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

M. E. Brinson
Centre for Communications Technology
London Metropolitan University
London N78DB, UK
mbrin72043@yahoo.co.uk

M. Margraf
Qucs and QucsStudio Project Founder
Berlin
Germany
michael.margraf@alumni.tu-berlin.de

***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 being two examples of open source circuit simulators that use ADMS. 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 the circuit simulator C++ code. At the start of a 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. In this paper QucsStudio "turn-key" modelling is illustrated by the design of a single stage RF amplifier circuit.**

***Keywords*--QucsStudio, ADMS, Verilog-A, compact device modelling, turn-key component modelling.**

## I. INTRODUCTION

Until the adoption of Verilog-A as the preferred analogue hardware description language for compact semiconductor device modelling by the Compact Model Council [1], 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 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 a 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 other important circuit design, simulation and manufacturing features.

## 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 compile Verilog-A model code to C++ code manually. 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 to be re-compiled and re-linked to generate a new extended 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 modeling process required users to have an advanced knowledge of C++ programming 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. QucsStudio-ADMS "Turn-Key" Verilog-A Compact Device Modelling

A primary aim of the QucsStudio Verilog-A compact device modelling system is to provide the circuit simulation software 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. 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. The implemented modelling system has been called a "turn-key" system 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 changed 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, outlining each of the QucsStudio modelling stages. In this diagram the modelling process is shown starting from Verilog-A code entry at the top, using a built-in text editor, followed by attachment of a model Verilog-A code file (XXXX.va) to a QucsStudio "C++ compiled model " icon, through synthesis of the C++ model code, using the ADMS software, to C++ compilation and finally construction of a model subcircuit at the bottom. 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 library (XXXX.dll) for the model under construction. The last two stages only take place if the original Verilog-A module code is error free. On successful generation of the "C++ compiled model" it is linked to the main body of QucsStudio software and becomes a standalone simulation component. It can also be combined with other QucsStudio components by attaching it to a QucsStudio schematic diagram, and indeed to other "C++ compiled models", to form a subcircuit or macromodel. During the simulation of circuits which include C++ compiled models changes to their Verlog-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.

