Notes on Constructing Qucs Verilog-A Compact Device Models and Circuit Macromodels

Mike Brinson and Stefan Jahn

***Abstract*—The Qucs project regularly receives requests from users of the circuit simulation package for more information on how to add Verilog-A compact device models and circuit macro-modelling to Qucs. The notes presented in this paper outline the steps that model developers must follow when adding new Verilog-A models to either Qucs-0.0.16 (the current production version of the package) or Qucs 0.0.17 (the current development version of the package). At the present time Qucs is developed using the Linux operating system, the GNU-Linux autotools and the freely available ADMS Verilog-A model synthesizer/compiler. Hence, these notes apply only to the Linux operating system version of Qucs. Qucs users who are interested in constructing their own experimental Verilog-A models are advised to switch to the Linux operating system if they are not already using it.**

***Index Terms*—Qucs, equation-defined devices, compact device models, Verilog-A, macromodels and subcircuits.**

I. Introduction

VERILOG-A compact device model construction using the ADMS Verilog-A synthesizer/compiler [1] was added to the Qucs simulator to allow compact semiconductor device model, and integrated circuit macromodel development, via an internationally standardized hardware description language. In recent years Verilog-A has been adopted by the modelling community as the language of choice for new model construction, primarily because of its extensive modelling capabilities and ease of use. These notes make no attempt to teach the fundamentals of the Verilog-A language or its use for device modelling or circuit macromodelling. Readers who are not familiar with Verilog-A should consult the books by Patrick and Miller [2] and Kundert and Zinke [3]. Details of the latest version of the Verilog-A language can also be found in the Verilog-A language reference manual [4]. Qucs first used the ADMS Verilog-A compiler in October 2006. At that time a series of complex changes had to be made manually to the Qucs C++ code in order to add schematic capture symbol C++ code to the simulator and to merge model C++ code generated by the ADMS Verilog-A compiler with the Qucs core code. Instructions for this process were described in the following publications: Stefan Jahn and Hélène Parruitte [5] and Mike Brinson and Stefan Jahn [6]. Although the initial procedure for adding Verilog-A models to Qucs was perfectly viable it was more suited to Qucs developers or Qucs users who had a good knowledge of the software techniques employed in Qucs development. Indeed a number of Qucs users mastered the process and used the ADMS/Qucs combination as a powerful model development system, see for example reference [7]. One of the changes introduced in Qucs version 0.0.16 has been a procedure which allows significant simplification in the Qucs Verilog-A model development route. The modified development route still requires users to make a number of manual changes to the core Qucs C++ code. However, such changes have been minimized and combined with a set of convenient automated tools which allow easy entry of Verilog-A code via a colour highlighted editor and automatic generation of model schematic capture symbols plus their C++ code. The overall process still employs static C++ libraries which requires that the entire Qucs C++ code be recompiled after a new model is merged with the core Qucs code. In the long term it is proposed that Qucs will move to the use of dynamic C++ code, allowing a simpler compiling and linking procedure to be adopted in future Qucs releases. However, it is estimated that this will require a significant amount of work on the existing Qucs SVN code and could only be done if development time can be found for the generation of a more "turn-key" approach to Qucs Verilog-A model development[1]. These notes outline the process for adding new Verilog-A models to the Linux versions of Qucs. The complete process, from entering Verilog-A text to model testing, is introduced via the construction of a non-linear resistance model. As a starting point it is assumed that readers have downloaded the current Qucs and ADMS SVN code from http://qucs.sourceforge.net/ (Qucs) and http://www.noovela.com:8001/svn/adms/trunk/ (ADMS) then compiled the Qucs/ADMS packages successfully.

II. Qucs Equation-defined device and Verilog-A Model Construction

Over the last few years Qucs has evolved from simply another circuit simulator to a software package which offers users a range of powerful modelling facilities for the development of new compact semiconductor device models and integrated circuit macromodels. The simulator includes a very stable equation-defined device model (EDD) and an equivalent radio frequency version of EDD. These components allow interactive development of new non-linear Qucs component models. However, readers should be aware that these models are of an interpretive form and do not, in most instances, simulate at the same speed as native C++ models. Their

Mike Brinson is a Professor at the Centre for Communications Technology, London Metropolitan University, London UK. He is also a member of the Qucs Development Team.

