Chapter 6
Qucs modelling and simulation of analog/RF devices and circuits

M.E. Brinson
Centre for Communications Technology, London Metropolitan University,
London, UK.
e-mail: mbrin72043@yahoo.co.uk

Abstract: Trends in compact device modeling and analog circuit simulation point towards a growing interest among the modeling community in the standardization of Verilog-A as an equation based modeling language for compact semiconductor device model and circuit macromodel development. . This chapter introduces the principles of compact device modeling with equation-defined devices and Verilog-A models. For completeness circuit macromodel principles and construction are also included. It also describes the use of the different types of equation based models in analog and RF circuit simulation. Throughout the text the properties of a range of analog and RF circuits with different levels of complexity are introduced and their performance investigated with the "Quite universal circuit simulator" (Qucs) and its related software package QucsStudio. All the device and circuit modeling techniques introduced in this chapter form part of the standard features implemented in Qucs and QucsStudio.

Keywords: Qucs; QucsStudio; analog and RF circuit simulation; macromodeling; equation-defined device modeling; verilog-A compact device modeling; post-simulation data analysis; SPICE.

6.1 Introduction

The structure of a conventional analog circuit simulator consists essentially of two interlinked parts; firstly a simulation engine and secondly models of simple components and more complex devices. Both parts are equally important, for without adequate functionality in either the software is of little practical value. Early analog circuit simulator engines [1] included linear and non-linear DC analysis , small signal AC analysis and transient circuit analysis. Later developments [2] added small signal AC noise analysis , pole-zero analysis and in some instances large signal distortion analysis to the list of available analysis features. More recently, this list has

been extended to include RF circuit analysis through the addition of scattering parameter simulation [3], harmonic balance techniques [4] and other high frequency circuit analysis procedures [5]. Analog circuit simulation software is normally distributed with built-in component models. A very minimal set comprises models for the fundamental R, C and L components, current and voltage sources, and a range of diode, BJT, JFET and MOSFET semiconductor devices. In most circuit simulators these are "hard wired" into the FORTRAN, C or C++ simulator code, making the inclusion of new semiconductor device models difficult without a good working knowledge of advanced programming techniques and a simulators "Model Programming Interface". In many instances the effort involved in adding new models to a simulator is considerable, taking significant amounts of time, particularly in the case of models of nonlinear devices which require the partial derivatives of device currents and charges to be generated for DC and transient simulation. Throughout model construction considerable care is needed to ensure that new models work correctly and do not inadvertently introduce current or charge discontinuities in the modeled device characteristics. It is also worth noting however, that device models built using a compiled programming language can result in very efficient run time models which have reduced simulation times, especially during transient analysis. For this reason compact semiconductor device modelling in FORTRAN, C or C++ has become one of the preferred techniques among the modeling community. Today, modeling trends suggest that there is a growing interest in simulation software packages licensed under the GNU General Public license (GPL), or other open source licenses, which are linked to device modeling tools involving either the VHDL-AMS [6] or the Verilog-AMS [7] hardware description languages. [1] One circuit simulator in this category is the 'Quite universal circuit simulator' (Qucs) [8] which employs the ADMS analog Verilog-A model synthesizer [9] for compact model development. To meet the diverse modeling needs of different groups of users, modern circuit simulation software is having to adapt to provide built-in device modeling facilities that allow users the opportunity to experiment with new simulation models. [2] These can be of widely differing complexity. One of the primary aims of Qucs is to offer users a range of flexible simulation and device modeling tools which can be easily understood and applied to circuit design and model construction regardless of a users knowledge of advanced electronics or device modeling techniques. This approach has had a strong influence on the Qucs software development team by encouraging the inclusion of features which support the construction of models built from simple electrical networks to complex communications systems. This chapter presents a number of different device modeling techniques, concentrating primarily on macromodeling, equation-defined device modeling and Verilog-A compact semiconductor device modeling. All of these modeling techniques are part of the standard features

[1] Analog hardware description languages include routines for the automatic generation of the partial derivatives of device currents and charges, making compact model construction a simpler process when compared to directly coding nonlinear models in C++ or other programming languages.

[2] Feedback from the Qucs sourceforge.net Internet site indicate that users of the software range from physics teachers in schools to PhD students, engineers, scientists and other established academic researchers

implemented in Qucs and its closely related software package QucsStudio [10]. An important general aim of this chapter is to encourage readers to try for them-selfs developing device models using one or more of the described modeling techniques. The text includes a number of example models taken from analog circuit design and RF circuit engineering.

6.2 Circuit and system modeling using macromodels

6.2.1 Subcircuits and macromodels

The concepts underpinning subcircuits and their application in analog circuit simulation were introduced in Chapter 5 of this book. Subcircuits are a powerful tool in the repertoire of modeling features commonly implemented in circuit simulation software. Their primary task is to group a set of components as an independent item which can be used over and over again. Subcircuits are identified by a unique schematic symbol or netlist name. In general the circuit within the body of a subcircuit consists of a number of predefined components, user defined subcircuits and macromodels, and Qucs library elements, which are connected to outside circuits by a number of specified input and output signal terminals. Although macromodels are in many ways similar to subcircuits they are subtle different. Their primary purpose is to functionally model the terminal characteristics of a device or circuit, if applicable in all simulation domains, while minimizing the complexity of the circuitry within a macromodel body. In the early days of analog circuit simulation macromodels allowed integrated circuit component level models to be replaced by functional blocks which simulated with a significantly faster speed. One of the best known integrated circuit macromodels was developed for operational amplifiers by Boyle et al. [11]. The process employed for constructing macromodels is essentially the same as that employed for subcircuits. Hence, many of the schematic capture techniques introduced previously are also applicable to macromodels.

6.2.2 Building a special purpose signal generator subcircuit/macromodel

Figure 5.9 illustrates a transient test circuit for a two phase switched capacitor integrator with series input and output sample and hold circuits and a low-pass output filter. Strictly, the subcircuits forming the switched capacitor integrator circuit are not all pure subcircuits but models which have a mixture of subcircuit and macromodel attributes. The simplest of these is the non-overlapping two phase clock generator shown in Figure 6.1. This model has a subcircuit structure but attributes which

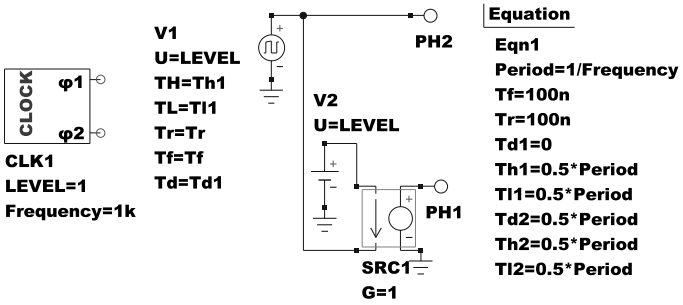


Fig. 6.1 Non-overlapping two phase clock generator macromodel symbol and internal electrical network.

have the functional properties of a macromodel. The non-overlapping time signals [3] of outputs $\phi 1$ and $\phi 2$ are obtained by inverting output $\phi 2$ to give second output $\phi 1$. Macromodel parameters *Frequency* and *LEVEL* set the output signal clock period and voltage amplitude respectively.

6.2.3 A two phase switched capacitor sample and hold macromodel

The sample and hold circuit illustrated in Figure 6.2 also has the properties of a subcircuit but the functionality of a macromodel. In this circuit the sample and hold function is synthesized with Qucs voltage controlled switches and a number of other fundamental electrical components. This macromodel in no way resembles the actual circuit of an integrated or discreet component version of a sample and hold device. Switch *S1* is closed when clock signal *PH1* is high. [4] While *S1* is closed capacitor *C* charges with time constant $C \cdot Ron$, causing its voltage to follow the input signal voltage. In the second half of the clock period switch *S1* is open and *S2* closed, Due to the fact that the time constant $C \cdot Roff$ [5] is very large compared to the capacitor charging time constant, the voltage across *C* now remains constant and the voltage at output pin *Pout* becomes identical to the capacitor charging voltage. Figure 6.3 presents a typical set of signal waveforms obtained from the transient simulation of the sample and hold macromodel when driven by a two phase clock generator with a period forty times smaller than the 1kHz input signal.

---

[3] These signal are different at all points in a clock cycle except for the time when the both output signals have an amplitude equal to LEVEL/2.

[4] The reverse is true for voltage controlled switch *S2* due to the non-overlapping nature of the two phase clock generator.

[5] Strictly, the series resistance of the path from *C* through switch *S2* to the voltage controlled source *SRC2* is $Roff$ plus the input resistance of *SRC2*, making the actual path resistance much greater than $Roff$, ensuring that the capacitor voltage does not change during $\phi 2$ of the clock cycle.

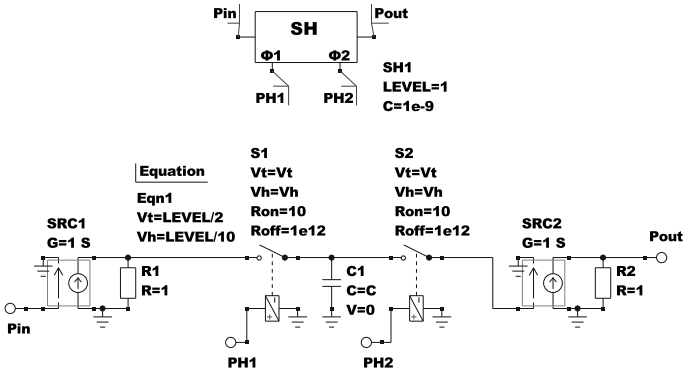


Fig. 6.2 A two phase sample and hold circuit symbol and internal electrical network.

6.2.4 A two phase switched capacitor inverting integrator macromodel

The magnitude of the switched capacitor inverting integrator gain [27], *gainInt*, shown in Figure 6.4 is given by equation 6.1

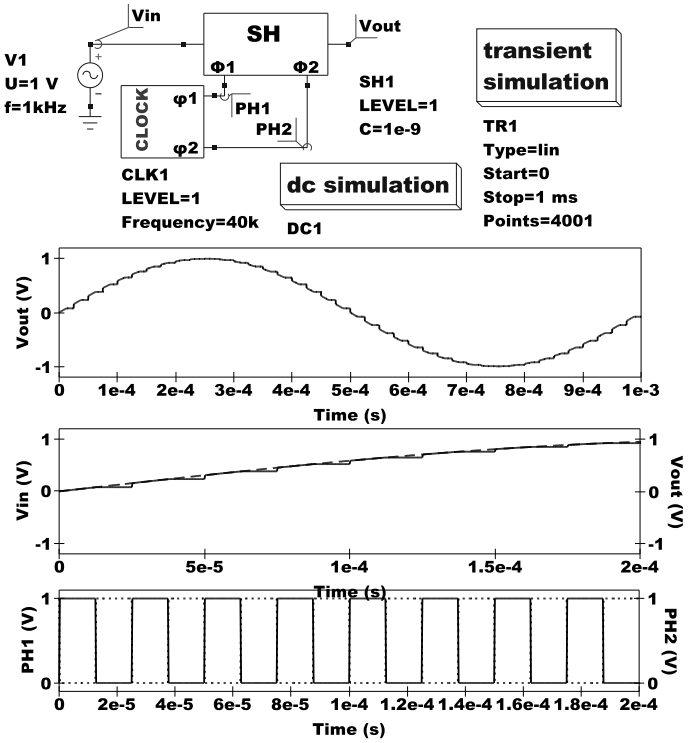


Fig. 6.3 A two phase switched sample and hold test circuit and simulation waveforms; middle waveforms: Vin left-side scale, dashed line; Out right-scale, solid line; bottom waveforms; PH1 left-side scale , solid line; PH2 right-side scale , dotted line.

$$gainInt = \frac{\omega_0 \cdot (\omega \cdot T/2)}{\omega \cdot sin((\omega \cdot T/2))} \tag{6.1}$$

