Compact Device Modeling and Simulation with Qucs/Qucs-S/Xyce Modular Libraries

Mike Brinson
Centre for Communications Technology
London Metropolitan University
UK
Email: mbrin72043@yahoo.co.uk

Felix Salfelder
School of Computing
University of Leeds
UK
Email: felix@salfelder.org

***Abstract*—The rapid development of new semiconductor materials and devices has highlighted the need for compact modeling and circuit simulation tools that can be easily adapted to accommodate emerging technologies. In most instances device modeling tools employ non-linear behavioural sources and Verilog-A modules for model prototype construction. This paper is concerned with the properties and application of modular user-defined/plugin library toolkit that combines the best features of behavioural source and Verilog-A modeling practice while encouraging user extensions. The toolkit has been implemented as a Qucs/Qucs-S/Xyce modular library that is loadable on demand. To demonstrate its capabilities and flexibility a series of compact device models are introduced and their simulated performance presented and evaluated**

***Keywords*—Circuit simulation, Qucs, Qucs-S, Xyce, compact device modeling, device and probe toolkits, simulation scripts, test benches, FOSS.**

I. Introduction

The continuous development of new semiconductor materials and devices has for some time been a catalyst driving improvements in compact modelling and circuit simulation. Increasing numbers, and indeed the diversity of emerging technology devices, implies that future modelling and simulation tools must allow users to select only those models applicable to specific design or simulation tasks. Previous and current generations of circuit simulators provide at least a minimal set of component and device models based on SPICE 2 [1] and 3 [2]. These are considered to be an industrial baseline standard for circuit design. Over time new models have been added or existing model features extended. The adoption of Verilog-A as a "hardware description language" has both simplified and standardised compact model construction [3]. However, adding significant numbers of new models to a circuit simulator does increase the overall size and complexity of the software, which in turn makes code maintenance more onerous than it need be. One solution to this trend is to release circuit simulators with libraries of predefined models that can be loaded by users as needed. Modern FOSS circuit simulators, for example Qucs [4] and Qucs-S/Xyce [5][6], implement schematic capture and post-simulation data processing features as front end and back software for a simulation engine. Schematic capture adds a possible compatibility issue, namely that individual device symbols need to be transferable between different circuit simulators. Hence, when constructing new compact device models it is important that their physical properties are defined in a universal hardware description language with, whenever possible, standardised drawing symbols. This paper introduces a toolkit library for constructing and simulating new device models from linear and non-linear building blocks, where their properties are defined by netlists and Verilog-A modules plus a unified set of schematic drawing symbols. The toolkit has been implemented with the Qucs-S/Xyce package as a user-defined library. It will also be available as a plugin for the next generation Qucs modular circuit simulator [7]. To illustrate the toolkit properties it's structure, modeling features, and applications are described. A number of Qucs-S/Xyce compact device models plus simulation test benches are also included, and their performance discussed.

II. Background to "As to Why" Modular

The original Qucs project has been struggling to meet the moving requirements of circuit simulation needs. Early releases of Qucs have provided a well defined interface between the simulator engine (Qucsator [4]) and the user interface. Over time the user interface has been extended to drive other simulators, as well as optimisation tools for specific niche applications. The code quality has suffered extensively from those well meaning additions. Today, Qucs is considered an orphaned project, yet some forks exist that hold up partial functionality, on platforms that still provide a version of Qt4 [8] that has reached end-of-life several years ago.

The Qucs user interface that features interactive circuit analysis integrated with a schematic editor has been unique at the time, and even to date similar approaches are only available as non-free software. Past developments around this user interface have made changes to the file formats, and added support for simulator backends incompatible with the intended internals. Early additions have found their way into the main line, later ideas have lead to a dispersal of features, independent copies of the project (Qucs-S, CanEDA [10]), or unmaintained patches. Independently, the Gnucsator project [11] provides an alternative simulator with a different set of strengths and weaknesses as a drop-in-replacement for Qucsator.

Aiming at more stability and new features, attempts were made to refactor Qucs, even before the transition to Qt5 [8] became imminent. The high unorganised complexity of the code made it very difficult to retain global functionality while cleaning up locally. Combining the explorative work towards Qt5 with partially refactored subroutines as well as fresh non-trivial ideas from the Gnucap project [11] lead to what is to become the ”modular Qucs” library. As of today, a small and shrinking neutral library provides only the essential data structures and infrastructure for an interactive circuit modelling user interface. Making use of this library, other features found in traditional Qucs, or especially in Qucs-S, will be added independently and optionally.