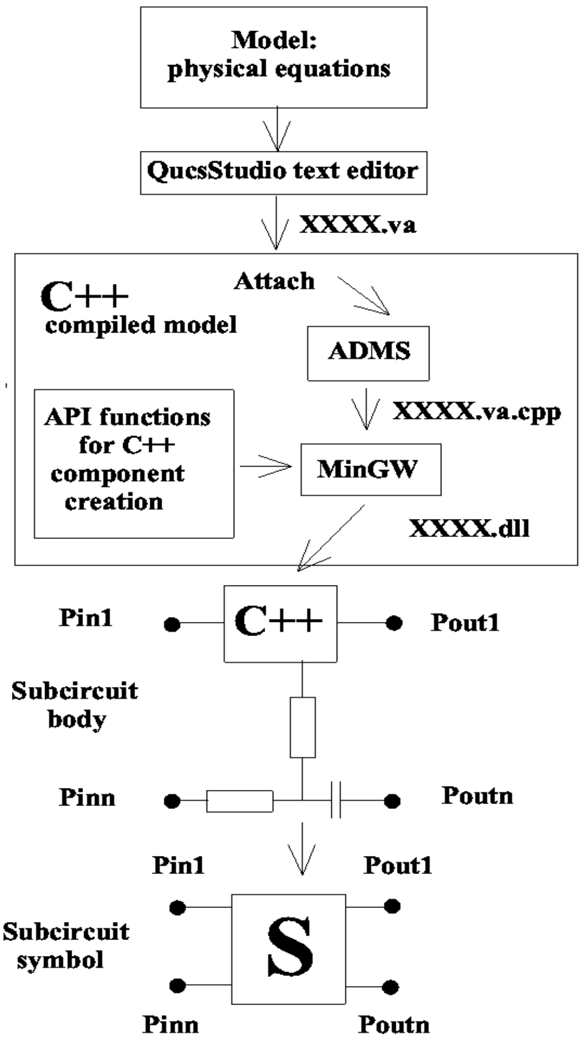


**Figure 1. A flow chart outlining the QucsStudio "Turn-key" Verilog-A compact model development system**

To demonstrate the QucsStudio Verilog-A "turn-key" modelling procedure the construction of a simple RF npn BJT compact device model is presented next. The model information given in Fig. 2 is based on a large signal Ebors-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 inductance and capacitance. The Verilog-A code for the RF npn BJT is listed in Fig. 3. On attaching this code to a QucsStudio C++ compiled model icon the software tries to extract the external node names and parameters. If successful QucsStudio draws a group of named terminals attached to the C++ compiled model icon. These correspond in name and list order to the "inout" terminals given in the Verilog-A module statement. Shown in Fig.4 is a single stage class A RF npn BJT amplifier circuit, with collector feedback, configured as a small signal AC simulation over the frequency band 1MHz to 8GHz.

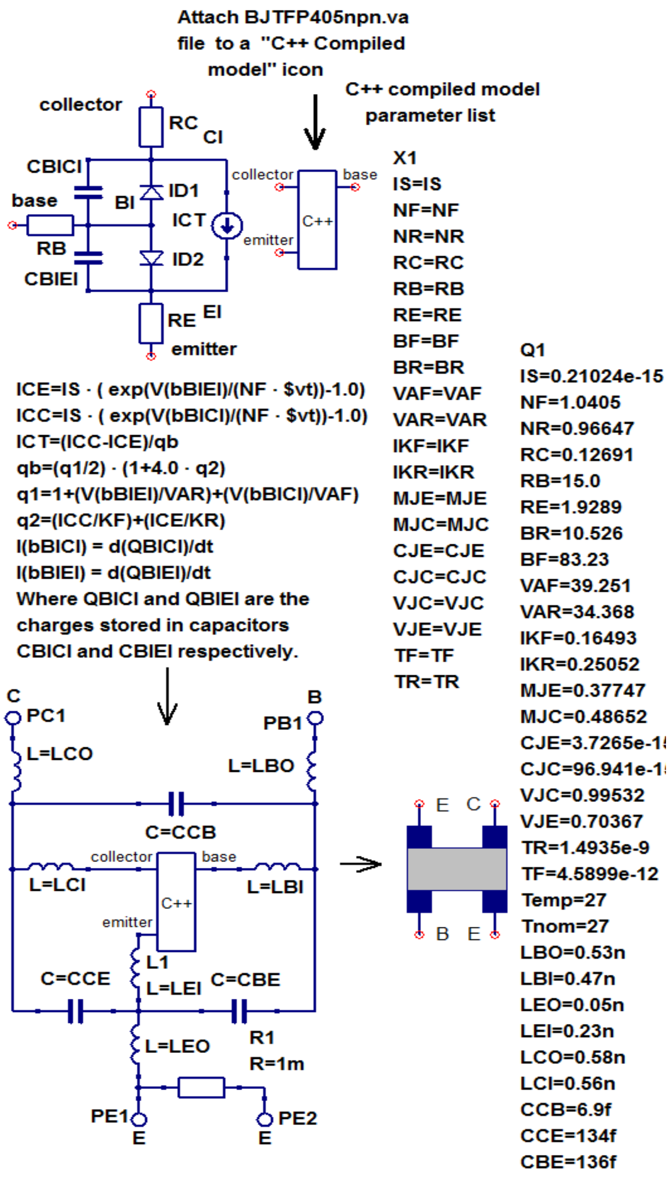


**Figure 2. A QucsStudio "turn-key" RF npn BJT model outline showing: Ebors-Moll equations and equivalent circuit plus second order high-level current injection effects and internal capacitance equations; C++ compiled model icon with parameters (X1); subcircuit body and symbol plus parameters (Q1).**

## IV. QucsStudio C++ compiled Model Programming Interface

Central to the operation of the QucsStudio Verilog-A "turn-key" modelling system is a C++ compiled model component. An outline of the structure and content of a compiled model coponent is listed in Fig.5. In general QucsStudio built-in component models are defined by a C++ code template which lists model properties. The list contains amongst other things the number of external and internal nodes as well as pointers to the parameter list and to the schematic symbol. It also contains function calls, like (tEvaluate)Matrix in Fig.5, that determine the physical operation of a component. Many of these are optional. The production version of the QucsStudio software is provided with a number of detailed examples of 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.

```
//  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
```