Where $\omega_0 = C_1/(T \cdot C_2)$ radians, $\omega = 2 \cdot \pi \cdot f_s$ radians , and $T = 1/f_c$ seconds. Here, fs is the input signal frequency in Hz and f_c is the two phase clock frequency in Hz. The macromodel circuit shown in Figure 6.4 illustrates the construction of a complex macromodel where the components forming the body of the macromodel have differing hierarchies. In this particular example a mixture of fundamental components plus two additional macromodels, *SWcapPC* and *SPA*, make up the circuit. The value of component *C2* is determined from the design equations located in Qucs Equation block Eqn1 plus the numerical values of macromodel parameters *C1*, *gainInt*, f_s and f_c. Macromodel *SWcapPC* represents a functional model of a parallel switched capacitor realisation of a continuous signal domain resistor of value $R = T/C_1 = 1/(f_c \cdot C_1)$ Ohms. Macromodel *SPA* represents a basic single pole operational amplifier. This macromodel is an extension of the Qucs simple operational amplifier model produced by the addition of a DC input offset voltage source, differential input resistance, differential input capacitance, output resistance and a single pole voltage gain transfer function to the basic Qucs operational amplifier model. [6]

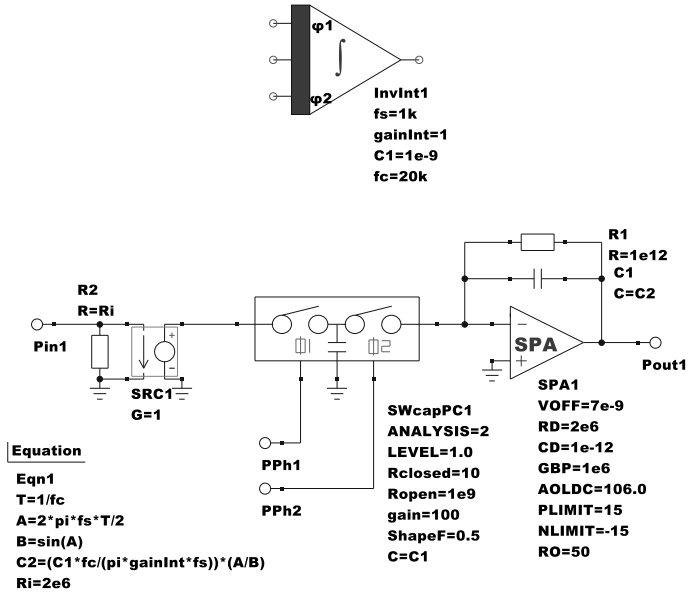


Fig. 6.4 A two phase switched capacitor inverting integrator macromodel.

[6] The single pole voltage transfer characteristic is determined by the macromodel parameters *GBP* (differential gain bandwidth product), and *AOLDC* (open loop DC gain in dB).

6.3 Qucs equation-defined device models

6.3.1 Background

A high percentage of device models include nonlinear current or charge characteristics [12]. Modeling of nonlinear devices requires special simulation circuit elements which allow branch current to be expressed as a function of branch voltage and branch charge to be expressed as a function of both branch voltage and current. Often in the past, this form of highly nonlinear model was not implemented in circuit simulators. However, with the implementation of polynomial nonlinearities in SPICE 2g6 [13] this situation changed, providing subcircuits and macromodels with the capability to model components with nonlinear I/V characteristics. A more generalised form of nonlinear model was included in SPICE 3 with the introduction of the B type nonlinear dependent source. The B dependent source can be specified either as a nonlinear dependent current source or as a nonlinear dependent voltage source defined by algebraic equations written in terms of numeric constants, mathematical operators, predefined functions, component values, node voltages and branch currents. This was a significant break through as it allowed the performance of nonlinear device models to be evaluated during simulation using generalised explicit algebraic expressions.

6.3.2 The structure and properties of Qucs equation-defined devices (EDD)

The Qucs equation-defined device (EDD) [14] can be considered as a superset of the SPICE 3 B type nonlinear dependent source. The EDD device is available in both Qucs and QucsStudio. It comprises one or more [7] two terminal components with electrical properties that are determined by branch current, branch voltage and branch stored charge. Each EDD branch current can be defined by a symbolic voltage function (derived from any branch voltage in the same EDD). Similarly, the EDD stored branch charge can be expressed as a symbolic function of branch voltage and current (again derived from any branch in the same EDD). Qucs EDD models may be combined with other EDD, Qucs built-in component models, user defined and library subcircuits and macromodels and Qucs Equation blocks to form new subcircuits and macromodels. Individual EDD branches are characterised by current I_n, voltage V_n and stored charge Q_n, with $1 \leq n \leq 8$. The properties of the Qucs EDD two terminal component are assumed to be explicit in nature, implying that each branch is defined by:

[7] In the current EDD implementation a maximum of eight two terminal components per EDD are allowed.

$$I_n = I(V_1, V_2 ... V_n) \quad and \quad g = \frac{dI_n}{dV_n} \tag{6.2}$$

$$Q_n = Q(V_1, V_2 ... V_n, I_1, I_2 ... I_n) \quad and \quad c = \frac{dQ_n}{dV_n} = \frac{dQ(V_n)}{dV_n} + \frac{dQ(I_n)}{dV_n} \tag{6.3}$$

Where g is the branch conductance and c is the branch capacitance. EDD branches with $I_n = 0$ and $Q_n = 0$, that are connected to signals originating outside a specific EDD, have the same properties as high impedance probes and may be used as voltage signal sensors. Similarly, EDD branches with $I_n = V_n$ can be used to convert a current to a voltage of the same amplitude. [8] EDD current and charge symbolic equations can include numerical constants, branch variables, variables from Qucs Equation blocks, subcircuit and macromodel parameters, and the mathematical operations and functions defined in the Verilog-A hardware description language. During simulation Qucs automatically creates [9] the EDD branch current and charge partial derivatives needed for nonlinear DC and transient simulation. Qucs EDD are an essential element in equation-defined quantity modelling which implies that the primary elements in the main body of a device model, or a macromodel, are physical quantities expressed as nonlinear equations rather than simple numerical values [15].

6.3.3 A simple EDD signal multiplier macromodel

Conventional linear voltage and current controlled sources can be used to directly add or subtract voltage and current signals. The same is not true for the multiply and divide operations. Shown in Figure 6.5 is an EDD model of a two input multiplier modeled at a functional hierarchical level. The central element in this macromodel is the three branch EDD labelled *D1* in Figure 6.5. Each branch is numbered for easy identification. Branches two and three act as high impedance voltage probes sensing the voltages connected to inputs *IN1* and *IN2*. EDD branch one [10] generates current $I1 = X \cdot V2 \cdot V3$. Current controlled voltage source $SRC1$ has a gain of one Ohm, yielding a voltage of $X \cdot V2 \cdot V3$ amplitude at macromodel output MULT. Figure 6.6 shows an EDD multiplier with two sinusoidal signal inputs (one a carrier signal and the other a modulating signal) that demonstrates the basic features of amplitude signal modulation. The multiplier post transient simulation date is also shown plotted

[8] It is common practice with EDD modeling to use EDD branch currents to calculate the numerical value of device model equations then to convert these to a voltage for use in other EDD (for calculating the numerical values of further model equations) or as output variables.

[9] These are generated in symbolic form by Qucs.

[10] Note that EDD branch one has no voltage applied to it ($V1 = 0.0$ V). It is simply used to generate current $I1$.

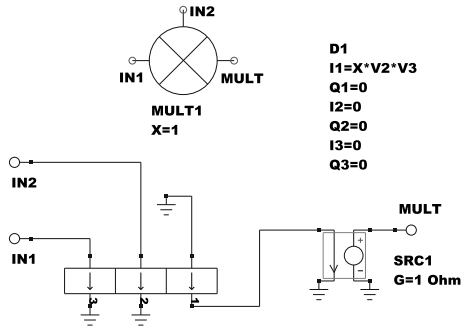


Fig. 6.5 A Qucs EDD macromodel for a two input functional signal multiplier with user definable gain.

in Figure 6.6 for both the time and frequency domains, [11] demonstrating the power of the Qucs post simulation data processing facilities.

6.3.4 Using Qucs EDD to model nonlinear two and multi-terminal devices

One of the major reasons why the EDD element was developed for Qucs was to allow nonlinear two and multi-terminal devices to be easily modeled. An example of a two terminal device is a nonlinear resistance with

$$R = R0 \cdot (1 + A \cdot V_R + B \cdot V_R \cdot V_R), \tag{6.4}$$

where $R0$ is the nominal device resistance in Ω at zero applied voltage, and A and B are voltage dependent resistive coefficients. A single branch EDD model for the nonlinear resistance is presented in Figure 6.7. Notice that in this model the value of R is written as a current contribution in terms of EDD Voltage $V1$ and resistance R. [12] Figure 6.8 shows the nonlinear resistor connected as a resistive voltage divider network where the value of R is a function of applied input voltage Vsw. The value of R varies from 1kΩ to a value around 500kΩ when Vsw is 100V. The plot of R against Vsw illustrated in Figure 6.8 confirms the nonlinear nature of resistor R.

[11] Frequency domain signal spectra were generated using the fast Fourier transform (fft) data analysis function listed in Qucs post-simulation analysis template Equation blocks *Eqn1*, *Eqn3* and *Eqn4*, Figure 6.6.

[12] To ensure that $I1$ can never be infinite if EDD parameter $R0$ is inadvertently set to a value of zero Ohms a small positive value is added to $R0$ in the EDD expression for $I1$. In practical cases this does not affect the accuracy of the nonlinear resistance model.

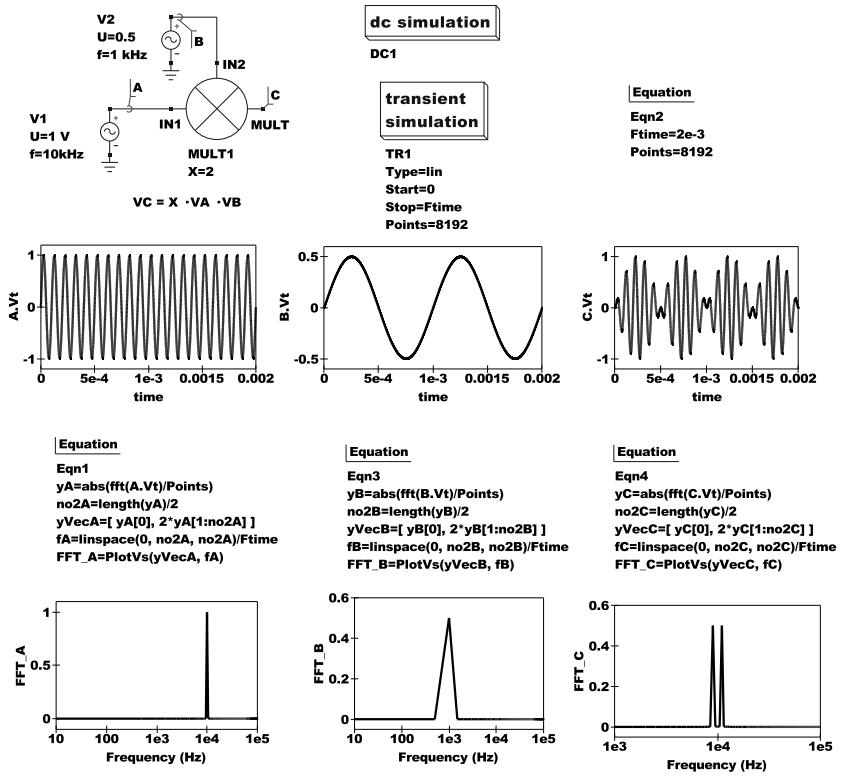


Fig. 6.6 A Qucs EDD functional signal multiplier test circuit and example time domain signal waveforms plus frequency domain signal spectra.