Such additions will include the support for file formats (i. e. schematics, netlists, component models as well as simulation results or raw data), the support for component libraries, including modelling languages as well as drivers for simulation engines and algorithms. Naturally, the user interface will be extensible in a similar way, with relevant units provides as optional extensions. While our approach can be considered a well informed port to Qt5, it recombines existing ideas to address particular issues with the current legacy implementation. We believe that it is worthwhile to spend extra time on this end in order to achieve more in the long run.

a) Code quality aspects: Making use of C++ language features, especially the restriction of visibility of attributes will make the code more human readable. A test suite with both unit tests and end-to-end tests and rigorous coverage tracking will help keeping the library stable. Modularisation requires strict decoupling of the functionality and highlights the interfaces, hence modules encapsulate implementation details yet expose and document the program structure.

b) Tailoring and Experimentation: Since most parts of modular Qucs will be distributed as independent modules, alternative approaches will remain feasible. This includes but is not limited to data exchange with existing EDA software and projects. For example, integrating Qucs into an existing workflow can build a bridge into free software.

c) Versioning and compatibility: Different revisions of data used in traditional Qucs, and similarly in any other software involved, may require version specific customisation. Incompatibilities such as variations of component library versions or modelling dialects can be addressed through alternative plugins. Special needs can be satisfied easily and do not have to be carried along with the main project.

d) Decentralisation and Maintenance: Additional features that do not require changes to the main project, can be explored and shared across users. There is no longer a requirement for a review process or to wait for developers to respond. In the past, unfinished contributions have been held back, while the monolithic code base has moved on in other directions.

III. Overview of the Emerging Architecture Stressing Plugins for Legacy Qucs and Xyce

With legacy Qucs divided into a library and optional plugins, new possibilities will arise in applications based on Qucs. One of the motivations behind Qucs-S is to run SPICE simulators. Any two SPICE implementations are slightly different, and hence plugins supporting one will pave the way to add any other in the long run. In this work, we focus on use cases that make use of Xyce as a simulation engine. These extensions will be implemented in turns, either derived from existing plugins, ported from Qucs-S or implemented from scratch. The block diagram diagram drawn in Figure 1 outlines how the modular approach to Qucs will facilitate the integration of those features. Running Xyce from modular Qucs will initially

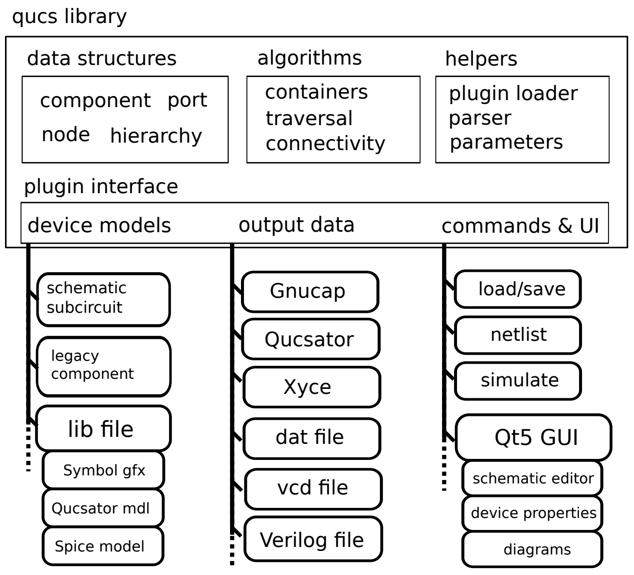


Fig. 1. Modular Qucs, structural view. A slim library provides circuit modeling essentials and infrastructure for extensions. Replaceable plugins (in rectangles with rounded corners) add further functionality that may be application specific and usually involves choices.

require a set of symbols for which the simulator implements models. On this end, modular Qucs provides a generic base class for circuit components and a modular objective parser for various of schematics, netlists and device model file formats. As of now, this is only implemented for traditional Qucs schematics, component libraries (”.lib”) and also, structural Verilog. Particularly, ”.lib” file contents are hidden behind a component interface, and the interpretation is left to plugins. To support .lib files, modular Qucs allows for an arbitrary set of sections within a component symbol (this is sometimes referred to as ”multi view”). Modular Qucs loads traditional .lib files, which are very similar in structure to Qucs-S .lib files.

Next, a plugin for Xyce netlisting will eject the netlist in a suitable Spice format. Models from the appropriate sections in the symbols are accessible by the netlister, and need to be selected according to the target language. This is the same procedure as in the already existing netlister targetting

Qucsator based simulators. Unlike in traditional Qucs, the current modular library manages nets and component models explicitly, reducing the responsibilities of the netlister to traversing the circuit model and translating the parts.

Running Xyce the way Qucs-S does involves invoking a child process. Modular Qucs provides an interface for pluggable simulator drivers. Simulator drivers are designed to meet the requirements for future applications. For example, some simulators provide a shared library interface (c.f. kicad–ngspice), or require persistency to provide interactive elements. To match the Qucs-S experience, running a subprocess is a possibility, and a trivially modified Qucsator driver will run Xyce. Retrieving and processing data from a simulator is currently under construction. As a stop-gap ".dat" files can be used, as they were used in traditional Qucs and Qucs-S. Modular Qucs does not dictate the data types used or data path provided by plugins. Native support for Spice raw, and spreadsheet std formats, as used by Xyce and similar simulators are intended.

