Qucs Help Documentation

Release 0.0.19

Qucs Team

Nov 05, 2017

Contents

Contents:

CHAPTER 1

Background

The 'Quite universal circuit simulator' Qucs (pronounced: kju:ks) is an open source circuit simulator developed by a group of engineers, scientists and mathematicians under the GNU General Public License (GPL). Qucs is the brain-child of German Engineers Michael Margraf and Stefan Jahn. Since its initial public release in 2003 around twenty contributors, from all regions of the world, have invested their expertise and time to support the development of the software. Both binary and source code releases take place at regular intervals. Qucs numbered releases and day-to-day development code snapshots can be downloaded from (http://qucs.sourceforge.net). Versions are available for Linux (Ubuntu and other distributions), Mac OS X © and the Windows © 32 bit operating system.

In the period since Qucs was first released it has evolved into an advanced circuit simulation and device modelling tool with a user friendly "graphical user interface" (GUI) for circuit schematic capture, for investigating circuit and device properties from DC to RF and beyond, and for launching other circuit simulation software, including the FreeHDL (VHDL) and Icarus Verilog digital simulators. Qucs includes built-in code for processing and visualising simulation output data. Qucs also allows users to process post-simulation data with the popular Octave numerical data analysis package. Similarly, circuit performance optimisation is possible using the A SPICE Circuit Optimizer (ASCO) package or Python code linked to Qucs.

Between 2003, and January 2015, the sourceforge Qucs download statistics show that over one million downloads of the software have been recorded. As well as extensive circuit simulation capabilities Qucs supports a full range of device modelling features, including non-linear and RF equation-defined device modelling and the use of the Verilog-A hardware description language (HDL) for compact device modelling and macromodelling. Recent extensions to the software aim to diversify the Qucs modelling facilities by running the Berkeley "Model and Algorithm Prototyping Platform" (MAPP) in parallel with Qucs, using Octave launched from the Qucs GUI. In the future, as the Qucs project evolves, the software will also provide circuit designers with a choice of simulation engine selected from the Qucs built-in code, ngspice and Xyce ©.

Qucs is a large software package which takes time to learn. Incidentally, this statement is also true for other GPL circuit simulators. New users must realise that to get the best from the software some effort is required on their part. In particular, one of the best ways to become familiar with Qucs is to learn a few basic user rules and how to apply them. Once these have been mastered users can move on with confidence to next level of understanding. Eventually, a stage will be reached which allows Qucs to be used productively to model devices and to investigate the performance of circuits. Qucs is equally easy to use by absolute beginners, like school children learning the physics of electrical circuits consisting of a battery and one or more resistors, as it is by cutting edge engineers working on the modelling of sub-nano sized RF MOS transistors with hundreds of physical parameters.

The primary purpose of these notes is to provide Qucs users with a source of reference for the operation and capabilities of the software. The information provided also indicates any known limitations and, if available, provides details of any work-arounds. Qucs is a high level scientific/engineering tool who's operation and performance does require users to understand the basic mathematical, scientific and engineering principles underlying the operation of electronic devices and the design and analysis of electronic circuits. Hence, the individual sections of the Qucs-Help document include material of a technical nature mixed in with details of the software operation. Most sections introduce a number of worked design and simulation examples. These have been graded to help readers with different levels of understand get the best from the Qucs circuit simulator. Qucs-Help is a dynamic document which will change with every new release of the Qucs software. At this time, Qucs release 0.0.19, the document is far from complete but given time it will improve.

CHAPTER 2

Getting Started with Qucs Analogue Circuit Simulation

Qucs is a scientific/engineering software package for analogue and digital circuit simulation, including linear and non-linear DC analysis, small signal S parameter circuit analysis, time domain transient analysis and VHDL/Verilog digital circuit simulation. This section of the Qucs-Help document introduces readers to the basic steps involved in Qucs analogue circuit simulation. When Qucs is launched for the first time, it creates a directory called `.qucs` within the user's home directory. All files involved in Qucs simulations are saved in the `.qucs` directory or in one of it's sub-directories. After Qucs has been launched, the software displays a Graphical User Interface window (GUI) similar, or the same, to the one shown in Figure 1.

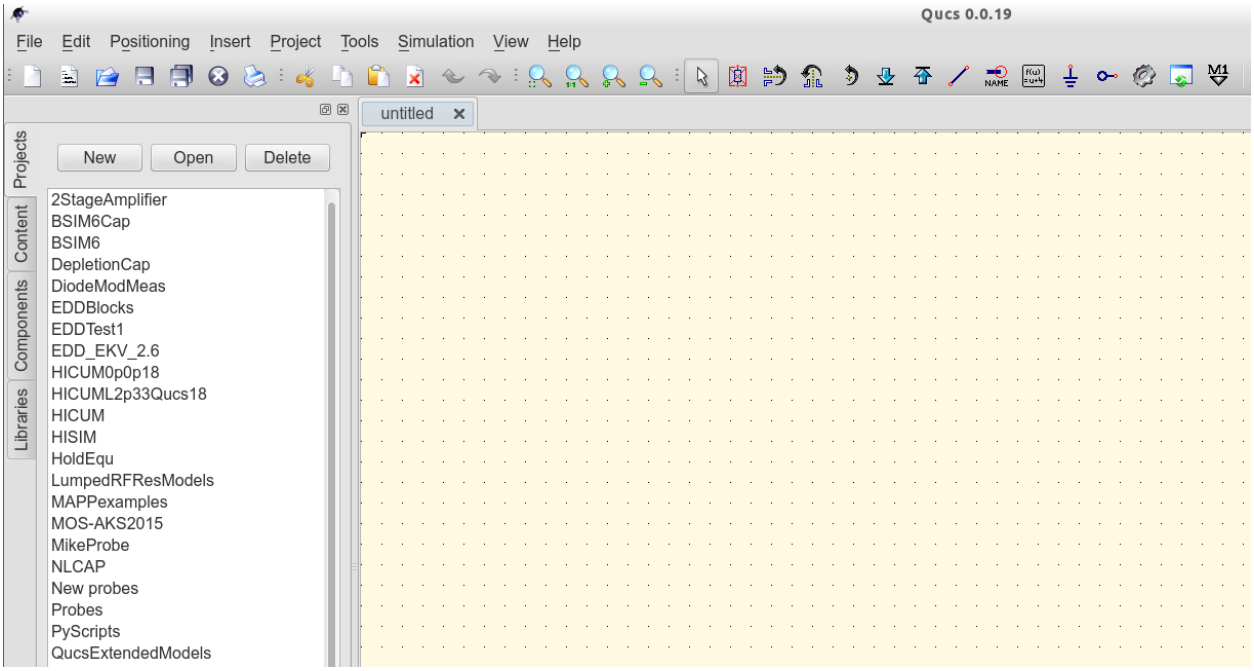


Figure 1 - Qucs main window

Before using Qucs it is advisable to set the program application settings. This is done from the **File** → **Application Settings** menu. Clicking on **Application Settings** causes the **EditQucsProperties** window to be displayed, see Figure

2. Complete, with appropriate entries for your Qucs installation, the **Settings**, **Source Code Editor**, **File Types** and **Locations** menus.

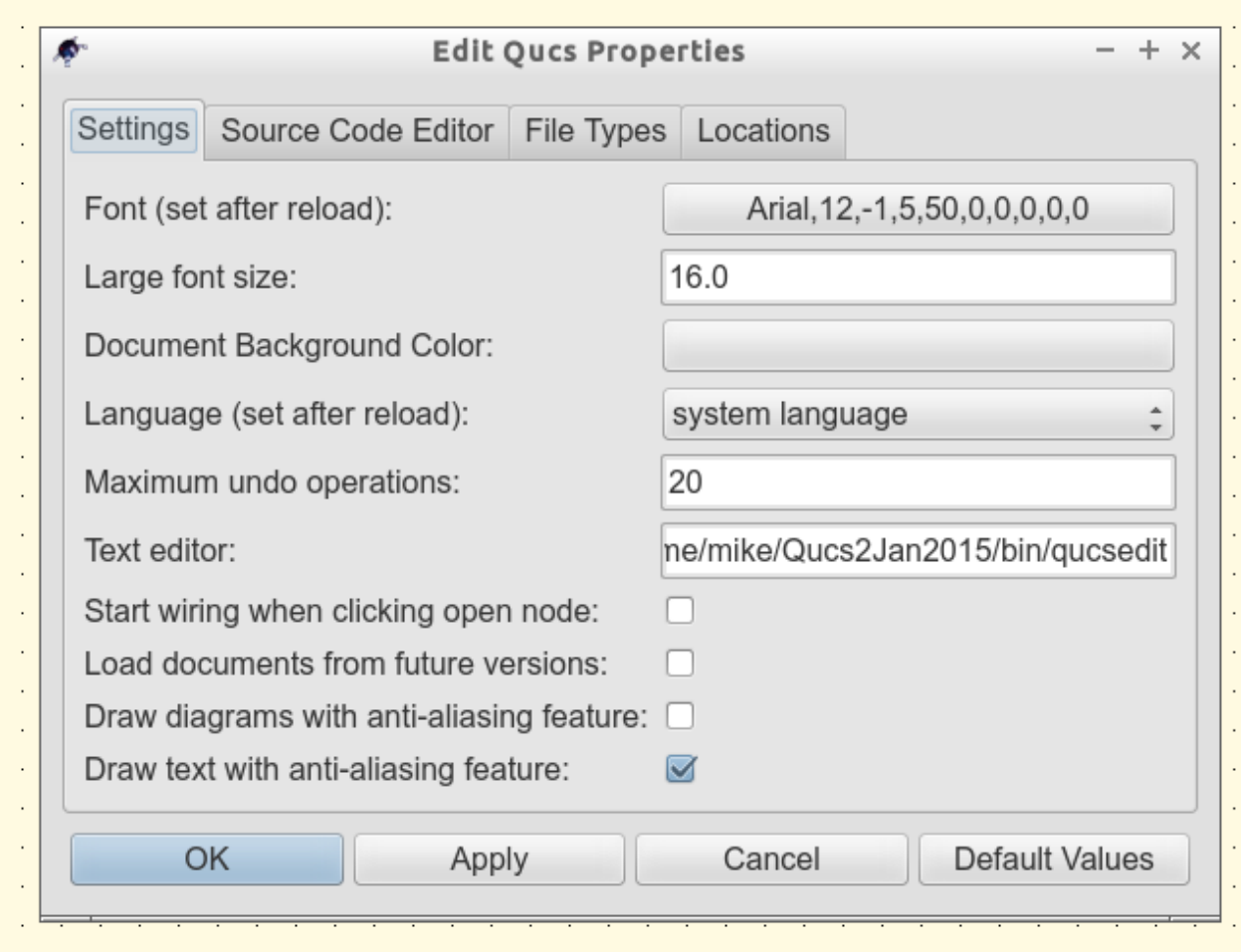


Figure 2 - QucsEditProperties window

On launching Qucs a working area labelled (6) appears at the centre of the GUI. This window is used for displaying schematics, numerical and algebraic model and circuit design data, numerical output data, and signal waveforms and numerial data visualised as graphs, see Figure 3. Clicking, with the left hand mouse button on any of the entries in the tabular bar labelled (5) allows users to quickly switch between the currently open documents. On the left hand side of the Qucs main window is a third area labelled (1) whose content depends on the status of **Projects** (2), **Content** (3), **Components** (4) or **Libraries**. After running Qucs, the **Projects** tab is activated. However note, when Qucs is launched for the first time the **Projects** list is empty.

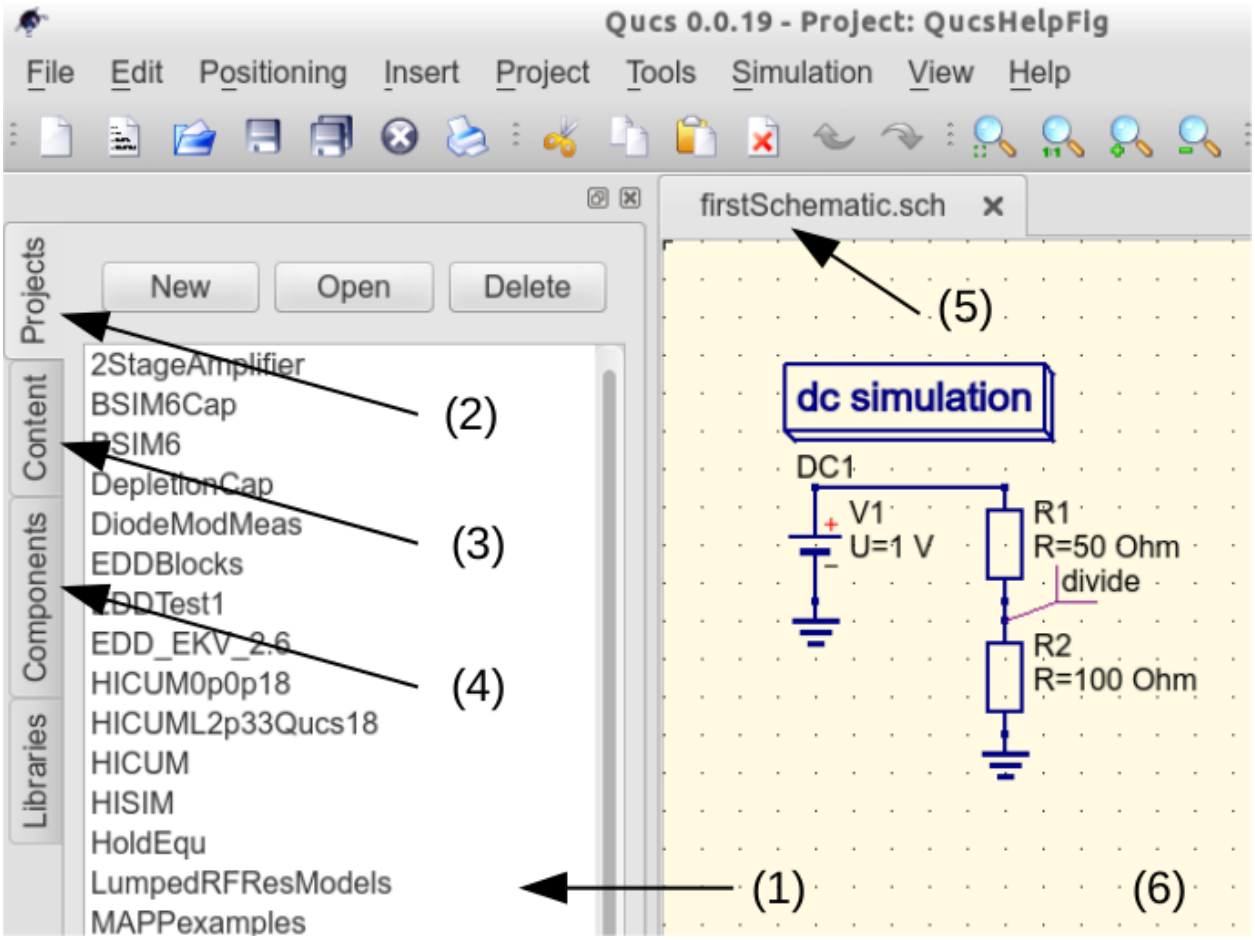


Figure 3 - Qucs main window with working areas labelled

To enter a new project left click on the **New** button located on the right above window (1). This action causes a Qucs GUI dialogue to open. Enter the name of a Qucs project in the box provided, for example enter *QucsHelpFig* and click on the **OK** button. Qucs then creates a project directory in the `~/.qucs` directory. In the example shown in Figure 3 this is called **QucsHelpFig_prj**. Every file belonging to this new project is saved within the **QucsHelpFig_prj** directory. On creation a new project is immediately opened and it's name displayed on the Qucs window title bar. The left tabular bar is then switched to **Content**, and the content of the currently opened project displayed. To save an open document click on the **save** button (or use the main menu: **File** $\longrightarrow$ **Save**). This step initiates a sequence which saves the document displayed in area (6). To complete the save sequence the program will request the name of your new document. Enter *firstSchematic*, or some other suitable name, and click on the **OK** button to complete the save sequence.

As a first example to help you get started with Qucs enter and run the simple DC circuit shown in Figure 3. The circuit illustrated is a two resistor voltage divider network connected to a fixed value DC voltage source. Start by clicking on the **Components** tab. This action causes a combo box to be displayed from which a component group may be choosen and the required components selected. Choose components group **lumped components** and click on the first symbol: **Resistor**. Next move the mouse cursor into area (6). Pressing the right mouse button rotates the **Resistor** symbol. Similarly, pressing the left mouse button places the component onto the schematic at the place the mouse cursor is pointing at. Repeat this process for all components shown in Figure 3. The independent DC voltage source is located in the **sources** group. The ground symbol can be found in the **lumped components** group or selected from the Qucs toolbar. The icon requesting DC simulation is listed in the **simulations** group. To edit the parameters of the second resistor, double-click on it. A dialogue opens which allows the resistor value to be changed; enter *100 Ohm* in the edit field on the right hand side and click enter.

To connect the circuit components shown in Figure 3, click on the wire toolbar button (or use the main menu: **Insert** $\longrightarrow$ **Wire**). Move the cursor onto an open component port (indicated by a small red circle at the end of a blue wire). Clicking on it starts the wire drawing sequence. Now move the drawing cursor to the end point of a wire (normally this is a second red circle attached to a placed component) and click again. Two components are now connected. Repeat the drawing sequence as many times as required to wire up the example circuit. If you want to change the corner direction of a wire, click on the right mouse button before moving to an end point. You can also end a wire without clicking on an open port or on a wire; just double-click the left mouse button.

As a final step before DC simulation label the node, or nodes, who's DC voltage is required, for example the wire connecting resistors **R1** and **R2**. Click on the label toolbar button (or use the menu: **Insert** $\longrightarrow$ **Wire Label**). Now click on the chosen wire. A dialogue opens allowing a node name to be entered. Type *divide* and click the **OK** button. If you have drawn the test schematic correctly the entered schematic should look the same, or be similar to, the one shown in Figure 3.

To start DC simulation click on the **Simulate** toolbar button (or use menu: **Simulation** $\rightarrow$ **Simulate**). A simulation window opens and a sliding bar reports simulation progress. Normally, all this happens so fast that you only see a short flickering on the PC display (this depends on the speed of your PC). After finishing a simulation successfully Qucs opens a data display window. This replaces the schematic entry window labelled (6) in Figure 3. Next the **Components** $\longrightarrow$ **diagrams** toolbar is opened. This allows the simulation results to be listed. Click on the **Tabular** item and move it to the display working area, placing it by clicking the left hand mouse button. A dialogue opens allowing selection of the named signals you wish to list, see Figure 4. On the left hand side of the **Tabular** dialogue (called Edit Diagram Properties) is listed the node name: **divide.V**. Double-click on it and it will be transferred to the right hand side of the dialogue. Leave the dialogue by clicking the **OK** button. The DC simulation voltage data for node **divide** should now be listed in a box on the data display window, with a value of 0.666667 volts.