Stafan Jahn is the Manager of the Qucs project. He is based in Munich, Germany.

[1]The investment of a large amount of Qucs "Developers" time to make the change C++ static code to dynamic code would only be worth while if significant numbers of Qucs users planned to use the proposed "turn-key" Verilog-A model development route. At this time the availability of stable non-linear EDD and REDD components, within the Qucs modelling facilities, and the latest simplified Verilog-A development route is considered to be more than adequate to meet the requirements of the majority of Qucs model designers.

great advantage is the fact that they allow fast and easy construction of new models where changes can be simply made prior to testing. On the other hand Verilog-A models are normally compiled to C++ code. The compiled code, when linked to the body of the Qucs simulator, allows faster model operation which often approaches the speed obtained by hand-crafted C++ device models. The main downside factors to Verilog-A model development are as follows: model developers require; (1) a good working knowledge of the Verilog-A hardware description language, (2) a good understanding of the Qucs C++ simulator code and (3) substantial time to complete the development phase involving the merger of the schematic capture C++ code and the ADMS generated C++ code to the main body of the Qucs code. In general the development of Verilog-A models is normally restricted to components where the investment of model development time is justified, for example complex semiconductor models for MOSFET devices, advanced circuit macromodels such as switched capacitor mixed-mode designs or complex models which require large amounts of computation each time the model is accessed during simulation. 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 [8].

III. Building a Qucs EDD Model for a Voltage Controlled Non-linear Resistor

Fig. 1 shows a Qucs EDD model for a simple non-linear resistor with a resistance value that is a quadratic function of the device branch voltage. Two coefficients A and B determine the shape of the non-linear resistive function. In Fig. 1 these coefficients, with the nominal value of the device resistance $R0$, are passed as subcircuit parameters to the EDD component. Fig. 2 illustrates a resistive voltage divider test circuit consisting of the non-linear resistor in series with a standard 1k Ohm resistor. Fig. 2 also gives a number of plots of the circuit properties as a function of applied input DC voltage Vsw.

IV. Construction of a Qucs Verilog-A Model for a Voltage Controlled Non-linear Resistor: Part 1; Entering Verilog-A code and generation of a schematic Symbol

Qucs version 0.0.16 includes a text editor which selectively colour highlights different Verilog-A statements, numbers and comments, making entry and checking of compact model code particularly easy. Illustrated in Fig. 3 is the Verilog-A code for the nonlinear resistor introduced in the last section of this paper. 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. Initially, this is in a simple block form which can be edited using the Qucs ”Painting“ tools to give any desired schematic symbol outline. Fig. 4 presents both the original