**Figure 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 ].**

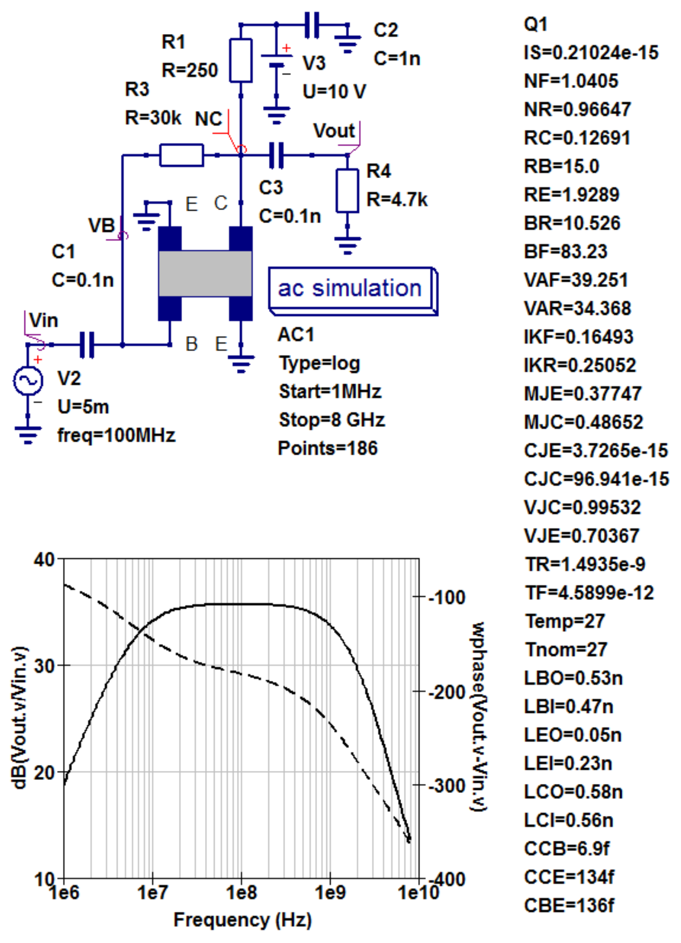


**Figure 4. A class A npn BJT RF amplifier with collector feedback: small signal AC test circuit and example gain and phase response curves; legend: solid line = dB(Vout.v/Vin.v) the amplifier gain in dB and dotted line = wphase(Vout.v/Vin.v) the unwrapped phase of the amplifier gain in degrees.**

In the case of the RF npn BJT 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. 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.

## V. Adding Verilog-A Natures to QucsStudio

A high percentage of compact device models include a mixture of linear and non-linear R, L and C components. The 2.3.0 version of ADMS appears to treat all R and C components as non-linear elements. Also in this version of ADMS it is not permitted to express the connection of inductance L between circuit nodes p and n as V(p,n) <+ L*ddt(I(p,n)). However, this limitation can be overcome by combining a capacitor with a gyrator [13] to form a linear or non-linear inductance. In those compact models with a significant number of R, C and L elements simulation run

```
// 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)
};
```

**Figure 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

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

times can be reduced, especially transient simulation, by ensuring that the QucsStudio "turn-key"modeling system uses linear R, C and L components whenever possible rather than non-linear devices. In the QucsStudio software the selection of linear or non-linear fundamental R, C or L components has been implemented by adding three new Verilog-A "natures" to the standard disciplines and natures file "disciplines.vams". These are listed in Table I. A parameter, called insideQucsStudio, is also defined by QucsStudio to allow automatic linear or non-linear component selection from within