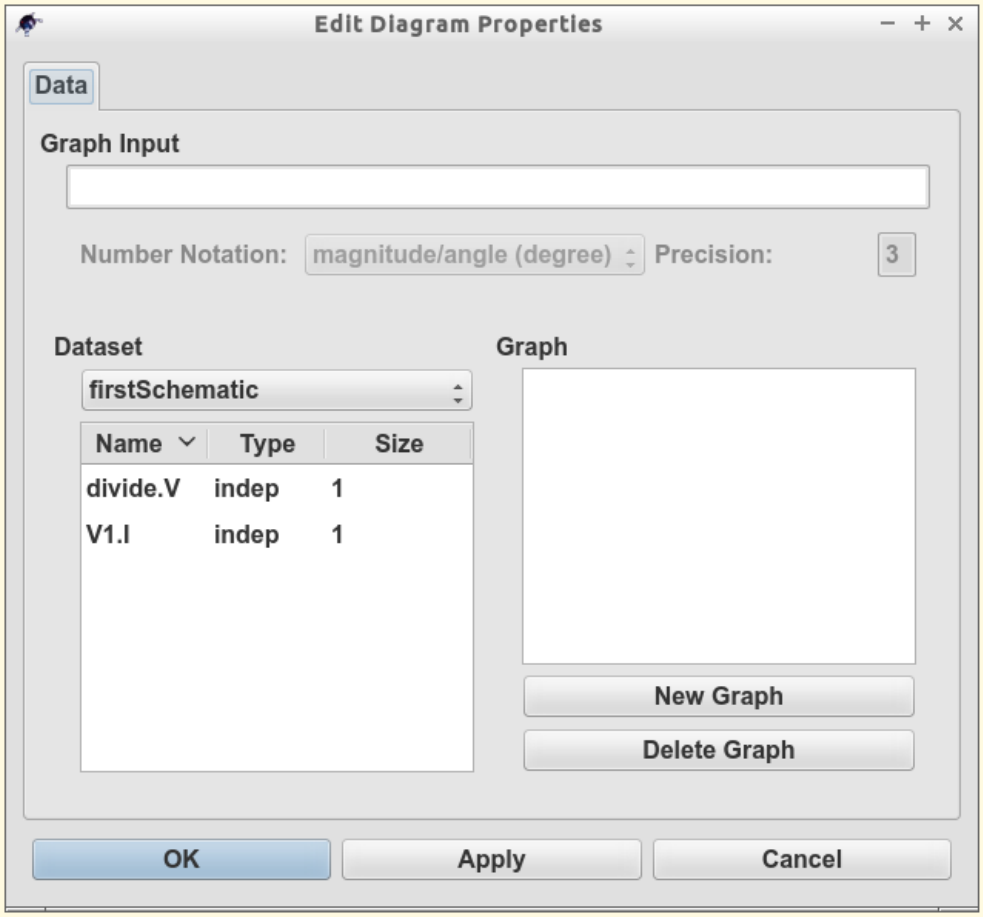


Figure 4 - Qucs data display window showing a **Tabular** dialogue

CHAPTER 3

Getting Started with Optimization

For circuit optimization, Qucs uses the ASCO tool (http://asco.sourceforge.net/). A brief description on how to prepare your schematic, execute and interprete the results is given below. Before using this functionality, ASCO must be installed on the computer.

Optimization of a circuit is nothing more than the minimization of a cost function. It can either be the delay or the rise time of a digital circuit, or the power or gain of an analog circuit. Another possibility is defining the optimization problem as a composition of functions, leading in this case to the definition of a figure-of-merit.

To setup a netlist for optimization two things must be added to the already existing netlist: insert equation(s) and the optimization component block. Take the schematic from Figure 1 and change it until you have the resulting schematic given in Figure 2.

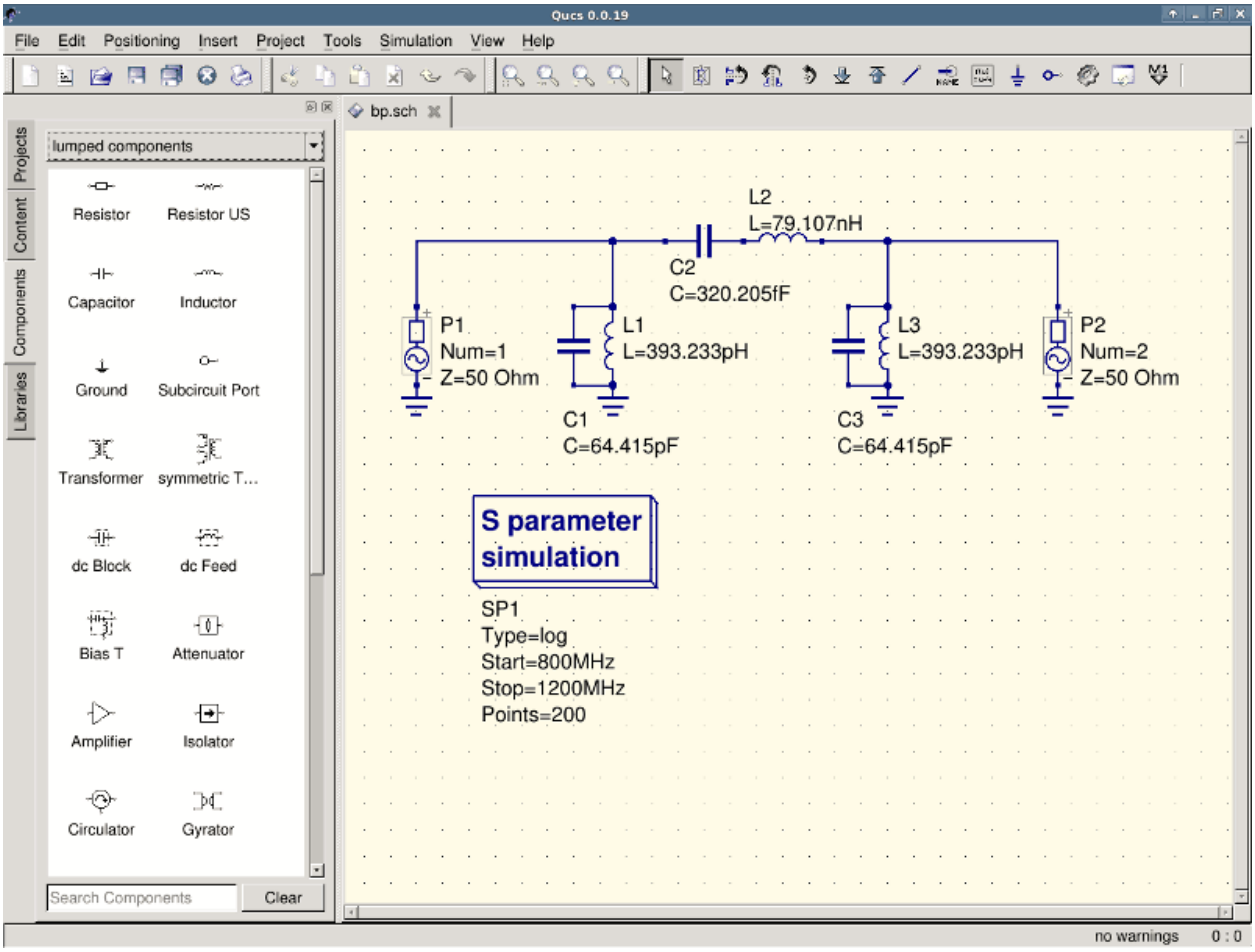


Figure 1 - Initial schematic.

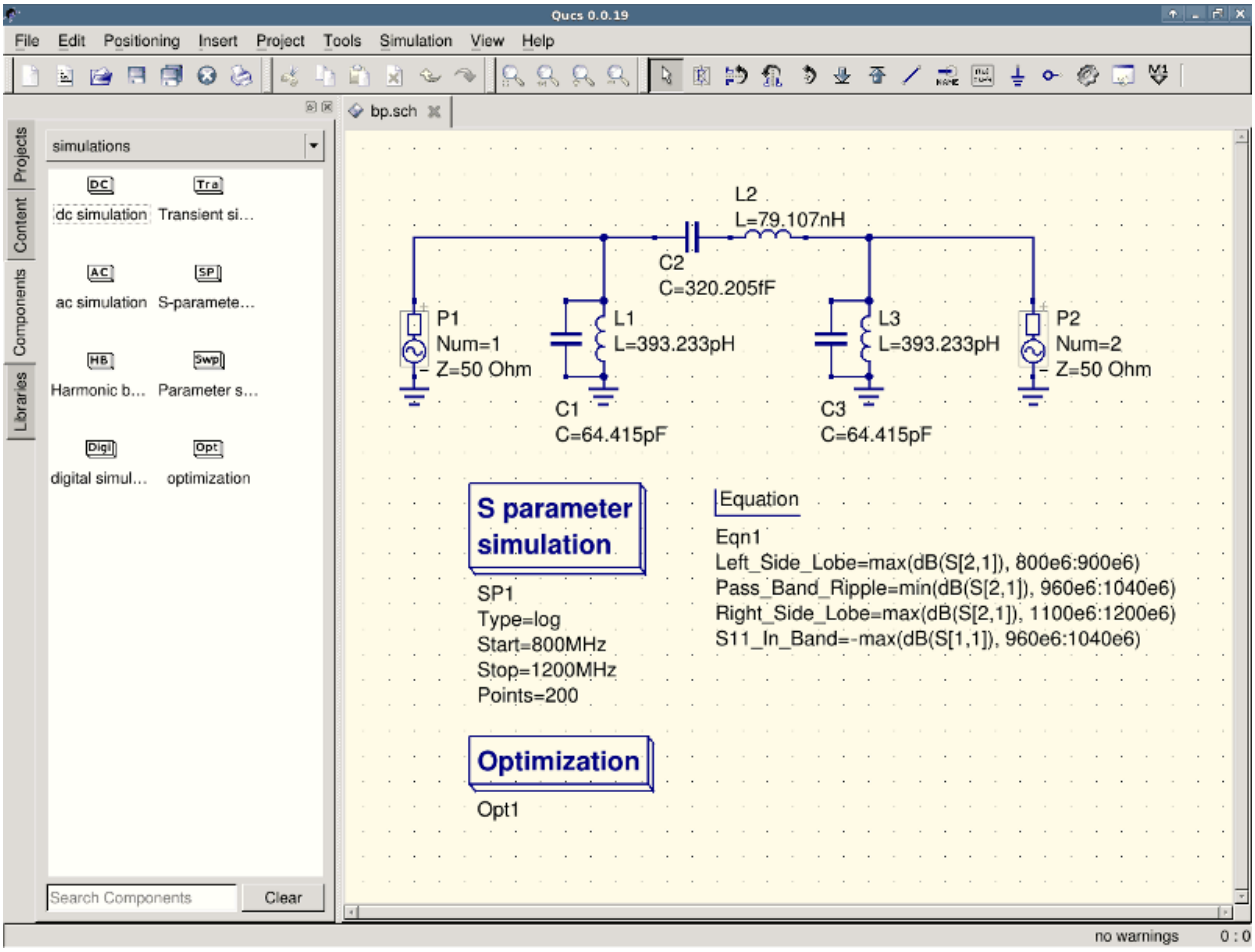


Figure 2 - Prepared schematic.

Now, open the optimization component and select the optimization tab. From the existing parameters, special attention should be paid to ‘Maximum number of iterations’, ‘Constant F’ and ‘Crossing over factor’. Over- or underestimation can lead to a premature convergence of the optimizer to a local minimum or, a very long optimization time.

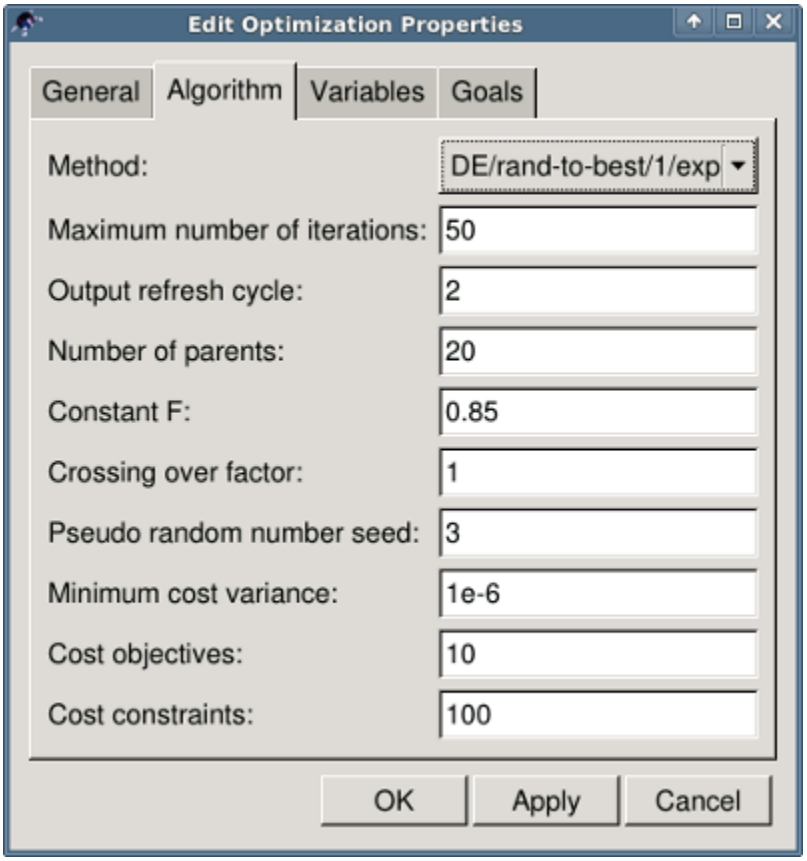


Figure 3 - Optimization dialog, algorithm options.

In the Variables tab, defining which circuit elements will be chosen from the allowed range, as shown in Figure 4. The variable names correspond to the identifiers placed into properties of components and **not** the components' names.

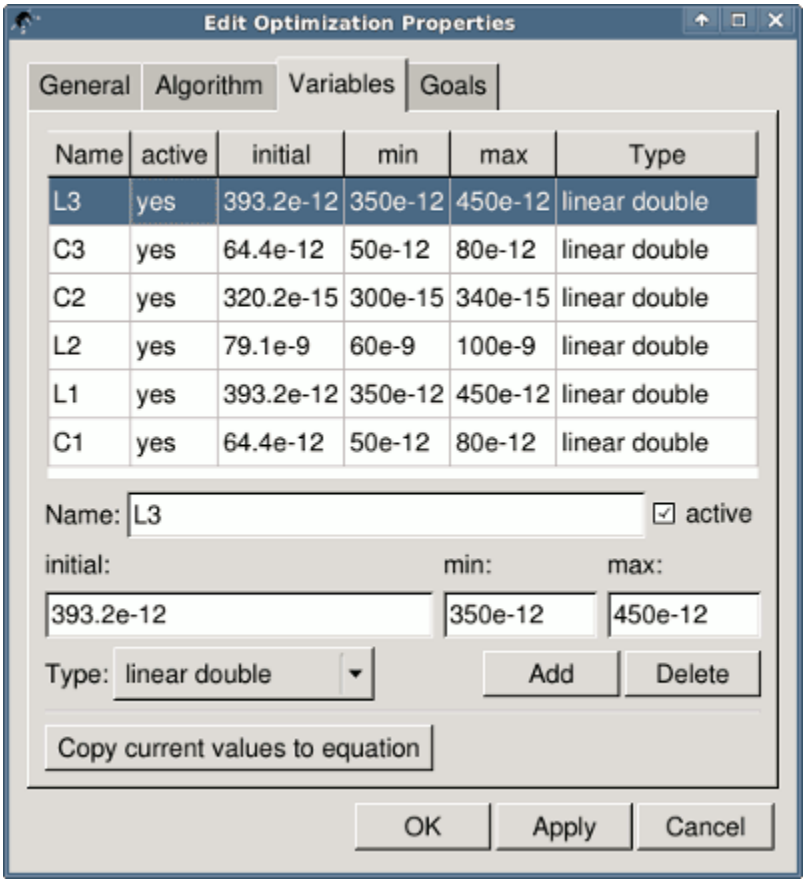


Figure 4 - Optimization dialog, variables options.

Finally, go to Goals where the optimization objective (maximize, minimize) and constraints (less, greater, equal) are defined. ASCO then automatically combines them into a single cost function, that is then minimized.

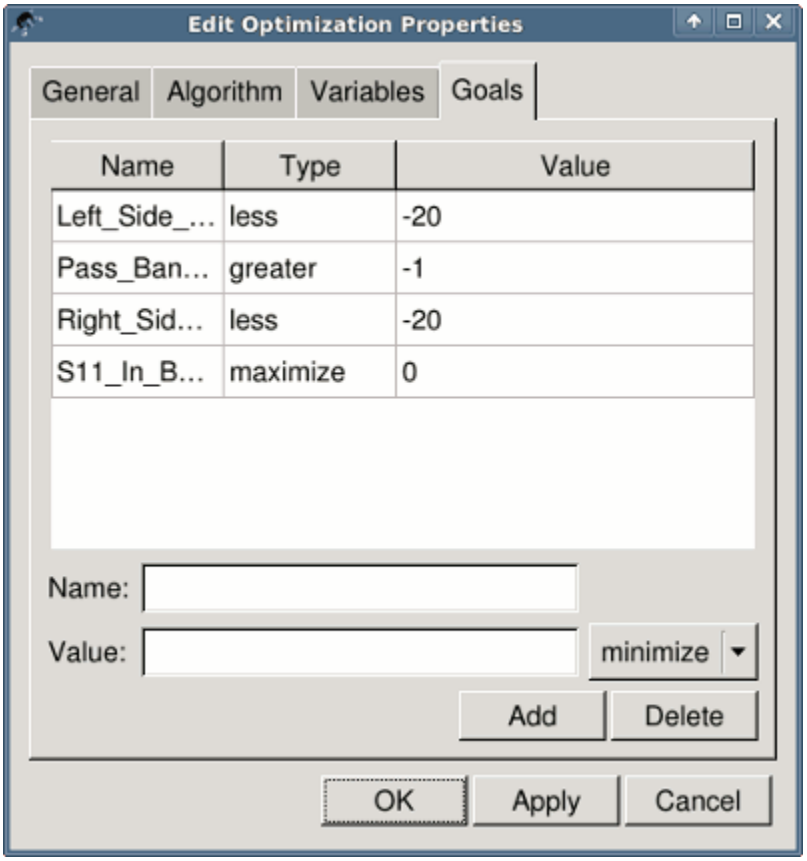


Figure 5 - Optimization dialog, goals options.

The next step is to change the schematic, and define which circuit elements are to be optimized. The resulting schematic is show in Figure 6.