6.3.5 Building a Qucs EDD model for an n type RF MESFET transistor

The Metal and semiconductor FET or MESFET is a Schottky barrier gate FET which is often manufactured from Gallium Arsenide. It has become popular for RF applications because of its high electron mobility and usable gain at high frequencies. The device was developed and modelled by Walter R. Curtice in 1980 while at the RCA Laboratory in Princeton, New Jersey USA [16]. An improved SPICE model for this device was published by Statz et al. (Raytheon) in 1987 [17] and adopted as the MESFET model for the SPICE 3 circuit simulator (semiconductor device type Z). An equivalent circuit for a MESFET model is given in Figure 6.9. Figure 6.10 presents a Qucs EDD MESFET model with a similar structure to the equivalent circuit shown in Figure 6.9. For completeness the EDD model has lead inductance (Lge, Lde and Lse) added. The Qucs EDD compact model of an n type MESFET shown in Figure 6.10 is based on the Curtice hyperbolic tangent model with subthreshold modification [18], D1 and D2 diode currents are modeled by the

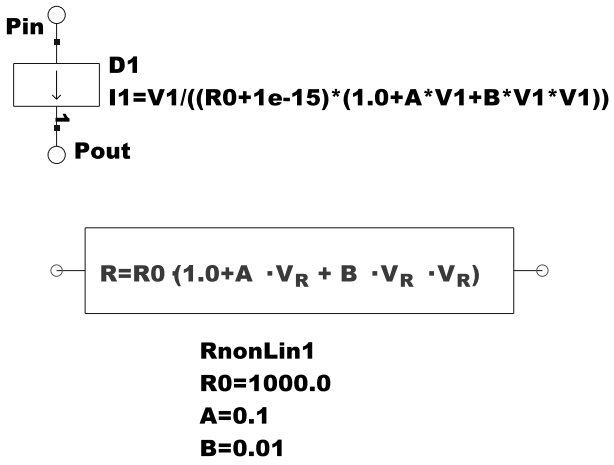


Fig. 6.7 A Qucs EDD model of a nonlinear resistance with linear and quadratic voltage dependent resistive terms.

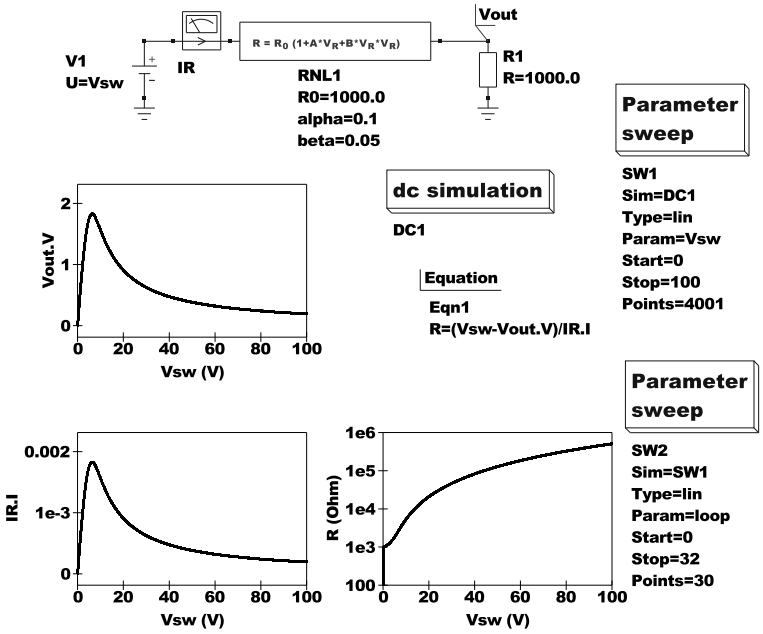


Fig. 6.8 A resistive divider test circuit for a nonlinear R with typical signal waveforms plus extracted values of R plotted as a function of applied voltage Vsw.

full exponential diode equations with the *exp* function replaced by the Verilog-A *limexp* function, MESFET charge is calculated from fixed capacitance plus transit charge. A number of temperature dependent features are also modeled. However, the example EDD Curtice MESFET model does not include any reverse voltage

breakdown effects. Similarly, no attempt has been made to include in the model internally generated noise currents and voltages. [13] To allow easy conversion of the EDD model to Verilog-A code the MESFET model equations are expressed in current contribution form. Definitions for the MESFET parameters employed in the Curtice model can be found in the Verilog-A parameter list for a Qucs Statz MESFET model given in Figure 6.19. Figure 6.11 shows an S-parameter test circuit for simulating the performance of a MESFET transistor, over a wide frequency band, plus a typical set of plotted output data.

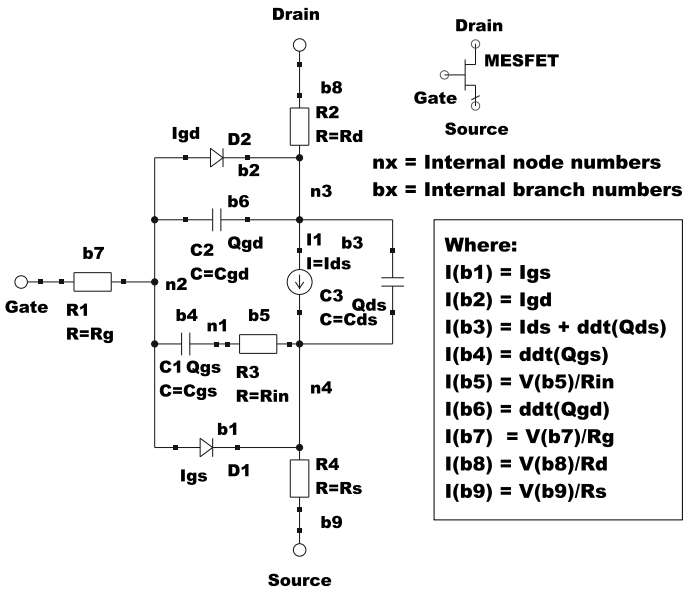


Fig. 6.9 Qucs large signal n type MESHED model with internal branches and nodes numbered.

6.4 Qucs and QucsStudio compact model development with the ADMS Verilog-A model synthesizer/compiler

6.4.1 Qucs implementation

Over the last few years Qucs has evolved from a basic circuit simulator, with RF extensions, to a software package that offers stable analog simulation plus a range of powerful modeling tools for the development of compact device models and integrated circuit macromodels. The simulator component list includes a very stable

[13] See later sections of this chapter for comments on adding noise to compact device models.

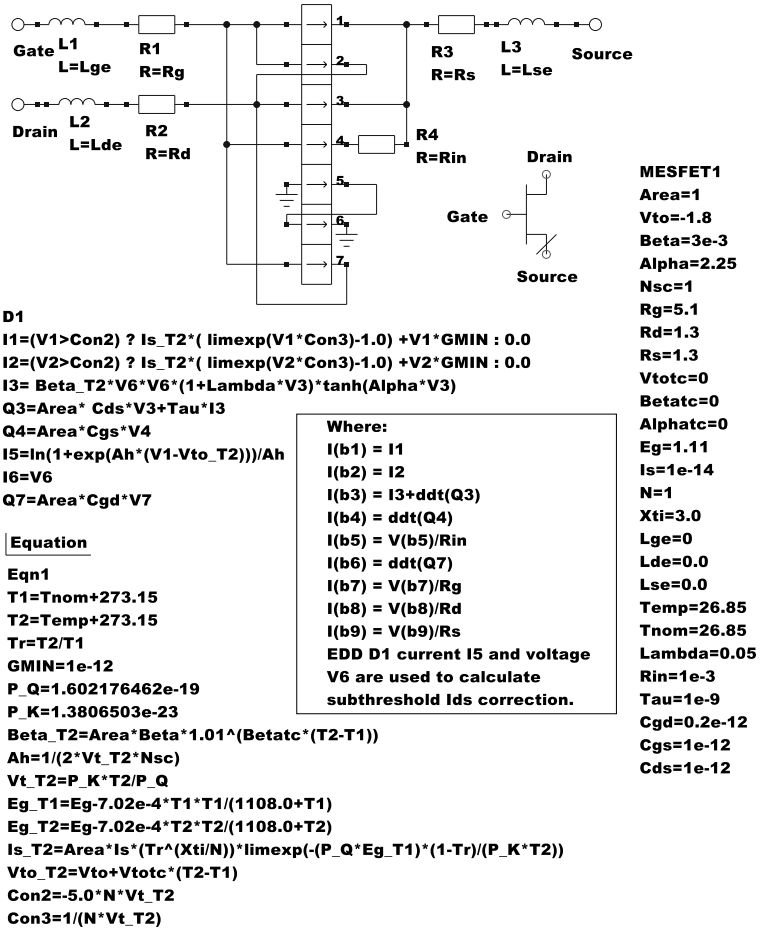


Fig. 6.10 Qucs EDD large signal n type MESFET model and schematic diagram symbol plus parameter list.

equation-defined device (EDD) which allows interactive development of nonlinear component models. However, due to the interactive nature of the Qucs EDD model it is not, in most instances, similar in simulation speed to native C++ models. The great advantage of EDD models is the fact that they are easy to construct while allowing changes to be simply made while testing. On the other hand Verilog-A compact models are normally compiled to C++ code. The compiled code, when linked to to the main body of Qucs code, allows faster operation often approaching the speed obtained by hand crafted C++ device models. The main downside factors to Qucs Verilog-A model development are as follows: a good working knowledge of the Verilog-A hardware description language is an important prerequisite, a good understanding of the Qucs C++ model interface code is essential, and time to complete the development phase must be found. In general the development of Verilog-A

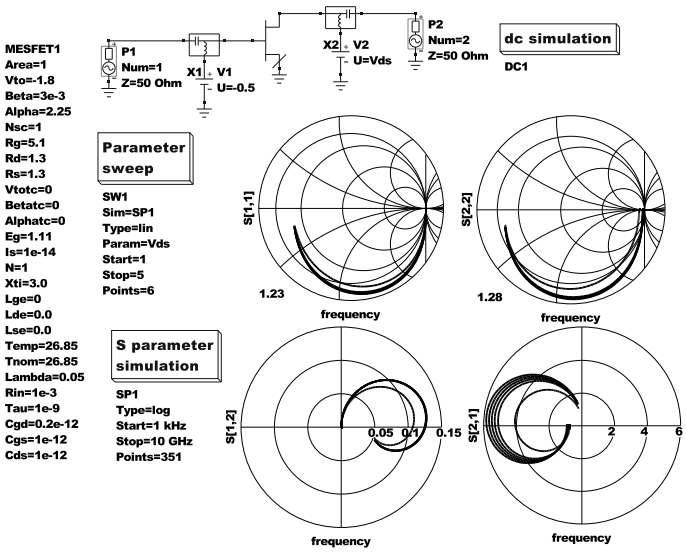


Fig. 6.11 MESFET S-parameter test circuit plus a typical set of plotted simulation data.

models is normally restricted to components where the investment of model development time is justified, for example complex semiconductor models for MOSFET devices or advanced circuit macromodels for switched capacitor mixed-mode circuits. Moreover, it is also advisable to initially test the design of a new model using Qucs EDD based models as a prerequisite to investing time on Verilog-A model development. A series of examples outlining extended semiconductor diode model construction based on a Qucs EDD and a Verilog-A template technique can be found in a recent publication by Brinson, Jahn and Nabijou [19].