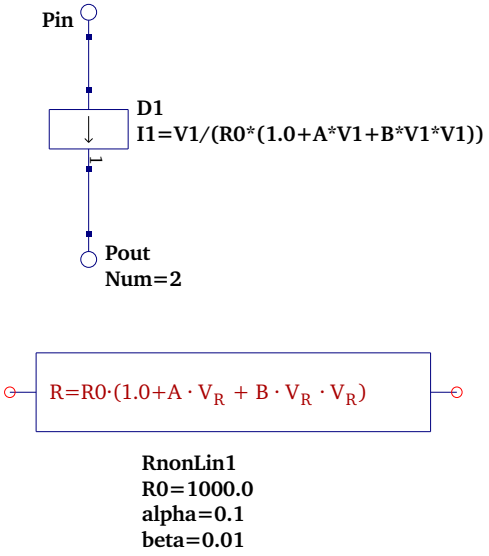

Fig. 1. A Qucs non-linear resistance EDD model plus its schematic symbol

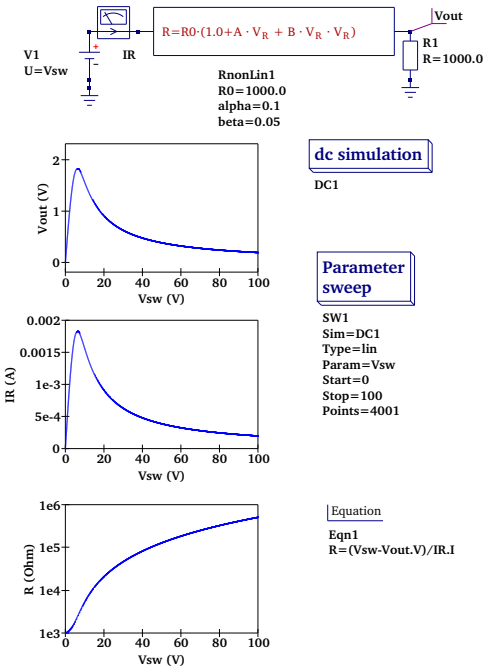

Fig. 2. A non-linear resistance test voltage divider test circuit and a set of typical simulation graphs for the circuit properties

block symbol and a basic edited symbol, including the non-linear resistance equation. Saving the model symbol causes Qucs to automatically generate the C++ code for the new symbol. In the case of the RnonLin model this is held in file $RnonLin.dat$ [2]. This code, shown in Fig. 5, is needed at a later stage of the Verilog-A model development process.

[2]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.