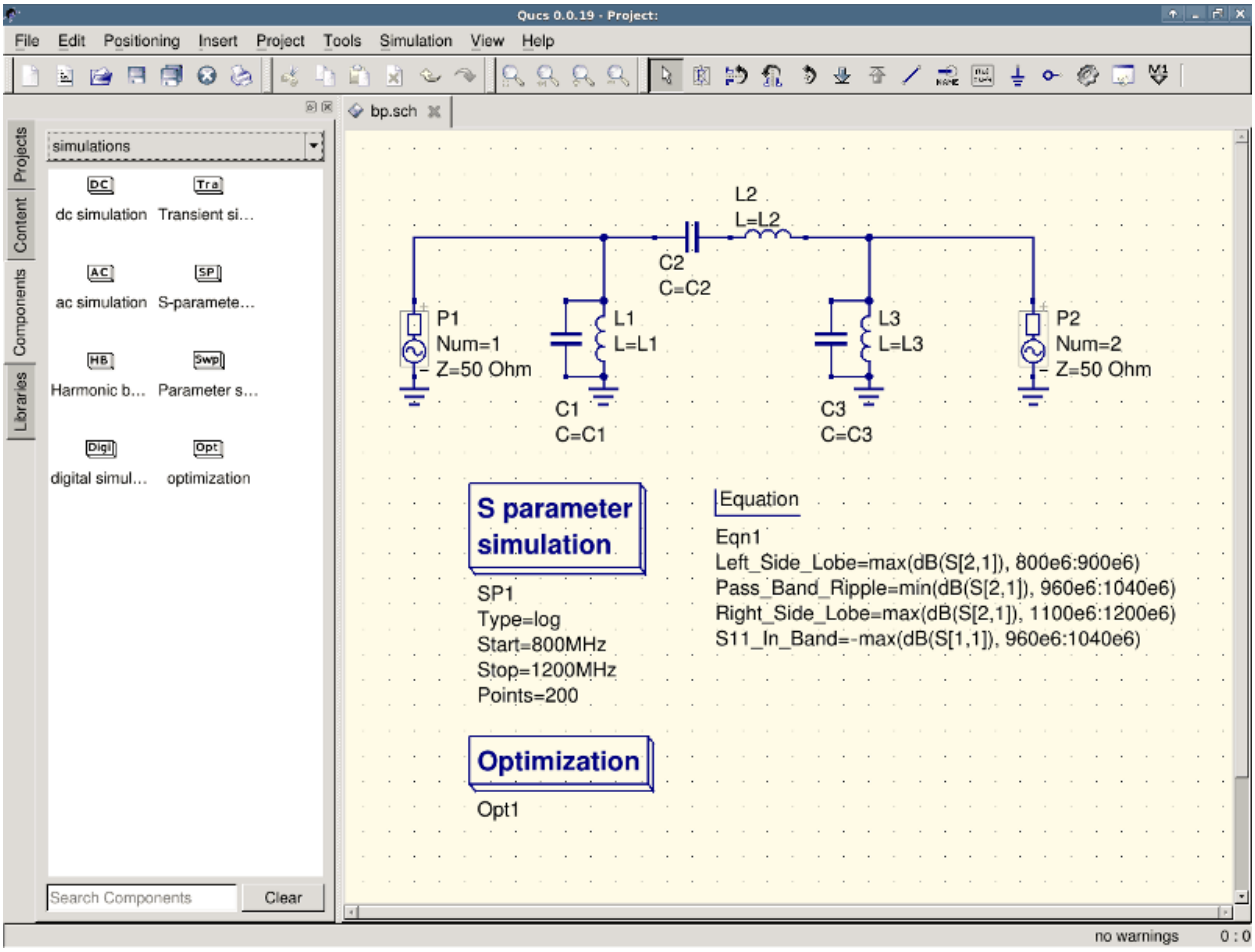
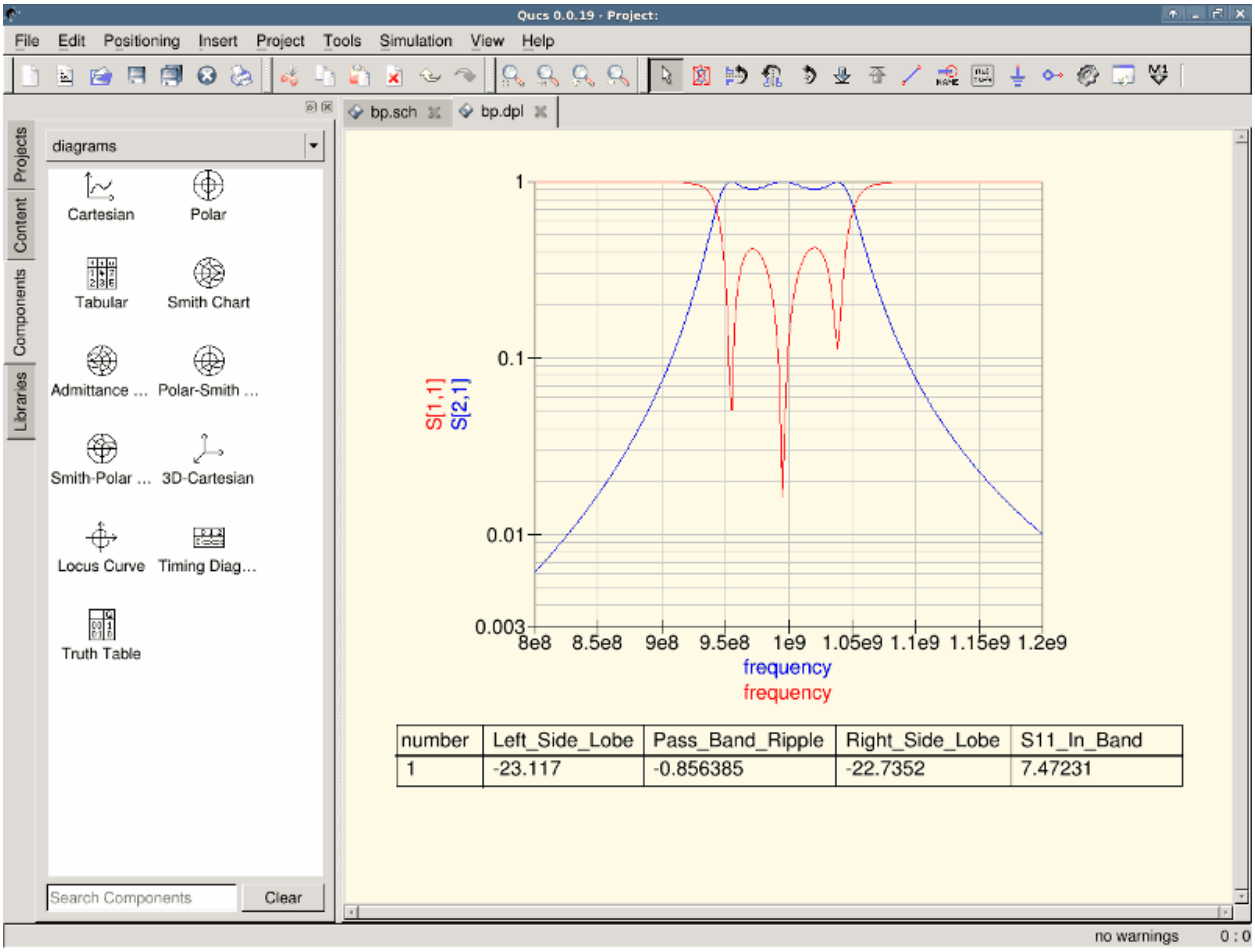


Figure 6 - New Qucs main window.

The last step is to run the optimization, i.e. the simulation by pressing F2. Once finished, which takes a few seconds on a modern computer, the best simulation results is shown in the graphical waveform viewer.



number	Left_Side_Lobe	Pass_Band_Ripple	Right_Side_Lobe	S11_In_Band
1	-23.117	-0.856385	-22.7352	7.47231

Figure 7 - Qucs results window.

The best found circuit sizes can be found in the optimization dialog, in the Variables tab. They are now the initial values for each one of introduced variables (Figure 8).

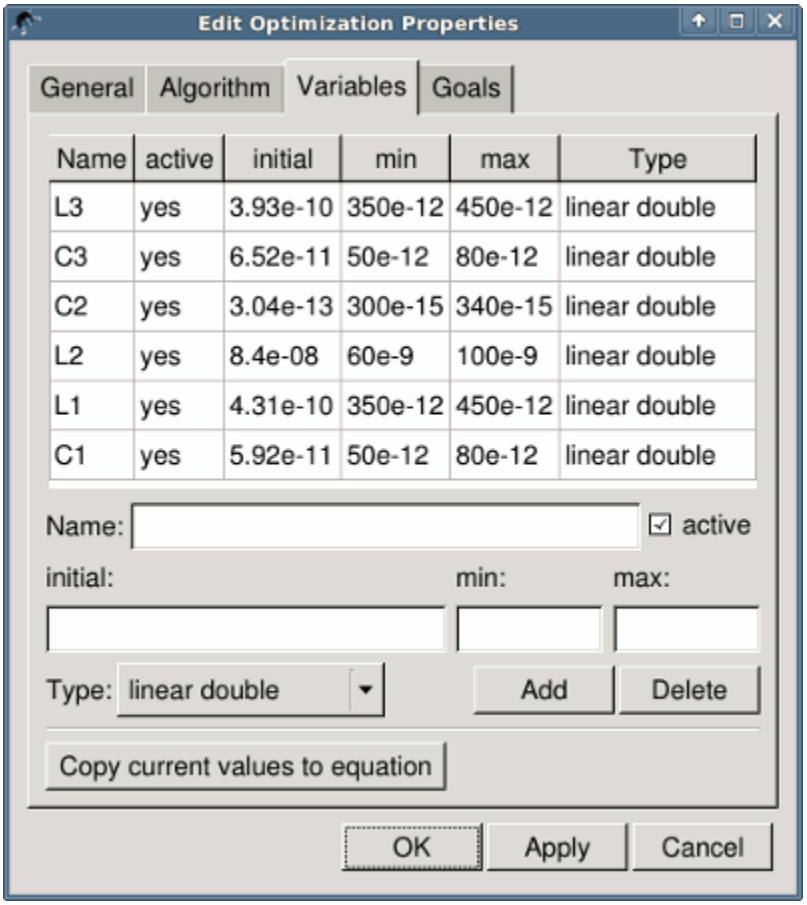


Figure 8 - The best found circuit sizes.

By clicking the “Copy current values to equation” button, an equation component defining all the optimization variables with the values of the “initial” column will be copied to the clipboard and can be pasted to the schematic after closing the optimization dialog. The resulting schematic will be as shown in the next figure.

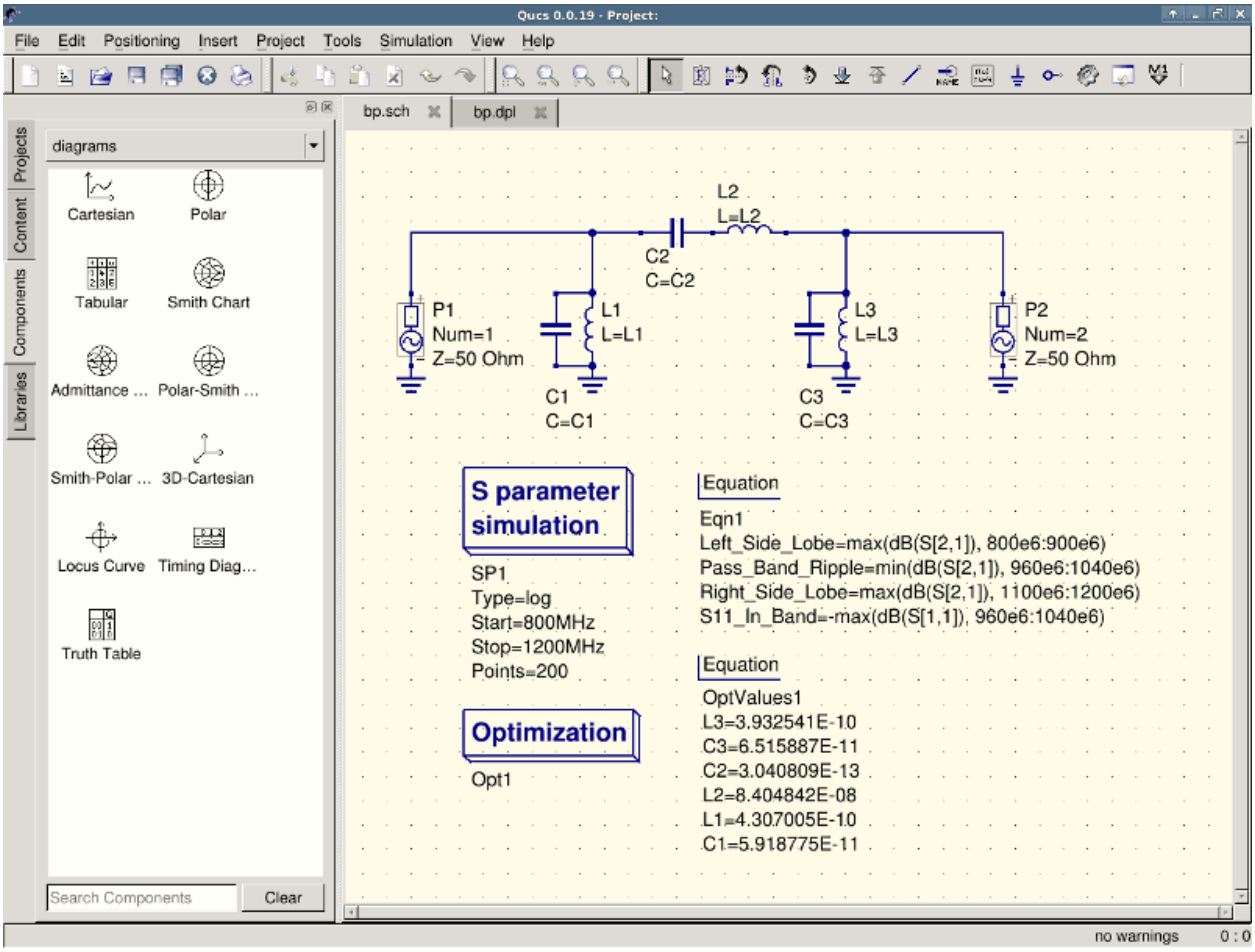


Figure 9 - Schematic with optimized values.

in case you need to do further modifications to the schematic, the optimization component can now be disabled and the optimized values from the pasted equation will be used.

You can change the number of figures shown for the optimized values in the optimization dialog by right-clicking on the “initial” table header and selecting the “Set precision” menu, as shown in the following figure.

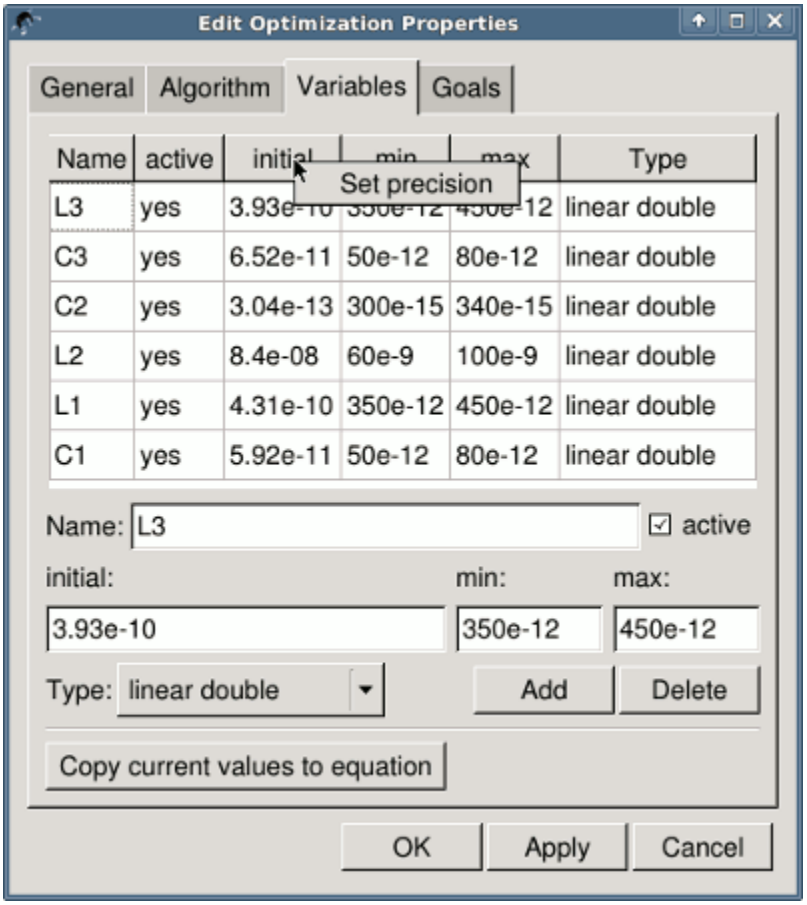


Figure 10 - Changing the displayed variables precision.

CHAPTER 4

Getting Started with Octave Scripts

Qucs can also be used to develop Octave scripts (see http://www.octave.org). This document should give you a short description on how to do this.

If the user creates a new text document and saves it with the Octave extension, e.g. 'name.m' then the file will be listed at the Octave files of the active project. The script can be executed with F2 key or by pressing the simulate button in the toolbar. The output can bee seen in the Octave window that opens automatically (per default on the right-hand side). At the bottom of the Octave window there is a command line where the user can enter single commands. It has a history function that can be used with the cursor up/down keys.

There are two Octave functions that load Qucs simulation results from a dataset file: loadQucsVariable() and loadQucsDataset(). Please use the help function in the Octave command line to learn more about them (i.e. type help loadQucsVariable and help loadQucsDataset).

4.1 Postprocessing

Octave can also be used for automatic postprocessing of a Qucs simulation result. This is done by editing the data display file of a schematic (Document Settings... in File menu). If the filename of an Octave script (filename extension m) from the same project is entered, this script will be executed after the simulation is finished.

CHAPTER 5

Short Description of Actions

5.1 General Actions

(valid in all modes)

mouse wheel	Scrolls vertically the drawing area. You can also scroll outside the current size.
mouse wheel + Shift Button	Scrolls horizontally the drawing area. You can also scroll outside the current size.
mouse wheel + Ctrl Button	Zooms into or outof the drawing area.
drag'n'drop file into document area	Tries to open file as Qucs schematic or data display.

5.2 "Select"-Mode

(Menu: Edit->Select)

left mouse button	Selects the element below the mouse cursor. If several components are placed there, you can clicking several times in order to select the wanted one. Keeping the mouse button pressed, you can move the component below the mouse cursor and all selected ones. If you want to fine position the components, press the CTRL key during moving and the grid is disabled. Keeping the mouse button pressed without any element below it opens a rectangle. After releasing the mouse button, all elements within this rectangle are selected. A selected diagram or painting can be resized by pressing the left mouse button over one of its corners and moving by keeping the button pressed. After clicking on a component text, it can be edited directly. The enter key jumps to the next property. If the property is a selection list, it can only be changed with the cursor up/down keys. Clicking on a circuit node enters the "wire mode".
left mouse button + Ctrl Button	Allows more than one element to be selected, i.e. selecting an element does not deselect the others. Clicking on a selected element deselects it. This mode is also valid for selecting by opening a rectangle (see item before).
right mouse button	Clicking on a wire selects a single straight line instead of the complete line.
double-click right mouse button	Opens a dialog to edit the element properties (The labels of wires, the parameters of components, etc.).

5.3 "Insert Component"-Mode

(Click on a component/diagram in the left area)

left mouse button	Place a new instance of the component onto the schematic.
right mouse button	Rotate the component. (Has no effect on diagrams.)

5.4 "Wire"-Mode

(Menu: Insert->Wire)

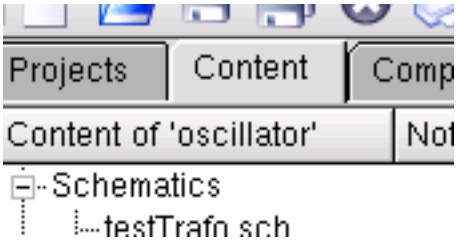
left mouse button	Sets the starting/ending point of the wire.
right mouse button	Changes the direction of the wire corner (first left/right or first up/down).
double-click right mouse button	Ends a wire without being on a wire or a port.

5.5 "Paste"-Mode

(Menu: Edit->Paste)

left mouse button	Place the elements onto the schematic (from the clipboard).
right mouse button	Rotate the elements.

5.6 Mouse in “Content” Tab



left click	Selects file.	
double-click	Opens file.	
right click	Displays menu with:	
	“open”	• open selected file
	“rename”	• change name of selected file
	“delete”	• delete selected file
	“copy file”	• copy schematic file. Only file operations are performed. Dataset and dispaly settings in schematic properties are kept untouched.

5.7 Keyboard

Many actions can be activated/done by the keyboard strokes. This can be seen in the main menu right beside the command. Some further key commands are shown in the following list:

“Delete” or “Backspace”	Deletes the selected elements or enters the delete mode if no element is selected.
Cursor left/right	Changes the position of selected markers on their graphs. If no marker is selected, move selected elements. If no element is selected, scroll document area.
Cursor up/down	Changes the position of selected markers on more-dimensional graphs. If no marker is selected, move selected elements. If no element is selected, scroll document area.
Tabulator	Changes to the next open document (according to the TabBar above).

