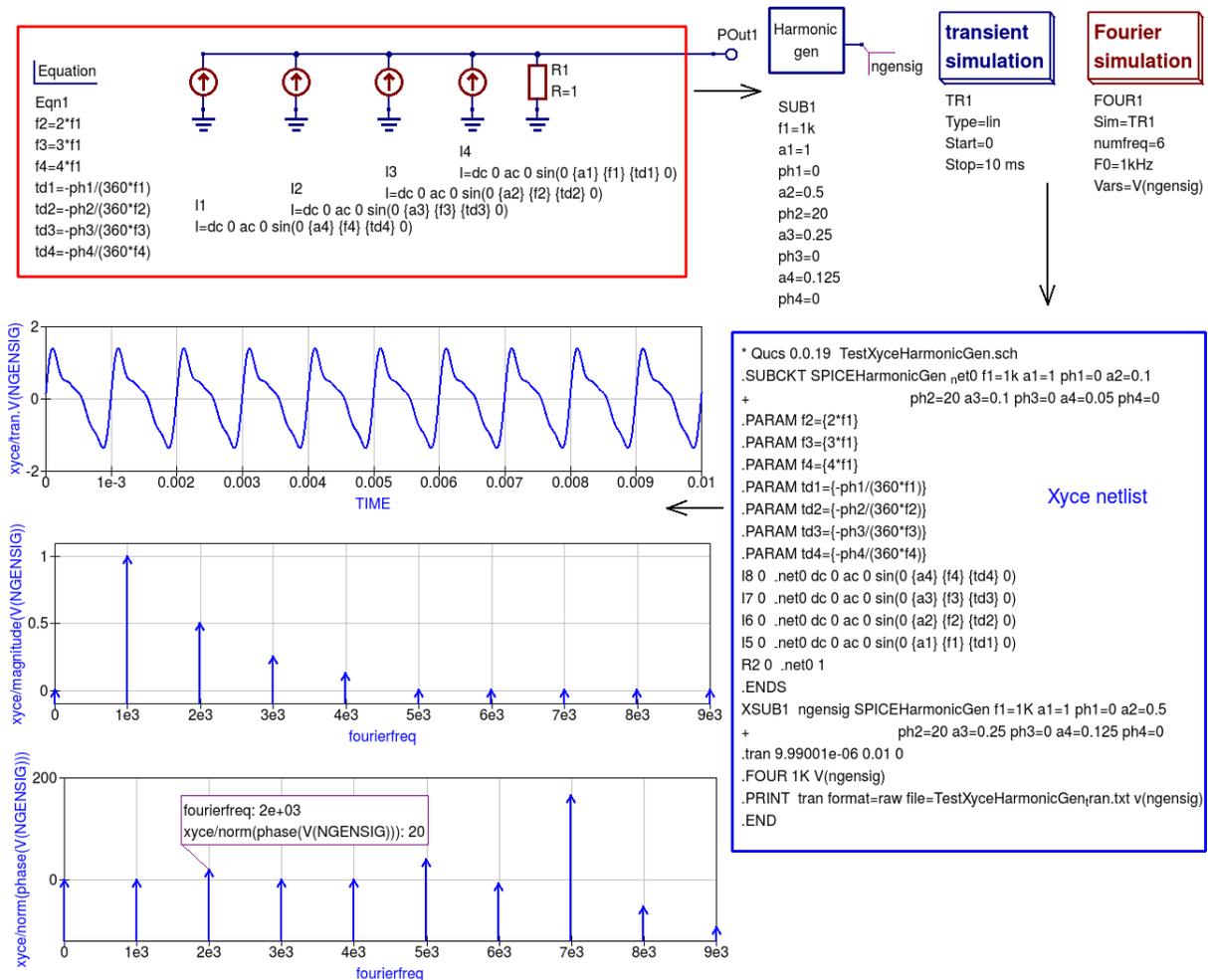


Figure 3.8 Xyce subcircuit sinusoidal harmonic signal generator.

3.1.3 A second more complex example of Spice4qucs subcircuits with parameters

Variable assignment equations, defined in Qucs *Equation Eqn* blocks and embedded in a subcircuit, are converted by Spice4qucs into SPICE `.PARAM` statements. These are listed in the initial section of the SPICE-netlist of the circuit being simulated, or in the first section of a subcircuit netlist, allowing their values to be determined before the start of a simulation. With Qucs *Equation Eqn* blocks it is important to remember that the variables defined cannot be functions of circuit voltage or current or any other voltage/current dependent properties. Restrictions placed by Spice4qucs on the use of Qucs *Equation Eqn* blocks are considered in detail in Chapter 4. However, one fundamental rule that must be followed at all times is that Qucs simulation icons must not be placed inside a subcircuit.

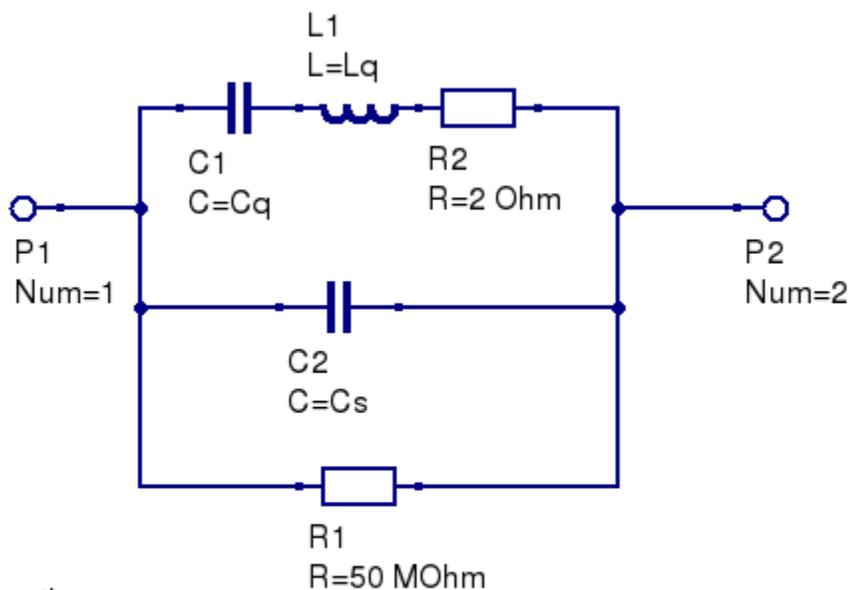
The electrical equivalent circuit of a HC-49/U 8.86 MHz Quartz crystal resonator is shown in Figure 3.9. In this model the crystal resonator is represented as the RCL parallel electric network illustrated in the following two schematics:

- `quarz.sch` — Quartz crystal resonator subcircuit; Figure 3.9.
- `quarz_test.sch` — Spice4qucs test circuit; Figure 3.10.

These files can be found in the Qucs-S subdirectory `examples\ngspice\`.

Figure 3.9 shows the crystal resonator subcircuit. A brief introduction to the theory of crystal resonators can be found

at https://en.wikipedia.org/wiki/Crystal_oscillator.



Equation

Eqn1

$$Cq=1/(4*\pi^2*f^2*Lq)$$

Figure 3.9 Equivalent circuit of Quartz crystal resonator.

In the HC-49/U Quartz crystal resonator model the RCL network has two resonant frequencies: a series resonance frequency f , where

$$f = \frac{1}{2\pi\sqrt{L_q C_q}}$$

and a parallel resonance frequency f_p , where

$$f_p = \frac{1}{2\pi\sqrt{L_q C_q}} \sqrt{1 + \frac{C_q}{C_s}}$$

Transposing equation f yields an expression for the series capacitance C_q , where

$$C_q = \frac{1}{4\pi^2 f^2 L_q^2}$$

This equation is placed in Qucs *Equation Eqn1* block inside the Quartz crystal resonator subcircuit.

Performing an *AC simulation* with Ngspice and Xyce, using the test circuit given in Figure 3.10, yields the amplitude response data plotted in Figure 3.11, Ngspice transfer coefficient K (ac.k) and Xyce voltage ac.V (OUT).

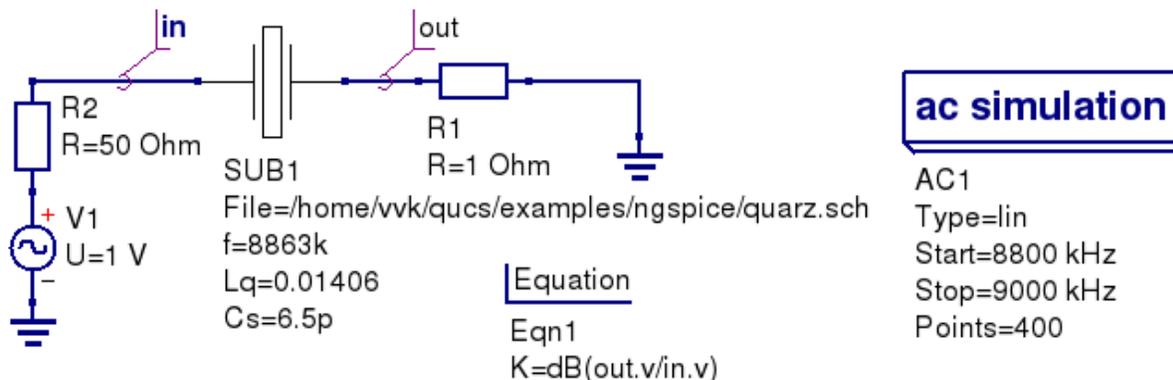


Figure 3.10 Test circuit for Quartz crystal resonator.

Figure 3.11 indicates that the Ngspace and Xyce plotted results are identical. The only difference being that Xyce simulation result postprocessing is not implemented. Hence, only the Xyce output voltage can be plotted; this is done by choosing a logarithmic Y scale, then the Xyce plot effectively displays a scaled decibel output. The two resonant frequencies f and f_p are clearly visible on these plots.

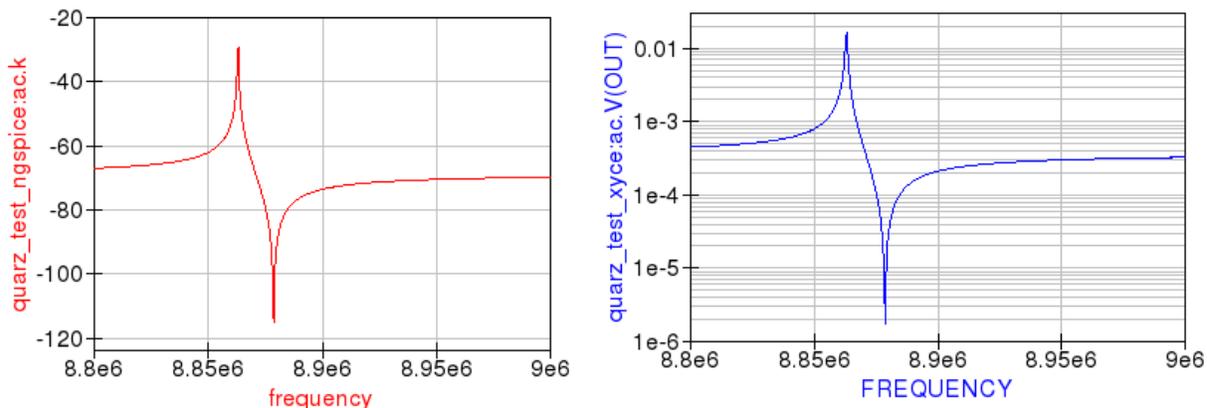


Figure 3.11 Magnitude response of HC-49/U Quartz crystal.

Subcircuits are converted by Splice4qucs into SPICE .SUBCKT routines. The SPICE netlist for the Quartz crystal resonator test circuit, Figure 3.10, shown below illustrates how the Splice4qucs handles SPICE .PARAM, .SUBCIRCUIT and subcircuit X call statements, placing them in the correct position within the SPICE netlist of the circuit being simulated.

```

1 * Qucs 0.0.19 /home/vvk/qucs/examples/ngspice/quarz_test.sch
2 * Qucs 0.0.19 /home/vvk/qucs/examples/ngspice/quarz.sch
3 .SUBCKT quartz _net0 _net1 f=8863k Lq=0.01406 Cs=6.5p
4 .PARAM Cq={1/(4*3.1415926539^2*f^2*Lq)}
5 R1 _net0 _net1 50MEG
6 C2 _net0 _net1 {CS}
7 R2 _net2 _net1 2
8 L1 _net3 _net2 {LQ}
9 C1 _net0 _net3 {CQ}
10 .ENDS
11 R1 out 0 1
12 V1 _net0 0 DC 0 SIN(0 1 1G 0 0) AC 1
13 R2 _net0 in 50
14 XSUB1 in out quartz f=8863K Lq=0.01406 Cs=6.5P

```

```
15 .control
16 set filetype=ascii
17 AC LIN 400 8800K 9000K
18 let K=dB(V(out)/V(in))
19 write quartz_test_ac.txt v(in) v(out) K
20 destroy all
21 exit
22 .endc
23 .END
```

3.2 Component and circuit libraries

Library components are supported in `spice4qucs` subsystem. You can use all library components. Some libraries have embedded original SPICE code of components. You should prefer to use these libraries to archive the best result with Spice simulation of Qucs schematics. The example of library component (IRFZ44 MOSFET from *MOSFETS* library) usage is `examples/ngspice/irfz44_switch.sch`

3.3 Using manufacturers component data libraries

Electronic components manufacturers often provide spice models of components in datasheets. You can attach these datasheet spice models using *SPICE netlist* component. You need to perform the following steps to use Spice-model from component datasheet. Spice netlist builder substitutes SPICE-models directly to output netlist without any conversions.

1. Extract Spice netlist text and save it as text file. You can use any extension for this file. Preferable are **.ckt** , **.cir** , **.sp**
2. Place on schematic component *SPICE netlist* and attach SPICE netlist nodes to component port using standard *SPICE component properties* dialog.
3. Simulate schematic with Ngspice/Xyce.

It's need to note that SPICE-netlist of component **must not** be ended by `.END` directive. In this case simulator exits after it reads `.SUBCKT` routine and simulation cannot be executed.

The example of spice model usage (LM358 opamp) is shown in the Figure 3.12

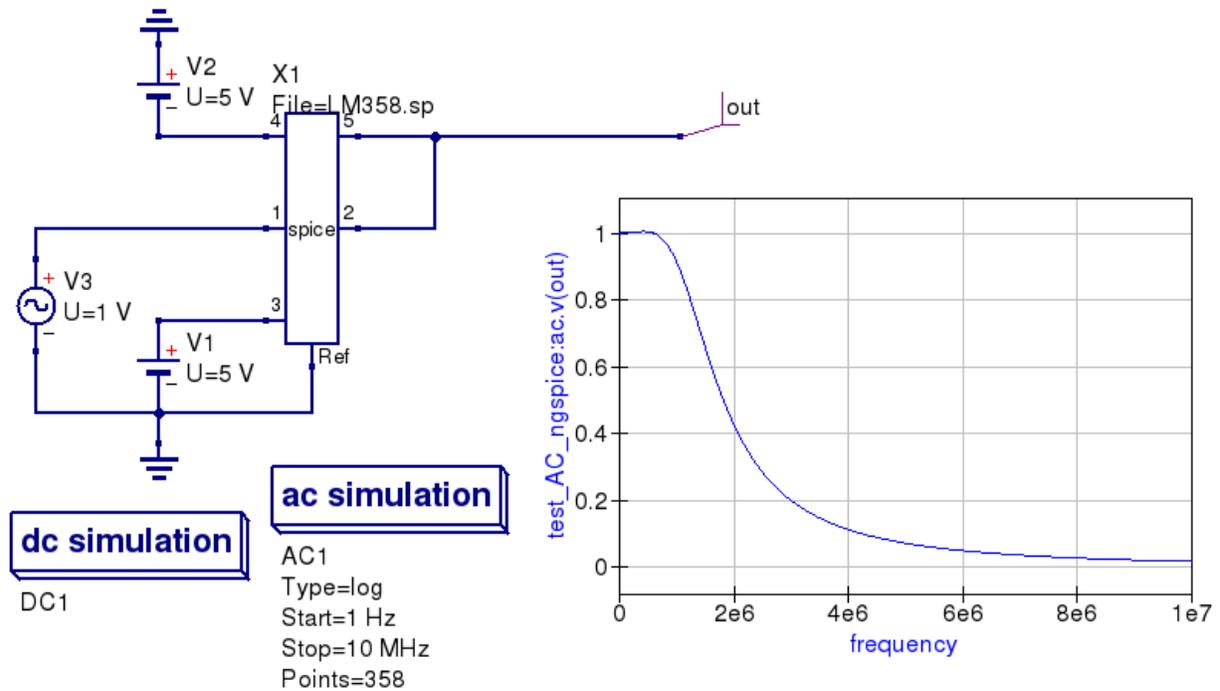


Figure 3.12 AC Simulation of LM358 opamp with Ngspice.

Here is the netlist of LM358 spice-model. Model can be found in LM358 datasheet.

```

1 * from http://www.ti.com/lit/zip/sloj045
2 * LM358 OPERATIONAL AMPLIFIER "MACROMODEL" SUBCIRCUIT
3 * CREATED USING PARTS RELEASE 4.01 ON 09/08/89 AT 10:54
4 * (REV N/A) SUPPLY VOLTAGE: +/-5V
5 * CONNECTIONS: NON-INVERTING INPUT
6 * | INVERTING INPUT
7 * | | POSITIVE POWER SUPPLY
8 * | | | NEGATIVE POWER SUPPLY
9 * | | | | OUTPUT
10 * | | | | |
11 .SUBCKT LM358 1 2 3 4 5
12 *
13 C1 11 12 5.544E-12
14 C2 6 7 20.00E-12
15 DC 5 53 DX
16 DE 54 5 DX
17 DLP 90 91 DX
18 DLN 92 90 DX
19 DP 4 3 DX
20 EGND 99 0 POLY(2) (3,0) (4,0) 0 .5 .5
21 FB 7 99 POLY(5) VB VC VE VLP VLN 0 15.91E6 -20E6 20E6 20E6 -20E6
22 GA 6 0 11 12 125.7E-6
23 GCM 0 6 10 99 7.067E-9
24 IEE 3 10 DC 10.04E-6
25 HLIM 90 0 VLIM 1K
26 Q1 11 2 13 QX
27 Q2 12 1 14 QX
28 R2 6 9 100.0E3
29 RC1 4 11 7.957E3

```

```

30 RC2  4 12 7.957E3
31 RE1 13 10 2.773E3
32 RE2 14 10 2.773E3
33 REE 10 99 19.92E6
34 RO1  8  5 50
35 RO2  7 99 50
36 RP   3  4 30.31E3
37 VB   9  0 DC 0
38 VC 3 53 DC 2.100
39 VE  54  4 DC .6
40 VLIM 7  8 DC 0
41 VLP  91  0 DC 40
42 VLN  0 92 DC 40
43 .MODEL DX D(IS=800.0E-18)
44 .MODEL QX PNP(IS=800.0E-18 BF=250)
45 .ENDS
46

```

3.4 Usage of unmodified SPICE Libraries

3.4.1 SpiceLibComp device

You can use an unmodified SPICE libraries with new `SpiceLibComp` device. This component could be found at the *File components* group. This component have three properties:

- `File` is full SPICE library file (usually `*.lib`, `*.cir`, or `*.sp` files) path. You can use unmodified library [here](#).
- `Device` is SUBCKT entry name that represents desired device. Every component is defined as subcircuit and identified by `.SUBCKT` entry name. This property holds device name. You need to fill this property manually.
- `SymPattern` is symbol pattern for device. You can select one of predefined symbol patterns or use automatic pattern. Automatic pattern is simple rectangular symbol with pins.

Let's consider SPICE library structure. There exists a SPICE library file `ad822.cir` that contains AD822 model. Here is library source code:

```

1  .SUBCKT AD822 1 2 99 50 25
2  *
3  * INPUT STAGE & POLE AT 5 MHZ
4  R3 5 99 2456
5  R4 6 99 2456
6  CIN 1 2 5E-12
7  C2 5 6 6.48E-12
8  I1 4 50 108E-6
9  IOS 1 2 1E-12
10 EOS 7 1 POLY(1) (12,98) 100E-6 1
11 J1 5 2 4 JX
12 J2 6 7 4 JX
13 GB1 50 2 POLY(3) (2,4) (2,5) (2,50) 0 1E-12 1E-12 1E-12
14 GB2 50 7 POLY(3) (7,4) (7,5) (7,50) 0 1E-12 1E-12 1E-12
15 *
16 * GAIN STAGE & POLE AT 13.4 HZ
17 EREF 98 0 (30,0) 1
18 R5 9 98 2.313E6
19 C3 9 25 32E-12
20 G1 98 9 (6,5) 4.07E-4

```

```

21 V1 8 98 0
22 V2 98 10 -1
23 D1 9 10 DX
24 D2 8 9 DX
25 *
26 * COMMON-MODE GAIN NETWORK WITH ZERO AT 1 KHZ
27 R21 11 12 1E6
28 R22 12 98 100
29 C14 11 12 159E-12
30 E13 11 98 POLY(2) (2,98) (1,98) 0 0.5 0.5
31 *
32 * POLE AT 10 MHZ
33 R23 18 98 1E6
34 C15 18 98 15.9E-15
35 G15 98 18 (9,98) 1E-6
36 *
37 * OUTPUT STAGE
38 ES 26 51 POLY(1) (18,98) 1.72 1
39 RS 26 22 500
40 V3 23 51 1.03951
41 V4 21 23 1.36
42 C16 20 25 2E-12
43 C17 24 25 2E-12
44 RG1 20 97 1E8
45 RG2 24 97 1E8
46 Q1 20 20 97 PNP
47 Q2 20 21 22 NPN
48 Q3 24 23 22 PNP
49 Q4 24 24 51 NPN
50 Q5 25 20 97 PNP 20
51 Q6 25 24 51 NPN 20
52 VP 96 97 0
53 VN 51 52 0
54 EP 96 0 POLY(1) (99,0) 0.01 1
55 EN 52 0 POLY(1) (50,0) -0.015 1
56 R25 30 99 63.5E3
57 R26 30 50 63.5E3
58 FSY1 99 0 VP 1
59 FSY2 0 50 VN 1
60 *
61 * MODELS USED
62 *
63 .MODEL JX NJF (BETA=7.67E-4 VTO=-2.000 IS=1E-12)
64 .MODEL NPN NPN (BF=120 VAF=150 VAR=15 RB=2E3 RE=4 RC=200)
65 .MODEL PNP PNP (BF=120 VAF=150 VAR=15 RB=2E3 RE=4 RC=900)
66 .MODEL DX D (IS=1E-15)
67 .ENDS AD822

```

This library example contains only one model defined by one subcircuit entry, but you can use any library containing any amount of device models.

Let's use AD822 opamp model. Create new schematic and place `SpiceLibComp` device on schematic (Figure 3.13). Select `ad822.cir` file in the first property. Then fill `ad822` (device name) in the second property.

You can either create an automatic component symbol, either use one of the predefined patterns. At current state only `opamp3t` and `opamp5t` patterns are available. These patterns represents three- and five-terminal opamps respectively. Symbol patterns are Qucs XML files. They are placed in the `share/qucs/symbols` subdirectory of the Qucs installation root. These files have `*.sym` extension. Symbol pattern format will be considered further.

SPICE netlist builder performs automatic port assignment for subcircuit pins. If automatic symbol is used symbol pin names will be automatically filled from the `.SUBCKT` entry definition. See Figure 3.13 for example of the automatic pin assignment.

If symbol pattern is used, the first `.SUBCKT` entry port will be automatically mapped to the first symbol port, etc. Symbol port sequence is defined in the symbol pattern file (`*.sym`) in Port description lines.

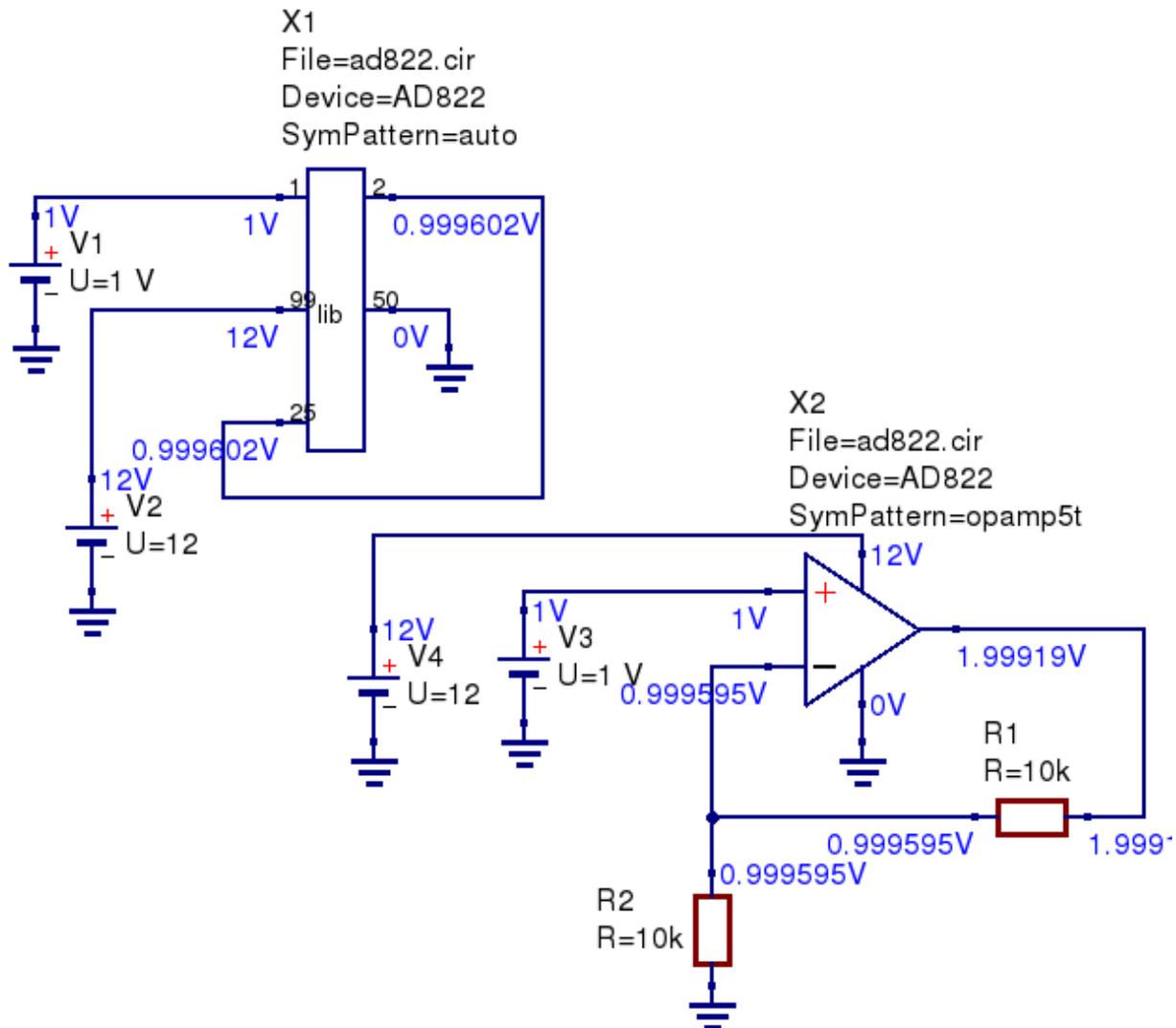


Figure 3.13 LM358 opamp library model usage with `SpiceLibComp` device

3.4.2 Symbol pattern files format description

Let's consider symbol files format. Symbols have `*.sym` extension and are placed in `share/qucs/symbols` subdirectory of the Qucs installation tree. Qucs automatically scans content of this subdirectory and displays all found valid symbols in drop-down list in the third property (`SymPattern`) of the `SpiceLibComp` device. User can select any symbol for new SPICE library device. It's need to create a new symbol file and place it into symbols directory to add new symbols to the existing Qucs installation.

Let's consider symbol file format. Symbols have Qucs XML schematic format without header. An example of symbol

file (five-terminal opamp) is shown in the listing below:

```

1 <Symbol>
2 <Line -20 -40 0 80 #000080 2 1>
3 <Line -20 -40 60 40 #000080 2 1>
4 <Line -20 40 60 -40 #000080 2 1>
5 <Line 40 0 20 0 #000080 2 1>
6 <Line -40 -20 20 0 #000080 2 1>
7 <Line -40 20 20 0 #000080 2 1>
8 <Line -15 20 10 0 #000000 2 1>
9 <Line -10 -25 0 10 #ff0000 0 1>
10 <Line -15 -20 10 0 #ff0000 0 1>
11 <Line 10 -20 0 -20 #000080 2 1>
12 <Line 10 20 0 20 #000080 2 1>
13 <.PortSym 10 40 4 0>
14 <.PortSym 10 -40 3 0>
15 <.PortSym -40 20 2 0>
16 <.PortSym -40 -20 1 0>
17 <.PortSym 60 0 5 180>
18 <.ID 30 24 OP>
19 </Symbol>

```

Automatic symbol files preparation is not yet implemented, but you can use Qucs schematic editor to create new symbol files. You may use the following sequence to create new symbol:

- Create Qucs subcircuit. Subcircuit may be empty. Place desired ports on it;
- Attach symbol to it using switching to symbol mode by F9 keystroke. Wire subcircuit ports to symbol and paint symbol outline.
- Save subcircuit, open it with any test editor and copy-paste symbol code form it into the symbol file.

Please pay attention to the proper port mapping. Let's consider port definition line format:

```
<.PortSym 10 -40 3 0>
```

This port definition consists of five space separated fields. The fourth field (3) contains port number. This port number should match SPICE `.SUBCKT` port number (not port name!) to proper component wiring. You may need to edit this field manually.

For example AD822 has the following definition in our library:

```
.SUBCKT AD822 1 2 99 50 25
```

Subcircuit node list follows after the subcircuit name (`AD822`). Subcircuit nodes will be mapped to component port in the following sequence:

- Node 1 — to Port 1
- Node 2 — to Port 2
- Node 99 — to Port 3
- Node 50 — to Port 4
- Node 25 — to Port 5

3.5 Usage of the whole SPICE library

Qucs-S supports usage of the whole SPICE libraries. Such libraries will be visible in the **QucsLib** tool and left-side **Library** dock. Library modification will be not required, but user may need to attach components symbols as resource

files.

Let's consider how to use it. SPICE library again will be treated as a set of `.SUBCKT` entries. You should give `*.lib` extension an existing SPICE library and put in into `$HOME/.qucs/user_lib` or system Qucs library directory (for example `/usr/share/qucs-s/library` for Unix). Then you can get access to this newly added SPICE library via QucsLib tool or from the left-side dock. You will see its name and component list (Figure 3.14).

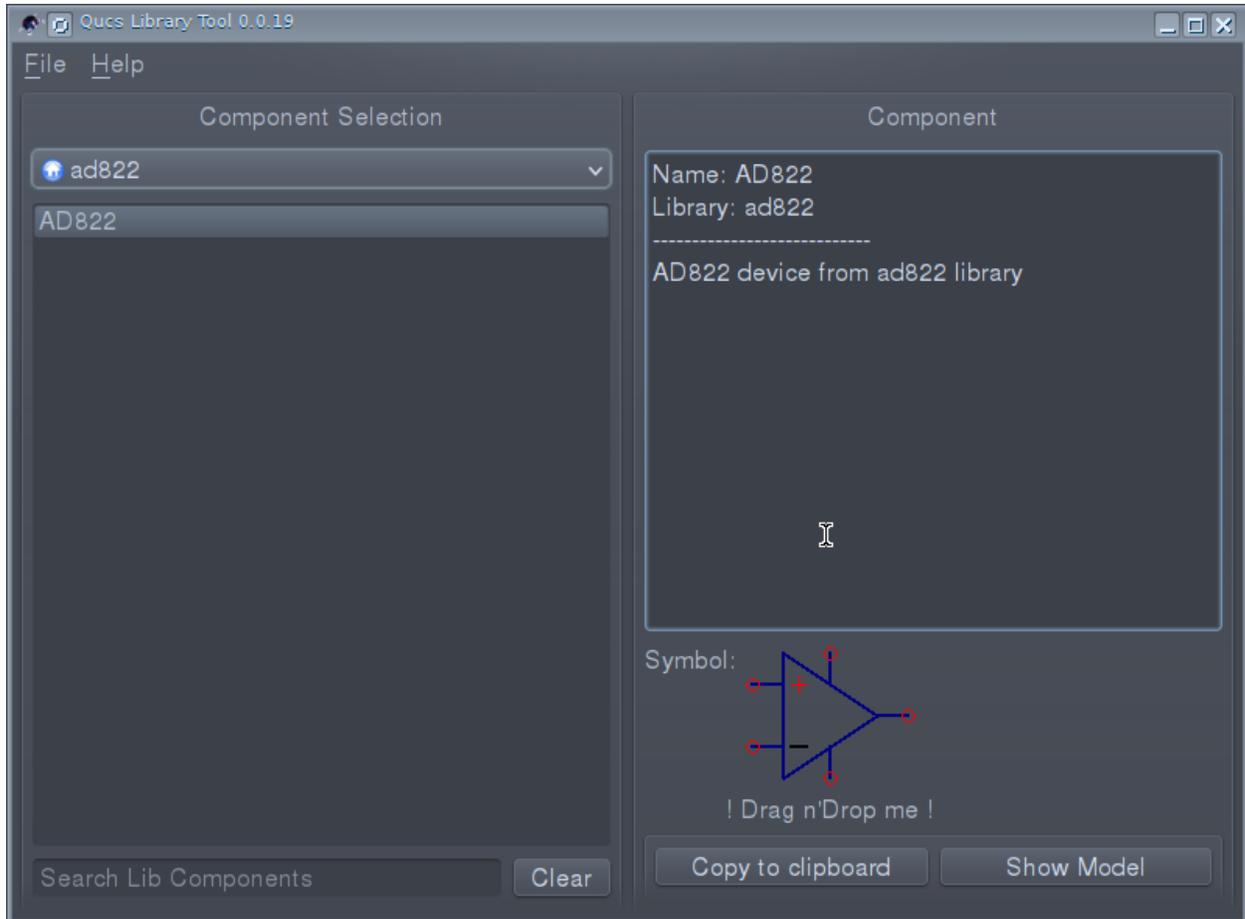


Figure 3.14 An example of a SPICE library view in the Qucs library manager.

Every `.SUBCKT` found is considered as a single component. It will be shown in the library manager and it will be available for drag'n'drop. Subcircuits are available via an existing `SpiceLibComp` component.

Default symbol will be created, if there is no symbol attached to component. But you can attach user symbol to every component. Symbol file (Qucs XML) should be placed at the `library_name` subdirectory. For example, you should create `opamps` subdirectory for `opamps.lib` file and put all necessary symbol files into it. This location is used also for other resources such as XSPICE CodeModel sources (`*.mod` and `*.ifs` files). Symbol file `*.sym` format is considered in the previous section. Two symbol types are implemented:

1. Default symbol for all components in library. It should be placed at `library_name.sym` file. For example `opamps.sym` will be treated as the default symbol for `opamps.lib` library.
2. Symbol for every component (`component_name.sym` file). For example, `LM358.sym` will be mapped to `LM358` component.

Default symbol will be substituted, if component symbol file is not found.

Please keep in mind that SPICE subcircuit names are case-insensitive, but symbol file names may be case-sensitive for some specific platforms. In other words `ad822.sym` may not work for `AD822` component. But `AD822.sym` will be attached properly.

You can look at SPICE library example with attached symbols here: https://github.com/ra3xdh/qucs_spicelib Clone this repository and copy `ad822.lib` file and `ad822` subdirectory into `$HOME/.qucs/user_lib` directory. This library contains one component (AD822 opamp).

Let's consider library and resource files tree:

```
qucs_spicelib/  
- ad822  
  |   - ad822.sym  
- ad822.lib
```

As you can see, resources are placed into `ad822` subdirectory. It contains one default symbol file `ad822.sym` that is placed in resource subdirectory.

3.6 Libraries blacklisting

Every library may consist of simulator-incompatible components. For example XSPICE devices will not work with Xyce backend. And Qucsator microwave devices will not work with any of SPICE.

Library blacklisting serves to hide simulator incompatible libraries in Qucs-S library manager view. User will not see such libraries. This prevents simulator compatibility issue.

There are three `*.blacklist` files in the system Qucs-S library directory (for example `$QUCSDIR/share/qucs-s/library/`). All files have plain text format and contain a list of simulator-incompatible library files with extension in the current directory. File entries are newline-separated. These libraries will be blacklisted and will be not shown in the library manager if appropriate simulator is selected as default.

Here is blacklist files list:

- `ngspice.blacklist` — The list of Ngspice-incompatible libraries;
- `xyce.blacklist` — The list of XYCE-incompatible libraries;
- `qucs.blacklist` — The list of Qucsator-incompatible libraries;

For example, let's consider contents of the `qucs.blacklist` file:

```
AnalogueCM.lib  
Cores.lib  
Transformers.lib  
Xanalogue.lib
```

All of these libraries contain SPICE-only components (XSPICE analogue blocks) and will not work with Qucsator.

back to the top

Chapter 4. Device and component modelling with algebraic equations

4.1 The role of algebraic and numeric equations in circuit simulation

Algebraic/numerical equations play the following important roles in circuit simulation:

1. Circuit device parametrization,
2. Post-processing of simulation data, and
3. Definition of user-defined components.

With the spice4qucs subsystem the first and second operations in the above list and the third item can be performed easily.

Spice4qucs supports the following algebraic/numeric equations:

1. The usual Qucs equations. These are converted automatically to SPICE `.PARAM` statements and `ngnutmeg` scripts. Equations that don't include simulation variables (for example, node voltages and device currents) are passed as `.PARAM` statements in the generated SPICE netlist. In contrast equations that include one or more simulation variables are placed in the SPICE simulation file between the `ngspice .control` and `.endc` statements. The `.control` and `.endc` block is normally located following one or more SPICE simulation commands. Please NOTE equations which include simulation variables can only be processed with `ngspice` because Xyce is not equipped with a suitable post-processor for this purpose.
2. `.PARAM` items. Such statements are passed directly as a `.PARAM` entry in a generated SPICE netlist.
3. `.GLOBAL_PARAM` sections. This feature works in the same manner as a `.PARAM` item.
4. `.OPTIONS` sections. This feature provides a way of changing the value of internal `ngspice` or Xyce defined variables, such as, for example `GMIN`.
5. `Ngnutmeg` scripts. These are directly passed to the `ngnutmeg` post-processor after simulation. Again please note this feature is not supported by Xyce.

Icons representing the last four equation types can be found in the *Spice sections* group of the *Components* palette.

4.2 Qucs equations usage with ngspice and Xyce

Ngspice fully supports equations. Only mathematical functions related to S-parameter simulation will (such as `stoz()`, `stoy()`, etc.) are not implemented. Complex number arithmetic is also supported. Similarly, physical constants (`q`, `kB`, etc.) are also allowed. Qucs equations can be used for both parametrization and post-processing purposes. However, please remember that SPICE variable names are not case sensitive but Qucsator is case dependent. Hence, `Kv` and `KV` refer to different variables for Qucsator and the same variable for `ngspice` and Xyce. Algebraic expressions are evaluated from the left to right with brackets, in the normal fashion, determining evaluation order.

As mentioned above, equations that contain simulator variables, such as node voltages and device currents, are converted into `ngnutmeg` script. The following example (complex power calculation in RC-circuits, Fig 4.1) illustrates the usage of Qucs equations with `ngspice`. This circuit simulates correctly with both `ngspice` and Qucsator.

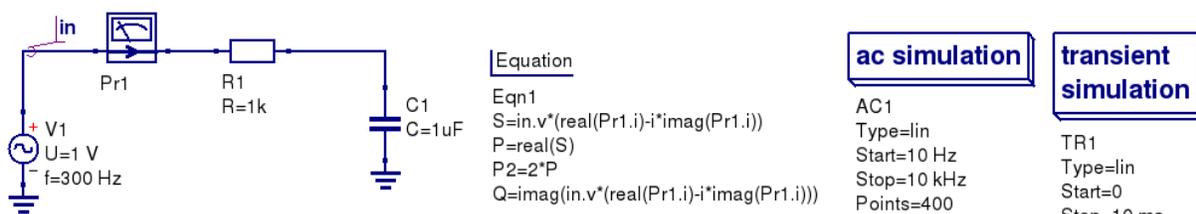


Figure 4.1 Total power in an RC-circuit.

It's need to evaluate the following equations:

Total power

$$S = abs(\dot{U} \cdot \dot{I})$$

Active power:

$$P = \Re[\dot{U} \cdot \dot{I}]$$

Reactive power

$$Q = \Im[\dot{U} \cdot \dot{I}]$$

The simulated results are shown in the Figure 4.2.

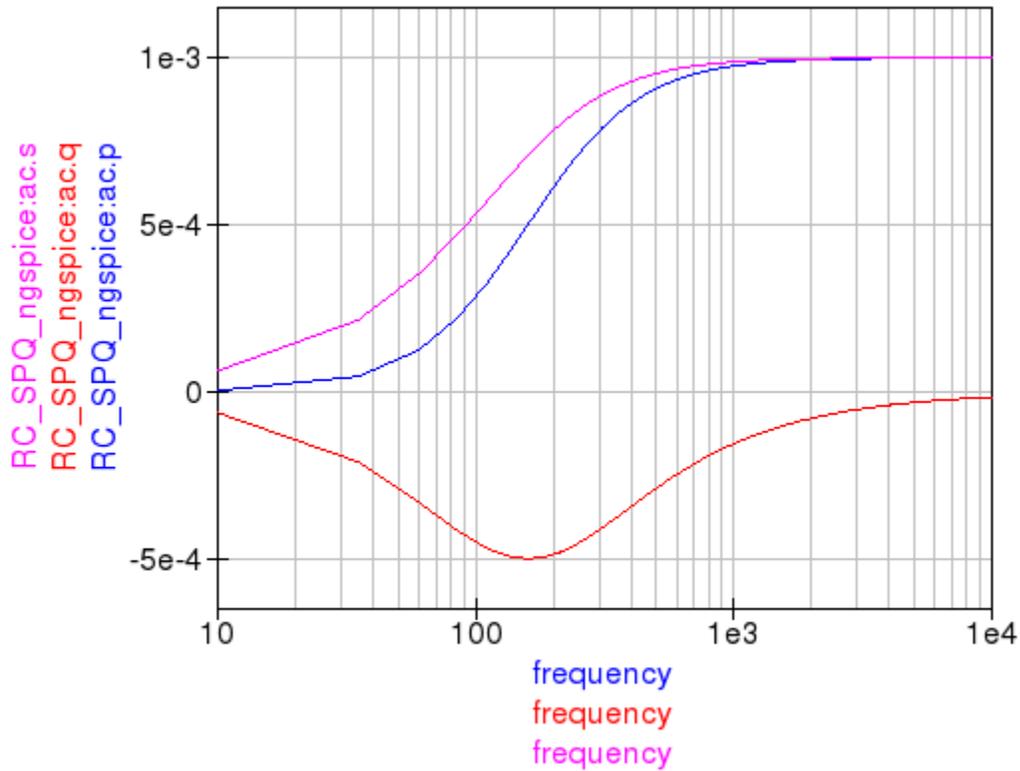


Figure 4.2 Total, active, and reactive power curves.

The Xyce circuit simulator is distributed without a post-simulation data processor like ngnutmeg. Unfortunately, this implies that Xyce can only partial process Qucs equations. In particularly, Xyce only supports parametrization. Algebraic equations that include node voltages or device currents are ignored by Xyce.

Here is a small example. It illustrates how parametrization could be used with all of three simulation back ends. Parametrisation is used to estimate the resonant frequency f_{res} of the RCL circuit:

$$f_{res} = \frac{1}{2\pi\sqrt{LC}}$$

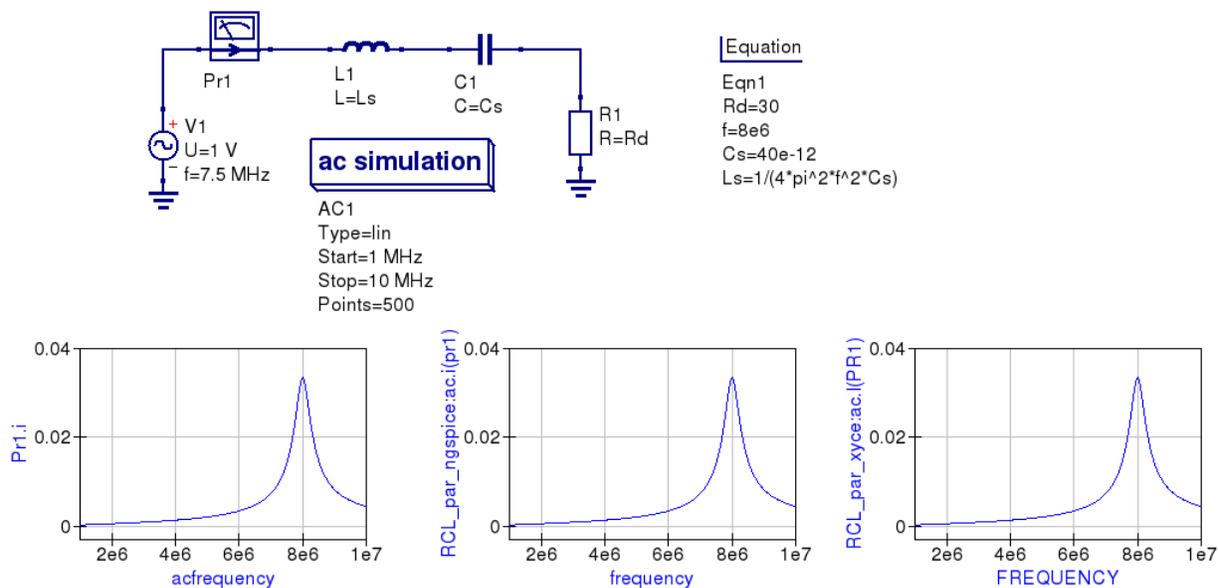


Figure 4.3 Parametrised RCL-circuit.

You can see that simulation results for all three circuits are identical.

NOTE:

There is an important note on equations order. SPICE simulators are susceptible to equations order, but equations order has not matter for Qucsator. It concerns both parametric equations and postprocessor equations. `Spice4qucs` don't care on equations order. User should select proper equations order to avoid `Undefined variable...` simulation errors. This concern also two or more `Equation` components on schematic.

4.3 Manipulating simulation data with algebraic measurement scripts

Post-processing of the simulation data is very important feature of a circuit simulator. There are two general ways employ data postprocessing with `ngspice` and `qucsator`.

Firstly, a special component *Nutmeg Equation* has been implemented. It works in a similar fashion to the established Qucs Equation component. It's properties dialog (Fig.4.4) is opened by double clicking on the *Nutmeg Equation* icon.

