Verilog-A Compact Semiconductor Device Modelling and Circuit Macromodelling with the QucsStudio-ADMS "Turn-Key" Modelling System

Mike E. Brinson, and Michael Margraf

***Abstract*—The Verilog-A "Analogue Device Model Synthesizer" (ADMS) has in recent years become an established modelling tool for GNU General Public License circuit simulator development. Qucs and ngspice are two examples of open source circuit simulators that employ ADMS for compact semiconductor model construction. This paper presents a "turn-key" compact device modelling and circuit macromodelling system based on ADMS and implemented in the QucsStudio circuit design, simulation and manufacturing environment. A core feature of the new system is a modelling procedure which does not require users to manually patch, by hand, circuit simulator C++ code. At the start of QucsStudio simulation the software automatically detects any changes in Verilog-A model code, re-compiling and dynamically linking the modified code to the body of the QucsStudio code. The inherent flexibility of the "turn-key" system encourages rapid experimentation with analogue and RF compact device models and circuit macromodels. In this paper QucsStudio "turn-key" modelling is illustrated by the design of a single stage RF amplifier circuit and the Harmonic Balance large signal AC simulation of a 50 Ω RF diode switch.**

***Index Terms*— QucsStudio, ADMS, Verilog-A, compact device modelling, turn-key component modelling**

I. Introduction

UNTIL the adoption by the Compact Model Council [1] of Verilog-A as the preferred analogue hardware description language for compact semiconductor device modelling C had been the standard modelling language. However, hand coding of compact device models in C was often found to be very tedious, time consuming and subject to error, particularly when determining the partial derivatives of the device currents and charges needed in DC and transient simulation of non-linear circuits. In contrast to C, the Verilog-AMS hardware description language provides built-in tools which automatically generate partial derivatives, making compact device modelling a much more straight forward process. Current trends suggest that there is growing acceptance, by the compact semiconductor device modelling community, of the Verilog-AMS subset Verilog-A as the preferred compact modelling language. The standardization of Verilog-AMS [2] and specifically the addition of a number of compact modelling enhancements to its analog Verilog-A subset [3] have also greatly influenced Verilog-A usage. The release of the Verilog-A "Analogue Device Model Synthesizer" (ADMS) software [4] under the GNU General Public License has also accelerated the rate at which Verilog-A has been accepted and used by the modelling community as a viable replacement for C. Moreover, the growing number of commercial [5] [6] and open source circuit simulators [7] [8] which use Verilog-A for compact semiconductor device and circuit macromodelling is testimony to the importance of Verilog-A in the development of circuit simulator technology. This paper outlines the structure and operation of a new "turn-key" Verilog-A compact model development system which automatically re-compiles and dynamically links modified Verilog-A model code to the C++ body of a circuit simulator prior to the start of a simulation sequence. The new compact modelling system has been implemented in a freely available circuit design, simulation and manufacturing environment called QucsStudio [9]. QucsStudio is released under the GNU General Public License for use with the Microsoft Windows® operating system and includes a second generation version of the popular Qucs circuit simulator plus additional circuit design, simulation and manufacturing features.

Mike Brinson is with the Centre for Communications Technology, London Metropolitan University, London N78DB, UK; (e-mail: mbrin72043@yahoo.co.uk).

Michael Margraf is the Qucs and QucsStudio Project Founder, Berlin, Germany; (michael.margraf@alumni.tu-berlin.de).

II. Verilog-A Compact Device Modelling with Qucs-ADMS

Verilog-A based compact device modelling [10] was first implemented in Qucs version 0.0.11. In the original Qucs modelling technique the ADMS Verilog-A to C++ synthesizer was used to manually compile Verilog-A model code to C++ code. After conversion the C++ code also had to be manually merged with the main body of the Qucs circuit simulator [11]. Similarly, the Qucs graphical user interface code needed to be patched to add a new model symbol to the simulators library of built-in component symbols. Finally, due to the fact that the Qucs simulator uses C++ static model libraries, the entire simulator C++ code had also to be re-compiled and re-linked to generate a new extended circuit simulator each time a compact device model was added to the software. In principle, it was possible to add compact device models to Qucs using the previously described procedure. In practice, the modelling process required users to have an advanced knowledge of C++ programming techniques coupled with a good understanding of the Qucs model application interface, making the process of adding new compact models one which was more suited to Qucs developers rather than the wider Qucs user community. The original Qucs compact device modelling process was further complicated in that it was designed primarily to function with software development tools supplied with the Linux operating system rather than more universally available Microsoft Windows® operating system.

III. Qucs Studio-ADMS "Turn-Key" Verilog-A Compact Device Modelling

One of the primary aims of the QucsStudio Verilog-A compact device modelling system is to provide circuit design engineers with a simple modelling tool that does not require the main body of the circuit simulator C++ code to be patched by hand when adding new device models to the QucsStudio circuit simulator. In contrast to the original Qucs modelling scheme the QucStudio version is based on dynamic linked model libraries rather than static model libraries. This change has had a major effect on software utility, allowing a wider user base access to the supplied modelling tools. The implemented modelling system has been called a "turn-key" system by its developers to emphasize that it takes over responsibility for determining when a compact device model needs to be re-compiled and re-linked. Changes in Verilog-A model code act like a key turning on the compilation and linking of modified code. When changes take place, edited models are automatically updated at the start of the next user requested circuit simulation. Fig. 1 presents a simple flow chart of this process, outlining each of the QucsStudio modelling stages. In this diagram the modelling sequence is shown starting from Verilog-A code entry using a built-in text editor, at the top of the diagram, followed by attachment of a model Verilog-A code file (XXXX.va in Fig. 1) to a QucsStudio "C++ compiled model icon", through synthesis of the C++ model code, using the ADMS software, to C++ compilation and dll code linking with the MinGW C++ tools, and finally construction of a model subcircuit, at the bottom of the diagram. At the start of the modelling process equations representing the different physical aspects of device operation are entered into the QucsStudio software as Verilog-A "module" code. A convenient colour highlighted text editor is provided with QucsStudio for this task. Phase two involves the synthesis of the C++ code from the entered Verilog-A code. With the Verilog-A model code visible on the QucsStudio text editor display window, pressing key F2 causes the generation of the compact model C++ code to take place, followed by C++ compilation using the MinGW tools to form a dynamic linked model library entry (XXXX.dll in Fig. 1). The last two stages only take place if the original Verilog-A module code is error free. On successful generation of a "C++ compiled model", it becomes a stand-alone component which is dynamically linked to the main body of the QucsStudio code. C++ compiled models can also be combined with other QucsStudio components by attaching them to a QucsStudio schematic diagram, to form a subcircuit or macromodel. During the simulation of circuits which include C++ compiled models any changes in their Verilog-A code will automatically trigger the "turn-key" modelling process ensuring that the Verilog-A compact device models are kept up to date at all times.