6.4.2 Construction of a Qucs Verilog-A model for a non-linear resistor

Qucs version 0.0.16 includes a text editor which selectively colour highlights different statements, numbers and comments in the Verilog-A hardware description language, making entry and checking of compact model code particularly easy. In order to develop Qucs Verilog-A compact device models with the ADMS synthesizer/compiler (version 2.3) it must be installed on a personal computer running the Linux operating system. Illustrated in Figure 6.12 is the Verilog-A code for the non-linear resistor introduced in a previous section. Once the Verilog-A code for a Qucs model is entered and checked, pressing key “F9“ on the keyboard will automatically generate a Qucs schematic symbol for the new model. This is in a simple block form which can be edited using the Qucs ”Painting“ tools to give any desired schematic

```
//  Qucs Verilog-A voltage controlled R model: RnonLin.va.
//  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 2011.
//
`include "disciplines.vams"
`include "constants.vams"
//
module RnonLin(P0, P1);
inout P0, P1;
electrical P0, P1;
`define attr(txt) (*txt*)
parameter real R0    =  1000.0 from [1e-20 : inf]
 `attr(info = "Value of R at V(Rp,Rn) = 0V" unit="V");
parameter real alpha =  0.0    from [-inf : inf]
 `attr(info = "Linear resistive coefficient" unit = "Ohm/Volt");
parameter real beta  =  0.0    from [-inf : inf]
 `attr(info = "Quadratic resistive coefficient" unit = "Ohm/volt^2");
analog begin
// Current contribution
I(P0, P1)  <+  V(P0, P1)/(R0*(1+alpha*V(P0,P1)+beta*V(P0, P1)*V(P0,P1));
end
endmodule
```

Fig. 6.12 Verilog-A code for the example non-linear resistor.

symbol outline. Figure 6.13 presents both the original block symbol and the edited symbol for the nonlinear resistance model. Saving the model symbol causes Qucs

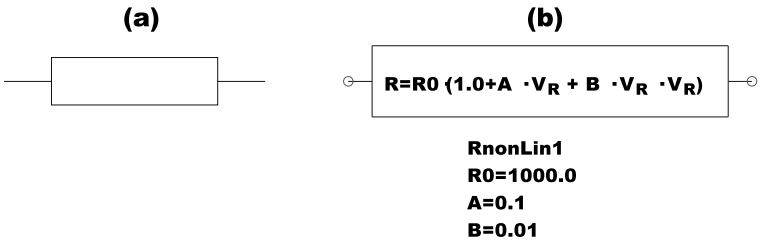


Fig. 6.13 Qucs schematic symbols for the example non-linear resistance: (a) original block symbol and (b) final edited symbol.

to automatically generate C++ code for the new symbol. In the case of the RnonLin model this is held in file "RnonLin.dat" [14]. This code, shown in Figure 6.14, is needed at a later stage of the Verilog-A model development process. The next step in the construction of a Verilog-A model for the non-linear resistor example involves compiling the Verilog-A code with the ADMS Verilog-A compiler. First copy file *RnonLin.va* from project directory *QucsEDDVerilogA_prj* to Qucs source code di-

[14] Note that file "RnonLin.dat" is stored in the development project directory with all the other files associated with the current project. Project *QucsEDDVerilogA_prj* (a sub-directory under directory *.qucs*) in the non-linear resistor example.

```
// symbol drawing code
 Lines.append (new Line (-100, 0, -90, 0, QPen (QColor ("#000080"), 2,
Qt::SolidLine)));
 Lines.append (new Line (-90, 20, 110, 20, QPen (QColor ("#000080"), 2,
Qt::SolidLine)));
 Lines.append (new Line (-90, -20, 110, -20, QPen (QColor ("#000080"), 2,
Qt::SolidLine)));
 Lines.append (new Line (-90, -20, -90, 20, QPen (QColor ("#000080"), 2,
Qt::SolidLine)));
 Texts.append (new Text (-77, -6, "R = R_{0} (1+A*V_{R}+B*V_{R}*V_{R})",
                  QColor("#aa0000"), 8, 1, 0));
 Lines.append (new Line (110, 0, 120, 0, QPen (QColor ("#000080"), 2,
                  Qt::SolidLine)));
 Lines.append (new Line (110, -20, 110, 20, QPen (QColor ("#000080"), 2,
                  Qt::SolidLine)));

 // terminal definitions
 Ports.append (new Port (-100, 0)); /* P0 */
 Ports.append (new Port (120, 0));  /* P1 */

 // symbol boundings
 x1 = -100;   y1 = -20;
 x2 = 120;   y2 = 20;

 // property text position
 tx = -30; ty = 24;
```

Fig. 6.14 Qucs generated schematic capture C++ symbol code for the RnonLin nonlinear resistor example.

rectory /tmp/qucs-core/src/components/verilog. [15] From a terminal window change your working directory to the Qucs Verilog-A directory and compile file *RnonLin.va* with the command:

admsXml RnonLin.va -e qucsVersion.xml -e qucsMODULEcore.xml

Provided the compilation is error free the following message, or similar, is displayed on the computer screen.

```
[info...] admsXml-2.3.0 (1188) Mar 27 2011 13:52:07
[warning] RnonLin: device not handled by the adms qucs interface
[warning] please ensure extra code to be added to the interface
[info...] RnonLin.core.cpp and RnonLin.core.h: files created
[info...] elapsed time: 0 (second)
[info...] admst iterations: 9246 (2127 freed)
```

Repeat the first compilation of file *RnonLin.va* by a second compile using the command:

admsXml RnonLin.va -e qucsVersion.xml -e qucsMODULEdefs.xml

Again, provided the compilation is error free the following message, or similar, is displayed on the computer screen.

```
[info...] admsXml-2.3.0 (1188) Mar 27 2011 13:52:07
[warning] RnonLin: device not handled by the adms qucs interface
[warning] please ensure extra code to be added to the interface
```

[15] This directory reference assumes that the qucs package has been installed using the directions given on the Qucs sourceforge.net web site. Other locations are allowed, using a home directory like for example, /home/mike/qucs-0.0.16/qucs-core/src/components/verilog.

```
[info...] RnonLin.defs.h: file created
[info...] elapsed time: 0 (second)
[info...] admst iterations: 7403 (1367 freed)
```

Repeat the second compilation of file *RnonLin.va* with a third compile using the command:

admsXml RnonLin.va -e qucsVersion.xml -e qucsMODULEgui.xml

Again, provided the compilation is error free the following message, or similar, is displayed on the computer screen.

```
[info...] admsXml-2.3.0 (1188) Mar 27 2011 13:52:07
[warning] RnonLin: device not handled by the adms qucs interface
[warning] please ensure extra code to be added to the interface
[info...] RnonLin.gui.cpp and RnonLin.gui.h: files created
[info...] elapsed time: 0 (second)
[info...] admst iterations: 7522 (1342 freed)
```

Repeat the third complation of file *RnonLin.va* with a fourth compile using the command:

admsXml RnonLin.va -e analogfunction.xml

Again, provided the compilation is error free the following message, or similar, is displayed on the computer screen.

```
[info...] admsXml-2.3.0 (1188) Mar 27 2011 13:52:07
[info...] RnonLin.analogfunction.h created
[info...] RnonLin.analogfunction.cpp created
[info...] elapsed time: 0 (second)
[info...] admst iterations: 4654 (1085 freed)
```

Provided the instructions above were correctly actioned directory /tmp/qucs-core/src/components/verilog should contain the following files relating to component RnonLin: RnonLin.va, RnonLin.core.cpp, RnonLin.core.h, RnonLin.defs.h, RnonLin.gui.cpp, RnonLin.gui.h, RnonLin.analogfunction.cpp,and RnonLin.analogfunction.h. Copy file RnonLin.gui.cpp, and file RnonLin.gui.h to Qucs directory /tmp/qucs/qucs/components; while leaving a copy of all eight RnonLin files in directory /tmp/qucs-core/src/components/verilog. Change your working directory to Qucs directory /tmp/qucs/qucs/components and rename file RnonLin.gui.cpp to RnonLin.cpp and file RnonLin.gui.h to RnonLin.h. The next stage in the Verilog-A model construction procedure involves making three changes to the RnonLin.cpp C++ file (Figure 6.15); change the file name RnonLin.gui.h to RnonLin.h in line `#include ''RnonLin,gui.h``, and the "T" in the line Name = "T"; to some other more appropriate abbreviation, like "RNL"; replace the text "// put in here symbol drawing code and terminal definitions" at the bottom of file RnonLin.cpp with the C++ code held in file RnonLin.dat, see Figure 6.14; replace the tx and ty lines in file RnonLin.cpp with the following C++ code [16]:

[16] Each time a new model is constructed the initial values for tx and ty will have a different value depending on the size of the new schematic symbol.

```
/*
 * RnonLin.cpp - device implementations for RnonLin module
 *
 * 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.
 *
 */

#include "RnonLin.gui.h"

RnonLin::RnonLin()
{
  Description = QObject::tr ("RnonLin verilog device");

  Props.append (new Property ("R0", "1000.0", false,
    QObject::tr ("Nominal resistance at alpha and beta = 0")
    +" ("+QObject::tr ("Ohm")+")"));
  Props.append (new Property ("alpha", "0.1", false,
    QObject::tr ("Linear resistance coefficient")
    +" ("+QObject::tr ("Ohm/V")+")"));
  Props.append (new Property ("beta", "0.05", false,
    QObject::tr ("Quadratic reistance coefficient")
    +" ("+QObject::tr ("Ohm/(V^2)")+")"));
  Props.append (new Property ("Temp", "26.85", false,
    QObject::tr ("simulation temperature")));

  createSymbol ();
  tx = x2 + 4;
  ty = y1 + 4;
  Model = "RnonLin";
  Name  = "T";
}

Component * RnonLin::newOne()
{
  RnonLin * p = new RnonLin();
  p->Props.getFirst()->Value = Props.getFirst()->Value;
  p->recreate(0);
  return p;
}

Element * RnonLin::info(QString& Name, char * &BitmapFile, bool getNewOne)
{
   Name = QObject::tr("RnonLin");
   BitmapFile = (char *) "RnonLin";

   if(getNewOne) return new RnonLin();
   return 0;
}

void RnonLin::createSymbol()
{
  // put in here symbol drawing code and terminal definitions
}
```

Fig. 6.15 RnonLin.cpp C++ code.

// tx = x2+4;
// ty = y1+4;

tx = -30; ty = -24;

Qucs directory /tmp/qucs/qucs/bitmaps contains 32 bit by 32 bit png graphics files. These files are displayed on the left side of the Qucs main window when the ”Components“ tag is clicked and represent a simple outline of the schematic symbol. However, because of the 32 bit by 32 bit bitmap representation they are often only very approximate pictures of the schematic capture symbol and in general do not contain the detail of the schematic symbol. Fig.6.16 illustrates an enlarged view of simple icon picture for the RnonLin icon. In this picture the non-linear resistive equation has been replaced by the letters RNL. When finished the png file must be saved in Qucs directory /tmp/qucs/qucs/bitmaps as file RnonLin.png, i.e. as the same name as the model name in file RnonLin.cpp. The Gimp Image Editor or KolourPaint program are ideal tools for constructing Qucs component icon pictures. After constructing the RnonLin icon picture and saving it as file RnonLin.png the name of the new model must be added to the component pictures list in file Makefile.am located in directory /tmp/qucs/qucs/bitmaps. Add the name RnonLin.png to the end of list ”XPMS = ...“ and save file Makefile.am. Having constructed the C++ code for the new Verilog-A model, and its associated schematic capture symbol, all that remains to do is to register the new symbol with (1) the qucs-core C++ code and (2) with the qucs C++ code. In this section of these notes merging the new model code with the qucs-core code is presented in detail. The next section continues the same theme and introduces the procedure for registering the new model with the GUI qucs C++ code. Figure 6.17 gives details of the RnonLin entries that have to made to file Makefile.am in directory /tmp/qucs-core/src/components/verilog. After adding the RnonLin model information to file Makefile.am save the modified file in directory /tmp/qucs-core/src/components/verilog. All that remains to do when registering a new model with the qucs-core C++ code is to add the new model to two additional files: change the working directory to /tmp/qucs-core/src/components and open file components.h for editing with a text editor. Add an include statement for the RnonLin model as indicated in the following code section:

```
#include "verilog/swcapZM1.core.h"
#include "verilog/swcapBLInt.core.h"
#include "verilog/RnonLin.core.h"
#include "verilog/dff_SR.core.h"
#include "verilog/tff_SR.core.h"
```

Change the working directory to /tmp/qucs-core/src and open file module.cpp for editing with a text editor. Add a *REGISTER_CIRCUIT* entry for the RnonLin model as indicated in the following code section:

```
REGISTER_CIRCUIT (HPF);
REGISTER_CIRCUIT (swcapZM1);
REGISTER_CIRCUIT (swcapBLInt);
REGISTER_CIRCUIT (RnonLIN)
REGISTER_CIRCUIT (dff_SR);
```