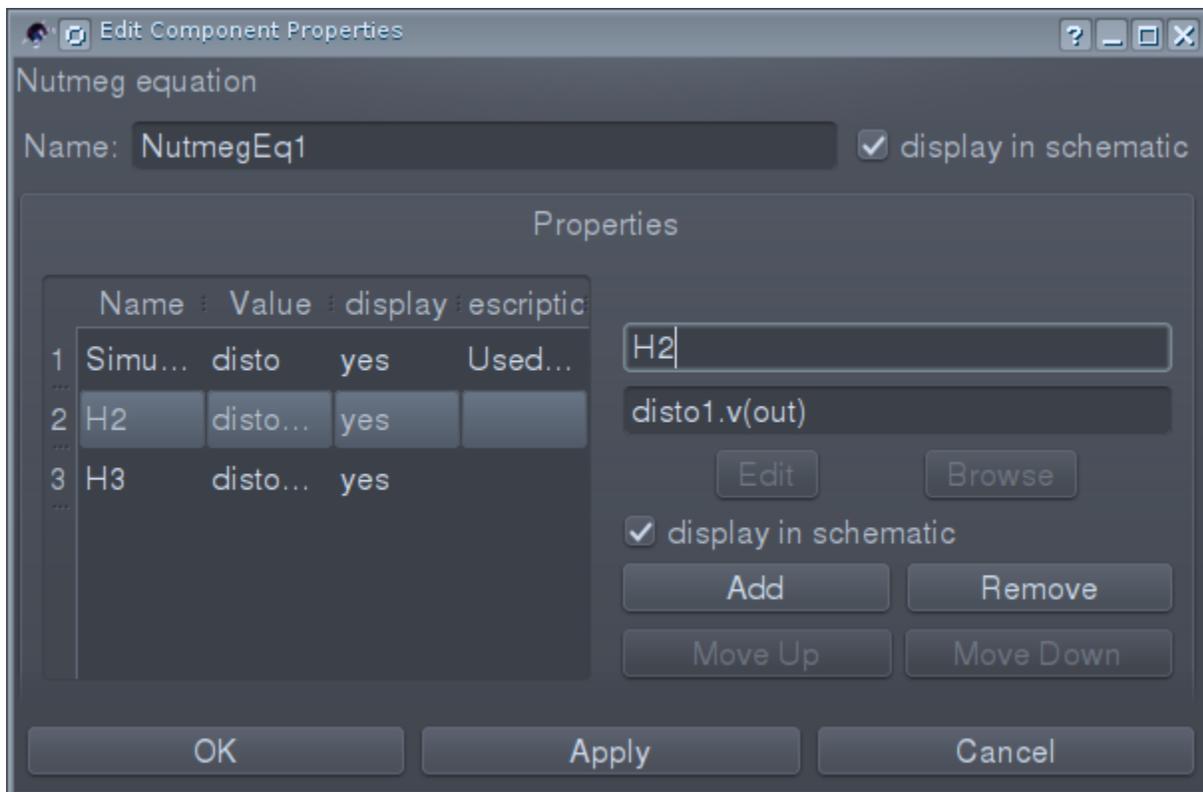


Figure 4.4 The *Ngnutmeg Equation* properties dialog.

You need to specify (as first parameter) the type of simulation to which the *ngnutmeg* script is be linked. The following simulation types are allowed:

- AC
- TRAN
- DISTORTION
- DC
- All simulations

If type “*All simulations*” is selected, equations will be evaluated for all simulations. You should use the standard SPICE notation for node voltages and device current, for example; node voltages are specified as $v(\text{node})$ or $V(n1, n2)$. In a similar fashion probe currents are specified in SPICE terms as $VPr1\#branch$ which represents the current flowing in Qucs probe named $Pr1$. *Spic4qucs* allows the use of all of the *ngnutmeg* functions and operators without any limitations. However, please take into account that variables in *ngnutmeg* equations are case independent!

All other equations/parameters form *ngnutmeg* equations. These are converted into *ngnutmeg* `let` statements:

```
let Var1 = Expression1
let Var2 = Expression2
let VarN = ExpressionN
```

Expressions are evaluated from the first to last with brackets determining the order of priority. You should take into account expression order when writing *ngspice* equations.

The following example (Fig.4.5) illustrates how the two equation types are used.

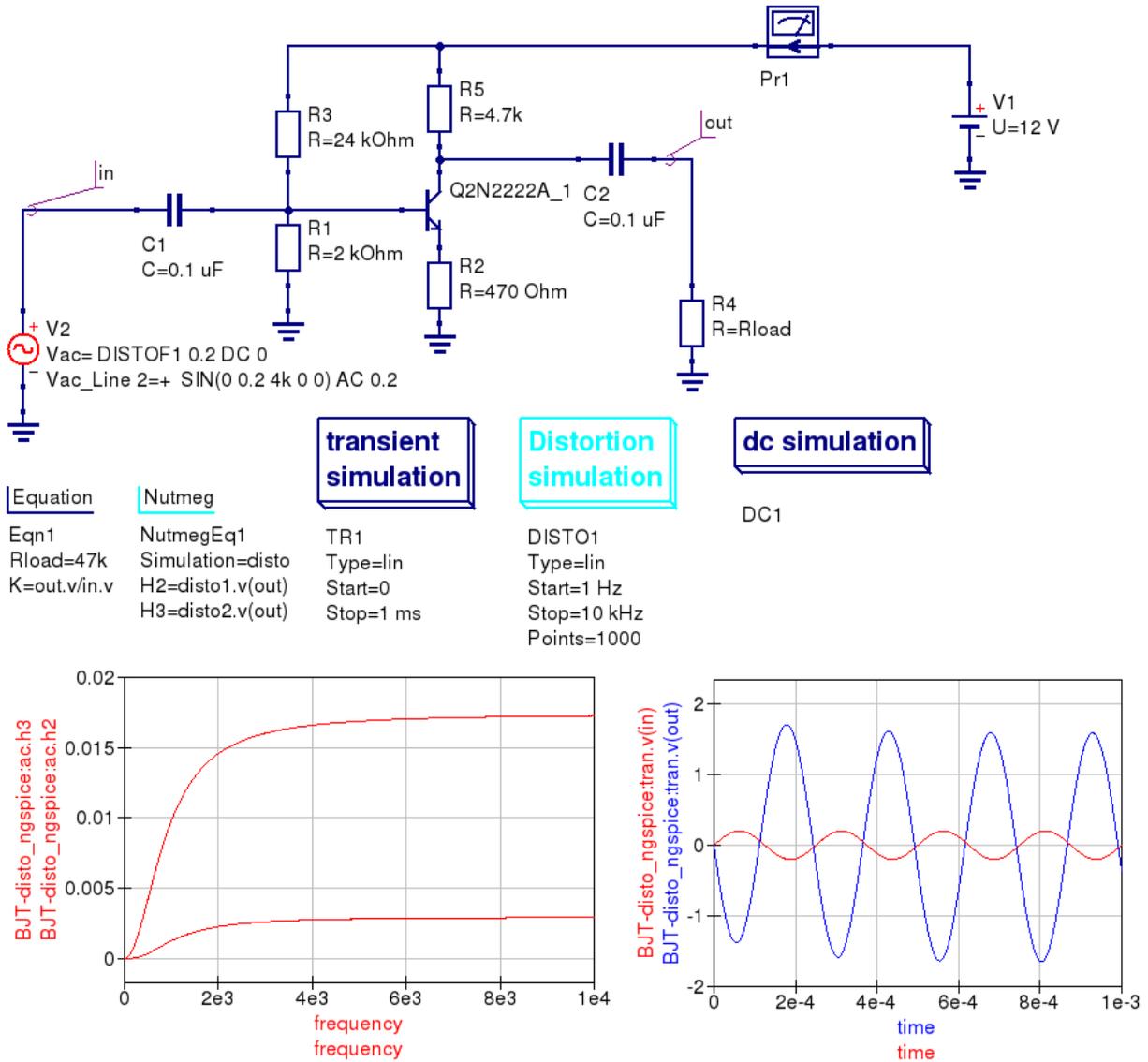


Figure 4.5 Distortion analysis with ngnutmeg simulation data postprocessing.

The second way of postprocessing simulation output data uses the normal Qucs *Equation* component. However, please note that spice4qucs allows the use of SPICE notation in Qucs equations. The following example shows how this feature can be utilized.

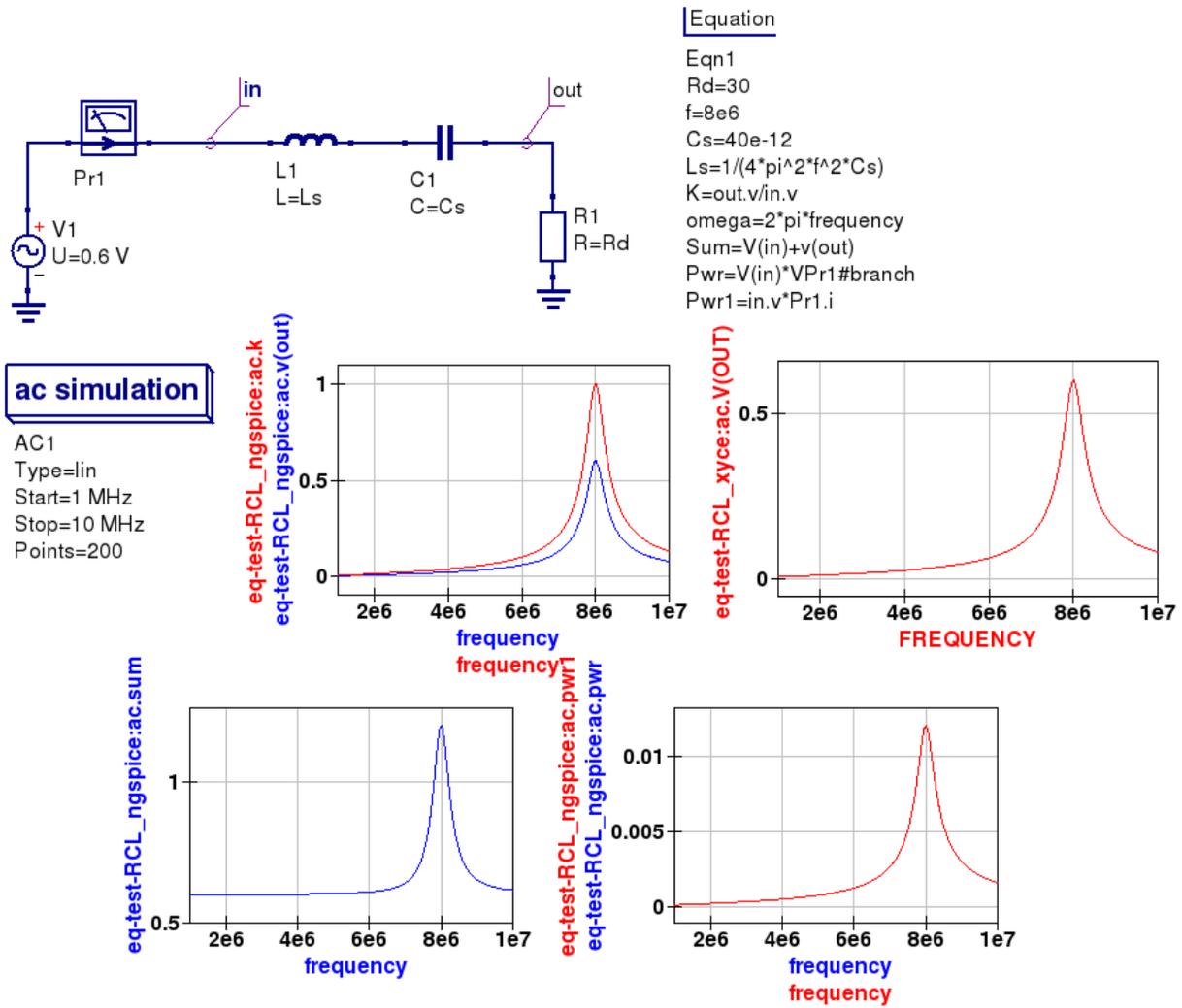


Figure 4.6 Using SPICE notation in Qucs equations.

4.4 Qucs Equation-Defined Device (EDD) models

Qucs EDD models are described by current equations and charge equations. EDD has N branches. Current equations bind current I_N flowing through a branch with voltage V_N across branch N :

to

$$I_1 = f_1(V_1, \dots, V_N)$$

...

$$I_N = f_N(V_1, \dots, V_N)$$

Charge equations bind charge Q_N accumulated by a branch with voltage V_N across branch N and current I_N flowing through branch N :

to

$$Q_1 = g_1(V_1, \dots, V_N)$$

...

$$Q_N = g_N(V_1, \dots, V_N, I_1, \dots, I_N)$$

Qucs equation notation must be used in EDD equations. Qucs notation is converted to SPICE notation automatically, where the Qucs EDD function is synthesised by a SPICE netlist builder to form an electrical equivalent circuit built around SPICE B-type sources.

The Spice4qucs subsystem supports both EDD current and charge equations. You can simulate EDD models with ngspice and Xyce without any special adaptations. All SPICE mathematical functions are allowed. The following examples (Figures 4.7 and 4.8) demonstrate how EDD based circuits are simulated.

The first example illustrates a set of IV-curves for a Tunnel diode, where the Tunnel diode IV-curve is approximated by the following equation:

$$I = I_s \left(e^{\frac{V}{\varphi T}} - 1 \right) + I_v e^{k(V-V_v)} + I_p \cdot \frac{V}{V_p} e^{\frac{V_p-V}{V_p}}$$

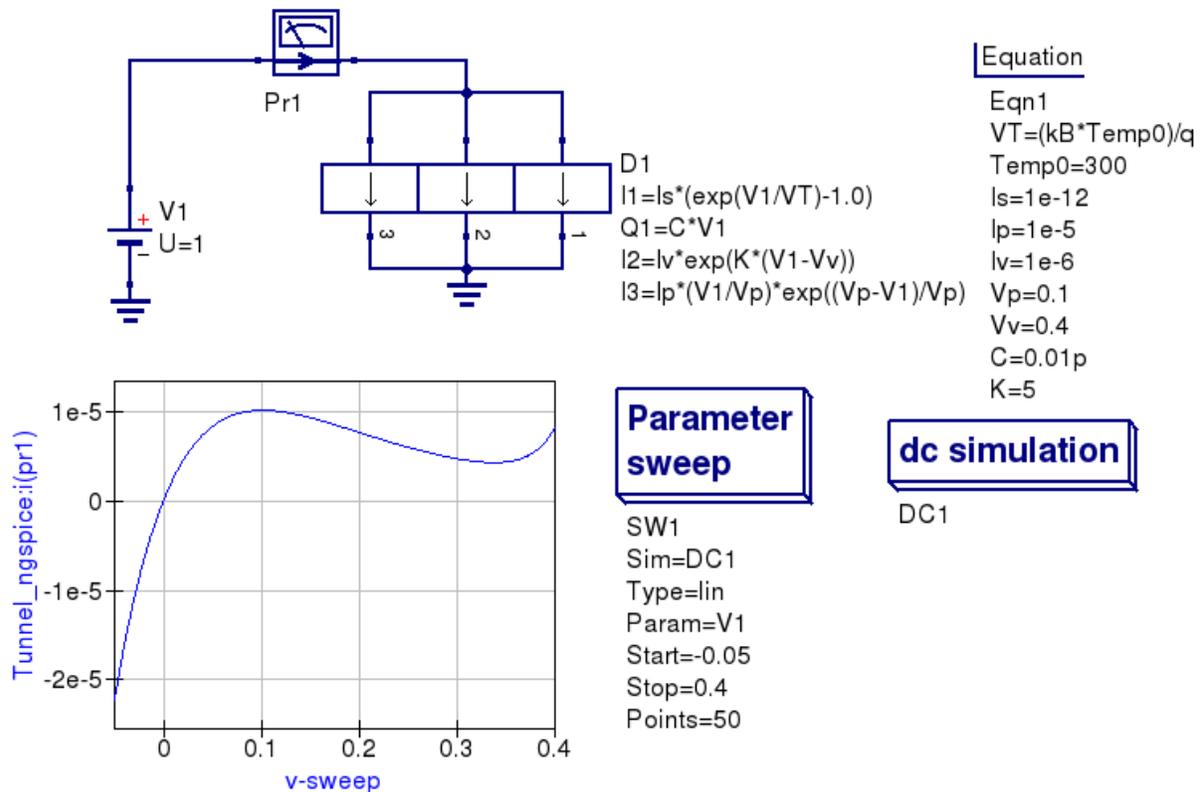


Figure 4.7 Tunnel diode simulation using an EDD compact device model.

The second example illustrates how a nonlinear capacitor can be approximated by a polynomial that binds capacitor charge Q with applied voltage V

$$Q = C_1 V + \frac{C_2 V^2}{2} + \frac{C_3 V^3}{3} + \dots + \frac{C_N V^N}{N}$$

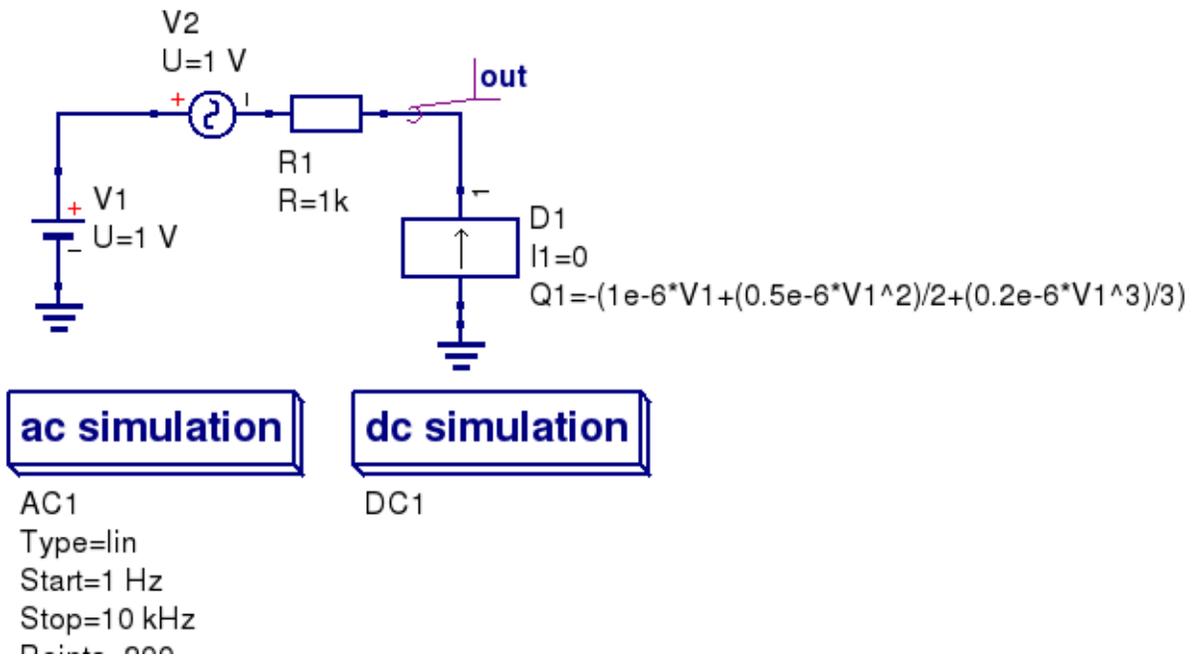


Figure 4.8 A non-linear capacitor simulation using ngspice and Xyce

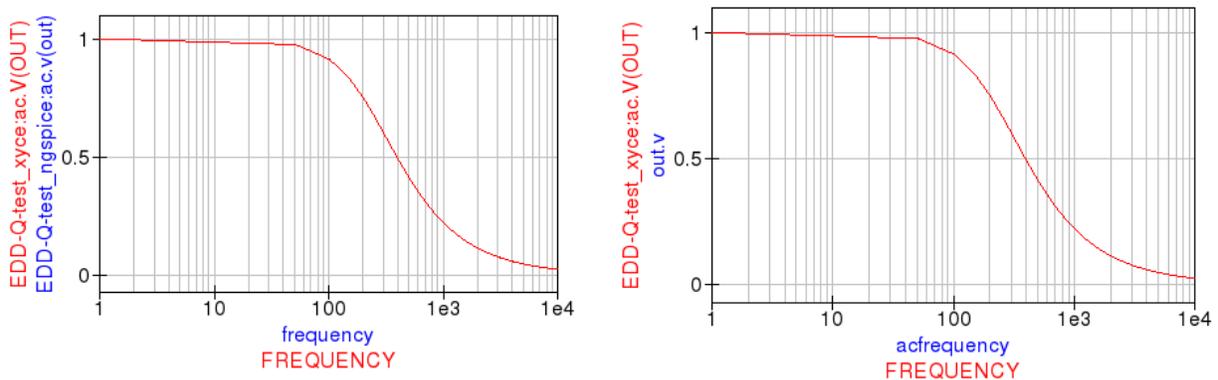


Figure 4.9 The magnitude response of an RC circuit with a non-linear capacitor.

The spice4qucs special component called *Equation defined source* could be used as alternative to the Qucs EDD. This component is located in group *Spice components*. It implements a behavioural B-type SPICE voltage or current source. See chapter 5.1 of ngspice manual for more information. The example introduced in Figure 4.10 shows how this source is used. Please note that SPICE notation must be used with B-source expressions.

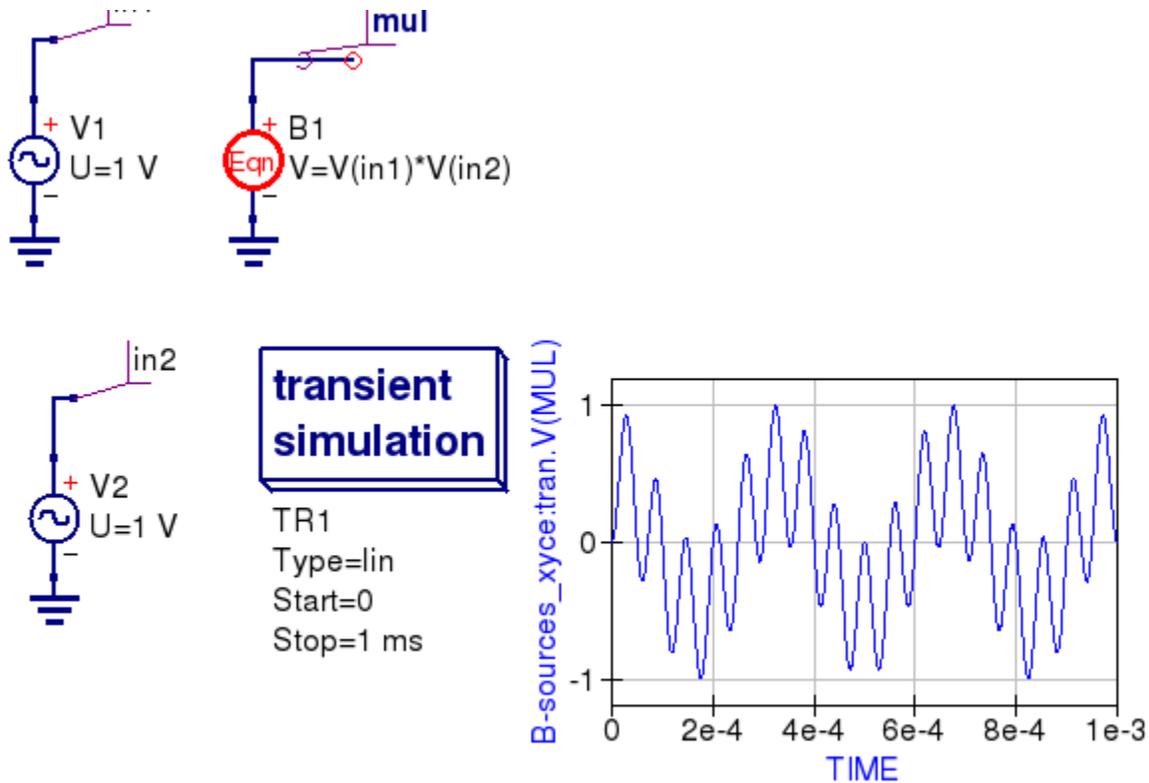


Figure 4.10 A voltage multiplier model with B-type sources.

4.5 Qucs Frequency Equation-Defined Device (FEDD) models

Qucs RFEDD devices is not yet supported by spice4qucs. As a temporary work around behavioural R,C,L models and B-type sources should be used instead. Moreover, the ngspice `hertz` variable is defined to represent signal frequency, allowing models with the same function as the Qucs RFEDD model to be synthesised. Refer to chapters 3.2.4, 3.2.8, 3.2.12, and 5.1 of the official ngspice manual for further information.

4.6 ngspice and Xyce translation/synthesis of EDD and FEDD models

back to the top

Chapter 5. More advanced circuit simulation techniques.

5.1 Fourier simulation

The Qucs-S implementation of *Fourier simulation* allows users to perform a Fourier analysis of one or more time domain circuit signals and to investigate their spectrum in the frequency domain. Qucs-S *Fourier simulation* is implemented by Ngspice, Xyce and SPICE OPUS. Fourier simulation is available to Qucs-S users via a special icon called *Fourier simulation*. This icon is located in the *simulations* group. To request a *Fourier simulation* place a copy of the “*Fourier simulation*” icon on the current work schematic alongside a *transient simulation* icon. Qucs-S *Fourier simulation* uses the simulation data generated by a *transient simulation* and has no meaning without a set of transient

time domain data. The link between *Fourier simulation* and *transient simulation* is formed by entering the name of the coupled *transient simulation* as the first property of *Fourier simulation*.

The *Fourier analysis* property list has the following entries:

1. `Sim` — Linked transient simulation icon name.
2. `numfreq` — Number of harmonics: variable number for ngspice and SPICE OPUS but fixed at 10 for Xyce.
3. `F0` — This parameter is the fundamental frequency of the generated frequency domain spectrum.
4. `Vars` — This parameter is a list of output signals. These may be node voltages and currents. In the list each entry must be space separated.

Fourier simulation creates four output vectors for each specified output signal, for example in the case of signal `v(out)`:

1. `magnitude(v(out))` — Magnitude spectrum.
2. `phase(v(out))` — Phase spectrum (in degrees-).
3. `norm(mag(v(out)))` — Normalized magnitude spectrum.
4. `norm(phase(v(out)))` — Normalized phase spectrum.

Qucs-S allows each of these four display vectors to be plotted.

Here is a small example of a *Fourier simulation* which demonstrates the main features introduced above and the relation between small signal AC simulation and *Fourier simulation*.

Figure 5.1 Fourier and small signal AC analysis of a single stage transistor amplifier.

5.1.1 Additional Ngspice, SPICE OPUS and Xyce *Fourier simulation* examples

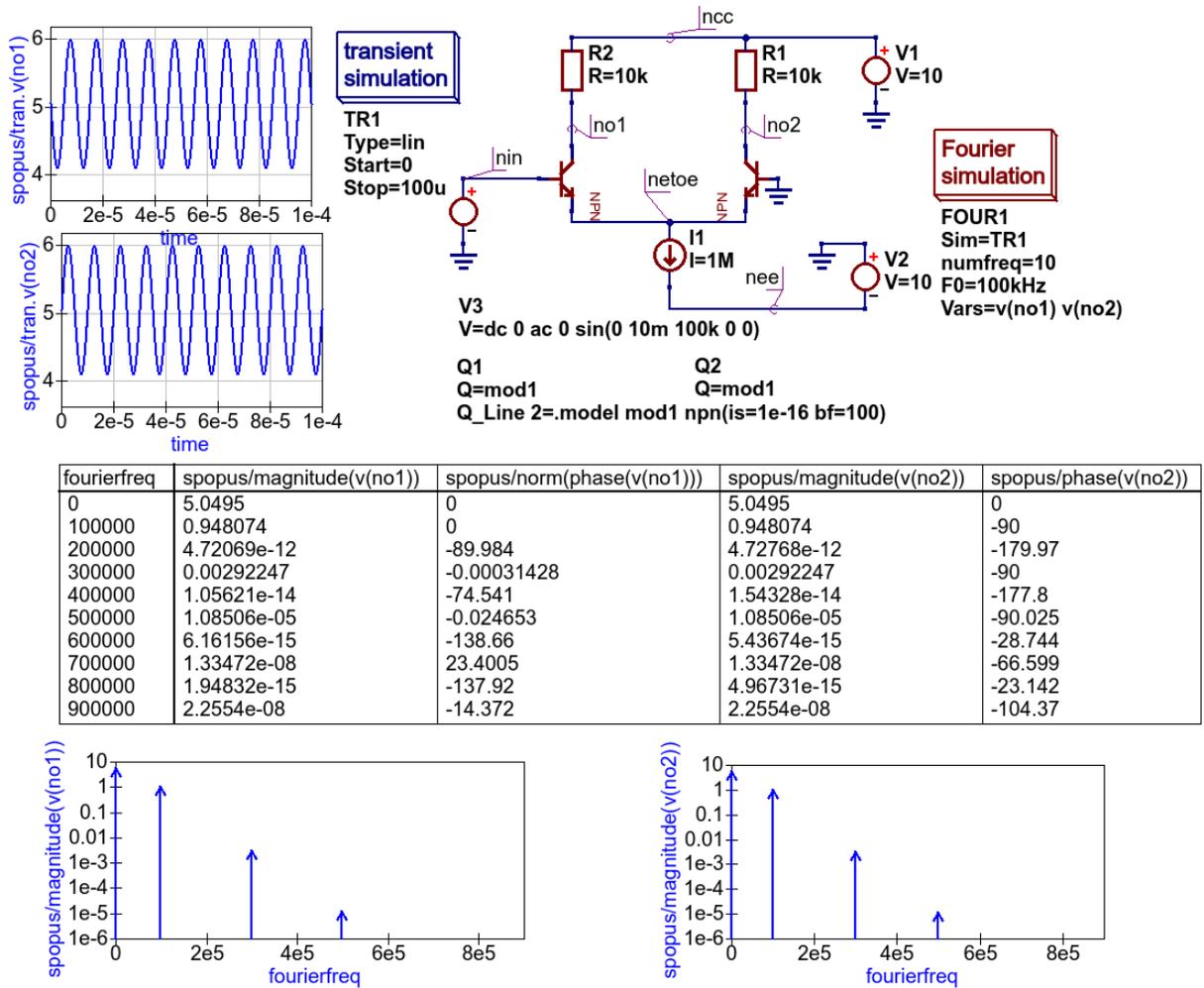
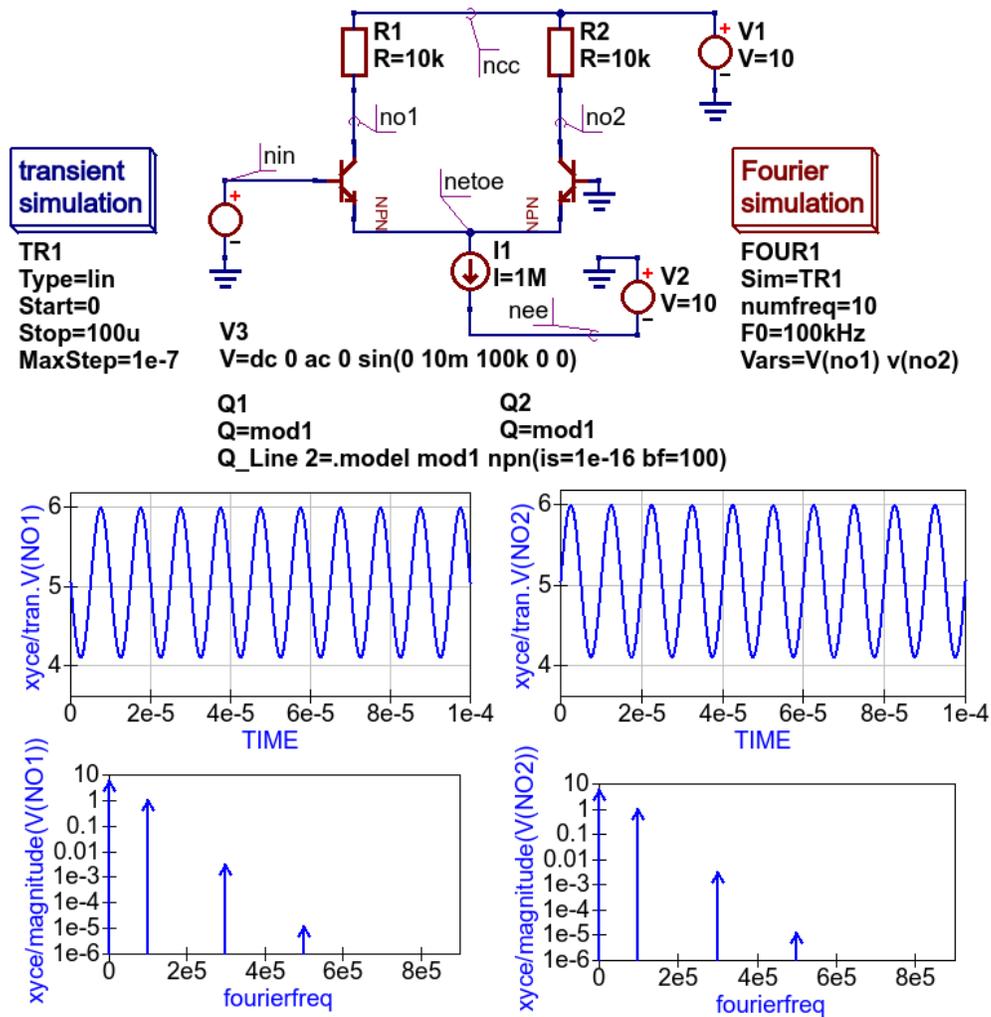


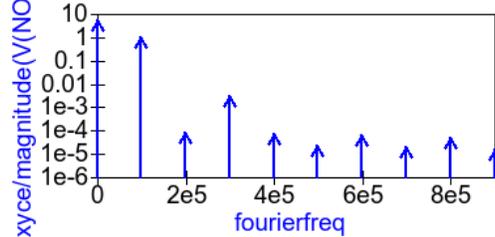
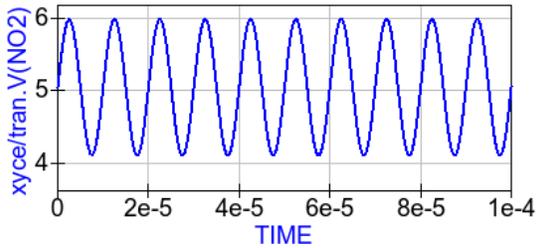
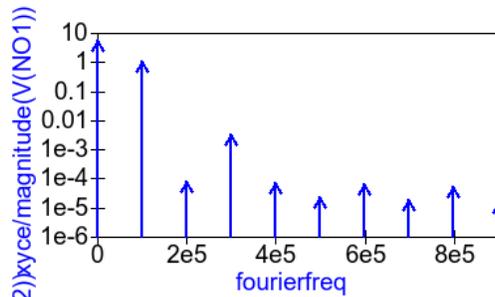
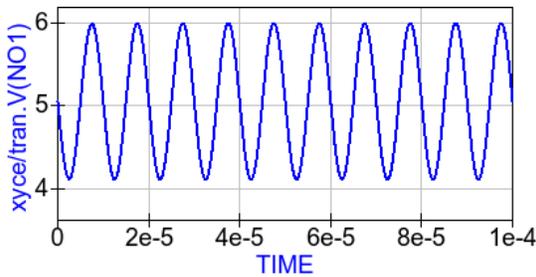
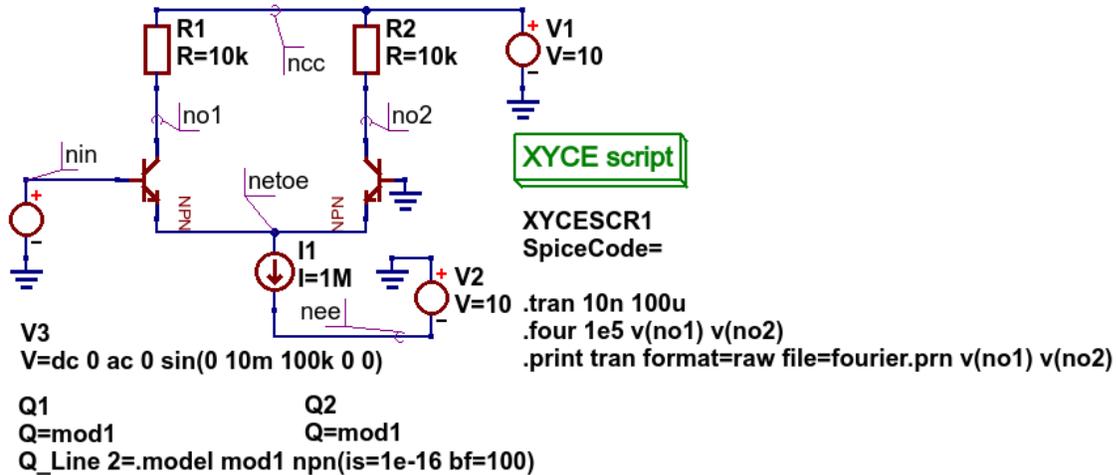
Figure 5.1.1 SPICE OPUS *Fourier simulation* basic example.

nation of the Ngnutmeg *fouriermn* statement.



fourierfreq	xyce/magnitude(V(NO1))	xyce/phase(V(NO2))	xyce/magnitude(V(NO2))	xyce/phase(V(NO2))
0	5.0495	0	5.0495	0
100000	0.948186	2.82399e-06	0.948186	2.82399e-06
200000	4.60627e-08	91.5242	4.60627e-08	91.5242
300000	0.00292558	0.000846228	0.00292558	0.000846228
400000	4.625e-08	93.037	4.625e-08	93.037
500000	1.08456e-05	0.244106	1.08456e-05	0.244106
600000	4.6556e-08	94.5279	4.6556e-08	94.5279
700000	5.91129e-08	51.9625	5.91129e-08	51.9625
800000	4.69876e-08	95.985	4.69876e-08	95.985
900000	4.72339e-08	96.513	4.72339e-08	96.513

Figure 5.1.3 Xyce *Fourier simulation* basic example: simulation controlled by *transient simulation* and *Fourier simulation* Icons.



fourierfreq	xyce/magnitude(V(NO1))	xyce/phase(V(NO2))	xyce/magnitude(V(NO2))	xyce/phase(V(NO2))
0	5.04947	0	5.04954	0
100000	0.948183	0.00110035	0.948183	0.00110035
200000	7.12974e-05	63.4429	7.12974e-05	63.4429
300000	0.00292211	0.377471	0.00292211	0.377471
400000	6.697e-05	38.2772	6.697e-05	38.2772
500000	2.16806e-05	64.3461	2.16806e-05	64.3461
600000	5.875e-05	15.3975	5.875e-05	15.3975
700000	1.72782e-05	78.6788	1.72782e-05	78.6788
800000	4.67963e-05	-4.69127	4.67963e-05	-4.69127
900000	1.39432e-05	50.6946	1.39432e-05	50.6946

Figure 5.1.4 Xyce *Fourier simulation*: controlled by *Xyce script*; see section 2.1.6 of the Xyce Reference Guide for an explanation of the *.four* and associated *.print* statement.

5.2 Distortion simulation

SPICE *Distortion analysis* provides a small signal distortion analysis of a circuit being simulated. To request a distortion analysis place a copy of the special *Distortion analysis* icon on the current work schematic. It undertakes a simulation similar to the small signal *AC analysis*, but calculates the circuit distortion components instead. SPICE

Distortion analysis is only available with ngspice. The calculated distortion components, for example values for the second and third harmonic components, can be extracted using ngnutmeg script statements. Refer to the official ngspice manual for the details of the available ngnutmeg operators and functions.

Ngspice *Distortion analysis* requires that the circuit being simulated is driven by a special *AC voltage* source component. This extended signal source can be found in the *Spice components* group. You need to specify voltage source parameters *DISTOF1* and/or *DISTOF2* for the ngspice *Distortion analysis* to function correctly. Refer to the official ngspice manual for a detailed description of all the ngspice *Distortion analysis* features. Please NOTE that the standard Qucs AC source will not work with ngspice *Distortion analysis*.

Here is an basic example of the application of SPICE *Distortion analysis* for estimating the distortion components of a single stage transistor amplifier.

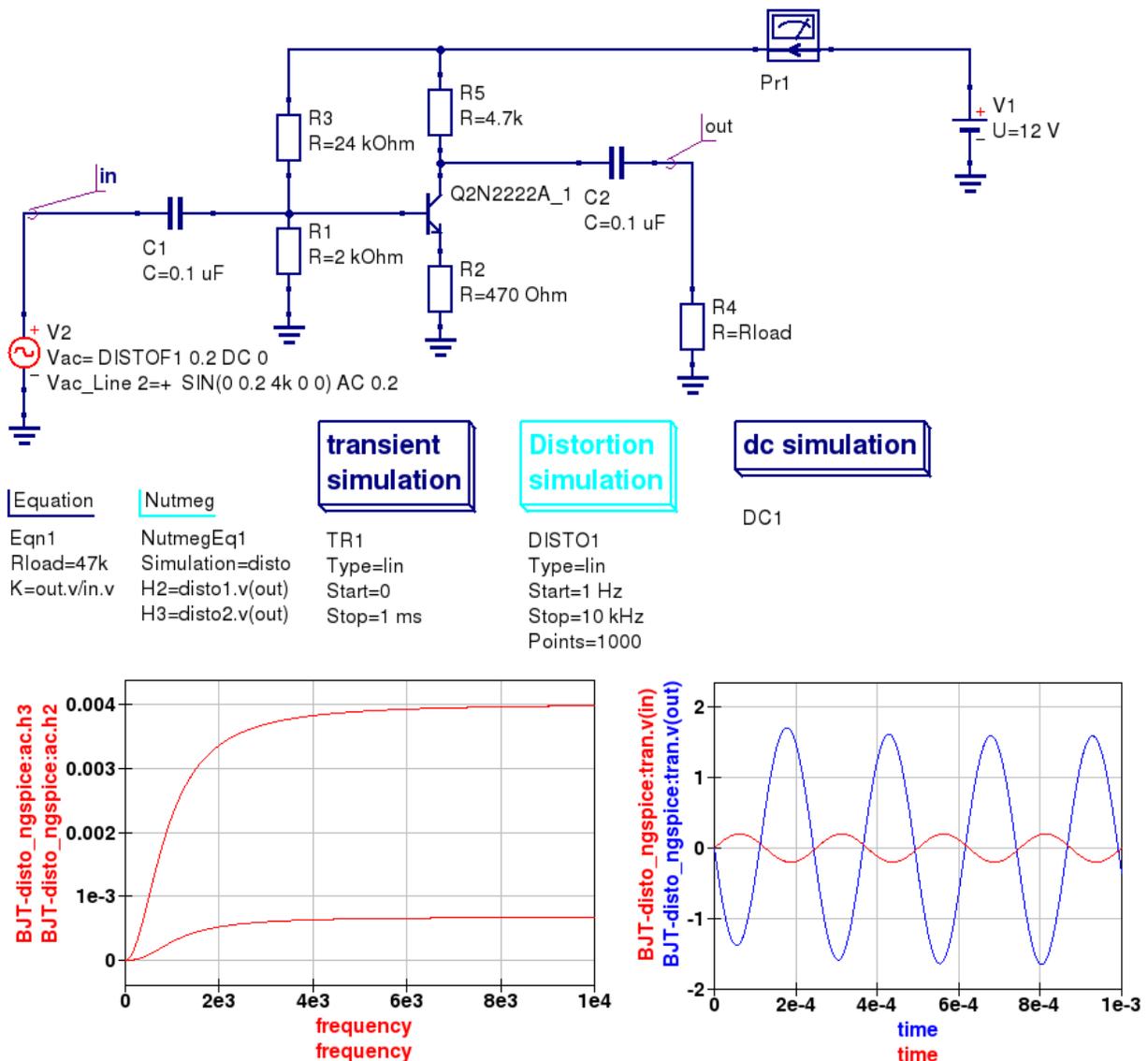


Figure 5.2 SPICE small signal distortion analysis of a single stage transistor amplifier.

5.3 Noise simulation

SPICE noise simulation allows the calculation of total circuit noise over a specified frequency bandwidth. ngspice noise simulation creates two vectors:

1. `onoise_total` — Integrated output noise.
2. `inoise_total` — Equivalent input noise.

Spice4qucs allows these vectors to be plotted. NOTE the Noise simulation at a single signal frequency only outputs a single set of noise data. At this time only ngspice has noise analysis implemented. However, in the near future is expected that noise simulation will be added to Xyce.

To set up a *Noise analysis* add the following four parameters to the *Noise analysis* icon drop-down list:

1. `Bandwidth limits` — Analysis start and stop frequencies in Hz.
2. `Points count` — The number of noise simulation frequency points.
3. `Output` — The output parameter name; this may be a node voltage or branch current.
4. `Source` — Name of the input voltage source. A standard Qucs voltage source is allowed in this context.

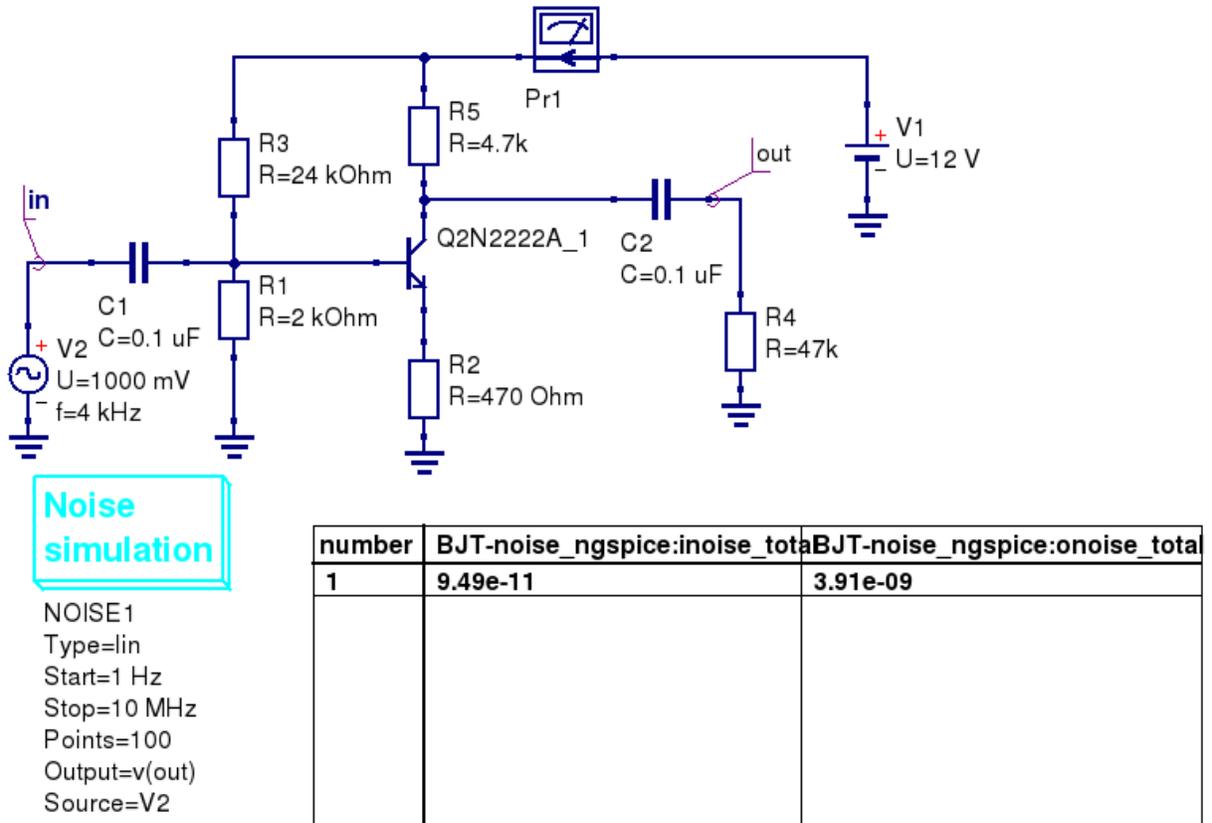


Figure 5.3 Noise analysis of a single stage transistor amplifier.

