Qucs Help Documentation

Release 0.0.19

Qucs Team (2014)

29.11.2016

Inhaltsverzeichnis

Contents:

KAPITEL 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.

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.

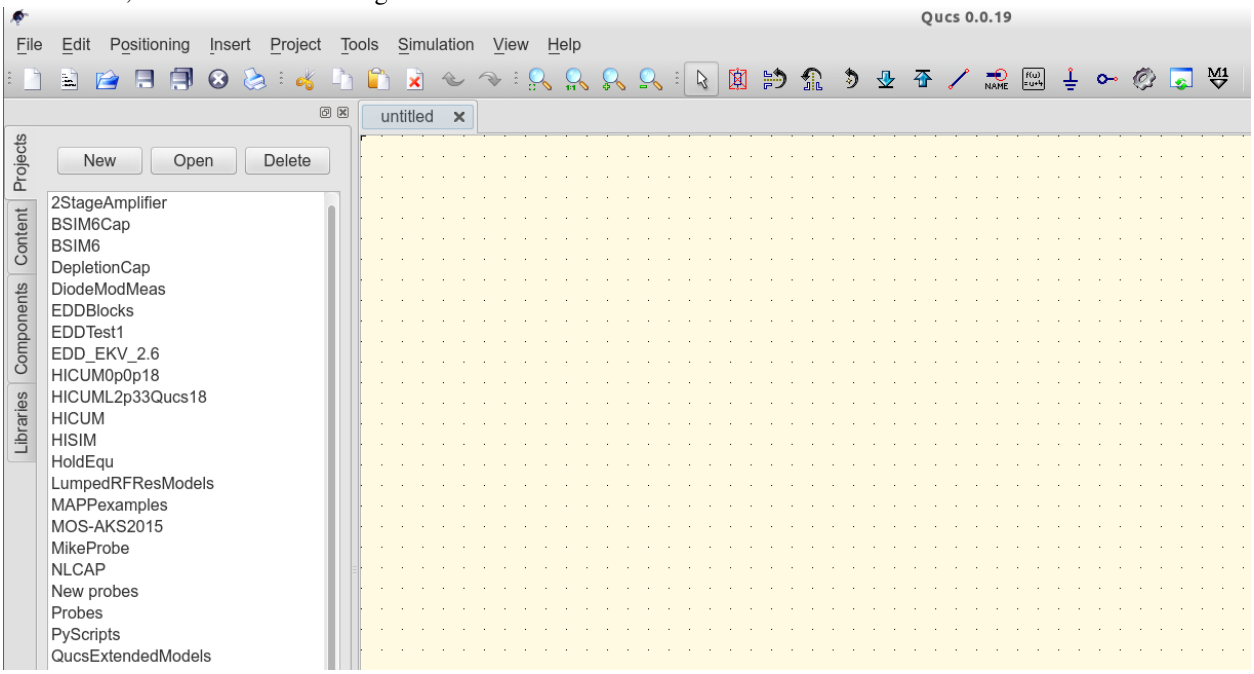


Abbildung 1 - Qucs Hauptfenster

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.

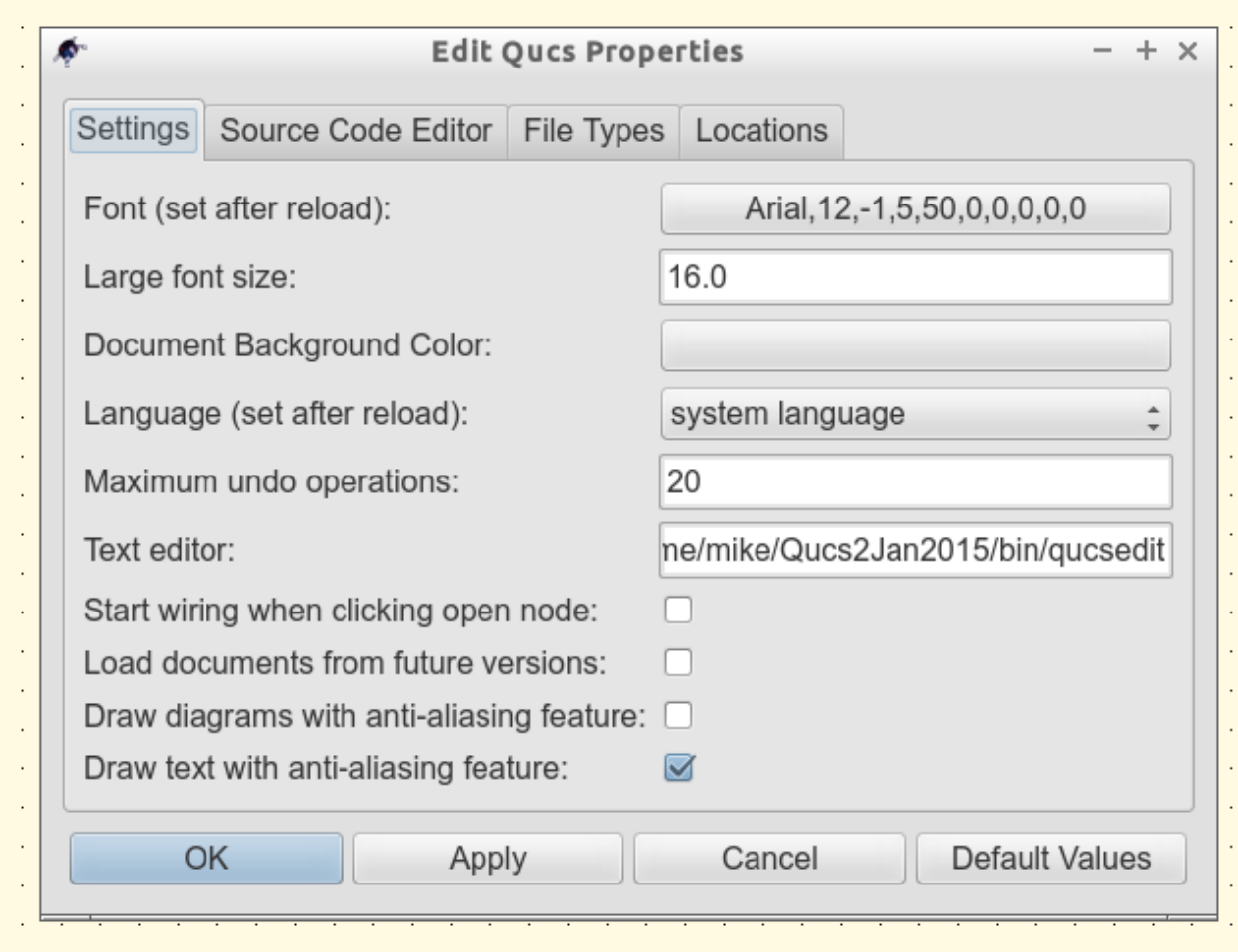


Figure 2 - QucsEditProperties window

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.

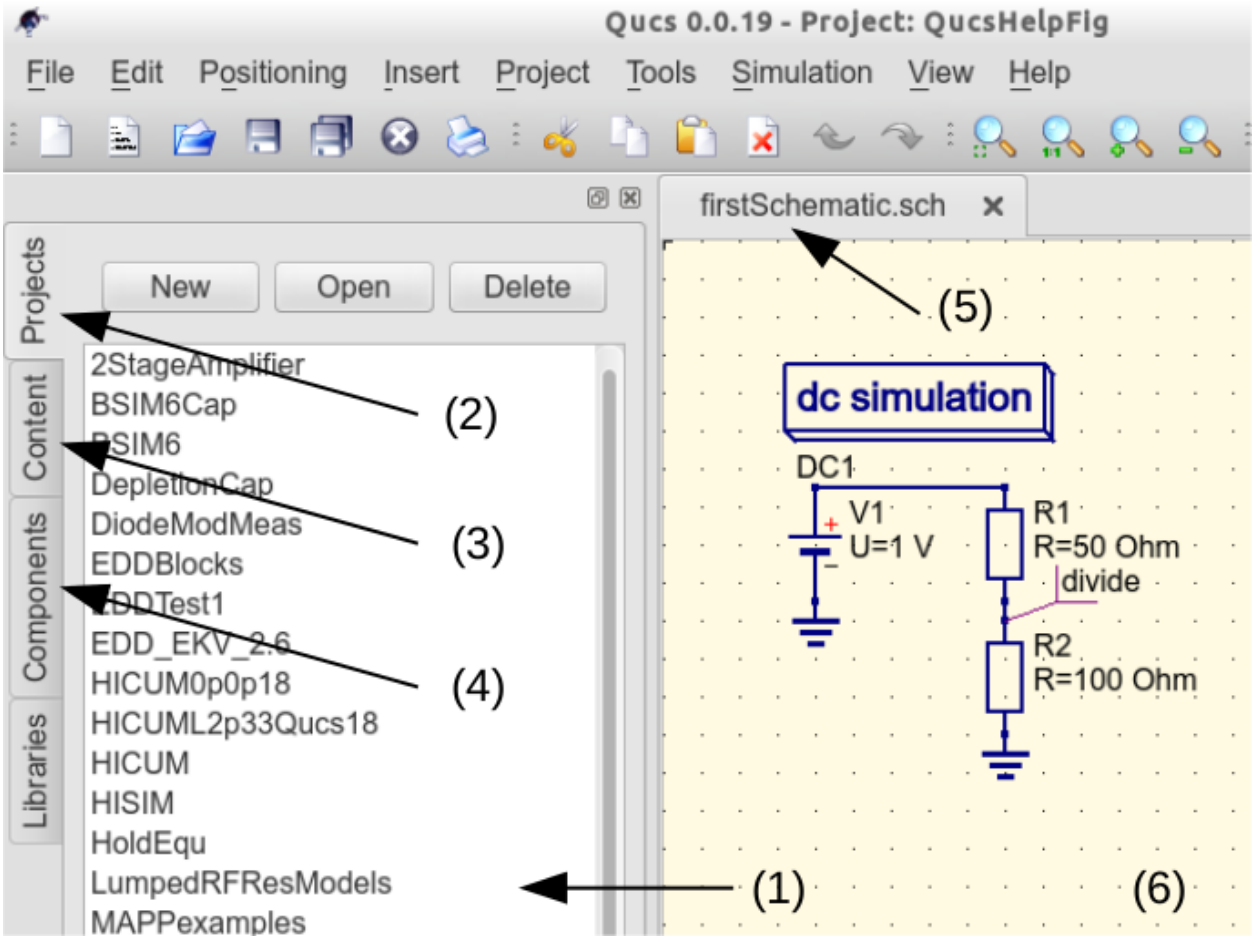


Figure 3 - Qucs main window with working areas labelled

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.

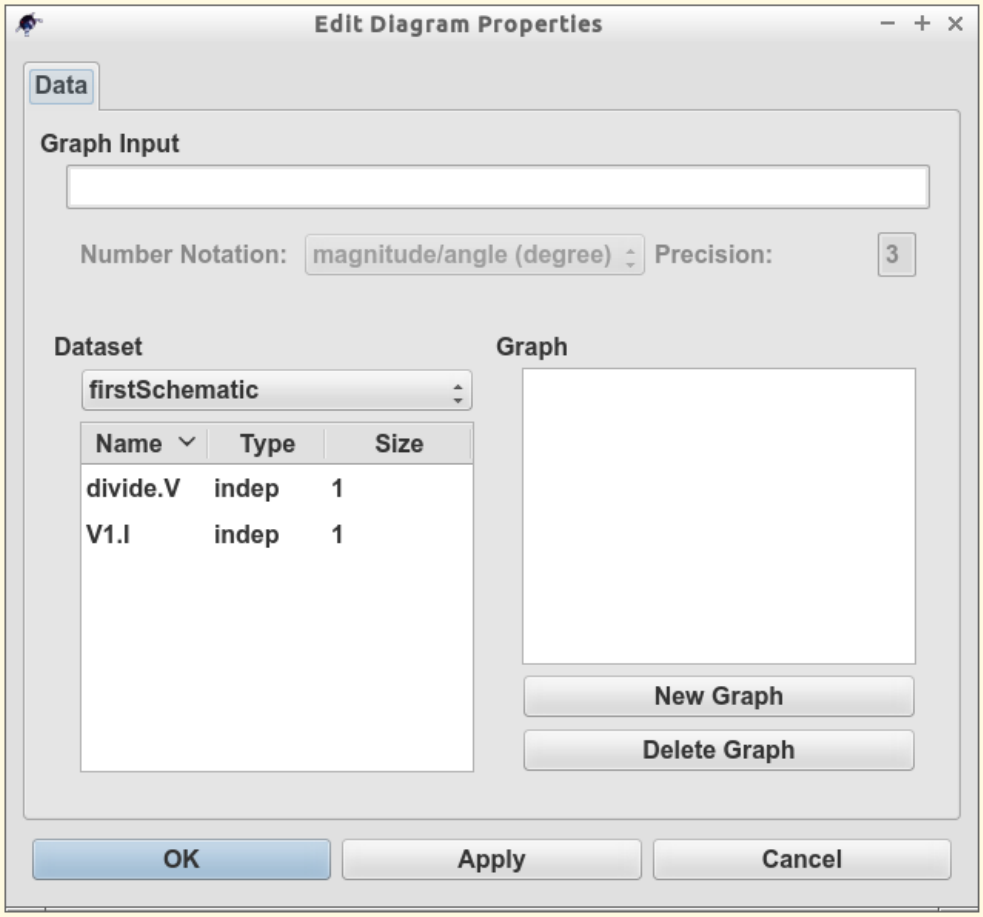


Figure 4 - Qucs data display window showing a **Tabular** dialogue

Einführung in die Optimierung

Für die Schaltkreisoptimierung verwendet Qucs ein Programm namens ASCO (http://asco.sourceforge.net/). Es folgt eine kurze Beschreibung, wie man ein Schaltplan dafür vorbereitet, die Optimierung ausführt und die Ergebnisse interpretieren kann. Bevor man diese Funktionalität benutzen kann, muss ASCO auf Ihrem Computer installiert sein.

Die Optimierung eines Schaltkreises ist nicht mehr als die Minimierung einer Kostenfunktion. Das kann entweder die Verzögerungszeit oder die Anstiegszeit in einer digitalen Schaltung, oder die Leistungsverstärkung einer analogen Schaltung sein. Eine andere Möglichkeit ist die Definition des Optimierungsproblems als eine Zusammensetzung von Funktionen, was in diesem Fall zu einem Gütefaktor führt.

Um einen Schaltplan für ein Optimierung vorzubereiten, müssen zwei Dinge hinzugefügt werden: Gleichungen und die Optimierungskomponente. Nehmen Sie den Schaltplan aus Abbildung 1 und verändern Sie es solange, bis Sie den Schaltplan in Abbildung 2 erhalten.

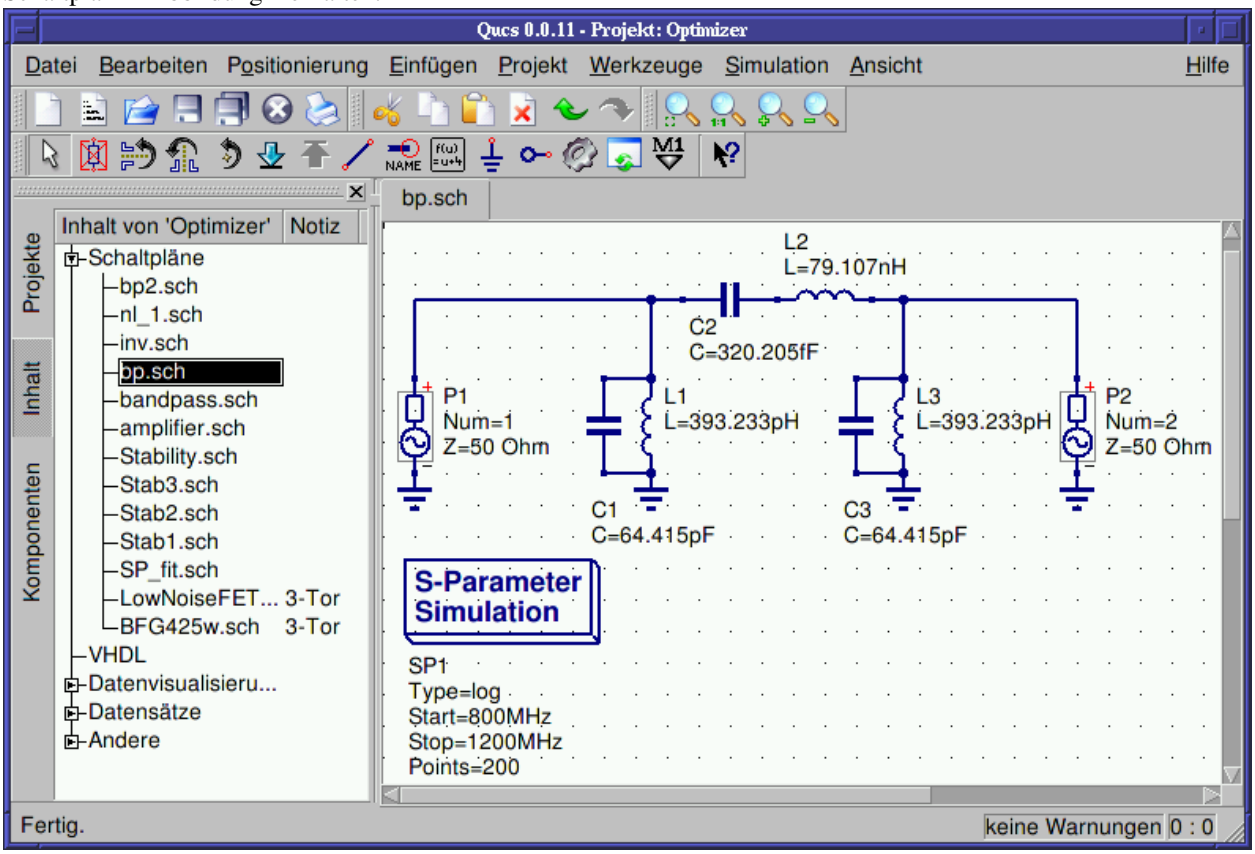


Abbildung 1 - Ursprünglicher Schaltplan.