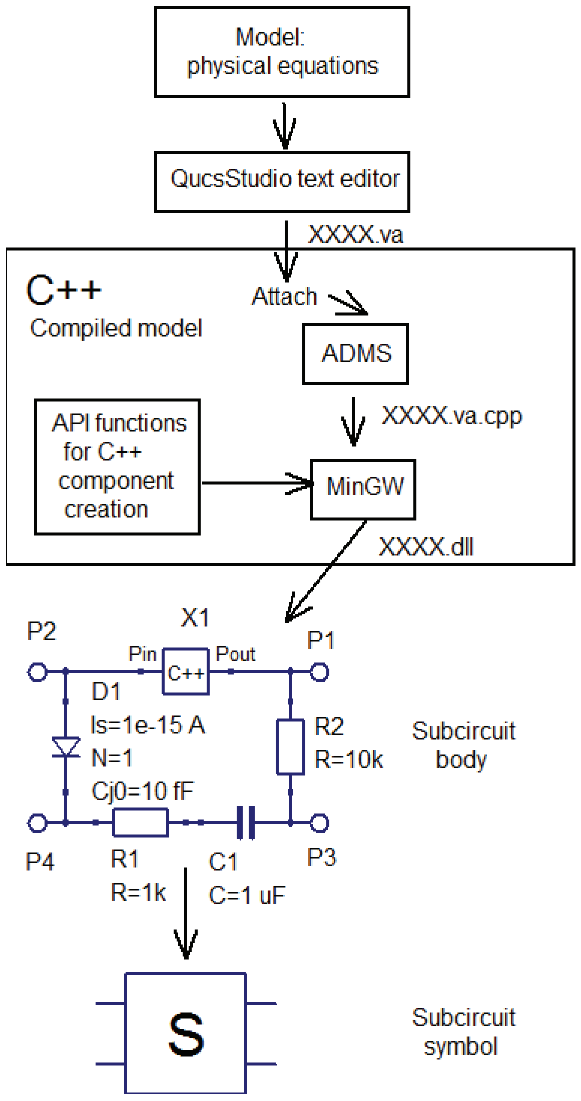
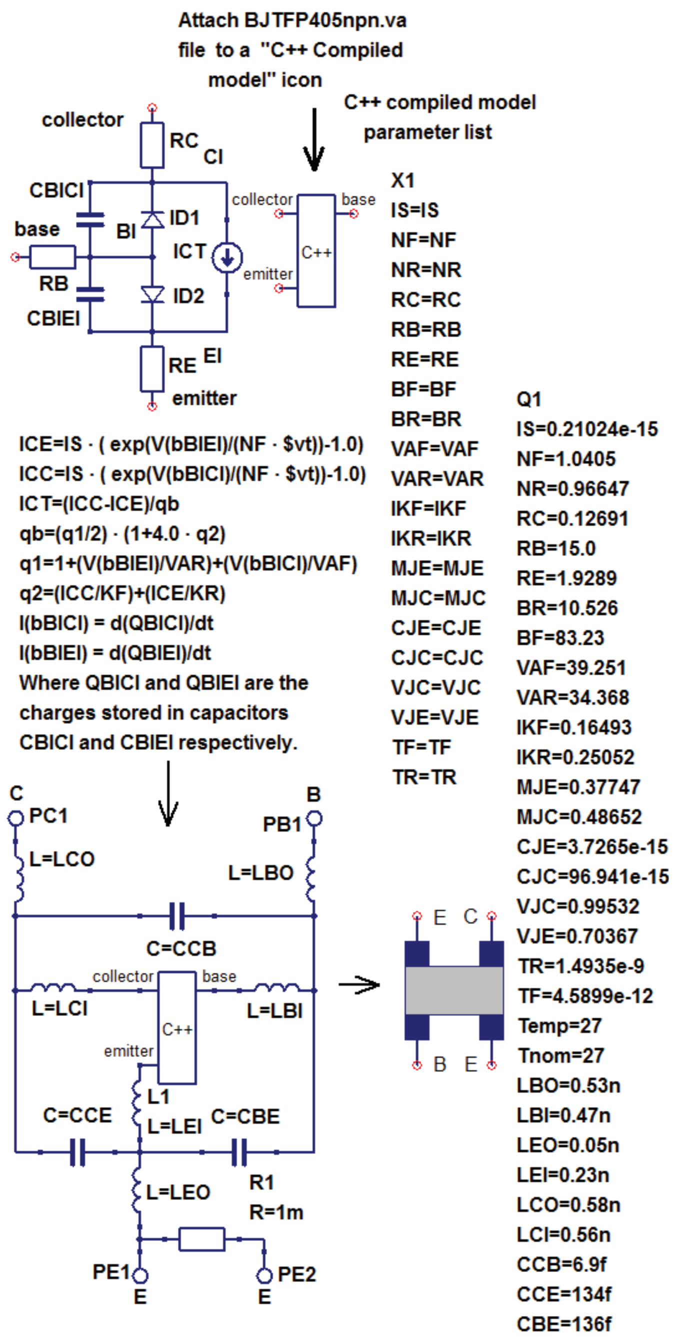


Fig. 1. A flow chart outlining the QucsStudio "Turn-key" Verilog-A compact model development system.

To demonstrate the QucsStudio Verilog-A "turn-key" modelling procedure the construction of a simple RF npn BJT compact semiconductor device model is presented next. The model information given in Fig. 2 is based on a large signal Ebers-Moll bipolar transistor equivalent circuit, a simplified set of non-linear device equations, including second order high-level current injection effects and internal capacitance, plus a subcircuit schematic showing external lead inductance and capacitance.

The Verilog-A code for the RF npn BJT model is listed in Fig. 3. On attaching this code to the QucsStudio C++ compiled model icon the QucsStudio software tries to extract the device input/output node names and the device parameter names and default values. If successful, QucsStudio draws a group of named terminals attached to the original C++ compiled model icon. These correspond in name and list order to the "inout" terminals given in the Verilog-A module statement. The software also attaches a list of device parameter properties to the C++ compiled model icon associated with the new model.



ICE=IS · (exp(V(bBIEI)/(NF · $vt))-1.0)
ICC=IS · (exp(V(bBICI)/(NF · $vt))-1.0)
ICT=(ICC-ICE)/qb
qb=(q1/2) · (1+4.0 · q2)
q1=1+(V(bBIEI)/VAR)+(V(bBICI)/VAF)
q2=(ICC/KF)+(ICE/KR)
I(bBICI) = d(QBICI)/dt
I(bBIEI) = d(QBIEI)/dt
Where QBICI and QBIEI are the charges stored in capacitors CBICI and CBIEI respectively.

```
X1
IS=IS
NF=NF
NR=NR
RC=RC
RB=RB
RE=RE
BF=BF
BR=BR
VAF=VAF
VAR=VAR
IKF=IKF
IKR=IKR
MJE=MJE
MJC=MJC
CJE=CJE
CJC=CJC
VJC=VJC
VJE=VJE
TF=TF
TR=TR
```

```
Q1
IS=0.21024e-15
NF=1.0405
NR=0.96647
RC=0.12691
RB=15.0
RE=1.9289
BR=10.526
BF=83.23
VAF=39.251
VAR=34.368
IKF=0.16493
IKR=0.25052
MJE=0.37747
MJC=0.48652
CJE=3.7265e-15
CJC=96.941e-15
VJC=0.99532
VJE=0.70367
TR=1.4935e-9
TF=4.5899e-12
Temp=27
Tnom=27
LBO=0.53n
LBI=0.47n
LEO=0.05n
LEI=0.23n
LCO=0.58n
LCI=0.56n
CCB=6.9f
CCE=134f
CBE=136f
```

Fig. 2. A QucsStudio "turn-key" RF npn BJT model outline showing: Ebers-Moll equations and equivalent circuit plus second order high-level current injection effects and internal capacitance equations; C++ compiled model icon with parameters (X1); subcircuit body and symbol plus parameters (Q1).