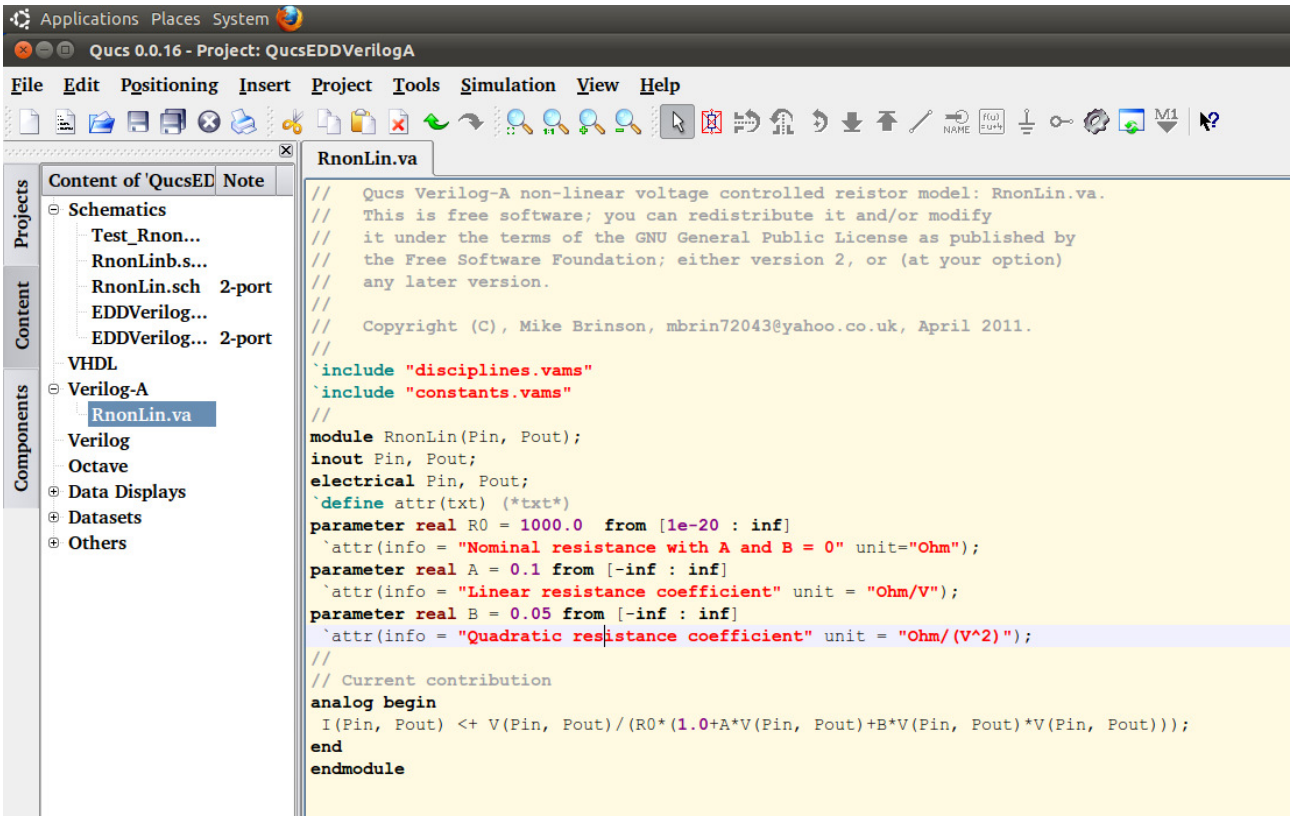

Fig. 3. Verilog-A code for the example non-linear resistor

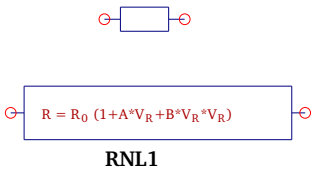

Fig. 4. Qucs schematic symbols for the example non-linear resistance: (a) original block symbol and (b) final edited symbol

V. Construction of a Qucs Verilog-A Model for a Voltage Controlled Non-linear Resistor: Part 2; Compiling the Verilog-A Model Code

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 to generate a C++ code representation of the original Verilog-A code. First copy file $RnonlLin.va$ from project directory $QucsEDDVerilogA\_prj$ to Qucs Verilog-A source code directory **/tmp/qucs-core/src/components/verilog**.[3] 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

[3]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**. However, use of a home directory does assume that Qucs has been installed in the specified home directory.

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 $RnonLin.va$ with 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
[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 compilation 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)

VI. Construction of a Qucs Verilog-A Model for a Voltage Controlled Non-linear Resistor: Part 3; combining RLin.gui.cpp code with Qucs C++ symbol code for component RnonLin

Provided the instructions in section 2 of this paper were correctly actioned directory **/tmp/qucs-core/src/components/verilog** should contain the following files relating to component RnonLin: (1) $RnonLin.va$, (2) $RnonLin.core.cpp$, (3) $RnonLin.core.h$, (4) $RnonLin.defs.h$, (5) $RnonLin.gui.cpp$, (6) $RnonLin.gui.h$, (7) $RnonLin.analogfunction.cpp$, and (8) $RnonLin.analogfunction.h$. Copy file (5) $RnonLin.gui.cpp$, and file (6) $RnonLin.gui.h$, 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 $RnonLin.cpp$ file (Fig. 6)[4]: (1) change the file name from $RnonLin.gui.h$ to $RnonLin.h$, and the T in the line Name = "T"; to some other more appropriate abbreviation, like Name = "RNL"; (2) replace the blue instruction text at the bottom of file $RnonLin.cpp$ with the C++ code held in file $RnonLin.dat$, see Fig.5; (3) replace the green highlighted code in file $RnonLin.cpp$ with the following C++ code[5]:

```
// tx = x2+4;
// ty = y1+4;
tx = -10; ty = -24;
```

VII. Construction of a Qucs Verilog-A Model for a Voltage Controlled Non-linear Resistor: Part 4; constructing a 32 bit by 32 bit icon for component RnonLin

Qucs directory **/tmp/qucs/qucs/bitmaps** contains 32 bit by 32 bit png graphics files. These files are displayed on the left-hand side of the Qucs main window when the "Components" tag is clicked and represent a simple outline of a 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.7 illustrates an enlarged view of simple icon picture for the RnonLin icon. In this picture the non-linear resistive equation given on the model schematic symbol 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. with the same name as the model name in file $RnonLin.cpp$. The Gimp Image Editor and 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 compo-

[4]Please note the required changes are highlighted in different colours for demonstration purposes; normally the C++ files generated by the ADMS Verilog-A synthesizer/compiler contain only text with a black attribute.