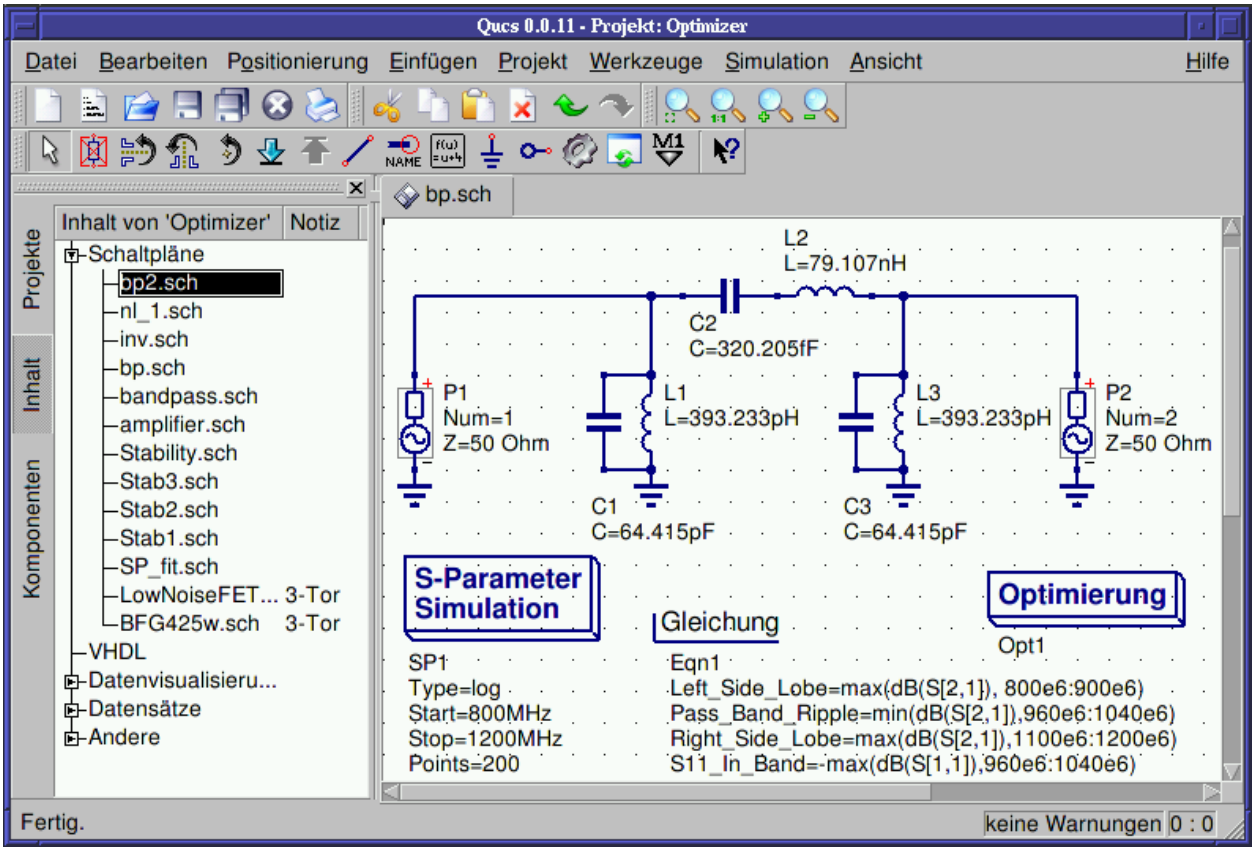


Abbildung 2 - Vorbereiteter Schaltplan.

Jetzt öffnen Sie die Optimierungskomponente und wählen die Algorithmusschaltfläche an. Aus den existierenden Parametern sollte besonders auf ‘Maximale Anzahl der Iterationen’, ‘Constant F’ und ‘Crossing over factor’ geachtet werden. Über- oder Unterschätzung kann zur vorzeitigen Konvergenz des Optimierers in einem lokalen Optimum führen oder auch zu sehr langen Optimierungszeiten.

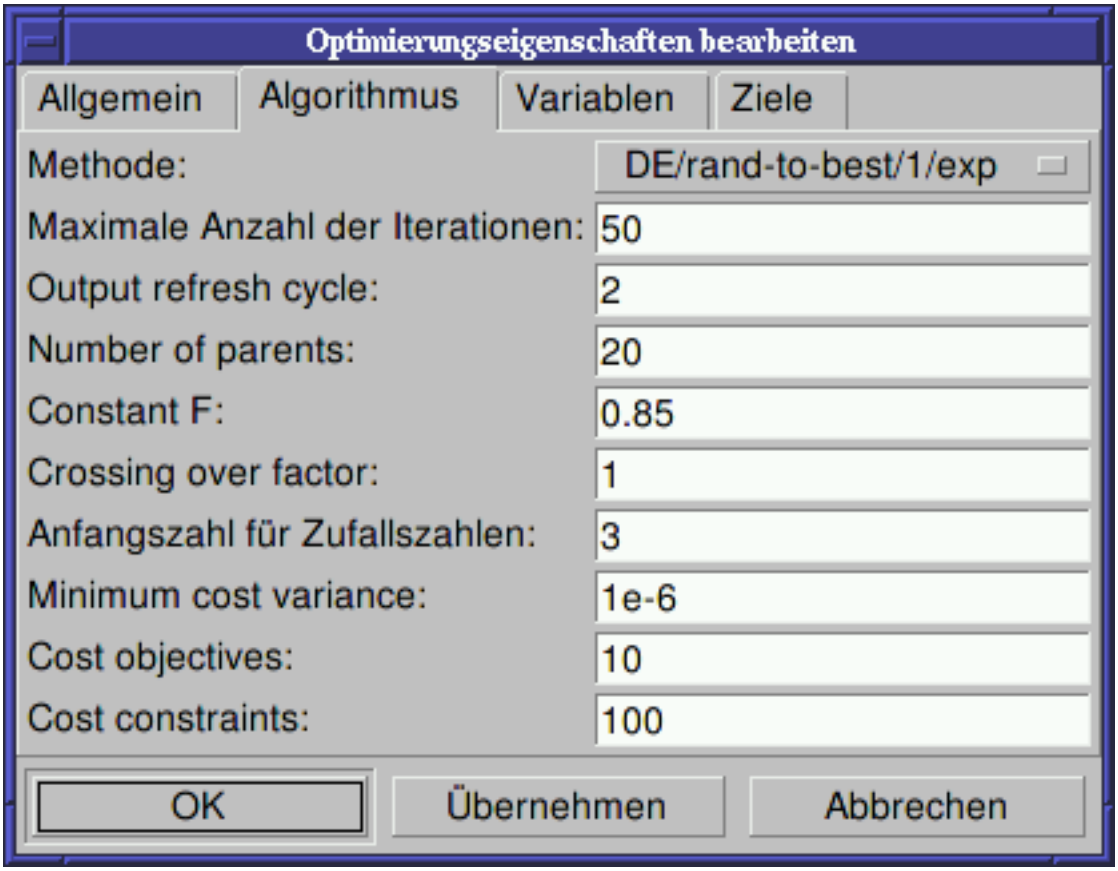


Abbildung 3 - Optimierungsdialog, Algorithmusoptionen.

Die Variablenschaltfläche, wo die Schaltkreiselemente definiert werden, die in einem bestimmten Interval optimiert werden können, ist in Abbildung 4 dargestellt. Die Variablennamen korrespondieren zu den Namen, die in die Komponenteneigenschaften platziert wurden und **nicht** zu den Namen der Komponenten.

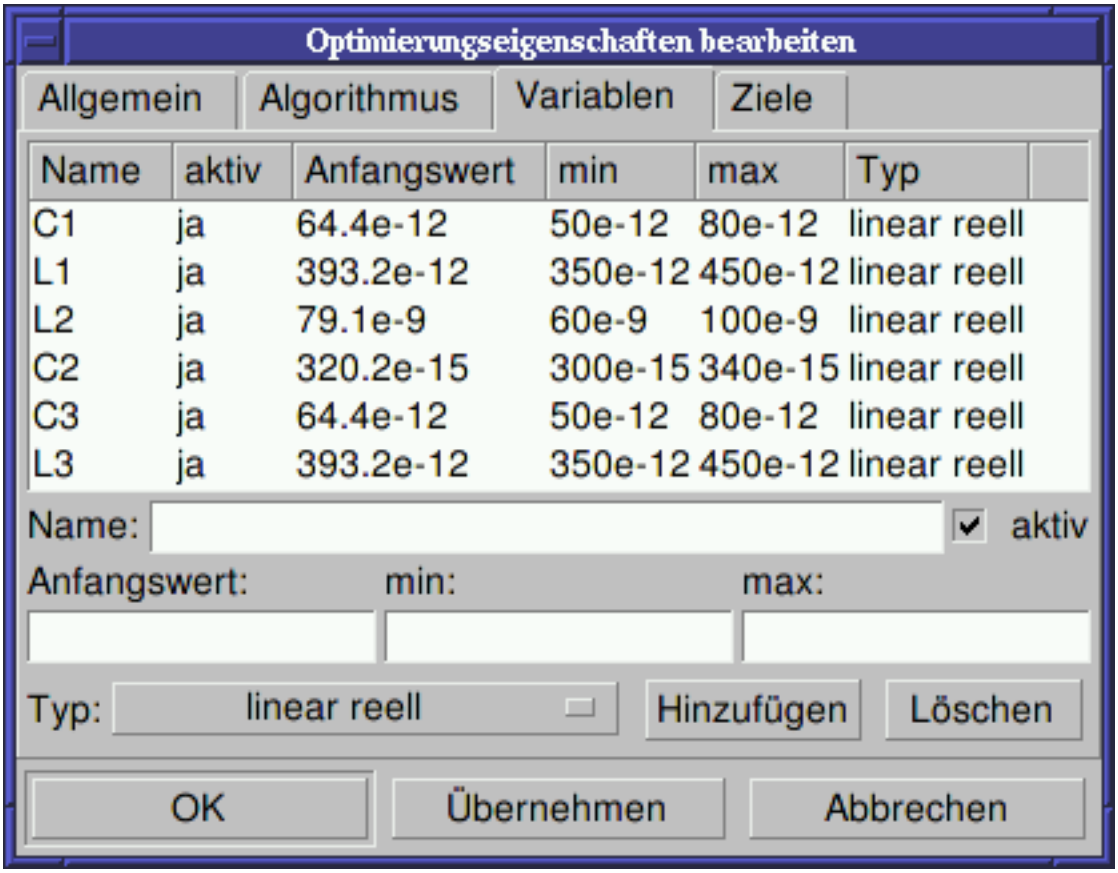


Abbildung 4 - Optimierungsdialog, Variablenoptionen.

Schließlich müssen noch die Ziele der Optimierung (maximieren, minimieren) und Optimierungsgrenzen (kleiner, größer, gleich) in der Zielschaltfläche eingegeben werden. ASCO kombiniert diese Ziele zu einer einzigen Kostenfunktion, die dann minimiert wird.

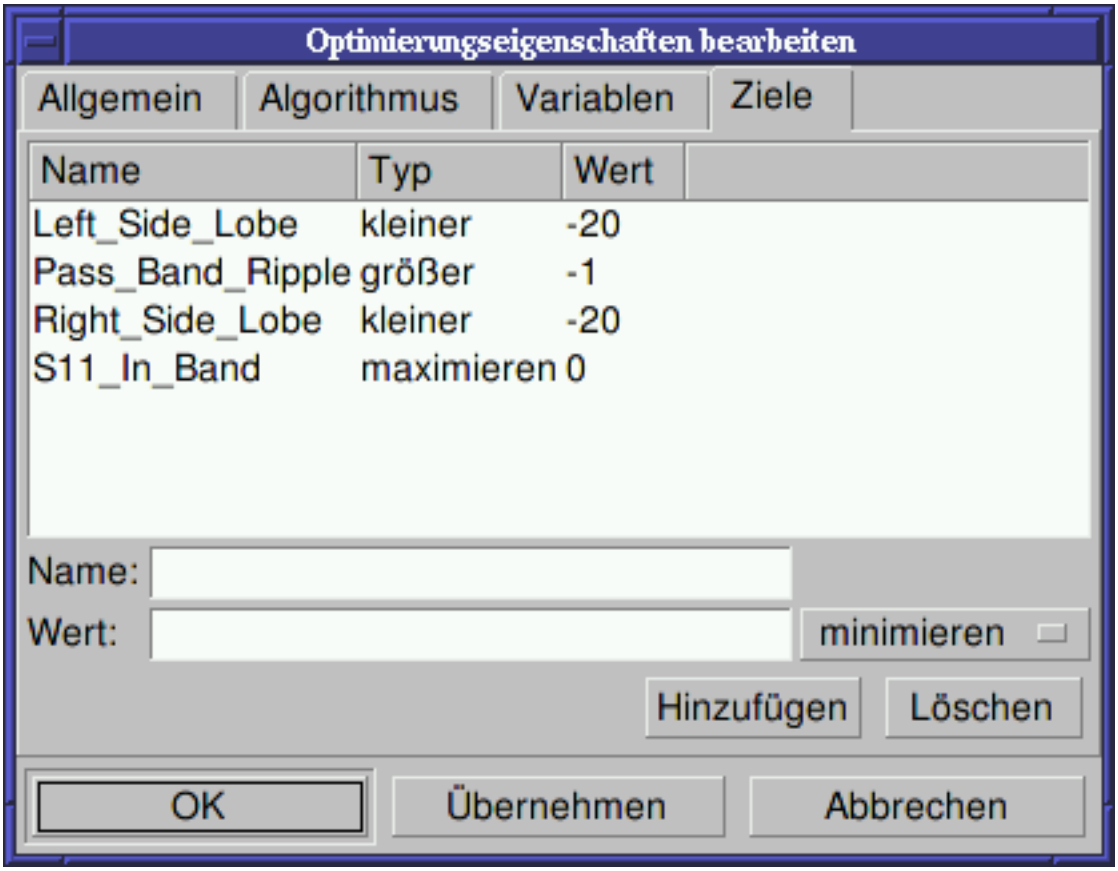


Abbildung 5 - Optimierungsdialog, Zieloptionen.

Der nächste Schritt ist die Veränderung des Schaltplans und die Definition der Schaltkreiselemente, die optimiert werden sollen. Der enstehende Schaltplan wird in Abbildung 6 dargestellt.