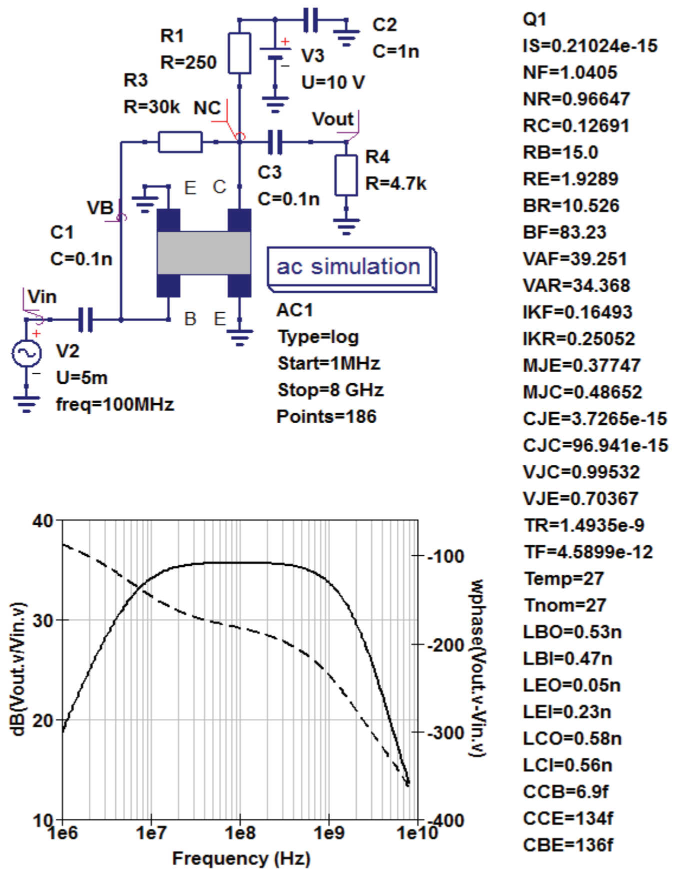
```
//  Verilog-A npn RF BJT model
`include "disciplines.vams"
`include "constants.vams"
module BJTFP405npn(collector,base,emitter);
inout collector,base,emitter ;  electrical collector,base,emitter;
electrical CI, BI, EI, nl1; // Internal nodes.
`define CTOK 273.15
`define GMIN 1e-12
`define TWOQ 2*`P_Q
parameter real IS = 0.21024e-15 from [1e-20 : inf);
parameter real NF = 1.0405 from [0.5 : inf) ;
parameter real NR = 0.96647 from [0.5 : inf) ;
parameter real RC = 0.12691 from [1e-20 : inf);
parameter real RB = 15.0 from [1e-20 : inf);
parameter real RE = 1.9289 from [1e-20 :inf;)
parameter real BF = 83.23 from [1e-20 : inf) ;
parameter real BR = 10.526 from [1e-20 : inf);
parameter real VAF = 39.251 from [1e-20 : inf);
parameter real VAR = 34.368 from [1e-20 : inf);
parameter real IKF = 0.16493 from [1e-20 : inf);
parameter real IKR = 0.25052 from [1e-20 : inf);
parameter real MJE = 0.37747 from [1e-20 : inf);
parameter real MJC = 0.48652 from [1e-20 : inf);
parameter real CJE = 3.7265e-15 from [1e-20 : inf);
parameter real CJC = 96.941e-15 from [1e-20 : inf);
parameter real VJC = 0.99532 from [1e-20 : inf);
parameter real VJE = 0.70367 from [1e-20 : inf);
parameter real TF  = 4.5898  from [1e-20 : inf);
parameter real TR  = 1.4935  from [1e-20 : inf);
parameter real Temp = 27 from [-273.15 : inf);
real con1, con2, con3, con4, con5, con6, con7;
real  x1, y1, x2, y2, z1, z2, con8, con9, QBICI, QBIEI, q1O2;
real IEC, ICC, q1, q2, T1, T2,  VJCO2, VJEO2;
// Model branches
branch (collector, CI)  bcollectorCI;   branch (base, BI)  bbaseB1;
branch (EI, emitter)  bElemitter;  branch (BI, CI)  bBICI;
branch (BI, EI) bBIEI;  branch (CI, EI) bCIEI;
analog begin
  con1 = 1.0/(NF*$vt);  con2 = 1.0/(NR*$vt);
  VJCO2 = VJC/2;  VJEO2 = VJE/2;  con3 = 1.0-MJE;
  con4 = 1.0-MJC; con5 = exp(MJE*ln(2)); con6 = exp(MJC*ln(2));
 // Current contributions
  I(bcollectorCI) <+ V(bcollectorCI)/RC;  I(bbaseBI) <+ V(bbaseBI)/RB;
   I(bElemitter)  <+ V(bElemitter)/RE;
  IEC = IS*(limexp(V(bBICI)*con2)-1.0);  ICC = IS*(limexp(V(bBIEI)*con1)-1.0);
  q1=1.0 + V(bBICI)/VAF + V(bBIEI)/VAR;   q2  = ICC/IKF + IEC/IKR;
  I(bBICI) <+ IEC/BR + `GMIN*V(bBICI);  I(bBIEI) <+ ICC/BF + `GMIN*V(bBIEI);
  q1O2 = q1/2;  I(bCIEI)  <+ (ICC-IEC)/(1e-20+q1O2*sqrt(1.0+4*q2));
  y1 = 1.0-V(bBICI)/VJ  y2 = 1.0-V(bBIEI)/VJE;
  z1 = exp(con4*ln(y1));  z2 = exp(con3*ln(y2));
  QBICI = (V(bBICI)>=VJCO2)
          ? TR*IEC+CJC*con6*(V(bBICI)*V(bBICI)+con4*V(bBICI))
          : TR*IEC+CJC*((VJC/con4)*(1.0-z1));
  QBIEI = (V(bBIEI)>=VJEO2)
          ? TF*ICC+CJE*con5*(V(bBIEI)*V(bBIEI)+con3*V(bBIEI))
           : TF*ICC+CJE*((VJE/con3)*(1.0-z2));
 I(bBICI) <+ ddt(QBICI);  I(bBIEI) <+ ddt(QBIEI);
end
endmodule
```

Fig. 3. Verilog-A code for a simplified RF npn BJT model: the model parameters have the same meaning as those defined in the SPICE 3f5 BJT model [12].

Shown in Fig. 4 is a single stage, class A, RF npn BJT amplifier circuit, with collector feedback, configured for small signal AC simulation over the frequency band 1MHz to 1GHz.

IV. QucsStudio C++ Compiled Model Programming Interface

Central to the operation of the QucsStudio Verilog-A “turn-key” modelling system is the C++ compiled model component. The structure and function of this new modelling element is given in the template listed in Fig.5. In general QucsStudio built-in component models, and user defined models, are specified by the variables, properties and functions listed in this template. The template includes, amongst other things, the number of external and internal nodes as well as pointers to a parameter list and a schematic symbol specification. It also contains function calls, like (tEvaluate)Matrix in Fig.5, that determine the physical operation of a component. Many of the catalogued items are optional. The current release of the QucsStudio software includes a number of detailed examples showing component model template entries. These are fully documented and provide a wealth of important model building data. The values for the variables listed in the component template are generated by ADMS during synthesis of the model C++ code.



```
Q1
IS=0.21024e-15
NF=1.0405
NR=0.96647
RC=0.12691
RB=15.0
RE=1.9289
BR=10.526
BF=83.23
VAF=39.251
VAR=34.368
IKF=0.16493
IKR=0.25052
MJE=0.37747
MJC=0.48652
CJE=3.7265e-15
CJC=96.941e-15
VJC=0.99532
VJE=0.70367
TR=1.4935e-9
TF=4.5899e-12
Temp=27
Tnom=27
LBO=0.53n
LBI=0.47n
LEO=0.05n
LEI=0.23n
LCO=0.58n
LCI=0.56n
CCB=6.9f
CCE=134f
CBE=136f
```

Fig. 4. A class A npn BJT RF amplifier with collector feedback: small signal AC test circuit and example gain and phase response curves; legend: solid line = dB(Vout.v/Vin.v) the amplifier gain in dB and dotted line = wphase(Vout.v/Vin.v) the unwrapped phase of the amplifier gain in degrees.