5.4 One and two parameter sweep controlled simulations

Both one and two **Parameter sweep** simulations (in nested loops) are implemented with Ngspice, SPICE OPUS and Xyce. However, there is no warranty that proper results will be obtained with the Xyce time domain simulation linked

to **Parameter sweep** changes, mainly because Xyce uses an adaptive time step for each step of a sweep variable. **Parameter sweep** simulations operating with DC and frequency domain circuit simulation do not suffer from this problem and normally report accurate output data.

The differences between the Qucs-S and Qucs **Parameter sweep** definitions are listed below

1. Qucs-S uses a component name instead of a variable name to set a sweep component instance value; for example use C1, R1, etc. to sweep capacitance and resistance values of components C1 and R1.
2. Ngspice and Xyce allow model parameter values to be swept using the following notation: Ngspice uses @dev[param] and Xyce uses dev:param. This notation is selected by setting the **Parameter sweep** variable *SweepModel* to true. Note also that the Ngspice nutmeg command *altermod* can also be used to change the value of a component or model parameter value. Qucs legacy devices use notation Component_name.Parameter_name. This notation is selected by setting the **Parameter sweep** variable *SweepModel* to false. Table 5.1 shows the allowed combinations of *SweepModel* and parameter values. All other combinations are illegal and will give incorrect output data or cause Qucs-S to crash and should no be used.
3. Qucs-S does not allow the use of .PARAM and .GLOBAL_PARAM names as sweep variables.

Table 5.1 Allowed combinations of Component/Model identifiers and *SweepModel* access codes

Simulator	<i>SweepModel</i>	Component access	Model access
Qucsator	FALSE	Value	
	FALSE		Device.parameter_value
Ngspice	FALSE	Name	
	TRUE		@Device_name[parameter_name]
Xyce	FALSE	Name	
	TRUE		Device_name:parameter_name

Figure 5.4 shows how changing the values of collector resistance effects the mid-band gain of a single stage BJT amplifier. Theoretically, the ideal gain is given by $R2/R4$, suggesting good agreement between the simulated output data and theory. The schematic illustrated in Figure 5.4 also presents a technique for scanning a component value in different simulation domains. In this example the same component value (R2) is changed by a **Parameter sweep** icon linked to individual simulation icons (SW3+TR1 and SW2+AC1).

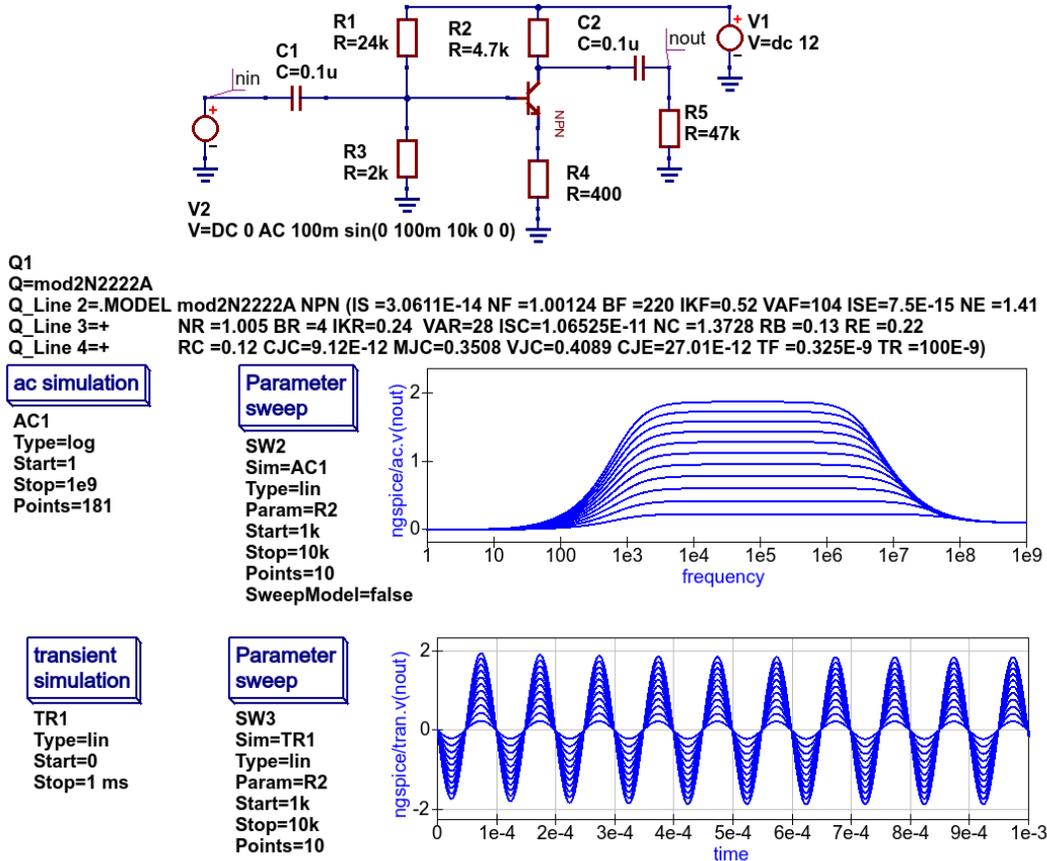


Figure 5.4 Ngspice component sweep example.

The example shown in Figure 5.5 demonstrates the effect of changing capacitor C1 on the low frequency response of the single stage BJT amplifier introduced in Figure 5.4. Figures 5.6 and 5.7 introduce further extensions of the Qucs-S swept parameter capabilities. Notice that Xyce allows semiconductor, and indeed other component models with parameters specified by the .MODEL statement, to be swept in DC simulations. However, this is not the case with Ngspice and SPICE OPUS DC simulations where only independent voltage and current source values and resistor values can be swept. This limitation follows directly from the original SPICE 3f5 simulator C code. In contrast to Ngspice and SPICE OPUS, Xyce includes a .STEP statement which supports an extended range of swept component parameter features, making it similar to the original Qucs swept parameter simulation.

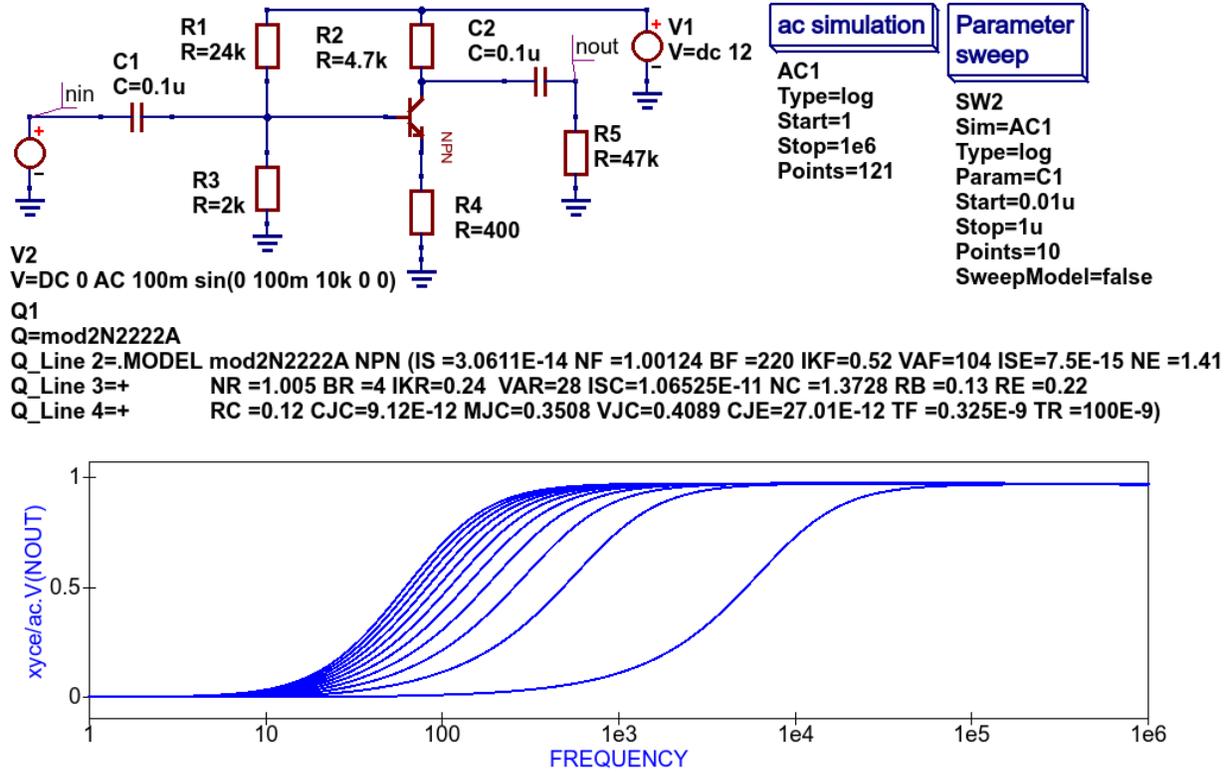


Figure 5.5 Xyce component sweep example two.

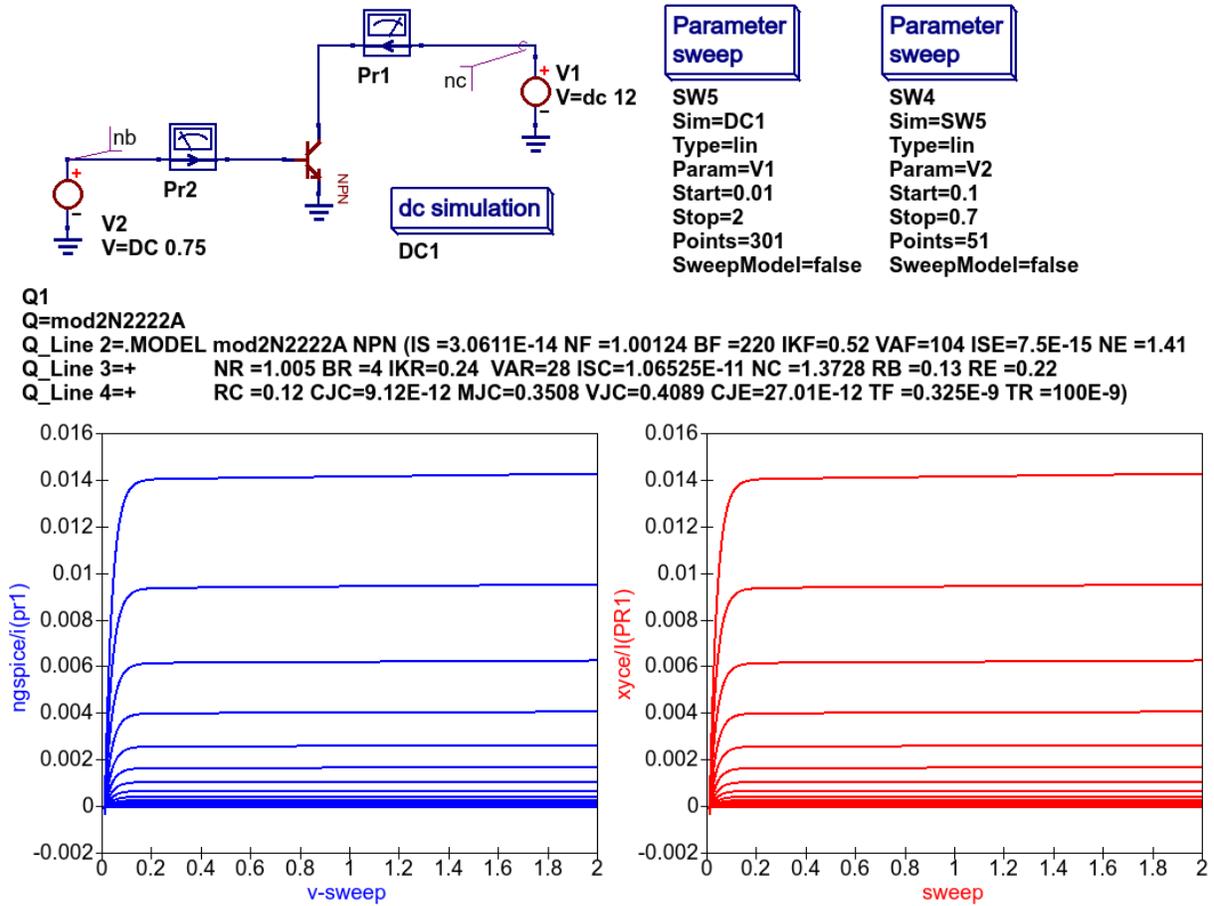


Figure 5.6 Two variable nested loop parameter scan: Ngspice and Xyce BJT output characteristics.

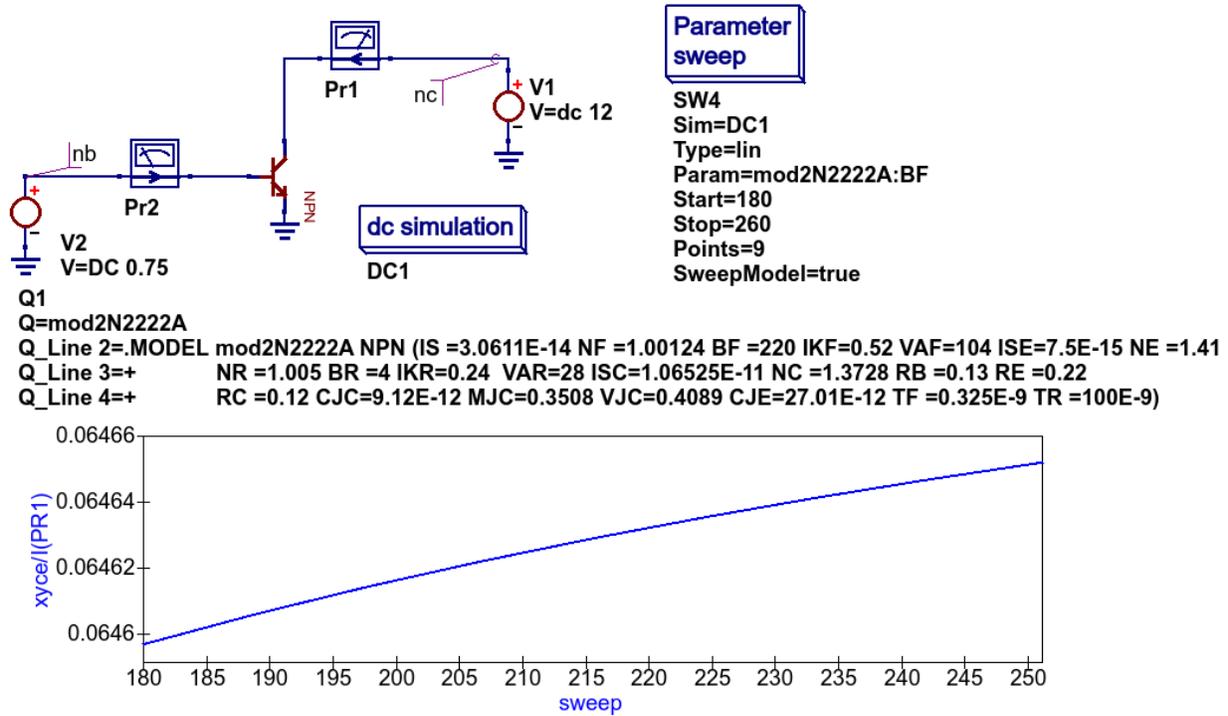


Figure 5.7 Effects of BJT BF parameter scan on DC collector current: Xyce simulation.

5.5 Qucs and SPICE simulation of device and circuit temperature properties

5.6 Spectrum analysis with Ngspice and Nutmeg scripting

Qucs-S have no unified simulation type “**Spectrum analysis**” for all simulation backends. But you may use Nutmeg scripting to implement Spectrum analysis if Ngspice or SpiceOpus is selected as the default simulation kernel.

Let’s consider double balanced passive diode mixer circuit.

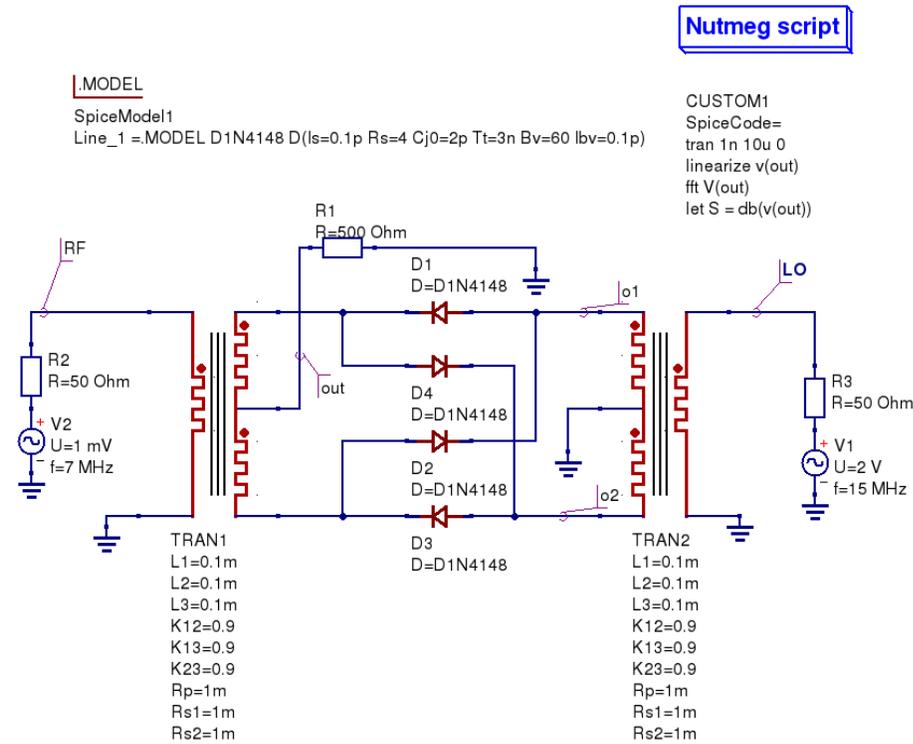


Figure 5.8 Diode double balanced mixer simulation

Balanced mixer circuit has two inputs: local oscillator $f_{LO} = 15\text{MHz}$ (LO node) and RF signal $f_{RF} = 7\text{MHz}$ (RF node on schematic) and gives a set of signals at the outputs. Transformer models are taken from the **Transformer** library from the Qucs-S distribution. Output signal is taken from the `out` node. It contains components with the following frequencies:

$$f_{out} = \pm m f_{RF} \pm n f_{LO} \quad \text{where } m, n \neq 0$$

The following two components are the strongest (upper IF and lower IF respectively):

$$f_{IF2} = f_{LO} + f_{RF}$$

$$f_{IF1} = f_{LO} - f_{RF}$$

We should see these signals as peaks at the spectrum plot.

We want to obtain mixer output voltage plot $V(\text{out})$. It's need to use Nutmeg scripting to obtain the spectrum. **Nutmeg script** component serves for this purpose at the presented circuit. Let's consider Nutmeg script structure. Such structure is need to be used for every spectrum analysis. Nutmeg script source code is presented here:

```

1 tran 1n 10u 0
2 linearize v(out)
3 fft V(out)
4 let S = db(v(out))
    
```

Spectrum calculation is performed by the `fft()` operator at the line #3. The argument of this function is transient simulation result vector (voltage or current). And it's need to perform a transient simulation before. Transient simulation is performed at the line #1. Simulation step is $t_s = 1\text{ns}$ and duration is $T_d = 10\mu\text{s}$. This gives

$$N = \frac{T_d}{2t_s} = \frac{10\mu\text{s}}{2 \cdot 1\text{ns}} = 5000$$

spectrum points.

Frequency step will be:

$$F = \frac{1}{2Nt_s} = 100\text{kHz}$$

We can summarize that the smallest timestep and the longest duration gives the most precise frequency step and spectrum analysis precision. But it increases the simulation time.

Ngspice uses dynamic timestep calculation at simulation time. And real timestep may differ from the specified in the `tran` statement. It's need to perform simulation analysis linearization. Line#2 linearizes simulation result (output voltage `V(out)`). Vector `V(out)` contains now linearized transient simulation result and could be passed to the `fft()` input (line #3).

After FFT we can plot `V(out)` vector and see spectrum. But we can apply any postprocessing to it. For example we can express spectrum in decibels (dB) with `dB()` nutmeg function (line #4, `S` variable). You need to specify these two variables in the Nutmeg script properties (Figure 5.9)

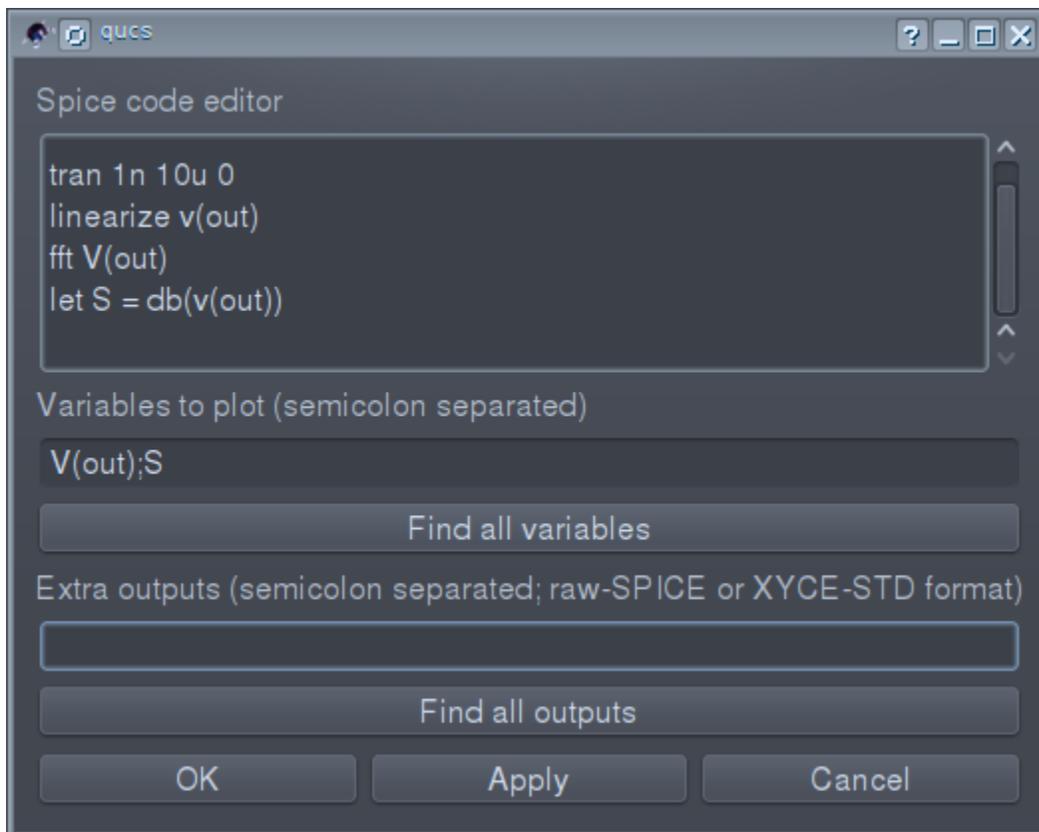


Figure 5.9 Nutmeg script properties setup

Simulation results are shown in the Figure 5.10. Both spectrum and logarithmic spectrum (dB) are shown.

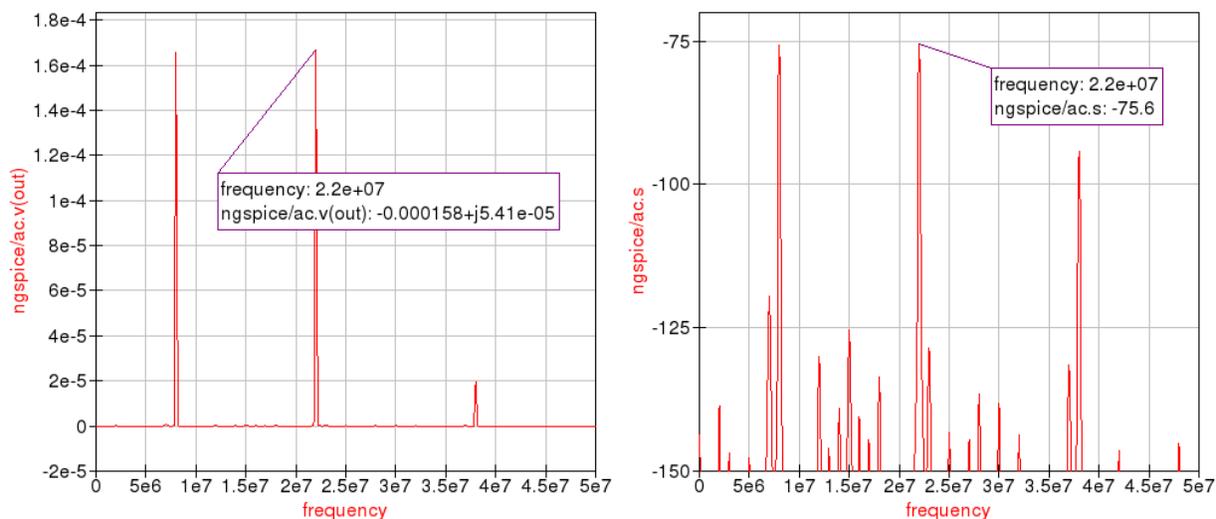


Figure 5.10 Spectrum simulation result.

We can see two main peak on spectrum ($f_{IF1} = 22\text{MHz}$ and $f_{IF2} = 8\text{MHz}$ respectively). RF and LO signals are rejected.

back to the top

Chapter 6. Ngspice, Xyce and SPICE OPUS post-simulation data processing with Qucs-S and Octave

6.1 Introduction to capabilities

In this chapter the Ngspice, Xyce and SPICE OPUS post-simulation data processing and visualization capabilities are introduced and applied to a number of example simulation case studies. Qucs-S makes use of the existing Qucs post-simulation data processing facilities BUT modifies their operation to take into account the numerical and algebraic functions provided by Ngspice and SPICE OPUS nutmeg scripts and Xyce embedded equations in SPICE .PRINT statements.

The following list presents a resume of the data manipulation and plotting features implemented by Qucs and the spice4qucs additions central to Qucs-S.

- **Qucs** : Built in MATLAB style numerical data processing; GUI data visualization; Octave data processing and plotting
- **Ngspice** : Spice nutmeg script controlled numerical data processing; Extended Qucs GUI style data visualization; H SPICE style .measurement post-simulation data processing; Octave data processing and plotting
- **SPICE OPUS** : Spice nutmeg script controlled numerical data processing; Extended Qucs GUI style data visualization; Octave data processing and plotting
- **XYCE** : Extended SPICE .PRINT statements with “algebraic and numeric” non-linear equations for numeric data processing; H SPICE style .measurement post-simulation data processing; Extended GUI style data visualization; Octave data processing and plotting

Qucs-S post-simulation data processing is linked to the use of Qucs **Equation blocks** and Qucs-S **Nutmeg Equation blocks**. To understand how Qucs-S deals with post-simulation data processing and visualization it is important that

readers become aware of a number of critical, highly significant, facts concerning Qucs **Equation blocks**. Figure 6.1 shows a very simple RC circuit who's performance is simulated with AC and transient simulation.

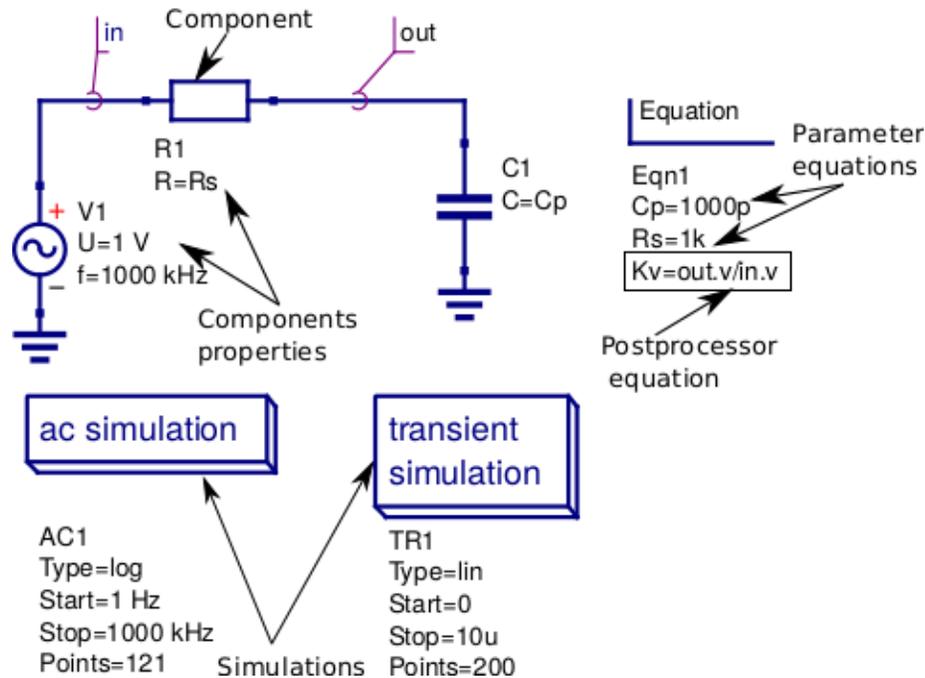


Figure 6.1 A basic RC test circuit with component values set by a Qucs **Equation block**: component properties, parameter equations and post processing equations are shown marked with arrows.

In Figure 6.1 **Equation block Eqn1** includes a mixture of variable assignments that are independent of simulation output data, component values C_p and R_s , and variable assignments that are functions of output data, variable K_v which is a function of $out.v$ and $in.v$. All **Equation block** variables that are NOT functions of output data are actioned before the start of a simulation and their values remain constant throughout simulation. In contrast **Equation block** variables that are functions of output data are actioned AFTER a simulation is finished. Notice that if there are more than one **Equation block** placed on a schematic they are joined together to form one larger **Equation block** before processing by Qucs. With Qucs the order of the left hand variables in an **Equation block** is not important because Qucs arranges the list into an order which gives the correct sequence during list processing. Readers need only remember that each named left hand side variable is allowed only one entry in the **Equation block** list. More than one entry with the same name flags an error message. Notice also that for all types of Qucs simulation the output data processed by an **Equation block** is named with identification letters after the “full stop .”, for example in Figure 6.1 there are both AC and transient icons BUT variable K_v is only defined for the SPICE AC simulation. Qucs **Equation block** variable entries are defined by right hand equations which are a mixture of numerical constants, named variables, functions, and mathematics operators defined in the “Qucs Help Index” documentation (see the section called “Short description of mathematical functions”). Please NOTE that all the functions in this list are only applicable to Qucs and ONLY a percentage are available with the Ngspice, Xyce and SPICE OPUS simulators. The next few sections of this document provide more detail on the Qucs functions that can be used with Ngspice, Xyce and SPICE OPUS.

6.2 Ngspice and SPICE OPUS output data post-processing

The Ngspice and SPICE OPUS circuit simulators also use **Nutmeg Equation blocks** for output data post-processing. Figure 6.2 illustrates how **Nutmeg Equation blocks** are applied to the data post-processing task. Unlike Qucs **Equation blocks** the Qucs-S **Nutmeg Equation blocks** are characterised by being linked to each different type of Qucs-S SPICE simulation, for example in Figure 6.2 there are two **Nutmeg Equation blocks** one for AC simulation and one for transient simulation. Qucs-S **Nutmeg Equation block** entries result in SPICE nutmeg *let* statements being placed between the relevant SPICE *.control* and *.endc* statements in a synthesised SPICE netlist generated by Qucs-S prior

to simulation by Ngspice or SPICE OPUS. Notice that one *let* statement is generated per **Nutmeg Equation block** entry and that the order of the variables is important because ALL named variables must be defined before they are used in subsequent variable assignments. Opposite to Qucs these variables are NOT arranged by Qucs-S in an order that ensures all left hand variables can be evaluated correctly prior to use in other statements during post-simulation data processing. In most cases these entries will represent some form of post-simulation output data processing action, where the right hand equation entry can be a function of numeric constants, previously defined variables, device parameters, Ngspice or SPICE OPUS nutmeg operators and functions and data output item names. The latter need to be expressed in SPICE format rather than the standard Qucs format described previously. Node voltages are selected using the SPICE notation $V(n)$ or $V(n1,n2)$, where voltage $V(n)$ is referenced to ground and $V(n1,n2)$ indicates the voltage difference between nodes $n1$ and $n2$. Currents flowing in a circuit are recorded through the use of a zero value independent voltage source, via the SPICE notation $Vxxx\#branch$ (see Figure 6.2), or by placing a Qucs current probe in the circuit being simulated and recording its value using $VPrxxx\#branch$ (see following examples). Also notice that in the version of the RC test circuit introduced in Figure 6.2 the component values C_p and R_s are no longer set by a Qucs **Equation block** but are allocated numerical values at component symbol level.

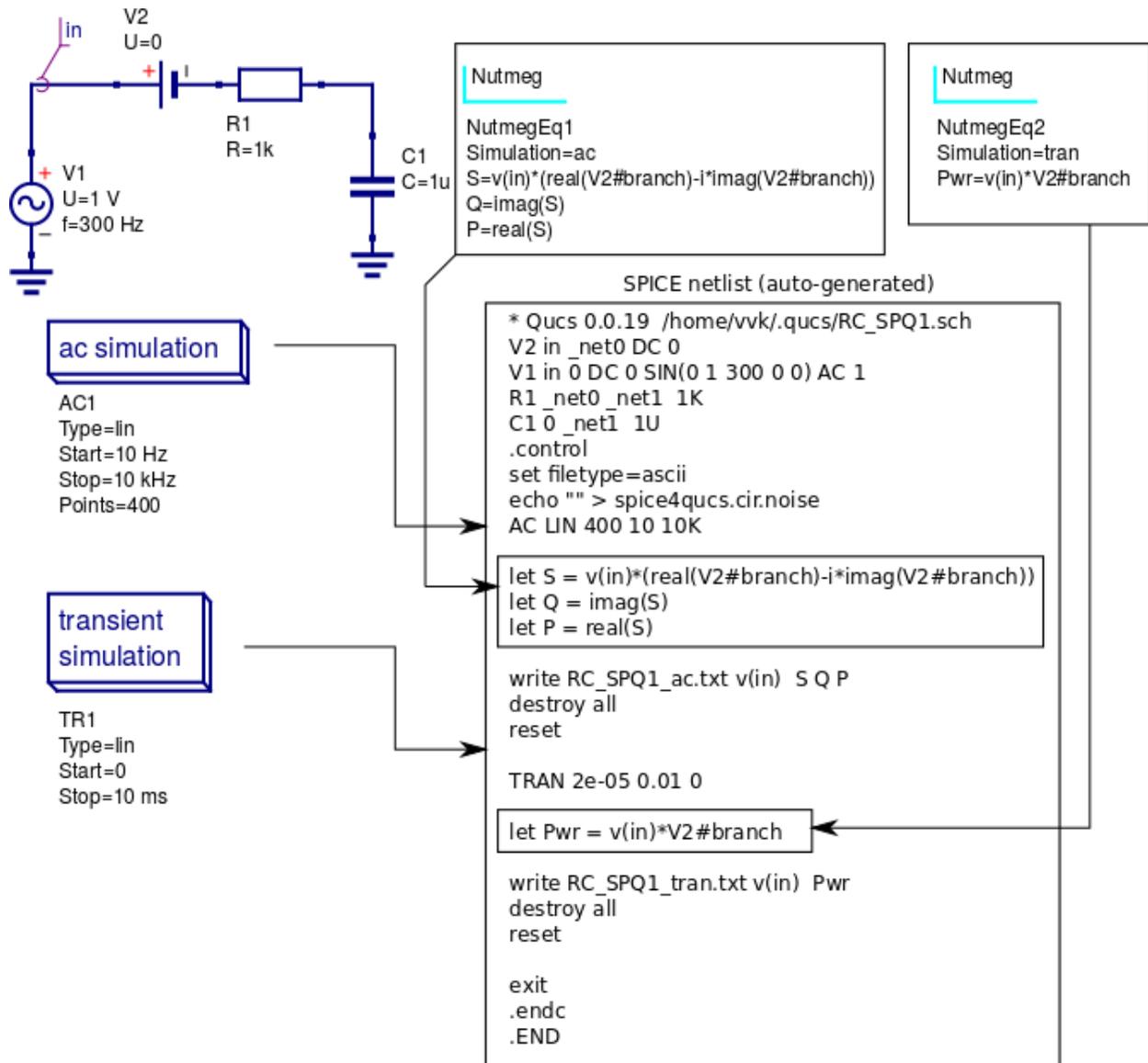


Figure 6.2 The basic RC test circuit introduced in Figure 6.1 with post-simulation controlled by Qucs-S **Nutmeg Equation blocks** NutmegEq1 and NutmegEq2.

6.3 Ngspice, SPICE OPUS and XYCE data post-processing operators and functions

Ngspice and SPICE OPUS both use extended versions of the SPICE 3f5 nutmeg software for manipulating and visualizing simulation output data. Xyce does NOT include a version of SPICE nutmeg BUT employs an extension of the SPICE .PRINT statement to generate tables of output values for post simulation processing. At a first reading of the Xyce manuals the lack of nutmeg would appear to be a serious omission. However, by adding equations composed of numerical values, output variables, mathematical operators and “Analogue behavioural modelling” functions as arguments to SPICE .PRINT statements it becomes possible to manipulate output data in a fashion similar to SPICE nutmeg. The Xyce extended form of .PRINT statement allows “Analogue behavioural modelling (ABM)” equations embedded in { }. Such equations ONLY operate on real quantities and hence some restrictions apply to AC and HB simulation, see later notes.

The following list presents a summary of the operators and functions implemented by the Ngspice, SPICE OPUS and Xyce circuit simulators. These can be used for simulation output data manipulation using Ngspice and SPICE OPUS nutmeg scripts and Xyce .PRINT netlist statements. A more detailed explanation of their function can be found in the individual simulator manuals listed in the reference section at the end of spice4qucs-help document.

- **Ngspice**

- **Operators:**

```
+ -, +, *, /, ^, %, ,
+ gt, lt, ge, le, ne, and, or, not, eq
+ >, <, >=, <=, <>, &, |, !, =
```

- **Functions:**

```
+ mag(), ph(), cph(), unwrap(), j(), real(), imag(), db(), log(), ln(),
+ exp(), abs(), sqrt(), sin(), cos(), tan(), atan(), sinh(), cosh(), tanh(),
+ floor(), ceil(), name(), mean(), arg(), group-delay(), vector(), initvec(),
+ length(), interpolate(), deriv(), vecd(), vecmin(), minimum(),
+ vecmax(), maximum(), fft(), ifft(), sortorder(), rnd,
+ sgauss(), sunif(), poisson(), exponential()
```

- **Constants:**

```
+ pi, e, c, i, kelvin, echarge, boltz, planck, yes, no, TRUE, FALSE
```

- **SPICE OPUS**

- **Operators:**

```
+ -, +, *, /, ^, %, ,
+ gt, lt, ge, le, ne, and, or, not, eq. ;, [], [%]
```

- **Functions:**

```
+ abs(), mag(), magnitude(), db(), ph(), phase(), unwrap(), real(), re(),
+ imag(), im(), j(), ln(), log(), log10(), exp(), sqrt(), sin(),
+ cos(), tan(), atan(), floor(), ceil(), round(), length(), mean(), sum(),
+ min(), max(), vector(), unitvec(), rnd(), rndunif(), rndgauss(),
+ interpolate(), deriv(), integrate(), timer, clock(), area()
```

- **Constants:**

```
+ pi, e, c, i, kelvin, echarge, boltz, planck, yes, no, true, false
```

- **Xyce**

- Operators:

```
+ -, +, *, /, **
+ ==, !=, >, >=, <, <=
```

- Functions:

```
+ abs(), agauss(), gauss(), ddt(), ddx(), if(), int(), limit(), m(), min(),
↪max(),
+ pwr(), pow(), pwrs(), rand(), sgn(), stp(), sqrt(),
+ table(), uramp(), acos(), acosh(), arctan(), asin(), asinh(), atan(),
↪atanh(),
+ atan2(), cos(), cosh(), exp(), ln(), log(), log10(),
+ sin(), sinh(), tan(), tanh()
```

- Constants:

```
+ PI, EXP
```

Comparing the above lists with the Qucs list of post processing functions readers will observe that many of the Qucs RF data manipulation functions and electronic data analysis and plotting functions, like for example function `PlotVs()`, are NOT supported by Ngspice, SPICE OPUS and Xyce. Normally, this is not a particular problem because other means for generating these missing functions have been implemented by Xyce and Qucs-S Development Teams. This topic is expanded further in sections 6.5 and 6.6. One additional point to note concerning the above list is that all the operator, function and constant names are reserved words and must NOT be used for other purposes, like for example, naming circuit nodes. If they are used out of context the SPICE engines often fail when passing circuit netlists. In the case of misuse in “naming circuit nodes” Qucs-S will report an error and stop passing a circuit netlist.

Figures 6.3 and 6.4 introduce a single stage BJT common emitter amplifier circuit set up for AC small signal simulation with Ngspice. The **Nutmeg equation block** demonstrates the basic use of post simulation scripts for extracting circuit properties from output data. Results obtained with SPICE OPUS are identical to those shown in Figure 6.5 when Ngspice function `cph()` is replaced by SPICE OPUS function `phase()`. Please note only one **Nutmeg equation block** of each simulation type, for example `ac`, is allowed per schematic.

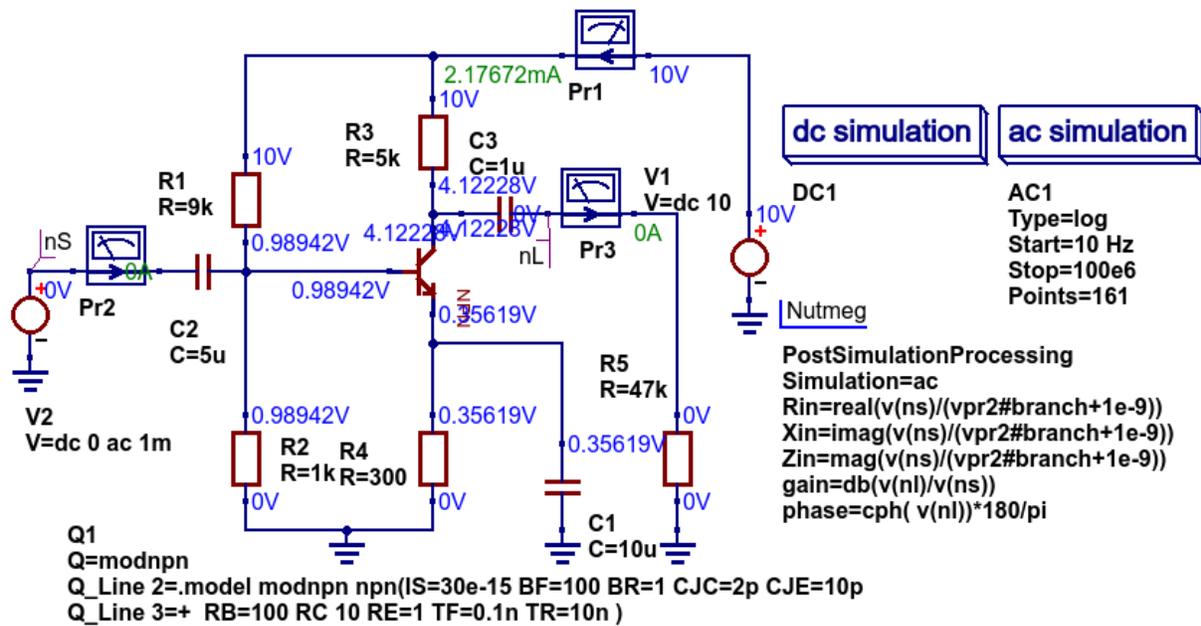


Figure 6.3 Test circuit for a single stage BJT common emitter amplifier and post simulation **Nutmeg equation** script.

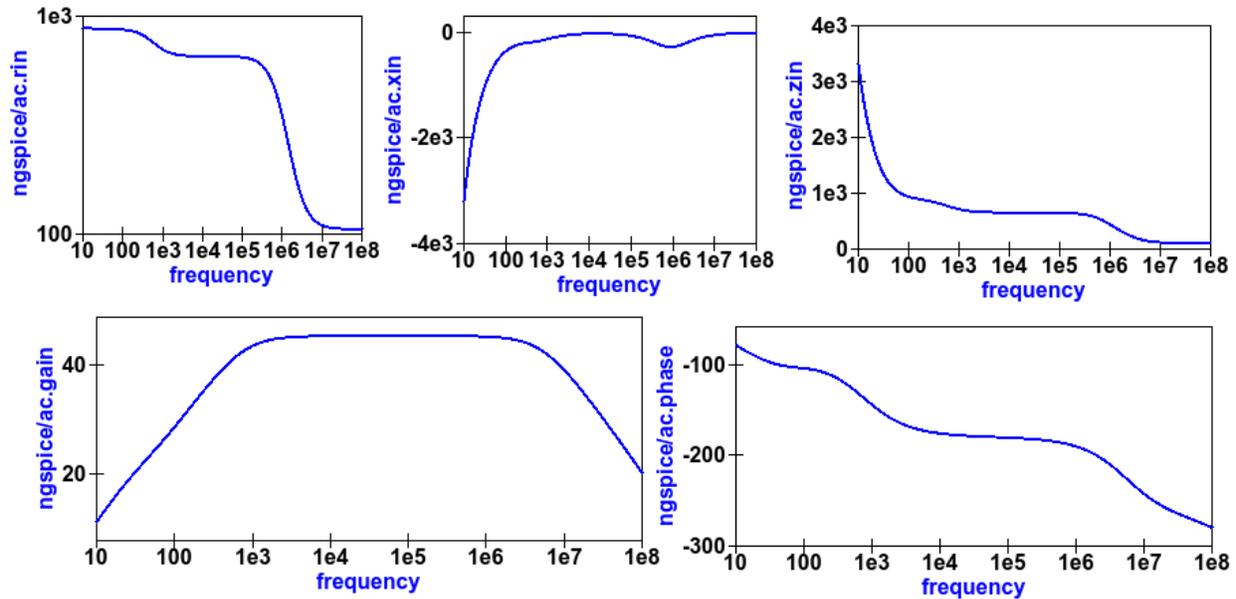
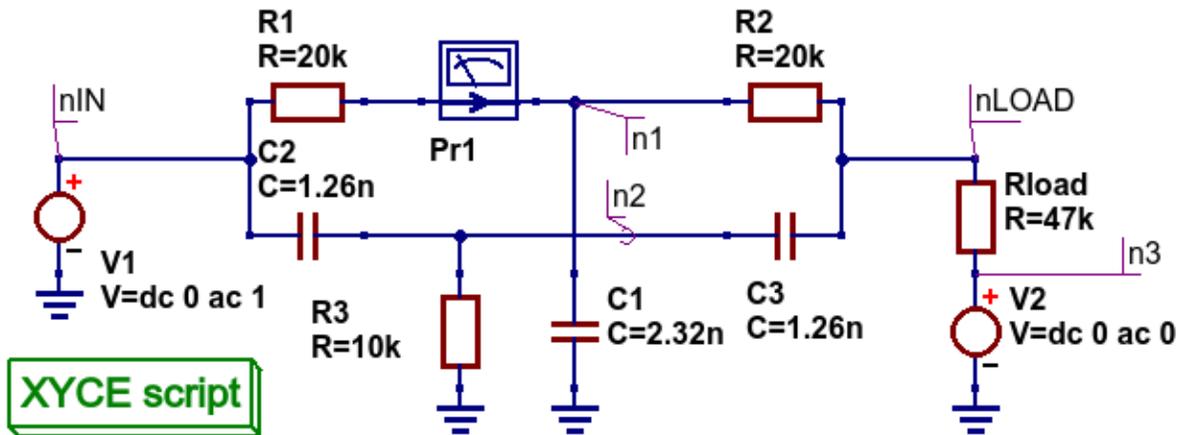


Figure 6.4 Plots of the single stage common emitter amplifier, R_{in} (in OHM), X_{in} (in Ohm), Z_{in} (in Ohm), voltage gain (in dB) and phase (in degrees) against frequency.