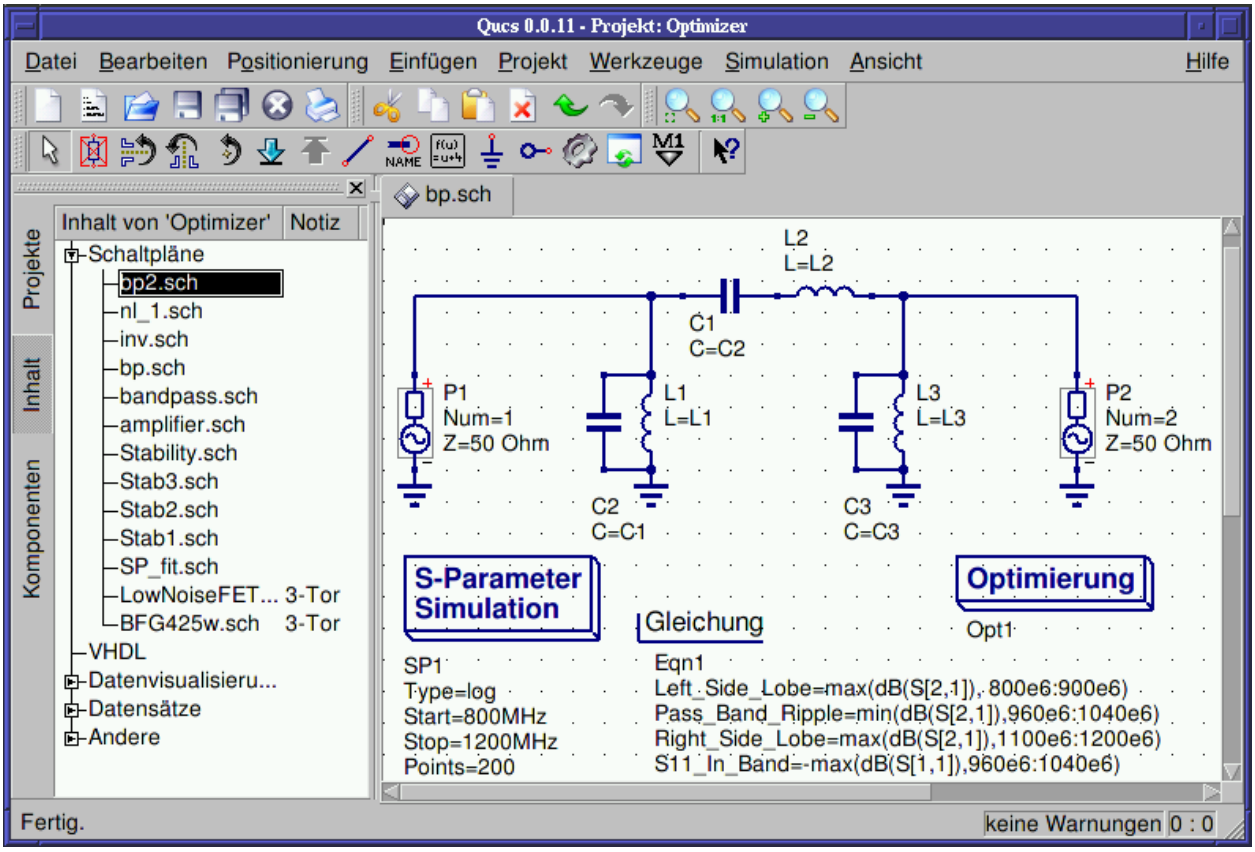


Abbildung 6 - Neues Qucs Hauptfenster.

Der letzte Schritt ist die Ausführung der Optimierung, d.h. das Starten der Simulation durch Drücken von F2. Wenn die Optimierung beendet ist, was auf einem modernen Computer ein paar Sekunden dauert, werden die besten Simulationsergebnisse angezeigt.

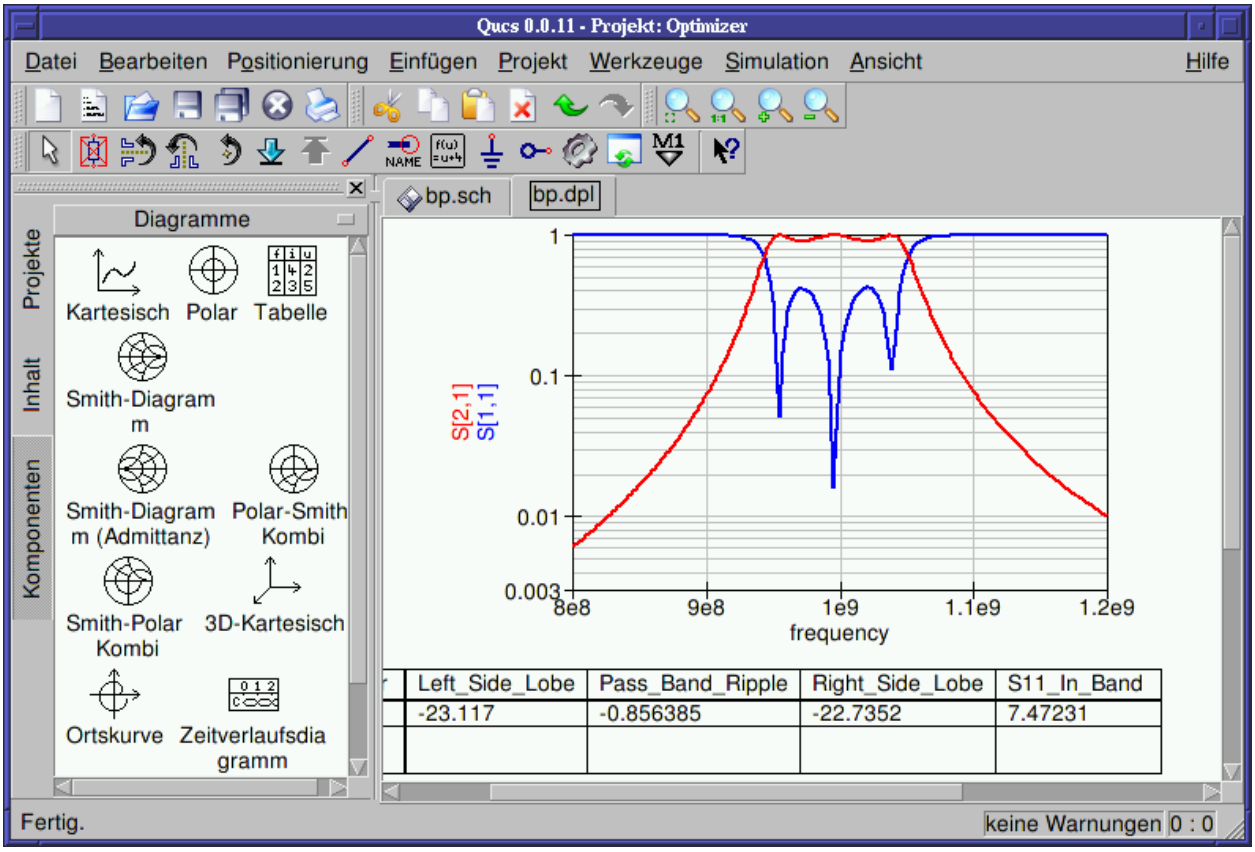


Abbildung 7 - Qucs Ergebnisfenster.

Die besten Schaltkreisgrößen findet man im Optimierungsdialog auf der Variablenschaltfläche. Es sind jetzt die Anfangswerte für jede der eingeführten Variablen (siehe Abbildung 8).

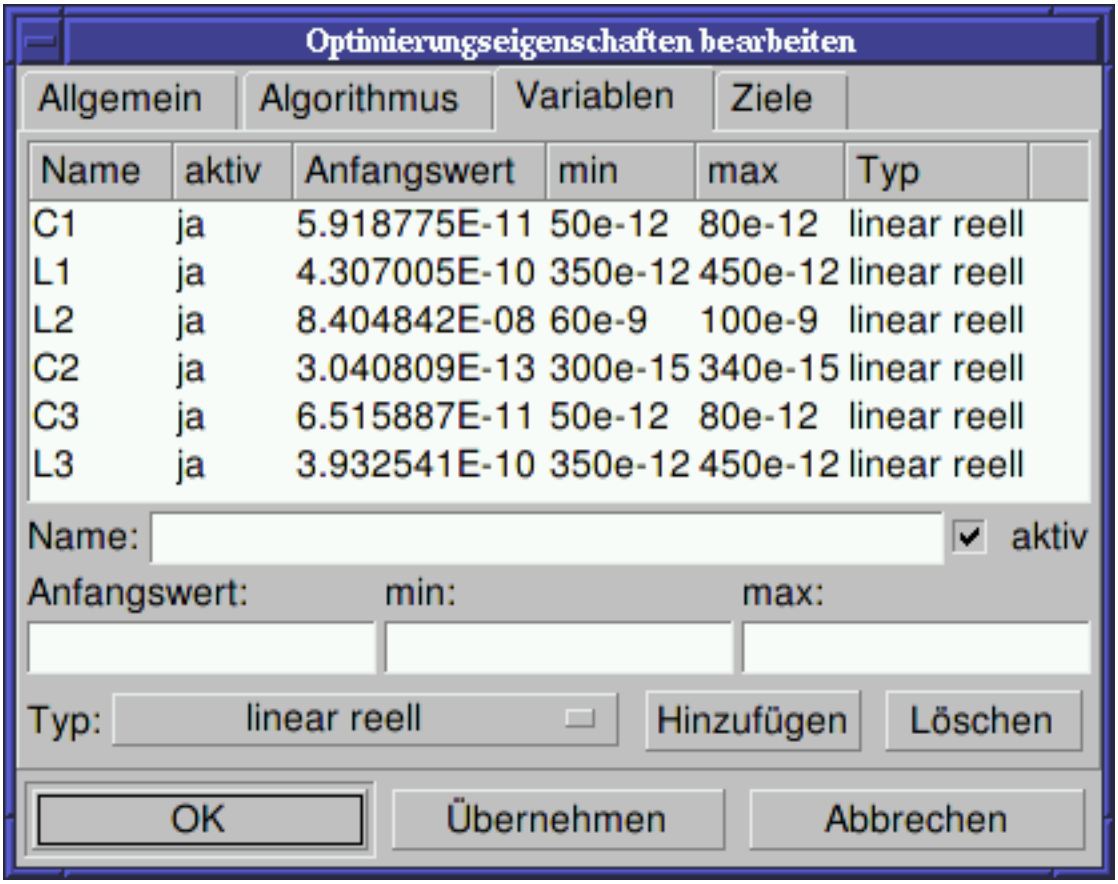


Abbildung 8 - Die optimierten Schaltkreisgrößen.

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.

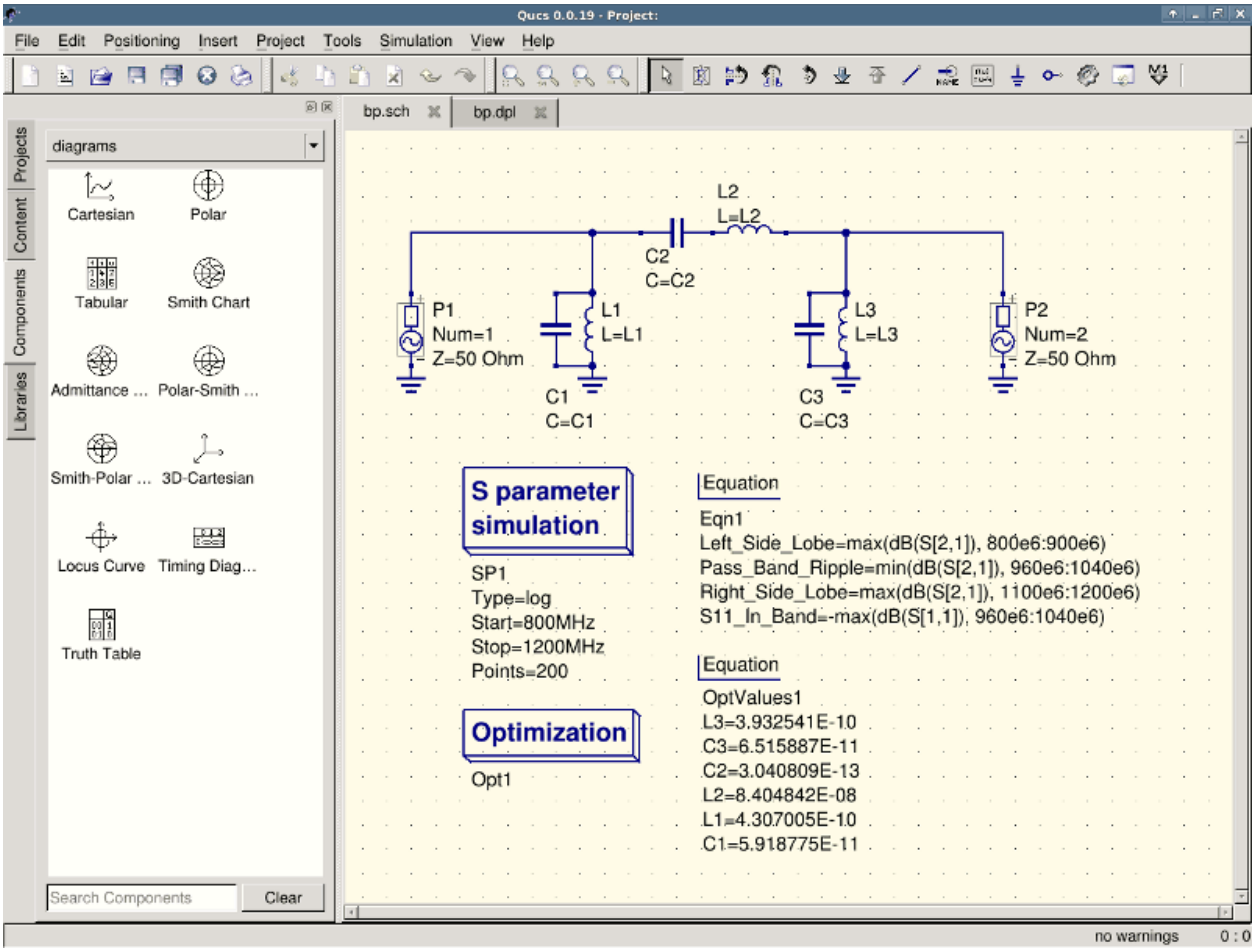


Figure 9 - Schematic with optimized values.

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.

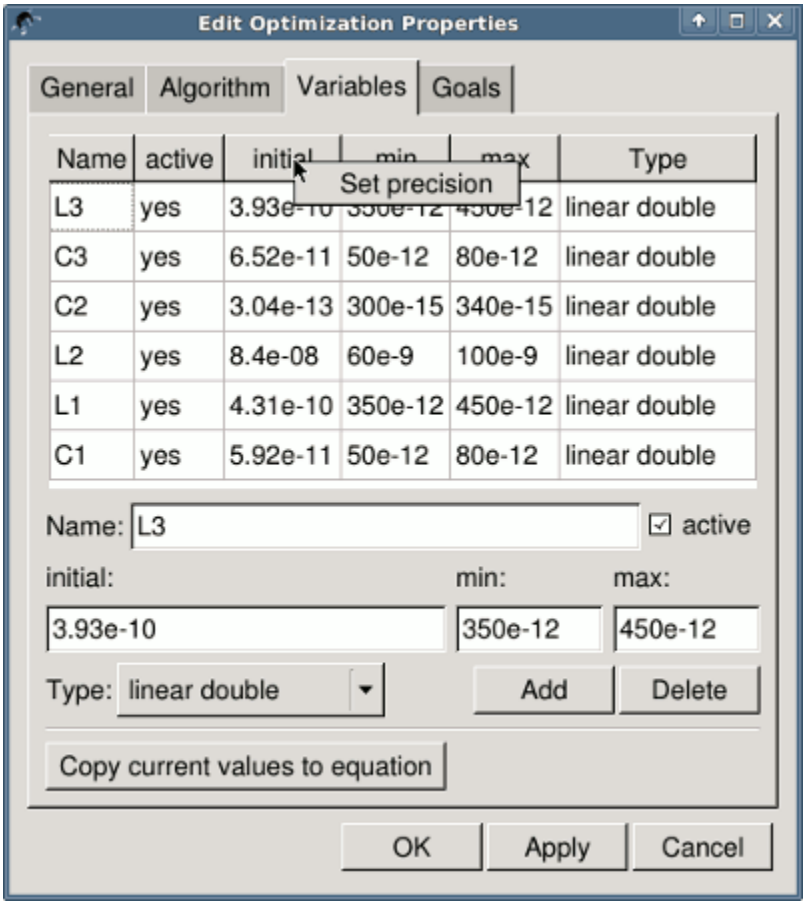


Figure 10 - Changing the displayed variables precision.

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.