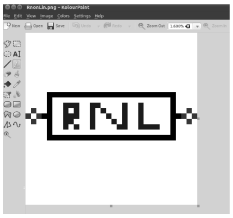


Fig. 6.16 Enlarged picture of a 32 bit by 32 bit RnonLin icon picture

```
# the verilog devices library rules

noinst_LIBRARIES = libverilog.a

libverilog_a_SOURCES = HBT_X.analogfunction.cpp HBT_X.core.cpp \
      hicumL2V2p1.analogfunction.cpp hicumL2V2p1.core.cpp \
                              ▼
       RnonLin.analogfunction.cpp RnonLin.core.cpp \
      \
      dff_SR.analogfunction.cpp dff_SR.core.cpp \
      tff_SR.analogfunction.cpp tff_SR.core.cpp \
                              ▼
       hpribin4bit.analogfunction.cpp hpribin4bit.core.cpp

noinst_HEADERS = HBT_X.analogfunction.h HBT_X.defs.h HBT_X.core.h \
      hicumL2V2p1.analogfunction.h hicumL2V2p1.defs.h hicumL2V2p1.core.h \
                              ▼
        swcapBLInt.analogfunction.h swcapBLInt.defs.h swcapBLInt.core.h \
      RnonLin.analogfunction.h RnonLin.defs.h RnonLin.core.h \
      \
      dff_SR.analogfunction.h dff_SR.defs.h dff_SR.core.h \
      tff_SR.analogfunction.h tff_SR.defs.h tff_SR.core.h \
                              ▼
      hpribin4bit.analogfunction.h hpribin4bit.defs.h hpribin4bit.core.h


VERILOG_FILES = constants.vams disciplines.vams \
 fbh_hbt-2_2a.va hicumL2V2p11.va mod_amp.va hicumL2V2p22.va log_amp.va \


 greytobinary4bit.va comp_1bit.va comp_2bit.va comp_4bit.va hpribin4bit.va \
 SPA.va LPF.va HPF.va swcapZM1.va swcapBLInt.va RnonLin.va

if MAINTAINER_MODE
                ▼
RnonLin.analogfunction.cpp: analogfunction.xml
RnonLin.analogfunction.cpp: RnonLin.va
      $(ADMSXML) $< -e analogfunction.xml
RnonLin.core.cpp: RnonLin.defs.h qucsVersion.xml qucsMODULEcore.xml
RnonLin.core.cpp: RnonLin.va
      $(ADMSXML) $< -e qucsVersion.xml -e qucsMODULEcore.xml
RnonLin.defs.h: qucsVersion.xml qucsMODULEdefs.xml
RnonLin.defs.h: RnonLin.va
      $(ADMSXML) $< -e qucsVersion.xml -e qucsMODULEdefs.xml
RnonLin.gui.cpp: qucsVersion.xml qucsMODULEgui.xml
RnonLin.gui.cpp: RnonLin.va
      $(ADMSXML) $< -e qucsVersion.xml -e qucsMODULEgui.xml
```

Fig. 6.17 Additions to qucs-core Makefile.am located in directory /tmp/qucs-core/src/components/verilog: the black arrows indicate a continuing list.

Note that the adjacent names of the Verilog-A models registered with the version of Qucs that the new RnonLin model is being added too are most likely to vary from the component names given in the last two code segments. The code lists shown are different to the standard Qucs-0.0.16 SVN code due to previously added user constructed Verilog-A models. After editing files components.h and module.cpp make sure that they are saved in their respective directories. Change the current directory to /tmp/qucs/qucs/components and open file *Makefile.am* for editing with a text ed-

itor. Add the RnonLin entry to *Makefile.am* as indicated below in the short segment of C++ code:

```
libcomponents_a_SOURCES= ..............................
            :
LPF.cpp HPF.cpp swcapBLInt.cpp RnonLin.cpp
noinst_HEADERS= ......................................
            :
LPF.h HPF.h swcapBLInt.h RnonLin.h
```

Again all that remains to do when registering a new model with the Qucs C++ code is to add the name of the new model to two additional files: change the working directory to /tmp/qucs/qucs/components and open file *components.h* for editing with a text editor. Add an include statement for the RnonLin model as indicated in the following code section:

```
#include "swcapZM1.h"
#include "swcapBLInt.h"
#include "RnonLin.h"
#include "dff_SR.h"
#include "tff_SR.h"
```

Change the working directory to /tmp/qucs/qucs and open file *module.cpp* for editing with a rext editor. Add a REGISTER_VERILOG statement for the RnonLin model as indicated in the following section:

```
REGISTER_VERILOG (photodiode);
REGISTER_VERILOG (phototransistor);
REGISTER_VERILOG (nigbt);
REGISTER_VERILOG (RnonLin);
```

Note new Verilog-A models are normally added at the end of the *module.cpp* Verilog-A model registration list. After editing files *components.h* and *module.cpp* make sure they saved in their respective directories. The last phase in the construction of a new Verilog-A model for Qucs is to recompile the qucs and qucs-core C++ code. If a new Verilog-A model has been added to the Qucs correctly without problems then compilation of the modified C++ code should take place without error. However, if the C++ compiler reports one or more compilation errors then check the Qucs code section where the errors are reported to have occurred and make the necessary changes to correct them. Finally, test that the new Verilog-A model correctly operates in different simulation domains. In the case of the Verilog-A nonlinear resistor model testing indicates that the model functions correctly with the same results as those shown in Figure 6.8 for the EDD model.

6.5 QucsStudio implementation

QucsStudio is the brain child of German Engineer Michael Margraf. The first version of this software package was released for general use in February 2011. At its core is a new analog simulator developed from Qucs. In some respects this is a second generation Qucs that has evolved from the original project with the express aim of being a test project to create a complete development environment for electrical engineers, incorporating a graphical user interface, an advanced circuit simulator, printed circuit layout and numerical data processing. Although the

core circuit simulator has a lot in common with the original Qucs the new analog simulator is not fully compatible with Qucs. QucsStudio is free software that is allowed to be used and distributed freely. Currently, QucsStudio supports Microsoft Windows®only. It runs without any special installation procedures. Decompressing the QucsStudio distribution "zip" file creates directory QucsStudio with the circuit simulator and other package components inside. It is run by executing `QucsStudio\bin\qucs.exe`. The current production version of the package is QucsStudio-1.2.0.zip. Versions of QucsStudio up to 1.2.0 do not include code for generating Verilog-A compact device models. However, the next release (version 1.3.0) will include a full "turn-key" Verilog-A model development system. [17] One important difference between the original Qucs and QucsStudio Verilog-A development routes concerns the fact that QucsStudio model code is compiled into C++ dynamic linked libraries (dll) rather than the C++ static libraries adopted by Qucs. This change allows model code to be written in Verilog-A, compiled to C++ and then compiled again by a C++ compiler to form a dll component which in turn can be linked to the main body of compiled QucsStudio code without having to recompile the entire circuit simulator code. Hence, it becomes possible to design a user transparent, and user friendly, compact device modeling system [18] which compiles Verilog-A code, without user intervention, each time the code is changed and simulated. As an example of the steps involved in the "turn-key" modeling process the stages needed to build a Verilog-A version of the EDD nonlinear resistor model are described next. The QucsStudio built-in text editor displays Verilog-A code in a colour coded highlighted format, making code entry [19] and debugging of Verilog-A code a straight forward process. Figure 6.18 illustrates, using a simple flow chart, the steps needed to construct a QucsStudio subcircuit version of a Verilog-A compact device model. Starting with a blank schematic drawing sheet a copy of the compiled model symbol [20] is dragged and dropped onto the empty drawing sheet. Notice at this early stage in the process that the compiled model symbol has no numbered pins attached to its square box outline. Stage two involves attaching a Verilog-A or a C++ file to the compiled model symbol. [21] Both types of file must meet the require-

[17] A "turn-key" Verilog-A model construction system is currently being developed by Michael Margraf and Mike Brinson. At this time a working experimental system based on a new compiled C++ model, Verilog-A model code compiled by ADMS 2.3, embedded within a subcircuit is undergoing test. It is expected that QucsStudio 1.3.0 will be published for general use in the near future. To be followed at a later date by a compact device model development system based on the new series 3 version of ADMS.

[18] Hence, removing the need for users to patch the QucsStudio C++ code by hand when adding new models.

[19] These notes make no attempt to explain the fundamentals of the Verilog-A hardware description language or its use in device modeling and circuit macromodeling. Readers who are not familiar with Verilog-A should consult the books by Patrick and Miller [20] and Kundert and Zinke [21]. Details of the latest specification for Verilog-A can also be found in the Verilog-AMS language reference manual.

[20] The compiled model symbol can be found in the QucsStudio "devices" icon list.

[21] This is done using the compiled model icon drop down menu to change the file name entry in the Edit Properties list.

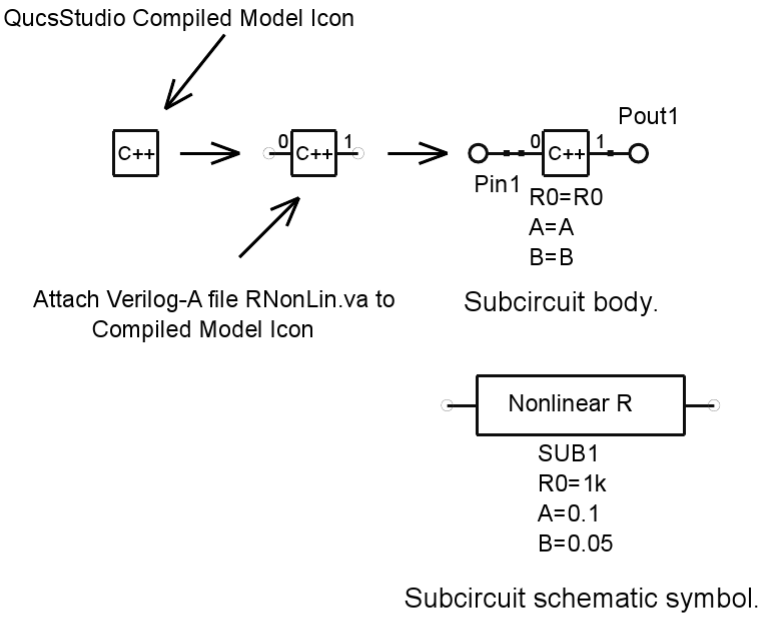


Fig. 6.18 A flow chart depicting the stages in the construction of a QucsStudio Verilog-A subcircuit model.

ments of the QucsStudio "Model Programing Interface". Verilog-A files attached to the compiled model symbol are compiled by the ADMS 2.3 synthesizer/compiler, yielding a C++ file with extension cpp. If either the ADMS compilation sequence, or attaching a C++ file, are successful then model pins are added to the compiled model symbol and a dll file is generated for the new compiled model. In the case of the Verilog-A nonlinear resistor model pins 0 (*P0*) and 1 (*P1*) are added to the compiled model symbol. These correspond to the named pins listed in the Verilog-A module statement. However, if after the ADMS compile process the compile model symbol does NOT have any pins attached then the ADMS compile sequence failed and the Verilog-A code has one or more errors and must be debugged and recompiled. A similar comment applies to C++ attached files. Finally, to complete the new subcircuit compact device model, subcircuit interface pins are added to the body of the subcircuit. These pines are then given names (*Pin1* and *Pout1* in Figure 6.18) and a symbol drawn to represent the subcircuit. If at any time in the future when the new compact device model takes part in a circuit simulation, and the Verilog-A model code has been changed, it will be automatically recompiled by QucsStudio/ADMS prior to the start of a simulation. Testing the Verilog-A version of the nonlinear resistance with the test circuit shown in Figure 6.8 yields the same simulation results as the EDD waveforms.