[5]Each time a new model is constructed the initial values for tx and ty, highlighted in green, 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. RnonLin.cpp C++ code with items for change indicated in red, green and blue

Qucs Editor 0.0.16 - File: /home/mike/.qucs/QucsEDDVerilogA_prj/RnonLin.dat

Line: 1 - Column: 1

```
// symbol drawing code
Lines.append (new Line (-80, 0, -70, 0, QPen (QColor ("#000080"), 2, Qt::SolidLine)));
Lines.append (new Line (-70, -20, 150, -20, QPen (QColor ("#000080"), 2, Qt::SolidLine)));
Lines.append (new Line (-70, 20, 150, 20, QPen (QColor ("#000080"), 2, Qt::SolidLine)));
Lines.append (new Line (-70, -20, -70, 20, QPen (QColor ("#000080"), 2, Qt::SolidLine)));
Lines.append (new Line (150, 0, 160, 0, QPen (QColor ("#000080"), 2, Qt::SolidLine)));
Lines.append (new Line (150, -20, 150, 20, QPen (QColor ("#000080"), 2, Qt::SolidLine)));
Texts.append (new Text (-40, -10, "R=R0 \x00B7(1 + A \x00B7 V_{R} + B \x00B7 V_{R} \x00B7 V_{R}", QColor ("#aa0000"), 8, 1, 0));

// terminal definitions
Ports.append (new Port (-80, 0)); /* Pin */
Ports.append (new Port (160, 0)); /* Pout */

// symbol boundings
x1 = -80;   y1 = -20;
x2 = 160;   y2 = 20;

// property text position
tx = -10; ty = 24;
```

Fig. 5. Qucs generated schematic capture C++ symbol code for the RnonLin non-linear resistor example

nent pictures list in file $Makefile.am$ located in directory **/tmp/qucs/qucs/bitmaps**. Add the file name $RnonLin.png$ to the end of list "XPMS = ..." and save file $Makefile.am$.

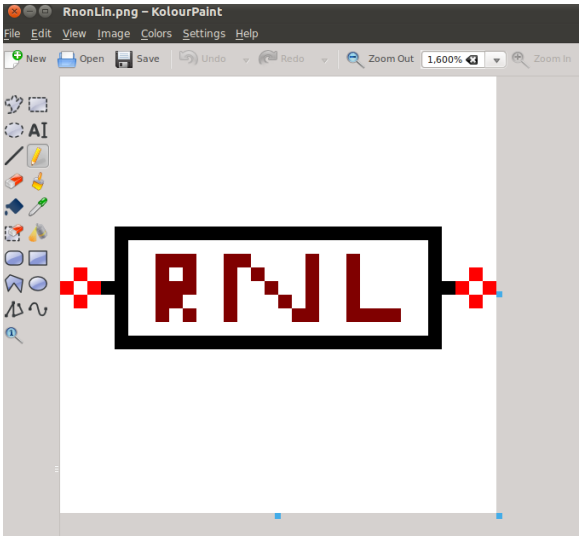

Fig. 7. Enlarged picture of a 32 bit by 32 bit RnonLin icon picture

VIII. Construction of a Qucs Verilog-A Model for a Voltage Controlled Non-linear Resistor: Part 5; Registering the RnonLin Verilog-A Model with the Qucs-core C++ code

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 the instructions for merging the new model code with the qucs-core code is presented. The next section continues the same theme and introduces the procedure for registering the new model with the GUI qucs C++ code. Fig. 8 gives details of the RnonLin entries that have to be made to file $Makefile.am$ in directory **/tmp/qucs-core/src/components/verilog**. In Fig. 8 the additions are highlighted in red. 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 name of the new model to two additional files: (1) 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"
```

(2) change the working directory to **/tmp/qucs-core/src** and open file $module.cpp$ for editing with a text editor. Add a $REGISTER\_CIRCUIT$ statement 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);
REGISTER_CIRCUIT (tff_SR);
```

Note that the names of the Verilog-A models on either side of the RnonLin model entry are most likely to vary from the 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 they are saved in their respective directories.