Kurze Beschreibung der Funktionen

5.1 Generelle Funktionen

(gültig in allen Betriebszuständen)

Mausrad	Bewegt den Mauszeiger vertikal durch den Zeichenbereich. Kann auch zum Bewegen außerhalb der gegenwärtigen Größe genutzt werden.
Mausrad + Umstelltaste	Bewegt den Mauszeiger horizontal durch den Zeichenbereich. Kann auch zum Bewegen außerhalb der gegenwärtigen Größe genutzt werden.
Mausrad + Ctrl-Taste	Vergrößert oder verkleinert den betrachteten Zeichen-Bereich.
drag'n'drop file into document area	(Menüeintrag: Bearbeiten->Auswählen (Esc))

5.2 linke Maustaste

(Menüeintrag: Bearbeiten->Auswählen (Esc))

rechte Maustaste	Wählt das Element aus, das sich unter dem Mauszeiger befindet. Wenn sich dort mehrere Komponenten befinden, so kann man einige Mal klicken, bis die gewünschte Komponente ausgewählt ist.Während man die Maustaste gedrückt hält, kann man die Maus bewegen und somit die selektierte Komponente bewegen. Sind mehrere Komponenten ausgewählt, so werden diese alle bewegt.Wird die Maustaste gedrückt, während sich unter ihr kein Element befindet, so öffnet sich ein Rechteck. Durch das Bewegen der Maus bei gedrückter Maustaste wird ein Rechteck aufgezogen. Nach dem Loslassen der Maustaste sind alle Elemente ausgewählt, die sich vollständig innerhalb das Rechtecks befanden.Ein ausgewähltes Diagramm oder eine Zeichnung kann vergrößert oder verkleinert werden, indem man auf eine der Ecken des Diagramms mit dem Mauszeiger klickt und während dessen die Maus bewegt.Nach Drücken auf Komponententext kann dieser direkt editiert werden. Mit der Eingabetaste gelangt man zur nächsten Zeile. Wenn der momentane Komponententext eine Auswahlliste ist, so kann er nur mit den Cursor hoch/runter Tasten verändert werden.
linke Maustaste + Ctrl-Taste	Erlaubt es, mehr als ein einziges Element auszuwählen, z.B. das Auswählen eines Elementes hebt die Auswahl für ein anderes zuvor selektiertes Element nicht auf. Diese Funktion kann ebenfall erreicht werden durch das Aufziehen eines Rechtecks mit der Maus. Dazu wird die linke Maustaste gedrückt und während dessen die Maus bewegt. Alle vollständig in dem Rechteck befindlichen Elemente werden ausgewählt (Eintrag “linke Maustaste” vorher beachten).
“Draht”-Modus	Klick auf einen Draht selektiert einen einzigen geraden Draht anstatt eines kompletten Leitungszuges.
“Einfügen Komponente”-Modus	Öffnet einen Dialog zum Verändern der Elementeigenschaften (Die Beschriftung von Drähten, die Parameter von Komponenten, etc.).

5.3 “Einfügen Komponente”-Modus

(Klick auf eine Komponente/Diagramm in der Hauptarbeitsfläche)

rechte Maustaste	Plaziert eine neue Instanz einer Komponente in das Hauptarbeitsfenster.
“Draht”-Modus	(Menüeintrag: Einfügen->Draht (STRG-E))

5.4 linke Maustaste

(Menüeintrag: Einfügen->Draht (STRG-E))

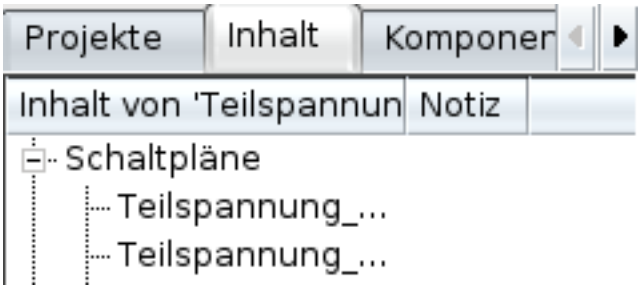
rechte Maustaste	Setzt den Start-/Endpunkt eines Drahtes.
“Draht”-Modus	Beendet einen Draht ohne an einem Draht oder Port zu sein.
“Einfügen Komponente”-Modus	(Menüeintrag: Bearbeiten->Einfügen (STRG-V))

5.5 linke Maustaste

(Menüeintrag: Bearbeiten->Einfügen (STRG-V))

rechte Maustaste	Plaziert eine Komponente in dem Hauptarbeitsfenster (aus der Zwischenablage).
“Draht”-Modus	Dreht das Element.

5.6 Wählt eine Datei aus.



doppel-Klick	Wählt eine Datei aus.	
rechts Klick	Öffnet eine Datei.	
Tastatur	Zeigt ein Menü mit:	
	“Öffnen”	• Öffnet die ausgewählte Datei
	“Umbenennen”	• Ändert den Namen der ausgewählten Datei
	“Löschen”	• Löscht die ausgewählte Datei
	“Gruppe löschen”	• Löscht die ausgewählte Datei und alle von ihr abgangigen Dateien (Schematic, Data-Diagramm, Simulationsdaten)

5.7 Tastatur

Viele Aktionen können über Tastatureingaben aktivert/ausgeführt werden. Die zu den Befehlen gehörigen Tastatureingaben kann man immer rechts neben den Texten im Menü finden. Einige weitere Tastaturkommandos sind in der folgenden Liste dargestellt:

Cursor links/rechts	Löscht das gegenwärtig ausgewählte Element oder schaltet den “Löschen”-Modus ein, wenn kein Element ausgewählt wurde.
Cursor hoch/runter	Ändert die Position des Markers innerhalb eines Graphen.Wenn kein Marker ausgewählt wurde, werden die selektierten Elemente bewegt.Wenn kein Element ausgewählt wurde, bewegt man sich durch das Dokument.
Tabulator-Taste	Ändert die Position des ausgewählten Markers in einem mehrdimensionalen Graphen.Wenn kein Marker ausgewählt wurde, werden die selektierten Elemente bewegt.Wenn kein Element ausgewählt wurde, bewegt man sich durch das Dokument.
Tabulator-Taste	Geht zum nächsten geöffneten Dokument (bezogen auf die oben vorhandenen Tabulator-Reiter).

KAPITEL 6

Arbeiten mit Schaltungshierarchien

Unterschaltungen werden benutzt, um mehr Klarheit in die Gesamtschaltung zu bringen. Die Funktion ist sehr vorteilhaft, wenn es sich um große Schaltungen handelt oder wenn Schaltungsteile mehrfach benutzt werden.

In Qucs wird jede Schaltung, die Ports enthält, als Unterschaltung betrachtet. Man erhält eine Unterschaltung, indem man den “Anschluß einfügen”-Knopf in der Knopfleiste betätigt, auf den Reiter “Komponenten” geht, dort den Blättertext “diskrete Komponenten” und darin die Komponente “Schaltkreis-Anschluß” auswählt. Alternativ kann auch aus dem Menü der Eintrag “Einfügen->Anschluß” benutzt werden. Nachdem an allen Ein- und Ausgängen der Schaltung Ports plaziert wurden, wird die Schaltung gespeichert, z.B. über “CRTL-S”. Durch einen Blick in den “Inhalts-Reiter” (Abbildung 1) erkennt man, das hinter dem Dateinamen die “Notiz” “2-Port” hinzugefügt wurde. Diese “Notiz” markiert alle Schaltungen bei denen es sich um Unterschaltungen handelt. Jetzt wird ein Schaltungsdesign geöffnet, in dem die Unterschaltung verwendet werden soll. Durch einen Klick auf den Namen der Unterschaltung im “Inhalt”-Reiter kann eine neue Komponente in die Schaltung eingefügt werden. Anschließend kann eine Simulation durchgeführt werden. Das Ergebnis ist wie erwartet das gleiche, als würde man die Komponenten direkt im Hauptarbeitsfenster plazieren.

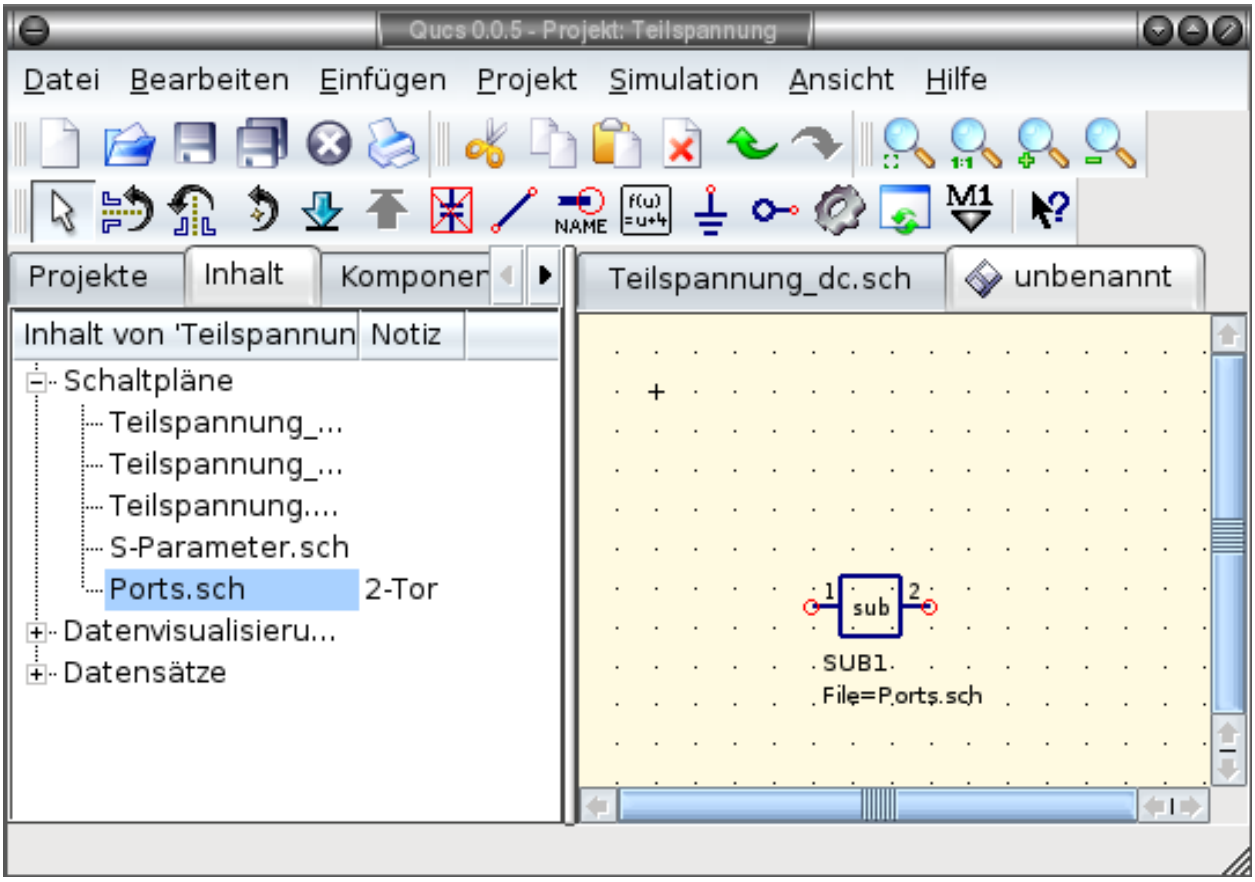


Abbildung 1 - Zugriff auf eine Unterschaltung

Wenn die Komponente markiert ist (durch das Anklicken der selbigen), die eine Unterschaltung darstellt, so kann deren Inhalt durch den Pfeil in der Knopfleiste mit der Beschriftung “Gehe in den Schaltkreis hinein (STRG-I), betrachtet werden. Alternativ erfolgt der Aufruf der Funktion über die Menüauswahl “Bearbeiten->Gehe in den Schaltkreis hinein (STRG-I)”. Das Verlassen der Unterschaltung erfolgt über die Betätigung des Knopfes “Verlasse den Schaltkreis (STRG-H)” bzw. über den Menüeintrag “Bearbeiten->Verlasse den Schaltkreis (STRG-H)”.

Wird nicht das vordefinierte Symbol für die Unterschaltung gewünscht, so kann ein eigenes Symbol gezeichnet werden und eigener Text hinzugefügt werden. Dazu geht man einfach in den Schaltungsentwurf mit der Unterschaltung hinein und wählt aus dem Menü den Eintrag “Datei->Schaltkreissymbol bearbeiten” aus. Wenn bis jetzt noch kein Symbol für die Schaltung gezeichnet wurde, so wird automatisch ein vordefiniertes Symbol erzeugt. Dieses Symbol kann durch das Zeichnen von Linien und Ecken modifiziert werden. Nachdem das Zeichnen beendet ist, wird das Symbol gespeichert. Anschließend wird es in einem anderen Schaltungsdesign plaziert, und voila, nun hat man ein neues Unterschaltungs-Symbol generiert.

Wenn die Komponente markiert ist (durch das Anklicken der selbigen), die eine Unterschaltung darstellt, so kann deren Inhalt durch den Pfeil in der Knopfleiste mit der Beschriftung “Gehe in den Schaltkreis hinein (STRG-I), betrachtet werden. Alternativ erfolgt der Aufruf der Funktion über die Menüauswahl “Bearbeiten->Gehe in den Schaltkreis hinein (STRG-I)”. Das Verlassen der Unterschaltung erfolgt über die Betätigung des Knopfes “Verlasse den Schaltkreis (STRG-H)” bzw. über den Menüeintrag “Bearbeiten->Verlasse den Schaltkreis (STRG-H)”.

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.

KAPITEL 7

Erste digitale Schritte

Qucs ist auch eine grafische Benutzeroberfläche für die Durchführung von digitalen Simulationen. Dieses Dokument enthält einen kurze Beschreibung, wie dies vonstatten geht.

Qucs verwendet das Programm FreeHDL (http://www.freehdl.seul.org), um digitale Simulationen durchzuführen. Das bedeutet, dass sowohl das FreeHDL-Packet, als auch der GNU C++ Compiler auf dem Computer installiert sein müssen.

Der Unterschied zwischen analogen und digitalen Simulationen ist nicht sehr groß. Falls also Erste Schritte bereits gelesen wurde, ist es jetzt recht einfach, auch eine digitale Simulation zum Laufen zu bringen. Es soll beispielhaft die Lokigtabelle eines einfachen logischen UNDs berechnet werden. Mit Hilfe der digitalen Komponenten in der Kombobox des Komponenten-Reiters auf der linken Seite sollte es gelingen, den Schaltplan, der in Abbildung 1 zu sehen ist, nachzubauen. Der Digitalsimulations-Block kann unter den anderen Simulationen gefunden werden.

Die Digitalquellen S1 und S2 sind die Eingänge, der Knoten mit der Bezeichnung Ausgang ist der Augang. Nach dem Starten der Simulation öffnet sich die Seite für die Datenvisualisierung. Das Diagramm Logiktabelle wird darauf platziert und die Variable Ausgang anschließend eingefügt. Jetzt wird die Logiktabelle eines UNDs mit zwei Eingängen angezeigt. Gratulation, die erste Digitalsimulation ist fertig!