CHAPTER 6

Working with Subcircuits

Subcircuits are used to bring more clarity into a schematic. This is very useful in large circuits or in circuits, in which a component block appears several times.

In Qucs, each schematic containing a subcircuit port is a subcircuit. You can get a subcircuit port by using the toolbar, the components listview (in lumped components) or the menu (Insert->Insert port). After placing all subcircuit ports (two for example) you need to save the schematic (e.g. CTRL-S). By taking a look into the content listview (figure 1) you see that now there is a “2-port” right beside the schematic name (column “Note”). This note marks all documents which are subcircuits. Now change to a schematic where you want to use the subcircuit. Then press on the subcircuit name (content listview). By entering the document area again, you see that you now can place the subcirciut into the main circuit. Do so and complete the schematic. You can now perform a simulation. The result is the same as if all the components of the subcircuit are placed into the circuit directly.

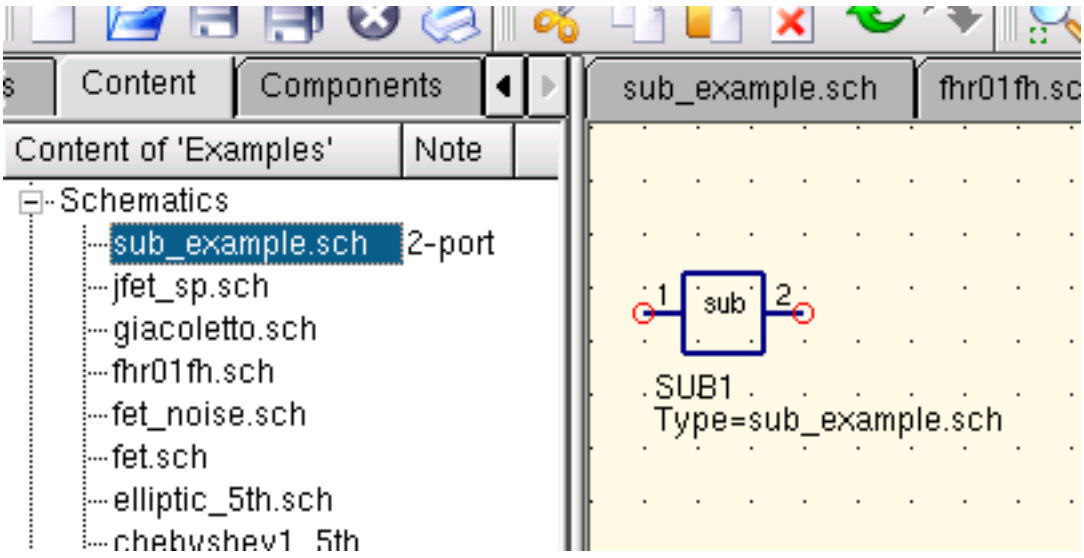


Figure 1 - Accesing a subcircuit

If you select a subcircuit component (click on its symbol in the schematic) you can step into the subcircuit schematic by pressing CTRL-I (of course, this function is also reachable via toolbar and via menu). You can step back by pressing

CTRL-H.

If you do not like the component symbol of a subcircuit, you can draw your own symbol and put the component text at your favourite position. Just make the subcircuit schematic the current and go to the menu: File->Edit Circuit Symbol. If you have not yet drawn a symbol for this circuit. A simple symbol is created automatically. You now can edit this symbol by painting lines and arcs. After finished, save it. Now place it on another schematic, and you have a new symbol.

Just like all other components, subcircuits can have parameters. To create your own parameters, go back to the editor where you edited the subcircuit symbol and double-click on the subcircuit parameter text (SUB1 in the Figure 1). A dialog apperas and you now can fill in parameterswith default values and descriptions. When you are ready, close the dialog and save the subcircuit. In every schematic where the subcircuit is placed, it owns the new parameters which can be edited as in all other components.

6.1 Subcircuits with Parameters

A simple example using subcircuits with parameters and equations is provided here.

Create a subcircuit:

- Create a new project
- New schematic (for subcircuit)
- Add a resistor, inductor, and capacitor, wire them in series, add two ports
- Save the subcircuit as RLC.sch
- Give value of resistor as 'R1'
- Add equation 'ind = L1',
- Give value of inductor as 'ind'
- Give value of capacitor as 'C1'
- Save
- File > Edit Circuit Symbol
- Double click on the 'SUB File=name' tag under the rectangular box
  - Add name = R1, default value = 1
  - Add name = L1, default value = 1
  - Add name = C1, default value = 1
  - OK

Insert subcircuit and define parameters:

- New schematic (for testbench)
- Save Test_RLC.sch
- Project Contents > pick and place the above RLC subcircuit
- Add AC voltage source (V1) and ground
- Add AC simulation, from 140Hz to 180Hz, 201 points
- Set on the subcircuit symbol
  - R1=1

  - L1=100e-3
  - C1=10e-6
- Simulate
- Add a Cartesian diagram, plot V1.i
- The result should be the resonance of the RLC circuit.
- The parameters of the RLC subcircuit can be changed on the top schematic.

CHAPTER 7

Getting Started with Digital Simulations

Qucs is also a graphical user interface for performing digital simulations. This document should give you a short description on how to use it.

For digital simulations Qucs uses the FreeHDL program (http://www.freehdl.seul.org). So the FreeHDL package as well as the GNU C++ compiler must be installed on the computer.

There is no big difference in running an analog or a digital simulation. So having read the Getting Started for analog simulations, it is now easy to get a digital simulation work. Let us compute the truth table of a simple logical AND cell. Select the digital components in the combobox of the components tab on the left-hand side and build the circuit shown in figure 1. The digital simulation block can be found among the other simulation blocks.

The digital sources *S1* and *S2* are the inputs, the node labeled as *Output* is the output. After performing the simulation, the data display page opens. Place the diagram *truth table* on it and insert the variable *Output*. Now the truth table of a two-port AND cell is shown. Congratulation, the first digital simulation is done!

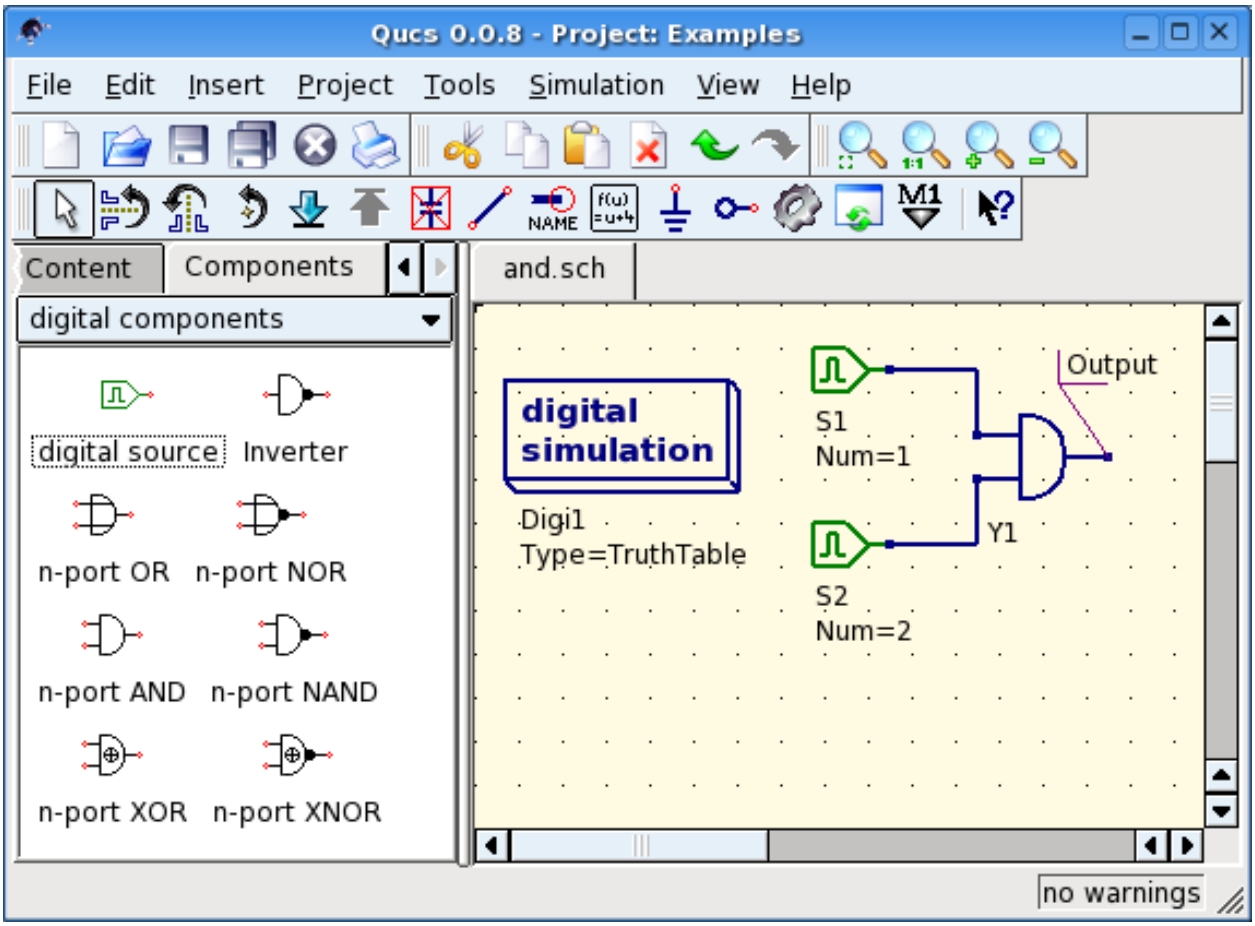


Figure 1 - Qucs main window

A truth table is not the only digital simulation that Qucs can perform. It is also possible to apply an arbitrary signal to a circuit and see the output signal in a timing diagram. To do so, the parameter *Type* of the simulation block must be changed to *TimeList* and the duration of the simulation must be entered in the next parameter. The digital sources have now a different meaning: They can output an arbitrary bit sequence by defining the first bit (low or high) and a list that sets the times until the next change of state. Note that this list repeats itself after its end. So, to create a 1GHz clock with pulse ratio 1:1, the list writes: 0.5ns; 0.5ns

To display the result of this simulation type, there is the diagram *timing diagram*. Here, the result of all output nodes can be shown row by row in one diagram. So, let's have fun...

7.1 VHDL File Component

More complex and more universal simulations can be performed using the component "VHDL file". This component can be picked up from the component list view (section "digital components"). Nevertheless the recommended usage is the following: The VHDL file should be a member of the project. Then go to the content list view and click on the file name. After entering the schematic area, the VHDL component can be placed.

The last entity block in the VHDL file defines the interface, that is, all input and output ports must be declared here. These ports are also shown by the schematic symbol and can be connected to the rest of the circuit. During simulation the source code of the VHDL file is placed into the top-level VHDL file. This must be considered as it causes some limitations. For example, the entity names within the VHDL file must differ from names already given to subcircuits. (After a simulation, the complete source code can be seen by pressing F6. Use it to get a feeling for this procedure.)

CHAPTER 8

Short Description of Mathematical Functions

The following operations and functions can be applied in Qucs equations. For detailed description please refer to the "Measurement Expressions Reference Manual". Parameters in brackets "[]" are optional.

8.1 Operators

8.1.1 Arithmetic Operators

`+x`	Unary plus
`-x`	Unary minus
`x+y`	Addition
`x-y`	Subtraction
`x*y`	Multiplication
`x/y`	Division
`x%y`	Modulo (remainder of division)
`x^y`	Power

8.1.2 Logical Operators

`!x`	Negation
`x&&y`	And
`x||y`	Or
`x^^y`	Exclusive or
`x?y:z`	Abbreviation for conditional expression - if `x` then `y` else `z`
`x==y`	Equal
`x!=y`	Not equal
`x<y`	Less than
`x<=y`	Less than or equal
`x>y`	Larger than
`x>=y`	Larger than or equal

8.2 Math Functions

8.2.1 Vectors and Matrices: Creation

`eye(n)`	Creates `n` x `n` identity matrix
`length(y)`	Returns the length of the `y` vector
`linspace(from,to,n)`	Real vector with `n` lin spaced components between `from` and `to`
`logspace(from,to,n)`	Real vector with `n` log spaced components between `from` and `to`

8.2.2 Vectors and Matrices: Basic Matrix Functions

`adjoint(x)`	Adjoint matrix of `x` (transposed and conjugate complex)
`det(x)`	Determinant of a matrix `x`
`inverse(x)`	Inverse matrix of `x`
`transpose(x)`	Transposed matrix of `x` (rows and columns exchanged)

8.2.3 Elementary Mathematical Functions: Basic Real and Complex Functions

`abs(x)`	Absolute value, magnitude of complex number
`angle(x)`	Phase angle in radians of a complex number. Synonym for `arg()`
`arg(x)`	Phase angle in radians of a complex number
`conj(x)`	Conjugate of a complex number
`deg2rad(x)`	Converts phase from degrees into radians
`hypot(x,y)`	Euclidean distance function
`imag(x)`	Imaginary part of a complex number
`mag(x)`	Magnitude of a complex number
`norm(x)`	Square of the absolute value of a vector
`phase(x)`	Phase angle in degrees of a complex number
`polar(m,p)`	Transform polar coordinates `m` and `p` into a complex number
`rad2deg(x)`	Converts phase from radians into degrees
`real(x)`	Real part of a complex number
`sign(x)`	Signum function
`sqr(x)`	Square (power of two) of a number
`sqrt(x)`	Square root
`unwrap(p[,tol[,step]])`	Unwrap angle `p` (radians) – defaults `step` = 2pi, `tol` = pi

8.2.4 Elementary Mathematical Functions: Exponential and Logarithmic Functions

`exp(x)`	Exponential function to basis e
`limexp(x)`	Limited exponential function
`log10(x)`	Decimal logarithm
`log2(x)`	Binary logarithm
`ln(x)`	Natural logarithm (base e)

8.2.5 Elementary Mathematical Functions: Trigonometry

`cos(x)`	Cosine function
`cosec(x)`	Cosecant
`cot(x)`	Cotangent function
`sec(x)`	Secant
`sin(x)`	Sine function
`tan(x)`	Tangent function

8.2.6 Elementary Mathematical Functions: Inverse Trigonometric Functions

`arccos(x)`	Arc cosine (also known as "inverse cosine")
`arccosec(x)`	Arc cosecant
`arccot(x)`	Arc cotangent
`arcsec(x)`	Arc secant
`arcsin(x)`	Arc sine (also known as "inverse sine")
`arctan(x[,y])`	Arc tangent (also known as "inverse tangent")

8.2.7 Elementary Mathematical Functions: Hyperbolic Functions

`cosh(x)`	Hyperbolic cosine
`cosech(x)`	Hyperbolic cosecant
`coth(x)`	Hyperbolic cotangent
`sech(x)`	Hyperbolic secant
`sinh(x)`	Hyperbolic sine
`tanh(x)`	Hyperbolic tangent

8.2.8 Elementary Mathematical Functions: Inverse Hyperbolic Functions

`arcosh(x)`	Hyperbolic area cosine
`arcosech(x)`	Hyperbolic area cosecant
`arcoth(x)`	Hyperbolic area cotangent
`arsech(x)`	Hyperbolic area secant
`arsinh(x)`	Hyperbolic area sine
`artanh(x)`	Hyperbolic area tangent

8.2.9 Elementary Mathematical Functions: Rounding

`ceil(x)`	Round to the next higher integer
`fix(x)`	Truncate decimal places from real number
`floor(x)`	Round to the next lower integer
`round(x)`	Round to nearest integer

8.2.10 Elementary Mathematical Functions: Special Mathematical Functions

`besseli0(x)`	Modified Bessel function of order zero
`besselj(n,x)`	Bessel function of first kind and `n`-th order
`bessely(n,x)`	Bessel function of second kind and `n`-th order
`erf(x)`	Error function
`erfc(x)`	Complementary error function
`erfinv(x)`	Inverse error function
`erfcinv(x)`	Inverse complementary error function
`sinc(x)`	Sinc function (sin(`x`)/`x` or 1 at `x` = 0)
`step(x)`	Step function

8.2.11 Data Analysis: Basic Statistics

`avg(x[,range])`	Average of vector `x`. If range given `x` must have a single data dependency
`cumavg(x)`	Cumulative average of vector elements
`max(x,y)`	Returns the greater of the values `x` and `y`
`max(x[,range])`	Maximum of vector `x`. If range given `x` must have a single data dependency
`min(x,y)`	Returns the lesser of the values `x` and `y`
`min(x[,range])`	Minimum of vector `x`. If range is given `x` must have a single data dependency
`rms(x)`	Root Mean Square of vector elements
`runavg(x)`	Running average of vector elements
`stddev(x)`	Standard deviation of vector elements
`variance(x)`	Variance of vector elements
`random()`	Random number between 0.0 and 1.0
`srandom(x)`	Give random seed

8.2.12 Data Analysis: Basic Operation

`cumprod(x)`	Cumulative product of vector elements
`cumsum(x)`	Cumulative sum of vector elements
`interpolate(f,x[,n])`	Spline interpolation of vector `f` using `n` equidistant points of `x`
`prod(x)`	Product of vector elements
`sum(x)`	Sum of vector elements
`xvalue(f,yval)`	Returns x-value nearest to `yval` in single dependency vector `f`
`yvalue(f,xval)`	Returns y-value nearest to `xval` in single dependency vector `f`

8.2.13 Data Analysis: Differentiation and Integration

`ddx(expr,var)`	Derives mathematical expression `expr` with respect to the variable `var`
`diff(y,x[,n])`	Differentiate vector `y` with respect to vector `x` `n` times. Defaults to `n` = 1
`integrate(x,h)`	Integrate vector `x` numerically assuming a constant step-size `h`

8.2.14 Data Analysis: Signal Processing

`dft(x)`	Discrete Fourier Transform of vector `x`
`fft(x)`	Fast Fourier Transform of vector `x`
`fftshift(x)`	Shuffles the FFT values of vector `x` to move DC to the center of the vector
`Freq2Time(V,f)`	Inverse Discrete Fourier Transform of function `V(f)` interpreting it physically
`idft(x)`	Inverse Discrete Fourier Transform of vector `x`
`ifft(x)`	Inverse Fast Fourier Transform of vector `x`
`kbd(x[,n])`	Kaiser-Bessel derived window
`Time2Freq(v,t)`	Discrete Fourier Transform of function `v(t)` interpreting it physically

8.3 Electronics Functions

8.3.1 Unit Conversion

`dB(x)`	dB value
`dbm(x)`	Convert voltage to power in dBm
`dbm2w(x)`	Convert power in dBm to power in Watts
`w2dbm(x)`	Convert power in Watts to power in dBm
`vt(t)`	Thermal voltage for a given temperature `t` in Kelvin

8.3.2 Reflection Coefficients and VSWR

`rtoswr(x)`	Converts reflection coefficient to voltage standing wave ratio (VSWR)
`rtoy(x[,zref])`	Converts reflection coefficient to admittance; default `zref` = 50 ohms
`rtoz(x[,zref])`	Converts reflection coefficient to impedance; default `zref` = 50 ohms
`ytor(x[,zref])`	Converts admittance to reflection coefficient; default `zref` = 50 ohms
`ztor(x[,zref])`	Converts impedance to reflection coefficient; default `zref` = 50 ohms

8.3.3 N-Port Matrix Conversions

`stos(s,zref[,z0])`	Converts S-parameter matrix to S-parameter matrix with a different Z0
`stoy(s[,zref])`	Converts S-parameter matrix to Y-parameter matrix
`stoz(s[,zref])`	Converts S-parameter matrix to Z-parameter matrix
`twoport(m,from,to)`	Converts a two-port matrix: `from` and `to` are 'Y', 'Z', 'H', 'G', 'A', 'S' and 'T'.
`ytos(y[,z0])`	Converts Y-parameter matrix to S-parameter matrix
`ytoz(y)`	Converts Y-parameter matrix to Z-parameter matrix
`ztos(z[,z0])`	Converts Z-parameter matrix to S-parameter matrix
`ztoy(z)`	Converts Z-parameter matrix to Y-parameter matrix

8.3.4 Amplifiers

`GaCircle(s,Ga[,arcs])`	Available power gain `Ga` circles (source plane)
`GpCircle(s,Gp[,arcs])`	Operating power gain `Gp` circles (load plane)
`Mu(s)`	Mu stability factor of a two-port S-parameter matrix
`Mu2(s)`	Mu' stability factor of a two-port S-parameter matrix
`NoiseCircle(Sopt,Fmin,Rn,F[,Arcs])`	Noise Figure(s) `F` circles
`PlotVs(data,dep)`	Returns data selected from `data`: dependency `dep`
`Rollet(s)`	Rollet stability factor of a two-port S-parameter matrix
`StabCircleL(s[,arcs])`	Stability circle in the load plane
`StabCircleS(s[,arcs])`	Stability circle in the source plane
`StabFactor(s)`	Stability factor of a two-port S-parameter matrix
`StabMeasure(s)`	Stability measure B1 of a two-port S-parameter matrix

8.4 Nomenclature

8.4.1 Ranges

`LO:HI`	Range from `LO` to `HI`
`:HI`	Up to `HI`
`LO:`	From `LO`
`:`	No range limitations

8.4.2 Matrices and Matrix Elements

`M`	The whole matrix `M`
`M[2,3]`	Element being in 2nd row and 3rd column of matrix `M`
`M[:,3]`	Vector consisting of 3rd column of matrix `M`

8.4.3 Immediate

`2.5`	Real number
`1.4+j5.1`	Complex number
`[1,3,5,7]`	Vector
`[11,12;21,22]`	Matrix

8.4.4 Number suffixes

E	exa, 1e+18
P	peta, 1e+15
T	tera, 1e+12
G	giga, 1e+9
M	mega, 1e+6
k	kilo, 1e+3
m	milli, 1e-3
u	micro, 1e-6
n	nano, 1e-9
p	pico, 1e-12
f	femto, 1e-15
a	atto, 1e-18

8.4.5 Name of Values

S[1,1]	S-parameter value
nodename.V	DC voltage at node *nodename*
name.I	DC current through component *name*
nodename.v	AC voltage at node *nodename*
name.i	AC current through component *name*
nodename.vn	AC noise voltage at node *nodename*
name.in	AC noise current through component *name*
nodename.Vt	Transient voltage at node *nodename*
name.It	Transient current through component *name*

Note: All voltages and currents are peak values. Note: Noise voltages are RMS values at 1 Hz bandwidth.