In the case of the RF npn BJT model the translated C++ model code is stored in file BJTFP405npn.va.cpp and compiled by the MinGW tools to produce a BJTFP405npn dynamically linkable library model. In order to synthesize the C++ code needed for simulation of an analogue model the QucsStudio-ADMS-MinGW tools undertake the required operations in terms of a model application programming interface (API), specifically designed for QucsStudio C++ component model creation. The currently available API model functions are defined in Fig.6.

```
// component definition
EXPORT tComponentInfo compInfo = {
 isNonLinear,        // component type ('isNonLinear' or 'isLinear')
 "BJTFP405npn",      // model identifier
 "VerilogAMS model of BJTFP405npn",  // component description
 3,                  // number of external nodes
 3,                  // number of internal nodes
 0,                  // number of inputs (system simulations only)
 20,                 // number of parameters
 params,             // pointer to list of parameters
 0,                  // size of global variable buffer in bytes
 0,                  // pointer to component icon (0 = unused)
 0,                  // size of component icon
 -1,                 // index of parameter determining schematic symbol (-1 = unused)
 0,                  // pointer to list of schematic symbols
 (tEvaluate)fillMatrix,   // function calculating analog model (0 = no model exists)
 0,                  // function calculating noise model (0 = noise free)
 0,                  // function calculating system model (0 = no model exists)
 0,                  // string with digital Verilog model (0 = no model exists)
 0                   // string with digital VHDL model (0 = no model exists)
};
```

Fig. 5. QucsStudio component definition template.

```
1. setA(Node1, Node2, num);
      sets a single linear matrix element
2. setG(Node1, Node2, conductance);
      sets a linear conductance
3. setR(Node1, Node2, resistance);
      sets a linear resistance
4. setC(Node1, Node2, capacitance, initial voltage);
      sets a linear capacitance
5. setL(Node1, Node2, internal Node, inductance, initial current);
      sets a linear inductance
6. setM(Node1, Node2, mutual inductance, initial current);
      sets a linear mutual inductance
7. setI(Node1, Node2, current);
      sets a linear current
8. setDelayedI(Node1, Node2, line constant, delay, buffer pointer);
      sets a linear time-delayed current
9. setIQ(Node1, Node2, current, charge);
      sets a non-linear current and charge
10. setGC(Node1, Node2, Node3, Node4, current derivative, charge derivative);
      sets a non-linear conductance and capacitance
11. setNoiseA(Node1, Node2, noise current density);
      sets a single linear noise matrix element
12. setNoiseG(Node1, Node2, noise current density);
      sets a linear noise current
13. setNoiseNG(Node1, Node2, noise current density);
      sets a non-linear noise current
```

Fig. 6. QucsStudio C++ model application programming interface functions: function parameters have their usual meaning as implied by their names.

V. Adding Verilog-A Natures to QucsStudio

A high percentage of RF compact device models and circuit macromodels often include a mixture of fundamental linear and non-linear R, L and C components. However, version 2.3.0 of ADMS appears to treat all R and C components as non-linear elements. Also, with this version of ADMS, it is not permitted to express the relation between inductance L, inductance current I and inductance voltage V, between nodes p and n, as

$$V(p,n) \ <+ \ L*ddt(I(p,n)) \qquad (1)$$

One way to overcome the latter restriction is to combine a linear capacitor, or a non-linear capacitor, with a gyrator [13] to form a linear or a non-linear inductance which has an identical value to the capacitor value. In compact models with a significant number of R, L and C components the number of model floating point calculations undertaken during simulation can be reduced by ensuring that the QucsStudio "turn-key" modelling system uses linear R, L and C components, whenever possible. It is also worth noting that Harmonic Balance simulation at RF frequencies has a higher probability of reaching a satisfactory convergence to a correct solution when simulating circuits with compact device models comprising a balanced mixture of linear and nonlinear components. Unfortunately, it appears that ADMS version 2.3.0 does not include a mechanism for selecting linear or non-linear versions of the fundamental electrical components R, L and C. In the QucsStudio software this selection process has been implemented by adding three new linear Verilog-A "natures" to the standard disciplines and natures file, "disciplines.vams". These linear additions, R, G and C, are defined in Table I.

TABLE I
QucsStudio Natures plus Modified Electrical Discipline for Linear R and C Components

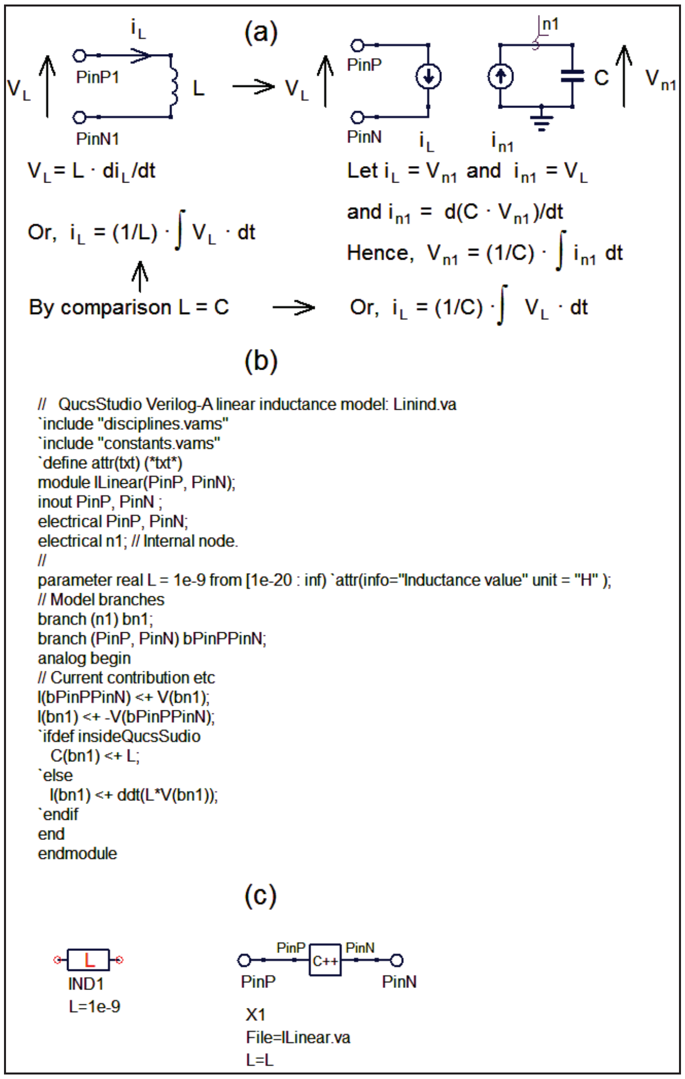
nature Resistance
units = "ohms";
access = R;
endnature
	nature Conductance
	units = "s";
	access = G;
	endnature
		nature Capacitance
		units = "F";
		access = C;
		endnature

	discipline electrical
	potential Voltage;
	flow Current;
	flow Conductance;
	flow Resistance;
	flow Capacitance;
	end discipline	

A parameter, called *insideQucsStudio*, is also defined by QucsStudio to allow automatic linear or non-linear component selection from within blocks of Verilog-A model code, for example in a simple resistive case:

```
`ifdef insideQucsStudio
        R(b1)  <+ Rvalue;  // Linear R
`else
        I(b1)  <+ V(b1)/Rvalue; // Non-linear R
`endif;
```

Similarly, the QucsStudio parameter *insideQucsStudio* can be used as a switch to define an inductance. The Verilog-A module code for a linear inductance is listed in Fig. 7.