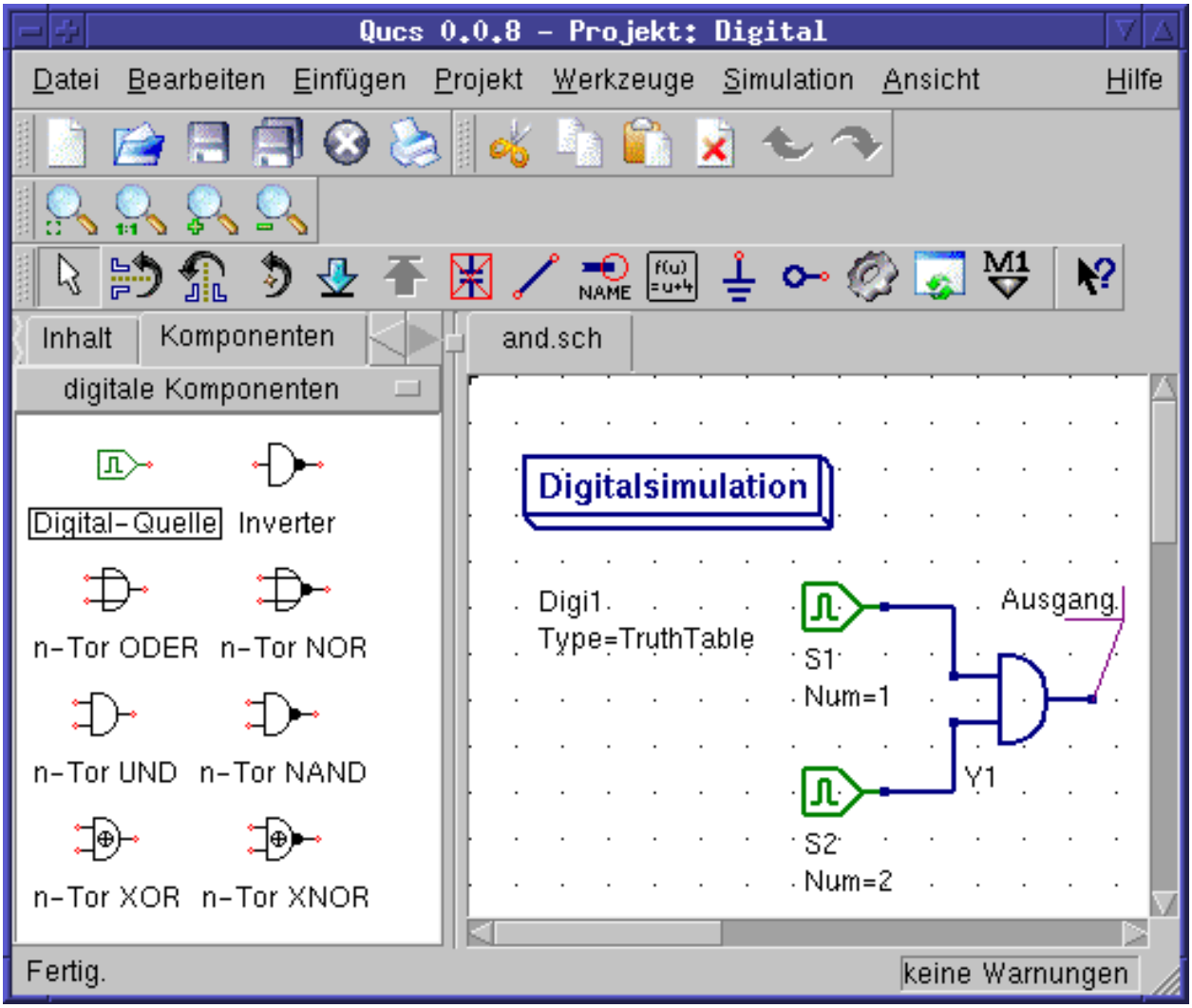


Abbildung 1 - Qucs Hauptfenster

Die Logiktabelle ist nicht die einzige Digitalsimulation, die Qucs durchführen kann. Es ist weiterhin möglich, ein beliebiges Signal in einen Schaltkreis einzuspeisen. Die Ausgangssignale können dann mit Hilfe eines Zeitdiagramms angezeigt werden. Um das zu erreichen, muss der Parameter Type des Simulations-Blocks auf TimeList gesetzt werden und die Zeitdauer der Simulation in den darauf folgenden Parameter eingetragen werden. Die Digitalquellen haben jetzt eine andere Bedeutung: Sie geben eine beliebige Bitsequenz aus, indem ein Ausgangszustand, das erste Bit, (low oder high) und eine Liste von Zeiten, an denen der Zustand gewechselt werden soll, definiert wird. Es ist zu beachten, dass diese Liste nach derem Ende wiederholt wird. Um ein 1GHz Taktsignal mit einem Tastverhältnis von 1:1 zu erzeugen, sieht die erwähnte Liste so aus: 0.5ns; 0.5ns

Um sich die Ergebnisse dieser Simulationsart anzusehen, gibt es den Diagrammtyp Zeitverlaufsdiagramm. Darin können die Ergebnisse aller Knoten Zeile für Zeile angezeigt werden. Viel Spaß dabei...

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.)

KAPITEL 8

Kurze Beschreibung der mathematischen Funktionen

Die folgenden Operationen und Funktionen können in Gleichungen von Qucs benutzt werden. Eine detaillierte Beschreibung entnehmen Sie bitte dem “Measurement Expressions Reference Manual”. Parameter in rechteckigen Klammern “[]” sind optional.

8.1 Operatoren

8.1.1 Arithmetische Operatoren

+x	Unär Plus
-x	Unär Minus
x+y	Addition
x-y	Subtraktion
x*y	Multiplikation
x/y	Division
x%y	Modulo-Operation (Nachkommateil einer Division)
x^y	Potenz

8.1.2 Logische Operatoren

!x	Negation
x&&y	Und
x||y	Oder
x^^y	Exklusiv-Oder
x?y:z	Abbreviation for conditional expression - if x then y else z
x==y	Gleich
x!=y	Ungleich
x<y	Kleiner als
x<=y	Kleiner als oder gleich
x>y	Größer als
x>=y	Größer als oder gleich

8.2 Mathematische Funktionen

8.2.1 Vektoren und Matrizen: Generierung

`eye(n)`	n x n Einheits-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 Vektoren und Matrizen: Grundlegende Matrix-Funktionen

`adjoint(x)`	Transponierte und konjungiert komplexe Matrix zu x
`det(x)`	Determinante von x
`inverse(x)`	Inverse Matrix zu x
`transpose(x)`	Transponierte Matrix zu x (Zeilen und Spalten vertauscht)

8.2.3 Elementare mathematische Funktionen: Grundlegende reelle und komplexe Funktionen

`abs(x)`	Absoluter Wert, Betrag einer komplexen Zahl
`angle(x)`	Phase einer komplexen Zahl im Bogenmaß
`arg(x)`	Gleicher Ausdruck wie <code>angle(x)</code>
`conj(x)`	Konjungiert komplexe Werte der Zahl x
`deg2rad(x)`	Umrechnung von Grad nach Bogenmaß
`hypot(x,y)`	Euklidische Distanzfunktion
`imag(x)`	Imaginärteil einer komplexen Zahl
`mag(x)`	Gleicher Ausdruck wie abs(x)
`norm(x)`	Quadrat von <code>mag(x)</code>
`phase(x)`	Phase einer komplexen Zahl in Grad
`polar(m,p)`	Transform polar coordinates `m` and `p` into a complex number
`rad2deg(x)`	Umrechnung von Bogenmaß nach Grad
`real(x)`	Realteil einer komplexen Zahl
`sign(x)`	Berechnet die Signumfunktion
`sqr(x)`	Quadrat (<code>x</code> zur Potenz zwei)
`sqrt(x)`	Quadratwurzel
`unwrap(p[,tol[,step]])`	Unwrap angle `p` (radians) – defaults `step` = 2pi, `tol` = pi

8.2.4 Elementare mathematische Funktionen: Exponential- und Logarithmus-Funktionen

`exp(x)`	Exponentialfunktion zur Basis e
`limexp(x)`	Begrenzte Exponentialfunktion
`log10(x)`	Dekadischer Logarithmus
`log2(x)`	Binärer Logarithmus
`ln(x)`	Natürlicher Logarithmus

8.2.5 Elementare mathematische Funktionen: Trigonometrie

`cos(x)`	Kosinus
`cosec(x)`	Kosekans
`cot(x)`	Kotangens
`sec(x)`	Sekans
`sin(x)`	Sinus
`tan(x)`	Tangens

8.2.6 Elementare mathematische Funktionen: Inverse trigonometrische Funktionen

`arccos(x)`	Arkuskosinus
`arccosec(x)`	Arkuskosekans
`arccot(x)`	Arkuskotangens
`arcsec(x)`	Arkussekans
`arcsin(x)`	Arkussinus
`arctan(x[,y])`	Arkustangens

8.2.7 Elementare mathematische Funktionen: Hyperbolische Funktionen

`cosh(x)`	Kosinus hyperbolicus
`cosech(x)`	Kosekans hyperbolicus
`coth(x)`	Kotangens hyperbolicus
`sech(x)`	Sekans hyperbolicus
`sinh(x)`	Sinus hyperbolicus
`tanh(x)`	Tangens hyperbolicus

8.2.8 Elementare mathematische Funktionen: Inverse hyperbolische Funktionen

`arcosh(x)`	Area Kosinus hyperbolicus
`arcosech(x)`	Area Kosekans hyperbolicus
`arcoth(x)`	Area Kotangens hyperbolicus
`arsech(x)`	Area Sekans hyperbolicus
`arsinh(x)`	Area Sinus hyperbolicus
`artanh(x)`	Area Tangens hyperbolicus

8.2.9 Elementare mathematische Funktionen: Runden

`ceil(x)`	Rundet zur nächstgrößeren Ganzzahl
`fix(x)`	Schneidet Nachkommastellen von reellen Zahlen ab
`floor(x)`	Rundet zur nächstkleineren Ganzzahl
`round(x)`	Rundet zur nächsten Ganzzahl

8.2.10 Elementare mathematische Funktionen: Spezielle Funktionen

`besseli0(x)`	Modifizierte Besselfunktion nullter Ordnung
`besselj(n,x)`	Besselfunktion erster Art und n-ter Ordnung
`bessely(n,x)`	Besselfunktion zweiter Art und n-ter Ordnung
`erf(x)`	Fehlerfunktion
`erfc(x)`	Komplementäre Fehlerfunktion
`erfinv(x)`	Inverse Fehlerfunktion
`erfcinv(x)`	Inverse komplementäre Fehlerfunktion
`sinc(x)`	Sinc-Funktion (sin(`x`)/`x` und 1 bei `x` =0)
`step(x)`	Sprungfunktion

8.2.11 Datenanalyse: Grundlegende Statistik-Funktionen

avg(x[,Bereich])	Average of vector `x`. If range given `x` must have a single data dependency
`cumavg(x)`	Kumulativer Mittelwert der Werte eines Vektors
`max(x,y)`	Liefert den größeren der beiden Werte x und y
max(x[,Bereich])	Maximum of vector `x`. If range given `x` must have a single data dependency
`min(x,y)`	Liefert den kleineren der beiden Werte x und y
min(x[,Bereich])	Minimum of vector `x`. If range is given `x` must have a single data dependency
`rms(x)`	Effektivwert aus den Werten eines Vektors
`runavg(x)`	Gleitender Mittelwert der Werte eines Vektors
`stddev(x)`	Standardabweichung der Werte eines Vektors
`variance(x)`	Varianz der Werte eines Vektors
`random()`	Zufallszahl zwischen 0.0 und 1.0
`srandom(x)`	Anfangswert für Zufallsgenerator

8.2.12 Datenanalyse: Grundlegende Operationen

`cumprod(x)`	Kumulatives Produkt der Werte in einem Vektor
`cumsum(x)`	Kumulative Summe der Werte in einem Vektor
`interpolate(f,x[,n])`	Spline interpolation of vector `f` using `n` equidistant points of `x`
`prod(x)`	Produkt der Werte in einem Vektor
`sum(x)`	Summe der Werte in einem Vektor
`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 Datenanalyse: Differentiation und Integration

`ddx(expr,var)`	Differenziert den mathematischen Ausdruck expr bezüglich der Variable var
`diff(y,x[,n])`	Differentiate vector `y` with respect to vector `x` `n` times. Defaults to `n` = 1
`integrate(x,h)`	Integriert den Vektor x numerisch bei angenommener konstanter Schrittweite h

8.2.14 Datenanalyse: Signalverarbeitung

`dft(x)`	Berechnet die diskrete Fourier-Transformation (DFT) des Vektors x
`fft(x)`	Berechnet die schnelle Fourier-Transformation (FFT) des Vektors x
`fftshift(x)`	Shuffles the FFT values of vector `x` to move DC to the center of the vector
`Freq2Time(V,f)`	Berechnet die inverse diskrete Fourier-Transformation der Funktion V(f) und interpretiert die Werte physikalisch
`idft(x)`	Berechnet die inverse diskrete Fourier-Transformation (IDFT) des Vektors x
`ifft(x)`	Berechnet die inverse schnelle Fourier-Transformation (IFFT) des Vektors x
`kbd(x[,n])`	Kaiser-Bessel Fensterfunktion
`Time2Freq(v,t)`	Berechnet die diskrete Fourier-Transformation der Funktion v(t) und interpretiert die Werte physikalisch

8.3 Elektrotechnische Funktionen

8.3.1 Umrechnung von Maßeinheiten

`dB(x)`	Spannungsdezibel
`dbm(x)`	Wandelt Spannung in Leistung in dBm um
`dbm2w(x)`	Wandelt Leistung in dBm in Leistung in Watt um
`w2dbm(x)`	Wandelt Leistung in Watt in Leistung in dBm um
`vt(t)`	Temperaturspannung für eine gegebene Temperatur t in Kelvin

8.3.2 Reflexionskoeffizienten und Stehwellenverhältnisse

`rtoswr(x)`	Konvertiert einen Reflexionsfaktor in das (Spannungs-)Stehwellenverhältnis
`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 Transformation von N-Tor-Matrizen

`stos(s,zref[,z0])`	Converts S-parameter matrix to S-parameter matrix with a different Z0
`stoy(s[,zref])`	Konvertiert die S-Parameter-Matrix in die Y-Parameter-Matrix
`stoz(s[,zref])`	Konvertiert die S-Parameter-Matrix in die 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])`	Konvertiert die Y-Parameter-Matrix in die S-Parameter-Matrix
`ytoz(y)`	Konvertiert die Y-Parameter-Matrix in die Z-Parameter-Matrix
`ztos(z[,z0])`	Konvertiert die Z-Parameter-Matrix in die S-Parameter-Matrix
`ztoy(z)`	Konvertiert die Z-Parameter-Matrix in die Y-Parameter-Matrix

8.3.4 Verstärker

`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 Stabilitätsfaktor der Zweitor-S-Parameter-Matrix <code>s</code>
`Mu2(s)`	Mu' Stabilitätsfaktor der Zweitor-S-Parameter-Matrix <code>s</code>
`NoiseCircle(Sopt,Fmin,Rn,F[,Arcs])`	Noise Figure(s) `F` circles
`PlotVs(data,dep)`	Returns data selected from `data`: dependency `dep`
`Rollet(s)`	Rollet Stabilitätsfaktor der Zweitor-S-Parameter-Matrix `s`
`StabCircleL(s[,arcs])`	Stabilitätskreise in der Lastebene
`StabCircleS(s[,arcs])`	Stabilitätskreise in der Quellebene
`StabFactor(s)`	Stability factor of a two-port S-parameter matrix
`StabMeasure(s)`	Stabilitätsmaß B1 einer Zweitor-S-Parameter-Matrix

8.4 Schreibweisen

8.4.1 Intervalle

`LO:HI`	Intervall von LO bis HI
`:HI`	Bis zu HI
`LO:`	Von LO an
`:`	Keine Intervallgrenzen

8.4.2 Matrizen und ihre Elemente

`M`	Die gesamte Matrix M
`M[2,3]`	Element in der 2. Zeile und der 3. Spalte der Matrix M
`M[:,3]`	Vektor bestehend aus der 3. Spalte der Matrix M

8.4.3 Zahlen, Vektoren, Matrizen

`2.5`	Reelle Zahl
`1.4+j5.1`	Komplexe Zahl
`[1,3,5,7]`	Vektor
`[11,12;21,22]`	Matrix

8.4.4 Zahlenendungen

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 Wertenamen

S[1,1]	S-Parameterwert
knotenname.V	DC-Spannung am Knoten *knotenname*
name.I	DC-Strom durch die Komponente *name*
knotenname.v	AC-Spannung am Knoten *knotenname*
name.i	AC-Strom durch die Komponente *name*
knotenname.vn	AC-Rauschspannung am Knoten *knotenname*
name.in	AC-Rauschstrom durch die Komponente *name*
knotenname.Vt	Transientenspannung am Knoten *knotenname*
name.It	Transientenstrom durch die Komponente *name*

Bitte beachten: Alle Spannungen und Ströme sind Spitzenwerte. Rauschspannungen sind Effektivwerte in 1 Hz Bandbreite.

8.5 Konstanten

i, j	Imaginäre Einheit (“Quadratwurzel von -1”)
pi	4*arctan(1) = 3.14159...
e	Eulerzahl = 2.71828...
kB	Boltzmann-Konstante = 1.38065e-23 J/K
q	Elementarladung = 1.6021765e-19 C

Spezielle Zeichen

In der Textkomponente und in den Achsenbeschriftungen der Diagramme können besondere Zeichen verwendet werden. Das wird mit LaTeX-Befehlen bewerkstelligt. Die folgende Tabelle beinhaltet die derzeit verfügbaren Zeichen.

Beachte: Welche dieser Zeichen auch tatsächlich richtig angezeigt werden können, hängt von dem Zeichensatz, der von Qucs verwendet wird, ab.

Kleine griechische Buchstaben

LaTeX-Befehl	Unicode	Beschreibung
\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

Große griechische Buchstaben

LaTeX-Befehl	Unicode	Beschreibung
\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

Mathematische Symbole

LaTeX-Befehl	Unicode	Beschreibung
\cdot	0x00B7	Multiplikationspunkt (zentrierter Punkt)
\times	0x00D7	Multiplikationskreuz
\pm	0x00B1	Plus/Minus Zeichen
\mp	0x2213	Minus/Plus Zeichen
\partial	0x2202	Symbol für partielle Differentiation
\nabla	0x2207	Nablaoperator
\infty	0x221E	Symbol für die Unendlichkeit
\int	0x222B	Integralsymbol
\approx	0x2248	Näherungssymbol (geschwungenes Gleichheitszeichen)
\neq	0x2260	Ungleichzeichen
\in	0x220A	Symbol für “enthält”
\leq	0x2264	Kleiner-Gleich-Zeichen
\geq	0x2265	Größer-Gleich-Zeichen
\sim	0x223C	(mitteleuropäisches) Proportionalzeichen
\propto	0x221D	(amerikanisches) Proportionalzeichen
\diameter	0x00F8	Durchmesser-Zeichen (auch für Durchschnitt)
\onehalf	0x00BD	ein Halb
\onequarter	0x00BC	ein Viertel
\twosuperior	0x00B2	Quadrat (Zweierpotenz)
\threesuperior	0x00B3	Dreierpotenz
\ohm	0x03A9	Einheit für den elektrischen Widerstand (großes griechisches omega)

KAPITEL 10

Anpassungsnetzwerke

Das Erstellen von Anpassschaltungen ist eine häufig anzutreffende Aufgabe in der Mikrowellentechnik. Qucs kann das automatisch. Dies sind die dazu notwendigen Schritte:

Durchführen einer S-Parameter-Simulation zur Berechnung der Reflektionsfaktoren.