8.5 Constants

i, j	Imaginary unit (“square root of -1”)
pi	4*arctan(1) = 3.14159...
e	Euler = 2.71828...
kB	Boltzmann constant = 1.38065e-23 J/K
q	Elementary charge = 1.6021765e-19 C

CHAPTER 9

List of Special Characters

It is possible to use special characters in the text painting and in the text of the diagram axis labels. This is done by using LaTeX tags. The following table contains a list of currently available characters.

Note: Which of those characters are correctly displayed depends on the font used by Qucs!

Small Greek letters

LaTeX tag	Unicode	Description
\alpha	0x03B1	alpha
\beta	0x03B2	beta
\gamma	0x03B3	gamma
\delta	0x03B4	delta
\epsilon	0x03B5	epsilon
\zeta	0x03B6	zeta
\eta	0x03B7	eta
\theta	0x03B8	theta
\iota	0x03B9	iota
\kappa	0x03BA	kappa
\lambda	0x03BB	lambda
\mu	0x03BC	mu
\textmu	0x00B5	mu
\nu	0x03BD	nu
\xi	0x03BE	xi
\pi	0x03C0	pi
\varpi	0x03D6	pi
\rho	0x03C1	rho
\varrho	0x03F1	rho
\sigma	0x03C3	sigma
\tau	0x03C4	tau
\upsilon	0x03C5	upsilon
\phi	0x03C6	phi
\chi	0x03C7	chi
\psi	0x03C8	psi
\omega	0x03C9	omega

Capital Greek letters

LaTeX tag	Unicode	Description
\Gamma	0x0393	Gamma
\Delta	0x0394	Delta
\Theta	0x0398	Theta
\Lambda	0x039B	Lambda
\Xi	0x039E	Xi
\Pi	0x03A0	Pi
\Sigma	0x03A3	Sigma
\Upsilon	0x03A5	Upsilon
\Phi	0x03A6	Phi
\Psi	0x03A8	Psi
\Omega	0x03A9	Omega

Mathematical symbols

LaTeX tag	Unicode	Description
\cdot	0x00B7	multiplication dot (centered dot)
\times	0x00D7	multiplication cross
\pm	0x00B1	plus minus sign
\mp	0x2213	minus plus sign
\partial	0x2202	partial differentiation symbol
\nabla	0x2207	nabla operator
\infty	0x221E	infinity symbol
\int	0x222B	integral symbol
\approx	0x2248	approximation symbol (waved equal sign)
\neq	0x2260	not equal sign
\in	0x220A	“contained in” symbol
\leq	0x2264	less-equal sign
\geq	0x2265	greater-equal sign
\sim	0x223C	(central european) proportional sign
\propto	0x221D	(american) proportional sign
\diameter	0x00F8	diameter sign (also sign for average)
\onehalf	0x00BD	one half
\onequarter	0x00BC	one quarter
\twosuperior	0x00B2	square (power of two)
\threesuperior	0x00B3	power of three
\ohm	0x03A9	unit for resistance (capital Greek omega)

CHAPTER 10

Matching Circuits

Creating matching circuits is an often needed task in microwave technology. Qucs can do this automatically. These are the neccessary steps:

Perform an S-parameter simulation in order to calculate the reflexion coefficient.

Place a diagram and display the reflexion coefficient (i.e. S[1,1] for port 1, S[2,2] for port 2 etc.)

Set a marker on the graph and step to the desired frequency.

Click with the right mouse button on the marker and select “power matching” in the appearing menu.

A dialog appears where the values can be adjusted, for example the reference impedance can be chosen different from 50 ohms.

After clicking “create” the page switches back to the schematic and by moving the mouse cursor the matching circuit can be placed.

The left-hand side of the matching circuit is the input and the right-hand side must be connected to the circuit.

If the marker points to a variable called “Sopt”, the menu shows the option “noise matching”. Note that the only different to “power matching” is the fact that the conjugate complex reflexion coefficient is taken. So if the variable has another name, noise matching can be chosen by re-adjusting the values in the dialog.

The matching dialog can also be called by menu (Tools->matching circuit) or by short-cut (<CTRL-5>). But then all values has to be entered manually.

10.1 2-Port Matching Circuits

If the variable name in the marker text is an S-parameter, then an option exists for concurrently matching input and output of a two-port circuit. This works quite alike the above-mentioned steps. It results in two L-circuits: The very left node is for connecting port 1, the very right node is for connectiong port 2 and the two node in the middle are for connecting the two-port circuit.

CHAPTER 11

Installed Files

The Qucs system needs several programs. These are installed during the installation process. The path of Qucs is determined during the installation (`configure` script). The following explanations assume the default path (`/usr/local/`).

- `/usr/local/bin/qucs` - the GUI
- `/usr/local/bin/qucsator` - the simulator (console application)
- `/usr/local/bin/qucsedit` - a simple text editor
- `/usr/local/bin/qucshelp` - a small program displaying the help system
- `/usr/local/bin/qucstrans` - a program for calculation transmission line parameters
- `/usr/local/bin/qucsfilter` - a program synthesizing filter circuits
- `/usr/local/bin/qucsconv` - a file format converter (console application)

All programs are stand-alone applications and can be started independently. The main program (GUI)

- calls `qucsator` when performing a simulation,
- calls `qucsedit` when showing text files,
- calls `qucshelp` when showing the help system,
- calls `qucstrans` when calling this program from menu “Tools”,
- calls `qucsfilter` when calling this program from menu “Tools”,
- calls `qucsconv` when placing the SPICE component and when performing a simulation with the SPICE component.

Furthermore, the following directories are created during installation:

- `/usr/local/share/qucs/bitmaps` - contains all bitmaps (icons etc.)
- `/usr/local/share/qucs/docs` - contains HTML documents for the help system
- `/usr/local/share/qucs/lang` - contains the translation files

11.1 Command line arguments

`qucs [file1 [file2 ...]]`

`qucsator [-b] -i netlist -o dataset` (b = progress bar)

`qucsedit [-r] [file]` (r = read-only)

`qucshelp` (no arguments)

`qucsconv -if spice -of qucs -i netlist.inp -o netlist.net`

CHAPTER 12

Qucs File Formats

This document describes the schematic and library file formats of Qucs.

12.1 Schematic file format

This format is used for schematics (usually with suffix `.sch`) and for data displays (usually with suffix `.dpl`). The following text shows a short example of a schematic file.

```
<Qucs Schematic 0.0.6>
<Properties>
  <View=0,0,800,800,1,0,0>
</Properties>
<Symbol>
  <.ID -20 14 SUB>
</Symbol>
<Components>
  <R R1 1 180 150 15 -26 0 1 "50 Ohm" 1 "26.85" 0 "european" 0>
  <GND * 1 180 180 0 0 0 0>
</Components>
<Wires>
  <180 100 180 120 "" 0 0 0 "">
  <120 100 180 100 "Input" 170 70 21 "">
</Wires>
<Diagrams>
  <Polar 300 250 200 200 1 #c0c0c0 1 00 1 0 1 1 1 0 5 15 1 0 1 1 315 0 225 "" "" "">
    <"acnoise2:S[2,1]" #0000ff 0 3 0 0 0>
    <Mkr 6e+09 118 -195 3 0 0>
  </Polar>
</Diagrams>
<Paintings>
  <Arrow 210 320 50 -100 20 8 #000000 0 1>
</Paintings>
```

The file contains several section. Each of it is explained below. Every line consists of not more than one information block that starts with a less-sign < and ends with a greater-sign >.

12.1.1 Properties

The first section starts with `<Properties>` and ends with `</Properties>`. It contains the document properties of the file. Each line is optional. The following properties are supported:

- `<View=x1,y1,x2,y2,scale,xpos,ypos>` contains pixel position of the schematic window in the first four numbers, its current scale and the current position of the upper left corner (last two numbers).
- `<Grid=x,y,on>` contains the distance of the grid in pixel (first two numbers) and whether grid is on (last number 1) or off (last number 0).
- `<DataSet=name.dat>` contains the file name of the data set associated with this schematic.
- `<DataDisplay=name.dpl>` contains the file name of the data display page associated with this schematic (or the file name of the schematic if this document is a data display).
- `<OpenDisplay=yes>` contains 1 if the data display page opens automatically after simulation, otherwise contains 0.
- `<Script=name.m>` contains the file name of the octave script associated with this schematic.
- `<RunScript=0>` contains 1 if the octave script is executed after the simulation.
- `<showFrame=0>` specify if a frame is drawn and if so which size it is. valid values are 0 (do not show a frame), 1 (A5 landscape), 2 (A5 portrait), 3 (A4 landscape), 4 (A4 portrait), 5 (A3 landscape), 6 (A3 portrait), 7 (letter landscape) and 8 (letter portrait).
- `<FrameText0=NE555 sub-circuit model>`, `FrameText1=Draw by: anonymous`, `FrameText2=Date: 1984`, and `<FrameText3=Revision: 42>` specifiy the texts to be placed into the frame text boxes.

12.1.2 Symbol

This section starts with `<Symbol>` and ends with `</Symbol>`. It contains painting elements creating a schematic symbol for the file. This is usually only used for schematic files that meant to be a subcircuit.

Refers to “Symbol definition” in the “Shared file format” section at the end of this document.

12.1.3 Components

This section starts with `<Components>` and ends with `</Components>`. It contains the circuit components of the schematic. The line format is as follows:

```
<type name active x y xtext ytext mirrorX rotate "Value1" visible "Value2" visible ...
↪>
```

- The `type` identifies the component, e.g. `R` for a resistor, `C` for a capacitor.
- The `name` is the unique component identifier of the schematic, e.g. `R1` for the first resistor.
- A `1` in the `active` field shows that the component is active, i.e it is used in the simulation. A `0` shows it is inactive.
- The next two numbers are the x and y coordinates of the component center.

- The next two numbers are the x and y coordinates of the upper left corner of the component text. They are relative to the component center.
- The next two numbers indicate the mirroring about the x axis (`1` for mirrored, `0` for not mirrored) and the counter-clockwise rotation (multiple of 90 degree, i.e. 0...3).
- The next entries are the values of the component properties (in quotation marks) followed by an `1` if the property is visible on the schematic (otherwise `0`).

12.1.4 Wires

This section starts with `<Wires>` and ends with `</Wires>`. It contains the wires (electrical connection between circuit components) and their labels and node sets. The line format is as follows:

```
<x1 y1 x2 y2 "label" xlabel ylabel dlabel "node set">
```

- The first four numbers are the coordinates of the wire in pixels: x coordinate of starting point, y coordinate of starting point, x coordinate of end point and y coordinate of end point. All wires must be either horizontal (both x coordinates equal) or vertical (both y coordinates equal).
- The first string in quotation marks is the label name. It is empty if the user has not set a label on this wire.
- The next two numbers are the x and y coordinates of the label or zero if no label exists.
- The next number is the distance between the wire starting point and the point where the label is set on the wire.
- The last string in quotation marks is the node set of the wire, i.e. the initial voltage at this node used by the simulation engine to find the solution. This is empty if the user has not set a node set for this wire.

12.1.5 Diagrams

This section starts with `<Diagrams>` and ends with `</Diagrams>`. It contains the diagrams with their graphs and their markers. The line format is as follows (line break not allowed):

```
<diatype x y width height grid gridcolor gridstyle log xAutoscale xmin
 xstep xmax yAutoscale ymin ystep ymax zAutoscale zmin zstep zmax
 xrotate yrotate zrotate "xlabel" "ylabel" "zlabel" "[freq Hz;]*">
    <"graphvar" color thickness precision numberformat style axisside>
    <Mkr x y precision numberformat transparent>
</diatype>
```

Diagram line format:

- The `diatype` token specifies the type of diagram.
- The `x` and `y` numbers are the coordinate of lower left corner.
- The `width` and `height` numbers of diagram boundings.
- The `grid` flags with 1 if grid is on and 0 if grid is off.
- The `gridColor` in 24 bit hexadecimal RGB value, e.g. #FF0000 is red.
- The `gridstyle` is the line style sued of the grid.
- The `log` has two field to flag which axes have logarithmical scale.
- The `xAutoscale`, `xmin`, `xstep`, `xmax` configure the x-axis scaling, limits.
- The `yAutoscale`, `ymin`, `ystep`, `ymax` configure the y-axis scaling, limits.

- The `zAutoscale`, `zmin`, `zstep`, `zmax` configure the z-axis scaling, limits.
- The `xrotate`, `yrotate`, `zrotate` numbers set the 3D rotation.
- The `xlabel`, `ylabel`, `zlabel` hold the labels used on each axis.
- The list of frequencies `"[freq Hz;]*"` is used by `Phasor` and `Waveac`.

Here is a list of known diagram types:

- `Curve` for a locus curve diagram.
- `Smith` for an impedance Smith diagram.
- `ySmith` for an admittance Smith diagram.
- `PS` for a mixed polar/smith diagram.
- `SP` for a upper-half mixed polar/smith diagram.
- `Polar` for a polar diagram.
- `Rect` for a 2D-cartesian diagram.
- `Rect3D` for a 3D-cartesain diagram.
- `Tab` for a tabular diagram.
- `Time` for a timing diagram.
- `Truth` for a truth-table diagram.
- `Phasor` for a complex phasor diagram.
- `Waveac` for a wave as temporal diagram.

Graph line format:

- The `graphvar` specify the variable this graph is plotting for.
- The `color`, `thickness` and `style` refers to the pen used to draw the curve.
- The `precision` specify the number of digits used when displaying data values.
- The `numberformat` is an integer that specify how the number are formated (0 for real/imag, 1 for polar/deg and 2 for polar/rad).
- The `axisside` is an integer indicating on which side the Y axis should be placed ().

Marker line format:

- The `x` and `y` are the location of the marker.
- The `precision` ...
- The `numberformat` ...
- The `transparent`

12.1.6 Paintings

This section starts with `<Paintings>` and ends with `</Paintings>`. It contains the paintings that are within the schematic.

Refers to “Shared file format” section below.

12.2 Library file format

This format is used for libraries (usually with suffix `.lib`). The following text shows a short example of a library file.

```
<Qucs Library 0.0.14 "Ideal">
<DefaultSymbol>
  <.ID -26 13 D>
  <Line -30 0 60 0 #000080 2 1>
  <Line -6 -9 0 18 #000080 2 1>
  <Line 6 -9 0 18 #000080 2 1>
  <Line -6 0 12 -9 #000080 2 1>
  <Line -6 0 12 9 #000080 2 1>
  <Line -6 9 4 0 #000080 2 1>
  <.PortSym -30 0 1 0>
  <.PortSym 30 0 2 180>
</DefaultSymbol>
<Component VSum>
  <Description>
Voltage adder
  </Description>
  <Model>
.Def:Ideal_AP1 _net3 _net2 fc="1E3"
Sub:VSUB1 _net0 _net1 _net2 Type="VSub"
Sub:LP1F1 _net3 _net0 Type="LP1" fc="fc2" V0="0"
Sub:HP1F1 _net3 _net1 Type="HP1" fc="fc2"
Eqn:Eqn1 fc2="fc/0.6436" Export="yes"
.Def:End
  </Model>
  <ModelIncludes "HP1.sch.lst" "LP1.sch.lst" "VSub.sch.lst">
  <Symbol>
    <Ellipse -20 -20 40 40 #000080 2 1 #c0c0c0 1 0>
    <Line -10 0 20 0 #000080 1 1>
    <Line 0 -10 0 20 #000080 1 1>
    <Line 0 30 0 -10 #000080 2 1>
    <.PortSym 0 30 2 0>
    <.PortSym 30 0 3 180>
    <Line 20 0 10 0 #000080 2 1>
    <.ID 10 14 VADD>
    <Line 0 -20 0 -10 #000080 2 1>
    <.PortSym 0 -30 1 0>
  </Symbol>
</Component>
```

The first line specify that this file is a Qucs library file generated by Qucs 0.0.14 and that the library is named “Ideal”.

The file contains on optional `DefaultSymbol` section, followed by `Component` sections. Each section is explained below.

12.2.1 Default symbol

This section starts with `<DefaultSymbol>` and ends with `</DefaultSymbol>`. It contains painting elements creating a default schematic symbol for any subsequent component declaration that doesn’t define its own.

Refers to “Shared file format” section below.

12.2.2 Component

This section starts with `<Component>` and ends with `</Component>`. It contains the component definition for use with schematic documents.

The component section is an aggregation of the following sub-sections:

- `<Description>` and `</Description>` contain lines of free text describing the component function.
- `<Model>` and `</Model>` contain the Qucsator netlist lines for this component.
- `<ModelIncludes "value0" "value1" ...>` ...
- `<Spice>` and `</Spice>`` are optional and contain the Spice netlist lines for this component.
- `<Symbol>` and `</Symbol>` are optional and contain painting elements defining the schematic symbol to be used with this component. Refers to "Symbol definition" section below.