6.4 Xyce simulation output data post processing with the Xyce script component and SPICE .PRINT statements

Xyce simulation output data post processing uses an entirely different approach to that adopted by Ngspice and SPICE OPUS. Xyce is a circuit simulator developed from scratch some time after the release of SPICE 3f5. This route has allowed the Xyce Development Team to make software decisions which are not constrained except that the Xyce circuit simulator netlist should be compatible with the SPICE 3f5 netlist structure and statements. In contrast to both Ngspice and SPICE OPUS the Xyce .PRINT statement has been extended to allow additional types of output variables and embedded algebraic and numeric equations designated by a wrapping of brackets { }. In this section the extended form of the Xyce .PRINT statement is introduced, for each of the different simulation types, and its features and limitations explained. At the time of writing these notes Xyce AC node voltage data simulation data can only be manipulated using the real and imaginary components of individual variables and the basic mathematical operators (+, -, * and /), see the Xyce documentation for further details. However, embedded AC and HB equations are allowed provided they ONLY apply to variables represented by real data, for example the magnitude or phase of a node voltage. Xyce .PRINT statements use ABM to evaluate embedded data processing equations. However, the ABM package does not use complex numbers but only returns real numbers when calculating algebraic expressions. Xyce .PRINT statements are entered in a SPICE netlist between the netlist title on the first line and the last line .END entry. To generate a Xyce SPICE netlist, from a Qucs-S circuit schematic, which allows users to add simulation commands (AC, tran etc), .PRINT statements, and any other valid SPICE statement requires the addition of a specific control icon to Qucs-S. This component icon is called an **Xyce script**. Figure 6.5 shows a basic example of its use to set up and simulate the AC performance of a twin-tee notch filter circuit.



```

XYCESCR1
SpiceCode=
.ac dec 500 100 1e5
.print ac format=raw file=ac.txt v(nLOAD) vm(nLOAD) vp(nLOAD)
+      {vp(nLOAD)*180/pi} im(v1) im(v2) ip(v1) ip(v2)
+      im(vpr1) ip(vpr1)

```

Figure 6.5 Xyce AC simulation of a twin-tee notch filter controlled by a **Xyce script**.

Figures 6.6 and 6.7 give the Xyce SPICE netlist and plotted waveforms requested by the .PRINT statement shown in Figure 6.5.

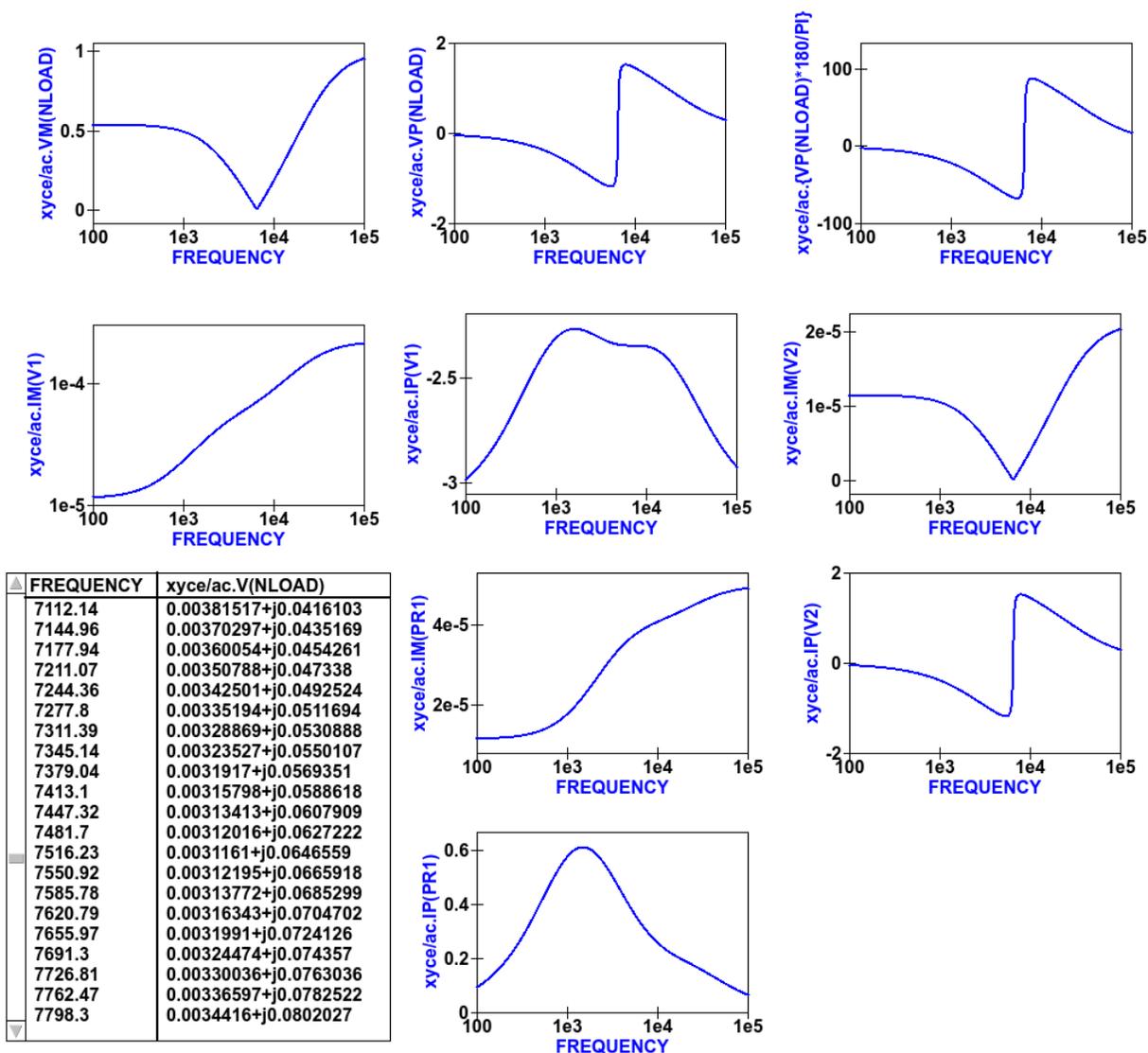
```

* Qucs 0.0.19 TwinTee.sch
C2 nIN n2 1.26n
C3 n2 nLOAD 1.26n
C1 n1 0 2.32n
R3 0 n2 10k
Rload n3 nLOAD 47k
V1 nIN 0 dc 0 ac 1
R1 _net0 nIN 20k
R2 nLOAD n1 20k
V2 n3 0 dc 0 ac 0
VPr1 _net0 n1 DC 0

.ac dec 500 100 1e5
.print ac format=raw file=ac.txt v(nLOAD) vm(nLOAD) vp(nLOAD)
+      {vp(nLOAD)*180/pi} im(v1) im(v2) ip(v1) ip(v2)
+      im(vpr1) ip(vpr1)
.END

```

Figure 6.6 Xyce twin-tee SPICE netlist generated by Qucs-S.



Figures 6.7 Qucs-S plotted waveforms for variables listed in the twin-tee .PRINT statement.

The Qucs-S processing of the AC version of the Xyce .PRINT statement allows the following types of output variable to be either tabulated (when complex numbers), or manipulated and plotted (when real numbers):

v(nx) or v(n1,n2)

- Node voltage with respect to ground or node voltage difference; complex number, tabulated by Qucs-S,

vr(nx) or vr(n1,n2)

- Node voltage real component with respect to ground or node voltage difference real part; real number, plotted by Qucs-S,

vi(nx) or vi(n1,n2)

- Node voltage imaginary component with respect to ground or node voltage difference imaginary part; real number, plotted by Qucs-S,

Vm(nx) or vm(n1,n2)

- Magnitude of a node voltage with respect to ground or magnitude of node voltage differences; real number, plotted by Qucs-S,

vp(nx) or vp(n1,n2)

- Phase of a node voltage with respect to ground or phase of node voltage differences; real number in radians, plotted by Qucs-S,

vdb(nx) or vdb(n1,n2)

- Magnitude of a node voltage with respect to ground or magnitude of node voltage differences; real number in dB, plotted by Qucs-S,

im(vx)

- Magnitude of current flowing in voltage source vx (it may be an independent voltage source or Qucs-S current probe); real number, plotted by Qucs-S,

ip(vx)

- Phase of current flowing in voltage source vx (it may be an independent voltage source or Qucs-S current probe); real number in radians, plotted by Qucs-S,

idb(vx)

- Magnitude of current flowing in voltage source vx (it may be an independent voltage source or Qucs-S current probe); real number in dB, plotted by Qucs-S.

Examples of these output data types are given in Figure 6.7. Figure 6.7 also shows readers how Xyce ABM equations can be used to convert phase data from radians to degrees. When using Xyce equations in .PRINT statements it is important to remember that ABM mathematical operators and functions **ONLY** work correctly with real numbers.

Post processing of Xyce HB simulation data is similar to AC data post processing in that the information outline above also applies to Xyce HB data. Figure 6.8 presents a typical HB simulation example. In this figure a single stage BJT amplifier, with feedback via an RC network, is driven by an AC signal of 50mV peak and 100kHz frequency. The HB simulation output data to be stored in an output file, hb.txt in Figure 6.8, is set by the .PRINT statement entered as part of the **Xyce script** icon. Figure 6.8 gives a selection of the resulting HB output data plots. Notice these are all represented by a complex conjugate style of graph. More details of this format and other aspects of Xyce HB simulation can be found in Chapter 13 section 4. All Xyce HB .PRINT statement variables must be of the same format to those introduced in the earlier paragraphs referencing Xyce AC simulation. Although multiple **Xyce script** icons are allowed this can result in problems during the post processing of AC and HB simulation data due to uncertainties in determining which frequency scale applies to each type of simulation. Hence, it is suggested that Xyce AC and HB **Xyce script** controlled simulations are not requested at the same time. Similarly, multiple .PRINT statements attached to a single **Xyce script** icon can result in simulation failure. A better approach is to use a single .PRINT statement and multiple SPICE continuation lines, see Figure 6.8.

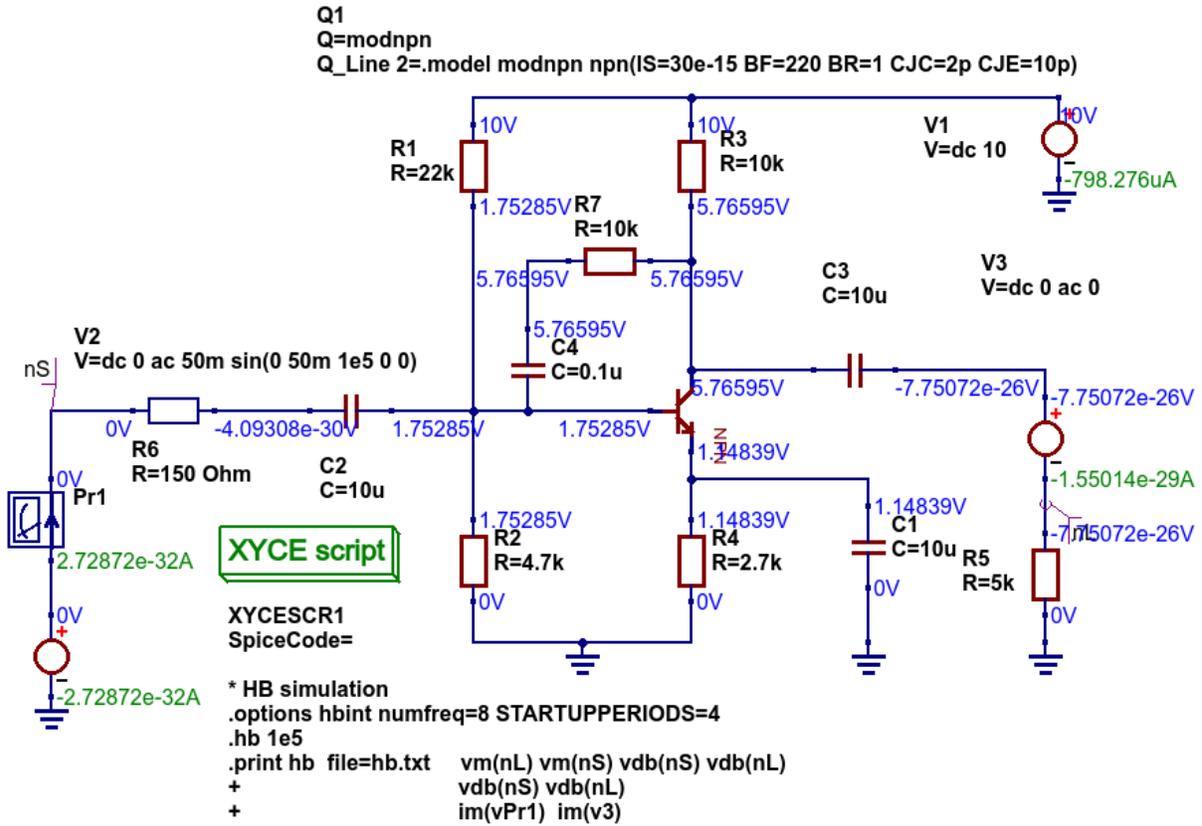


Figure 6.8 Xyce HB simulation of a single stage BJT amplifier with collector to base RC feedback network.

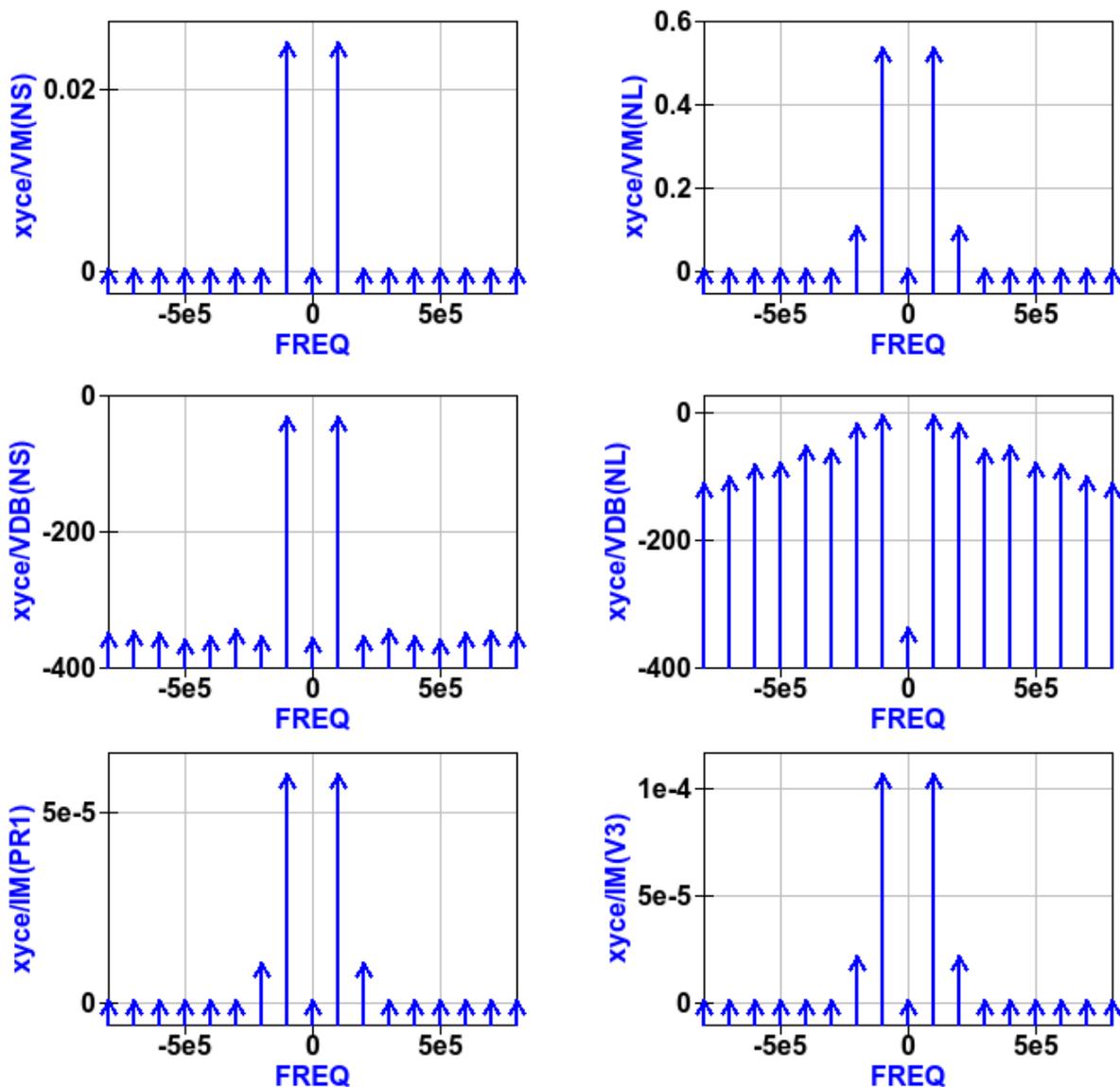


Figure 6.9 Plotted Xyce voltage and current output data for the BJT amplifier introduced in Figure 6.8.

In contrast to AC simulation the Xyce tran .PRINT statement allows the full range of built-in ABM mathematical functions to be employed when computing expressions that include node voltage and component current simulation data, see section 6.3. These functions only work correctly with real arguments; any variables represented by complex numbers with real and imaginary parts will cause an error. Bracketed, {...} expressions can be functions of constants, predefined variables, mathematical operators, implemented functions, node voltages, Qucs-S style probe currents, and the current flowing in SPICE style independent voltage sources. Xyce also allows B style non-linear dependent voltage and current sources to be used to compute transient simulation output data, like for example behavioural multiplication where the inputs are node voltages or component currents. Although this is a valid use of Xyce B sources the practice does have a number of disadvantages, namely that Xyce B sources do NOT work correctly with AC simulation, and secondly that the circuitry used to generate additional functions often adds nodes to the circuit under test, which as a consequence can slow down simulation. Hence, it is suggested that Xyce B sources should only be used when no other solution can be found.

The Qucs-S version of the Xyce transient .PRINT statement has the following syntax:

- `.PRINT tran format=raw file=tran.txt V(n1) {-----} V(d1) vpr1#branch`
.....

where `tran.txt` is the name of the output data file generated by a `.PRINT` statement, and

- $V(n_x)$, $V(n_1, n_2)$ are functions of circuit node voltages,
- $vprx\#branch$ or $vx\#branch$ are probe currents,
- `{-----}` represents an equation for computing an output quantity; Qucs-S identifies different quantities by their bracketed equation names at the top of the columns of data in file `tran.txt`,
- $I(\text{two-terminal device})$ where the two-terminal device can be one of V, I, B, E, G, H, D, R, L, C, and YMEMRESISTOR,
- $I_k(\text{three-or-more-termnal-device})$, see Xyce Reference Guide,
- $P(\text{two-terminal-device})$ or $W(\text{two-terminal-device})$ is the power dissipated in a two-terminal device,
- A full list of the allowed `tran.PRINT` output variables can be found in the Xyce User and Reference Guides.

The Xyce transient simulation shown in Figures 6.10 and 6.11 illustrate how the `.PRINT` statement syntax is used to store and plot circuit voltages, currents and equations. Notice that the test circuit in Figure 6.10 also demonstrates how the SPICE non-linear B style current source can be used to generate a function of circuit data.

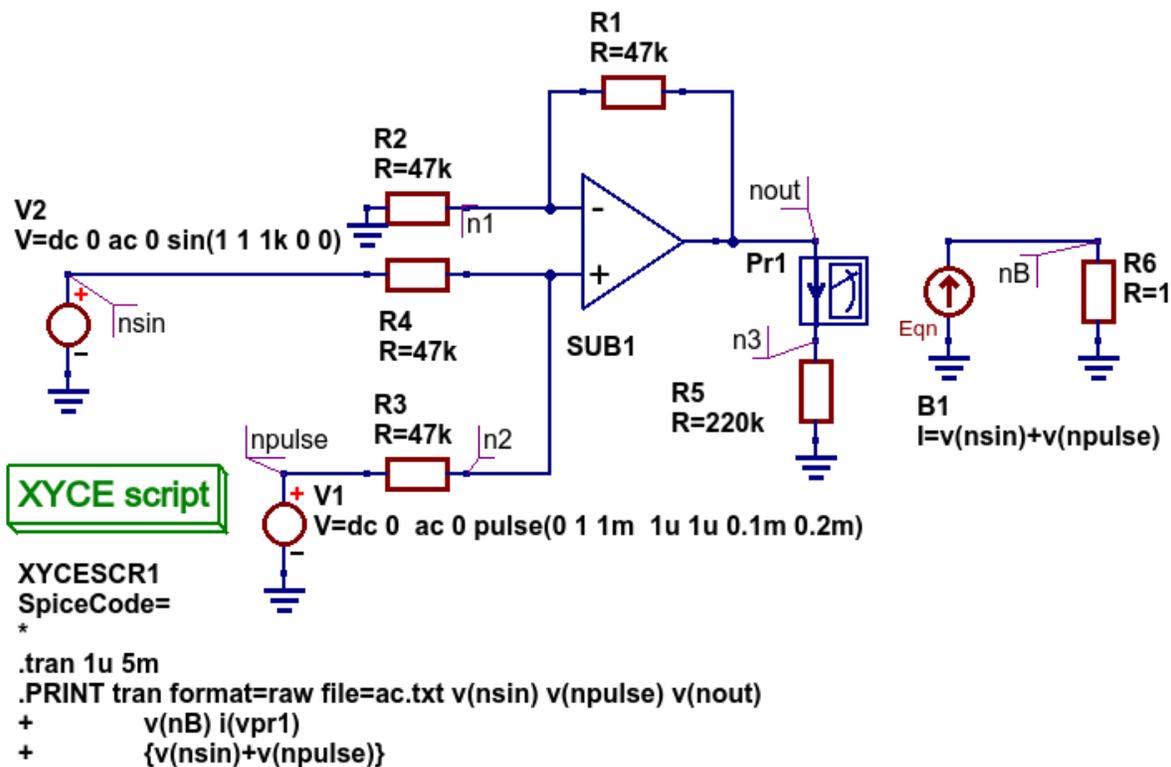


Figure 6.10 An ideal OPAMP adder test circuit which demonstrates the Xyce transient `.PRINT` statement syntax.

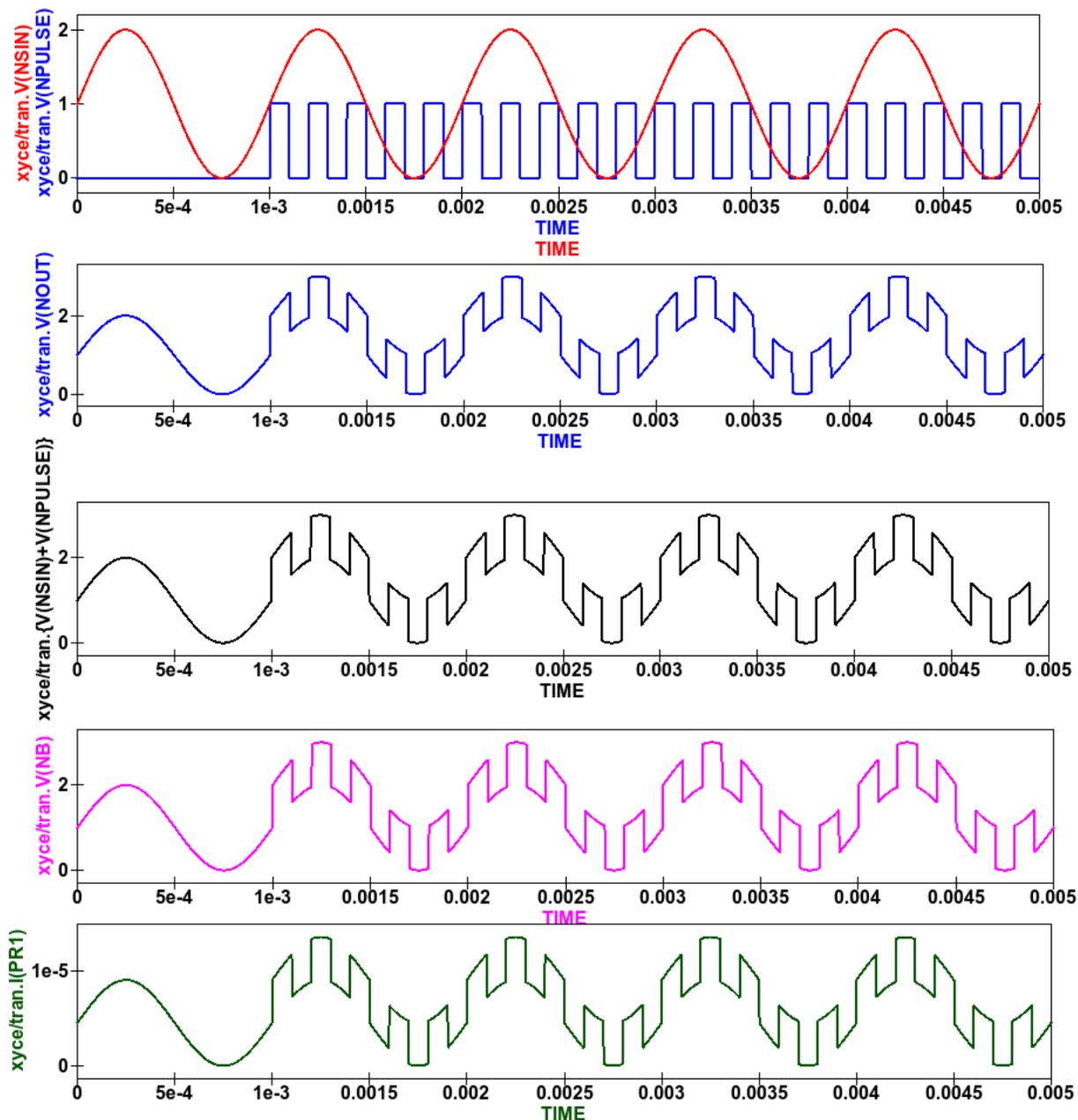


Figure 6.11 Qucs-S/Xyce plotted data illustrating different types of `.PRINT` argument.

The **Xyce script** component is designed to allow users to embed a Xyce SPICE netlist on a Qucs-S circuit schematic. The main purpose of this feature is to allow users to construct Xyce simulation netlists which contain the fundamental simulation commands, like `.ac`, `.tran` and `.hb`, and less common simulation statements, like `.four`, and `.sens`, plus other SPICE netlist statements, including data write statements. Anyone interested in exploring this topic further should read the Xyce user and reference documentation then experiment with a few trial simulations. In the future it is possible that the Qucs-S developers will add to this document a series of example simulations which demonstrate additional uses of the **Xyce script**.

6.5 Ngspice and Xyce H SPICE style .measurement output data processing

6.6 Qucs-S emulation of the Qucs PlotVs() function

The Qucs PlotVs() function allows users to select a specific output data vector as a plot X variable and to plot a different output data vector as the Y variable, for example in AC simulation users may require a plot where the X axis is in angular form ω in radians, where $\omega = 2 \cdot \pi \cdot f$ rather than frequency f in Hz. Figure 6.12 gives a simple RC low pass filter circuit with different output data visualization plots. In this example graph (a) shows a Qucs **locus** plot of node voltage $V(nout)$, graph (b) shows a Qucs **polar** plot of node voltage $V(nout)$, graph (c) shows a Qucs-S simulated **PlotVs()** plot of the imaginary part of voltage $V(nout)$ plotted against the real part of voltage $V(nout)$, and finally graph (d) presents the voltage transfer function $V(nout)/V(nin)$ plotted against frequency. Notice that graphs (a) and (c) are identical. Figure 6.12 also illustrates how Qucs **Equation** blocks and Qucs-S **Nutmeg** blocks can be used to set different properties on a single circuit schematic: remember **Equation** blocks are actioned before simulation and **Nutmeg** blocks after simulation. Unfortunately, the Qucs PlotVs() function is not implemented by Ngspice, SPICE OPUS or Xyce. To eliminate this deficiency the Qucs-S Development Team have added program code which simulates **PlotVs()** allowing users to select which Y axis output vector is plotted against a chosen X axis data vector. Figure 6.13 shows the Qucs-S drop-down menu which allows users to select both the X and Y data vector variables. On Qucs-S plots the simulated Qucs style PlotVs() graphs are indicated by an @ sign leading the X axis variable name. Notice that the key tab *New Graph* adds the user specified $Y@X$ item to the plot list on clicking it with the left-hand mouse button.

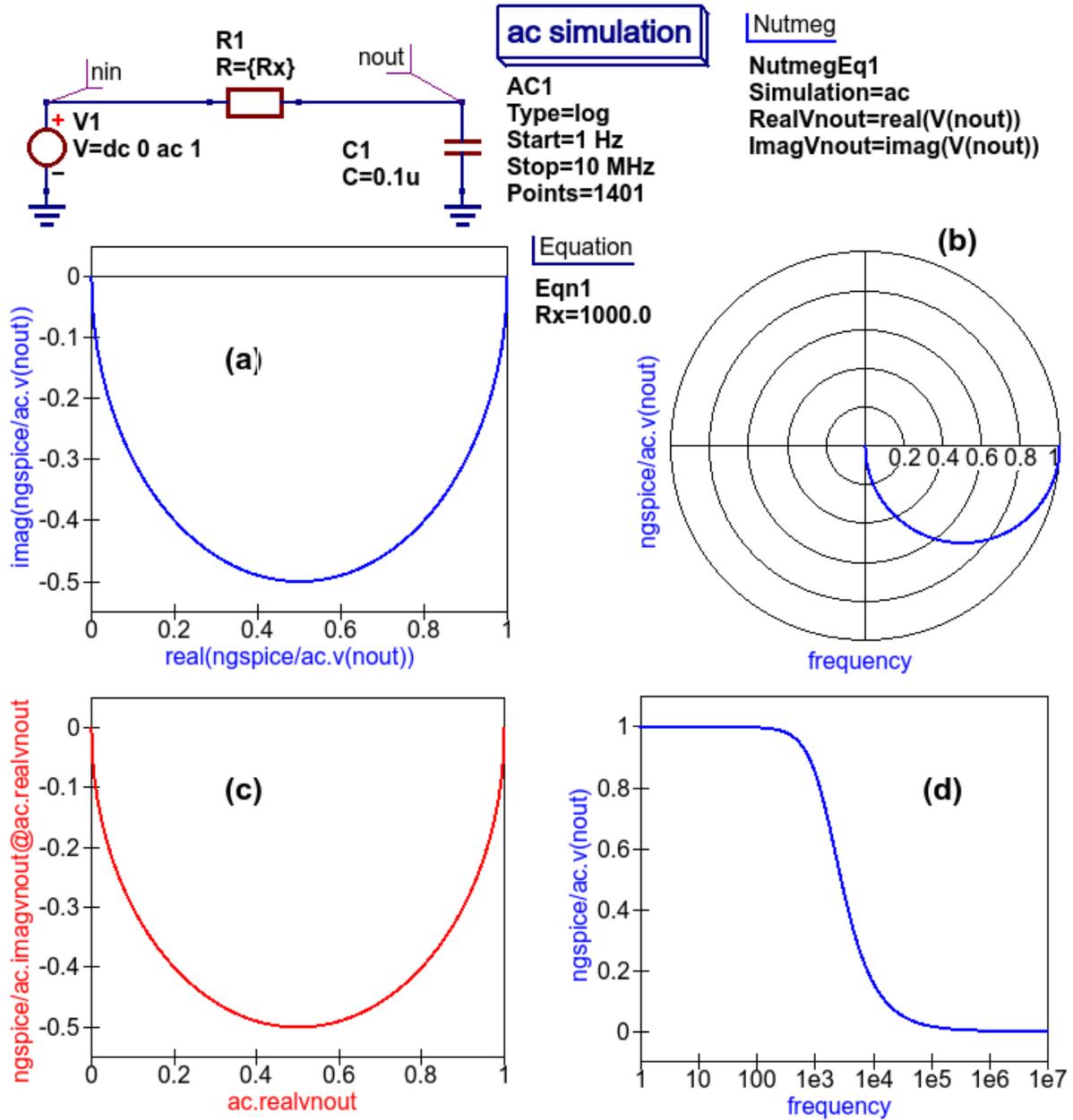


Figure 6.12 A simple RC low pass filter illustrating a number of different output data visualization plot styles.

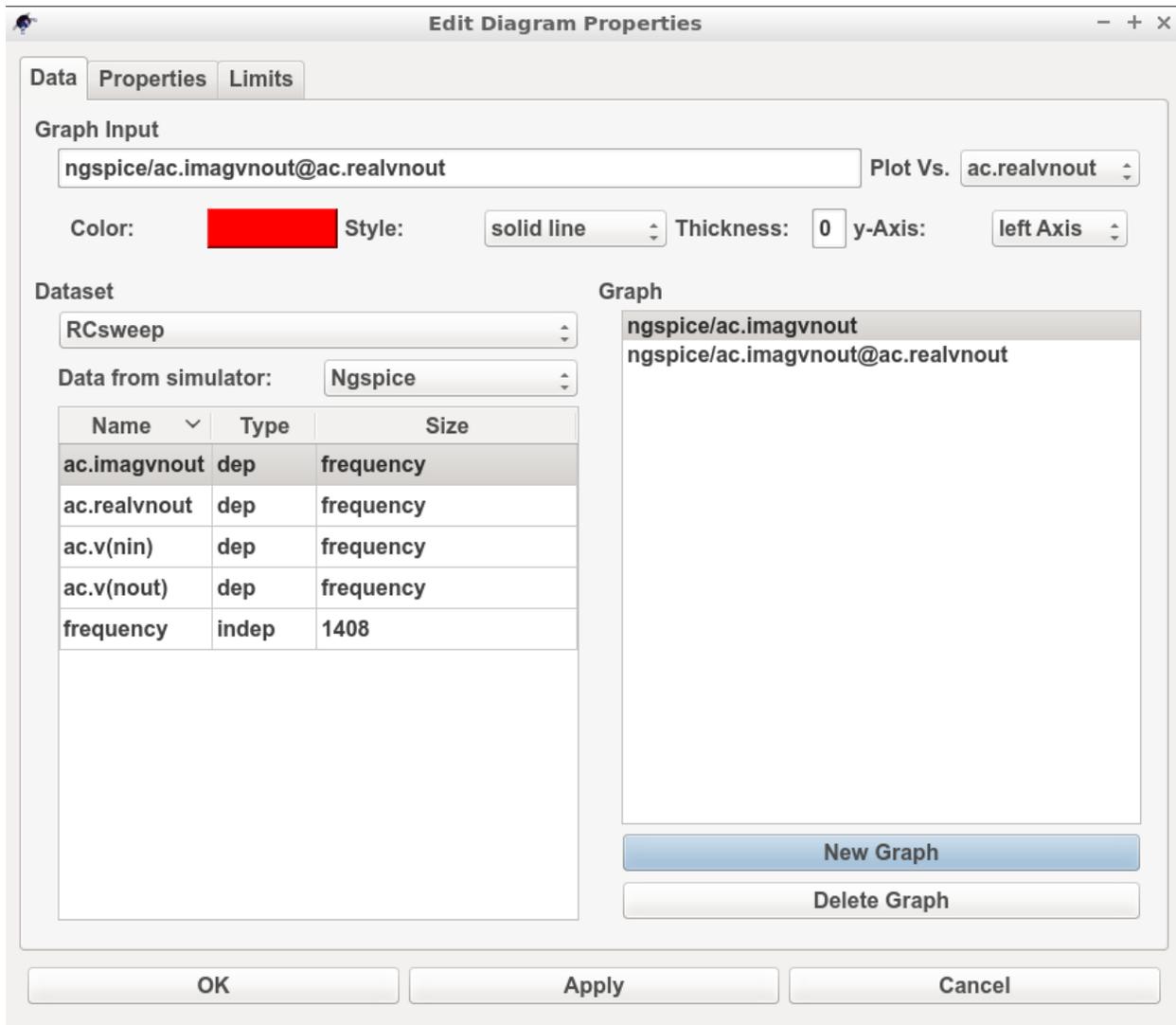


Figure 6.13 The Qucs-S drop-down menu showing The **Edit Diagram Properties** output data list and key tabs for generating a list of Y variables, for plotting against the default X variable, and composite $Y@X$ plot variables.

6.7 Qucs-S output data processing with the Octave numerical analysis and visualization package

6.7.1 Introduction

The Qucs output data post-processing package provides a sophisticated, and very practical, computer aided tool for analysing, and reporting, simulated performance of electronic systems and circuit designs. The fact that it allows schematics, data tables, two and three dimensional graphics plus blocks of user input text to be displayed simultaneously on an interactive graphical interface window, makes the tool suitable for generating “eye catching” slides, reports, theses, books and indeed any other equivalent media. Qucs output data post-processing has a structure and features which are similar to that available with the well known GPL Octave program. Octave is a numerical analysis and visualization package with an extensive range of optional **Tool Boxes**. However, the Qucs data post-processing tool was never intended to be a replacement for packages like Octave. Today, Qucs post-processing has evolved into a facility which allows simple every day data analysis and visualization tasks to be done with ease. Moreover, the post-processing capabilities can be easily learned and applied to most simulation data, making the Qucs data post-processing

routines ideal for both beginners and more knowledgeable users. Qucs-S also makes use of a high percentage of the Qucs post-processing capabilities. Throughout this document readers will find numerous examples of Qucs-S output data processing. In most circuit simulations the Qucs style output data processing is more than adequate for analysing and presenting simulation data. For those cases where a more sophisticated, and often more complex, form of simulation data analysis and visualization is required the Qucs/Qucs-S Development Teams have provided a link between output simulation data and the Octave package. This section introduces this link and describes how it is set up and employed. In order to use Octave with Qucs-S the Octave package must be installed on the computer running Qucs-S. Users are advised to install the Octave 4 series package (at time of writing the current release is Octave 4.0.3) because this includes a Qt based plotting system which interfaces well with Qucs-S. Once Octave is installed and working correctly Qucs-S must be informed by registering the location of the Octave binary on a Qucs-S menu. Firstly, click on the Qucs-S “File” tab (top right hand of GUI window). Secondly, click on menu item **Application Settings** or press keys ctrl+,. This action should result in the display of the menu window shown in Figure 6.14. Thirdly, click on the **Locations** menu tab. This action causes the display of the menu window shown in Figure 6.15. Enter the absolute directory location of the installed Octave program in the box labelled **Octave Path:**, for example `/usr/bin`. If the above sequence is followed correctly Qucs-S and Octave should be linked and ready for post-processing of Qucs-S output data by Octave.

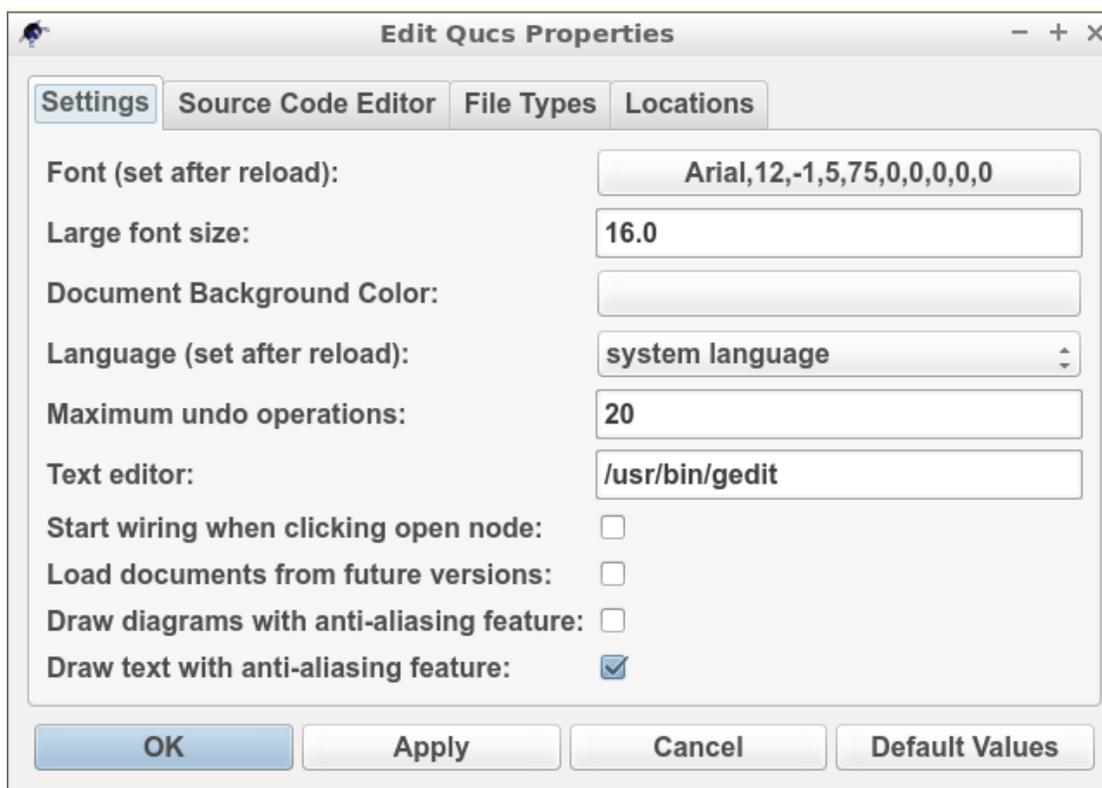
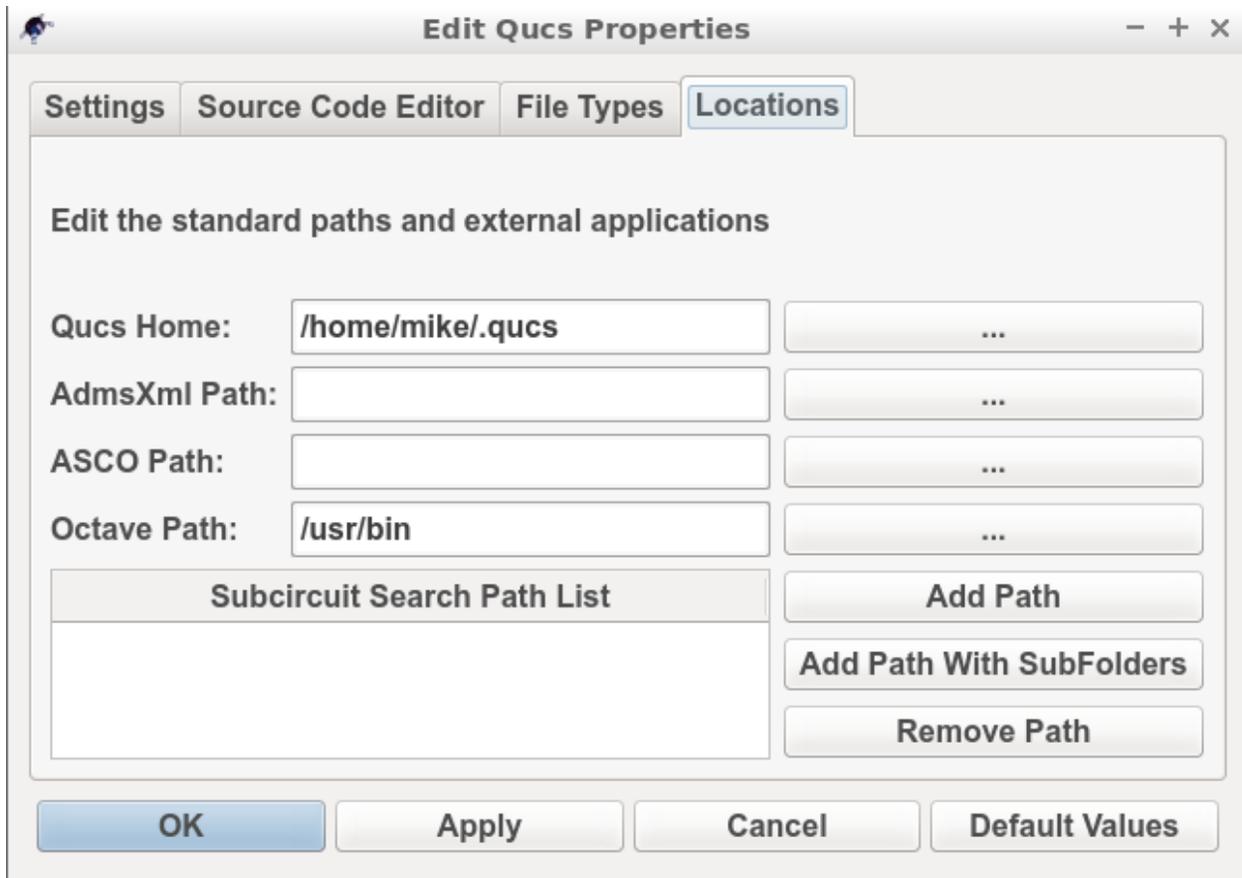


Figure 6.14 File – > Edit Qucs Properties – > **Application Settings** menu.

Figure 6.15 – > **Locations** menu.