6.5.1 Building a QucsStudio Verilog-A compact device model for an n type RF MESFET transistor

Figures 6.19, 6.20 and 6.21 list Verilog-A code for a Statz et al. model of an n type MESFET. This model has become popular since its inclusion as a standard component in SPICE 3. The Statz MESFET model can be regarded as a development of the Curtice MESFET model introduced in section 6.3.5 where the *Ids*, charge and noise equations are specific to the Statz model and the diode equations are similar to those in the Curtice model but have been extended to add reverse bias breakdown effects. It is also worth noting that the charge equations in the Statz model also include charge partitioning [18]. Part 1 of the Statz Verilog-A code gives definitions for the model parameters. Part 2 lists the internal variables, model branches, @(initial_model) block code and drain to source current equation. Finally, Part 3 gives the Verilog-A code for the main physical components of the MESFET compact model. The Statz MESFET compact model includes thermal noise generated by R_g, R_d and R_s, channel noise, gate noise and flicker noise. Each of these noise components are represented in Figure 6.21 by Verilog-A noise current contribution statements. Package capacitance and inductance are not included in the main body of the Statz Verilog-A code, mainly because these components can be easily added to the subcircuit version of the Statz MESFET model as is shown in Figure 6.22. The operation of the Statz MESFET model was checked by a series of basic DC, AC and transient tests. Figure 6.23 shows the S-parameter simulation test results obtained with the test circuit given in Figure 6.11. As expected these are very similar to the Curtice EDD S-parameter simulation data.

6.6 More RF examples

Qucs and QucsStudio are complex software packages with many built-in simulation and modeling capabilities whose properties and application are not obvious to everyone. Since their first release for general use the open source nature of the software has encouraged a stream of new features to be added by developers. Often new simulation features or new models are added at the request of users. Moreover, device modeling with macromodels, EDD models and Verilog-A compact device code add a measure of creativity to electronics circuit design which is only limited by the imagination of individual device modelers and circuit designers. In this section a selection of simulation and modeling features that demonstrate a number of the less well known aspects of both packages are presented.

```
//  QucsStudio Statz n type MESFET model; statz.va.
`include "disciplines.vams"
`include "constants.vams"
//
module Statz(Drain, Gate, Source);
inout Drain, Gate, Source;
electrical Drain, Gate, Source;
electrical n1, n2, n3, n4; // Internal nodes.
//
`define attr(txt) (*txt*)
`define CTOK 273.15
`define K1 7.02e-4
`define K2 1108.0
`define K3 400e-6
`define GMIN 1e-9
//
parameter real Area = 1 from [1 : inf]            `attr(info="area factor" );
parameter real Vto = -1.8 from [-inf : inf]       `attr(info="pinch-off voltage" unit = "V");
parameter real Beta = 3e-3 from [1e-9 : inf]      `attr(info="transconductance parameter" unit = "A/(V*V)");
parameter real Alpha = 2.25 from [1e-9 : inf]     `attr(info="saturation voltage parameter" unit="1/V" );
parameter real Lambda = 0.05 from [1e-9 :inf]     `attr(info="channel length modulation parameter" unit="1/V");
parameter real Nsc = 1 from [1e-9 : inf]          `attr(info="subthreshold conductance parameter" );
parameter real Cgd = 0.2e-12 from [0 : inf]       `attr(info="zero bias gate-drain junction capacitance" unit = "F");
parameter real Cgs = 1e-12 from [0 : inf]         `attr(info="zero bias gate-source junction capacitance" unit = "F");
parameter real Cds = 1e-12 from [0 : inf]
          `attr(info="zero bias drain-source junction capacitance" unit = "F");
parameter real Rin = 1e-3 from [1e-20 : inf]      `attr(info="channel resistance" unit = "Ohm" );
parameter real Rg = 5.1 from [1e-9 : inf]         `attr(info="gate resistance" unit = "Ohms");
parameter real Rd = 1.3 from [1e-9 : inf]         `attr(info="drain resistance" unit = "Ohms");
parameter real Rs = 1.3 from [1e-9 : inf]         `attr(info="source resistance" unit = "Ohms");
parameter real Vtotc = 0 from [-inf : inf]        `attr(info="Vto temperature coefficient");
parameter real Betatc = 0 from [-inf : inf]       `attr(info="Beta temperature coefficient" unit = "%/Celsius");
parameter real Alphatc = 0 from [-inf : inf]      `attr(info="Alpha temperature coefficient" unit = "%/Celsius");
parameter real Rgtc = 0 from [-inf : inf]
          `attr(info="gate resistance temperature coefficient" unit = "1/Celsius");
parameter real Rdtc = 0 from [-inf : inf]
          `attr(info="drain resistance temperature coefficient" unit = "1/Celsius");
parameter real Rstc = 0 from [-inf : inf]
          `attr(info="source resistance temperature coefficient" unit = "1/Celsius");
parameter real Fc = 0.5 from [1e-6 : inf]         `attr(info="forward-bias depletion capacitance coefficient");
parameter real Vbi = 1.0 from [1e-9 : inf]        `attr(info="built-in gate potential" unit = "V");
parameter real Eg = 1.11 from [1e-6 : inf]        `attr(info="energy gap" unit = "eV");
parameter real M = 0.5 from [1e-9 : inf]          `attr(info="grading coefficient");
parameter real Tau = 1e-9 from [1e-20 : inf]      `attr(info="transit time under gate" unit = "s");
parameter real Is = 1e-14 from [1e-20 : inf]      `attr(info="diode saturation current" unit = "I");
parameter real N = 1 from [1e-9 : inf]            `attr(info="diode emission coefficient");
parameter real Xti = 3.0 from [1e-9 : inf]        `attr(info="diode saturation current temperature coefficient");
parameter real Af = 1 from [0 : inf]              `attr(info="flicker noise exponent");
parameter real Kf = 0 from [0 : inf]              `attr(info="flicker noise coefficient");
parameter real Gdsnoi = 1 from [0 : inf]          `attr(info="shot noise coefficient");
parameter real B = 0.3 from [1e-9 :inf]           `attr(info="doping profile parameter" unit="1/V");
parameter real Bv = 1e9 from [-inf : inf]
          `attr(info="drain-gate junction reverse bias breakdown voltage" unit = "V" );
parameter real R1 = 1e9 from [1e-9 : inf]         `attr(info="breakdown slope resistance" unit = "Ohms");
parameter real Vdelta1 = 0.3 from [1e-9 :inf]     `attr(info="capacitance saturation transition voltage" unit="V");
parameter real Vdelta2 = 0.2 from [1e-9 :inf]     `attr(info="capacitance threshold transition voltage" unit="V");
parameter real Temp = 26.85 from [-273 : inf]     `attr(info="circuit temperature" unit = "Celsius");
parameter real Tnom = 26.85 from [-273 : inf]
          `attr(info="parameter measurement temperature" unit = "Celsius");
```

Fig. 6.19 Statz et al. MESFET Verilog-A model : Part 1; Parameters.

6.6.1 Harmonic Balance analysis

Both Qucs and QucsStudio include Harmonic Balance simulation. The QucsStudio implementation is more robust than the original Qucs version which still has the

```
 // Internal variables
real T1, T2, Vt_T2, Vto_T2, Rg_T2, Rd_T2, Rs_T2, Vf, Ah, Beta_T2, Ids;
real Tr, con1, Eg_T1, Eg_T2, Qgs1, Qgs2, Qgs, Qgd1, Qgd2, Qgd, Qds;
real Cgs_T2,Cgd_T2, Vbi_T2, F1, F2, F3;
real Igs1, Igs2, Is_T2,  Igd, Ids1;
real con2, con3, GMIN, Veff1, Veff2, Vnew, Vmax;
real fourkt, gm, An, thermal_pwr, flicker_pwr,  Alpha_T2, H1;
// Model branches
branch (n2, n4) b1;      branch (n2, n3) b2;      branch (n3, n4) b3;
branch (n2, n1) b4;      branch (n1, n4) b5;      branch (n2, n3) b6;
branch (Gate, n2) b7;    branch (Drain, n3) b8;  branch (n4, Source) b9;
//
analog begin
@(initial_model) begin
  Vmax=min(Fc*Vbi, Vmax);    T1=Tnom+`CTOK;    T2=$temperature;    Tr=T2/T1;
  con1=pow(Tr, 1.5);     Vt_T2=$vt;   Eg_T1=Eg-`K1*T1*T1/(`K2+T1);  Eg_T2=Eg-`K1*T2*T2/(`K2+T2);
  Vto_T2=Vto+Vtotc*(T2-T1);  Rg_T2=Rg*(1+Rgtc*(T2-T1));   Rd_T2=Rd*(1+Rdtc*(T2-T1));
  Rs_T2=Rs*(1+Rstc*(T2-T1));  Beta_T2=Area*Beta*pow(1.01, Betatc*(T2-T1));
  Ah=1/(2*Vt_T2*Nsc);   Vf=ln( 1+exp(Ah*(V(b1)-Vto_T2)))/Ah;
  Vbi_T2=(Tr*Vbi)-(2*Vt_T2*ln(con1)) - ( Tr*Eg_T1-Eg_T2);
  Cgs_T2=Area*Cgs*(1+M*(`K3*(T2-T1)-(Vbi_T2-Vbi)/Vbi));
  Cgd_T2=Area*Cgd*(1+M*(`K3*(T2-T1)-(Vbi_T2-Vbi)/Vbi));
  F1=(Vbi/(1-M))*(1-pow((1-Fc), (1-M)));  F2=pow((1-Fc), (1+M));
  F3=1-Fc*(1+M);  Is_T2=Area*Is*pow( Tr, (Xti/N))*limexp(-(`P_Q*Eg_T1)*(1-Tr)/(`P_K*T2));
  con2 = -5.0*N*Vt_T2;     con3 = 1.0/(N*Vt_T2);    fourkt=4.0*`P_K*T2;
  Alpha_T2=Alpha*( pow( 1.01, Alphatc*(T2-T1)));
end
// Drain to source current
if ( (V(b1)-Vto_T2) > 0 )
 begin
  if ( (0<V(b3))&&(V(b3)<(3/Alpha)) )  Ids=(Beta_T2*(1+Lambda*V(b3))*pow( (V(b1)-Vto_T2), 2) )*
                                          (1-pow( (1-Alpha*V(b3)/3), 3))/(1+Beta_T2*(V(b1)-Vto_T2));
  if ( V(b3)>=3/Alpha ) Ids=(Beta_T2*(1+Lambda*V(b3))*pow( (V(b1)-Vto_T2), 2))/(1+Beta_T2*(V(b1)-Vto_T2));
  if ( V(b3)<0 ) Ids=0;
 end
else Ids=0.0;
```

Fig. 6.20 Statz et al. MESFET Verilog-A model : Part 2; internal variables, model branches, @(initial_model) block code and drain to source current equation.

status of experimental. Qucs and QucsStudio EDD and Verilog-A compact device models work with most of the implemented types of simulation [22] including Harmonic Balance, making the software ideal for RF circuit simulation and design. Figure 6.24 illustrates a number of EDD equation manipulation modelling techniques, demonstrating how they function using Harmonic Balance simulation.

6.6.2 Qucs and QucsStudio filter synthesis

Filter synthesis forms part of both Qucs and QucsStudio built-in design capabilities. [23] The low-pass stepped impedance filter illustrated in Figure 6.25 shows a

[22] One exception is QucsStudio system simulation.

[23] The Qucs design routines were originally intended for RF discrete circuit design but has evolved in QucsStudio to include different RF manufacturing technologies.