(a)
i_L
V_L
PinP1
PinN1
L
V_L
PinP
PinN
i_L
i_{n1}
n1
C
V_{n1}
$V_L = L \cdot di_L/dt$
Or, $i_L = (1/L) \cdot \int V_L \cdot dt$
By comparison L = C
Let $i_L = V_{n1}$ and $i_{n1} = V_L$
and $i_{n1} = d(C \cdot V_{n1})/dt$
Hence, $V_{n1} = (1/C) \cdot \int i_{n1}\, dt$
Or, $i_L = (1/C) \cdot \int V_L \cdot dt$
(b)
(c)
L
IND1
L=1e-9
PinP
PinN
C++
PinP
PinN
X1
File=ILinear.va
L=L

```
//  QucsStudio Verilog-A linear inductance model: Linind.va
`include "disciplines.vams"
`include "constants.vams"
`define attr(txt) (*txt*)
module ILinear(PinP, PinN);
inout PinP, PinN ;
electrical PinP, PinN;
electrical n1; // Internal node.
//
parameter real L = 1e-9 from [1e-20 : inf) `attr(info="Inductance value" unit = "H" );
// Model branches
branch (n1) bn1;
branch (PinP, PinN) bPinPPinN;
analog begin
// Current contribution etc
I(bPinPPinN) <+ V(bn1);
I(bn1) <+ -V(bPinPPinN);
`ifdef insideQucsSudio
  C(bn1) <+ L;
`else
  I(bn1) <+ ddt(L*V(bn1));
`endif
end
endmodule
```

Fig. 7. QucsStudio linear inductance L: (a) equivalent circuit and mathematical model, (b) Verilog-A code and (c) schematic symbol.

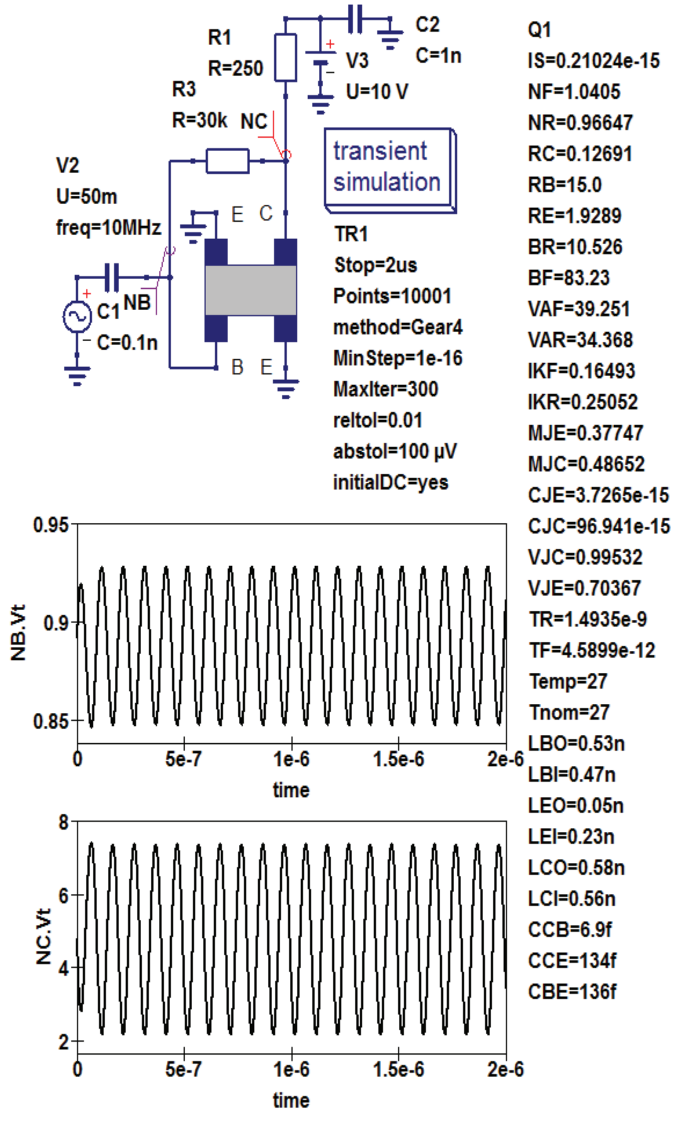
Fig. 8 presents the ILinear.va.cpp file generated by the ADMS software. The content of file ILinear.va.cpp illustrates how QucsStudio combines the predefined component specification template with calls to the QucsStudio C++ model application programming functions. Files of type XXXX.va.cpp are compiled by the MinGW C++ compiler, each time they change, to form a dynamically linked library component (dll) for a new component. QucsStudio also allows XXXX.cpp files to be attached to the C++ compiled model icon which in turn allows model developers, with the required programming skills, to construct optimized run time or minimized memory size device models.

```
/*
 * lLinear.va.cpp - device implementations for lLinear module
 * dynamically linked library for QucsStudio
 * created from VerilogAMS file by ADMS
 */
#include <qucsstudiolibrary.h>
#include <analogfunctions.h>
// component parameter definition
tParameterDef params[] = {
  {"L", "1e-9", false, "Inductance value"},
};
// external nodes
#undef PinP
#define PinP Nodes[0]
#undef PinN
#define PinN Nodes[1]
// internal nodes
#undef n1
#define n1 Nodes[2]
// global variables
// VerilogA analog functions.
// Evaluate VerilogA equations in analog block.
void fillMatrix(Component *theComp, Node **Nodes,
    tParameter *Parameters, tGlobalVar *gVar,
    EqnSystem *sys, double freq, double)
{
  // global variables.
  // Load device model input parameters.
  double L = getNumber(Parameters[0]);
  // ---------------- evaluate verilog initial model ----------------------
  // ---------------- evaluate verilog initial step -----------------------
  // ---------------- evaluate verilog initial instance -------------------
  // ---------------- evaluate verilog analog equations -------------------
  sys->setIQ(PinP,PinN,sys->getV(n1, GND),0);
  sys->setGC(PinP,PinN,n1,GND,1.0,0);
  sys->setIQ(n1,GND,(-sys->getV(PinP, PinN)),0);
  sys->setGC(n1,GND,PinP,PinN,(-1.0),0);
  sys->setIQ(n1,GND,0,(L*sys->getV(n1, GND)));
  sys->setGC(n1,GND,n1,GND,0,(L));
}
// component definition
EXPORT tComponentInfo compInfo = {
  isNonLinear,      // component type ('isNonLinear' or 'isLinear')
  "lLinear",            // model identifier
  "VerilogAMS model of lLinear",  // component description
  2,                    // number of external nodes
  1,                    // number of internal nodes
  0,                    // number of inputs (system simulations only)
  1,                   // number of parameters
  params,                 // pointer to list of parameters
  0,                    // size of global variable buffer in bytes
  0,                    // pointer to component icon (0 = unused)
  0,                    // size of component icon
  -1,                   // index of parameter determining schematic symbol (-1 = unused)
  0,                    // pointer to list of schematic symbols
  (tEvaluate)fillMatrix,     // function calculating analog model (0 = no model exists)
  0,                    // function calculating noise model (0 = noise free)
  0,                    // function calculating system model (0 = no model exists)
  0,                    // string with digital Verilog model (0 = no model exists)
  0                     // string with digital VHDL model (0 = no model exists)
};
```

Fig. 8. The contents of file lLinear.va.cpp showing the C++ code generated by the ADMS Verilog-A compiler and the QucsStudio C++ model application programming interface functions.