6.7.2 Using Octave for numerical analysis and visualization of Qucs-S simulation output data

By combining Qucs-S schematics with Octave script files (*name.m* style files) the post-simulation data processing provided by Qucs-S is extended to include significant extra facilities. Octave not only adds full numerical analysis and programming capabilities but also makes available all the features provided by the optional Octave **Tool Boxes**. To use Octave with Qucs-S for output data processing two Qucs-S files are required; firstly a Qucs-S schematic file called *xxxxxx.sch*, and secondly an Octave script file called *xxxxxx.m*, where name “xxxxxx” must be identical for both files. By using the same name Qucs-S assumes that script file *xxxxxx.m* is to be used to control the post-simulation processing of the output data generated by the simulation of file *xxxxxx.sch*. Figures 6.16 and 6.17 show examples of the *xxxxxx.sch* and *xxxxxx.m* files. Figure 6.16 presents a Qucs-S schematic of a basic RC low pass filter driven from an AC voltage signal comprising a series of independent AC current generators of one ampere magnitude and differing frequencies driving a one Ohm resistor. This circuit generates and filters a composite time domain signal with differing input (node nin) and output (node nout) voltage spectra. Figure 6.17 illustrates how *xxxxxx.m* files can be entered and edited using the Qucs-text editor. In this example the Octave file is called *S4Q_FFT2V_Test.m* and the Qucs-S schematic file *S4Q_FFT2V_Test.sch*. Figure 6.17 also shows the location of the (1) the Qucs-S window where Octave displays output data and messages and (2) the Qucs-S window (bottom **Octave Dock** window) where Octave commands/statements can be entered by users. Note that saved *xxxxxx.m* files are listed under the Octave subsection of the **Content** tab in the **Main Dock** window on the left-hand side of the Qucs-S GUI.

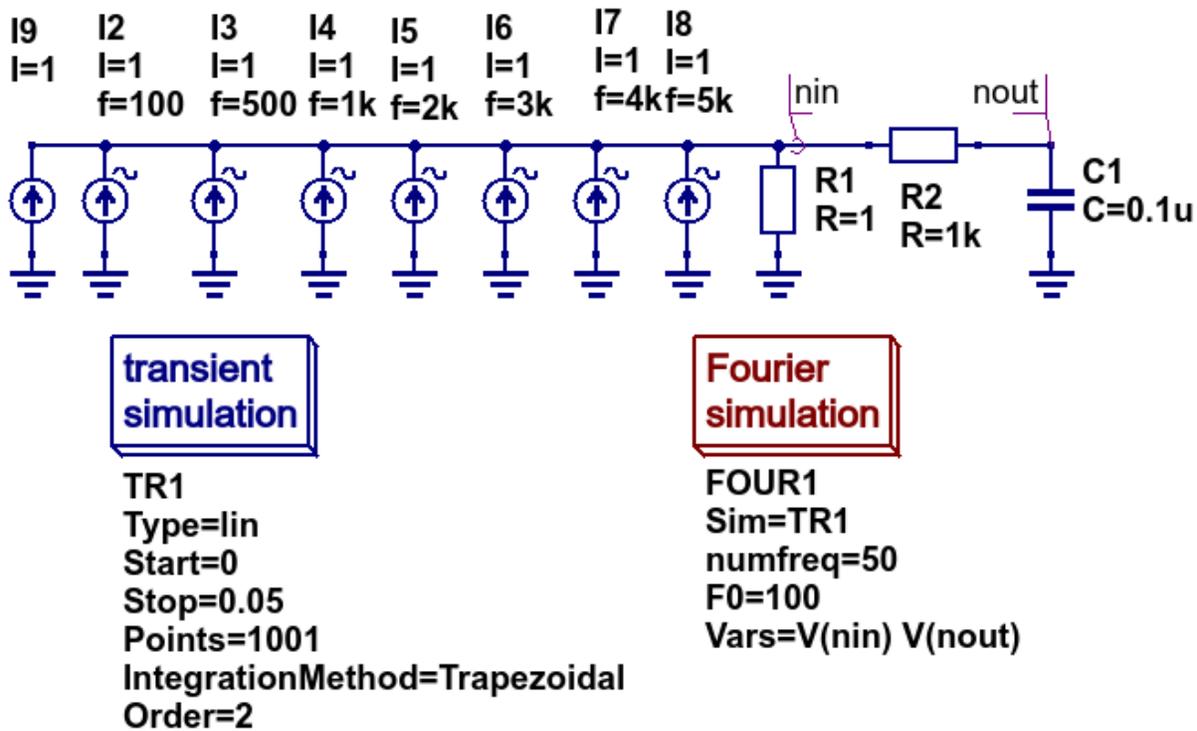


Figure 6.16 Qucs-S circuit schematic *S4Q_FFT2V_Test.sch*.

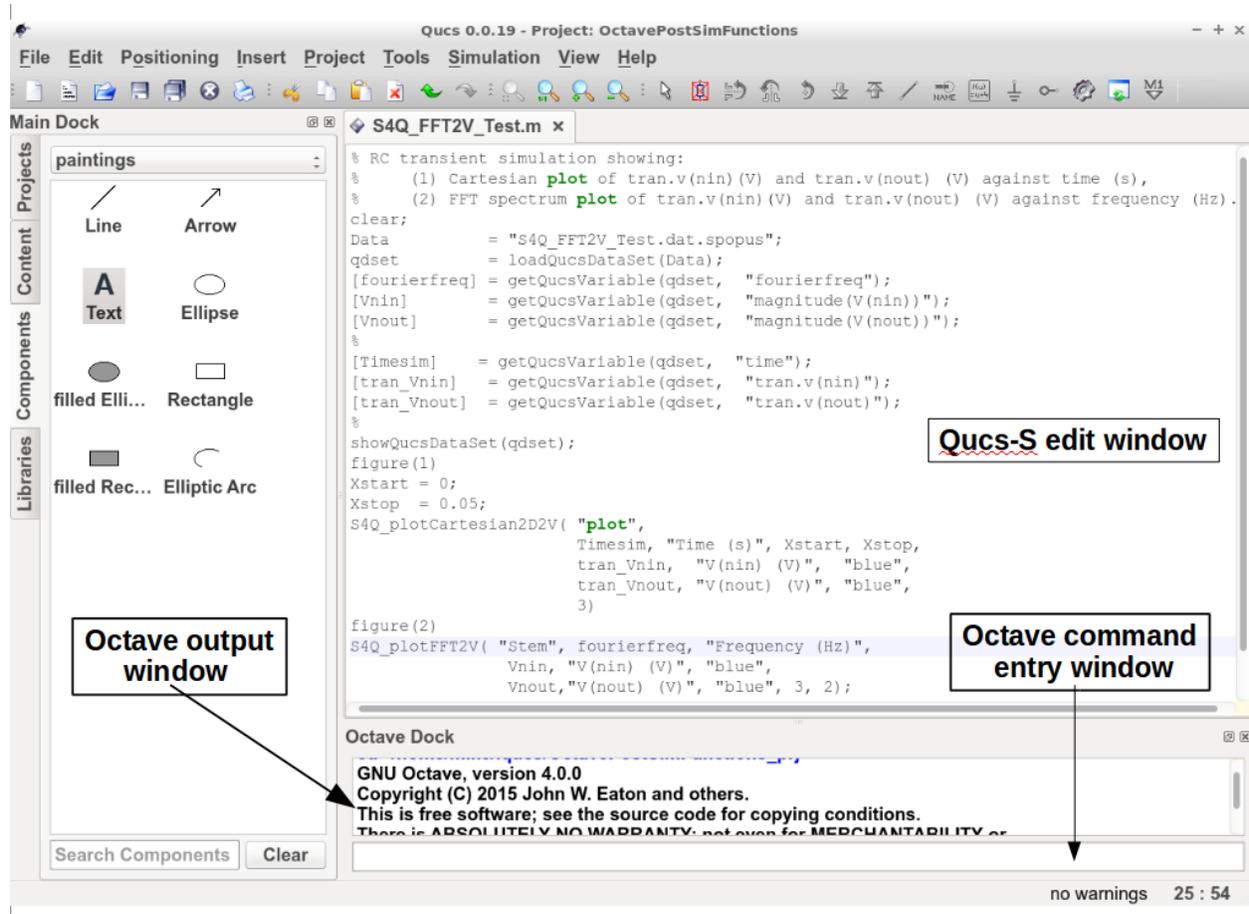


Figure 6.17 Octave post processing script `S4Q_FFT2V_Test.m`.

Once the `xxxxxx.sch` and `xxxxxx.m` files are entered they can be run by Qucs-S to generate circuit simulation output and undertake output processing with Octave. Qucs-S allows this to be done in two ways; firstly *manually* controlled by users and secondly *automatically* controlled by Qucs-S. Both methods require users to load the `xxxxxx.sch` file into the main Qucs-S GUI window before undertaking circuit simulation and output data post-processing:

Manual method

- Load `xxxxxx.sch*` into Qucs-GUI window
- Simulate circuit (press key F2)
- Type the name of the `xxxxxx.m` file without the `.m` extension in the **Octave Dock** command window
- If both the `xxxxxx.sch` and `xxxxxx.n` files are error free Qucs-S simulates the loaded circuit and undertakes the requested output data post-processing with Octave.
- Any requested visualization plots are displayed using Qt in new windows superimposed on the Qucs-S GUI

Automatic method

- Load `xxxxxx.sch*` into Qucs-GUI window
- Simulate circuit (press key F2)
- If both the `xxxxxx.sch` and `xxxxxx.n` files are error free Qucs-S simulates the loaded circuit and undertakes the requested output data post-processing with Octave.
- Any requested visualization plots are displayed using Qt in new windows superimposed on the Qucs-S GUI

Please note only one of the two Qucs-S/Octave simulation data post-processing methods can be active at any one time. To select which *tick* the correct boxes in the **Edit File Properties** window located under **File -> Document Settings**, see Figure 6.18. For example when box **open data display after simulation** is *ticked* and box **run script after simulation** is NOT ticked then the **Manual method** is selected. Reversing which box is ticked results in selection of the **Automatic method** of Octave post-simulation data processing.

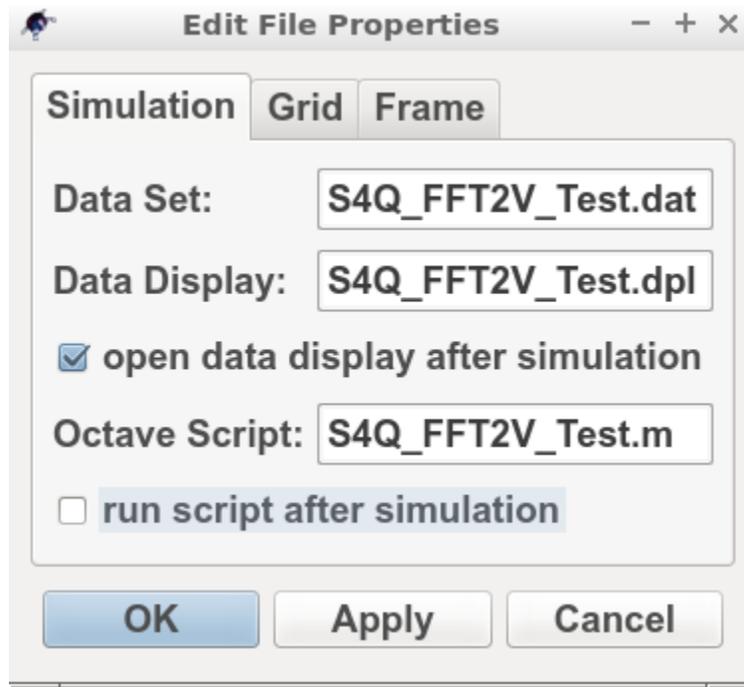


Figure 6.18 The **Edit File Properties** window.

Before introducing the structure and function of the Octave *xxxxxx.m* file the simulation post-processing results for the *S4Q_FFT2V_Test* example are introduced and briefly explained. This allows a number of the basic features required for an Octave *m* file to successfully process Qucs-S simulation output data to be listed before presenting the more complex features of individual Octave numerical analysis and plotting functions, and hopefully help all Qucs-S users understand the background and requirements for writing functioning Octave post-processing *m* scripts. Figure 6.16 shows a selection of the tabulated and graphical results for the RC filter circuit represented by schematic *S4Q_FFT2_Test.sch*. Octave simulation data post-processing scripts are required to undertake a number of basic tasks if they are to successfully extract useful data from simulation performance results: firstly they must be able to read the numerical output data generated by Qucs-S and convert this information into a numerical format which Octave can read and process, secondly they must instruct Octave as to the data processing tasks that it is required to undertake and thirdly they must be able to tabulate, and/or plot the transformed data in a format that can be easily understood by Qucs-S users. How this is done forms the central topic of the next part of this document.

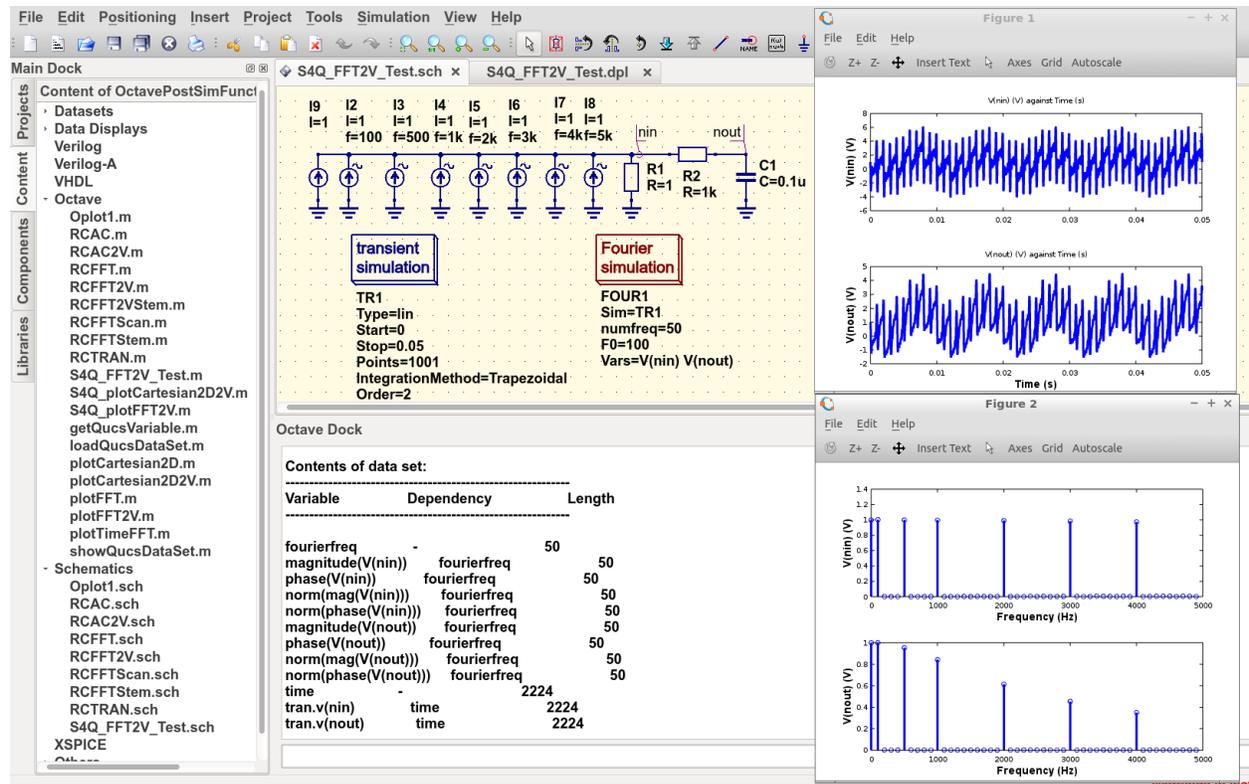


Figure 6.19 Qucs-S/Octave output data results for example circuit illustrated in Figure 6.16.

6.7.3 The structure and content of Octave xxxxxx.m script files

Octave xxxxxx.m script files are one of the principle tools available to Qucs-S users for controlling the post-processing of Qucs-S output data. They allow the resources provided by the Octave numerical analysis and visualization package, and its optional **Tool Boxes**, to be used as an advanced output data analysis tool, allowing detailed analysis of circuit and system performance. Applying Octave for this purpose is very worth while but does however, require users to be proficient with MATLAB/Octave numerical analysis and programming. Figure 6.20 shows a copy of the Octave post-process output data processing script *S4Q_FFT2V_Test.m* previously introduced in section 6.7.2. On the right-hand side of Figure 6.20 is snapshot of the Qucs-S **Main Dock** window where a numbered set of notes outlining each major section of the *S4Q_FFT2V_Test.m* script. The list of headings on the right-hand side of Figure 6.20 indicates where the files referenced in the *S4Q_FFT2V_Test.m* script are stored in the current Qucs-S project. Notice that copies of specific Octave functions written to process script xxxxxx.m are also stored in the current Qucs-S project directory, alongside xxxxxx.sch schematic files. Octave post-simulation output data processing scripts consist of a series of sections which are actioned as a sequence of sequential stages, typically these are

Section 1

- This section is at the start of a xxxxxx.m script. In most instances it consists of a group of comment statements which outline script specification and its use

Section 2

- Section 2 consists of a series of statements which define the name of the Qucs-S simulation output data file , *S4Q_FFT2V_.Test.dat.spopus* in Figure 6.20, and the name of the Octave variable (qdataset) that stores the Octave version of Qucs-S Data converted by function loadQucsDataSet(). In this example names, types and sizes of individual Octave variables held by qdataset are displayed in the **Octave Dock** output window by calling Octave function ShowQucsDataSet(qdataset). Notes 1. to 3., Figure 6.20, provide more detail.

Section 3

- Section 3 is primarily made up of a series of Octave statements which extract individual Qucs-S output quantities from qdset. The Octave function *GetQucsVariable()* is used for this purpose, see Notes 4. and 5., Figure 6.20. In many instances Section 3 would also include additional Octave statements for calculating values characterizing the properties of the circuit/model being simulated. The full power of the Octave matrix based numerical analysis programming language and its optional **Tool Boxes** are available for this purpose.

Section 4

- Section 4, the last section in the *xxxxxx.m* script is normally reserved for Octave code which outputs the calculated results from Section 3, see notes 6. and 7. Figure 6.20. In the majority of cases this output takes the form of plotted graphs, tabulated data or files. The exact form of the generated output data is entirely under the control of individual users and its form will largely depend a users Octave programming skills.

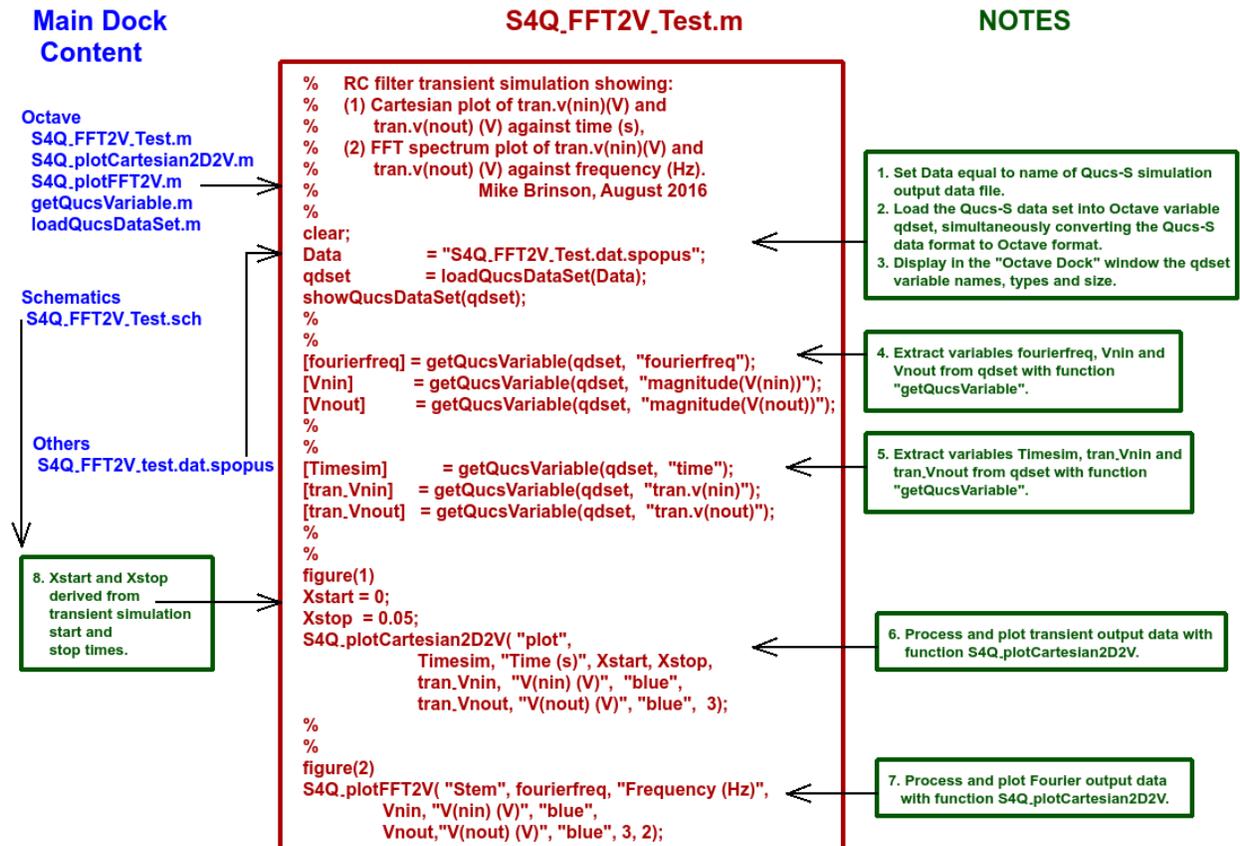


Figure 6.20 An example Octave *xxxxxx.m* script file.

Qucs simulation output data is stored in files designated by *xxxxxx.dat* where *xxxxxx* denotes the name of the schematic illustrating the circuit/model under simulation test. Individual *xxxxxx.dat* files are stored in project files, named *name_prj* and are listed in the **Main Dock** window under subheading **Datasets**. Qucs-S uses a variation of this procedure. This change is necessary because Qucs-S can undertake simulations with any of the external SPICE simulators Ngspice, SPICE OPUS and Xyce currently linked to Qucs-S. Output simulation data from each of these simulators are identified by adding an extra naming tag attached to the end of the original Qucs *xxxxxx.dat* name, yielding

- **Ngspice** : *xxxxxx.dat.ngspice*
- **SPICE OPUS** : *xxxxxx.dat.spopus*
- **Xyce** : *xxxxxx.dat.xyce*

These modified Qucs-S output data files are listed under the **Main Dock** window subheading **Others**. If the schematic under test includes more than one type of circuit simulation, for example see the schematic shown in Figure 6.16, the Qucs-S output data file will include output data for all types of simulation. Displaying the content of a Qucs-S data file lists individual output data items, including their simulation type, name and the numerical data. To assist Qucs and Qucs-S users apply Octave in post-processing simulation output data a number of Octave functions are distributed with each package. These functions are grouped into two main categories.

Group 1 : - Data extraction utilities

Function: `[x] = getQucsVariable(Data, "yyy")`, where

- *x* is the Octave name of the extracted data variable,
- *Data* is the name of the Qucs/Qucs-S data set, and
- "yyy" is the Qucs/Qucs-S name of the extracted data variable.

Function: `dataSet = loadQucsDataSet(dataSETFile)`, where *dataSet* is an array of structures containing the information from the Qucs/Qucs-S data file, and each structure contains the following fields

- *name* is the name of the variable associated with the data in the *data* field of the structure,
- *nameDep* is the name of the dependent variable associated with the data, for example in a transient simulation this will be *time* with another structure holding the *time* data as a variable,
- *dep* is 0 (FALSE) or 1 (TRUE) and flags if the data in the *data* field is dependent on another variable,
- *data* is a vector of values containing the numerical data for a specified variable.

Function: `showQucsDataSet(dataSet)`, where

- *dataset* is the name of the Octave data set which is to be displayed in the **Octave Dock** window.

Group 2 : - Visualization utilities

Function `S4Q_plotCartesian2D(Type, XName, YName, Xlabel, Ylabel, Xstart, Xstop, Linewidth)`, where

- *Type* is the plot style; "semilogx", or "semilogy" or "loglog" else "plot",
- *XName* is the X variable name,
- *YName* is the Y variable name,
- *Xlabel* is the X axis label,
- *Ylabel* is the Y axis label,
- *Xstart* is the X axis start value,
- *Xstop* is the X axis stop value, and
- *Linewidth* is the thickness of plotted line in pixels.

Function `S4Q_plotCarteaian2D2V(XName, Xlabel, Xstart, Xstop, Y1Name, Y1label, Y1Colour, Y2Name, Y2label, Y2Colour, Linewidth)`, where

- *XName* is the X variable name,
- *Xlabel* is the X axis label,
- *Xstart* is the X axis start value,
- *Xstop* is the X axis stop value,
- *Y1Name* is the Y1 variable name,
- *Y1label* is the Y1 axis label,
- *Y1Colour* is the Y1 plot colour,

- *Y2Name* is the Y2 variable name,
- *Y2label* is the Y2 axis label,
- *Y2Colour* is the Y2 plot colour, and
- *Linewidth* is the thickness of plotted line in pixels.

Function S4Q_plotFFT(*Type*, *VName*, *Xlabel*, *Xstart*, *Xstop*, *Ylabel*, *YColour*, *Linewidth*), where

- *Type* is the plot style; “Line” else “stem”,
- *VName* is Y variable plot variable name,
- *Xlabel* is the X axis label,
- *Xstart* is the X start value,
- *Xstop* is the X axis stop value,
- *Ylabel* is the Y axis label,
- *YColour* is the colour of the plot,
- *Linewidth* is the thickness of plotted line or stem curve in pixels.

Function S4Q_plotFFT2V(*Type*, *Freq*, *Xlabel*, *Y1Name*, *Y1label*, *Y1Colour*, *Y2Name*, *Y2label*, *Y2Colour*, *Linewidth*), where

- *Type* is the plot style; “Line” else “Stem”
- *Freq* is the X axis frequency vector,
- *Xlabel* is the X axis label,
- *Y1Name* is the Y1 variable name,
- *Y1label* is the Y1 axis label,
- *Y1Colour* is the Y1 plot colour,
- *Y2Name* is the Y2 variable name,
- *Y2label* is the Y2 axis label,
- *Y2Colour* is the Y2 plot colour,
- *Linewidth* is the thickness of plotted line or stem curve in pixels.

These Octave functions are distributed with the Qucs-S software package. In the future more will be added as the Qucs-S simulation and modelling community develops further useful analysis and visualization functions and sends them to the Qucs-S Development Team for publication as GPL Octave software.

6.7.4 A second Octave xxxxxx.m script file

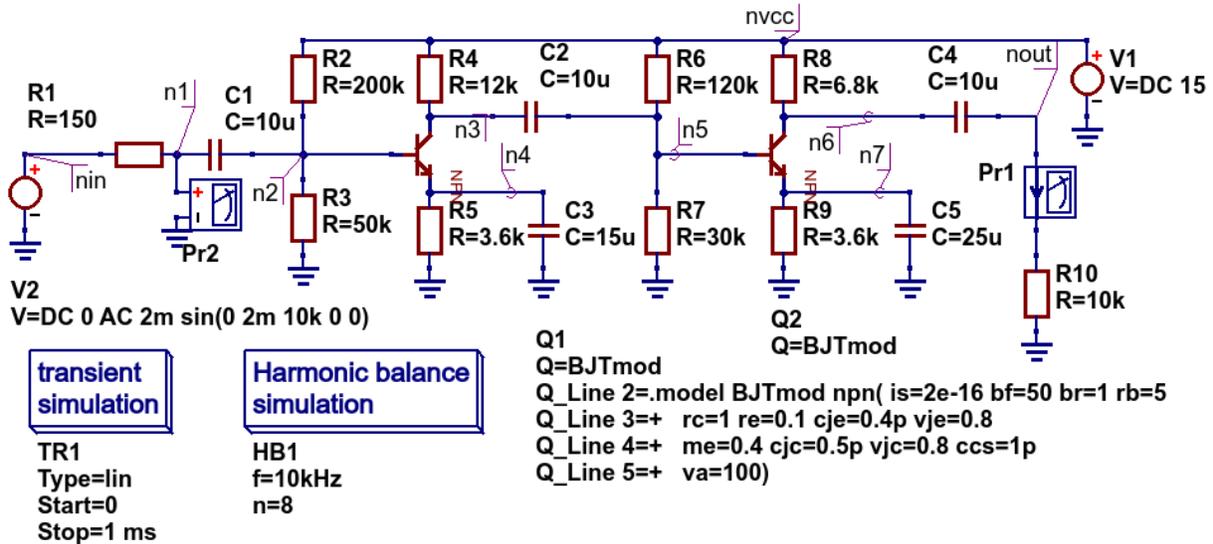


Figure 6.21 A two stage low power BJT amplifier schematic *testTwoStageBJT.sch*.

```

* Qucs 0.0.19 testTwoStageBJT.sch
R2 n2 nvcc 200k
R4 n3 nvcc 12k
C2 n3 n5 10u
R6 n5 nvcc 120k
R8 n6 nvcc 6.8k
R9 0 n7 3.6k
C3 n4 0 15u
R5 0 n4 3.6k
R3 0 n2 50k
C5 n7 0 25u
R1 n1 nin 150
C1 n1 n2 10u
V1 nvcc 0 DC 15
Q2 n6 n5 n7 BJTmod
Q1 n3 n2 n4 BJTmod
.model BJTmod npn( is=2e-16 bf=50 br=1 rb=5
+ rc=1 re=0.1 cje=0.4p vje=0.8
+ me=0.4 cjc=0.5p vjc=0.8 ccs=1p
+ va=100)
R7 0 n5 30k
V2 nin 0 DC 0 AC 2m sin(0 2m 10k 0 0)
R10 0 .net0 10k
C4 n6 nout 10u
VPr1 nout .net0 DC 0
EPr2 Pr2 0 n1 0 1.0
.options hbint numfreq=8 STARTUPPERIODS=2
.HB 10K
.PRINT hb I(VPr1) v(Pr2) v(n1) v(n2) v(n3) v(n4)
+ v(n5) v(n6) v(n7) v(nin) v(nout) v(nvcc)
.END
  
```

```

% testTwoStageBJT.m script GPL file.
% Tested with Xyce serial circuit simulator.
% Mike Brinson August 2016
%
% testTwoStageBJT.m
clear;
Data = "testTwoStageBJT.dat.xyce"; qdset = loadQucsDataSet(Data);
% Extract transient simulation data.
[time] = getQucsVariable(qdset, "TIME");
[tran_ipr1] = getQucsVariable(qdset, "tran.I(PR1)");
[tran_vpr2] = getQucsVariable(qdset, "tran.V(PR2)");
[tran_vnin] = getQucsVariable(qdset, "tran.V(NIN)");
[tran_vnout] = getQucsVariable(qdset, "tran.V(NOUT)");
%
% Extract HB simulation data.
[hbfrequency] = getQucsVariable(qdset, "hbfrequency");
[pr1_ib] = getQucsVariable(qdset, "PR1.Ib");
[pr2_vb] = getQucsVariable(qdset, "PR2.Vb");
[nin_vb] = getQucsVariable(qdset, "NIN.Vb");
[nout_vb] = getQucsVariable(qdset, "NOUT.Vb");
%
showQucsDataSet(qdset);
figure(1)
Xstart = 0; Xstop = 0.001;
S4Q.plotCartesian2D2V("plot", time, "Time (s)", Xstart, Xstop,
  tran_vnin, "V(nin) (V)", "blue", tran_vnout, "V(nout) (V)", "blue", 3);
%
figure(2)
S4Q.plotCartesian2D2V("plot", time, "Time (s)", Xstart, Xstop,
  tran_ipr1, "I(PR1) (A)", "blue", tran_vpr2, "V(PR2) (V)", "blue", 3);
%
figure(3)
S4Q.plotFFT2V("stem", hbfrequency, "Frequency (Hz)",
  abs(nin_vb), "V(nin) (V)", "blue", abs(nout_vb), "V(nout) (V)", "blue", 3);
%
figure(4)
S4Q.plotFFT2V("Stem", hbfrequency, "Frequency (Hz)", abs(pr1_ib),
  "I(PR1) (A)", "blue", abs(pr2_vb), "V(pr2) (V)", "blue", 3);
  
```

Figure 6.22 Xyce synthesised netlist and Octave script file.

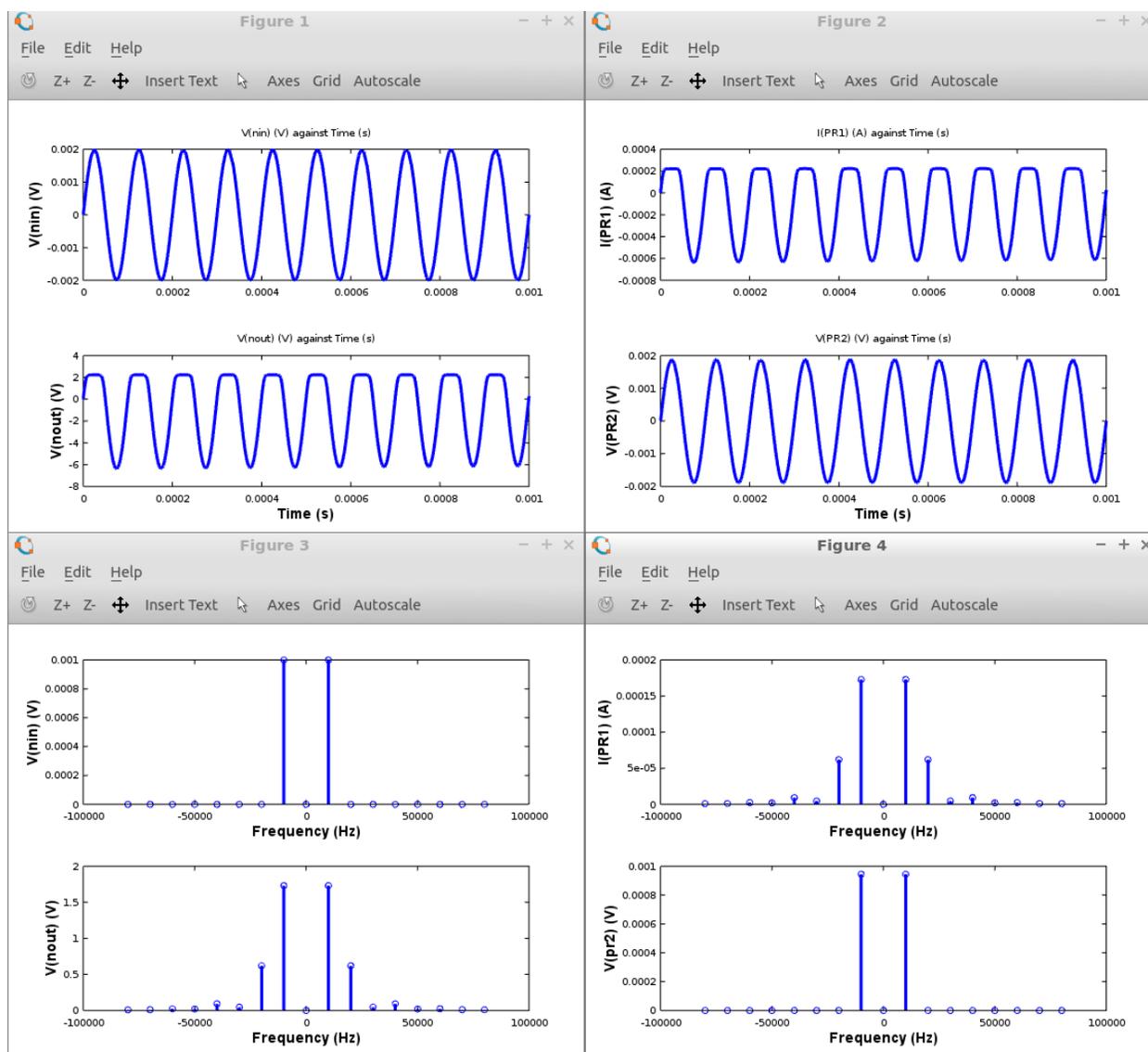


Figure 6.23 Octave plotted transient and Harmonic Balance performance graphs for the two stage low power BJT amplifier..

back to the top

Chapter 7. Qucs and SPICE simulation models that work with ngspice, Xyce and SPICE OPUS

7.1 Introduction

For a circuit simulator to be a useful circuit design aid it must be able to simulate a range of analogue and digital circuits which include passive components, semiconductor devices, integrated circuits and non-electrical devices when needed. By combining Qucs with ngspice and Xyce the number of available simulation models has increased significantly, making the spice4qucs version of Qucs more flexible and powerful, when compared to earlier Qucs releases. One of the primary motives behind the development of spice4qucs was to provide Qucs users with access to published SPICE component models while keeping all the existing Qucs models and simulation capabilities unchanged. With the first

release of spice4qucs, as Qucs-0.0.19S, this aim has largely been achieved. However, there are still significant gaps in the Qucs-0.0.19S simulation capabilities (for example no SPICE 3f5 .PZ simulation yet) and model coverage (for example the number of power analogue and digital models are limited). More work is planned on model development for later releases of the software, including improvements to power device models and the introduction of XSPICE digital models for true mixed-mode analogue-digital simulation. Any improvements and additions to the Qucs-0.0.19S model complement will be recorded in this document as they are introduced by the Qucs Development Team.

This chapter of the spice4qucs-help document consists of two parts; firstly a brief component specification and a more detailed technical reference, and secondly a selection of typical simulation examples which illustrate the use of the various component models. Part two has been added as an aid to help Qucs users appreciate the new style software and the differences between Qucs-0.0.19S and earlier releases of Qucs.

No two circuit simulators are equipped with an identical number, and the same identical types, of circuit simulation models. This is even true with the various implementations of SPICE developed from SPICE 3f5. Hence, by combining Qucs, ngspice and Xyce within one circuit simulation software package there has to be a way of identifying which models work with which simulator. A second feature that further complicates model selection is the fact that supposedly identical models representing the same generic device, for example a BJT, may be based on different physical device equations and a different number of device parameters. In an attempt to identify which model works with which simulator the Qucs Development Team have adopted the following model symbol colouring scheme; existing Qucs models are coloured dark blue (no change), SPICE models which work with both ngspice and Xyce are coloured red, SPICE models that only work with ngspice are coloured cyan and SPICE models that only work with Xyce are coloured dark green. This scheme is not perfect because a number of the original Qucs models also work with ngspice and Xyce. However, for legacy reasons the Qucs Development Team has decided not to change the colours of these models at this time. This decision will probably be reviewed in later releases of Qucs.

The models shown in Figure 7.1 are the original Qucs-0.18 models which can be included in ngspice and Xyce simulations. Please NOTE that for those Qucs users who do not wish to simulate circuits with either ngspice or Xyce all the models distributed with Qucs-0.0.18 work with Qucs-0.0.19S without any modification via the usual *Simulation* (key F2) command. So far no attempt has been made to interface Qucs Verilog-A models with ngspice or Xyce. This task is scheduled for a later spice4qucs development phase.

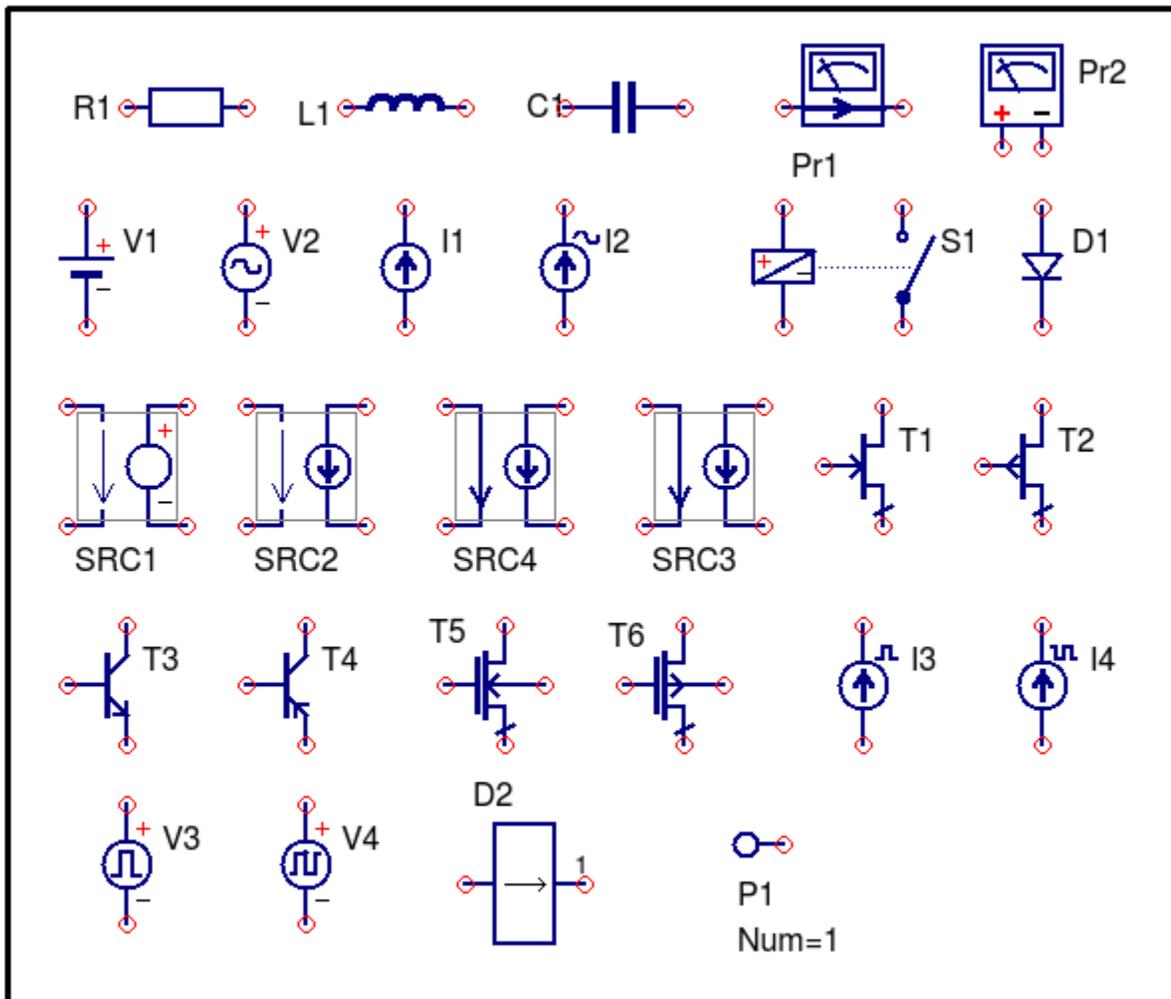
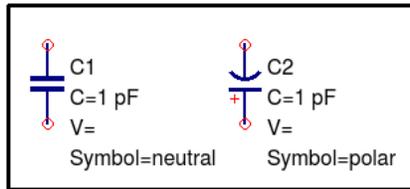


Figure 7.1. Qucs-0.0.18 models that work with ngspice and (sometimes) Xyce.

7.2 Spice4qucs component specifications and technical reference

Capacitor (C)

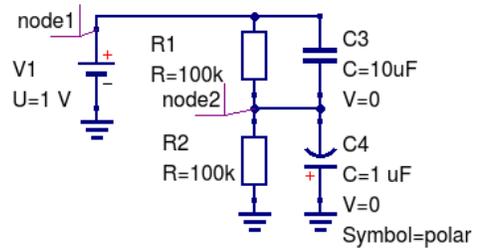
Symbol



dc simulation

DC1

Simple test



Parameters

C	Cnom value of C in Farads.
V	Initial voltage for transient simulation in Volts.