Einfügen eines Diagramms zur Darstellung eines Reflektionsfaktors (z.B. S[1,1] für Tor 1, S[2,2] für Tor 2 usw.)

Setzen eines Markers auf die Kurve des Reflektionsfaktors und schrittweise zur gewünschten Frequenz gehen.

Mit der rechten Maustaste auf den Marker klicken und “Leistungsanpassung” in dem erscheinenden Menü auswählen.

Es öffnet sich ein Dialogfenster, in dem die Werte auch von Hand angepasst werden können. Die Referenzimpedanz kann beispielsweise von 50 Ohm abweichend gewählt werden.

Nach dem Bestätigen mit dem “Erstellen”-Knopf wird auf den Schaltplan zurückgeschaltet und durch Bewegen der Maus kann die fertige Anpassschaltung eingefügt werden.

Die linke Seite der Anpassschaltung ist der Eingang und die rechte Seite muss mit dem Schaltkreis verbunden werden.

Falls der Marker auf eine Variable namens “Sopt” zeigt, beinhaltet das Menü die Option “Rauschanpassung”. Beachte, dass der einzige Unterschied zur “Leistungsanpassung” die Verwendung des konjugiert komplexen Reflektionsfaktors ist. Wenn die Variable also einen anderen Namen hat, kann Rauschanpassung durch eine kleine Änderung der Werte in dem Dialog erreicht werden.

Der Anpassungsdialog kann auch über das Menü aufgerufen werden (Werkzeuge->Anpassnetzwerk) oder durch das Tastenkürzel (<CTRL-5>). Dann müssen aber alle Werte von Hand eingetragen werden.

10.1 Zweitor-Anpassungsnetzwerke

Falls der Variablenname in dem Marker ein S-Parameter ist, dann existiert ein weiterer Menüpunkt für die gleichzeitige Anpassung am Ein- und Ausgang des Zweitors. Das funktioniert sehr ähnlich wie mit den oben beschriebenen Schritten. Das Ergebnis sind zwei Anpassnetzwerke: Der ganz linke Knoten muss an Tor 1 angeschlossen werden und der ganz rechte Knoten an Tor 2. Die zwei Knoten in der Mitte müssen mit dem Zweitor verbunden werden.

KAPITEL 11

Installierte Dateien

Das Qucs-System benötgt verschiedene Programme. Diese werden während des Installationsvorgangs mitinstalliert. Der Installationspfad von Qucs wird durch die Parameterübergabe beim Aufruf des Konfigurationsscripts bestimmt (z.B. configure –prefix=/pfad/zum/gewünschten/Verzeichnis). Die folgenden Ausführungen beziehen sich auf den angenommenen voreingestellten Installationpfad (/usr/local/).

- /usr/local/bin/qucs - die Benutzeroberfläche (GUI)
- /usr/local/bin/qucsator - Der Simulator (Konsolenprogramm)
- /usr/local/bin/qucsedit - Ein einfacher Texteditor
- /usr/local/bin/qucshelp - Ein kleines Programm zur Darstellung der Hilfe-Seiten
- `/usr/local/bin/qucstrans` - a program for calculation transmission line parameters
- `/usr/local/bin/qucsfilter` - a program synthesizing filter circuits
- /usr/local/bin/qucsconv - Ein Dateiformat-Konverter (Konsolenprogramm)

Alle Programme sind einzelnen Anwendungen, die unabhängig voneinander gestartet werden können. Die Benutzeroberfläche (GUI)

- ruft qucsator auf, wenn eine Simulation durchgeführt werden soll,
- ruft qucsedit auf, wenn eine Textdatei angezeigt werden soll,
- ruft qucshelp auf, wenn die Hilfe-Seiten dargestellt werden sollen,
- calls `qucstrans` when calling this program from menu “Tools”,
- calls `qucsfilter` when calling this program from menu “Tools”,
- ruft qucsconv auf, wenn eine SPICE-Komponente plaziert wird und wenn eine Simulation mit der SPICE-Komponente durchgeführt werden soll.

Desweiteren werden die folgenden Verzeichnisse während der Installation erzeugt:

- /usr/local/share/qucs/bitmaps - Enthält alle Bitmap-Bilder (Icons, etc.)
- /usr/local/share/qucs/docs - Enthält alle HTML-Dokumente für die Hilfe-Seiten
- /usr/local/share/qucs/lang - Enthält die Datien für die Sprachanpassung

11.1 Kommandozeilen Argumente

```
qucs [Datei1 [Datei2 ...]]
```

`qucsator [-b] -i Netzliste -o Datensatz` (b = Fortschrittsanzeige)

`qucsedit [-r] [Datei]` (r = nur-lesen)

`qucshelp (keine Argumente)`

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

Dateiformat der Schaltpläne

Dieses Dokument beschreibt kurz das Dateiformat der Schaltpläne von Qucs. Das Format wird für Schaltpläne (normalerweise mit der Dateiendung `.sch`) und für Datenvisualisierungen (normalerweise mit der Dateiendung `.dpl`) verwendet. Der folgende Text zeigt ein kurzes Beispiel für eine solche Datei.

```
<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>
```

Die Datei beinhaltet mehrere Abschnitte. Jeder dieser Abschnitte wird nachfolgend erklärt. Jede Zeile besteht aus einem einzigen Informationsblock, der mit dem Kleiner-Zeichen < beginnt und mit dem Größer-Zeichen > endet.

12.1 Eigenschaften

Der erste Abschnitt beginnt mit `<Properties>` und endet mit `</Properties>`. Er beinhaltet die Dokumenteneigenschaften der Datei. Jede dieser Zeilen ist optional. Die folgenden Eigenschaften werden unterstützt.

- `<View=x1,y1,x2,y2,scale,xpos,ypos>` beinhaltet die Pixelposition des Schaltplanfensters in den ersten vier Zahlen, die aktuelle Skalierung und die aktuelle Position der linken oberen Ecke (die letzten beiden Zahlen).

- <Grid=x,y,on> beinhaltet den Gitternetzabstand in Pixeln (die ersten beiden Zahlen) und ob das Gitternetz sichtbar ist (letzte Zahl 1) oder nicht (letzte Zahl 0).
- <DataSet=name.dat> beinhaltet den Dateinamen des Datensatzes, der mit diesem Schaltplan assoziiert wird.
- <DataDisplay=name.dpl> beinhaltet den Dateinamen der Datenvisualisierung, die mit diesem Schaltplan assoziiert wird (bzw. den Dateinamen des Schaltplan, falls das Dokument eine Dateinvisualisierung ist).
- <OpenDisplay=yes> beinhaltet eine 1, falls die Datenvisualisierung automatisch nach der Simulation angezeigt werden soll, anderenfalls eine 0.

12.2 Symbol

Dieser Abschnitt beginnt mit <Symbol> und endet mit </Symbol>. Er beinhaltet die Zeichnungselemente, die das Schaltplansymbol dieser Datei bilden. Das wird normalerweise nur bei Schaltplänen verwendet, die eine Unterschaltung darstellen.

12.3 Komponenten

Dieser Abschnitt beginnt mit <Components> und endet mit </Components>. Er beinhaltet die Schaltkreiskomponenten des Schaltplans. Das Zeilenformat ist wie folgt aufgebaut:

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

- Der type identifiziert die Komponente, z.B. steht R für einen Widerstand und C für einen Kondensator.
- Der name ist der Komponentenidentifizierer in dem Schaltplan, z.B. steht R1 für den ersten Widerstand.
- Eine 1 in dem active Feld zeigt an, dass die Komponenten aktiv ist, d.h. dass sie während der Simulation verwendet wird. Eine 0 zeigt an, dass die Komponente nicht aktiv ist.
- Die nächsten beiden Zahlen sind die x- und y-Koordinaten des Komponentenzentrums.
- Die folgenden beiden Zahlen sind die x- und y-Koordinaten der linken oberen Ecke des Komponententextes. Sie sind relativ zum Komponentenzentrum.
- Die nächsten beiden Zahlen zeigen an, ob die Komponente an der x-Achse gespiegelt ist (1 für gespiegelt, 0 für nicht gespiegelt) ist und ob die Komponente entgegen des Uhrzeigersinns gedreht ist (Vielfache von 90 Grad, d.h. 0...3).
- Die nächsten beiden Einträge sind die Werte der Komponenteneigenschaften (in Anführungszeichen) gefolgt von einer 1, falls die Eigenschaft in dem Schaltplan angezeigt wird (ansonsten eine 0).

12.4 Verbindungen

Der Abschnitt beginnt mit <Wires> und endet mit </Wires>. Er beinhaltet die Drähte (elektrische Verbindungen zwischen den Schlatkreiskomponenten) und ihre Bezeichnungen bzw. zusätzlichen Eigenschaften. Das Zeilenformat sieht wie folgt aus:

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

- Die ersten vier Zahlen sind die Koordinaten des Drahtes in Pixel: x-Koordinate des Startpunktes, y-Koordinate des Startpunktes, x-Koordinate des Endpunktes und y-Koordinate des Endpunktes. Alle Drähte müssen entweder horizontal (beide x-Koordinaten gleich) oder vertikal (beide y-Koordinaten gleich) sein.

- Die erste Zeichenkette in Anführungszeichen ist der Name des Bezeichners. Er ist leer, wenn der Benutzer keine Drahtbezeichnung eingegeben hat.
- Die nächsten beiden Zahlen sind die x- und y-Koordinaten der Bezeichnung oder Null, falls es keine Bezeichnung gibt.
- Die folgenden beiden Zahlen sind der Abstand zwischen dem Startpunkt des Drahtes und dem Punkt, an dem der Bezeichner des Drahtes angezeigt werden soll.
- Die letzte Zeichenkette in Anführungszeichen ist der Anfangswert für die Knotenspannung an diesem Draht. Sie ist leer, falls der Benutzer keine Knotenspannung für diesen Draht angegeben hat..

12.5 Diagramme

Der Abschnitt beginnt mit *<Diagrams>*‘ und endet mit `</Diagrams>`. Er beinhaltet die Diagramme mit ihren Kurven und Markierungen.”

```
<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">
```

- Die ersten beiden Zahlen sind die x-und y-Koordinaten der linken unteren Ecke.
- Die nächsten beiden Zahlen sind die Breite und Höhe der Diagrammgrenzen.
- Die fünfte Zahl ist 1 falls das Gitternetz angezeigt werden soll und 0 falls nicht.
- Das nächste ist die Farbe des Gitternetzes als hexadezimaler 24-Bit RGB-Wert, z.B. ist #FF0000 rot.
- Die nächste Zahl legt den Stil des Gitternetzes fest.
- Die nächste Zahl legt fest, welche Achsen eine logarithmische Einteilung haben.

12.6 Zeichnungen

Der Abschnitt beginnt mit `<Paintings>` und endet mit `</Paintings>`. Er beinhaltet die Zeichnungselemente, die sich in dem Schaltplan befinden.”

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).

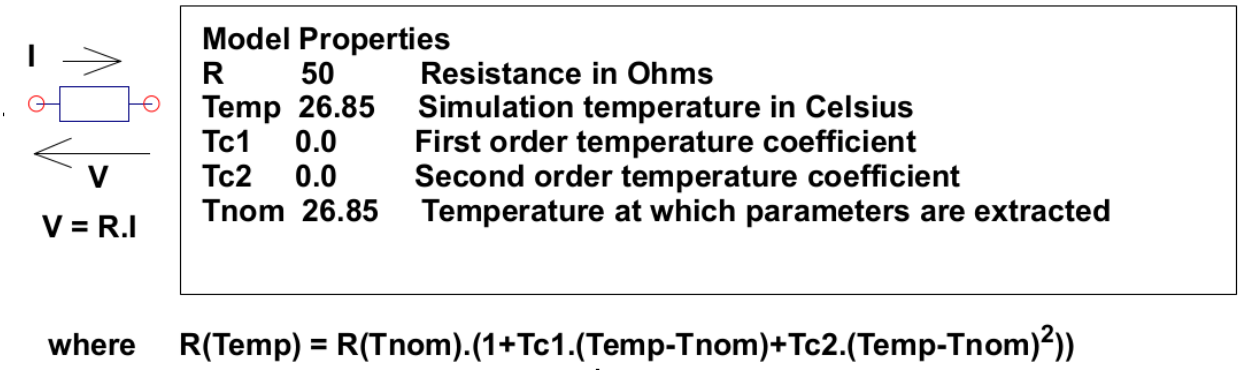


Figure 1 - Qucs built-in resistor model.

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.

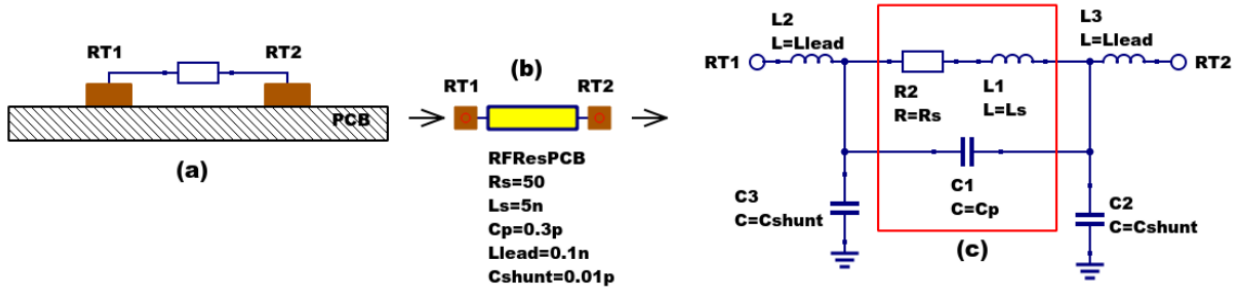


Figure 2 - PCB mounted resistor: (a) axial component mounting, (b) Qucs symbol and (c) equivalent circuit model.

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.

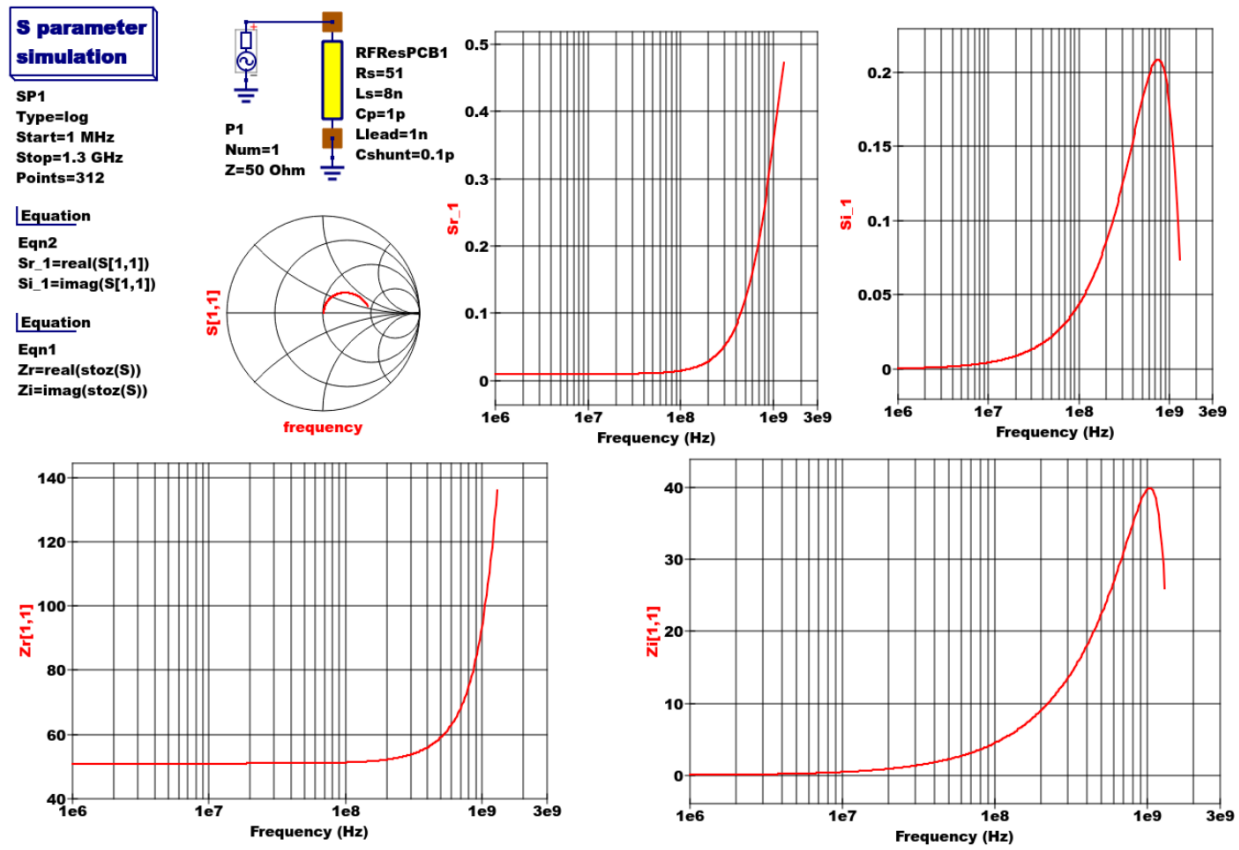


Figure 3 - Qucs S parameter simulation test circuit and plotted output data for a MF axial resistor: *Rs*=51Ω, *Ls*=8nH, *Cp*=1pF, *Llead*=1nH and *Cshunt*=0.1pF.