VI. Transient Simulation of a Class A RF npn BJT Amplifier

The circuit diagram drawn in Fig. 9 shows a single stage class A npn BJT RF amplifier with signal outputs taken directly from the BJT base and collector terminals respectively. Fig. 9 also illustrates a typical set of base and collector transient simulation waveforms for a 50mV peak, 10MHz sinusoidal signal applied to the amplifier input. Although the QucsStudio "turn-key" modelling system is primarily designed to be a fast simple to use development tool



transient simulation
TR1
Stop=2us
Points=10001
method=Gear4
MinStep=1e-16
MaxIter=300
reltol=0.01
abstol=100 µV
initialDC=yes

Q1
IS=0.21024e-15
NF=1.0405
NR=0.96647
RC=0.12691
RB=15.0
RE=1.9289
BR=10.526
BF=83.23
VAF=39.251
VAR=34.368
IKF=0.16493
IKR=0.25052
MJE=0.37747
MJC=0.48652
CJE=3.7265e-15
CJC=96.941e-15
VJC=0.99532
VJE=0.70367
TR=1.4935e-9
TF=4.5899e-12
Temp=27
Tnom=27
LBO=0.53n
LBI=0.47n
LEO=0.05n
LEI=0.23n
LCO=0.58n
LCI=0.56n
CCB=6.9f
CCE=134f
CBE=136f

Fig. 9. A class A npn BJT RF amplifier with collector feedback: transient simulation test circuit with directly coupled voltage test probes NB and NC, and time response output waveforms for a sinusoidal input signal of 50mV peak, 10MHz frequency and zero input phase.

for constructing equation-defined compact device models it also works along-side an advanced post-simulation data processing and visualization package which incorporates the GNU GPL Octave numerical analysis software [14]. The current version of QucsStudio allows simulation output data to be numerically processed by the Octave package, following completion of a circuit simulation task. For example, the voltage amplitude spectra shown in Fig.10 were computed using the Octave "m" script listed in Fig. 11. This illustrates the use of Octave functions and statements for converting QucsStudio simulation data into the Octave data format, the use of the Octave function fft to perform a fast Fourier transform of output data NB.Vt and NC.Vt and finally how Octave visualization statements can generate the output data plots shown in Fig.10.

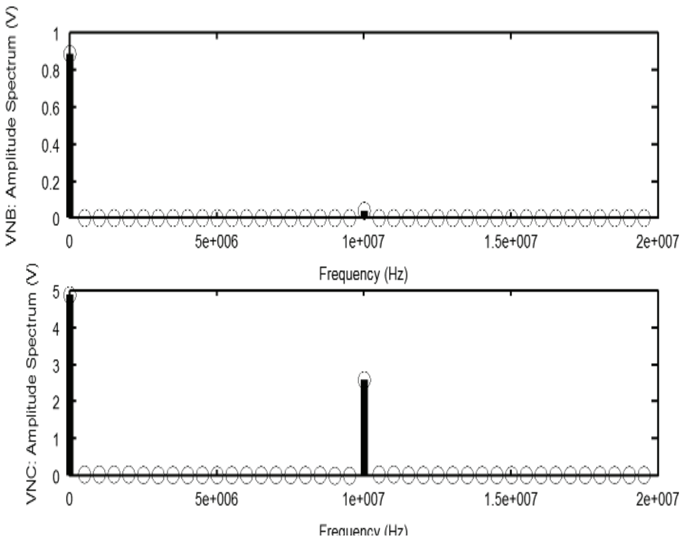


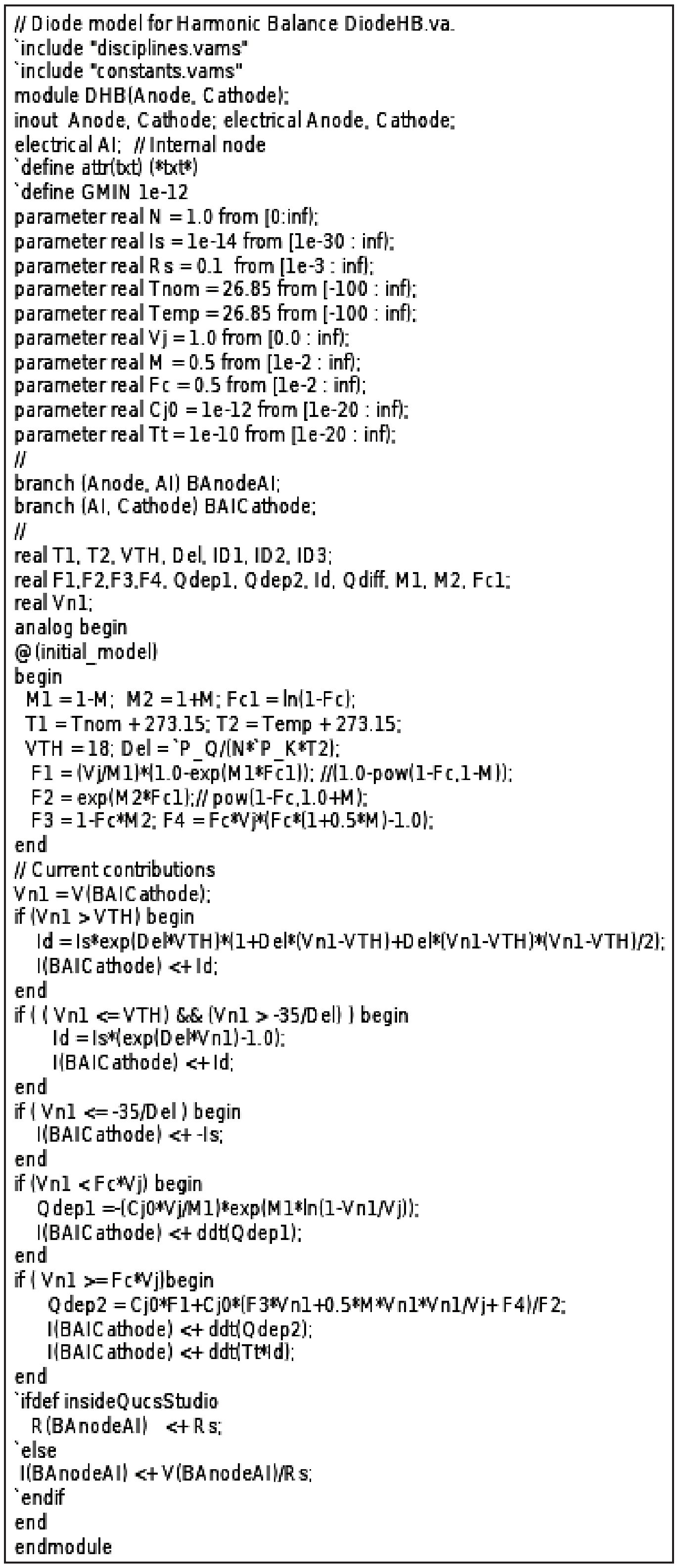
Fig. 10. Voltage amplitude spectra plotted against frequency for amplifier probe signals NB and NC.

```
% File testfeedbacknpnTRAN.m file
% Control file called on completion of transient simulation.
% Calls functions loadQucsDataset,loadQucsVariable
% and stemfft.
function stemfft(data, points, FinTim)
 yfft  = abs(fft(data/points));
 no2   = length(yfft)/2;
 yvec(1) = yfft(1);
 yvec([2:no2])  = 2*yfft([2:no2]);
 fc   = linspace(0, no2, no2)/FinTim;
 stem(fc, yvec, "linewidth", 4 , "color", "black");
endfunction

qucsFilename = 'TestfeedbacknpnTRAN.dat';
loadQucsDataset;
whos
clf()
newplot()
[VNB,Dep]=loadQucsVariable("TestfeedbacknpnTRAN.dat","NB.Vt");
[VNC,Dep1]=loadQucsVariable("TestfeedbacknpnTRAN.dat","NC.Vt");
[Time,Dep3]=loadQucsVariable("TestfeedbacknpnTRAN.dat","time");

subplot(2,1,1)
stemfft(VNB, 2048, 2e-6);
set(gca, "linewidth", 4, "fontsize", 20,  "fontname", "TimesRoman",
 "fontweight", "bold",  "xlim", [0,20e6], "xlabel", text("string",
 "Frequency (Hz)", "fontsize", 20, "fontname", "TimesRoman",
 "fontweight", "bold"), "ylabel",   text("string",
 "VNB: Amplitude Spectrum (V)", "fontsize", 20,
 "fontname", "TimesRoman", "fontweight", "bold", "rotation", 90));

subplot(2,1,2);
stemfft(VNC, 2048, 2e-6);
set(gca, "linewidth", 4, "fontsize", 20,  "fontname", "TimesRoman",
 "fontweight", "bold", "xlim", [0,20e6], "xlabel", text("string",
 "Frequency (Hz)", "fontsize", 20, "fontname", "TimesRoman",
 "fontweight","bold"), "ylabel",   text("string",
 "VNC: Amplitude Spectrum (V)", "fontsize", 20,
 "fontname", "Arial", "fontweight", "bold", "rotation", 90));
print("TestfeedbacknpnTRAN.png","-dpng");
```

Fig. 11. Octave "m" script for post-simulation amplifier data processing: Functions “loadQucsDataset” and “loadQucsVariable” are provided with the QucsStudio software for conversion of simulation output data to Octave internal format.