IV. Modular User-defined and Plugin Model Libraries

Conventional circuit simulators embed component type, connections and attributes within their netlist statements, while simultaneously allowing access to libraries of device models and subcircuits. A second approach is to define components and compact device models within modular user-defined or plugin libraries. These can be loaded when a circuit simulator is run and have the important advantage that users need only select those libraries that are required for a specific circuit design or device simulation. The proposed modular library extends the original SPICE capabilities to include linear components, semiconductor devices, SPICE style subcircuits, Verilog-A modules, non-linear blocks, and signal probes. A selection of typical library parts are shown in Fig. 2. With the exception of simulation control icons, for example Xyce script, INCLUDE SCRIPT and some other universal symbols like the earth symbol and schematic node names, all the other items are part of a user-defined library, where individual items are held as Qucs-S/Xyce subcircuits. In this context subcircuits are particularly important in that they isolate a model symbol from the internal netlist statements and allow parameters to be passed to a model. Hence, if the Xyce circuit simulator is changed to an alternative simulation engine the toolkit is likely to only require minimal adjustments to the library subcircuit netlists.

V. Introduction to Qucs-S/Xyce Compact Modeling Blocks

Qucs-S/Xyce user-defined libraries have subcircuits as the fundamental building block in their modular library structure. This was chosen because individual subcircuits have a unique drawing symbol and allow numeric data and algebraic equations to be passed as parameters. Passing algebraic equations is particularly important because it allows complex modeling blocks like EDD [15], e_value and e_table shown in Figure

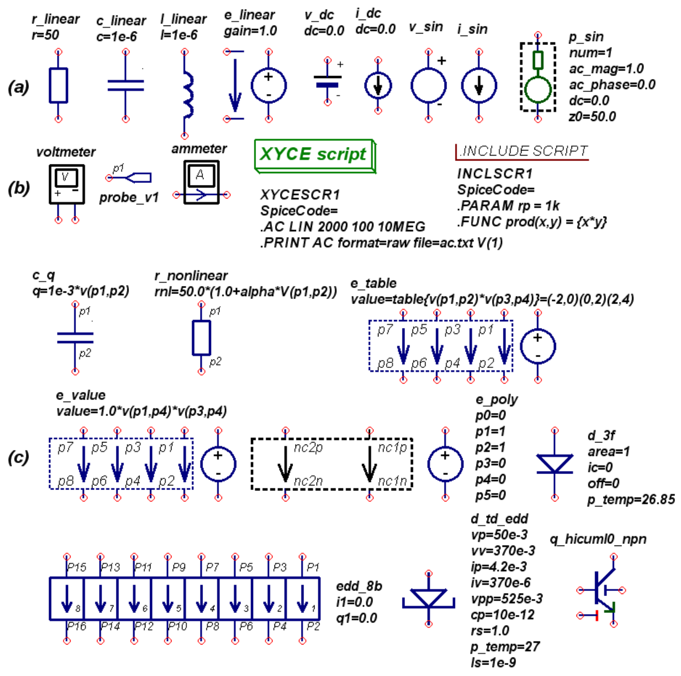
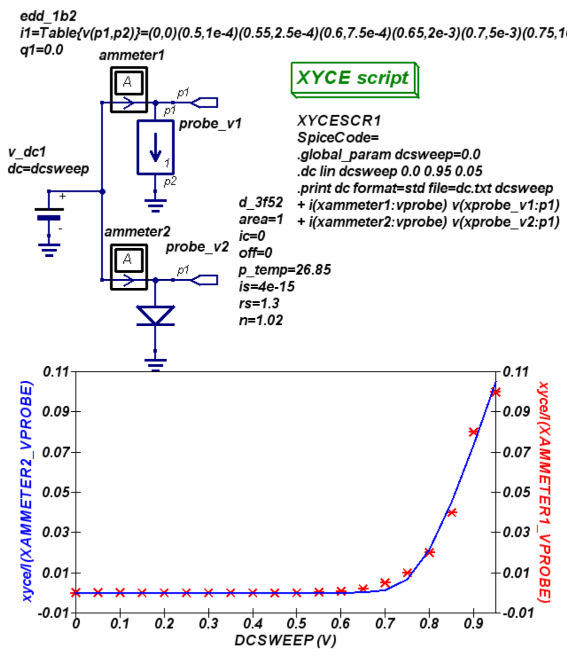


Fig. 2. Example Qucs-S/Xyce toolkit symbols: (a) fundamental components and signal sources, (b) measuring instruments and simulation control icons, and (c) non-linear components, modeling blocks, and compact device models.