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

IX. Construction of a Qucs Verilog-A Model for a Voltage Controlled Non-linear Resistor: Part 5; Registering the RnonLin Verilog-A Model with the Qucs C++ Code

Change the current directory to directory **/tmp/qucs/qucs/components** and open file $Makefile.am$ for editing with a text editor. Add the red highlighted text to the $Makefile.am$ as indicated below in the short segment of C++ code:

libcomponents_a_SOURCES= .....
:
LPF.cpp HPF.cpp swcapBLInt.cpp RonLin.cpp
noinst_HEADERS= .....
:
LPF.h HPF.h swcapBLInt.h RonLin.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: (1) 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"
```

(2) change the working directory to **/tmp/qucs/qucs** and open file $module.cpp$ for editing with a text editor. Add a $REGISTER\_VERILOG$ statement for the RnonLin model as indicated in the following code section:

```
REGISTER_VERILOGA_1 (photodiode);
REGISTER_VERILOGA_1 (phototransistor);
REGISTER_VERILOGA_1 (nigbt);
REGISTER_VERILOGA_1 (RnonLin);
```

Note new Verilog-A models are normally added at the end of the $module.cpp$ Verilog-A model register list. After editing files $components.h$ and $module.cpp$ make sure they are saved in their respective directories.

X. Construction of a Qucs Verilog-A Model for a Voltage Controlled Non-linear Resistor: Part 6; Finishing Model Construction and Testing

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 circuit simulator correctly compilation of the modified C++ code should take place without error. However, if the C++ compiler reports one or more compilation errors check the code section where the error is reported to have occurred and make the necessary changes to correct it. Finally test the new Verilog-A model correctly operates in different simulation domains. Running the same test circuit as that shown in Fig.2 indicates that the Verilog-A model functions correctly with a run time similar to the EDD non-linear resistance model.[6]

References

[1] L. Lemaitre, W. Grabinski and C. McAndrew, "Compact device modeling using Verilog-A and ADMS", Electron Technology Internet Journal. vol 35, pp. 1-5, 2003.

[2] D. Fitzpatrick and I. Miller, "Analog behavioral modeling with the Verilog-A language", Kluwer Academic Publishers, Boston/Dordrecht/London,1998.

[3] K. Kundert and O. Zinke, "The Designer's Guide to Verilog AMS", Kluwer Academic Publishers, Boston/Dordrecht/London,2004.

[4] Accellera, "Verilog-AMS Language Reference Manual, Version 2.2", 2004, http://www.accellera.org [accessed April 2011].

[5] S. Jahn and H. Parruitte, "Qucs: A Description; Verilog-AMS interface", 2006, http://qucs.sourceforge.net/docs/Verilog.pdf, [accessed April 2011].

[6] M.E. Brinson and S.Jahn, "Building device models and circuit macro-models with the Qucs GPL circuit simulator", MOS-AK meeting, Frankfurt/Oder, Germany, 2009, http://www.mos-ak.org/frankfurt_o/, [accessed April 2011].

[7] S. Maruthamuthu, "Qucs based Verilog-A small signal RF model of a CNFET", MOS-AK meeting, Paris, France, 2011, http://www.mos-ak.org/paris/posters/P09_Maruthamuthu_MOS-AK_Paris.pdf.

[8] M.E. Brinson, S. Jahn and H. Nabijou, "Qucs, SPICE and Modelica equation-defined modelling techniques for the construction of compact device models based on a common model template structure", MOS-AK meeting, Paris, France, 2011, http://mos-ak.org/paris/papers/P06_Brinson_MOS-AK_Paris.pdf.

[6]This is not surprising as the RnonLin model only contains a small number of floating point calculations, implying that the simulation run time is dominated by the model call overhead.