SPICE Format

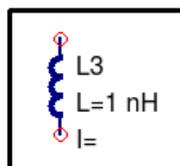
Cname = n1 n2 Cnom [IC=V]

Ngspice/Xyce netlist

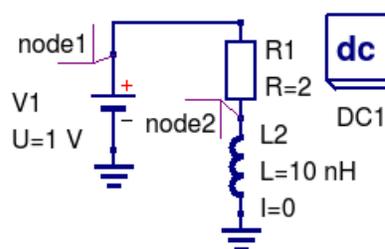
```
* Qucs 0.0.19 Capacitor.sch
V1 node1 0 DC 1
C3 node2 node1 10U IC=0
C4 0 node2 1U IC=0
R1 node2 node1 100K
R2 0 node2 100K
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

Inductor (L)

Symbol



Simple test



dc simulation

DC1

Parameters

L	Lnom value of L in Henries.
I	Initial current for transient simulation in Amps.

SPICE Format

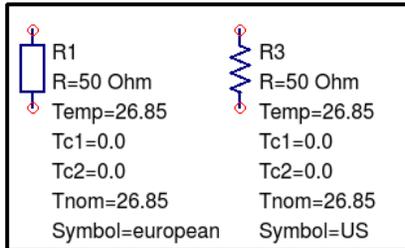
Lname = n1 n2 Lnom [IC=I]

Ngspice/Xyce netlist

```
* Qucs 0.0.19 Inductor.sch
R1 node2 node1 2
L2 0 node2 10N IC=0
V1 node1 0 DC 1
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

Resistor (R)

Symbol



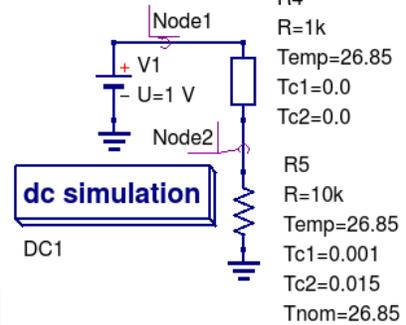
Parameters

R	R(Tnom) in Ohms.
Temp	Simulation temperature in degreesCelsius.
TC1	First order temperature coefficient in Ohms/Celsius.
TC2	Second order temperature coefficient in Ohms ² /Celsius ²
Tnom	Temperature at which nominal parameters are measured, in Celsius.
$\Delta T = Temp - Tnom$	
$R(Temp) = R(Tnom) \cdot (1 + TC1 \cdot DT + TC2 \cdot DT^2)$	

SPICE Format

Rname = n1 n2 Rnom [TC=TC1 , [TC2]]

Simple test



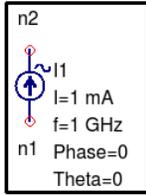
Ngspice/Xyce netlist

```
* Qucs 0.0.19 Resistor.sch
V1 Node1 0 DC 1
R4 Node2 Node1 1K
R5 0 Node2 10K
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

Please NOTE spice4qucs does not implement temperature effects yet.

Independent AC Current Source (I)

Symbol



Parameters

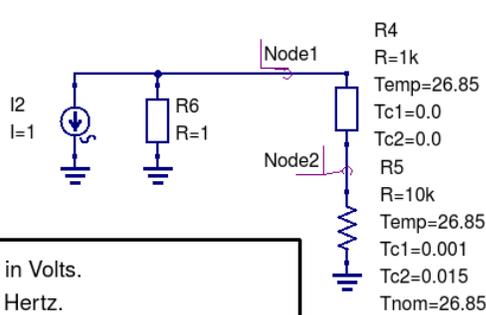
U	Peak current in Volts.
f	Frequency in Hertz.
Phase	Initial phase in degrees.
Theta	Damping factor (transient simulation only).

SPICE Format

```
Iname n1 n2 DC 0 sin( 0 I f 0 Theta) AC I
```

NOTE: The ACCurrentSource DC value is set to 0 amperes.
 The SPICE parameter IO is not used by Qucs and is set to zero amperes for the ngspice and Xyce generators.
 The SPICE parameter TD is not used by Qucs and is set to 0 seconds for ngspice and Xyce generators.

Simple test



ac simulation

```
AC1
Type=log
Start=1 Hz
Stop=10 GMHz
Points=201
```

transient simulation

```
TR1
Type=lin
Start=0
Stop=5 ms
```

Ngspice/Xyce netlist

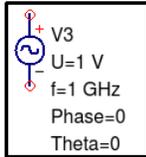
```
* Qucs 0.0.19 ACCurrentSource.sch
R4 Node2 Node1 1K
R5 0 Node2 10K
I2 Node1 0 DC 0 SIN(0 1 1G 0 0) AC 1
R6 0 Node1 1
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
TRAN 4.995e-06 0.005 0
write ACCurrentSource_ran.txt v(Node1) v(Node2)
destroy all
reset

AC DEC 21 1 10GMEG
write ACCurrentSource_c.txt v(Node1) v(Node2)
destroy all
reset

exit
.endc
.END
```

Independent AC Voltage Source (V)

Symbol



Parameters

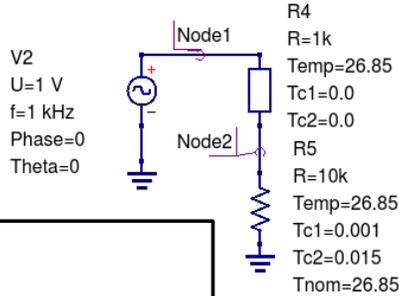
U	Peak voltage in Volts.
f	Frequency in Hertz.
Phase	Initial phase in degrees.
Theta	Damping factor (transient simulation only).

SPICE Format

```
Vname n1 n2 DC 0 sin( 0 U f 0 Theta) AC U
```

NOTE: The ACVoltage source DC value is set to 0 volts.
 The SPICE parameter VO is not used by Qucs and is set to zero Volts for the ngspice and Xyce generators.
 The SPICE parameter TD is not used by Qucs and is set to 0 seconds for ngspice and Xyce generators.

Simple test



ac simulation

```
AC1
Type=log
Start=1 Hz
Stop=10 GMHz
Points=201
```

transient simulation

```
TR1
Type=lin
Start=0
Stop=5 ms
```

Ngspice/Xyce netlist

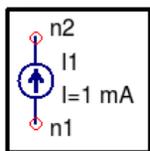
```
* Qucs 0.0.19 ACVoltageSource.sch
R4 Node2 Node1 1K
R5 0 Node2 10K
V2 Node1 0 DC 0 SIN(0 1 1K 0 0) AC 1
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
AC DEC 21 1 10GMEG
write ACVoltageSource_a.c.txt v(Node1) v(Node2)
destroy all
reset

TRAN 4.995e-06 0.005 0
write ACVoltageSource_r.cir.txt v(Node1) v(Node2)
destroy all
reset

exit
.endc
.END
```

Independent DC Current Source (I)

Symbol



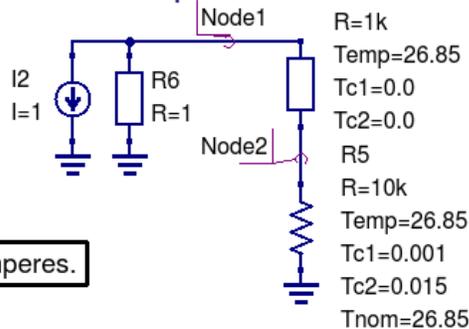
Parameters

I	Current in Amperes.
---	---------------------

SPICE Format

```
Iname = n1 n2 DC I
```

Simple test



dc simulation

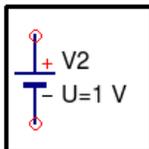
```
DC1
```

Ngspice/Xyce netlist

```
* Qucs 0.0.19 DCCurrentSource.sch
R4 Node2 Node1 1K
R5 0 Node2 10K
I2 Node1 0 DC 1
R6 0 Node1 1
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

Independent DC Voltage Source (V)

Symbol



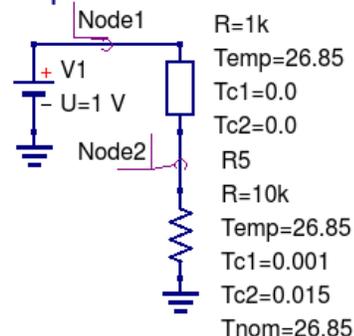
Parameters

U Voltage in Volts.

SPICE Format

Vname = n1 n2 DC U

Simple test



dc simulation

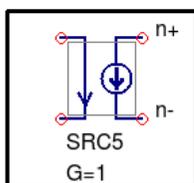
DC1

Ngspice/Xyce netlist

```
* Qucs 0.0.19 Resistor.sch
V1 Node1 0 DC 1
R4 Node2 Node1 1K
R5 0 Node2 10K
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

Linear Current Controlled Current Source (F)

Symbol



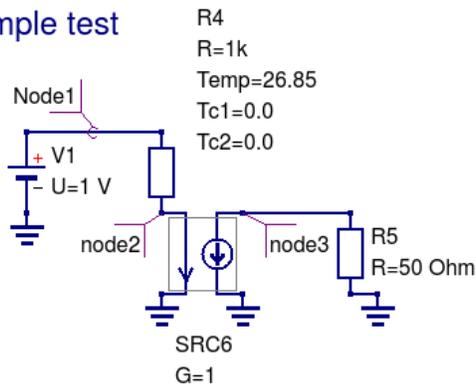
Parameters

G Forward transfer factor.

SPICE Format

Fname = n+ n- vname G

Simple test



dc simulation

DC1

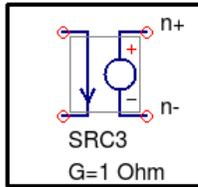
Ngspice/Xyce netlist

```
* Qucs 0.0.19 ICIS.sch
V1 Node1 0 DC 1
R4 node2 Node1 1K
FSRC6 node3 0 VSRC6 1
VSRC6 node2 0 DC 0
R5 0 node3 50
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

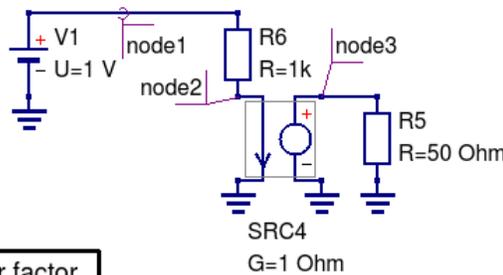
Qucs delay time T is NOT implemented.

Linear Current Controlled Voltage Source (H)

Symbol



Simple test



Ngspice/Xyce netlist

```
* Qucs 0.0.19 ICVS.sch
V1 node1 0 DC 1
R6 node2 node1 1K
R5 0 node3 50
HSRC4 node3 0 VSRC4 1
VSRC4 node2 0 DC 0
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

Parameters

G Forward transfer factor.

SPICE Format

Hname = n+ n- vname G

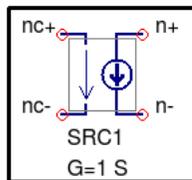
dc simulation

DC1

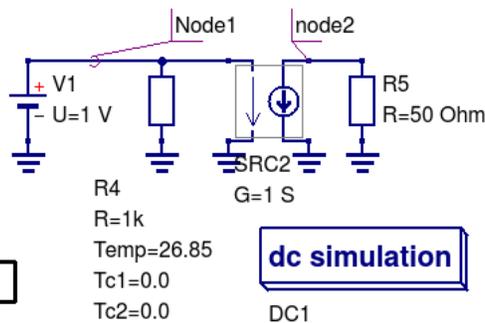
Qucs delay time T is NOT implemented.

Linear Voltage Controlled Current Source (G)

Symbol



Simple test



Ngspice/Xyce netlist

```
* Qucs 0.0.19 VCIS.sch
V1 Node1 0 DC 1
R4 0 Node1 1K
R5 0 node2 50
GSRC2 node2 0 Node1 0 1
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

Parameters

G Forward transfer factor.

SPICE Format

Gname = n+ n- nc+ nc- G

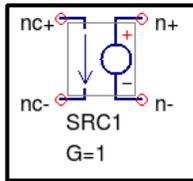
dc simulation

DC1

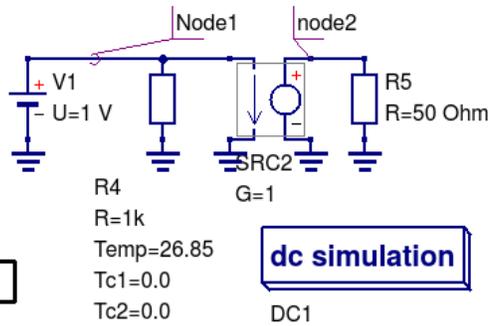
Qucs delay time T is NOT implemented.

Linear Voltage Controlled Voltage Source (E)

Symbol



Simple test



Ngspice/Xyce netlist

```
* Qucs 0.0.19 VCVS.sch
V1 Node1 0 DC 1
R4 0 Node1 1K
ESRC2 node2 0 Node1 0 1
R5 0 node2 50
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
exit
.endc
.END
```

Parameters

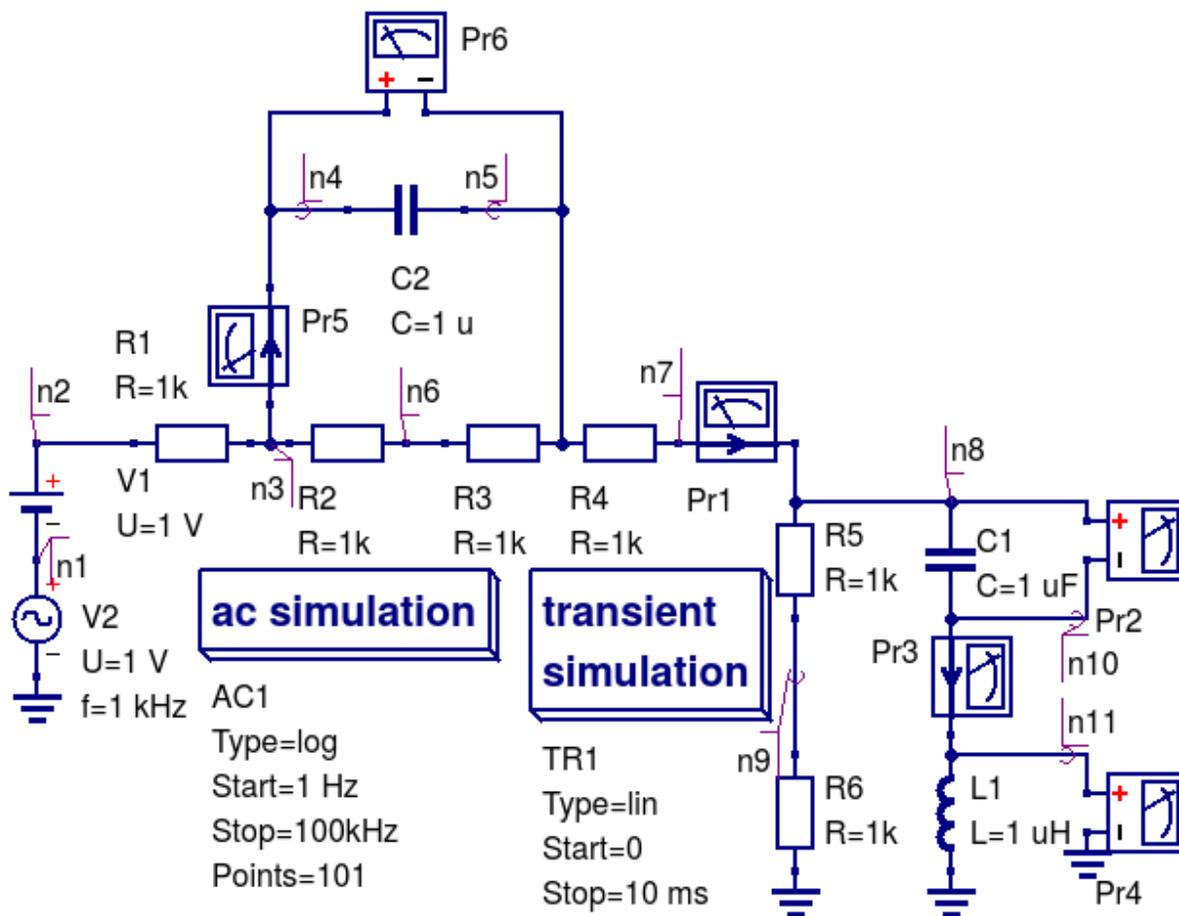
G Forward transfer factor.

SPICE Format

Ename = n+ n- nc+ nc- G

Qucs delay time T is NOT implemented.

Probes



Qucs Netlist

```
# Qucs 0.0.19 Probes.sch

R:R1 n2 n3 R="1k" Temp="26.85" Tc1="0.0" Tc2="0.0" Tnom="26.85"
R:R2 n3 n6 R="1k" Temp="26.85" Tc1="0.0" Tc2="0.0" Tnom="26.85"
R:R3 n6 n5 R="1k" Temp="26.85" Tc1="0.0" Tc2="0.0" Tnom="26.85"
R:R4 n5 n6 R="1k" Temp="26.85" Tc1="0.0" Tc2="0.0" Tnom="26.85"
IProbe:Pr1 n6 n8
R:R5 n10 n8 R="1k" Temp="26.85" Tc1="0.0" Tc2="0.0" Tnom="26.85"
C:C1 n11 n8 C="1 uF" V=""
IProbe:Pr3 n11 n12
R:R6 gnd n10 R="1k" Temp="26.85" Tc1="0.0" Tc2="0.0" Tnom="26.85"
L:L1 gnd n12 L="1 uH" I=""
IProbe:Pr5 n3 n4
C:C2 n4 n5 C="1 u" V=""
VProbe:Pr6 n4 n5
Vdc:V1 n2 n1 U="1 V"
Vac:V2 n1 gnd U="1 V" f="1 kHz" Phase="0" Theta="0"
VProbe:Pr2 n8 n11
VProbe:Pr4 n12 gnd
.AC:AC1 Type="log" Start="1 Hz" Stop="100kHz" Points="101" Noise="no"
.TR:TR1 Type="lin" Start="0" Stop="10 ms" Points="1001" IntegrationMethod="Trapezoidal"
      Order="2" InitialStep="1 ns" MinStep="1e-16" MaxIter="150" reitol="0.001"
      abstol="1 pA" vntol="1 uV" Temp="26.85" LTEreitol="1e-3" LTEabstol="1e-6"
      LTEfactor="1" Solver="CROUTLU" relaxTSR="no" initialDC="yes" MaxStep="0"
```

Ngspice Netlist

```

* Qucs 0.0.19 Probes.sch
R1 n2 n3 1K
R2 n3 n6 1K
R3 n6 n5 1K
R4 n5 n7 1K
VPr1 n7 n8 DC 0 AC 0
R5 n9 n8 1K
R6 0 n9 1K
VPr5 n3 n4 DC 0 AC 0
C2 n4 n5 1U
EPr6 Pr6 0 n4 n5 1.0
V1 n2 n1 DC 1
V2 n1 0 DC 0 SIN(0 1 1K 0 0) AC 1
C1 n10 n8 1U
VPr3 n10 n11 DC 0 AC 0
L1 0 n11 1U
EPr4 Pr4 0 n11 0 1.0
EPr2 Pr2 0 n8 n10 1.0
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
AC DEC 21 1 100K
write Probes.ac.txt v(Pr2) v(Pr4) v(Pr6) VPr1#branch
                VPr3#branch VPr5#branch v(n1) v(n10)
                v(n11) v(n2) v(n3) v(n4) v(n5) v(n6)
                v(n7) v(n8) v(n9)

destroy all
reset
TRAN 9.99001e-06 0.01 0
write Probesa.tran.txt v(Pr2) v(Pr4) v(Pr6) VPr1#branch
                VPr3#branch VPr5#branch v(n1) v(n10)
                v(n11) v(n2) v(n3) v(n4) v(n5) v(n6) v(n7)
                v(n8) v(n9)

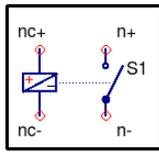
destroy all
reset
.exit
.endc
.END

```

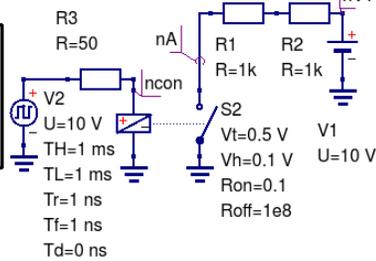
NOTE: To make the Qucs and ngspice netlists readable single lines of width greater than a page width have been indented and continued on one or more lines after the initial entry.

Relay (Voltage controlled switch) (S)

Symbol



Simple test



transient simulation

TR2
Type=lin
Start=0
Stop=10 ms

dc simulation

DC1

Parameters

Vt	Threshold voltage in Volts.
Vh	Hysteresis in volts.
Ron	Resistance in "on" state in Ohms.
Roff	Resistance in "off" state in Ohms.

Ngspice netlist

```
* Qucs 0.0.19 relay.sch
R1 nA ndby2 1K
R2 ndby2 nV1 1K
V1 nV1 0 DC 10
S1 nA 0 ncon 0 MOD_S1 OFF
.MODEL MOD_S1 sw vt=0.5 vh=0.1 ron=0.1 roff=1E8
R3_net0 ncon 50
V2_net0 0 DC 0 PULSE(0 10 0N 1N 1N 0.001 0.002) AC 0
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
TRAN 9.99001e-06 0.01 0
write Trelay1_tran.txt v(nA) v(nV1) v(ncon) v(ndby2)
destroy all
reset
exit
.endc
.END
```

ngspice Format

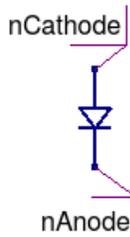
```
Sname n+ n- nc+ nc- model_name <ON> <OFF>
.model model_name sw vt=0.5 vh=0.1 ron=1 roff=1e6
```

Xyce Format

```
Sname n+ n- nc+ nc- model_name <ON> <OFF>
.model model_name vswitch voff=0 von=1 ron=1 roff=1e6
```

Diode (D)

Qucs
diode
model



D1
Is=1e-15 A
N=1
Cj0=1p
M=0.5
Vj=0.7 V
Fc=0.5
Cp=0.0 fF
Isr=0.0
Nr=2.0
Rs=0.1
Tt=0.0 ps
Ikf=0
Kf=1e-16
Af=1.0
Ffe=1.0
Bv=100
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

```
D1 nCathode nAnode DMOD_D1
+ AREA=1.0 Temp=26.85
.MODEL DMOD_D1 D (
+ Is=1e-15
+ N=1
+ Cj0=1e-12
+ M=0.5
+ Vj=0.7
+ Fc=0.5
+ Rs=0.1
+ Tt=0
+ Ikf=0
+ Kf=1e-16
+ Af=1
+ Bv=100
+ Ibv=0.001
+ Xti=3
+ Eg=1.11
+ Tcv=0
+ Trs=0
+ Ttt1=0
+ Ttt2=0
+ Tm1=0
+ Tm2=0
+ Tnom=26.85 )
```

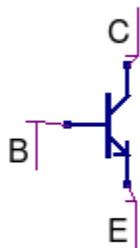
ngspice
netlist

```
D1 nCathode nAnode DMOD_D1
+ AREA=1.0 Temp=26.85
.MODEL DMOD_D1 D (
+ Is=1e-15
+ N=1
+ Cj0=1e-12
+ M=0.5
+ Vj=0.7
+ Fc=0.5
+ Rs=0.1
+ Tt=0
+ Ikf=0
+ Kf=1e-16
+ Af=1
+ Bv=100
+ Ibv=0.001
+ Xti=3
+ Eg=1.11
+ Tcv=0
+ Trs=0
+ Ttt1=0
+ Ttt2=0
+ Tm1=0
+ Tm2=0
+ Tnom=26.85 )
```

Xyce
netlist

BJT npn (Qucs T, ngspice Q)

Qucs
npn
BJT
model



T1
Type=npn
Is=1e-16
Nf=1
Nr=1
Ikf=0
Ikr=0
Vaf=0
Var=0
Ise=0
Ne=1.5
Isc=0
Nc=2
Bf=100
Br=1
Rbm=0
Irb=0
Rc=0
Re=0
Rb=0
Cje=0
Vje=0.75
Mje=0.33
Cjc=0
Vjc=0.75
Mjc=0.33
Xcjc=1.0
Cjs=0
Vjs=0.75
Mjs=0
Fc=0.5
Tf=0.0
Xtf=0.0
Vtf=0.0
Itf=0.0
Tr=0.0
Temp=26.85
Kf=0.0
Af=1.0
Efc=1.0

```

QT1 C B E QMOD_T1
+ AREA=1.0 TEMP=26.85
.MODEL QMOD_T1 npn
+ (Is=1e-16
+ Nf=1
+ Nr=1
+ Ikf=0
+ Ikr=0
+ Vaf=0
+ Var=0
+ Ise=0
+ Ne=1.5
+ Isc=0
+ Nc=2
+ Bf=100
+ Br=1
+ Rbm=0
+ Irb=0
+ Rc=0
+ Re=0
+ Rb=0
+ Cje=0
+ Vje=0.75
+ Mje=0.33
+ Cjc=0
+ Vjc=0.75
+ Mjc=0.33
+ Xcjc=1
+ Cjs=0
+ Vjs=0.75
+ Mjs=0
+ Fc=0.5
+ Tf=0
+ Xtf=0
+ Vtf=0
+ Itf=0
+ Tr=0

```

ngspice
netlist

Ngspice diode (D)

1. Symbol



D1
 D=
 D_Line 2=
 D_Line 3=
 D_Line 4=

2. General form

Dxxxx n n- mname <area=val> <m=val> <pj=val> <off> <ic=vd> <temp=val> <dtemp=val>

3. An example

3.1 Schematic entry



D2
 D=D123
 D_Line 2=.model D123 d(is=2.22e-15 bv=1200 ibv=1.3e-3
 D_Line 3+= cj0=2p tt=1u)

3.2 Generated ngspice netlist

```
D1 B A D123
.model D123 d(is=2.22e-15 bv=1200 ibv=1.3e-3
+ cj0=2p tt=1u)
```

4. Parameters

4.1. Junction DC parameters

BV, IBV, IK (IKV), IKR, IS (JS), ISW, N, RS

4.2. Junction capacitance parameters

CJO (CJ0), CJP (CJSW), FC, FCS, M (MJ),
 MJSW, VJ (PB), PHB, TT

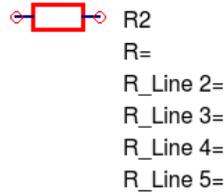
4.3 Temperature effects

EG, TM1, TM2, TNOM (TREF), TRS1 (TRS),
 TTT1, TTT2, XTI, TLEV, TLEVC, CTA (CTC),
 CTP, TCV

4.4 Noise modelling

Ngspice resistor (R)

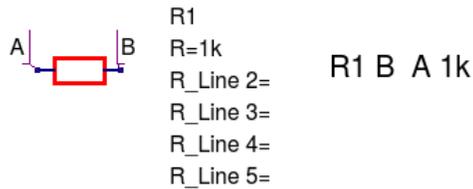
1. Symbol



2. Format

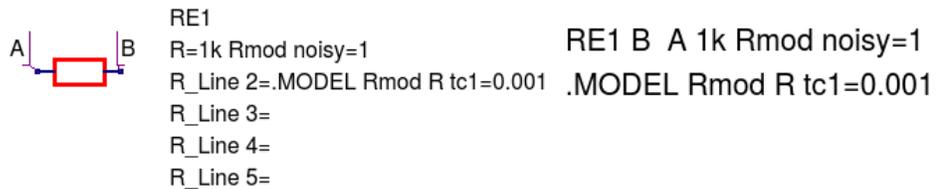
2.1 Fundamental R Rxxxx n+ n- value <mname> <ac=val> <temp=val> <dtemp=val>
+ <m=val> <scale=val> <noisy=0|1>

An example



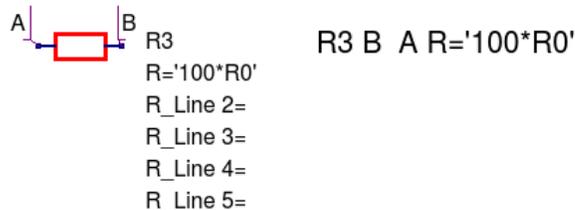
2.2 Semiconductor R Rxxxx n+ n- <value> <mname> <ac=val> <temp=val> <dtemp=val>
+ <l=length> >w=width> <m=val> <scale=val> <noisy=0|1>

An example



2.3 Behavioural R Rxxxx n+ n- R = 'expression'

An example



3. Parameters

Semiconductor R model

TC1, TC2, RSH, DEFW, NARROW, SHORT, TNOM, KF, AF

Ngspice capacitor (C)

1. Symbol



```
C1
C=
C_Line 2=
C_Line 3=
C_Line 4=
C_Line 5=
```

2. Format

2.1 Fundamental C Cxxxx n+ n- value <mname> <temp=val> <dtemp=val>
+ <m=val> <scale=val> <ic=init_ condition>

An example



```
C2      C2 B A 1uf
C=1uf
```

2.2 Semiconductor C Cxxxx n+ n- <value> <mname> <ac=val> <temp=val> <dtemp=val>
+ <l=length> >w=width> <m=val> <scale=val> <ic=init_ condition>

An example



```
C3 B A Cmod L=10u W=20u
.MODEL Cmod C tc1=0.01 tc2=0.001
C=Cmod L=10u W=20u
C_Line 2=.MODEL Cmod C tc1=0.01 tc2=0.001
```

2.3 Behavioural C Cxxxx n+ n- C = `expression`

An example



```
C4      C3 B A C='0.1pf+100*Cj0'
C='0.1p+100*Cj0'
```

3. Parameters

Semiconductor C model

CAP, CJ, CJSW, DEFW, NARROW, SHORT, TC1, TC2, TNOM, DI, THICK

Ngspice inductor (L)

1. Symbol



L1
L=
L_Line 2=
L_Line 3=
L_Line 4=
L_Line 5=

2. Format

2.1 Fundamental L Lxxxx n+ n- value <mname> <temp=val> <dtemp=val>
+ <nt=val> <m=val> <scale=val> <ic=init_ condition>

An example



L2
L=1uh

L2 B A 1uh

2.2 Behavioural L Lxxxx n+ n- L = 'expression'

An example



L3
L=`1000.0n+20.0*{Lh}`

.param LH = 16.0n
L3 B A L=`1000.0n+20.0*{LH}`

3. Parameters

L model

IND, CSECT LENGTH, TC1, TC2, TNOM, NT, MU

Ngspice mutual inductor (K)

1. Symbol

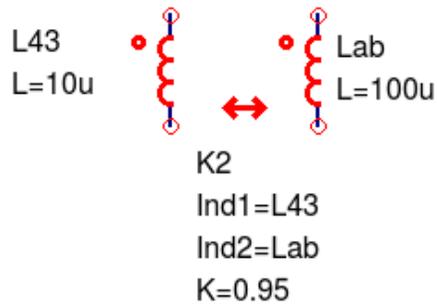


K1
 Ind1=
 Ind2=
 K=0.1

2. Format

Kxxxx Lyyyy Lzzzz value

An example



3. L dot notation: The first node of inductors Lyyyy and Lzzzz is identified by a "dot".

Ngspice independent AC voltage source (V)

1. Symbol

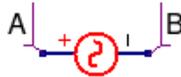


V1
 Vac=
 Vac_Line 2=
 Vac_Line 3=
 Vac_Line 4=
 Vac_Line 5=

2. Format

Vxxxx n+ n- DC 0 <AC <ACMAG <ACPHASE>>>
 + <Transient AC waveform specification>
 + <DISTOF1 <F1MAG <F1PHASE>>>
 + <DISTOF2 <F2MAG <F2PHASE>>>

An example

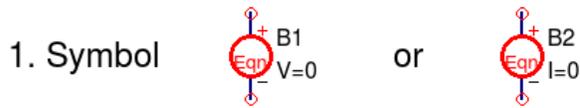


V2
 Vac=DC 0 AC 0.333 45.0
 Vac_Line 2=+ SIN(0 1 1MEG)
 Vac_Line 3=+ DISTOF1 0.1 -90.0

3. Parameters

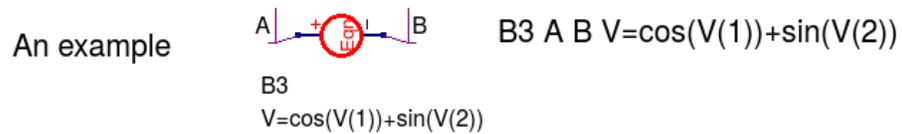
Transient AC waveform specification: SIN(VO VA FREQ TD THETA)

Ngspice non-linear dependent voltage and current sources (B)



2. Format Bxxxx n+ n- <V=exp> <I=exp> <temp=val> <dtemp=val>
+ <TC1=value> <TC2=value>

The expression for V or I may be any function of circuit voltages or current flowing through a voltage source.



3. Temperature coefficients $I(T) = I(Tnom) * (1 + TC1 * (T - Tnom) + TC2 * (T - Tnom)^2)$
 $V(T) = V(Tnom) * (1 + TC1 * (T - Tnom) + TC2 * (T - Tnom)^2)$
where T is the circuit temperature set by temp or by tnom and dtemp. If both temp and dtemp are specified dtemp is ignored.

4. Notes: The expression for V or I may be any function of circuit voltages or circuit current flowing through a voltage source.

5. Allowed operators in B source expressions

5.1 mathematical: +, -, *, /, ^, unary -

5.2 Logical: !=, <>, >=, <=, ==, >, <, ||, &&, !

6. Allowed functions in B source expressions

6.1 Trigonometric: cos, sin, tan, asin, acos, atan

6.2 Hyperbolic: cosh, sinh, tanh, asinh, acosh, atanh

6.3 Exponential and logarithmic: exp, ln, log

6.4 Functions of two variables: min, max, pow

6.5 Other functions: abs, sqrt, floor, ceil

u "Unit step function" with a value of 1 for arguments greater than 0.

u2 Returns a value of 0 for arguments less than 0, 1 for arguments greater than 1 else assumes a value of the argument between these limits.

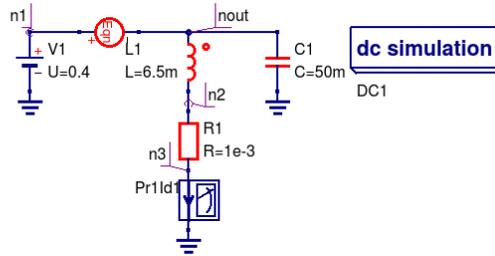
uramp "The integral of the unit step". For an input x, the value is 0 if

x is less than 0 or if x is greater than 0 the value is x

B source example 1: Simulation of the DC characteristics of a diode modelled with a B source pwl function; diode series resistor set at 1e-3 Ohm

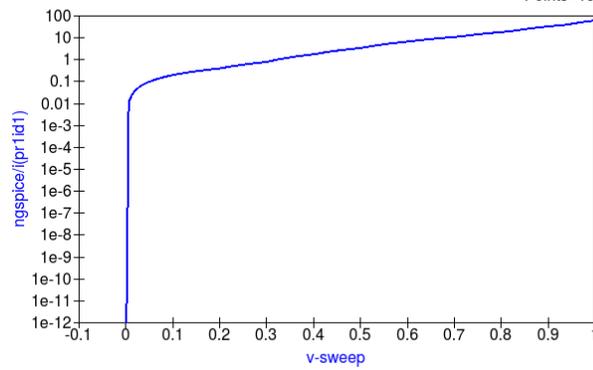
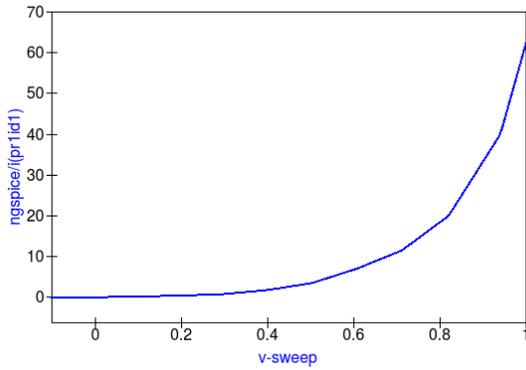
```

B1
I=pwl(V(n1,nout), -100, -1e-12, -50, -1e-12,0,0,
Line 2+= 0.1,0.2,0.2,0.4,0.3,0.8,0.4,1.8,0.5,3.5,0.6,7,
Line 3+= 0.7,11.5,0.8,20,0.9,40,1.00,100,1.26,500,1.5,800,
Line 4+= 1.7,1000,2.0,1600,2.5,2000,3.0,3000,3.5,3000)
    
```

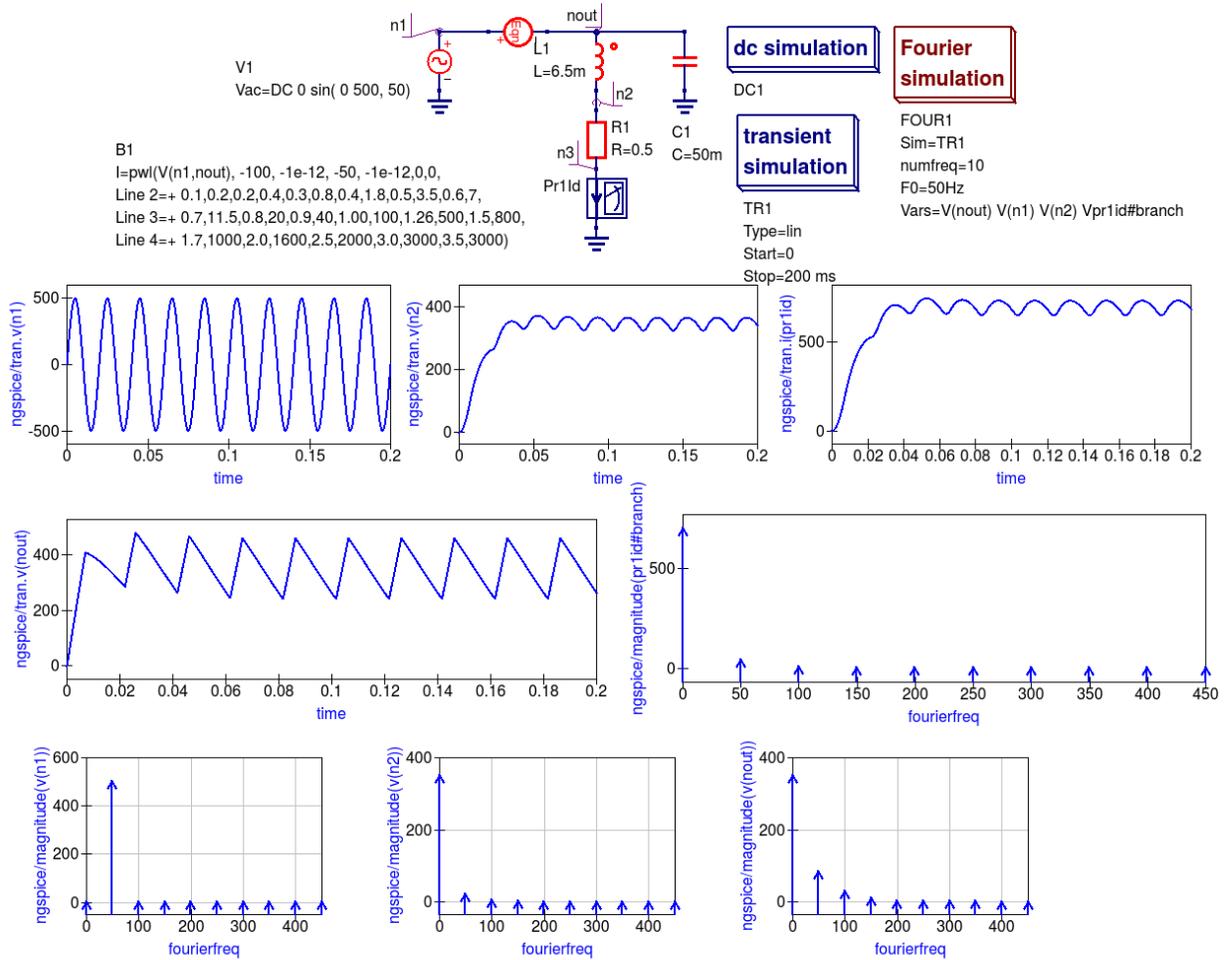


```

Parameter sweep
SW1
Sim=DC1
Type=lin
Param=V1
Start=-0.1
Stop=1
Points=156
    
```



B source example 2: Simulation of the properties of a high power half-wave rectifier circuit with 0.5 Ohm load



```

* Qucs 0.0.19 PDiodeHWR.sch
V1 n1 0 DC 0 sin( 0 500, 50)
C1 nout 0 50m
VPr1Id n3 0 DC 0 AC 0
L1 nout n2 6.5m
R1 n3 n2 0.5
B1 n1 nout I = pwl(V(n1,nout), -100, -1e-12, -50, -1e-12,0,0,
+ 0.1,0.2,0.2,0.4,0.3,0.8,0.4,1.8,0.5,3.5,0.6,7,
+ 0.7,11.5,0.8,20,0.9,40,1.00,100,1.26,500,1.5,800,
+ 1.7,1000,2.0,1600,2.5,2000,3.0,3000,3.5,3000)
.control
set filetype=ascii
echo "" > spice4qucs.cir.noise
TRAN 3.9992e-05 0.2 0
set nfreqs=10
FOURIER 50 V(nout) V(n1) V(n2) Vpr1Id#branch > spice4qucs.four
write PDiodeHWRtran.txt VPr1Id#branch v(n1) v(n2) v(n3) v(nout)
destroy all
reset

exit
.endc
.END

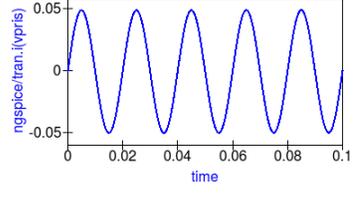
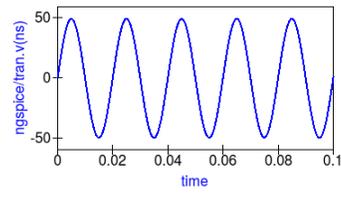
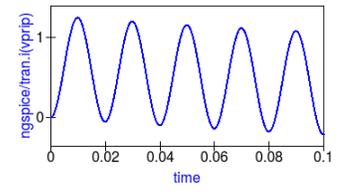
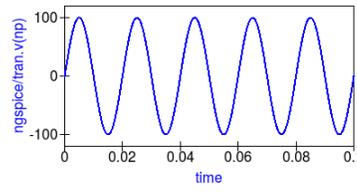
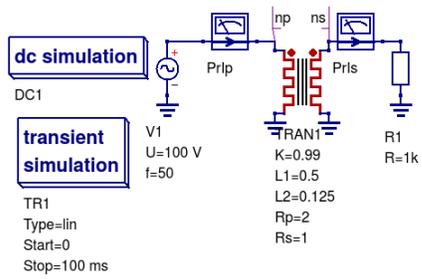
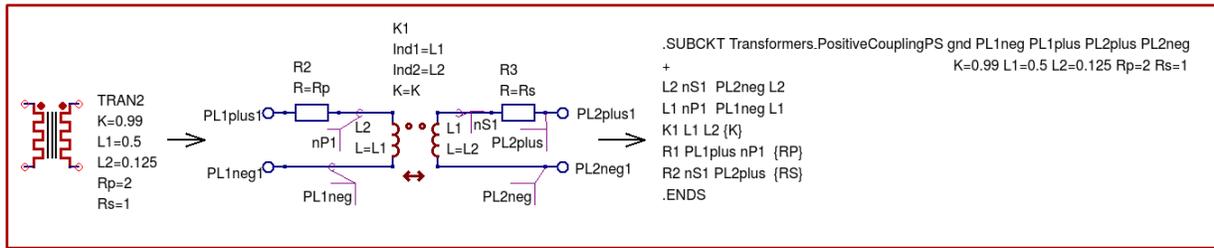
```

ngspice
netlist

6.3 Linear and non-linear transformer models

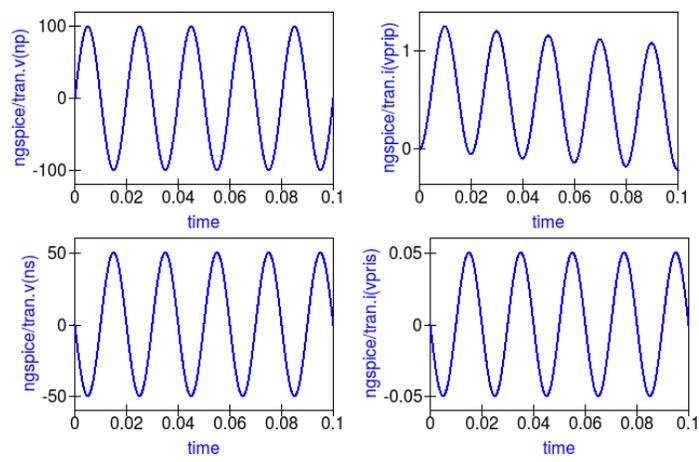
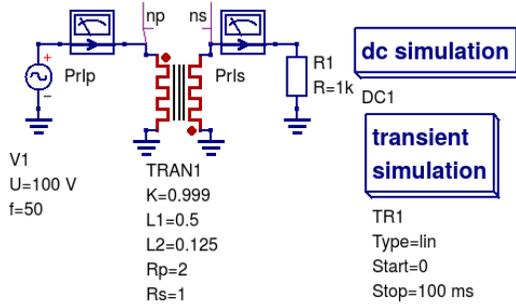
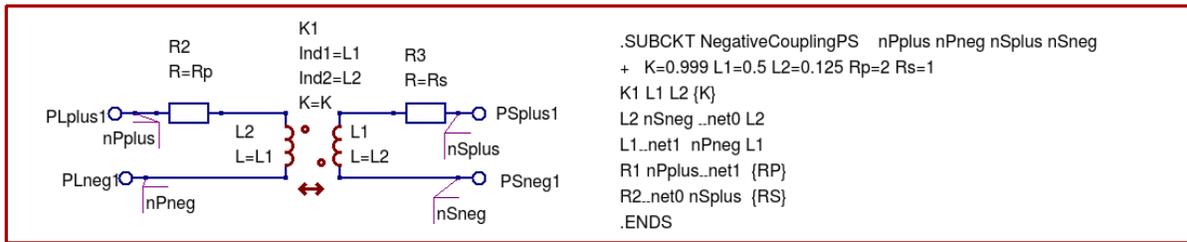
Qucs release 0.0.18, and earlier versions of the software, includes a number of transformer and coupled inductance models. These models are linear with none of the important non-linear effects found in real transformers, including for example, winding resistance, inductance fringing effects and core saturation. The transformer models introduced in this section introduce a number of physical effects which correct the linear transformer limitations. The ideas introduced in their design also act as a set of building blocks which can be used to construct more complex models. The non-linear transformer and core models can be found in the libraries called “Transformers” and “Cores” located in the spice4qucs system library.