2 to function as universal elements where internal equations can be modified during the simulation initialisation process. As a general rule those device symbols with interface nodes labelled with px or $ncxx$ allow parameters defined by algebraic equations or numeric data. The test bench in Figure 3 presents an example that illustrates how algebraic equations are passed to an EDD. In Figure 3 the lower circuit branch shows a diode represented by a SPICE diode model called d_3f52. The upper branch models the same diode with a single EDD where current $i1$ is passed as an algebraic equation in the form of a table controlled by voltage $v(p1, p2)$. During the initial stages of the simulation process Xyce expands all subcircuits to form an overall flattened netlist where nodes $p1$ and $p2$ are replaced by their names in a Xyce X subcircuit call statement. The ability to express subroutine parameters as algebraic equations or named `.global_params` adds a further extension to Qucs-S/Xyce simulation capabilities. Illustrated in Figure 4 is the diode d_3f52 with diode parameter rs set to named variable res where res is a `.global_param` that is changed by the `.step` statement in the XYCE script. Xyce `.global_param` items are a particularly interesting innovation in that they can change value during simulation, acting essentially as named variables rather than numeric constants. In a similar fashion to res independent voltage source v_dc1 has named value $dcsweep$ which is changed by Xyce statement `.dc`. The two step process produces a set of simulation data that shows how varying the diode series resistance rs effects the Id/Vd plotted characteristics at high Id currents.



```
edd_1b2
i1=Table{v(p1,p2)}=(0,0)(0.5,1e-4)(0.55,2.5e-4)(0.6,7.5e-4)(0.65,2e-3)(0.7,5e-3)(0.75,1
q1=0.0
```

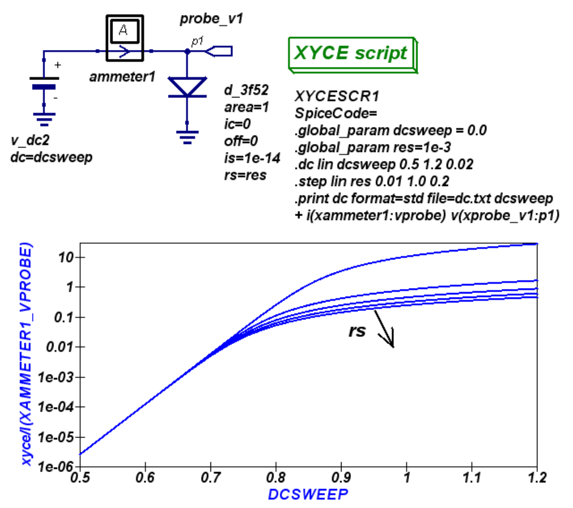
```
XYCESCR1
SpiceCode=
.global_param dcsweep=0.0
.dc lin dcsweep 0.0 0.95 0.05
.print dc format=std file=dc.txt dcsweep
+ i(xammeter1:vprobe) v(xprobe_v1:p1)
+ i(xammeter2:vprobe) v(xprobe_v2:p1)
```

```
d_3f52
area=1
ic=0
off=0
p_temp=26.85
is=4e-15
rs=1.3
n=1.02
```

Fig. 3. Example diode test bench and simulation results: solid line diode d_3f52 (left side vertical scale) , crosses EDD_1b2 tabular data (right side vertical scale). The $Table$ statement is truncated on the right side to fit column width.

VI. Behavioural Compact Device Modeling

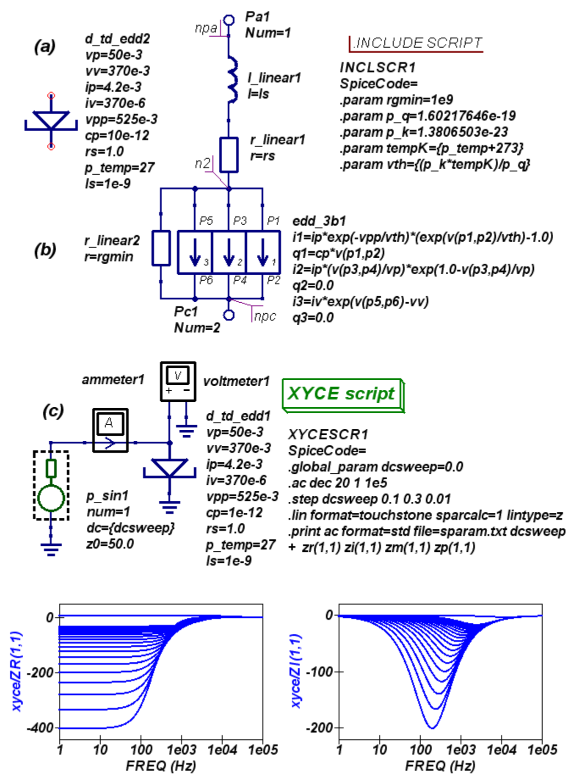
Behavioural compact device modeling based on an extended version of the SPICE non-linear `B` independent source and non-linear capacitance is a central feature of the Xyce compact device modeling. The Qucs-S/Xyce user-defined library toolkit adds an extended version of the Qucs EDD to the Xyce modeling repertoire. Figure 5 illustrates an application of Qucs-S/Xyce compact device behavioural modeling. In this example a tunnel diode model is constructed from the toolkit EDD and other components, where Figure 5 (a) illustrates the toolkit tunnel diode schematic symbol and (b) the Qucs-S subcircuit schematic and Xyce SPICE statements. The edd_3b1 in and qn parameters are algebraic equations written in terms of internal nodes pn. The circuit test bench is drawn in Figure 5 (c) with simulation output data plots for real and imaginary impedance components $ZR(1,1)$ and $ZI(1,1)$. These clearly demonstrate the power of the new toolkit. Here a combination of Xyce simulation statements that step `.global_param` $dcsweep$ values and undertake S-parameter analysis (Xyce `.lin` statement), as part of a small signal a.c. simulation, determine tunnel diode impedance under differing .d.c bias conditions. In the $dcsweep$ range 0.1V to 0.3V the tunnel diode has a negative real component of impedance $Z(1,1)$. At values of $dcsweep$ above 0.3V $ZR(1,1)$ becomes positive.



```
XYCESCR1
SpiceCode=
.global_param dcsweep = 0.0
.global_param res=1e-3
.dc lin dcsweep 0.5 1.2 0.02
.step lin res 0.01 1.0 0.2
.print dc format=std file=dc.txt dcsweep
+ i(xammeter1:vprobe) v(xprobe_v1:p1)
```

```
d_3f52
area=1
ic=0
off=0
is=1e-14
rs=res
```