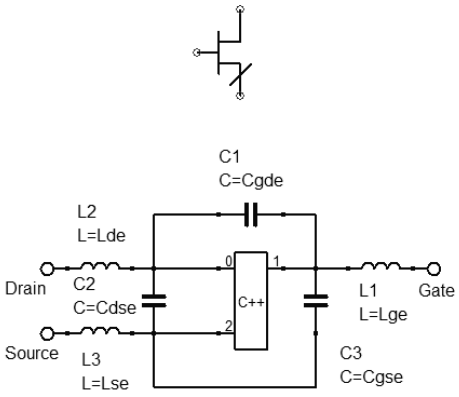
```
// Charge equations
  Veff1=0.5*( V(b4)+V(b6)+sqrt( pow( V(b4)-V(b6), 2)+Vdelta1*Vdelta1 ) );
  Vnew =0.5*( Veff1+Vto_T2+sqrt( pow( Veff1-Vto_T2, 2 )+Vdelta2*Vdelta2 ) );
   if (Vnew>Vmax)   Qgs=Cgs_T2*(2*Vbi_T2*(1-sqrt( 1-Vmax/Vbi_T2 ) )+(Vnew-Vmax)/sqrt(1-Vmax/Vbi) );
   else             Qgs=Cgs_T2*2*Vbi_T2*( 1-sqrt( 1-Vnew/Vbi));
    Veff2=0.5*( V(b4)+V(b6)-sqrt( pow( V(b4)-V(b6), 2)+Vdelta1*Vdelta1 ) );
    Qgd=Cgd_T2*Veff2;
    Qds = Area*Cds*V(b3)+Tau*Ids;
// Diode DC equations
  Igs1 = (V(b1) > con2) ? Is_T2*( limexp(V(b1)*con3) -1.0) + V(b1)*`GMIN : 0.0;
  Igs2 = (V(b1) < -Bv)  ? (V(b1)+Bv)/R1 : 0.0;
  Igd  = (V(b2) > con2) ? Is_T2*( limexp(V(b2)*con3) -1.0) + V(b2)*`GMIN : 0.0;
// Current contributions
 I(b1) <+ Igs1+Igs2;    I(b2) <+ Igd;   I(b3) <+ Ids;  I(b3) <+ ddt(Qds);  I(b4) <+ ddt(Qgs);
 I(b5) <+ Area*V(b5)/Rin;  I(b6) <+ ddt(Qgd);  I(b7) <+ V(b7)/Rg_T2;  I(b8) <+ Area*V(b8)/Rd_T2;
 I(b9) <+ Area*V(b9)/Rs_T2;
// Model noise equations
if ( V(b3) < 3/Alpha )
 begin
  H1=(1-(1-(Alpha*V(b3))/3))/(1+B*(V(b1)-Vto_T2));
  gm=2*Beta_T2*(V(b1)-Vto_T2)*(1+Lambda*V(b3))*H1+(Beta_T2*(1+Lambda*V(b3))*pow((V(b1)-Vto_T2),2))*
       B*H1/(1+B*(V(b1)-Vto_T2));
   An=1-V(b3)/(V(b1)-Vto_T2);
  thermal_pwr= (8*`P_K*T2*gm/3)*((1+An+An*An)/(1+An))*Gdsnoi;
 end
else
 begin
  gm=2*Beta_T2*(V(b1)-Vto_T2)*(1+Lambda*V(b3))/(1+B*(V(b1)-Vto_T2))+(Beta_T2*(1+Lambda*V(b3))*
       pow((V(b1)-Vto_T2),2))*B/pow( (1+B*(V(b1)-Vto_T2)),2);
  thermal_pwr=(8*`P_K*T2*gm/3)*Gdsnoi;
 end
 I(b3) <+ white_noise(thermal_pwr, "thermal");
 flicker_pwr = Kf*pow(Ids,Af);
 I(b3) <+ flicker_noise(flicker_pwr,1.0, "flicker");
 I(b8) <+ white_noise(Area*fourkt/Rd_T2, "thermal");
 I(b7) <+ white_noise(Area*fourkt/Rg_T2, "thermal");
 I(b9) <+ white_noise(Area*fourkt/Rs_T2, "thermal");
end
endmodule
```

Fig. 6.21 Statz et al. MESFET Verilog-A model : Part 3; charge equations, diode DC equations, current contributions and model noise equations.

QucsStudio designed circuit and its implementation in microstrip technology plus a typical set of simulated S-parameter data.

6.6.3 QucsStudio System simulation

System simulation is not implemented in Qucs but is restricted to QucsStudio. This is the latest type of simulation to be added to the software. Figure 6.26 shows a model for demonstrating the difference between eye diagrams with and without added noise noise in a signal channel.



Verilog-A Statz compact semiconductor device model for an n type MESFET
Properties:
1, Statz model - including subthreshold improvement
2. Gate - source current model - exponential diode model plus linear reverse breakdown model.
3. Gate - drain current model - exponential diode model
4. Internal dynamic capacitance properties modelled by semiconductor diode charge with charge partitioning .
5. Series terminal resistors Rd, Rg and Rs
6. Thermal noise generated by Rd, Rg and Rs
7. Channel noise
8. Gate noise
9. Flicker noise
10. External bias independant capacitors Cgse, Cgde and Cdse.
11. External lead inductance: Lge, Lde and Lse.

X1	MESFET1
File=Statz.va	Area=1
Area=Area	Vto=-1.8
Vto=Vto	Beta=3e-3
Beta=Beta	Alpha=2.25
Alpha=Alpha	Lambda=0.05
Lambda=Lambda	Nsc=1
Nsc=Nsc	Cgd=0.2e-12
Cgd=Cgd	Cgs=1e-12
Cgs=Cgs	Cds=1e-12
Cds=Cds	Rin=1e-3
Rin=Rin	Rg=5.1
Rg=Rg	Rd=1.3
Rd=Rd	Rs=1.3
Rs=Rs	Vtotc=0
Vtotc=Vtotc	Betatc=0
Betatc=Betatc	Alphatc=0
Alphatc=Alphatc	Rgtc=0
Rgtc=Rgtc	Rdtc=0
Rdtc=Rdtc	Rstc=0
Rstc=Rstc	Fc=0.5
Fc=Fc	Vbi=1.0
Vbi=Vbi	Eg=1.11
Eg=Eg	M=0.5
M=M	Tau=1e-9
Tau=Tau	Is=1e-14
Is=Is	N=1
N=N	Xti=3.0
Xti=Xti	Af=1
Af=Af	Kf=0
Kf=Kf	Gdsnoi=1
Gdsnoi=Gdsnoi	B=0.3
B=B	Bv=1e9
Bv=Bv	R1=1e9
R1=R1	Vdelta1=0.3
Vdelta1=Vdelta1	Vdelta2=0.2
Vdelta2=Vdelta2	Tnom=26.85
Temp=Temp	Cgse=300e-15
Tnom=Tnom	Cgde=30e-15
	Cdse=300e-15
	Lge=0.0
	Lde=0.0
	Lse=0.0
	Temp=26.85

Fig. 6.22 Statz Verilog-A/subcircuit model for a n type MESFET: subcircuit internal circuit including package capacitance and inductance parasitics plus subcircuit symbol and parameter lists.

6.7 From Qucs to QucsStudio: future directions in analog simulation and device modeling

Qucs and QucsStudio are current generation circuit simulators with common origins and development via an international group of developers. Over the last decade the development team has invested much time and effort to produce the current versions of the software. QucsStudio being the younger package is currently going through the greatest number of changes. This should not detract from Qucs which is still an

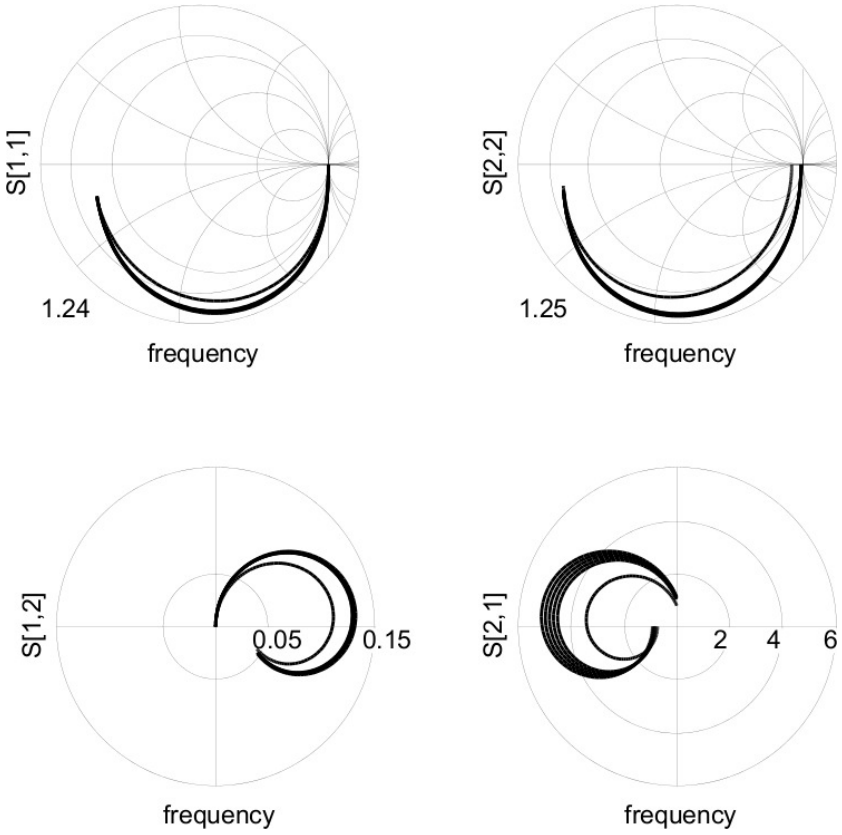


Fig. 6.23 Statz Verilog-A/subcircuit model S-parameter simulation data for a transistor with identical DC bias conditions to the device shown in Figure 6.12.

on going project with many thousands of regular users. So were does the future lie for both packages? This question can probably only be answered by the current and future users of the software. Today Qucs is a well established stable program with a dedicated group of users. Conversely, QucsStudio has many new features to offer but due to its current development status it is not as stable as Qucs. QucsStudio, at this time, supports the following features: more than 100 different circuit component types, linear and nonlinear DC analysis, small signal AC analysis (including noise and noise distribution analysis), S-Parameter analysis (including noise parameter calculations), transient analysis, harmonic balance analysis (including noise analysis), system simulation, parameter sweeps and optimization of analog circuits [22], Verilog digital simulation using ICARUS by S. Williams [23], VHDL digital simulation by T. Gingold [24], printed circuit board layout and Gerber viewer using KCAD by the KCAD team [25], numerical data processing with Octave by J. W. Eaton et al. [26], RF transmission line calculation (coaxial, microstrip, coplanar etc.),filter synthesis (LC, ladder, stepped-impedance, microstrip, active filter etc.), attenua-

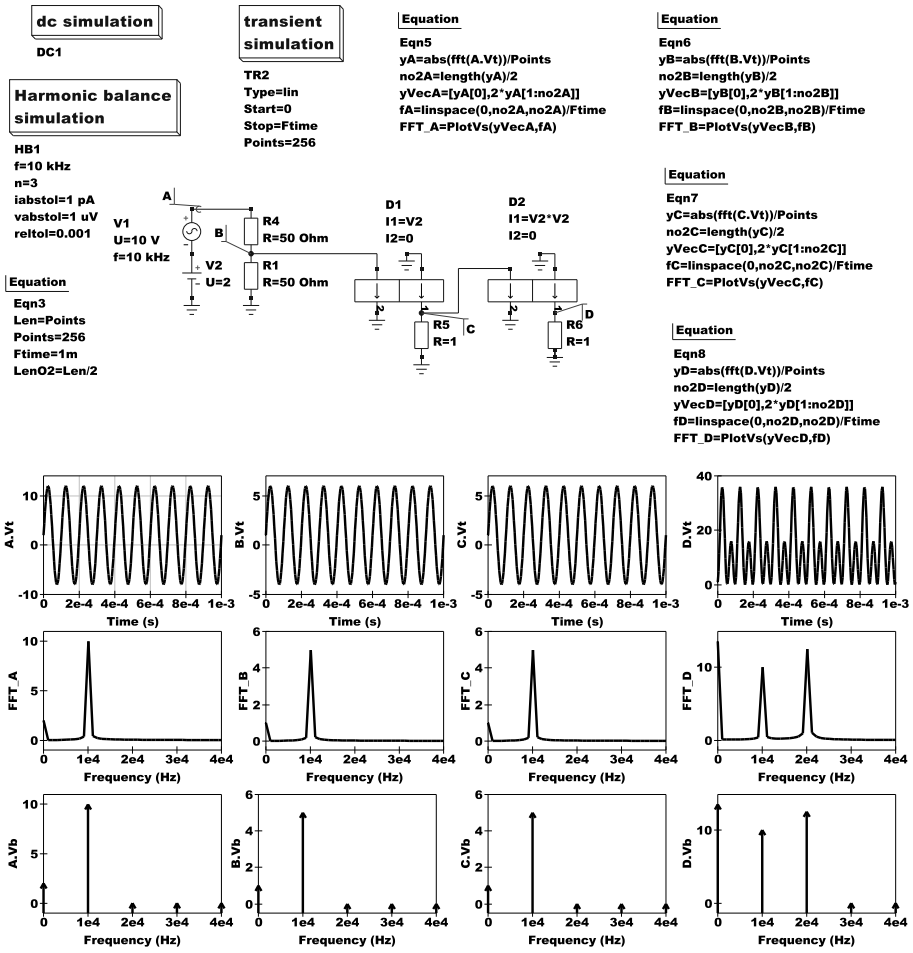


Fig. 6.24 Harmonic Balance simulation of series EDD models set up to demonstrate voltage probes and current to voltage conversion functions.

tor synthesis (resistive pi, tee and bridge topology), GPIB device control, ADMS Verilog-A interface for compact device modeling using compiled models and subcircuits, technical documentation for all models and types of simulation, and support for the German and English languages. The following extensions to the QucsStudio software are planned for future development: oscillations by Harmonic Balance, periodic steady-state analysis, mixed-mode analysis, EM field simulation, an extended range of transistor models including HICUM, BSIM, EKV, TOM etc.,an improved Verilog-AMS interface, bug fixes, small improvements and extensions. Which one of the two packages you choose for your simulation and modeling projects is simply a matter of personal choice so why not try both?