Two winding transformer model with in phase primary and secondary voltages and winding resistance



*

Two winding transformer model with out of phase primary and secondary voltages and winding resistance



Two winding transformer model with in phase primary and secondary voltages, winding resistance and fringing inductance

Where LM0 is the inductance of the primary winding with the secondary winding OPEN circuit, and Lmf is the inductance of the primary winding with the secondary winding SHORT circuit.

```

.SUBCKT LossyTransformer1 nPplus nPneg nSplus nSneg
+ Lmo=12u Lmf=0.24u Np=11 Ns=22 Rp=0.017 Rs=0.143
.PARAM K= {sqrt(1-Lmf/Lmo)}
.PARAM Lp={K*Lmo}
.PARAM N={Np/Ns}
.PARAM Ls={K*Lmo*(1/(N*N))}
.PARAM Lpf={(1-K)*Lmo}
.PARAM Lsf={(1-K)*Lmo*(1/(N*N))}
R2 nSneg n4 Rs
L1 n1 n2 Lp
L2 n3 n4 Ls
L3 n1 nPplus Lpf
L4 n3 nSplus Lsf
R1 n2 nPneg Rp
K1 L1 L2 K
Ind1=L1
Ind2=L2
K=K
.ENDS
    
```

Equation

$$Eqn1 \quad K = \sqrt{1 - Lmf/Lmo}$$

$$Lp = K * Lmo$$

$$N = Np / Ns$$

$$Ls = K * Lmo * (1 / (N * N))$$

$$Lpf = (1 - K) * Lmo$$

$$Lsf = (1 - K) * Lmo * (1 / (N * N))$$

V1
V=DC 0 AC 0 sin(0 100 50 0 0)

dc simulation

DC1

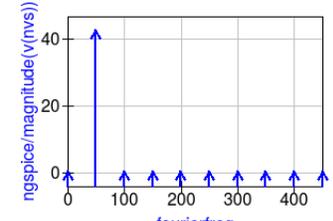
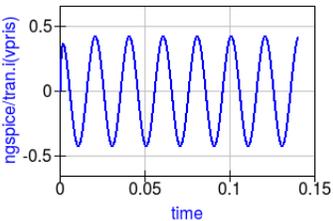
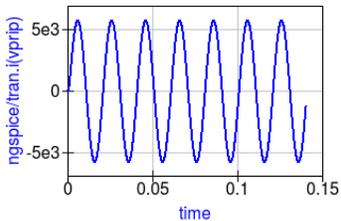
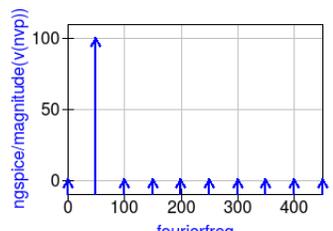
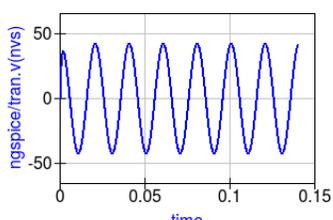
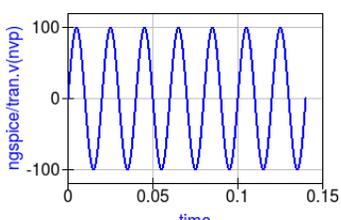
TRAN1
Lmo=12u
Lmf=0.24u
Np=11
Ns=22
Rp=0.017
Rs=0.143

transient simulation

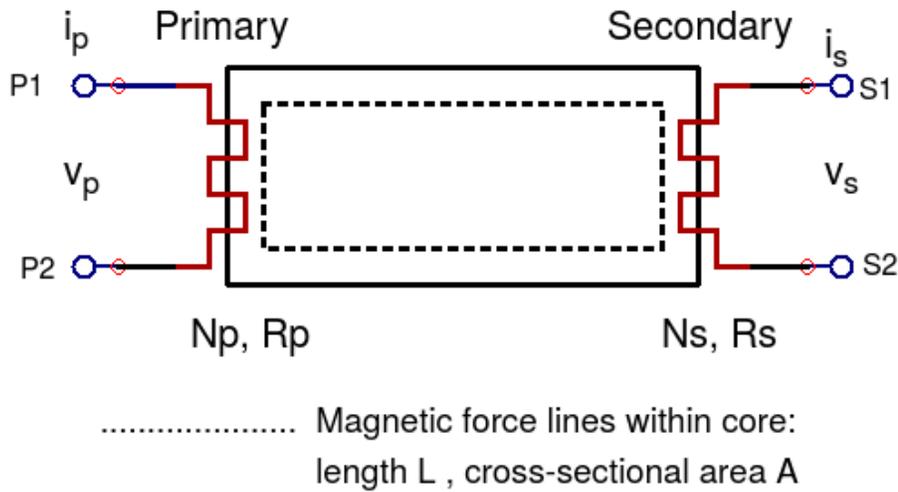
TR1
Type=lin
Start=0
Stop=140 ms

Fourier simulation

FOUR1
Sim=TR1
numfreq=10
F0=50
Vars=V(nVp) V(nVs)



Two winding transformer model with in phase primary and secondary voltages, winding resistance and core saturation



Notes:

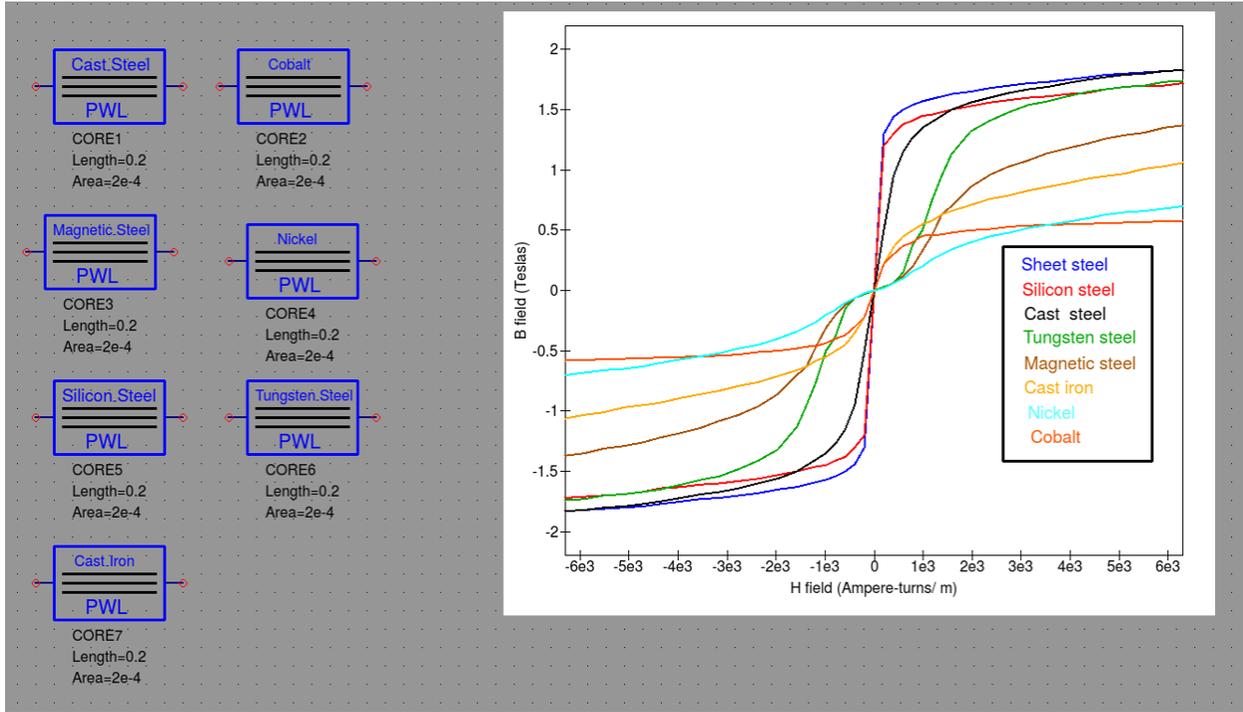
1. Magnetic field strength $H(t) = (N_p \cdot I_p(t) + N_s \cdot I_s(t))/L$.
2. Magnetic flux density $B(t) = f(H(t))$.
3. Primary and secondary winding voltages

$$v(p) = d/dt(N_p \cdot A \cdot B(t))$$

$$v(s) = d/dt(N_s \cdot A \cdot B(t))$$
4. Represent $B(t)$ with non-linear tabular or algebraic function $B(t) = f(H(t))$ whose value depends on the core material.

```
.SUBCKT LossyTransformer2 nPlus nPneg nSplus nSneg nH nB np=4470 ns=870 Rp=30 Rs=5 L=0.2 A=2e-4
L3 ndB 0 1
B3 0 n1 I = pwl ( V(nH), -10000, -1.506, -9000, -1.504, -8000, -1.5035, -7000, -1.053, -6000, -1.502, -5000, -1.501,
+ -4000, -1.5005, -3000, -1.5, -2500, -1.48, -2000, -1.45, -1500, -1.37, -1000, -1.0, -750, -0.825, -500, -0.55, -250, -0.3,
+ 0, 0, 250, 0.3, 500, 0.55, 750, 0.85, 1000, 1.0, 1500, 1.37, 2000, 1.45, 2500, 1.48,
+ 3000, 1.50, 4000, 1.5005, 5000, 1.501, 6000, 1.502, 7000, 1.503, 8000, 1.5035, 9000, 1.504, 10000, 1.506)
R4 0 nH 1
ESRC1 nP2 nPneg ndB 0 {np*A}
ESRC2 nS2 nSneg ndB 0 {ns*A}
R1 nP1 nPlus Rp
R2 nSplus nS1 Rs
HSRC3 nB 0 VSRC3 1
VSRC3 n1 ndB DC 0
HSRC4 nIp 0 VSRC4 1
VSRC4 nP1 nP2 DC 0
HSRC5 nIs 0 VSRC5 1
VSRC5 nS1 nS2 DC 0
BD2I0 0 nH I=({np}*V(nIp))+({ns}*V(nIs))/L
.ENDS
```

Spice4qucs magnetic core library: symbols and B/H specifications



Two winding transformer model with in phase primary and secondary voltages, winding resistance and core saturation (using XSPICE models)

```

.ICOUPLE1
A =Pcouple
A_Line 2=.model Pcouple icouple (num_turns={ntp})

.ICOUPLE2
A =Scouple
A_Line 2=.model Scouple icouple (num_turns={nts})

.SUBCKT Cores_SheetSteel gnd nPplus nPneg Length=0.2 Area=2e-4
ACORE nPplus nPneg SheetSteel
.model SheetSteel core ( mode=1 length = {Length} area={Area}
+ H.array = [ -6299 -5906 -5512 -5118 -4724 -4367 -3937 -3543 -3150 -2756 -2362 -1969 -1575 -1378 -1181 -984 -787 -591 -394 -197 0
+
197 394 591 787 984 1181 1378 1575 1969 2362 2756 3150 3543 3937 4367 4724 5118 5512 5906 6299 ]
+ B.array = [ -1.83 -1.82 -1.81 -1.80 -1.79 -1.77 -1.75 -1.73 -1.72 -1.70 -1.68 -1.65 -1.63 -1.61 -1.59 -1.57 -1.54 -1.50 -1.44 -1.30 0
+
1.30 1.44 1.50 1.54 1.57 1.59 1.61 1.63 1.65 1.68 1.70 1.72 1.73 1.75 1.77 1.79 1.80 1.81 1.82 1.83 ] )
.ENDS
*

.SUBCKT XSPICEPS1 Pplus Pneg Sneg Splus Rp=20 Rs=5 ntp=4350 nts=870
R1 Pplus n3 {Rp}
AICOUPLE1 n3 Pneg n1 0 Pcouple
.model Pcouple icouple (num_turns={ntp})
R2 n4 Sneg {Rs}
AICOUPLE2 n4 Splus n2 0 Scouple
.model Scouple icouple (num_turns={nts})
XCORE2 0 n1 n2 Cores_SheetSteel Length=0.2 Area=2E-4
.ENDS
    
```

```

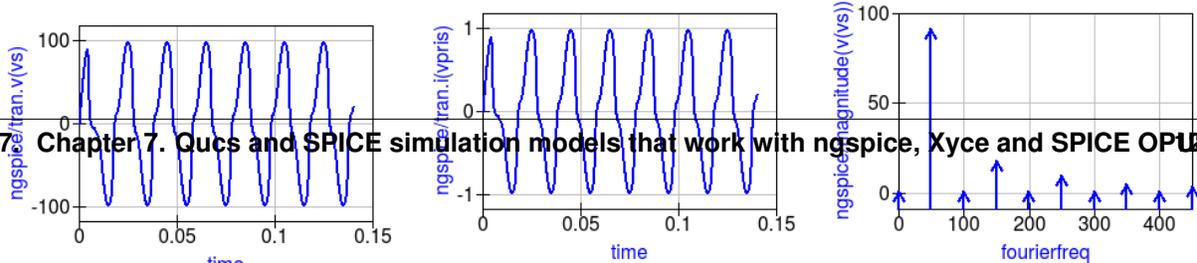
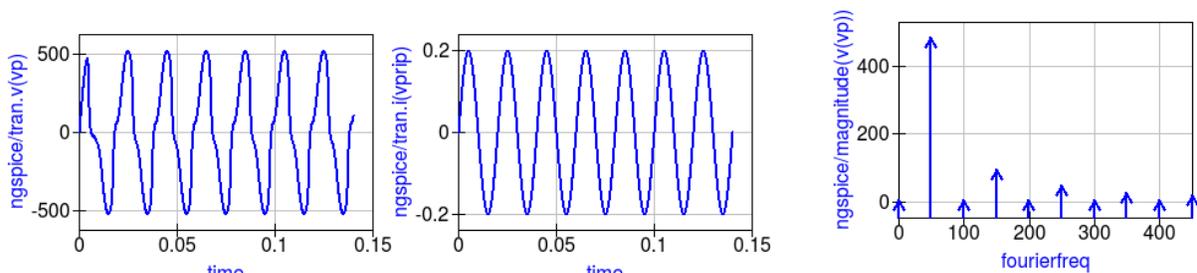
I1 DC1
I=DC 0 AC 0 sin(0 0.2 50 0 0)

R2 R=1e7

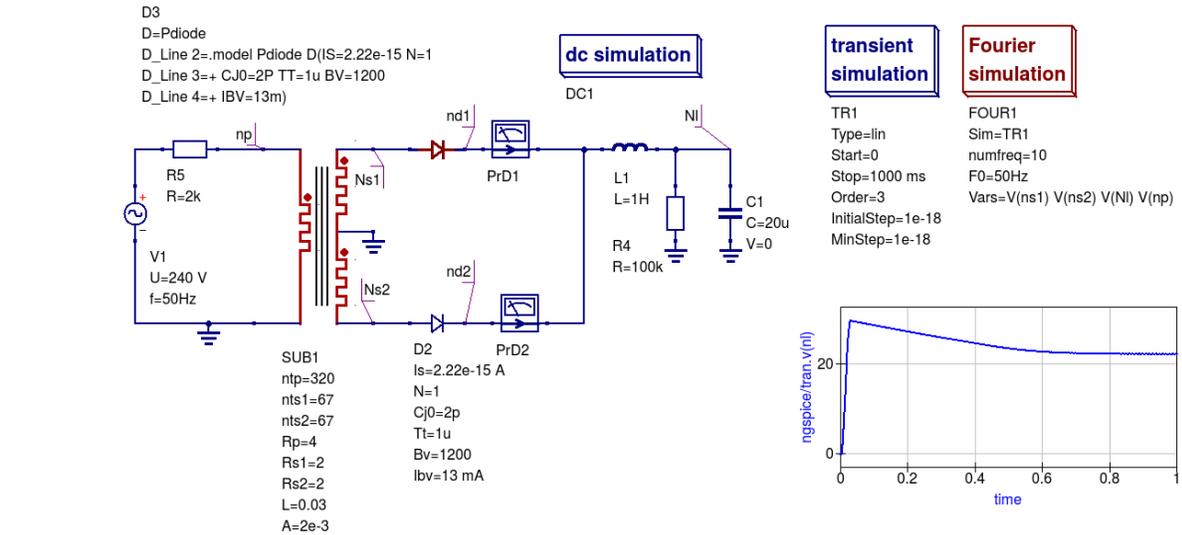
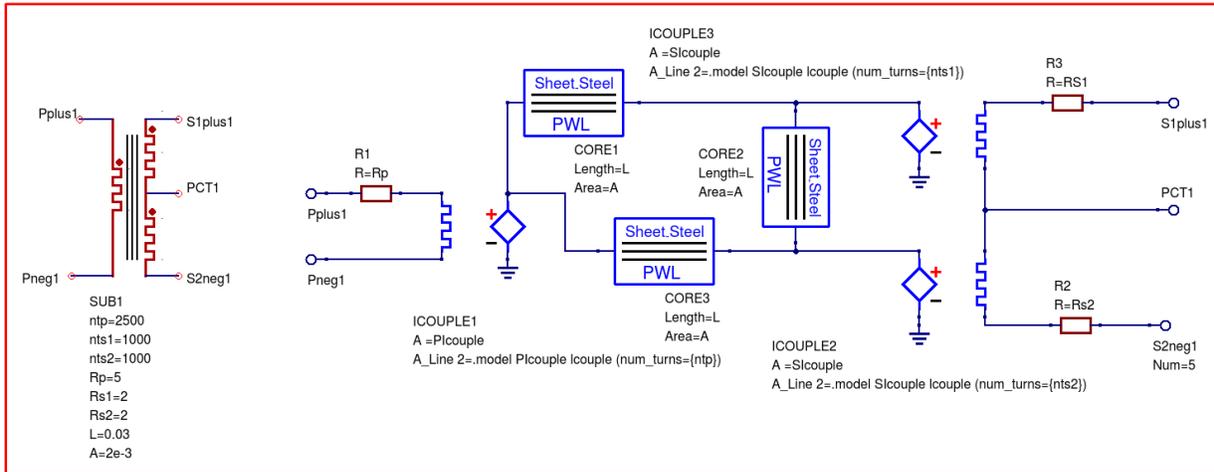
TRAN1
Rp=20
Rs=5
ntp=4350
nts=870

TR1
Type=lin
Start=0
Stop=140 ms

FOUR1
Sim=TR1
numfreq=10
F0=50
Vars=V(vp) V(vs)
    
```



Three winding transformer model with winding resistance and core saturation effects (using XSPICE models): full-wave rectifier example



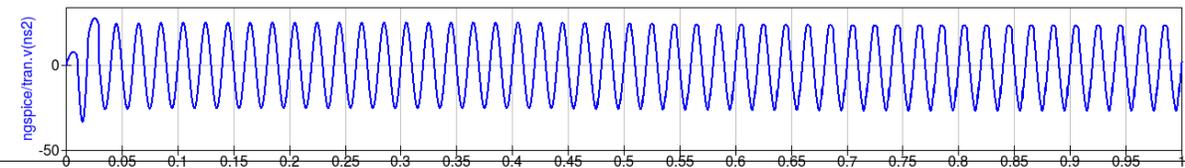
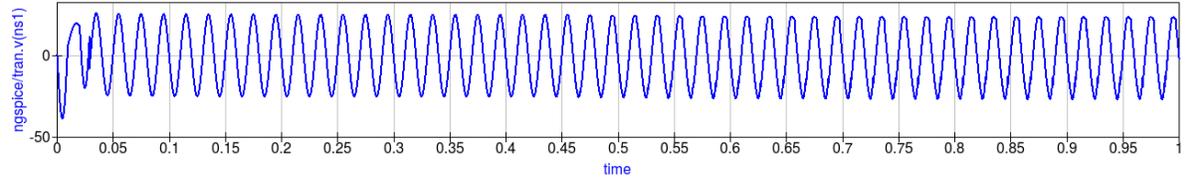
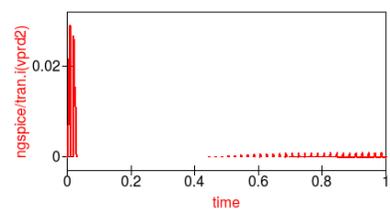
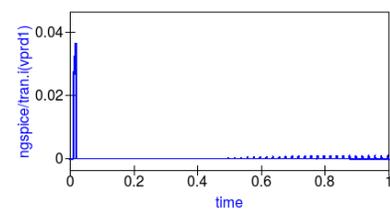
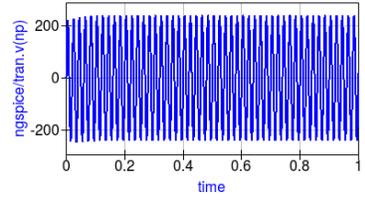
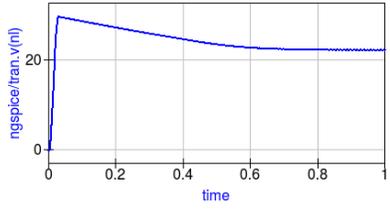
dc simulation

transient simulation

Fourier simulation

TR1
 Type=lin
 Start=0
 Stop=1000 ms
 Order=3
 InitialStep=1e-18
 MinStep=1e-18

FOUR1
 Sim=TR1
 numfreq=10
 F0=50Hz
 Vars=V(ns1) V(ns2) V(Nl) V(np)



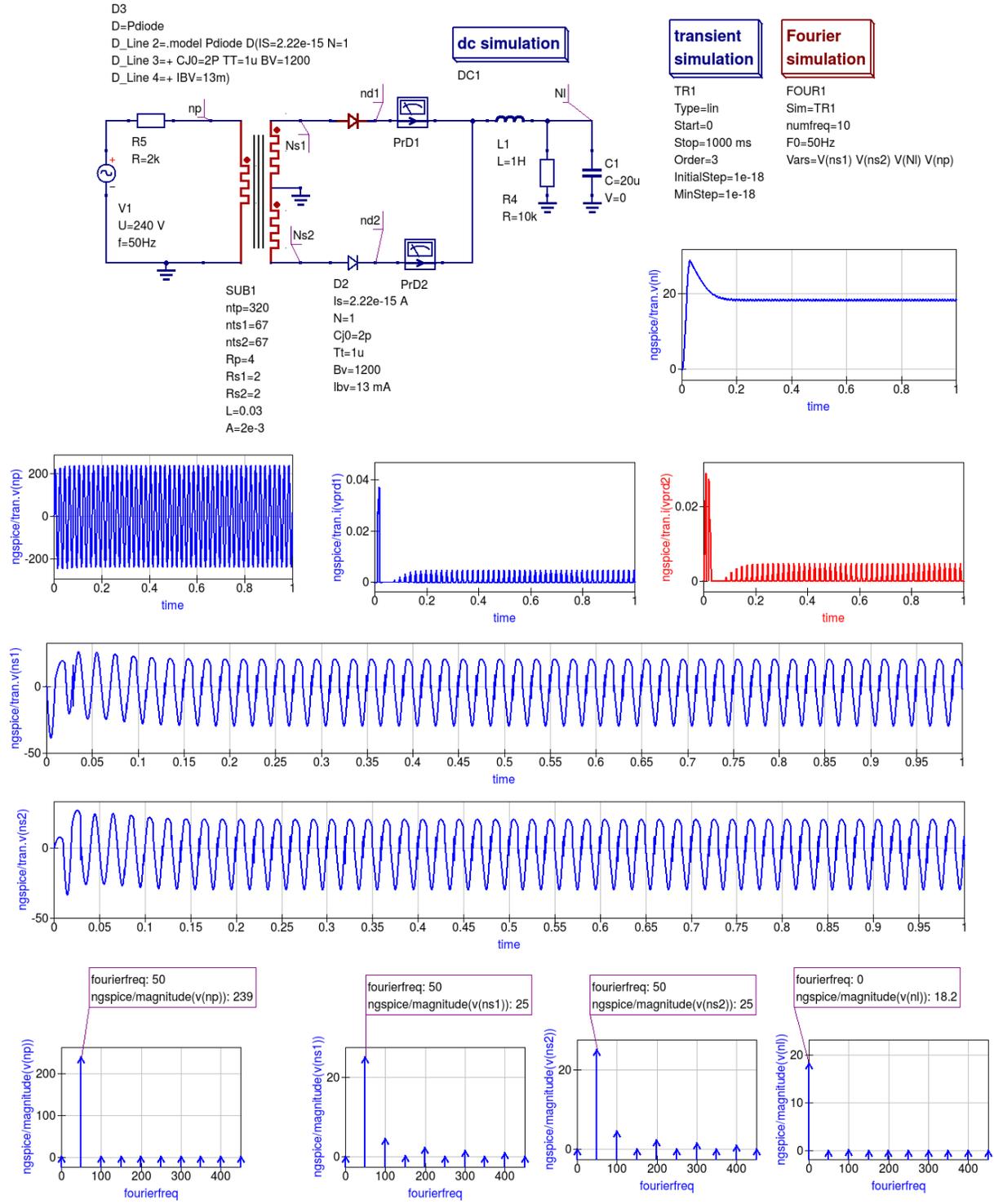
1.7. Chapter 7. Qucs and SPICE simulation models that work with ngspice, Xyce and SPICE OPUS

fourierfreq: 50
ngspice/magnitude(v(np)): 239

fourierfreq: 50
ngspice/magnitude(v(ns1)): 25

fourierfreq: 50
ngspice/magnitude(v(ns2)): 25

fourierfreq: 0
ngspice/magnitude(v(nl)): 22.2



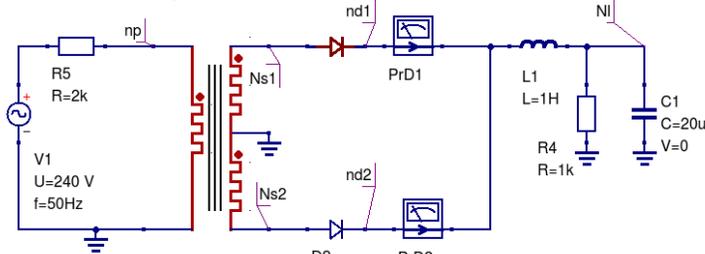
Output load = 10k||20uF

D3
 D=Pdiode
 D_Line 2=.model Pdiode D(IS=2.22e-15 N=1
 D_Line 3=+ Cj0=2P TT=1u BV=1200
 D_Line 4=+ IBV=13m)

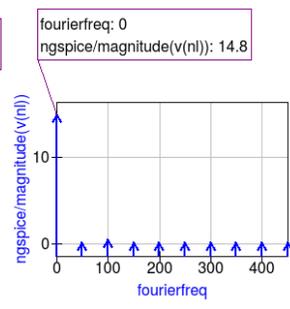
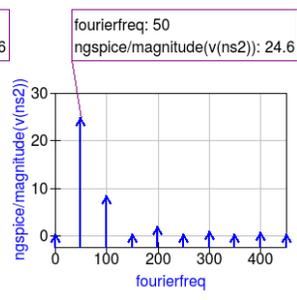
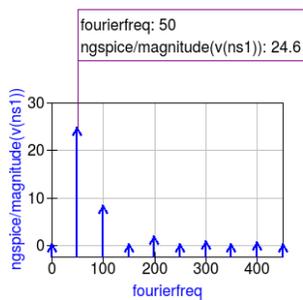
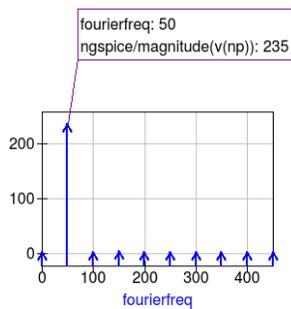
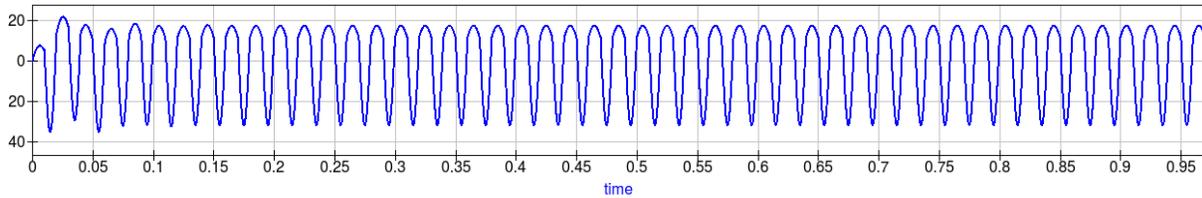
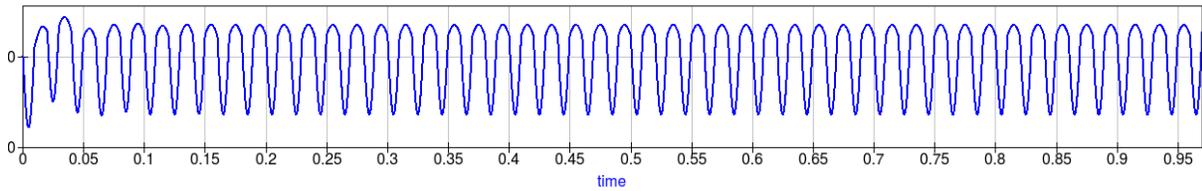
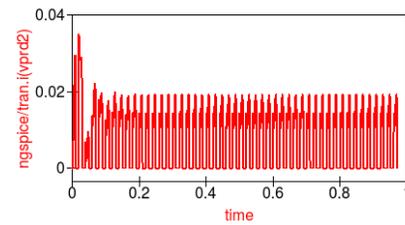
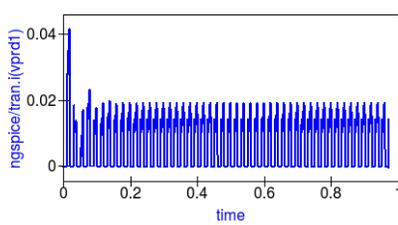
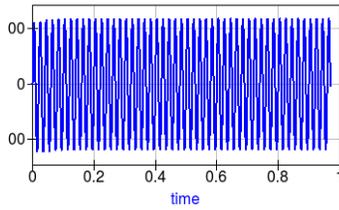
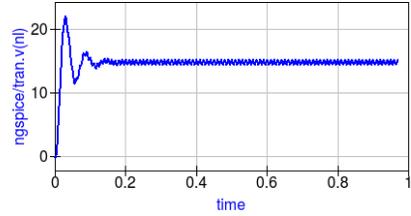
dc simulation

transient simulation

Fourier simulation



DC1
 L1
 L=1H
 R4
 R=1k
 C1
 C=20u
 V=0
 SUB1
 ntp=320
 nts1=67
 nts2=67
 Rp=4
 Rs1=2
 Rs2=2
 L=0.03
 A=2e-3
 D2
 Is=2.22e-15 A
 N=1
 Cj0=2p
 Tt=1u
 Bv=1200
 lbv=13 mA



Output load = 1k||20uF

7.4 More complex circuit simulations that demonstrate the use of spice4qucs models

back to the top

Chapter 8. Ngspice custom simulation technology

8.1 Introduction

8.2 Setting up single and multiple circuit simulations

8.3 Extracting circuit and device properties from ngspice simulations

8.4 Statistical circuit simulation including Monte-Carlo analysis

8.5 Building ngutmeg scripts for circuit simulation control and data analysis

back to the top

Chapter 9. XSPICE standard components and library

back to the top

Chapter 10. XSPICE user written device models and library

back to the top

Chapter 11. Introduction to mixed analogue/digital simulation

11.1 XSPICE basics

11.2 Mixed mode simulation with Ngspice

11.3 Mixed mode simulation with Xyce

11.4 Mixed mode simulation with SPICE OPUS

back to the top

Chapter 12. Verilog-A compact semiconductor device modelling

12.1 Introduction to Verilog-A for compact device modelling

12.2 The Qucs/ADMS Verilog-A “turn key” modelling system

12.3 The Qucs Verilog-A module synthesizer

12.4 Using Xyce for Verilog-A compact device modelling

back to the top

Chapter 13. RF simulation with Ngspice, Xyce and SPICE OPUS

13.1 Introduction to capabilities

The original motivation behind the development of Qucs was the need for an open source RF circuit simulator which was freely available to all interested in RF and microwave circuit and system design. Today, Qucs has become a relatively stable simulation package with good high frequency analysis capabilities like small signal AC two port and multi-port S parameter analysis, noise analysis and rudimentary single tone Harmonic Balance (HB) circuit simulation. For anyone interested in RF circuit design Qucs is distributed with a selection of built-in RF component models, including microstrip and coplanar technology components, making the package a good choice for investigation the performance of high frequency circuits. At RF, Qucs implements models and analysis features not included in the traditional SPICE 2g6 and 3f5 circuit simulators. In contrast to SPICE 3f5 the Ngspice, Xyce and SPICE OPUS GPL simulators have been extended with features which are designed specifically for RF circuit simulation. These include single tone and multi-tone HB simulation (Xyce) and a transient simulation shooting method (SPICE OPUS) for large signal AC steady state simulation. These RF simulation techniques, when coupled with the fact that Ngspice, Xyce and SPICE OPUS support small signal AC two port network analysis via the spice4qucs extension, makes the Qucs-S version of Qucs a useful addition to the GPL RF circuit simulation scene.

HB is a circuit simulation method that solves for the steady state solution of nonlinear circuits in the frequency domain. In HB simulation, the voltages and currents in a nonlinear circuit are represented by truncated Fourier series. HB computes the frequency spectrum of circuit voltages and currents when signals reach a steady state, following excitation with an external signal source. This source can be a large signal AC signal. In practice the HB simulation technique is often more efficient than transient analysis, particularly in situations where transient analysis can take a long time to reach a steady state solution due to widely differing frequency signals present in a circuit, for example amplitude or frequency modulated communications signals. HB is particularly suited to the simulation of analogue RF and microwave circuits.

In this chapter the Qucs-S RF capabilities are introduced and described. To demonstrate these new features a number of example RF circuit simulations are presented together with a new **Template** element which allows libraries of analysis and post-simulation data processing **Nutmeg** scripts to be stored and embedded in Qucs schematics. The idea of a predefined **Test Bench** is also outlined and applied to RF circuit simulation case studies.

The Qucs-S version of Qucs includes spice4qucs extensions which allow the package to be used for analysis of RF circuits. The central features of the spice4qucs RF elements are:

- Small signal AC two port S-parameter simulation (Ngspice, XYCE and SPICE OPUS)
- Small signal AC two port Y,Z etc. network simulation/analysis (Ngspice and SPICE OPUS)
- Single and multi-tone large signal AC Harmonic Balance simulation (Xyce only)
- Large signal AC transient simulation with steady state shooting methods (SPICE OPUS only)

- Emulation of Qucs RFEDD components (limited support at this time)
- A range of lumped RF and microwave components for use in high frequency circuit design (limited but growing support)

Where needed each of the above can make use of Octave scripts and functions in the analysis of simulation data.

Readers will have probably noticed from the list presented above that multi-port S-parameter modelling and RF simulation features are not implemented in Qucs-S. Currently, there are no immediate plans to add this extension to the existing Qucs-S simulation and modelling features. Anyone interested in multi-port S-parameter RF circuit analysis is advised to use the standard Qucs package.

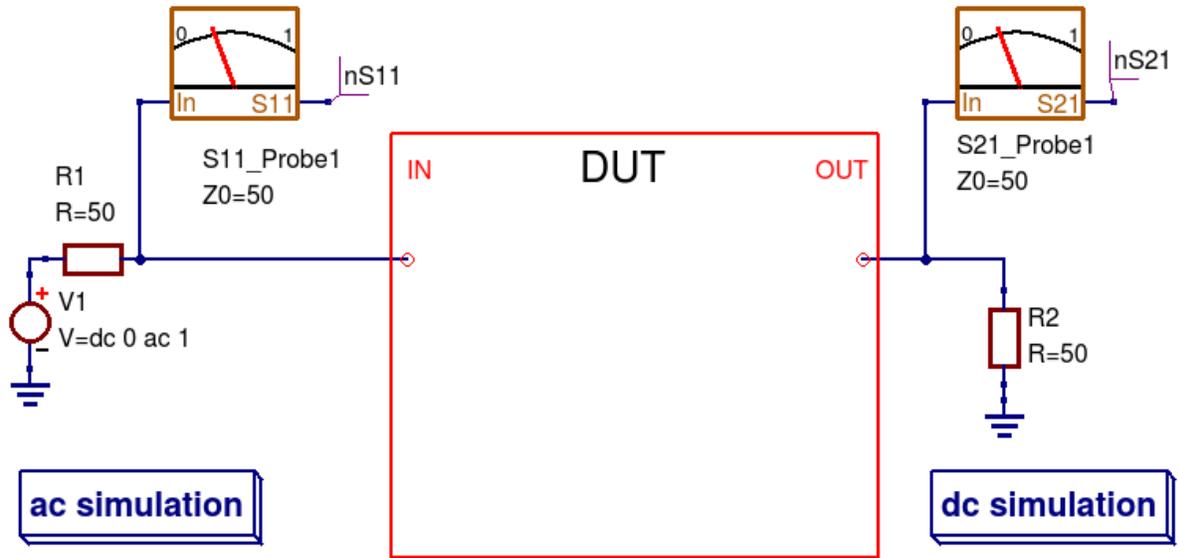
The S-parameter probes, templates and examples introduced in this chapter can be found in the Qucs-S examples directory. They are stored as project `spice4qucs-helpChapter13_prj`.

13.2 Small signal AC S-parameter simulation

S-parameter two port RF and microwave circuit simulation is not implemented in traditional SPICE 2g6 and 3f5 simulators. This is a serious omission because at RF frequencies S-parameter and other two-port network parameters are widely used in circuit analysis and design. To overcome this limitation four small signal AC analysis probes have been added to the `spice4qucs` RF extensions. When combined with signal sources they form a **Two-port S-parameter circuit test bench**. This test bench is shown in Figure 13.1. Its main features are space on the test bench schematic to place the device under test (DUT) circuit diagram, input AC signal sources with Z_0 characteristic impedance, Z_0 load impedances and the S-parameter measurement probes. Notice that two copies of the DUT are required; firstly to measure S_{11} and S_{12} and secondly S_{22} and S_{21} . The test bench also includes a detailed set of instructions on how to use it to measure simulated two-port S-parameters. The two-port S-parameter test bench illustrated in Figure 13.1 will work with the Ngspace, Xyce and SPICE OPUS circuit simulators.

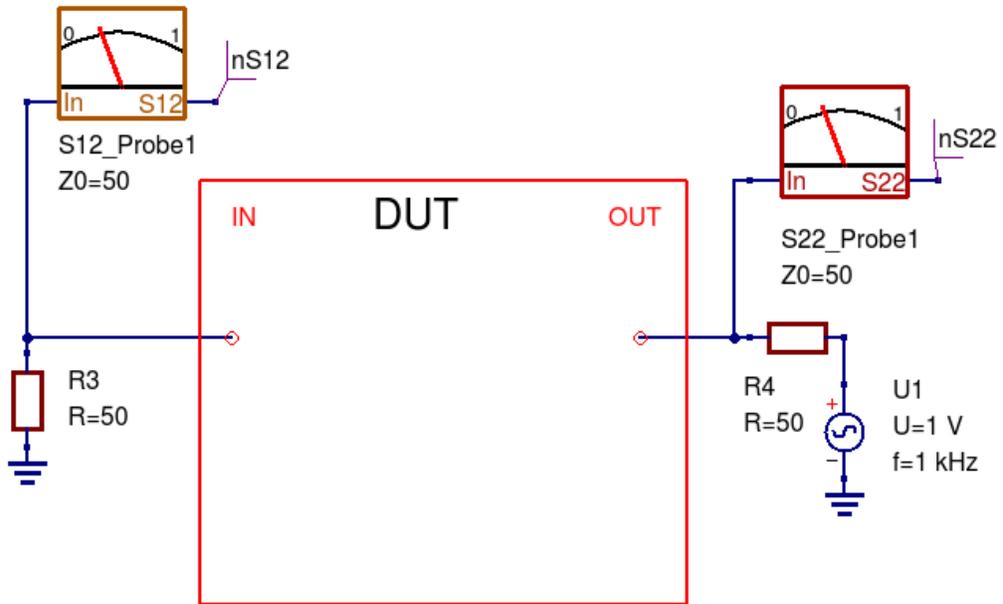
Figure 13.1 A small signal AC S-parameter test bench with S-parameter probes.

The schematic shown in Figure 13.2 demonstrates the use of the S-parameter test bench. In this example two identical copies of a Butterworth passive low pass filter circuit with $f_c = 100$ kHz and $Z_0 = 50$ Ohms are connected between the pairs of DUT terminals labelled **IN** and **OUT**. Notice that the orientation of both DUT is the same. The outputs from the S-parameter probes are called `nS11`, `nS12`, `nS21` and `S22` being represented as voltages specified by complex numbers. Figure 13.2 also shows typical plots of the magnitudes of the simulated S-parameters. For convenience the test-bench instructions have been deleted from Figure 13.3. Also, if required the size of the area allocated to each DUT can be changed, provided the test-bench signal and load circuit connections are not changed. Similarly, the value of Z_0 and the source and load resistors (R_1 , R_2 , R_3 and R_4 in Figure 13.2) can be changed from 50 Ohms.



AC1
 Type=log
 Start=1
 Stop=1e6
 Points=121

S11 and S21 Test bench



S12 and S22 Test bench

A test bench for simulating circuit two port small signal parameters as a function of frequency.

 Copy the four S parameter probes to your current project directory.
 Copy the test bench circuit onto an empty Qucs-S schematic sheet in your current project directory.

Place copies of the circuit under test in the box marked DUT. Make sure that the test circuit input and output terminals are connected to the test bench IN and OUT terminals. Note both

Select one of the Ngspice, Xyce or SPICE OPUS circuit simulators. Set the value of Z0 and resistors R1, R2, R3 and R4 to the required

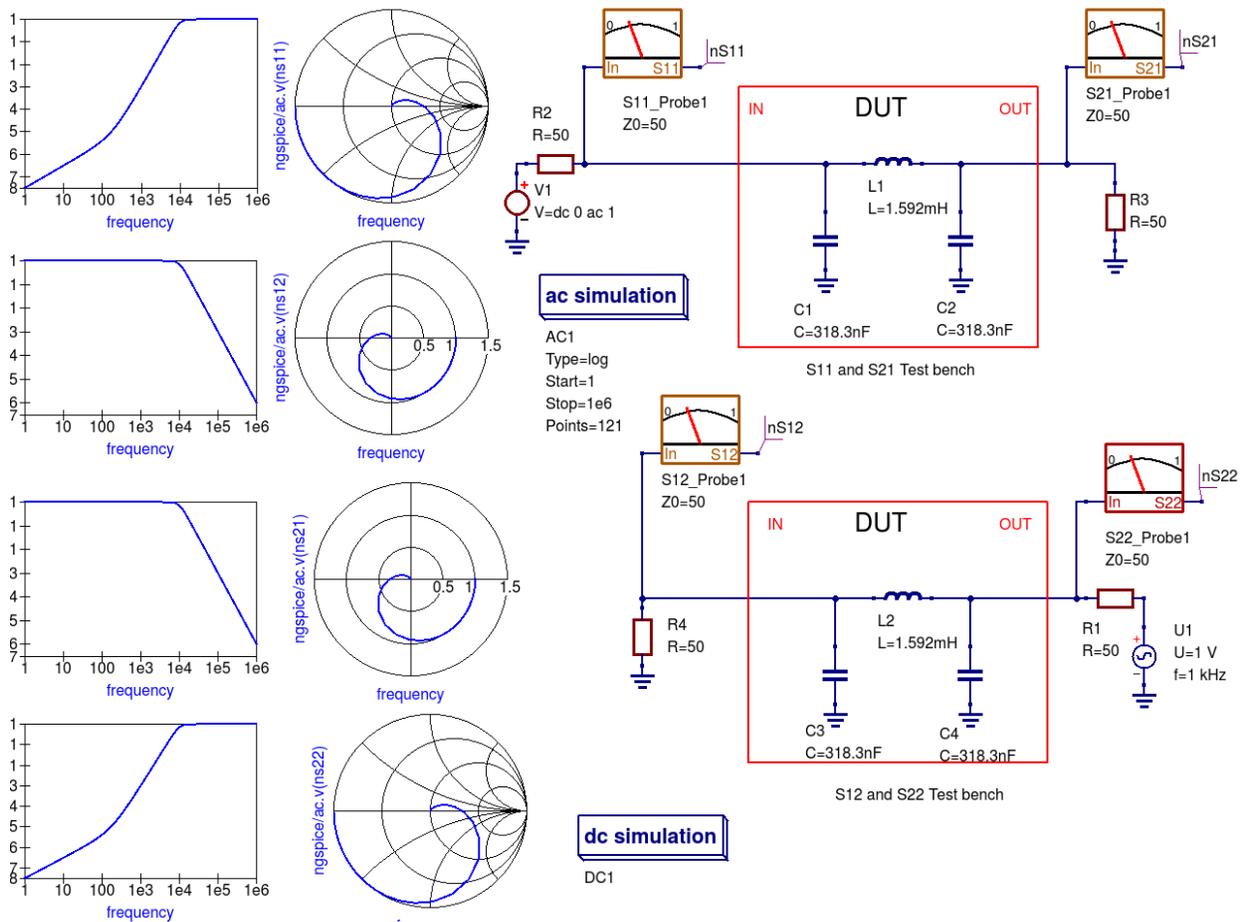


Figure 13.2 Extraction of a low pass filter small signal S-Parameters using a Qucs-S test-bench.

13.3 Small signal AC two port network simulation/analysis

The spice4qucs extensions introduce post-simulation data conversion for two-port networks centred around Qucs-S **Nutmeg scripts** and Qucs-S **Nutmeg equation** blocks. These are designed specifically to work with Ngspice and Xyce. Conversion of two port parameters from one format to another format is simply one example of the application of Qucs-S embedded nutmeg scripts for the control of circuit simulation and the extraction of circuit parameters from output data. The Xyce circuit simulator is more limited in that it does not have a post-processing scripting language for extracting transfer function parameters and other data from simulation output. However, it does allow SPICE style AC .PRINT statements which can include equations provided these are written in a form constructed from the real and imaginary components of circuit voltages and currents. In practice this is not very convenient, particularly when these equations involve many algebraic terms. At this time the Xyce facilities for the extraction of AC data items are at a rudimentary stage in the packages development and for this reason are not considered further in this document.

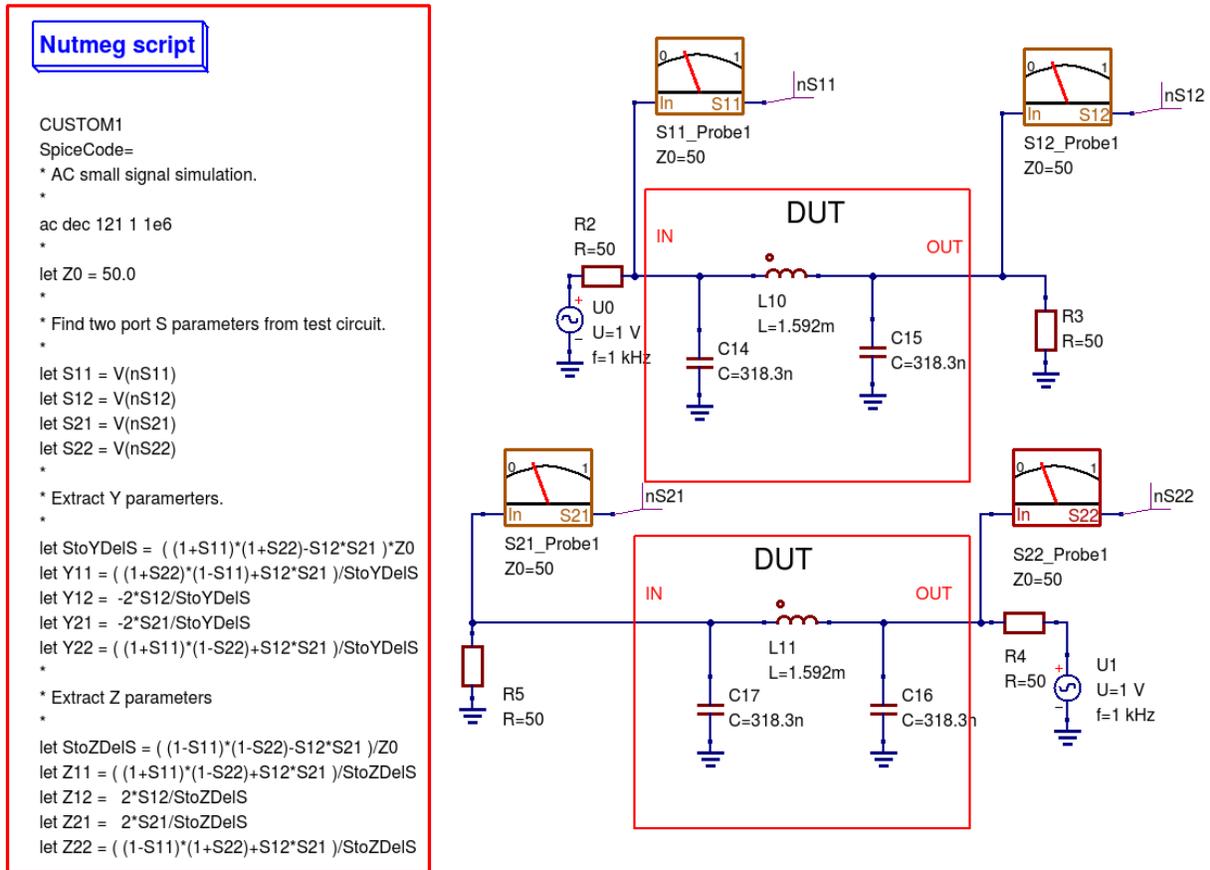


Figure 13.3 **Nutmeg script** controlled simulation and two-port parameter extraction.

Figure 13.3 presents an S-parameter test bench used to extract the S parameters of the same low pass filter introduced in Figure 13.1. However, unlike Figure 13.1 Qucs/Qucs-S simulation icons are NOT attached to the test-bench circuit schematic. Instead a Qucs-S **Nutmeg script** is used. This script controls the simulation sequence and provides post processing algebraic equations which generate small signal AC *Y* and *Z* parameters from the data output by the S-parameter probes. Figure 13.4 shows a set of simulation plots obtained with the Nutmeg script and SPICE OPUS. Identical data was recorded with Ngspice. However, one difference was noticed when simulating circuits via the **Nutmeg script** route. SPICE OPUS requires that the code words, like for example *ac* and *let*, must be entered with lower case letters, otherwise the SPICE OPUS simulation fails. Chapter 8 presents much more detail on how to set up **Nutmeg scripts** and gives a number of additional examples of their use in Qucs-S circuit simulation.