Fig. 4. A diode test bench illustrating combined non-linear DC simulation and parameter sweep using Xyce global parameters $dcsweep$ and res respectively.



```
d_td_edd2
vp=50e-3
vv=370e-3
ip=4.2e-3
iv=370e-6
vpp=525e-3
cp=10e-12
rs=1.0
p_temp=27
ls=1e-9
```

```
INCLSCR1
SpiceCode=
.param rgmin=1e9
.param p_q=1.60217646e-19
.param p_k=1.3806503e-23
.param tempK={p_temp+273}
.param vth={(p_k*tempK)/p_q}
```

```
edd_3b1
i1=ip*exp(-vpp/vth)*(exp(v(p1,p2)/vth)-1.0)
q1=cp*v(p1,p2)
i2=ip*(v(p3,p4)/vp)*exp(1.0-v(p3,p4)/vp)
q2=0.0
i3=iv*exp(v(p5,p6)-vv)
q3=0.0
```

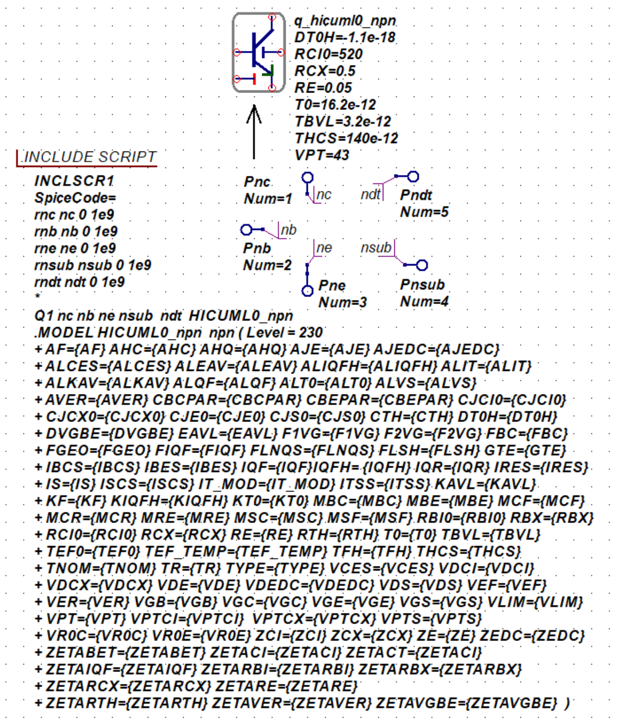
```
d_td_edd1
vp=50e-3
vv=370e-3
ip=4.2e-3
iv=370e-6
vpp=525e-3
cp=1e-12
rs=1.0
p_temp=27
ls=1e-9
```

```
XYCESCR1
SpiceCode=
.global_param dcsweep=0.0
.ac dec 20 1 1e5
.step dcsweep 0.1 0.3 0.01
.lin format=touchstone sparcalc=1 lintype=z
.print ac format=std file=sparam.txt dcsweep
+  zr(1,1) zi(1,1) zm(1,1) zp(1,1)
```

Fig. 5. A Qucs-S/Xyce tunnel diode behavioural model: (a) schematic symbol, (b) subcircuit schematic ans SPICE statements and (c) a test bench for S parameter simulation and extraction of $Zr(1,1)$ and $Zi(1,1)$ impedance.