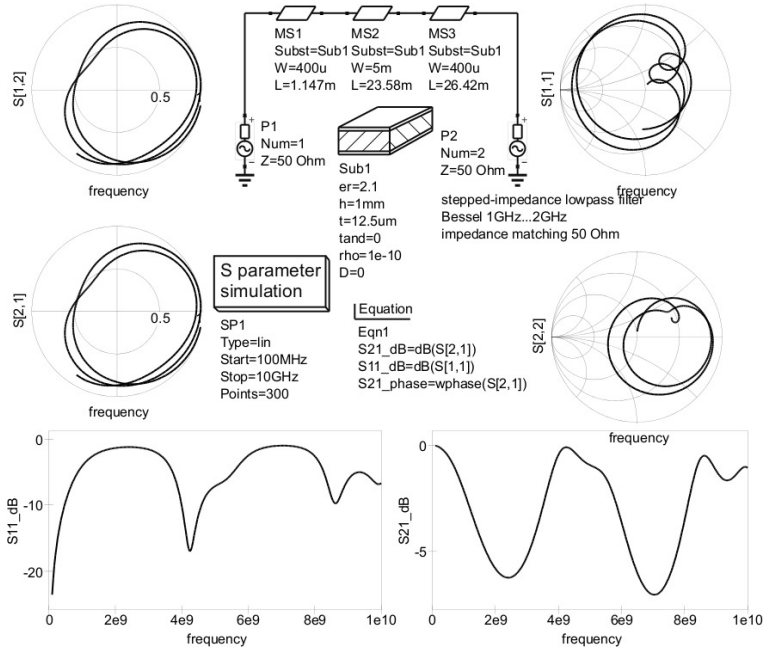


Fig. 6.25 QucsFilter designed microstrip low-pass filter test bench and typical S-parameter simulation data.

6.8 Summary

Qucs and the more recent QucsStudio are examples of engineering software that has become freely available through the open source software movement. In ten years or less the quality of open source circuit simulation software has improved to such an extent that todays software provides users with tools capable of tackling highly complex circuit design tasks. Packages like Qucs offer extensive simulation capabilities linked with high quality graphical interfaces plus stable simulation engines and extensive post-simulation data analysis tools. Probably, one of the most significant recent changes to take place has been the merging of high class compact device modeling tools with mature circuit simulators. Through the efforts of the "MOS Modeling and Parameter Extraction Working Group" to standardise the use of the Verilog-A hardware description language for modeling of compact semiconductor devices the open source software community has been able to add to our understanding of modeling techniques and their relationship to circuit simulation. This chapter has attempted to outline a number of powerful approaches to device modeling and to demonstrate their use in analog and RF simulation. By their very nature modeling and simulation are dynamic entities which will change and develop in the future as open source circuit simulation technology increases in sophistication.

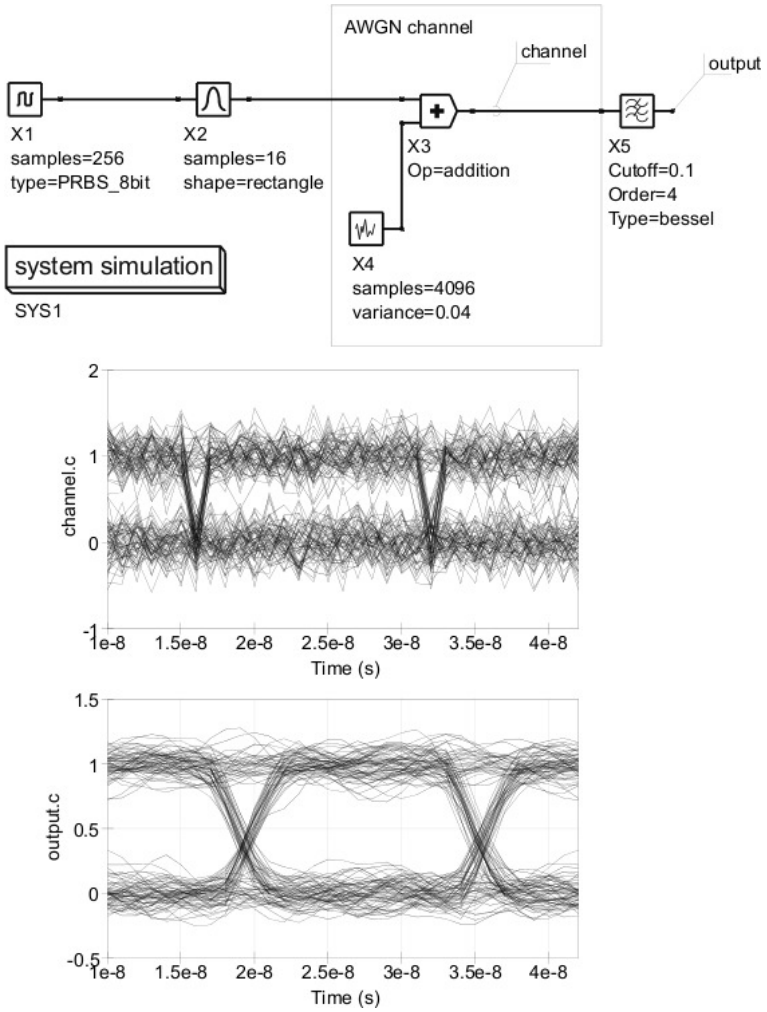


Fig. 6.26 A QucsStudio system simulation example which illustrates the effect of noise on channel performance.

References

1. Bowers J. C.; Sedore S. R. "SCEPTRE: A computer program for circuit and systems analysis", **1971**. Prentice-Hall, Inc,, Englewood Cliffs, N.J.
2. Johnson, B.; Quarles, T.; Newton, A.R.; Pederson, D.O.; Sangiovanni-Vincentelli, A. "SPICE3 Version 3f User's Manual", Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, **1992**.
3. "S-Parameter Simulation", **2005**. Agilent Technologies, Inc., Palo Alto, CA. Available from http://cp.literature.agilent.com/litweb/pdf/ads2005a/pdf/cktsimsp.pdf.
4. "Harmonic Balance Simulation", **2006**. Agilent Technologies, Inc., Palo Alto, CA. Available from http://cp.literature.agilent.com/litweb/pdf/ads2006/pdf/cktsimhb.pdf.

5. Perticaroli S.; Palma F.; "A novel PSS analysis implementation reusing the TRAN analysis of Ngspice circuit simulator", **2011**. MOS-AK/GSA Workshop, Universit Pierre et Marie Curie (UPMC), Paris. Available from http://www.mos-ak.org/paris/.
6. Christen E.; Bakalar K."VHDL-AMS-a hardware description language for analog and mixed-signal applications",Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on] Volume 46, Issue 10, Oct. **1999**, pp. 1263 - 1272.
7. "Verilog-AMS Language Reference Manual. Version 2.2". **2004**. Available from http://www.accellera.org.
8. Margraf, M.; Jahn, S.; Flucke, J.; Jacob, R.; habchi, V.; Ishikawa, T.; Gopala Krishna, A.; Brinson, M.; Parruitte, H,; Roucaries, B.; Kraut, G. "Qucs (Quite Universal Circuit Simulator)", Version 0.0.16, **2011**. Available from: http://qucs.sourceforge.net/index.html.
9. Lamaitre, L.; McAndrew, C.; Hamm, S. "ADMS - automated device model synthesizer", Proc CICC, **May 2002**, 27-30.
10. Margraf, M., "QucsStudio: A development environment for electrical engineers." **2011**. Available from homepage at http://www.mydarc.de/DD6UM/QucsStudio/qucsstudio.html.
11. Boyle G.R.; Cohn B. M.; Pederson D. O. and Soloman J. E. "Macromodeling of integrated circuit operational amplifiers". IEEE Journal of Solid-State Circuits, **December 1974**.
12. Chua L. C. "Introduction to nonlinear network theory". **1969**. McGraw-Hill, Inc.
13. Newton, A.R.; Pederson, D.O.; Sangiovanni-Vincentelli, A. "SPICE Version 2g User's Guide", Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, **1981**.
14. Lahn S.; Brinson M. E. "Interactive compact device modelling using Qucs equation-defined devices". International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, **2008**, 21:335-349.
15. Brinson M. E: Jahn S.;, Nabijou H. "Compact device modeling for established and emerging technologies with the Qucs GPL circuit simulator". MIXDES 2009, 16th International Conference Mixed design of integrated circuits and systems, June 25-27, **2009** Lodz, Poland, pp. 39-44.
16. Curtice W. R.; "A MESFET model for use in the design of GaAs integrated circuits", IEEE Transactions on Microwave Theory and Techniques, **1980**, MTT-28, pp. 448-456.
17. Statz H.; Newman P.; Smith I.W.; PucelR.A.; Haus H.A. "GaAs FET Device and Circuit Simulation in SPICE, **1987**. IEEE Transactions on Electron Devices, Vol. 34, pp. 160-169,
18. Divehar D.; "Comments on GaAs FET device and circuit simulation in SPICE" —textbf1987 IEEE Transactions on Electronic Devices, Vol. ED-34, pp 2564-2565.
19. Brinson M. E: Jahn S.;, Nabijou H. "Qucs, SPICE and Modelica equation-defined modelling techniques for the construction of compact device models based on a common model template structure". **2011**, MOS-AK/GSA Workshop, Universit Pierre et Marie Curie (UPMC), Paris. Available from http://www.mos-ak.org/paris/.
20. Fitzpatrick D.; Miller I. " Analog behavioral modeling with the Verilog-A language". **1998**, Kluwer Academic Publishers, London.
21. Kundert K.S.; Zinke O. "The Designer's Guide to Verilog AMS". **2004**. Kluwer Academic Publishers, London.
22. "ASCO (A SPICE optimizer). Available from http://asco.sourceforge.net/.
23. Williams S., "Icarus Verilog: Open source Verilog compiler", Version 0.9, **2011**. Available from:http://iverilog.icarus.com/.
24. Gingold T. "ghdl VHDL simulator. Available from http://gna.org/projects/ghdl/.
25. KiCAD team. "KiCAD EDA suite". Available from http://kicad.sourceforge.net/wiki/Main_Page.
26. Eaton, J.W., "GNU Octave", **1992**. Available from http://www.gnu.org/software/octave/.
27. Allen P. E.; Sanchez-Sinencio E' "Switched capacitor circuits". **1984**. Van Nostrand Rainhold, New York.