13.3 Analysis of the RF resistor model

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

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

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

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

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

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

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

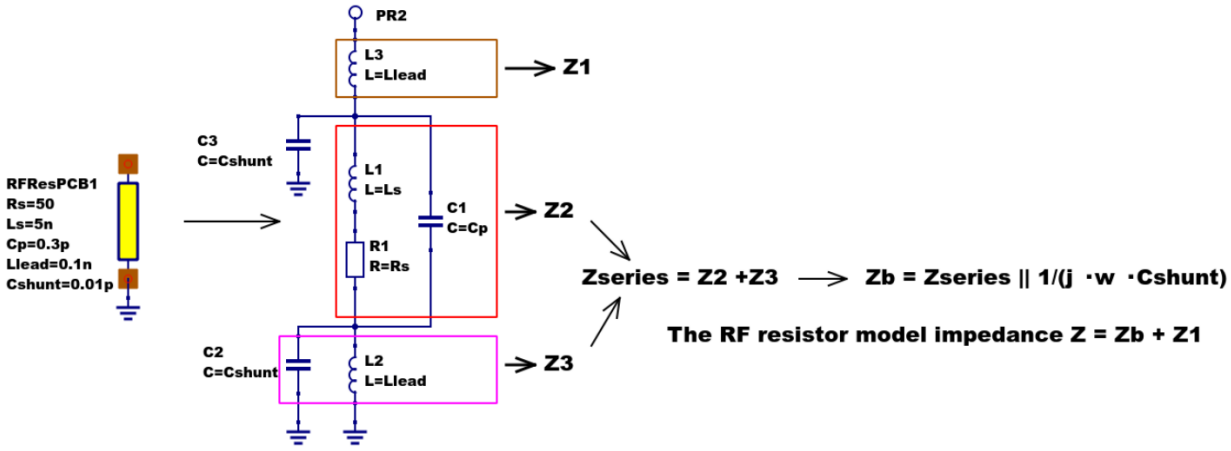


Figure 4 - RF resistor model rotated through 90 degrees and connected with one terminal grounded, similar to the test circuit in Figure. Sections of the model are shown grouped for calculation of the model impedance Z.

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.

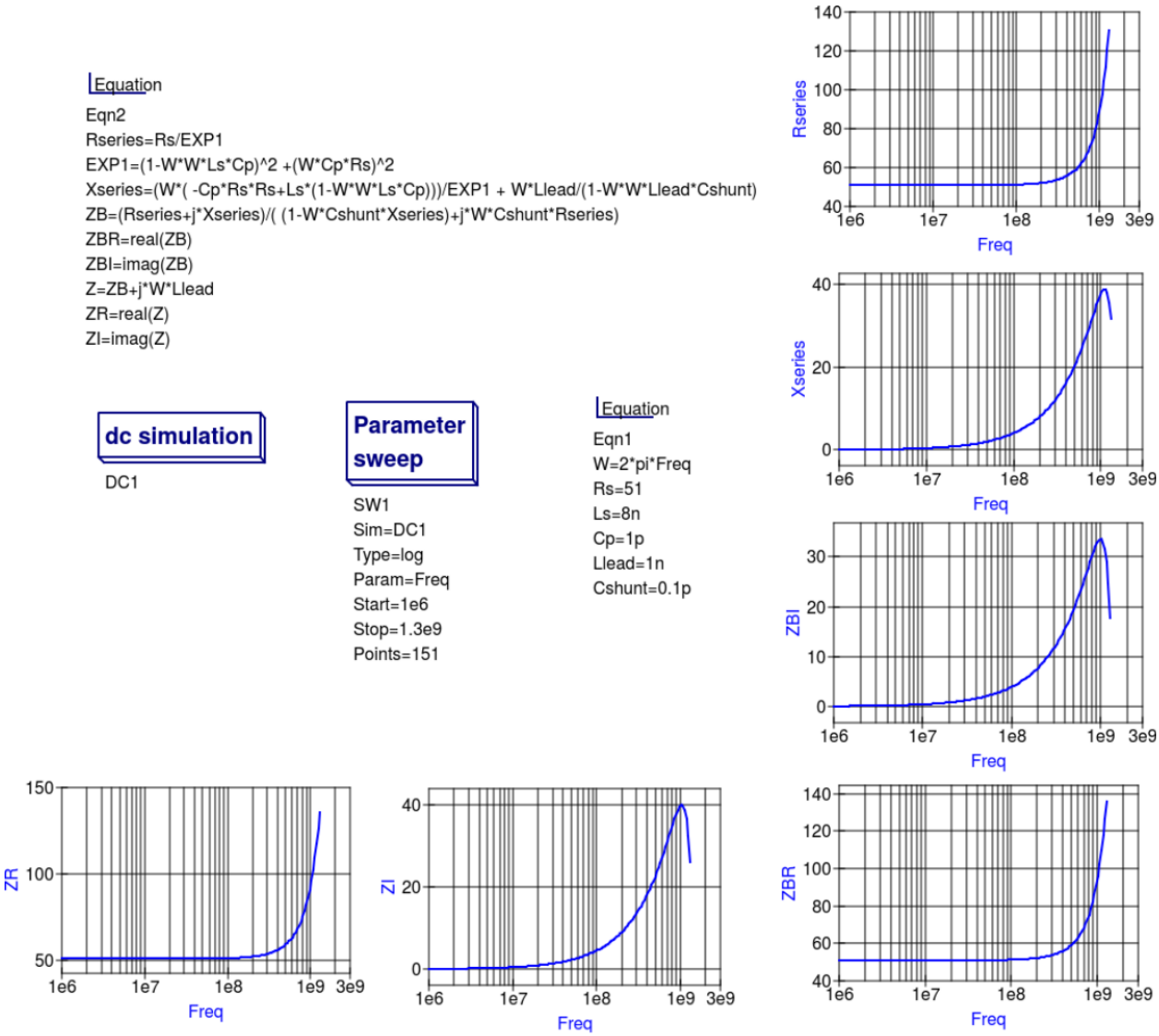


Figure 5- Theoretical analysis of RF resistance impedance Z using Qucs post processing facilities: note a dummy simulation icon, in this example DC simulation, is required to force Qucs to complete the analysis calculations.

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.

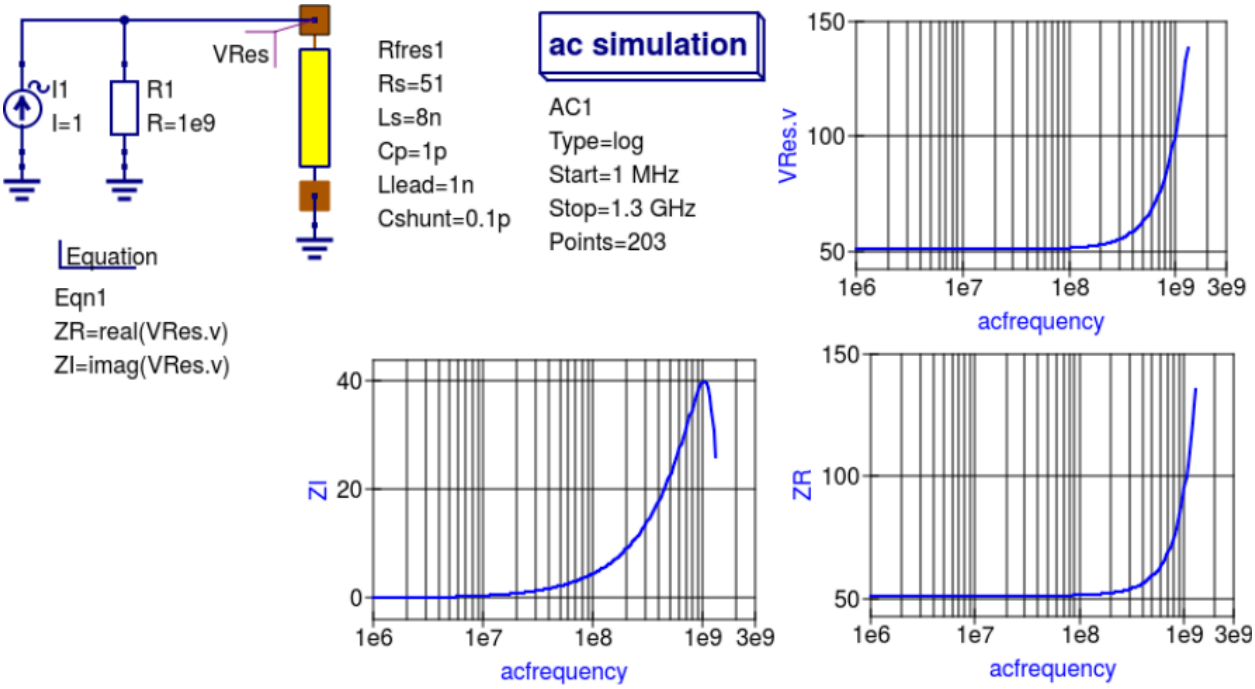


Figure 6 -A simple Qucs test circuit for demonstrating the use of an AC constant current source to measure electrical network impedance.

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.

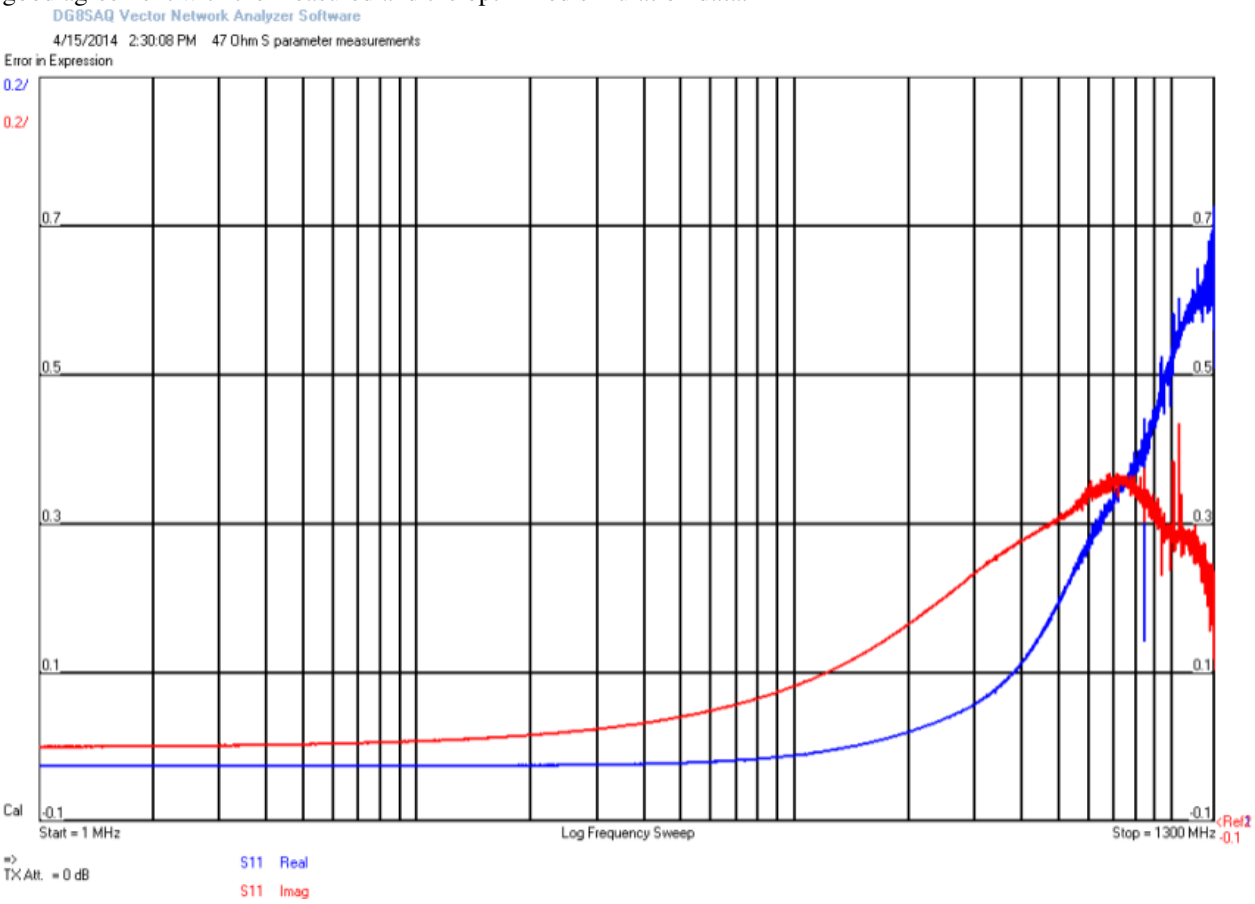


Figure 7 - DGSAQ Vector Network Analyser S parameter measurements for a 47 Ω axial RF resistor.

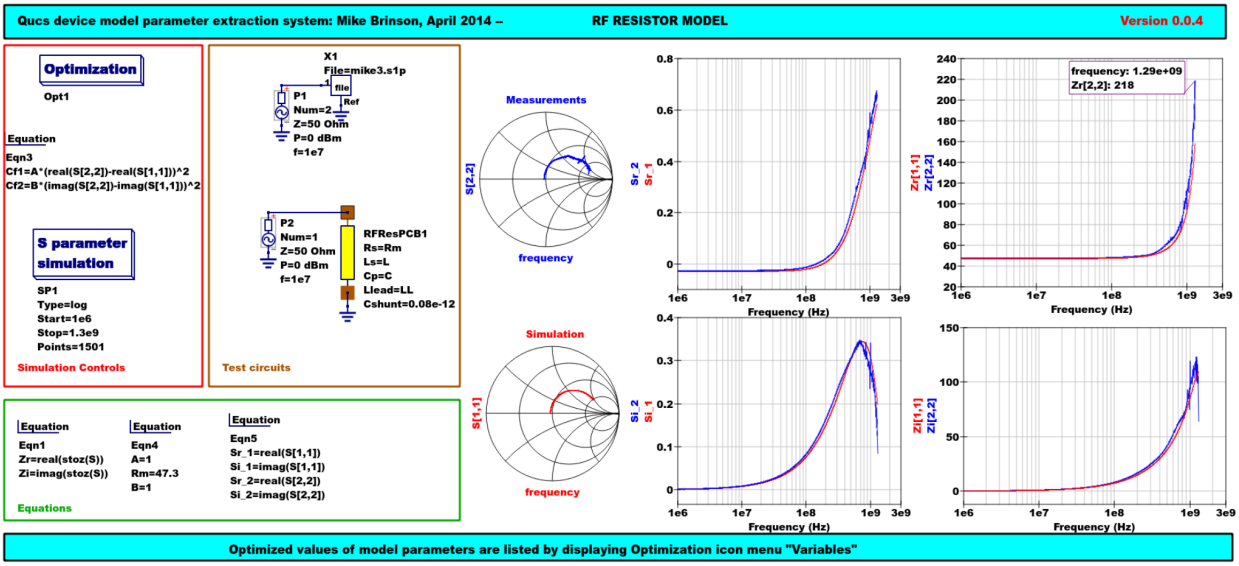


Figure 8 - Qucs device model parameter extraction system applied to a nominal 47 Ω resistor represented by the subcircuit model illustrated in Figure 2 (c). Fixed model parameter values: **Rs = Rm = 47.3** Ω, **CShunt = 0.08pF**; Optimised values: **Ls = L = 10.43nH**, **Llead = LL = 1.47nH**, **Cp = 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.