12.3 Shared file format

12.3.1 Painting elements

A painting line can be found in:

- The `Paintings` section of a schematic file.
- The `Symbol` sections of a schematic file.
- The `DefaultSymbol` section of a library file.
- The `Symbol` section (sub-section of `Component`) of a library file.

A painting line has one of the following format:

- `<Rectangle x y width height pencolor penwidth penstyle brushcolor brushstyle filled>` ...
- `<Ellipse x y width height pencolor penwidth penstyle brushcolor brushstyle filled>` ...
- `<EArc x y startangle spanangle width height pencolor penwidth penstyle brushcolor brushstyle filled>` ...
- `<Text x y size color angle "text">` ...
- `<Line x1 y1 x2 y2 pencolor penwidth penstyle >` ...
- `<Arrow x1 y1 x2 y2 x3 y3 pencolor penwidth penstyle >` ...

12.3.2 Symbol definition

A symbol definition can contains any painting element as described in the previous section. In addition to the painting elements, a symbol definition must contain one `.ID` line and one or more `.PortSym` lines.

The `.ID` line has the following format:

```
<.ID x y name "property1" "property2" ...>
```

Where:

- `x` and `y` are the center coordinates of the symbol.

- `name` will be used as a name prefix when instanciating this symbol on a schematic sheet.
- `propertyX` are used for symbol definition within a schematic file, these parameter will be associated with the symbol instance and communicated to the sub-schematic. The format for such a property is `` `displayed=name=value=description=unknown` ``.

The `.PortSym` line has the following format:

```
<.PortSym x y caption angle>
```

Where:

- `x` and `y` are the coordinates of the port.
- `caption` is the name/caption of the port.
- `angle` is an angle value, it is ignored (backward compatibility).

CHAPTER 13

Subcircuit and Verilog-A RF Circuit Models for Axial and Surface Mounted Resistors

Mike Brinson

13.1 Introduction

Resistors are one of the fundamental building blocks in electronic circuit design. In most instances conventional resistor circuit simulation models are characterized by I/V characteristics specified by Ohm's law. In reality the impedance of RF resistors is frequency dependent, being determined by component physical properties, component manufacturing technology and how components are connected in a circuit. At low frequencies fixed resistors have a nominal value at roomtemperature and can be modelled accurately by Ohm's law. At RF frequencies the fact that a resistor acts more like an inductance or a capacitance can play a crucial role in determining whether or not a circuit operates as designed. Similarly, if a resistor is modelled as an ideal component at a frequency where it exhibits significant reactive properties then the resulting simulation data are likely to be incorrect. The subcircuit and Verilog-A compact resistor models introduced in this Qucs note are designed to give good performance from low frequencies to RF frequencies not greater than a few GHz.

13.2 RF Resistor Models

The schematic symbol, I/V equation and parameters of the Qucs linear resistor model are shown in Figure 1. In contrast to this model Figure 2 illustrates the structure of a printed circuit board (PCB) mounted metal film (MF) axial RF resistor (a), its Qucs schematic symbol (b) and its equivalent circuit model (c). A thin film surface mounted (SMD) resistor can also be represented by the model shown in Figure 2 (c).

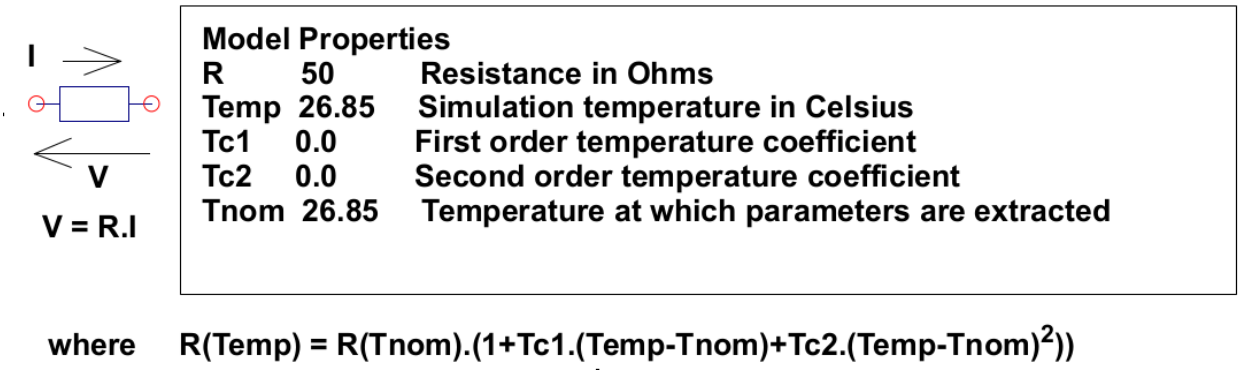


Figure 1 - Qucs built-in resistor model.

At signal frequencies where the largest dimension of an axial or SMD resistor is less than approximately 20 times the smallest signal wavelength a resistor can be modelled by a lumped passive circuit consisting of a resistor **Rs** in series with a small inductance **Ls** with the combination shunted by parasitic capacitor **Cp**. In Figure 2 **Rs** is the nominal value of resistor at its parameter extraction temperature **Tnom**, **Ls** represents the inductance associated with **Rs** where the value of **Ls** is largely determined by the trimming method employed during component manufacture to set the value of **Rs** to a specified tolerance. Similarly, capacitor **Cp** models a parasitic capacitance associated with **Rs** where the value of **Cp** is a function of the physical size of **Rs**. At RF frequencies it is important, for accurate operation, to add lead parasitic elements to the intrinsic equivalent circuit model shown within the red box draw in Figure 2. In Figure 2 **Llead** and **Cshunt** represent resistor series lead inductance and shunt capacitance to ground respectively.

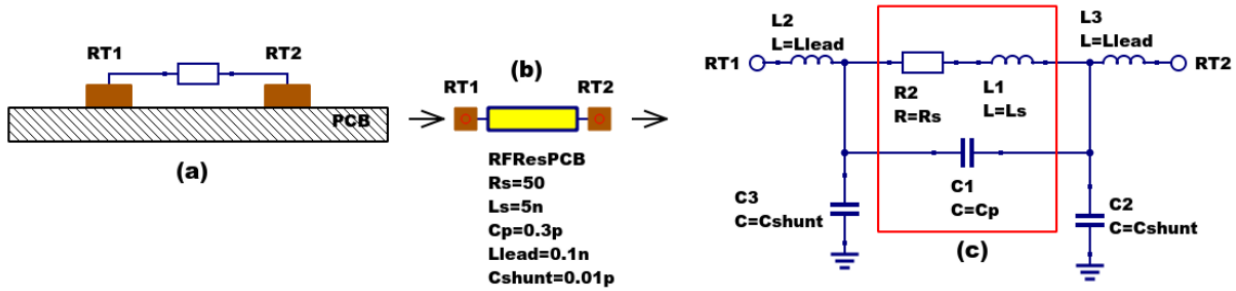


Figure 2 - PCB mounted resistor: (a) axial component mounting, (b) Qucs symbol and (c) equivalent circuit model.

A typical set of model parameters for a 51 Ω 5 % MF axial resistor are (1) **Ls = 8nH**, **Cp = 1pF**, **Llead = 1nH** and **Cshunt = 0.1pF**. Illustrated in Figure 3 is a basic S parameter test bench circuit for measuring the S parameters of an RF resistor over a frequency range 1 MHz to 1.3 GHz. This example also demonstrates how the real and imaginary parts of a resistor model impedance can be extracted from S parameter simulation data. The graphs in Figure 3 clearly demonstrate that the impedance of the typical MF RF resistor described in previous text and modelled by the equivalent circuit shown in Figure 2 is a strong function of frequency at higher frequencies in the band 1 MHz to 1.3 GHZ.

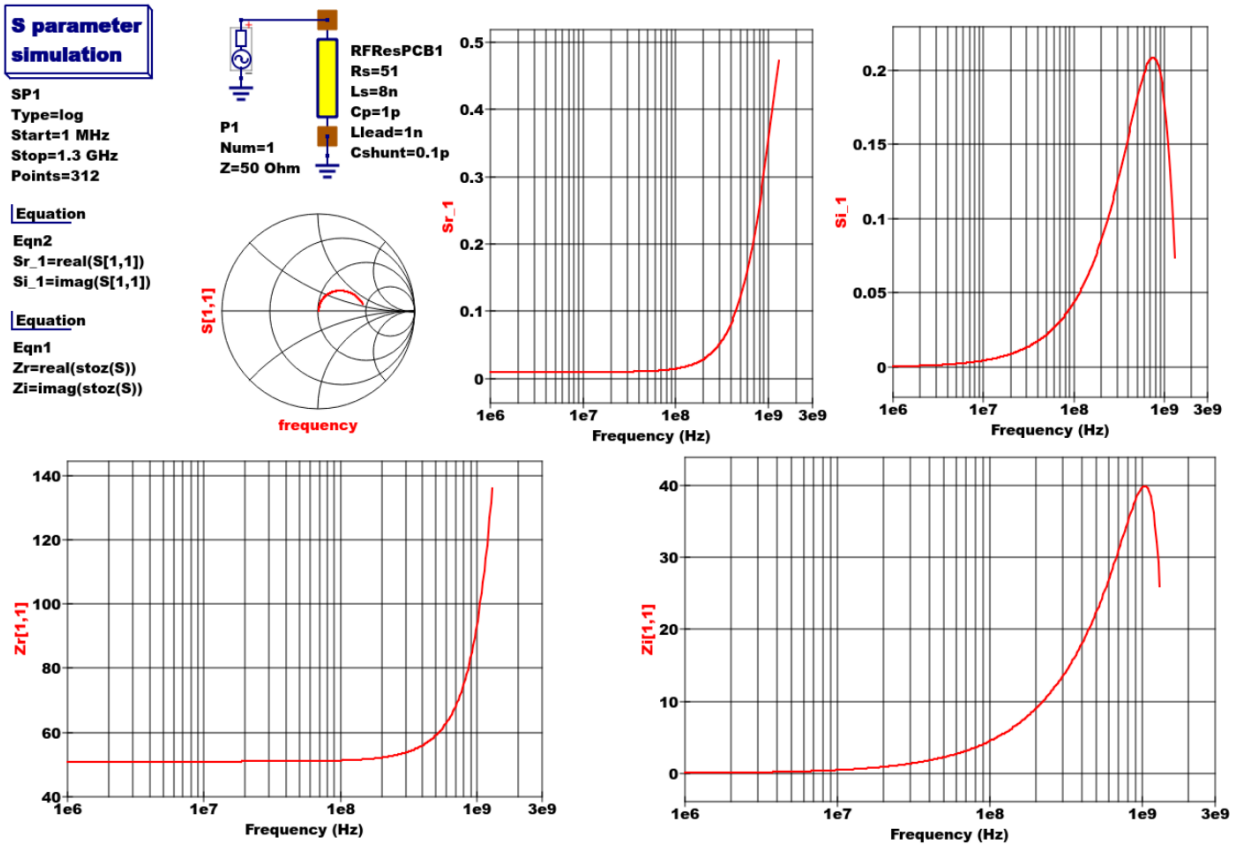


Figure 3 - Qucs S parameter simulation test circuit and plotted output data for a MF axial resistor: *Rs*=51Ω, *Ls*=8nH, *Cp*=1pF, *Llead*=1nH and *Cshunt*=0.1pF.

13.3 Analysis of the RF resistor model

A component level version of the proposed RF resistor model is shown in Figure 4, where

$$Z1 = j \cdot \omega \cdot Llead$$

$$Z2 = \frac{Rs + j \cdot \omega \cdot Ls \cdot (1 - \omega^2 \cdot Cp \cdot Ls) - j \cdot \omega \cdot Cp \cdot Rs^2}{(1 - \omega^2 \cdot Cp \cdot Ls)^2 + (\omega \cdot Cp \cdot Rs)^2}$$

$$Z3 = \frac{j \cdot \omega \cdot Llead}{(1 - \omega^2 \cdot Llead \cdot Cshunt)}$$

$$Zseries = Z1 + Z2 = Rseries + j \cdot Xseries$$

$$Zb = Zseries || XCshunt = \frac{Zseries}{(1 + j \cdot \omega \cdot Cshunt \cdot Zseries)} = ZBR + j \cdot \omega \cdot ZBI,$$

$$Z = j \cdot \omega \cdot Llead + Zb = ZR + j \cdot \omega \cdot ZI.$$

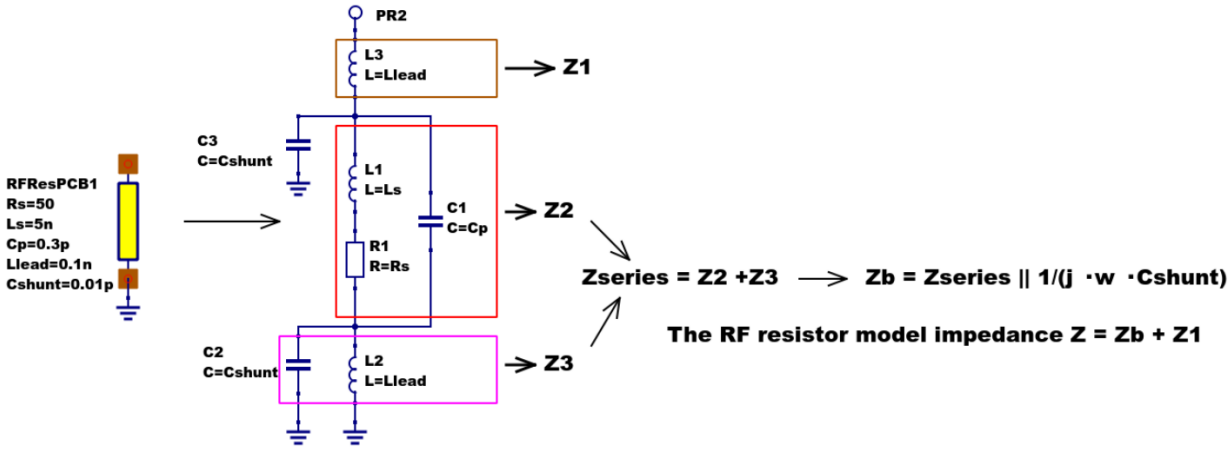


Figure 4 - RF resistor model rotated through 90 degrees and connected with one terminal grounded, similar to the test circuit in Figure. Sections of the model are shown grouped for calculation of the model impedance Z.

Figure 5 illustrates how a set of theoretical equations can be converted into Qucs equations for model simulation and post simulation data processing. In this example Qucs equation **Eqn1** holds values for RF resistor model parameters and Qucs equation **Eqn2** lists the model equations introduced at the start of this section. Figure 5 also gives a set of cartesian graphs of post simulation output data which illustrate how **ZR** and **ZI**, and other calculated items, vary with frequency over the range 1 MHz to 1.3 GHz.

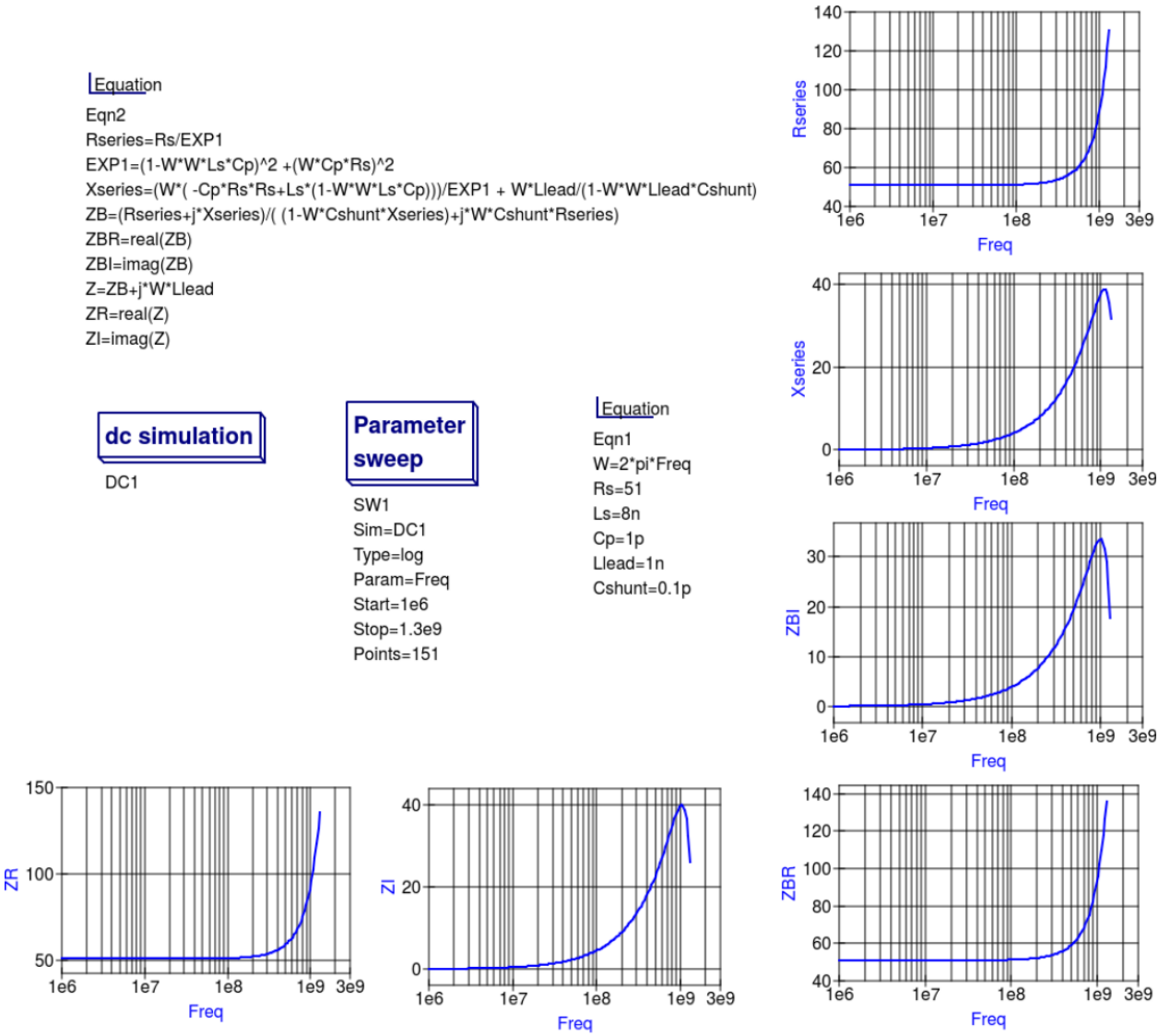


Figure 5- Theoretical analysis of RF resistance impedance Z using Qucs post processing facilities: note a dummy simulation icon, in this example DC simulation, is required to force Qucs to complete the analysis calculations.

13.4 Direct measurement of RF resistor impedance using a simulated impedance meter

A simple impedance meter for measuring the real and imaginary components of component and circuit impedance, using small signal AC simulation, is shown in Figure 6. The impedance measuring technique uses a 1 Amp AC constant current source applied to one terminal of a two port electrical network. The second terminal is grounded. A parallel high resistance resistor (1E9 Ω in Figure 6) shunts the network under measurement to ensure that there is always a direct current path to ground as required by the Qucs simulator during the calculation of simulation results. If required the 1 Amp AC source can be set at a lower value. In such cases the value of **VRes** must also be scaled to give the network impedance.