TABLE I. QUCSSTUDIO NATURES PLUS MODIFIED ELECTRICAL DISCIPLINE FOR LINEAR R AND C COMPONENTS

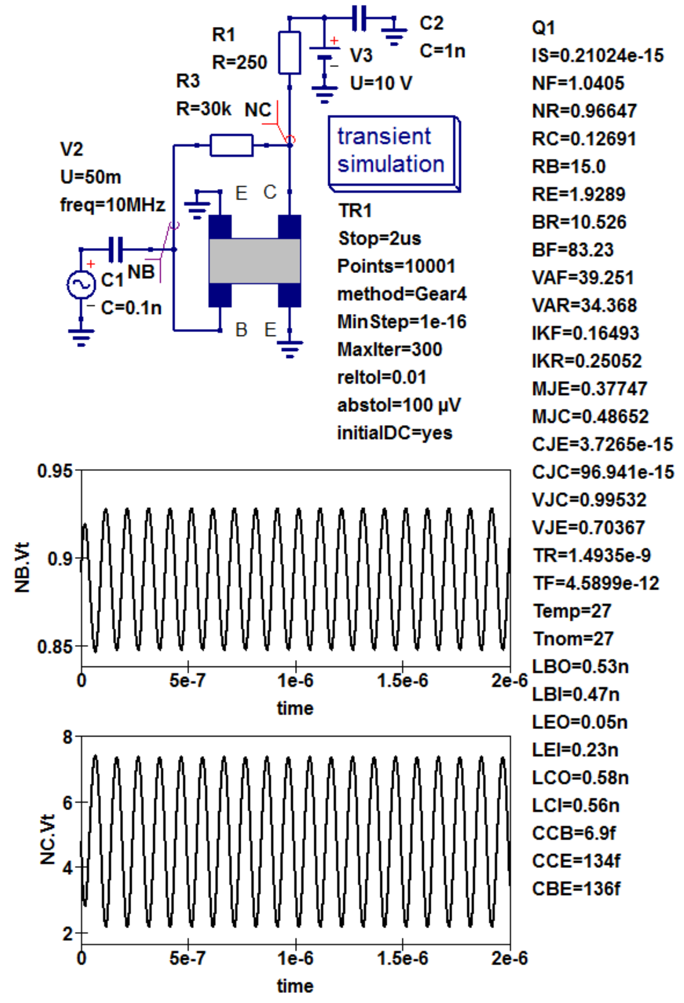
```
nature Resistance        nature Conductance        nature  Capacitance
      units = "ohms";          units = "s";              units = "F";
      access = R;              access                    access = C;
endnature                  endnature                 endnature

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

Verilog-A model code, for example in the resistive case:

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

## VI. TRANSIENT SIMULATION OF A CLASS A RF NPN BJT AMPLIFIER

The circuit diagram drawn in Fig. 7 shows a single stage class A npn BJT RF amplifier with signal outputs taken directly from the BJT base and collector terminals respectively. Fig. 7 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 for constructing equation-defined compact device models it also works along-side an advanced post-simulation data processing and visualization package incorporating 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.8 were computed using the Octave "m" script listed in Fig. 9. 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.8.



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

**Figure 7. 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.**

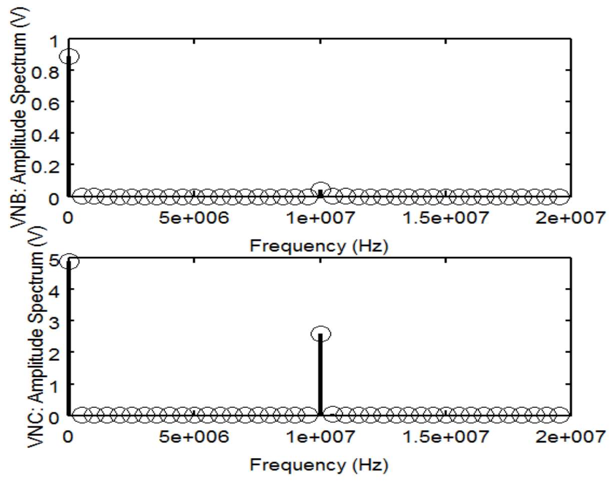


**Figure 8. Voltage amplitude spectra plotted against frequency for amplifier probe signals NB and NC.**

```
% 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");
```

**Figure 9. 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. 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 anuary 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].