VII. Harmonic Balance Analysis of an RF Diode Switch

Conventional SPICE 2g6 [15] and 3f5 [16] circuit simulators have only limited RF simulation functionality, for example, simulators based on SPICE 2g6 or 3f5 do not normally include in their simulation repertoire Harmonic Balance analysis, or indeed other equivalent RF analysis techniques. This section introduces a simple compact semiconductor diode model which is suitable for Harmonic Balance simulation, a 50 Ω RF diode switch subcircuit and a basic small signal/large signal AC/Harmonic Balance simulation test circuit for investigating RF switch performance. Listed in Fig. 12 is the Verilog-A code for the simplified compact semiconductor diode model. This model is characterized by a set of non-linear current and capacitance equations that have been modified to give quadratic continuous voltage function extensions to the basic diode current and capacitance characteristics at Vd > VTH. These changes ensure that the first and second order derivatives of current Id and charge Qd, with respect to diode voltage Vd, are continuous during Harmonic Balance simulation, ensuring a higher probability that circuit solution convergence occurs. For simplicity other RF switch properties, like noise and temperature effects, are not modeled in the demonstration example. Fig. 13 shows a subcircuit version of a typical two diode 50 Ω RF switch. In Fig. 13 (a) the diodes are represented by QucsStudio C++ compiled components. In Fig.13 (b) the 50 Ω RF switch symbol depicts a conventional on-off switch who’s properties are controlled by input signal Control; Control = 0V sets the switch to off and Control = 1V sets the switch to on. The remaining components which make up the switch subcircuit (L1, C1 and R1) are for signal isolation (L1), AC signal coupling (C1) and impedance matching (R1) purposes, respectively. Fig. 14 (a) introduces a very basic test circuit for observing the effect that RF switch performance has on signal transmission. In Fig. 14 (a) the control parameter state determines whether the switch is closed or open: closed when state=1 and open with state=0. The waveforms shown in Fig.14 (b) confirm the basic operation of the RF switch. With State=0 the switch is off and the RF output voltage is approximately zero volts at frequencies up to roughly 100 MHz. However, at higher frequencies the output RF signal rises, indicating that signal feed-through limits the isolation performance of the switch. With State=1 the switch is closed, transmitting RF signals from input to output. Over the frequency range 10 MHz to 10 GHZ the voltage transfer ratio is approximately one, clearly indicating good performance. However, below 10 MHz switch performance is largely determined by capacitive AC signal coupling. Similarly, above 10 GHz signal loss due to capacitive effects lowers the switch voltage transfer ratio below one.



```
// Diode model for Harmonic Balance DiodeHB.va.
`include "disciplines.vams"
`include "constants.vams"
module DHB(Anode, Cathode);
inout  Anode, Cathode; electrical Anode, Cathode;
electrical AI;  // Internal node
`define attr(txt) (*txt*)
`define GMIN 1e-12
parameter real N = 1.0 from [0:inf);
parameter real Is = 1e-14 from [1e-30 : inf);
parameter real Rs = 0.1  from [1e-3 : inf);
parameter real Tnom = 26.85 from [-100 : inf);
parameter real Temp = 26.85 from [-100 : inf);
parameter real Vj = 1.0 from [0.0 : inf);
parameter real M  = 0.5 from [1e-2 : inf);
parameter real Fc = 0.5 from [1e-2 : inf);
parameter real Cj0 = 1e-12 from [1e-20 : inf);
parameter real Tt = 1e-10 from [1e-20 : inf);
//
branch (Anode, AI) BAnodeAI;
branch (AI, Cathode) BAICathode;
//
real T1, T2, VTH, Del, ID1, ID2, ID3;
real F1,F2,F3,F4, Qdep1, Qdep2, Id, Qdiff, M1, M2, Fc1;
real Vn1;
analog begin
@(initial_model)
begin
  M1 = 1-M;  M2 = 1+M; Fc1 = ln(1-Fc);
  T1 = Tnom + 273.15; T2 = Temp + 273.15;
  VTH = 18; Del = `P_Q/(N*`P_K*T2);
   F1 = (Vj/M1)*(1.0-exp(M1*Fc1)); //(1.0-pow(1-Fc,1-M));
   F2 = exp(M2*Fc1);// pow(1-Fc,1.0+M);
   F3 = 1-Fc*M2; F4 = Fc*Vj*(Fc*(1+0.5*M)-1.0);
end
// Current contributions
Vn1 = V(BAICathode);
if (Vn1 > VTH) begin
    Id = Is*exp(Del*VTH)*(1+Del*(Vn1-VTH)+Del*(Vn1-VTH)*(Vn1-VTH)/2);
    I(BAICathode) <+ Id;
end
if ( ( Vn1 <= VTH) && (Vn1 > -35/Del) ) begin
       Id = Is*(exp(Del*Vn1)-1.0);
       I(BAICathode) <+ Id;
end
if ( Vn1 <= -35/Del ) begin
    I(BAICathode) <+ -Is;
end
if (Vn1 < Fc*Vj) begin
    Qdep1 =-(Cj0*Vj/M1)*exp(M1*ln(1-Vn1/Vj));
    I(BAICathode) <+ ddt(Qdep1);
end
if ( Vn1 >= Fc*Vj)begin
      Qdep2 = Cj0*F1+Cj0*(F3*Vn1+0.5*M*Vn1*Vn1/Vj+ F4)/F2;
      I(BAICathode) <+ ddt(Qdep2);
      I(BAICathode) <+ ddt(Tt*Id);
end
`ifdef insideQucsStudio
  R(BAnodeAI)    <+ Rs;
`else
 I(BAnodeAI) <+ V(BAnodeAI)/Rs;
`endif
end
endmodule
```

Fig. 12. Verilog-A code for a compact semiconductor diode model with quadratic function extensions for Id/Vd and depletion capacitance characteristics: to ensure convergence the exponential function quadratization threshold voltage (VTH) is set at 18V.