VII. Verilog-A Compact Device Modeling

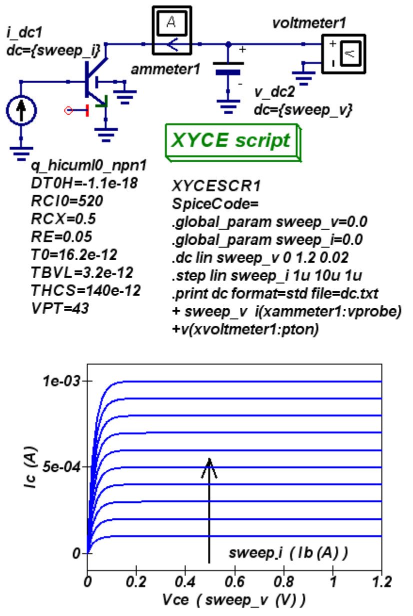
The Xyce simulator uses the Automatic Device Model Synthesiser (ADMS) [12] in a second approach to compact device modeling, allowing the package to compile and run a range of industrial level and experimental C++ code models. Xyce comes pre-packaged with an extended version of ADMS, including routines and detailed instructions for compiling and linking Verilog-A modules [13]. Both static and dynamic C++ code are supported. There is a strong link between Qucs/Qucs-S/Xyce behavioural models and their equivalent Verilog-A modules [16]. One of the most important advantages gained from combining Qucs/Qucs-S with Xyce is that the resulting circuit simulator package provides open access to the Compact Model Coalition (CMC) standardised SPICE models [14], examples being the diode-CMC, HICUM and MEXTRAM BJT, and the BSIM6 MOSFET models. The Qucs/Qucs-S/Xyce user-defined/plugin library toolkit provides a practical link between Xyce CMC models and the Qucs/Qucs-S graphical user interface. Shown in Figure 6 is the toolkit symbol and the Xyce subcircuit SPICE statements for the CMC $HICUML0\_npn$ BJT model. Figure 6 illustrates how a subcircuit is used to encapsulate SPICE code and pass subcircuit parameters via SPICE `Q` and `.model` statements. The $HICUML0\_npn$ subcircuit code also demonstrate an



```
INCLSCR1
SpiceCode=
rnc nc 0 1e9
rnb nb 0 1e9
rne ne 0 1e9
rnsub nsub 0 1e9
rndt ndt 0 1e9
*
Q1 nc nb ne nsub  ndt  HICUML0_npn
.MODEL HICUML0_npn  npn ( Level = 230
+ AF={AF} AHC={AHC} AHQ={AHQ} AJE={AJE} AJEDC={AJEDC}
+ ALCES={ALCES} ALEAV={ALEAV} ALIQFH={ALIQFH} ALIT={ALIT}
+ ALKAV={ALKAV} ALQF={ALQF} ALT0={ALT0} ALVS={ALVS}
+ AVER={AVER} CBCPAR={CBCPAR} CBEPAR={CBEPAR} CJCI0={CJCI0}
+ CJCX0={CJCX0} CJE0={CJE0} CJS0={CJS0} CTH={CTH} DT0H={DT0H}
+ DVGBE={DVGBE} EAVL={EAVL} F1VG={F1VG} F2VG={F2VG} FBC={FBC}
+ FGEO={FGEO} FIQF={FIQF} FLNQS={FLNQS} FLSH={FLSH} GTE={GTE}
+ IBCS={IBCS} IBES={IBES} IQF={IQF}IQFH= {IQFH} IQR={IQR} IRES={IRES}
+ IS={IS} ISCS={ISCS} IT_MOD={IT_MOD} ITSS={ITSS} KAVL={KAVL}
+ KF={KF} KIQFH={KIQFH} KT0={KT0} MBC={MBC} MBE={MBE} MCF={MCF}
+ MCR={MCR} MRE={MRE} MSC={MSC} MSF={MSF} RBI0={RBI0} RBX={RBX}
+ RCI0={RCI0} RCX={RCX} RE={RE} RTH={RTH} T0={T0} TBVL={TBVL}
+ TEF0={TEF0} TEF_TEMP={TEF_TEMP} TFH={TFH} THCS={THCS}
+ TNOM={TNOM} TR={TR} TYPE={TYPE} VCES={VCES} VDCI={VDCI}
+ VDCX={VDCX} VDE={VDE} VDEDC={VDEDC} VDS={VDS} VEF={VEF}
+ VER={VER} VGB={VGB} VGC={VGC} VGE={VGE} VGS={VGS} VLIM={VLIM}
+ VPT={VPT} VPTCI={VPTCI}  VPTCX={VPTCX} VPTS={VPTS}
+ VR0C={VR0C} VR0E={VR0E} ZCI={ZCI} ZCX={ZCX} ZE={ZE} ZEDC={ZEDC}
+ ZETABET={ZETABET} ZETACI={ZETACI} ZETACT={ZETACI}
+ ZETAIQF={ZETAIQF} ZETARBI={ZETARBI} ZETARBX={ZETARBX}
+ ZETARCX={ZETARCX} ZETARE={ZETARE}
+ ZETARTH={ZETARTH} ZETAVER={ZETAVER} ZETAVGBE={ZETAVGBE}  )
```