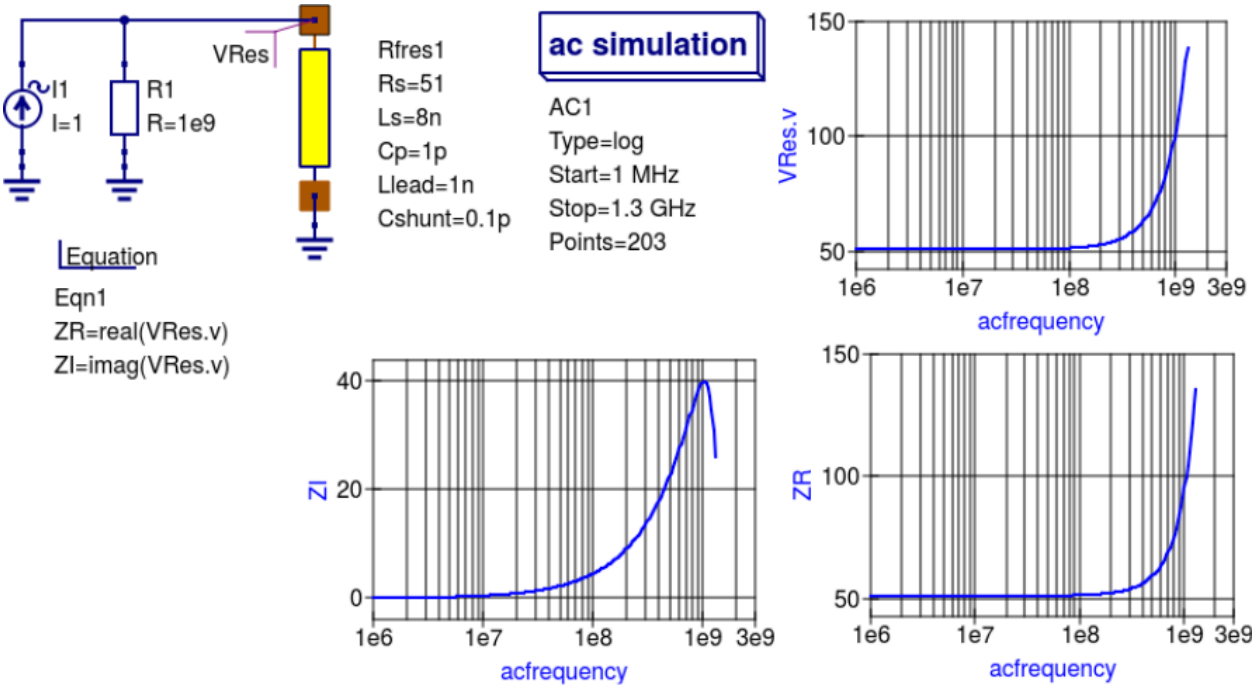


Figure 6 -A simple Qucs test circuit for demonstrating the use of an AC constant current source to measure electrical network impedance.

13.5 Extraction of RF resistance data from measured S parameters

In the past the cost of Vector Network Analyser systems for measuring S parameters has been prohibitively expensive for individual engineers to purchase. However, this scene is changing with the introduction of low cost systems like the DGSAQ Vector Network Analyser (VNWA)[1] . This instrument operates over a frequency band width of 1.3 GHz, providing a range of useful functions with highest accuracy at frequencies up to 500 MHz. This form of VNWA is particularly suited to Radio Amateur requirements and Qucs users interested in RF circuit analysis and design. Such equipment is ideal for measuring RF circuit S parameters and providing measured data for subcircuit and Verilog-A compact devicemodel parameter extraction. Shown in Figure 7 is a graph of measured S parameter data for a nominal 47 Ω resistor[2] . As well as displaying, and printing, measured data the DGSAQ Vector Network Analyser software can output data tabulated in Touchstone‘‘SnP‘‘[3] file format. These files can be read by Qucs and their contents attached to an S parameter file icon for inclusion in circuit schematic diagrams. Figure 8 shows this process as part of an RF resistor model parameter extraction technique involving DGSAQ VNWA measured S parameter data and Qucs simulated S parameter data.

The brown “Test circuits” box shows test circuits for firstly reading and processing the DGSAQ VNWA measured data listed in file mike3.s1p, and for secondly generating simulated S parameter data for an RF resistor specified by parameters **Ls =L**, **Cp = C**, **Llead = LL**, **Cshunt = 0.08 pF**, and **Rs = 47.3** Ω. Presented in Figure 9 are the Qucs Optimization controls” which are used to set the range of** L**, **C** and **LL** values that optimizer ASCO will select from to obtain the best fit between the measured and simulated S parameter data. Note in this parameter extraction system that **S[1,1]** refers to measured S parameter data and **S[2,2]** to simulated S parameter data. Two least squares cost functions called **CF1** and **CF2** are used as targets in the minimisation process. Values for **CF1** and **CF2** can be found in the red box called ‘‘Simulation Controls‘‘. In this parameter extraction example the least squares cost function **CF1** is employed to minimize the square of the difference between the real values of the S parameters and

[1] DG8SAQ VNWA 3 & 3E- Vector Network Analysers, SDR Kits Limited, Grangeside Business Centre, 129 Devizes Road, Trowbridge, Wilts, BA14-7sZ, United Kingdom, 2014.

[2] See DG8SAQ VNWA 3 & 3E- Vector Network Analysers- Getting Started Manual for Windows 7, Vista and Windows XP.

[3] (http://www.vhdl.org/ibis/connector/touchstone_spec11.pdf).

least squares cost function **CF2** is employed to minimize the square of the difference between the imaginary values of the S parameters. Qucs post-simulation processing is also used to extract values for the real and imaginary components of the RF resistor impedance. Both the S parameter data and the impedance data are displayed as graphs in Figure 8.

Notice in this example the SPICE optimizer ASCO is used to find the values of **L**, **C** and **LL** which minimize **CF1** and **CF2**. Also note that **Rs** and **Cshunt** are held at fixed values during optimization. In the case of *Rs* its nominal value can be found from DC or low frequency AC measurements. Similarly the value selected for *Cshunt* has been chosen to give a very small but representative value of the parasitic shunt capacitance.. After optimization finishes the minimized values of **L**, **C** and **LL** are given in the initial value column of the Qucs optimization Variables list, see Figure 9. For the 47 Ω resistor the post-minimization RF resistor model parameters are **Rs = 47.3** Ω, **Ls = 10.43 nH**, **Cp = 0.69 pF**, **Llead = 1.46 nH** and **Cshunt = 0.08 pF**. The theoretical simulation data illustrated in Figure 10 shows good agreement with the measured and the optimized simulation data.

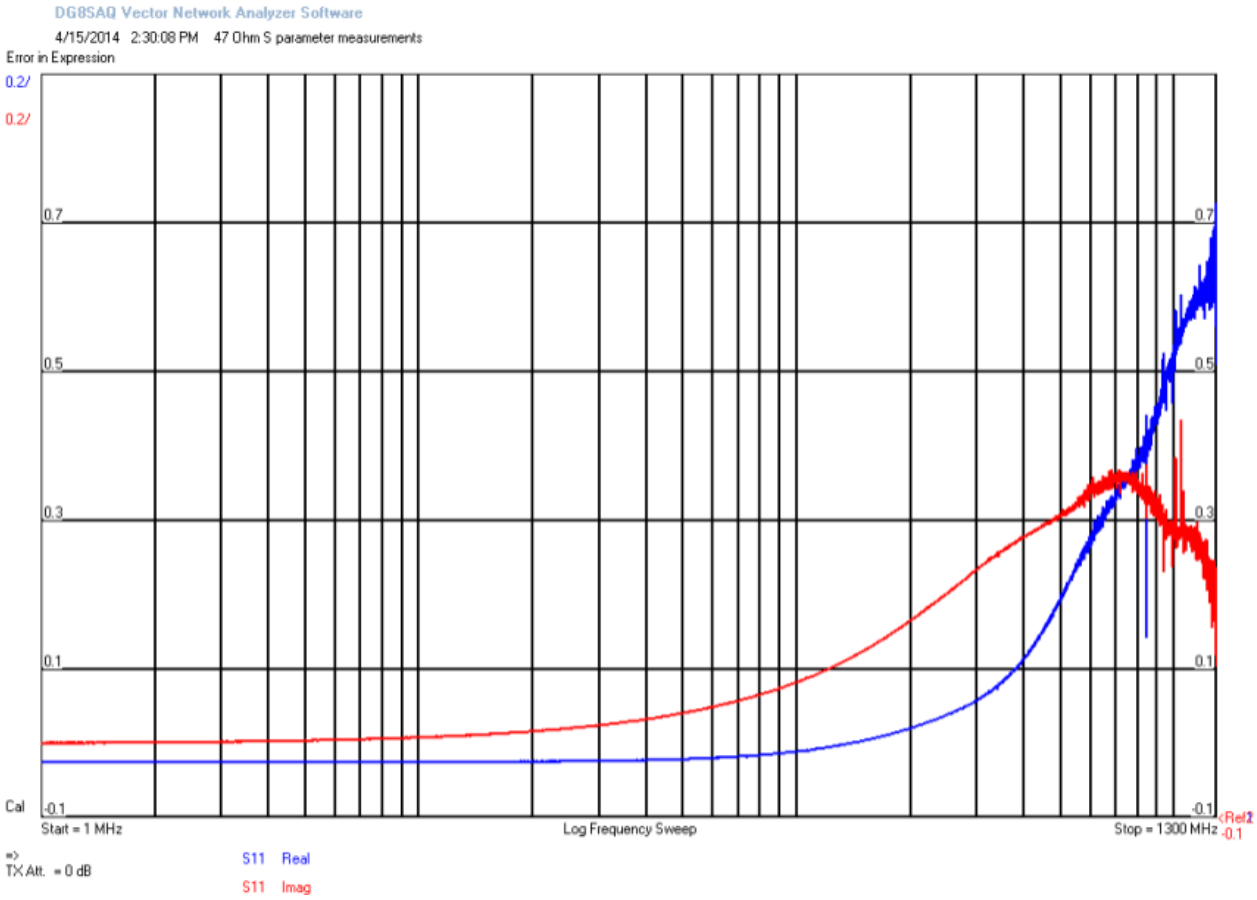


Figure 7 - DGSAQ Vector Network Analyser S parameter measurements for a 47 Ω axial RF resistor.

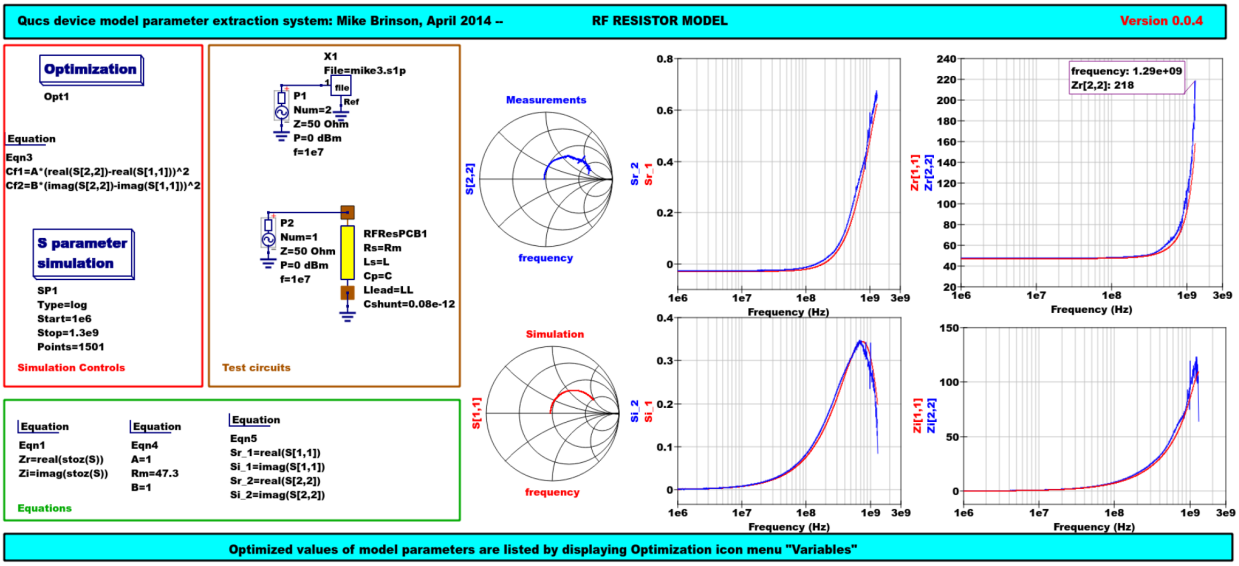


Figure 8 - Qucs device model parameter extraction system applied to a nominal 47 Ω resistor represented by the subcircuit model illustrated in Figure 2 (c). Fixed model parameter values: **Rs = Rm = 47.3** Ω, **CShunt = 0.08pF**; Optimised values: **Ls = L = 10.43nH**, **Llead = LL = 1.47nH**, **Cp = C = 0.69pF**. To reduce simulation time the ASCO cost variance was set to 1e-3. The ASCO method was set to DE/best/1/exp.

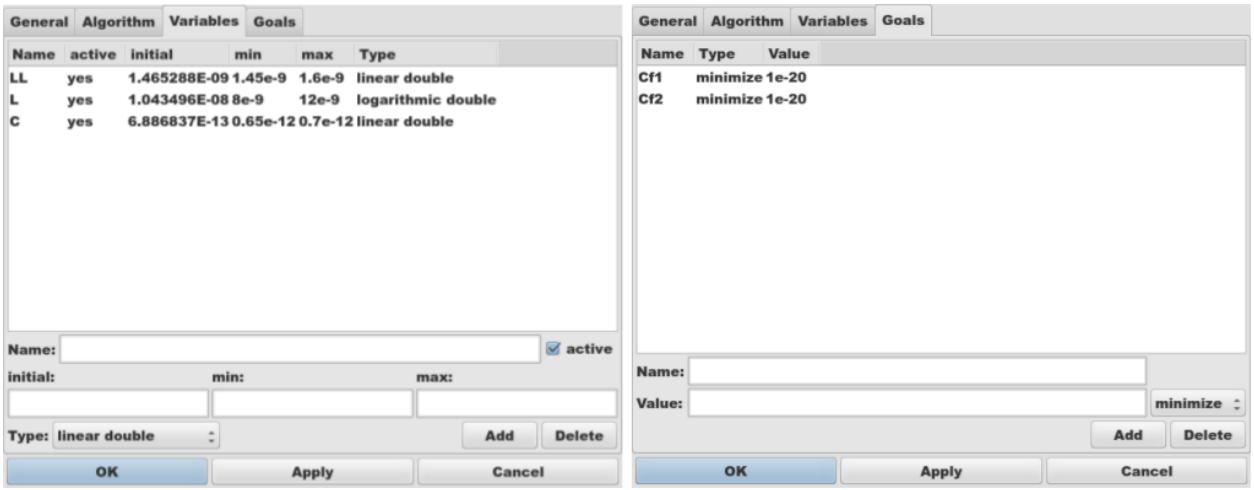


Figure 9 - Qucs Minimization Icon drop down menus: left ”Variables“ and right ”Goals‘‘.

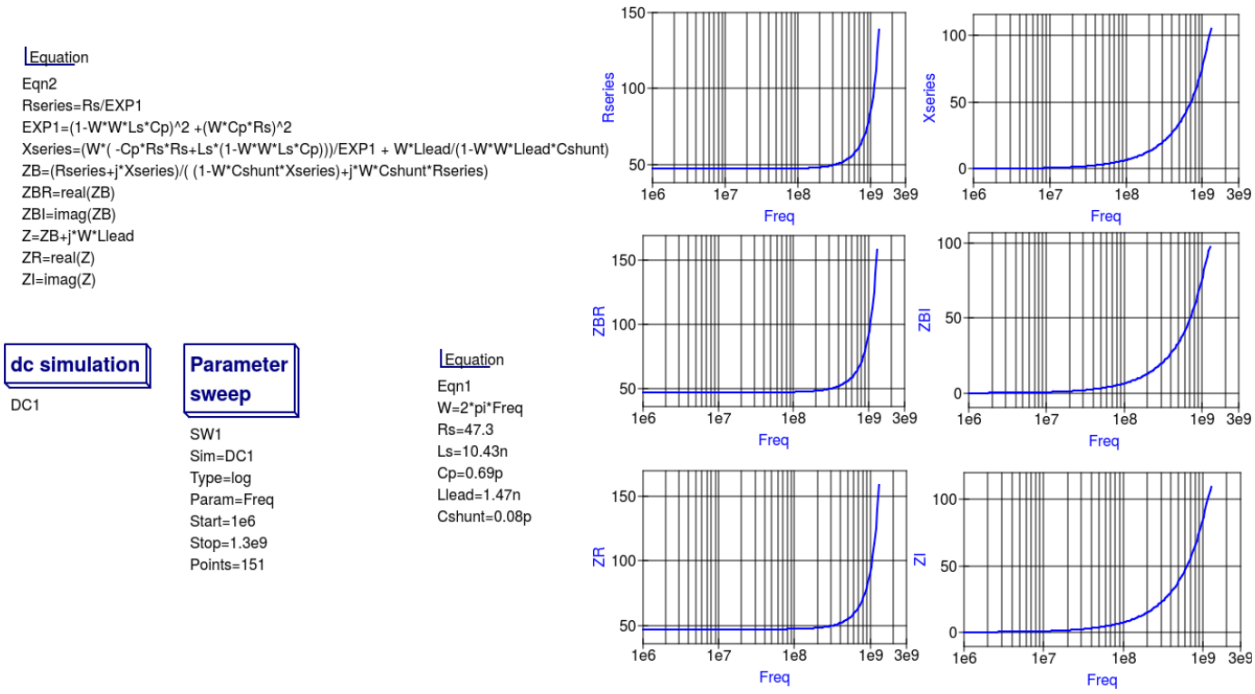


Figure 10 - Qucs simulation of nominal 47 Ω resistor based on theoretical analysis.l

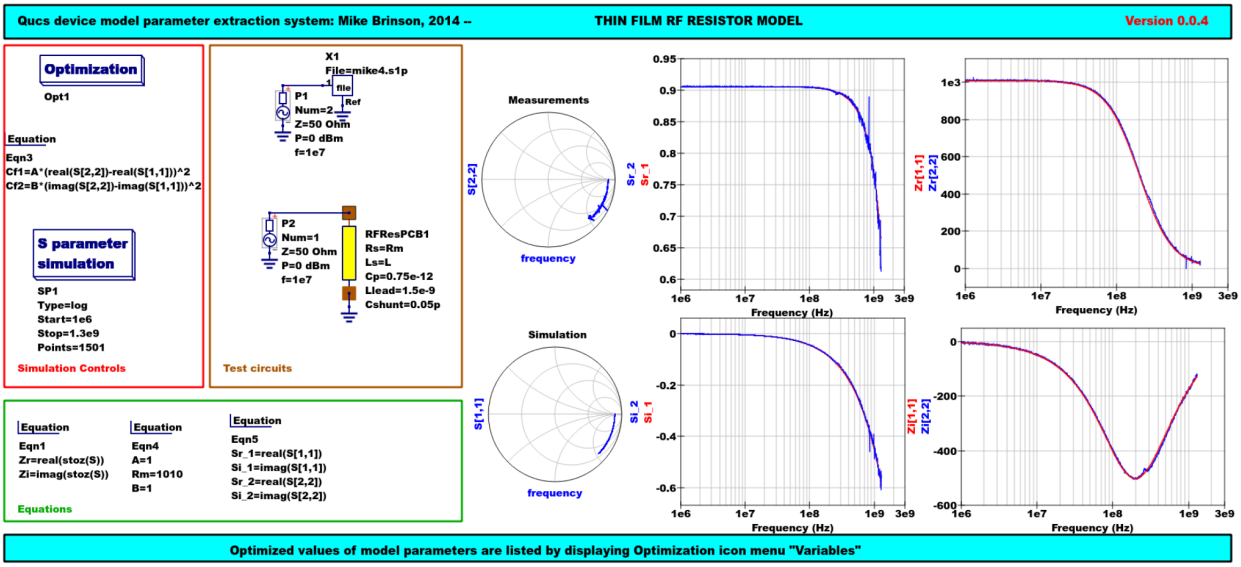


Figure 11 - Qucs device model parameter extraction system applied to a nominal 1000 Ω resistor represented by the subcircuit model illustrated in Figure 2(c).