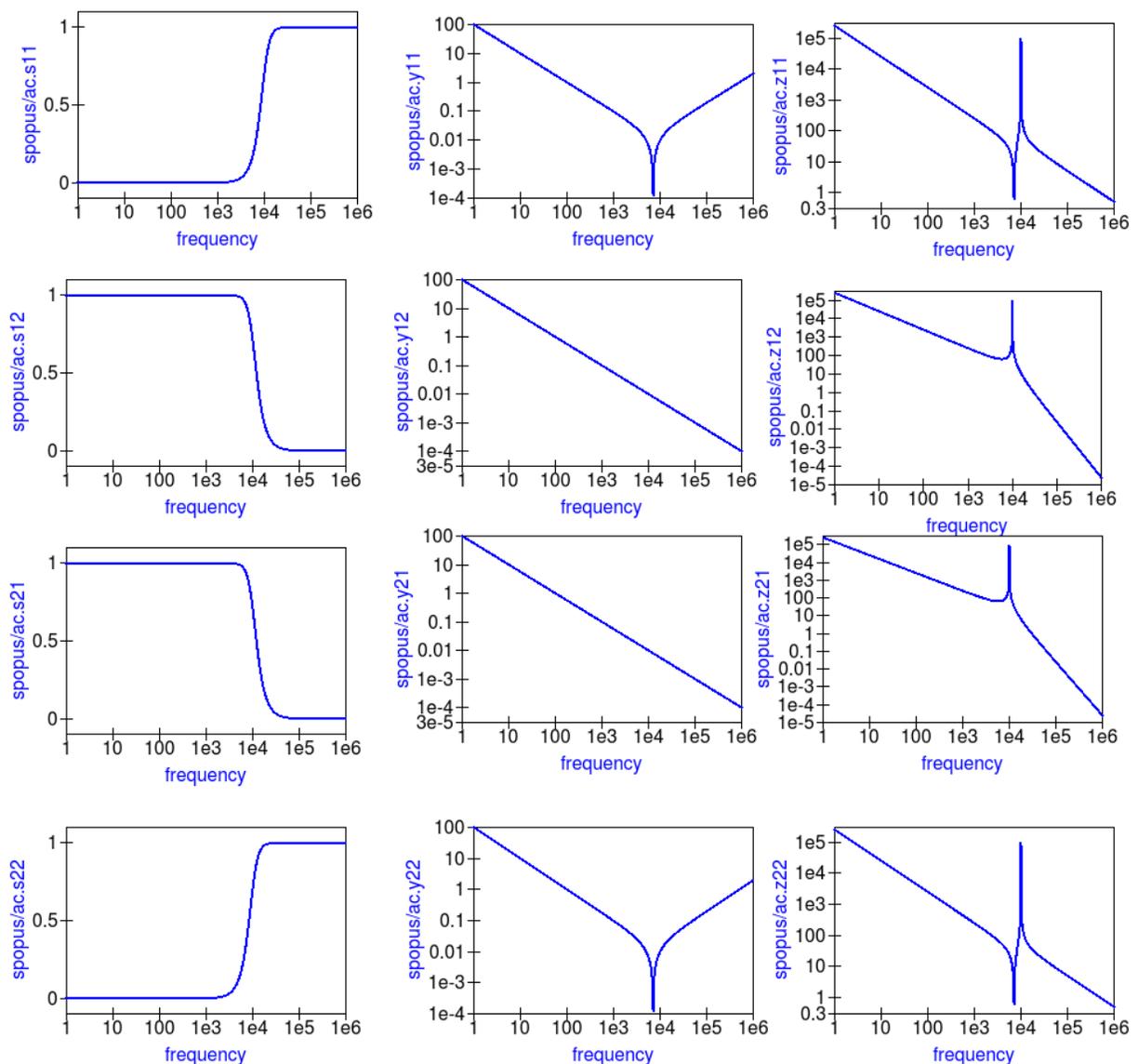


Figure 13.4 Typical S-parameter, Y parameter and Z parameter data for the test circuit given in Figure 13.3.

One of the pioneering circuit simulation features implemented by Qucs is the **Equation** block. This allows blocks of algebraic equations to be attached to a circuit schematic. Any equations which do NOT include quantities computed during simulation, like circuit voltages and currents, are evaluated prior to the start of simulation. These quantities remain fixed during simulation and may be referenced by the simulator when calculating voltages and currents. In contrast, if an **Equation** block includes variables which are functions of simulation variables these are evaluated, based on the stored simulation output data, after a simulation finishes. Qucs has a Octave style numerical analysis package built into the software for this purpose.

Qucs-S uses a slightly different approach to post simulation data processing. Both Ngspice and Xyce use an extended form of the SPICE nutmeg software for post-simulation data processing. Unfortunately, because Xyce does not include a feature equivalent to SPICE nutmeg, AC post-simulation data processing is not possible with Xyce. To setup and use a Qucs-S **Nutmeg** equation block place the **Nutmeg** icon on an empty schematic sheet and enter the individual variable equations in a similar fashion to Qucs **Equation** blocks. Once complete copy the **Nutmeg** equation block to the current work circuit schematic. Such **Nutmeg** equation blocks are called **Templates** by the Qucs-S Development Team. These templates can be saved in a project and used over and over again. Templates add a new and important

facility to Qucs-S which allows users to develop libraries of post simulation data processing scripts and store them for future use. See Chapters 7 and 8 for more details and examples of the use of Qucs-S **Custom Simulation** technology and **Nutmeg** equation blocks. Figure 13.5 shows Qucs-S **Templates** for the conversion of S -parameters to Y and Z two port parameters. Note that these do NOT include commands for simulations, for example `ac`, and do not have the same named variable defined more than ONCE.

```

Nutmeg
StoY_Conversion_Template1
Simulation=ac
StoY_Z0=50.0
StoY_DelSxZ0=( (1+V(nS11))*(1+V(nS22))-V(nS12)*V(nS21) )/StoY_Z0
Y11=( (1+V(nS22))*(1-V(nS11))+V(nS12)*V(nS21) )/StoY_DelSxZ0
Y12=-2*V(nS12)/StoY_DelSxZ0
Y21=-2*V(nS21)/StoY_DelSxZ0
Y22=((1+V(nS11))*(1-V(nS22))+V(nS12)*V(nS21))/StoY_DelSxZ0

```

```

Nutmeg
StoZ_Conversion_Template1
Simulation=ac
StoZ_Z0=50.0
StoZ_DelSxZ0=( (1-V(nS11))*(1-V(nS22))-V(nS12)*V(nS21) )/StoZ_Z0
Z11=( (1+V(nS11))*(1-V(nS22))+V(nS12)*V(nS21) )/StoZ_DelSxZ0
Z12=2*V(nS12)/StoZ_DelSxZ0
Z21=2*V(nS21)/StoZ_DelSxZ0
Z22=((1-V(nS11))*(1+V(nS22))+V(nS12)*V(nS21))/StoZ_DelSxZ0

```

Figure 13.5 Qucs-S **Nutmeg** equation block templates for S to Y and Z parameter conversion.

13.4 Single tone large signal AC Harmonic Balance simulation

The Spice4qucs subsystem supports Xyce single tone and multi-tone Harmonic Balance (HB). Unlike the rudimentary version of HB simulation implemented in Qucs the Xyce version can simulate circuits with a full range of SPICE components. It is also faster and much more stable. In general no changes to the SPICE semiconductor device or component models are required. To invoke single tone HB just place the Qucs-S HB simulation icon on a circuit schematic, define the number of harmonics and simulate the circuit with Xyce. The spice4qucs output data parser automatically converts output variable names to Qucs notation. For example, for node voltage `out plot out.Vb`.

Figure 13.6 shows the schematic and Figure 13.7 the simulation output plots for a basic diode circuit similar to the original Qucs HB example found on the Qucs web site. For comparison Figure 13.7 presents the output voltage spectrum plots generated by Qucs and Qucs-S/Xyce.

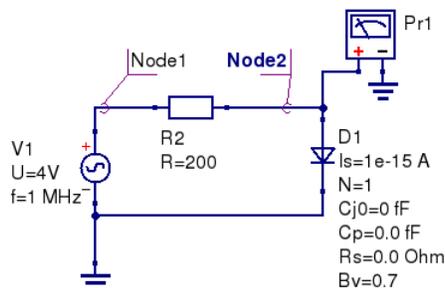


Figure 13.6 Diode clipper harmonic balance simulation.

The HB simulation results for the diode clipper circuit are shown in the Figure 13.7.

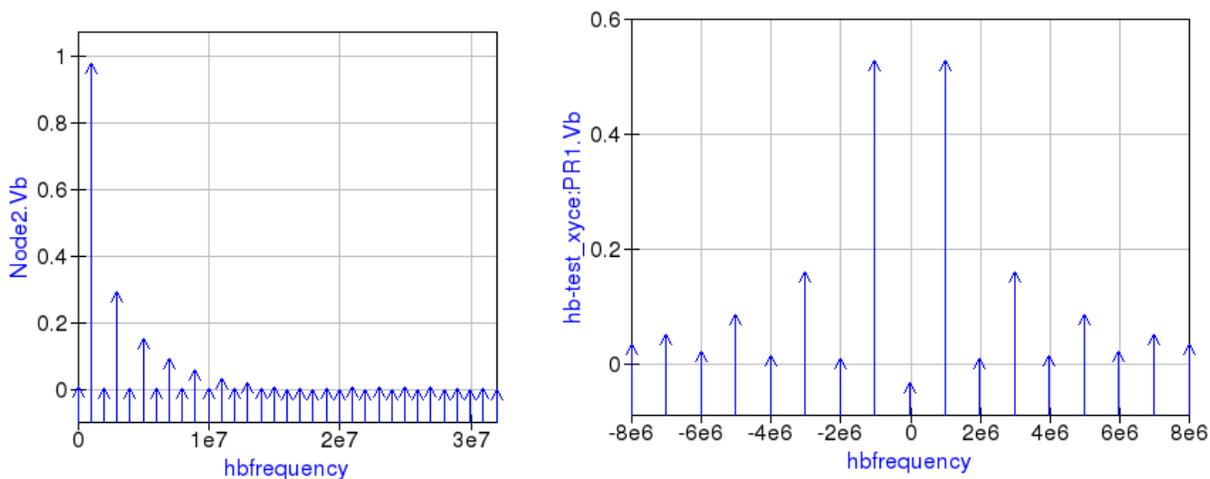


Figure 13.7 Output voltage spectrum at Node2 for Qucs (left plot), and measured with voltage probe Pr1 for Xyce (right plot).

Comparing these two plots highlights an obvious difference in the plot frequency scales. The Qucs-S/Xyce output plot is represented as a function of negative and positive frequency components. In this example there are eight harmonics ($n=8$) arranged as 8 positive frequencies and eight negative frequencies plus a DC component.

Qucs HB simulation data are output as a plot of frequency domain spectral amplitude components $|H|$, where

$$|H| = U(0), U(f_1), U(f_2), U(f_3), \dots$$

$U(0)$ is the DC spectral component, $U(f_n)$ is the magnitude of a harmonic component at frequency f_n and $n = 1, 2, 3, 4, \dots$. In contrast to Qucs, Xyce outputs HB voltage and current simulation data as plots of complex conjugate spectral components, where

$$|H| = U(0), 2 \cdot \sqrt{U(-f_1) \cdot \overline{U(f_1)}}, 2 \cdot \sqrt{U(-f_2) \cdot \overline{U(f_2)}}, \dots$$

yielding, eight very similar magnitude harmonic spectra values to Qucs $|H|$. If required the phase at each harmonic frequency can be extracted from the Xyce HB simulation data.

13.5 Multi-tone Large signal AC HB simulation

Since Xyce release 6.3 the package has supported multi-tone HB simulation. Xyce multi-tone allows more than one tone frequency in the `HB simulation` component properties box. Perform the following steps to setup a multi-

tone Xyce HB simulation:

- Specify a list of space separated frequencies in the f parameter box.
- Specify a comma separated list of the number of harmonic frequencies for each of the source signals in the n parameter box.
- Construct an input signal generator using two or more series AC voltage sources, with the required frequencies and amplitudes, or
- construct an input signal generator using two or more parallel AC current sources driving a one Ohm resistor.

Normally, multi-tone HB simulation signal sources consist of two or three AC sources with different frequencies and similar amplitudes. With two AC signal sources with nearly equal frequencies, that are not integer related, circuit modulation components can be extracted from circuit output spectra. A multi-tone HB example illustrating this feature is given in Figure 13.8, where two AC signals of 0.8 V peak and frequencies 0.95 MHz and 1.05 MHz are applied to a simple diode circuit. The frequencies of individual diode current spectral components are show as combinations of signal frequencies f_1 and f_2 and marked in red on Figure 13.9.

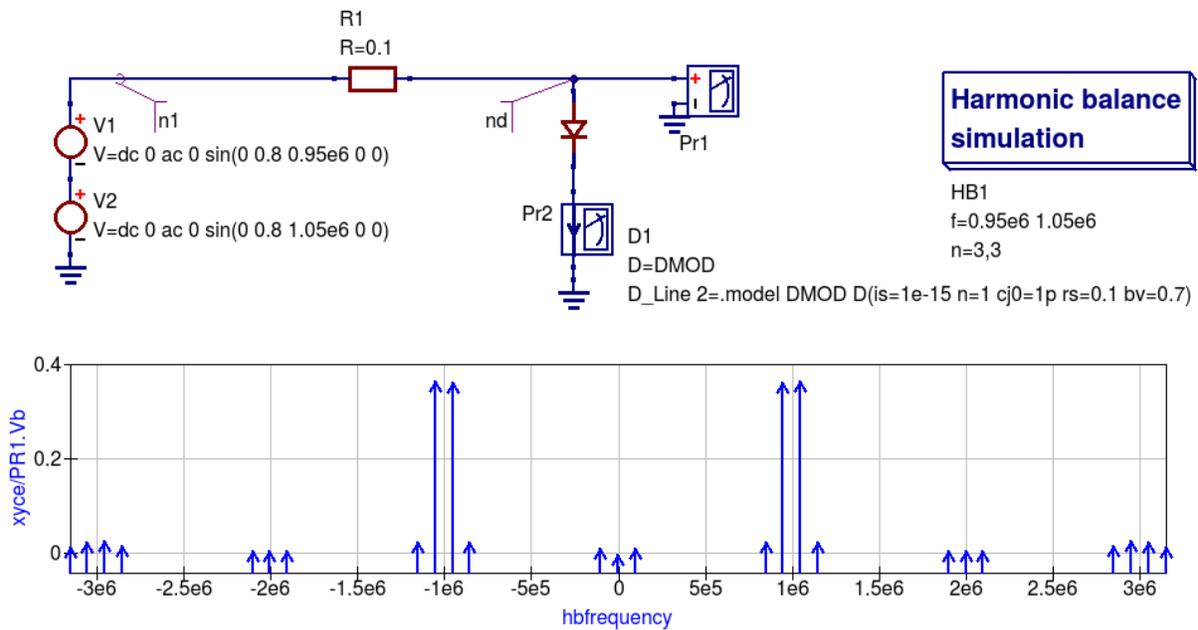


Figure 13.8 An example diode 2-tone Xyce HB simulation circuit plus diode voltage spectra.

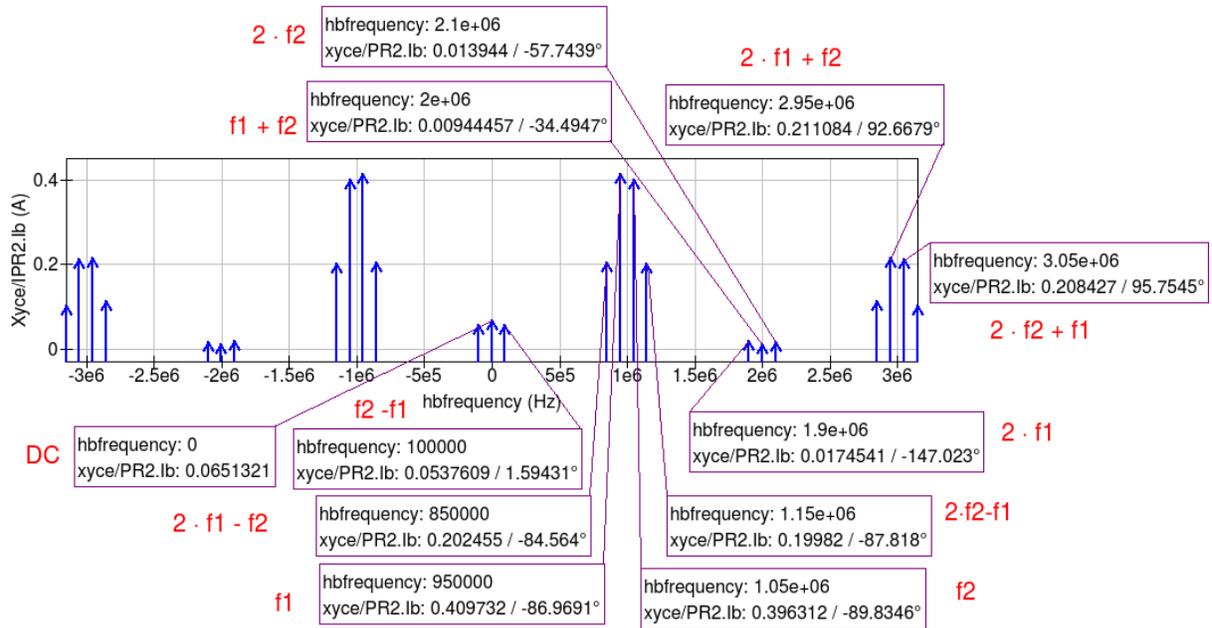


Figure 13.9 Diode 2-tone Xyce HB simulation current spectra.

13.6 The SPICE OPUS large signal AC steady state transient shooting method

Fourier analysis of circuits driven by periodic AC input signals works well at low to moderate frequencies provided that they have a period similar to the circuit time constants. Unfortunately, high frequency RF signals normally have very small periods, implying that an RF transient simulation has to run for a very large number of signal cycles before a steady state circuit response is reached. This can result in a prohibitively long simulation time which can grow at an alarming rate as the circuit size is increased. One way to reduce simulation time is to undertake the simulation of RF communication circuits in the AC domain. This approach forms the basis for the single and multi-tone Harmonic Balance methods introduced in previous sections of this chapter.

A second method, which is particularly suited to simulating RF communication circuits, is the so called “shooting method”. This is a modified form of time domain transient simulation. In the shooting method it is assumed that a non-linear circuit has a periodic solution which can be found from the circuit state where transients are NOT present. This state is called a steady state circuit solution. If $x(t)$ is a set of circuit variables obtained by time domain simulation at time t , then for periodicity $x(t) = x(t + T)$, where T is the period of the input signal. The time domain simulation starts by calculating the initial state $x(0)$, often using DC simulation when the input signal is zero. Using $x(0)$ as an initial state, a circuit under test is simulated in the time domain from $t = 0$ to $t = T$ than an estimate of the circuit state is made. This process is repeated, increasing time by T at each iteration, until $x(t + n \cdot T) = x(t + (n + 1) \cdot T)$ is satisfied within a reasonable tolerance. Unlike direct transient methods a circuit is only simulated over one period per solution iteration cycle. Hence, the shooting method can be more more efficient, provided that a steady state solution can be found in a number of iterations that are smaller than the number of periods simulated by direct transient simulation.

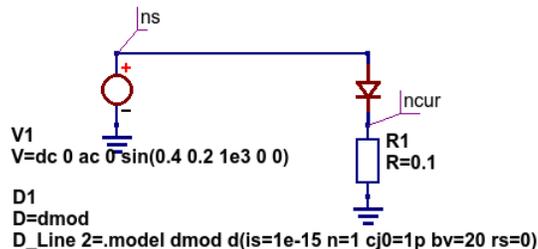
The SPICE OPUS implementation of the shooting method was first released with software version 2.25 in December 2006. It can be used to simulate the performance of linear and non-linear circuits with either small or large amplitude periodic input signals. It can be launched by Qucs-S using the new **Custom simulation** feature. SPICE OPUS steady response analysis in the transient domain is implemented as an additional nutmeg command called **ssse**. Nutmeg command **ssse** runs a time domain shooting method with extrapolation via the following statement:

```
ssse v([,]) [level] [step] [skip] [period] [history]
```

where $v([,])$ indicates the observed response of a voltage at a circuit node, referenced to ground, or a voltage difference

between two nodes, for example $v(n1)$ or $v(n1,n2)$ respectively; level indicates the circuit hierarchical at which the shooting method data is calculated, level=0 is a circuit expanded to component level, default=0; step is the time step for transient simulation (same meaning as the nutmeg tran command), default=1; skip is the time skipped before the shooting method starts sampling response $v([,])$, default=0; period is the number of periods taken into account for sampling, default=2; history is a flag which if set causes nutmeg to record data from all transient iterations. If history is NOT set then only the final steady state solution is recorded. In the above SPICE OPUS nutmeg `ssse` statement the brackets [] indicate optional quantities. Also note that SPICE OPUS option `sssetol` can be changed, if required, to improve simulation convergence.

Figure 13.20 introduces a simple test circuit designed to test the performance of a DC forward biased semiconductor diode subjected to an AC input voltage signal. The SPICE OPUS nutmeg script is shown in Figure 13.20 attached to a Qucs-S **Nutmeg script** icon. This script follows the statement rules required by the SPICE OPUS extended form of SPICE nutmeg. For comparison the example script shown in Figure 13.20 and Figure 13.21 includes entries for launching and saving the simulation data from transient, Fourier and `ssse` simulations. Notice that each set of simulated data is written to separate named files. The names of these files are registered by pressing the “Find all outputs” tab on the Qucs-S Custom simulation control script editing window, see Figure 13.21. Variables for post-simulation visualization can be found in a similar way by pressing the “Find all variables” tab. More details of the use of Qucs-S Custom simulation can be found in Chapter 8. Pressing key “F2” causes Qucs-S to simulate the current circuit schematic; firstly generating a Qucs circuit netlist, secondly synthesizing a SPICE style netlist from the Qucs netlist (Figure 13.22 shows the SPICE OPUS netlist generated by Qucs-S for the diode test circuit Custom simulation), and finally simulates the circuit netlist using the nutmeg statements located between the SPICE `.control` and `.endc` statements. Following successful simulation Qucs-S visualization features can be used to plot the transient and frequency domain data output. A typical set of plots is illustrated in Figure 13.23. Notice that the Fourier and `ssse` spectral data for the diode current are identical.



Nutmeg script

```

CUSTOM1
SpiceCode=

tran 1e-7 10m
let Id = v(ncur)/0.1
write testDiodeSMtran1.txt v(ns) Id
display

set nfreqs = 10
fourier 1k v(ns) v(ncur)
spec 0 10e3 1e3 v(ns) v(ncur)
let Idf = v(ncur)/0.1
write testDiodeSMfourier2.txt v(ns) Idf
display

ssse v(ncur) 0 1e-6 1e-3 20
display

spec 0 20e3 1e3 v(ncur)
let k = abs(v(ncur)/0.1)
write testDiodeSMsp1.txt k
display

```

Figure 13.20 SPICE OPUS shooting method test circuit for a semiconductor diode.

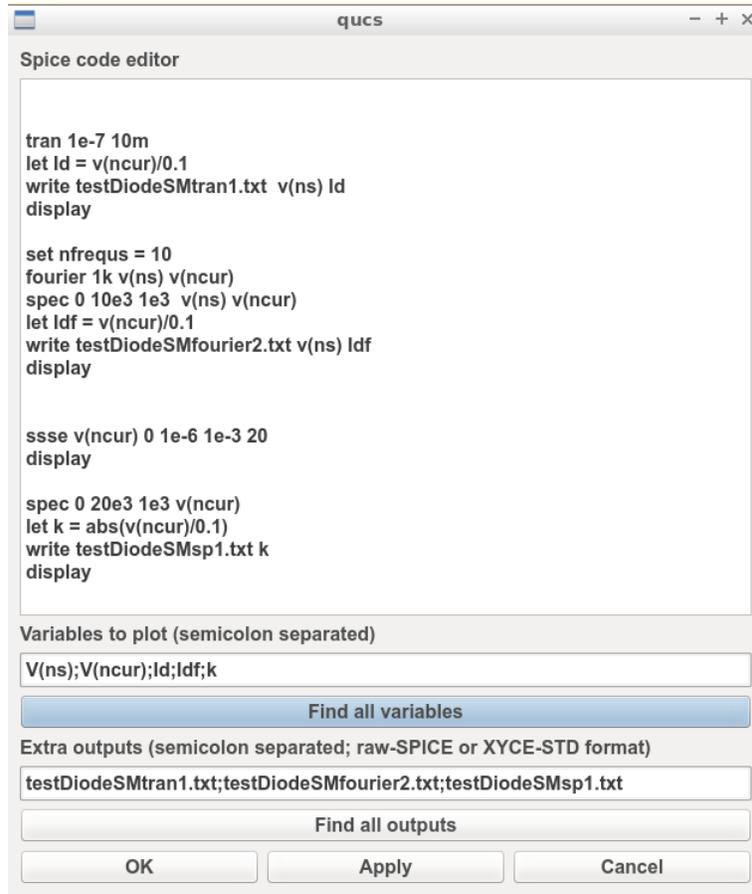


Figure 13.21 Qucs-S **Custom simulation** control script editing window.

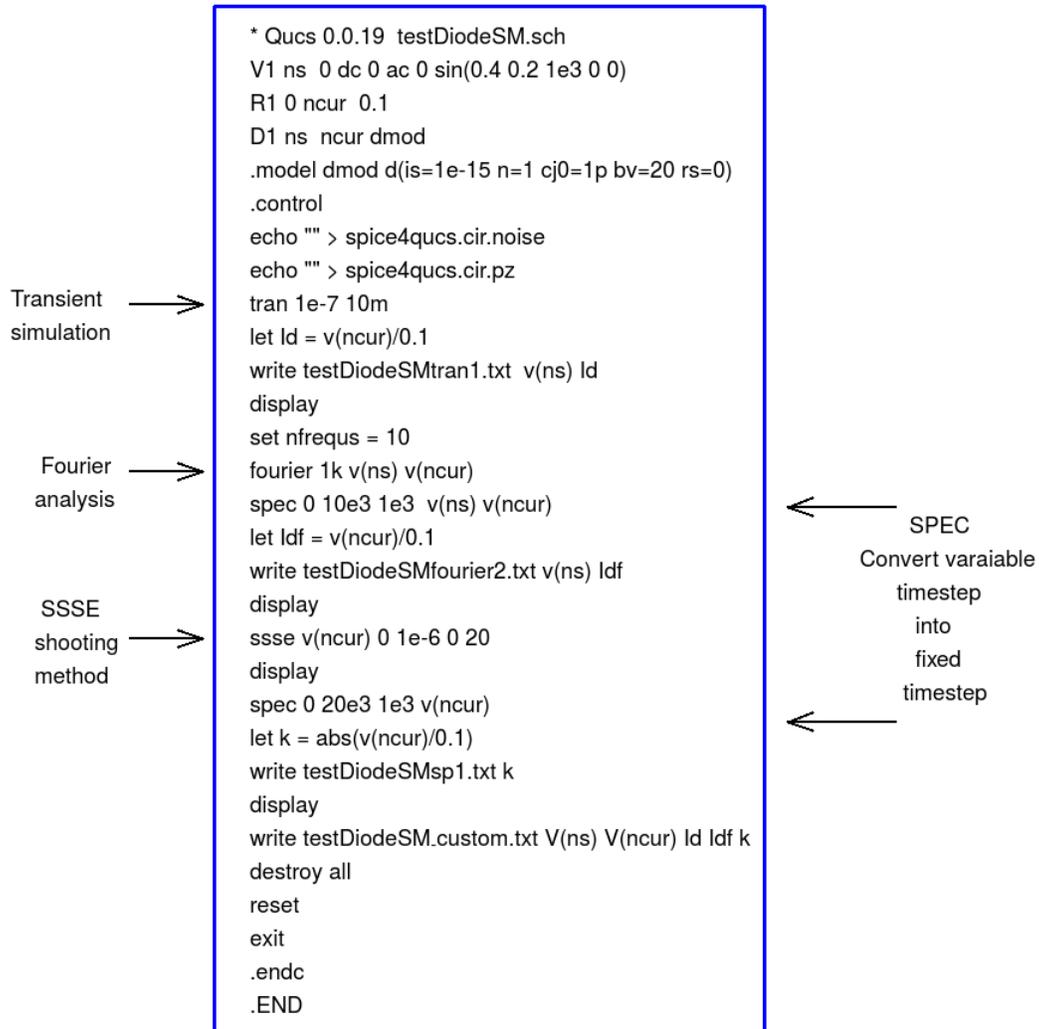


Figure 13.22 SPICE OPUS netlist for semiconductor diode transient, Fourier and ssse simulation.

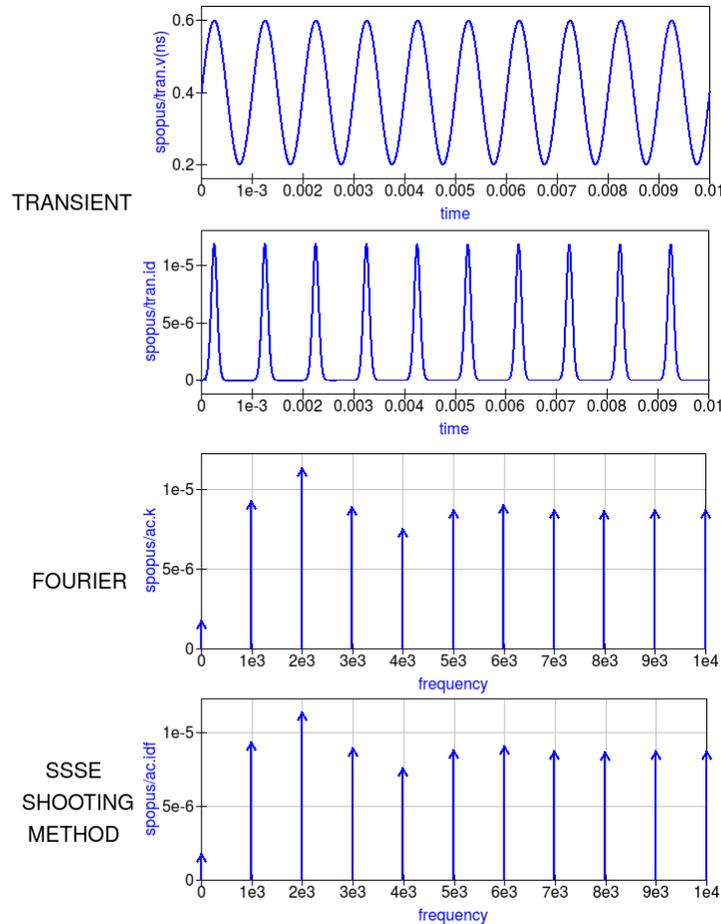


Figure 13.23 Transient, Fourier and `ssse` semiconductor diode current plots in the time and frequency domains.

13.7 Emulation of Qucs RFEDD components

RFEDD passive components (RCL) and B-type sources could be represented using `hertz` variable in equations. See official Ngspice manual for additional information.

13.8 RF device models

Circuit simulators based on SPICE include a range of passive component and active device models. Often the passive R, C and L models have ideal characteristics that only provide correct simulation data at low frequencies. SPICE active device models are the opposite in that they operate correctly over a wide band of signal frequencies, from low frequencies to RF or microwave frequencies. At RF and above it is unusual for active models to include device package parasitics. The models introduced in this section introduce readers to a number of passive models that provide more realistic simulation data at RF and higher frequencies. They provide more accurate simulation data for some of the currently available commercial components while simultaneously introducing readers to the modelling principles needed in RF component and device modelling.

As a starting point the modelling of RF R , C and L is introduced through the development of lumped element models for these components. In this context the term lumped element is taken to mean an electrical equivalent circuit which provides accurate device characteristics up to a signal frequency where the physical size of a component is not greater

than roughly 5% of the signal wavelength. Lumped component models of this form also have the advantage that they can be simulated in the time domain by SPICE based circuit simulators.

13.8.1 RF resistor models

The most common form of resistor used in the construction of circuits mounted on printed circuit boards (PCB) are:

- Carbon composite axial leaded resistors,
- Carbon or metal thin-film axial leaded resistors,
- Wire wound axial leaded resistors, and
- Surface mount chip resistors.

For RF circuits, metal thin-film axial resistors and surface mount chip resistors are the preferred types due to their superior RF performance and straight forward PCB mounting procedures. Cross sectional diagrams for these components and their electrical equivalent circuits are shown in Figure 13.8.1.

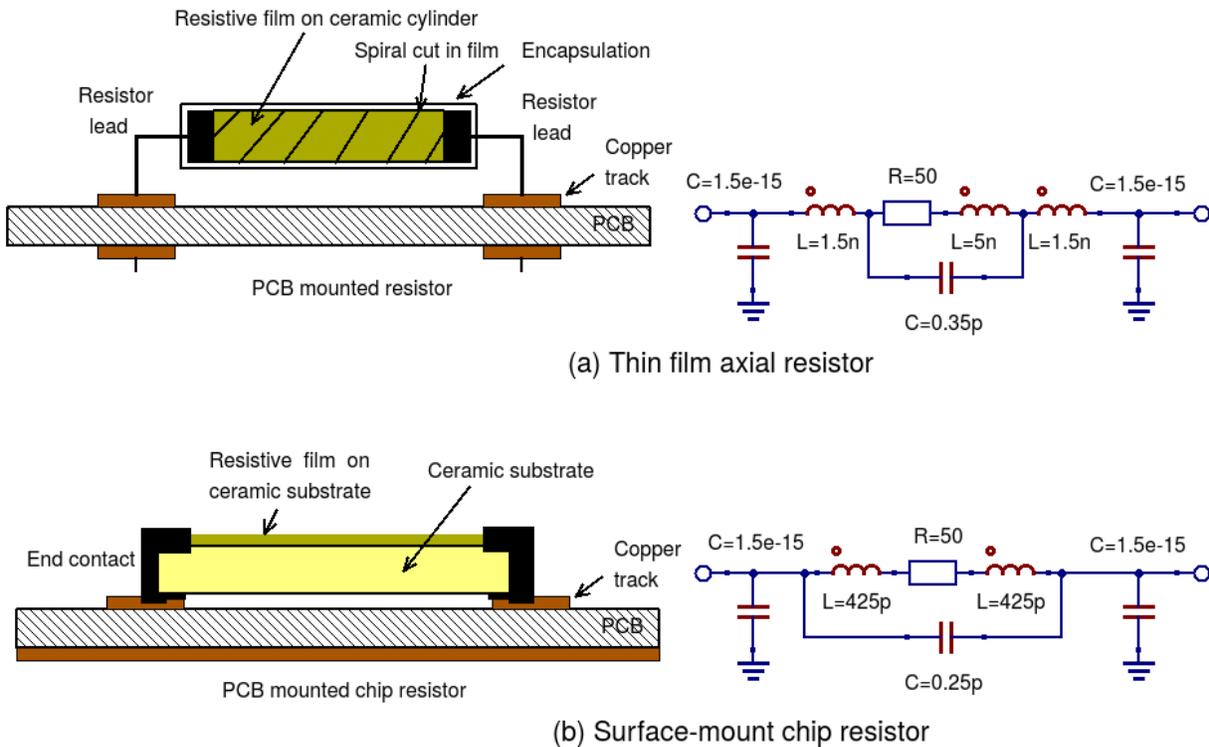


Figure 13.8.1 RF resistor cross sectional diagrams and electrical models for common RF lumped component resistors.

13.8.2 Metal thin-film axial leaded resistors

The values for the R , C , and L components in Figure 13.8.1 are often given by manufacturers, having been determined by measurement of the RF model small signal AC performance. For those components that do not have model parameters listed in their specification approximate values for each of the model parameters can be calculated using the following equations:

1. The inductance of the resistor lead wires are labelled $L1$ and $L2$ in Figure 13.8.2(a) and L is approximated by the equation for a single wire with a circular cross-section

$$L \approx \frac{\mu_0 \cdot Wlength}{2.0 \cdot \pi} \left[\ln \left(\frac{2.0 \cdot Wlength}{Wradius} \right) - 0.75 \right]$$

Here, $Wlength$ and $Wradius$ are the lead length and radius in metres, respectively. As a first approximation the inductance of the thin-film resistor can be estimated from the L associated with a thin strip of resistive material formed by a spiral of the thin-film resistive coating cut from the material deposited on a dielectric cylindrical tube, where

$$L \approx \frac{\mu_0 \cdot Lstrip}{2.0 \cdot \pi} \left[0.5 \cdot \ln \left(\frac{2.0 \cdot Lstrip}{Wstrip} \right) + \frac{Wstrip}{3.0 \cdot Lstrip} \right]$$

Here, $Lstrip$ is the length and $Wstrip$ the width of the thin-film strip, respectively. For the purposes of estimating Ls the thin-film resistor is assumed to be formed from a spiral with a cut traversing the resistor material four times, set by $Ns = 4$, giving a very rough order of magnitude for $Lstrip$ and $Wstrip$ of

$$Lstrip \approx 2.0 \cdot \pi \cdot Bradius \cdot Ns$$

$$Wstrip \approx 0.75 \cdot \frac{Blength}{Ns}$$

Where $Blength$ and $Bradius$ are the physical body length and body radius of the thin-film resistor in metres, respectively. To take account of the thin-film resistor end-caps, that connect the thin-film resistor to the component leads, the body length of the resistor is estimated to be roughly 0.75 times the external component length.

2. The thin-film resistor capacitors Cp and $Cpad$ can also be estimated using the following equations

$$Cp \approx \frac{Erb \cdot \epsilon_0 \cdot \pi \cdot Bradius \cdot Bradius}{0.75 \cdot Blength}$$

$$Cpad \approx \frac{Erp \cdot \epsilon_0 \cdot 1.5 \cdot Blength \cdot Bradius}{H}$$

Where Erb is the relative permeability of the resistor substrate, Erp is the relative permeability of the PCB and H is the distance below the resistor to ground. If H is greater than the PCB thickness it implies that there is no ground plane on the underside of the PCB and as a consequence capacitor Cp becomes very small or goes to zero.

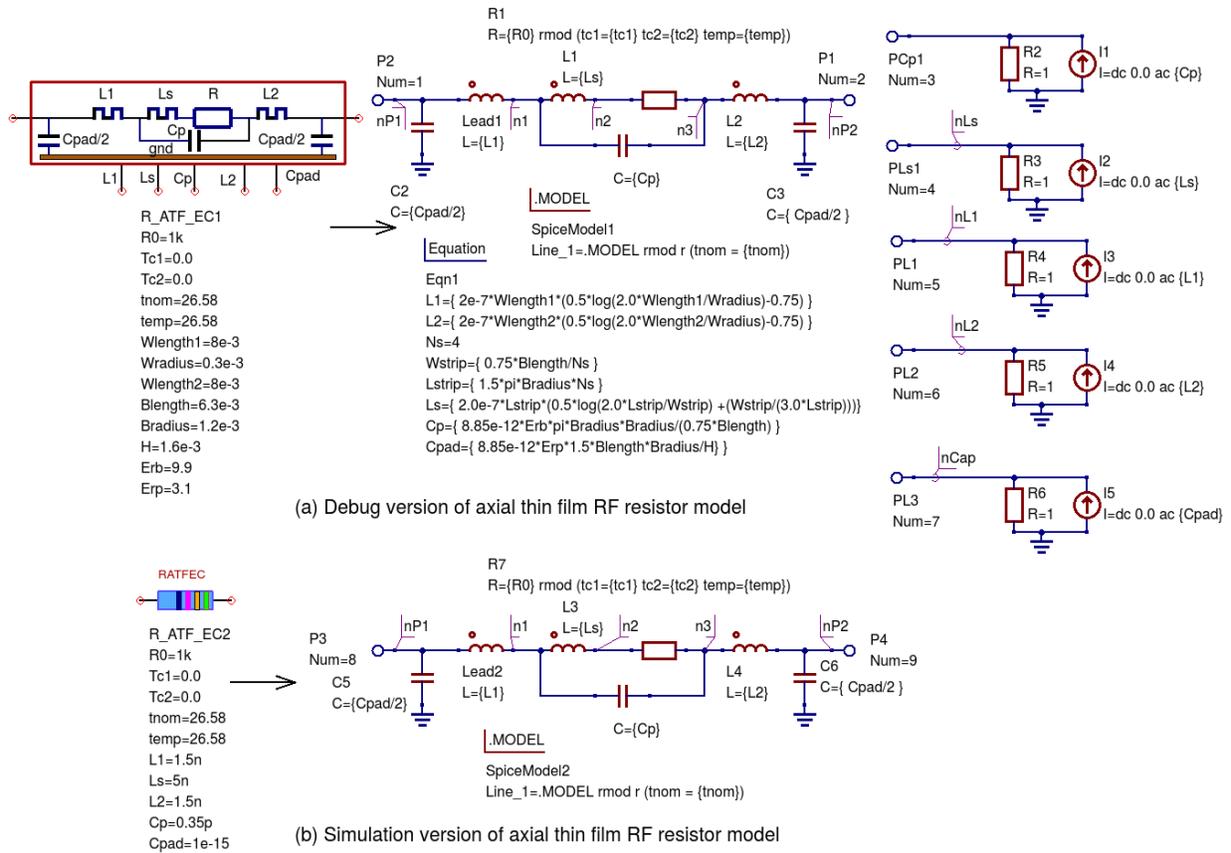


Figure 13.18.2 The RF thin-film axial led resistor model: (a) a debug version which estimates the model parameters from the component physical dimensions and material properties, and (b) a simulation version of the model that has the L , C and R values as parameters.

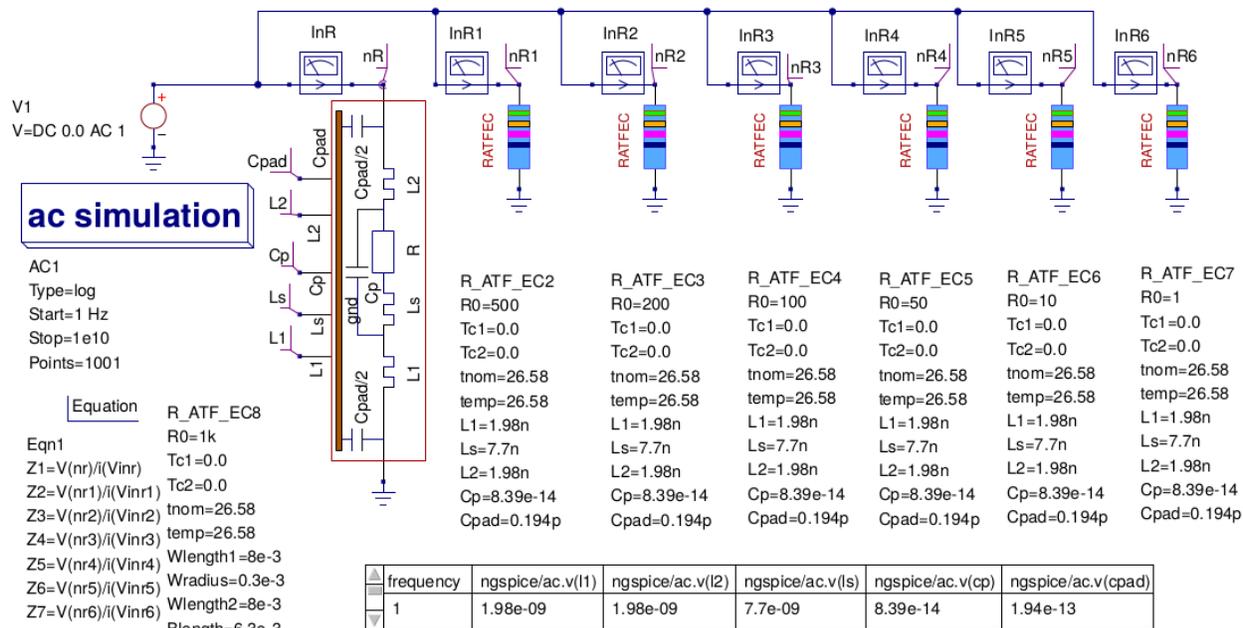


Figure 13.18.3 A basic test bench for simulating the small signal AC performance of a thin-film axial leaded resistor.

In Figure 13.18.3 the Qucs-S schematic symbol for a thin-film resistor shows a resistor colour code on a blue background. Please note that the drawn colour code is just indicative of a typical axial resistor value and is identical for all resistor symbols placed on a schematic. However, also note that the $R0$ parameter must be set to a specific value as required by the circuit under simulation test.

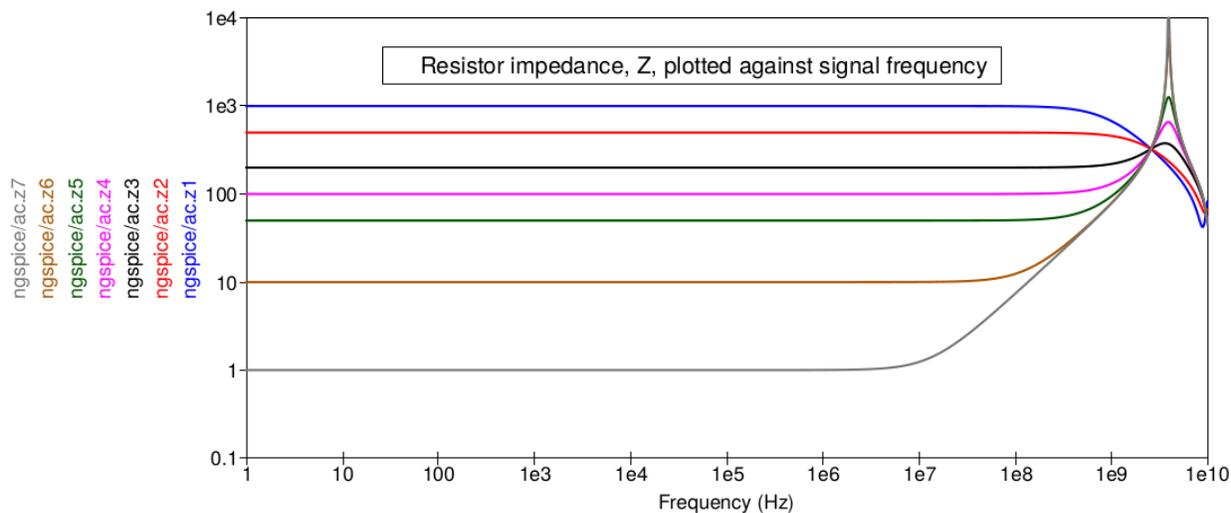


Figure 13.18.4 A set of resistor impedance data generated using the test bench shown in Figure 13.18.3.

Notice that in Figure 13.18.4 a resistor value of around 200 Ω gives the widest AC bandwidth, in this example approaching 1GHz. Hence, it is better to build a 50 Ω RF axial resistor from four parallel 200 Ω components rather than use a single 50 Ω resistor.

13.9 More example RF circuit simulations

back to the top

Chapter 14. Qucs-S/Octave circuit simulation and device parameter extraction interface

back to the top

Chapter 15. References

back to the top

A “Technical Description” of the Qucs simulator and implemented device models are available online at <http://qucs.sourceforge.net/tech/technical.html>.

Spice4qucs example schematics can be found in the spice4qucs source code “examples” directory.

Unofficial Qucs build with spice4qucs features enabled called release candidate 6 (rc6) can be downloaded from <https://github.com/ra3xdh/qucs/releases/tag/0.0.19S-rc6>.