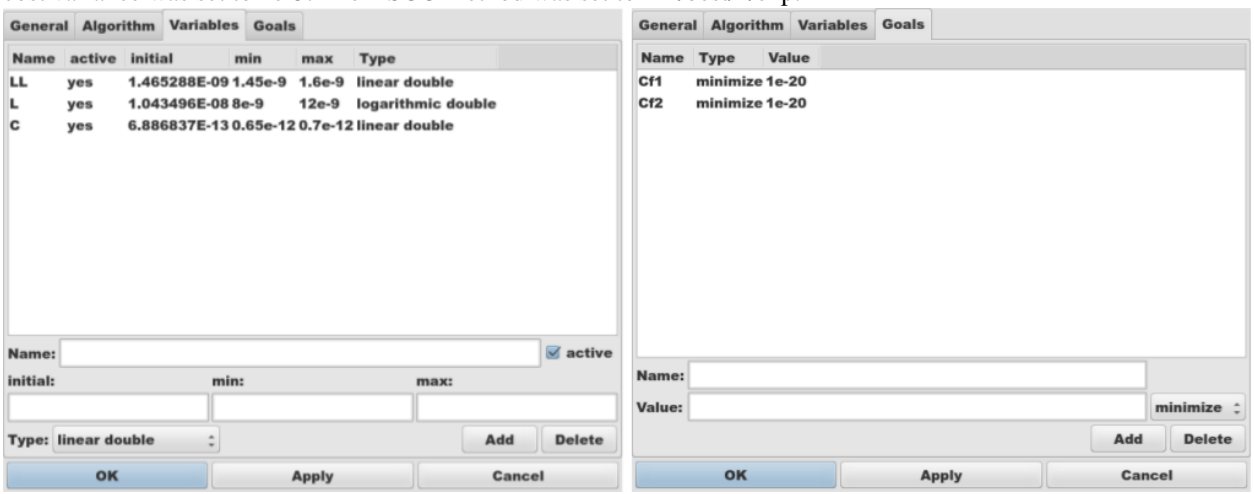


Figure 9 - Qucs Minimization Icon drop down menus: left ”Variables“ and right ”Goals‘‘.

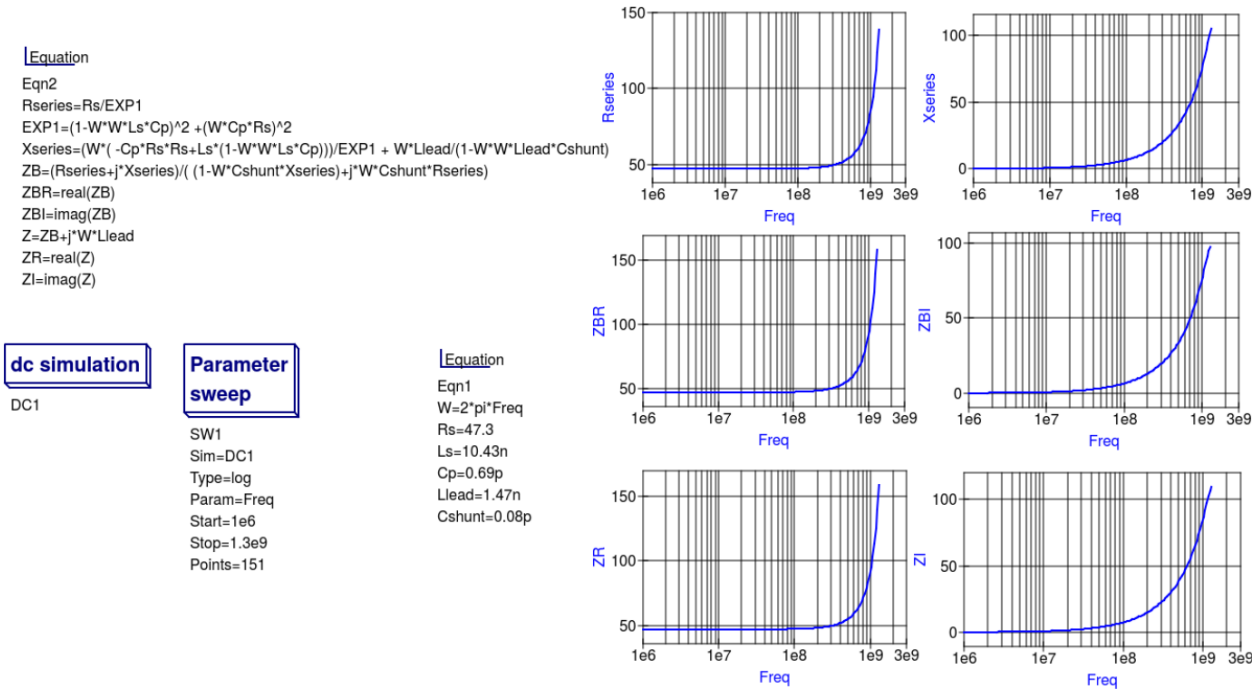


Figure 10 - Qucs simulation of nominal 47 Ω resistor based on theoretical analysis.l

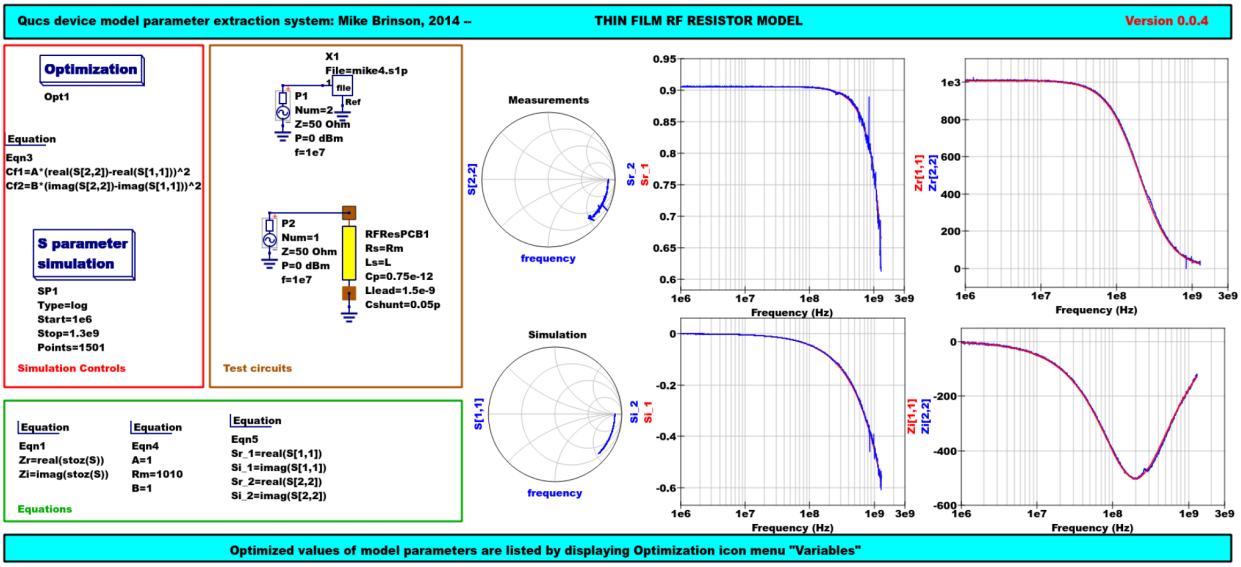


Figure 11 - Qucs device model parameter extraction system applied to a nominal 1000 Ω resistor represented by the subcircuit model illustrated in Figure 2(c).

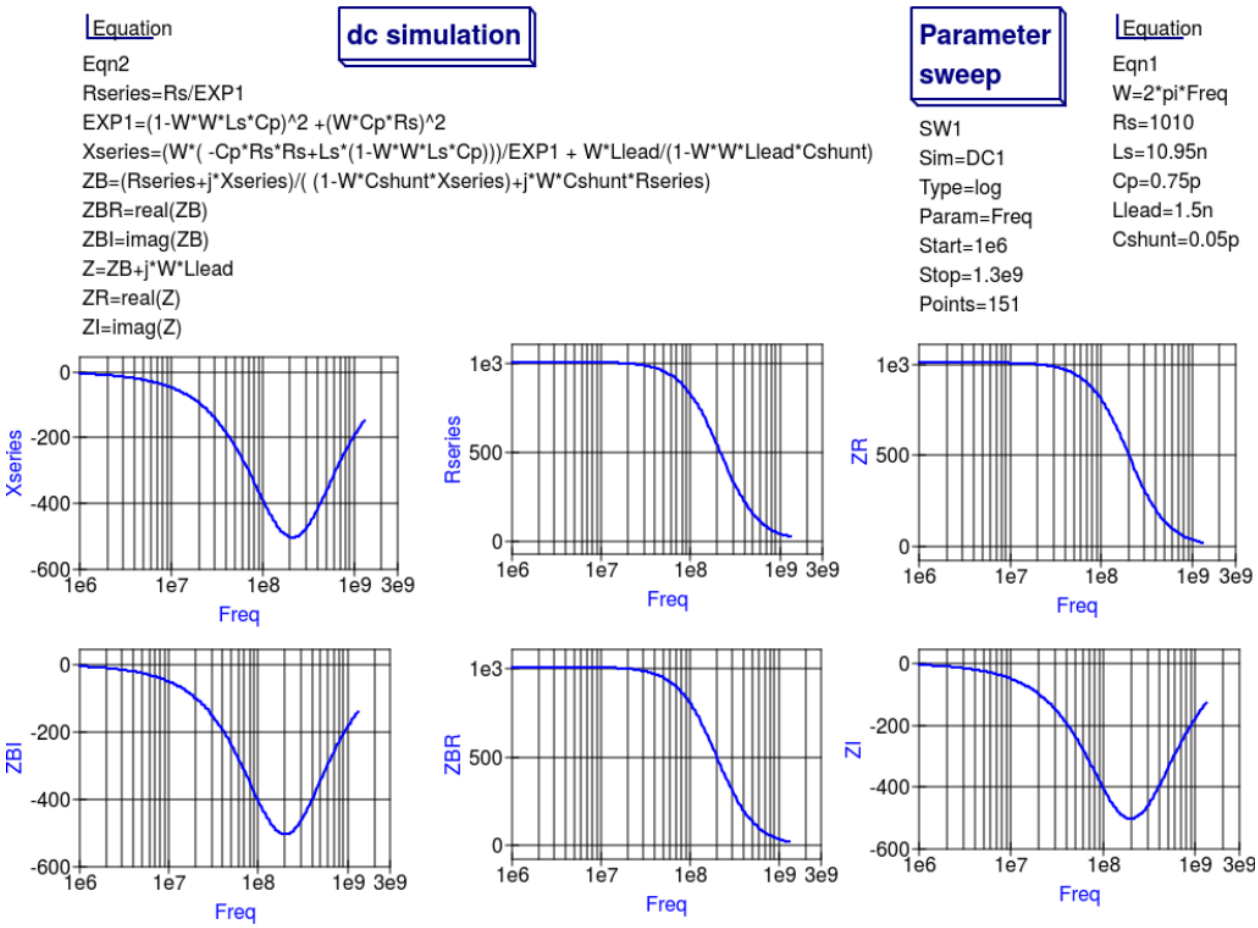


Figure 12 - Qucs simulation of nominal 1000 Ω resistor based on theoretical analysis.

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.

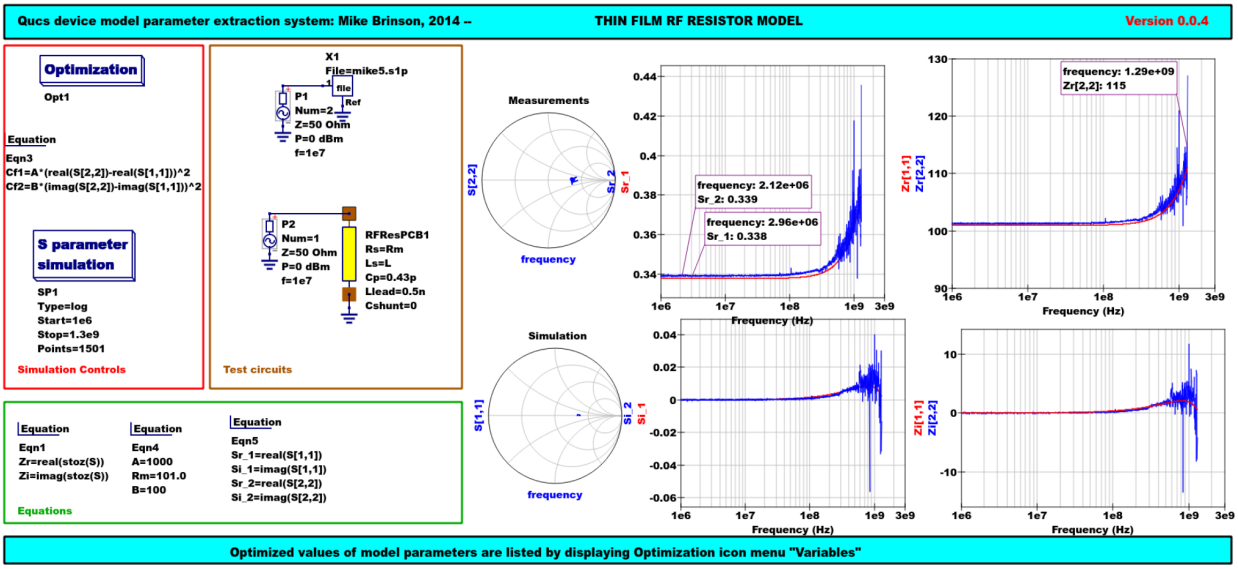


Figure 13 - Qucs device model parameter extraction system applied to a nominal 100 Ω SMD resistor represented by the subcircuit model illustrated in Figure 2 (c).

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");
```

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

```
parameter real Llead = 0.1e-9  from [1e-20 : inf)
 `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.

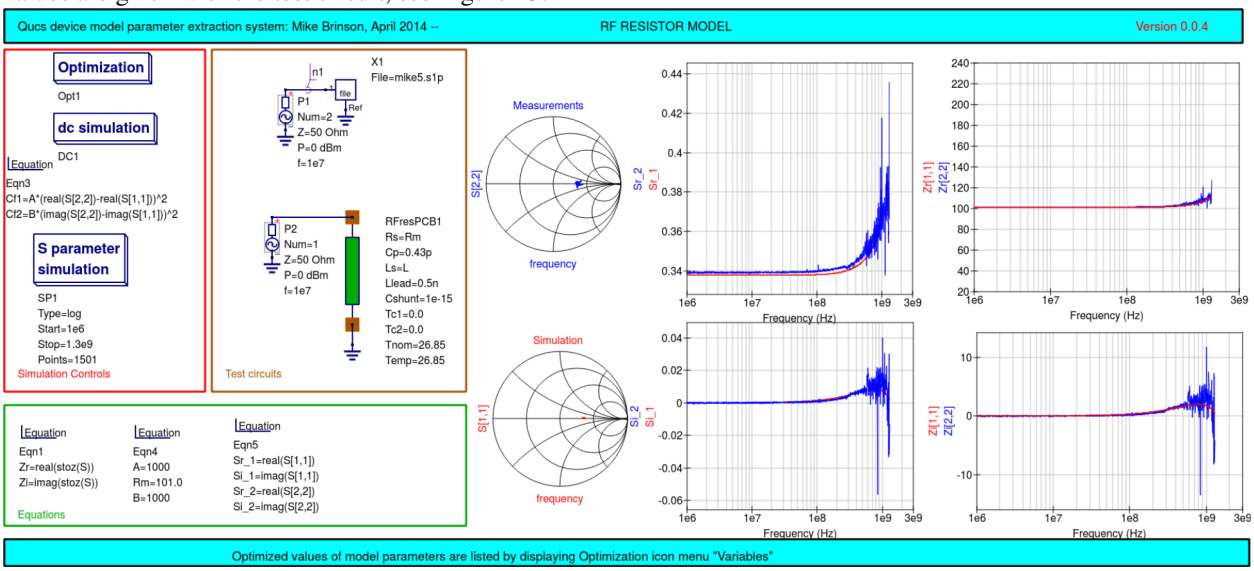


Figure 15 - Verilog-A models parameter data extraction for a 100 Ω axial thin film resistor. Fixed model parameter values: **Rs = Rm =101** Ω, **CShunt = 1e-15 F**, **Llead = LL = 0.5nH**, **Cp = C = 0.43pF**; Optimised values: **Ls = L = 3.99nH**. To reduce simulation time the ASCO cost variance was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

Technische Beschreibungen den Simulator betreffend

sind erhältlich unter http://qucs.sourceforge.net/tech/technical.html

Beispielschaltungen

sind erhältlich unter http://qucs.sourceforge.net/download.html#example

[5] Qucs release 0.0.18, or greater.