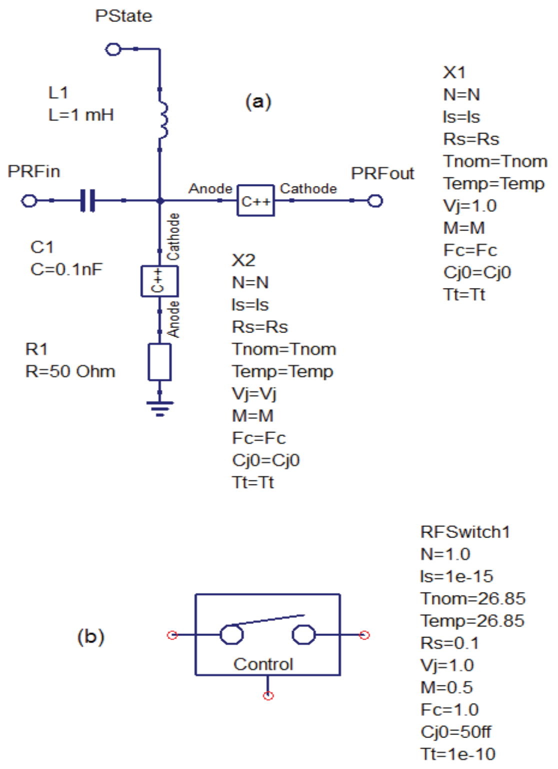


Fig. 13. A 50 Ω RF switch subcircuit constructed from a modified SPICE semiconductor diode models suitable for Harmonic Balance simulation: (a) the 50 Ω RF switch subcircuit and (b) the switch schematic capture symbol; when Control = 0 the switch is open and when Control = 1 the switch is closed.

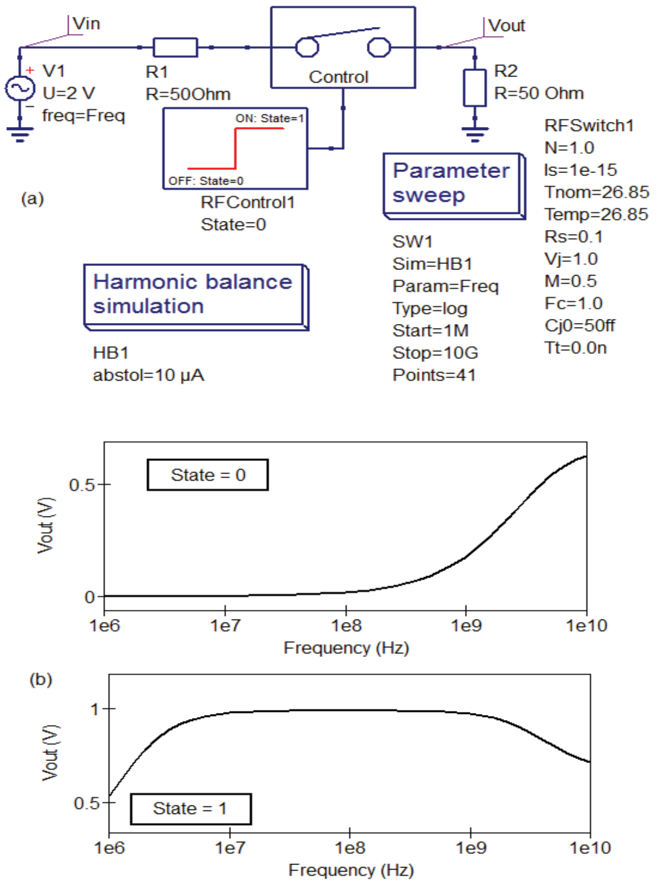


Fig. 14. (a) RF switch large signal AC test circuit; (b) typical output voltage transfer characteristics for a sinusoidal voltage Vin of 2V peak: the RF switch is closed when control parameter state=1 and open when state=0.

VIII. CONCLUSIONS

Compact device modelling with Verilog-A has become standard practice among commercial and GNU General Public License circuit simulators, with many packages adopting the ADMS Verilog-A to C++ model synthesizer as the central core in their device modelling strategy. Initial open source implementations of the ADMS model synthesizer often depended on model developers patching simulator C++ code when constructing new models. This approach not only requires developers to have a good understanding of a particular circuit simulators model application interface but is likely to be error prone. The introduction of the QucsStudio "turn-key" approach to compact device modeling provides for the first time, as far as the authors are aware, a freely available, fast, and simple to use, GNU General Public License modelling tool which does not require users to manually patch circuit simulator C++code.

REFERENCES

[1] Compact Model Council, TechAmerica, Arlington, VA. http://www.gela.org/About-TechAmerica, 2009. [accessed January 2012].

[2] Accellera, "Verilog-AMS Language Reference Manual, version 2.2", 2004, http://www.accellera.org, 2010. [accesssed January 2012].

[3] L. Lemaitre, G. Coram, C. McAndrew and K. Kundert, "Extensions to Verilog-A to support compact device modeling", Proceedings of the IEEE International Workshop on Behavioural Modeling and Simulation, BMAS, 7-8 Oct. 2003, pp, 134-138.

[4] L. Lemaitre. ADMS, http://adms.noovela.com:8001/, 2007. [accessed January 2012].

[5] Smash mixed signal simulator, Version 5.18.0, Dolphin Integration, France, http://www,dolphin.fr/medal/smash/smash_overview.php, 2011. [accessed January 2012].

[6] Symica Custom IC Design Toolkit, Symica LLC, 2009-2012, http://www.symica.com/products/symica-de, 2012. [accessed January 2012].

[7] P Nenzi, ngspice release 23, http://ngspice.sourceforge.net/index.html, 2011. [accessed January 2012].

[8] A. Davis, Gnucap, Version 0.35, http://www,gnu.org/software/gnucap/, 2008. [accessed January 2012].

[9] M. Margraf, QucsStudio, Version 1.3.0, http://www.mydarc.de/DD6UM/QucsStudio/qucsstudio.html, 2012. [accessed 2012].

[10] M. Margraf, S.Jahn, J. Flucke, R. Jacob, V. Habchi, T. Ishikawa, A. Gopala Krishna, M. Brinson, H. Parruitte, B.Roucaries and G. Kraut, Qucs (Quite universal circuit simulator), Version 0.0.16, 2011, http://qucs.sourceforge.net/index.html, [accessed January 2012].

[11] M. Brinson and S. Jahn, Building device models and circuit macromodels with the Qucs GPL circuit simulator, COMON project meeting, IHP, Frankfurt (Oder), Germany, 2009, http://www.mos-ak.org/frankfurt_o/papers/M_Brinson_Qucs_COMON_April_2_2009_final.pdf. [accessed January 2012].

[12] P. Antognetti and G. Massobrio (Editors), "Semiconductor device modeling with SPICE", McGraw-Hill Book Company, New York, 1988.

[13] S. Jahn and M.E. Brinson, "Interactive compact device modelling using Qucs equation-defined devices", International journal of Numerical Modelling: Electronic Networks, Devices and Fields, 2008, 21:335-349.

[14] J.W. Eaton et al., Octave, http://www.gnu.org/software/octave/about.html. [accessed January 2012].

[15] A.R. Newton, D.O. Pederson, A. Sangiovanni-Vincentelli, "A SPICE 2g User's Guide", Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1981.

[16] B. Johnson, T. Quarles, A.R. Newton, D.O. Pederson, A. Sangiovanni-Vincentelli, "A SPICE3 Version 3f User's Manual", Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1992.

M.E. Brinson received a first class honours BSc degree in the Physics and Technology of Electronics from the United Kingdom Council for National Academic Awards in 1965, and a PhD in Solid State Physics from London University in 1968. Since 1968 Dr. Brinson has held academic posts in Electronic and Computer Science. From 1997 till 2000 he was a visiting Professor of Analogue Microelectronics at Hochschule, Breman, Germany. Currently, he is a visiting Professor at the Centre for Communication Technology, London Metropolitan University. He is a Chartered Engineer (CEng) and a Fellow of the Institution of Engineering and Technology (FIET), a Chartered Physicist (CPhys), and a member of the Institute of Physics (MInstP). Prof. Brinson Joined the Qucs project development team in 2006, specialising in device and circuit modelling, testing and document preparation.

M. Margraf received the Dipl.-Ing degree and the Doctoral degree in Electrical Engineering from the University of Technology, Berlin, Germany, in 2001 and 2003 respectively. His main area of research has been modelling of noise and non-linearities in microwave circuits. Since 2003 Dr. Margraf has worked on programming open source applications for simulating electronic circuits.