Fig. 6. The $HICUMLO\_npn$ toolkit symbol and Xyce subcircuit SPICE statements: the Xyce statements listed by the `.INCLUDE SCRIPT` form the body of the $HICUML0\_npn$ subcircuit symbol. $HICUML0\_npn$ `.model` parameters are set to Xyce default values unless reassigned at the subcircuit symbol level.

efficient way to represent compact device models that are specified by a large number of numerical parameters. In the case of $HICUML0\_npn$ there are nearly 100 parameters. Moreover with some other CMC models, like for example the BSIM6 MOSFET, there are approaching 1000 parameters. The toolkit CMC model parameters are set to the default values provided with Xyce, making it a relatively simple task to modify those parameters who's values need changing to support a specific semiconductor technology. The Qucs-S/Xyce combination provides features for both single and double parameter sweep simulation. Figure 7 illustrates a d.c. test bench for generating, and plotting, the output characteristics of the CMC $HICUML0$ npn model. In this example test bench parameters $sweep\_v$ and $sweep\_i$ are defined by Xyce `.global_params` with their values controlled by `.dc` (Vce) and `.step` (Ib) statements respectively.



```
XYCESCR1
SpiceCode=
.global_param sweep_v=0.0
.global_param sweep_i=0.0
.dc lin sweep_v 0 1.2 0.02
.step lin sweep_i 1u 10u 1u
.print dc format=std file=dc.txt
+ sweep_v  i(xammeter1:vprobe)
+v(xvoltmeter1:pton)
```

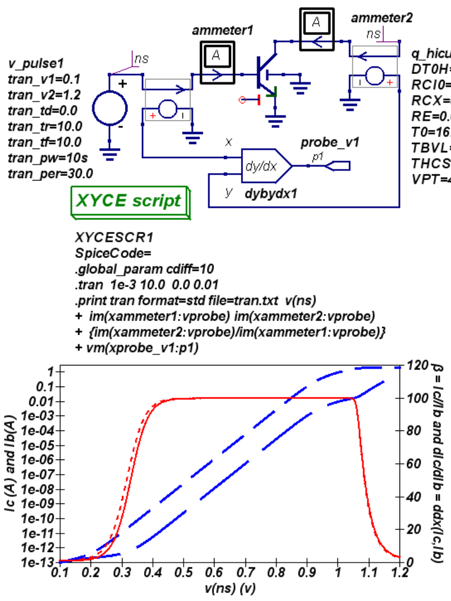
Fig. 7. A Qucs-S/Xyce test bench for simulating and displaying BJT Ic/Vce output characteristics with $1\mu\text{A} \leq Ib \leq 10\mu\text{A}$ in $1\mu\text{A}$ steps.

VIII. Adding New Diagnostic Probes

A consistent feature of circuit simulator software is the inclusion of a probe system for specifying which simulation output data are to be collected and stored in preparation for backend data post-processing and visualisation. The Qucs-S Xyce script uses SPICE `.print` statements for this purpose, see Figures 4, 6, 5 and 7. Furthermore, the original SPICE $.print$ statement has been extended by Xyce to allow algebraic equations enclosed in $\{...\}$ brackets. Figure 9 gives an example where the d.c. gain, $\beta = Ic/Ib$, is expressed as the ratio of ammeter currents $im(xammeter2 : vprobe)$ and $im(xammeter1 : vprobe)$. Data probes are an integral part of the Qucs/Qucs-S/Xyce user-defined/plug-in toolkit libraries. This has a distinct advantage because it encourages the development of special purpose probes for specific design/simulation tasks. The BJT test bench drawn in Figure 8 illustrates the use of a behavioural probe who's function is equivalent to Verilog-A `ddx()` [3]. In this example a ten second duration linear ramp varies the voltage at input node ns from 0.1V to 1.2V. Ammeters 1 and 2 measure the slowly changing Ib and Ic currents. Similarly, the $dybydx$ probe senses changes in the BJT currents and outputs the differential gain ratio dIc/dIb. Figure 9 gives details of the dIc/dIb probe subcircuit. The $dybydx$ probe input signals are $v(y)$ at node ny and $v(x)$ at node nx where, from Figure 8, $v(y) = Ic$ and $v(x) = Ib$. Hence

$$\begin{aligned} \frac{dy}{dx} &= \frac{dy/dt}{dx/dt} = \frac{dIc/dt}{dIb/dt} \\ &= \frac{\alpha}{if((\gamma \leq 1e-50), 1e-50, \delta)} \end{aligned} \qquad (1)$$

where $\alpha = Ic/cdiff$, $\gamma = \mid Ib/cdiff \mid$, $\delta = (Ib/cdiff) + 1e-50$. The $if(....)$ statement and the 1e-50 numerical constant are included in Equation 1 to minimize the effects of the discontinuity at $\gamma = 0.0$. Over the voltage range 0.3V $\leq$ 1.2V the d.c. and differential values of β plotted in Figure 8 were found to be in good agreement. However, at very low values of Ib there are apparent differences in the d.c. and differential β curves. Similarly, at $Vbe \geq$ 1.0V the simulated Ib values suggest the onset of strong injection effects.