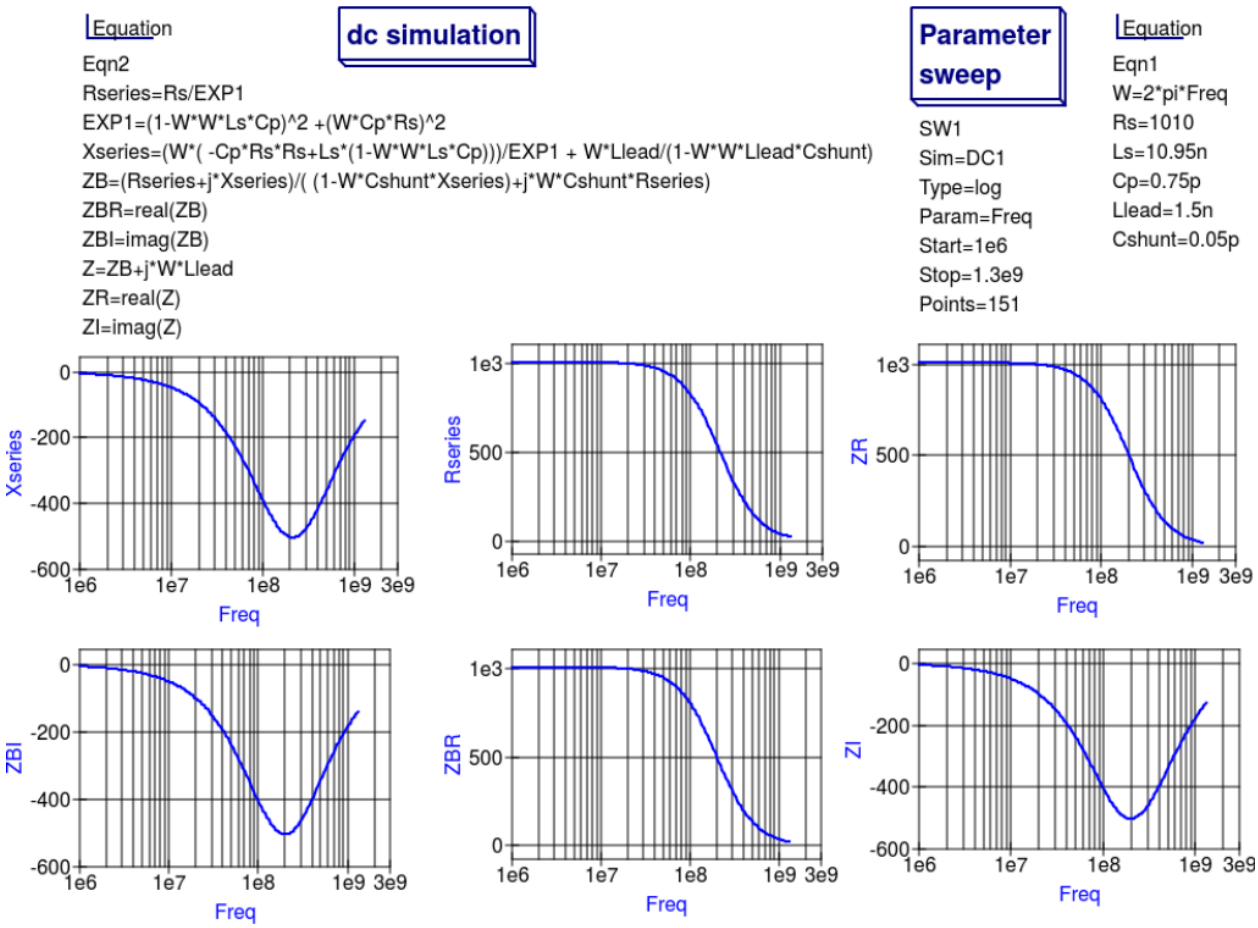


Figure 12 - Qucs simulation of nominal 1000 Ω resistor based on theoretical analysis.

13.6 Extraction of RF resistor parameters from measured S data for a nominal 1000 Ω axial resistor

At low resistance values the impedance of an RF resistor becomes inductive as the signal frequency is increased. This is due to the fact that the inductance **Ls** contribution dominates any reactance effects by **Cp**, **Llead** and **Cshunt**. However, as Rs is increased above a few hundred Ohm's the reverse becomes true with reactive effects dominated by contributions from **Cp**. Figures 11 and 12 demonstrate the dominance of **Cp** reactive effects at low to mid-range frequencies.

13.7 One more example: extraction of RF resistor parameters fro measured S data for a nominal 100 Ω SMD resistor

Figure 13 is included in this Qucs note purely for comparison purposes. SMD resistors are in general physically very small when compared to axial resistors. This results in lower values for the inductive and capacative parasitics which in turn ensures that the high frequency performance of SMD resistors is much improved.

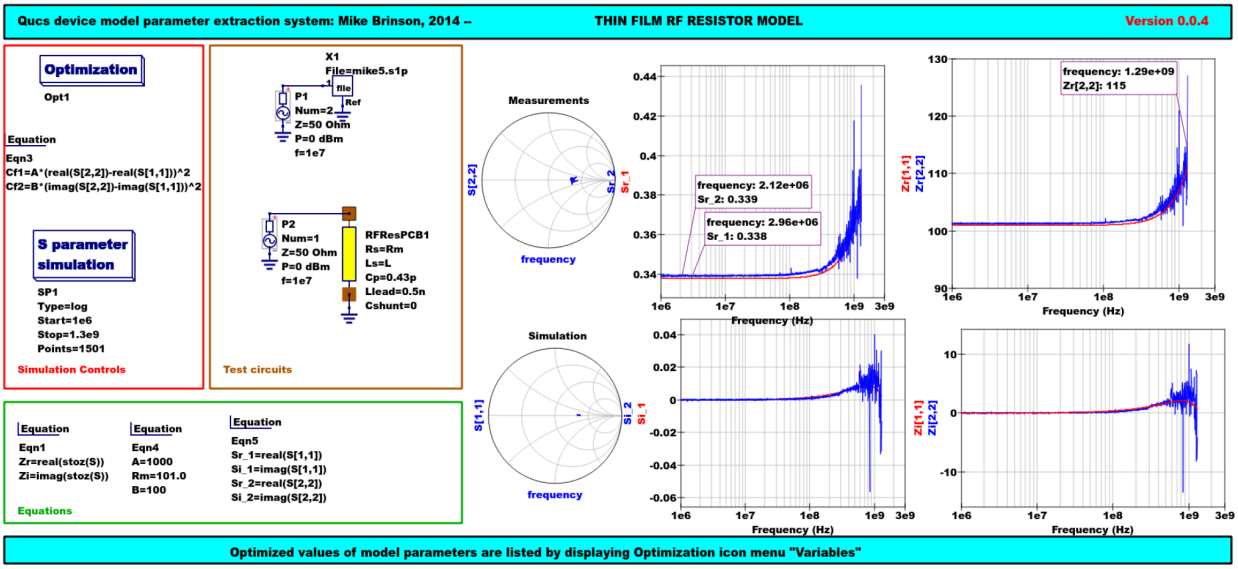


Figure 13 - Qucs device model parameter extraction system applied to a nominal 100 Ω SMD resistor represented by the subcircuit model illustrated in Figure 2 (c).

13.8 A Verilog-A RF resistor model

Listed below is an example Verilog-A code model for the RF resistor model introduced in Figure 2 (c). Due to the limitations of the Verilog-A language subset provided by version 2.3.4 of the "Analogue Device Model Synthesizer'' (ADMS)[4] inductors **Ls** and **Llead** are modelled by gyrators and capacitors with values identical to **Ls** or **Llead**.

```
//  Verilog-A module statement.
//
// RFresPCB.va  RF resistor (Thin film resistor, axial type, PCB mounting)
//
//   This is free software; you can redistribute it and/or modify
//   it under the terms of the GNU General Public License as published by
//   the Free Software Foundation; either version 2, or (at your option)
//   any later version.
//
//   Copyright (C), Mike Brinson, mbrin72043@yahoo.co.uk, April 2014.
//
`include "disciplines.vams"
`include "constants.vams"
//  Verilog-A module statement.
module RFresPCB(RT1, RT2);
inout RT1, RT2;              // Module external interface nodes.
electrical RT1, RT2;
electrical n1, n2, n3, nx, ny, nz;          // Internal nodes.
`define attr(txt) (*txt*)
parameter real Rs = 50         from [1e-20 : inf)
 `attr(info="RF resistance" unit="Ohm's");
parameter real Cp = 0.3e-12    from [0 : inf)
 `attr(info="Resistor shunt capacitance" unit="F");
parameter real Ls = 8.5e-9     from [1e-20 : inf)
 `attr(info="Series induuctance" unit="H");
parameter real Llead = 0.1e-9  from [1e-20 : inf)
```

[4] (http://sourceforge.net/projects/mot-adms/).

```
 `attr(info="Parasitic lead induuctance" unit="H");
parameter real Cshunt = 1e-10  from [1e-20 : inf)
 `attr(info="Parasitic shunt capacitance" unit="F");
parameter real Tc1 = 0.0       from [-100  : 100]
 `attr(info="First order temperature coefficient" unit ="Ohm/Celsius");
parameter real Tc2 = 0.0       from [-100  : 100]
 `attr(info="Second order temperature coefficient" unit ="(Ohm/Celsius)^2");
parameter real Tnom  = 26.85   from [-273.15 : 300]
 `attr(info="Parameter extraction temperature" unit="Celsius");
parameter real Temp  = 26.85   from [-273.15 : 300]
 `attr(info="Simulation temperature" unit="Celsius");
branch (RT1, n1)  bRT1n1;    // Branch statements
branch (n1, n2)   bn1n2;
branch (n1, n3)   bn1n3;
branch (n2, n3)   bn2n3;
branch (n3, RT2)  bn3RT2;
real Rst, FourKT, n, Tdiff, Rn;
analog begin  // Start of analog code
@(initial_model)
 begin
   Tdiff = Temp-Tnom;   FourKT =4.0*`P_K*Temp;
   Rst = Rs*(1.0+Tc1*Tdiff+Tc2*Tdiff*Tdiff);  Rn     = FourKT/Rst;
 end
 I(n1)     <+ ddt(Cshunt*V(n1));   I(bn1n2)  <+ V(bn1n2)/Rst;
 I(bn1n3)  <+ ddt(Cp*V(bn1n3));    I(n3)     <+ ddt(Cshunt*V(n3));
 I(bRT1n1)  <+ -V(nx); I(nx)      <+ V(bRT1n1);   // Llead
 I(nx)      <+ ddt(Llead*V(nx));
 I(bn2n3)   <+ -V(ny);  I(ny)      <+ V(bn2n3);  // Ls
 I(ny)      <+ ddt(Ls*V(ny));
 I(bn3RT2)  <+ -V(nz);  I(nz)      <+ V(bn3RT2);   // Llead
 I(nz)      <+ ddt(Llead*V(nz));
 I(bn1n2)    <+ white_noise(Rn, "thermal");  // Noise contribution
end  // End of analog code
endmodule
```

Module RFresPCB

Input Variables

Input Variables: instance=0 (bold) and model=9

name	description	default
Rs	RF resistance	50
Cp	Resistor shunt capacitance	0.3e-12
Ls	Series induuctance	8.5e-9
Llead	Parasitic lead induuctance	0.1e-9
Cshunt	Parasitic shunt capacitance	1e-10
Tc1	First order temperature coefficient	0.0
Tc2	Second order temperature coefficient	0.0
Tnom	Parameter extraction temperature	26.85
Temp	Simulation temperature	26.85

Output Variables

Output Variables: instance=0 (bold) and model=0 (red-underlined: temperature dependent)

name	description	dependencies

Nature/Discipline Definition

Nature

name	access	abstol	units
Current	I	1e-12	A
Charge	Q	1e-14	coul
Voltage	V	1e-6	V
Flux	Phi	1e-9	Wb
Magneto_Motive_Force	MMF	1e-12	A*turn
Temperature	Temp	1e-4	K
Power	Pwr	1e-9	W
Position	Pos	1e-6	m
Velocity	Vel	1e-6	m/s
Acceleration	Acc	1e-6	m/s^2
Impulse	Imp	1e-6	m/s^3
Force	F	1e-6	N
Angle	Theta	1e-6	rads
Angular_Velocity	Omega	1e-6	rads/s
Angular_Acceleration	Alpha	1e-6	rads/s^2
Angular_Force	Tau	1e-6	N*m

Discipline

name	potential	flow
logic		
electrical	Voltage	Current
voltage	Voltage	
current	Current	
magnetic	Magneto_Motive_Force	Flux
thermal	Temperature	Power
kinematic	Position	Force
kinematic_v	Velocity	Force
rotational	Angle	Angular_Force
rotational_omega	Angular_Velocity	Angular_Force

Model Equations

Notations used:

- *green: input parameter*
- *bar over: variable never used*
- *bar under: temperature dependent variable*
- *red: voltage dependent variable*

Initial Model

$\mathrm{Tdiff} = (\mathrm{Temp} - \mathrm{Tnom})$;

$\mathrm{FourKT} = ((4.0 \cdot 1.3806503\mathrm{e}{-23}) \cdot \mathrm{Temp})$;

$\mathrm{Rst} = (\mathrm{Rs} \cdot ((1.0 + (\mathrm{Tc1} \cdot \mathrm{Tdiff})) + ((\mathrm{Tc2} \cdot \mathrm{Tdiff}) \cdot \mathrm{Tdiff})))$;

$\mathrm{Rn} = \frac{\mathrm{FourKT}}{\mathrm{Rst}}$;

---------- end of Initial Model

$I(\mathrm{n1}, \mathrm{n1}) <+ \mathrm{ddt}((\mathrm{Cshunt} \cdot V(\mathrm{n1}, \mathrm{n1})))$;

$I(\mathrm{n1}, \mathrm{n1}) <+ \frac{V(\mathrm{n1}, \mathrm{n1})}{\mathrm{Rst}}$;

$I(\mathrm{n1}, \mathrm{n1}) <+ \mathrm{ddt}((\mathrm{Cp} \cdot V(\mathrm{n1}, \mathrm{n1})))$;

$I(\mathrm{n3}, \mathrm{n3}) <+ \mathrm{ddt}((\mathrm{Cshunt} \cdot V(\mathrm{n3}, \mathrm{n3})))$;

$I(\mathrm{RT1}, \mathrm{RT1}) <+ (-V(\mathrm{nx}, \mathrm{nx}))$;

$I(\mathrm{nx}, \mathrm{nx}) <+ V(\mathrm{RT1}, \mathrm{RT1})$;

$I(\mathrm{nx}, \mathrm{nx}) <+ \mathrm{ddt}((\mathrm{Llead} \cdot V(\mathrm{nx}, \mathrm{nx})))$;

$I(\mathrm{n2}, \mathrm{n2}) <+ (-V(\mathrm{ny}, \mathrm{ny}))$;

$I(\mathrm{ny}, \mathrm{ny}) <+ V(\mathrm{n2}, \mathrm{n2})$;

$I(\mathrm{ny}, \mathrm{ny}) <+ \mathrm{ddt}((\mathrm{Ls} \cdot V(\mathrm{ny}, \mathrm{ny})))$;

$I(\mathrm{n3}, \mathrm{n3}) <+ (-V(\mathrm{nz}, \mathrm{nz}))$;

$I(\mathrm{nz}, \mathrm{nz}) <+ V(\mathrm{n3}, \mathrm{n3})$;

$I(\mathrm{nz}, \mathrm{nz}) <+ \mathrm{ddt}((\mathrm{Llead} \cdot V(\mathrm{nz}, \mathrm{nz})))$;

$I(\mathrm{n1}, \mathrm{n1}) <+ \mathrm{white\_noise}(\mathrm{Rn}, \text{"thermal"})$;

Figure 14 - Details of the proposed RF resistor model: equations, variables and other data.

13.9 Extraction of Verilog-A RF resistor model parameters from measured S data for a 100 Ω axial resistor

This example demonstrates the use of ASCO for extracting Verilog-A model parameters from measured S parameter data. ASCO optimization yields a figure of 4nH for **L** in the model shown in Figure 2 (c). Other model parameter values are given with the test circuit, see Figure 15.

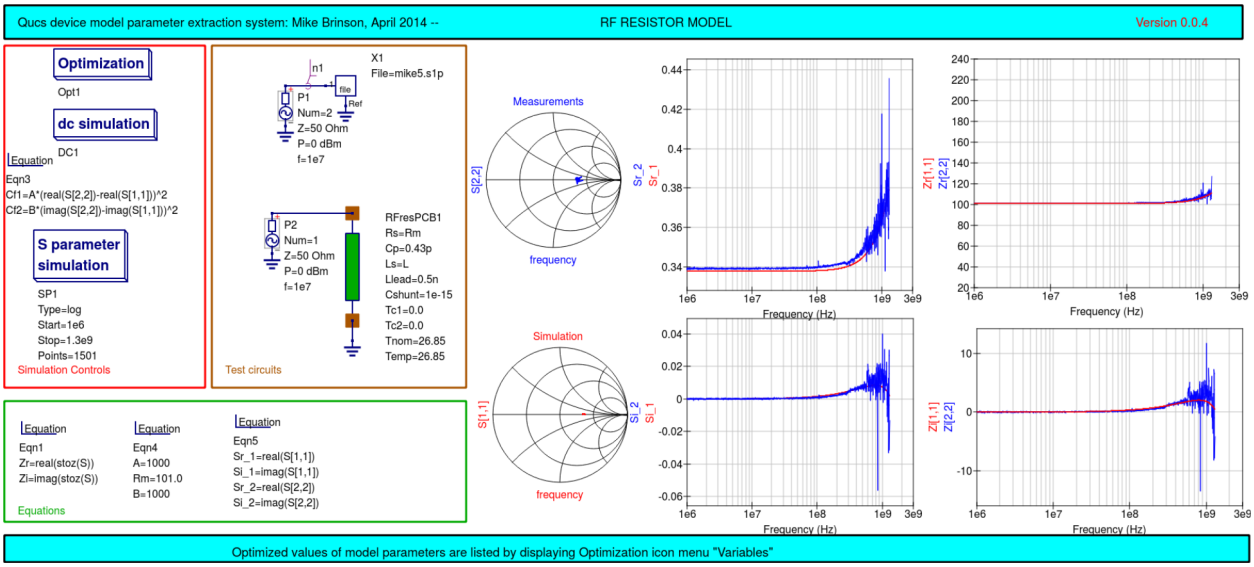


Figure 15 - Verilog-A models parameter data extraction for a 100 Ω axial thin film resistor. Fixed model parameter values: **Rs = Rm =101** Ω, **CShunt = 1e-15 F**, **Llead = LL = 0.5nH**, **Cp = C = 0.43pF**; Optimised values: **Ls = L = 3.99nH**. To reduce simulation time the ASCO cost variance was set to 1e-3. The ASCO method was set to DE/best/1/exp.

13.10 End Notes

This brief Qucs note outlines the fundamental properties of subicircuit and verilog-A compact component models for RF resistors. The use of optimization for the extraction of subcircuit and Verilog-A compact model parameters from measured S parameters is also demonstrated. The presented techniques form part of the simulation and device modelling capabilities available with the latest Qucs release[5].

Technical description concerning the simulator

> Available online at http://qucs.sourceforge.net/tech/technical.html

Example schematics

> Available online at http://qucs.sourceforge.net/download.html#example

[5] Qucs release 0.0.18, or greater.