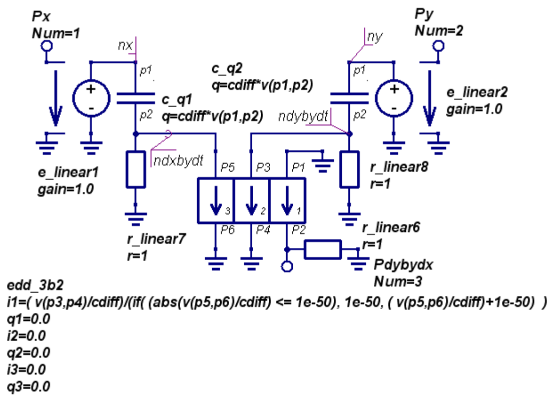


```
XYCESCR1
SpiceCode=
.global_param cdiff=10
.tran  1e-3 10.0  0.0 0.01
.print tran format=std file=tran.txt  v(ns)
+  im(xammeter1:vprobe) im(xammeter2:vprobe)
+  {im(xammeter2:vprobe)/im(xammeter1:vprobe)}
+ vm(xprobe_v1:p1)
```

Fig. 8. A time domain test bench for determining BJT current gain with ammeters and a dy/dx behavioural probe. Left scale: Ic (dash line) and Ib (long dash line). Right scale: current gain β d.c. data (solid line) and dy/dx data (dot line).

IX. Conclusions

The fast pace of semiconductor material and device developments has placed traditional circuit simulators under considerable pressure to keep in step with user needs. One approach to providing future circuit simulation tools with both the required analysis and modelling capabilities is to adopt a software structure that allows a high degree of flexibility through user plugins/libraries. The current generation of circuit simulation and device modeling tools largely employ nonlinear behavioural sources and Verilog-A modules for model prototype construction. This paper reports the properties and application of a modular user-defined/plugin library toolkit that combines the best features of behavioural source and Verilog-A modeling practice while encouraging user extensions. The toolkit has been implemented as a Qucs/Qucs-S/Xyce modular library that is loadable on demand and easily extended to meet the needs of specific circuit design projects. To demonstrate its capabilities and flexibility a series of compact device models are introduced in the text and their simulated performance presented and evaluated.



```
edd_3b2
i1=( v(p3,p4)/cdiff)/(if( (abs(v(p5,p6)/cdiff) <= 1e-50), 1e-50, ( v(p5,p6)/cdiff)+1e-50)  )
q1=0.0
i2=0.0
q2=0.0
i3=0.0
q3=0.0
```

Fig. 9. A behavioural modelling ddx() probe constructed with Qucs/Qucs-S/Xyce nonlinear capacitors and EDD toolkit models: voltages at node $ndxbydt = dx/dt$, and node $ndybydt = dy/dt$; similarly the output voltage at model pin $Pdybydx = dy/dx$.

References

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

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

[3] Accellera, "Verilog-AMS Language Reference Manual, version 2.2", available from: http://www.accellera.org, [Accessed April 2021].

[4] Sourceforge, Qucs project: Quite Universal Circuit Simulator, Available qucs.sourceforge.net. [accessed April 2021].

[5] V. Kusnetsov and M. Brinson, "Qucs-S: Qucs with SPICE". Version 0.0.22, https://ra3xdh.github.io/, 2020. [Accessed April 2021].

[6] Sandia National Laboratories, "Xyce Parallel electronic simulator: version 6.8", 2015, https://xyce.sandia.gov/ [Accessed April 2021].

[7] F. Salfelder and M. Brinson, "A modular approach to next generation Qucs", 1st Asia/South Pacific MOS-AK Workshop, (virtual/online) Feb. 25-26, 2021.

[8] Digia "Qt a C++ toolkit for cross-platform application development", available from: http://www.digia.com. [Accessed April 2021].

[9] Sourceforge, CanEDA project: CanEDA (Circuits and networks EDA), available from https://sourceforge.net/projects/caneda/. [Accessed April 2021].

[10] F. Salfelder, Gnucsator - "a Gnucap based simulation kernel for Qucs", available from: https://github.com/Qucs/gnucsator/. [Accessed April 2021].

[11] A. Davis, "Gnucap - the Gnu Circuit Analysis Package", available from: https://www.gnu.org/software/gnucap/. [Accessed April 2021].

[12] 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.

[13] Sandia National Laboratories," Xyce/ADMS Users' Guide", available from: https://xyce.sandia.gov/documentation/XyceADMSGuide.html. [Accessed April 2021].

[14] Si2, "Compact Model Coalition (CMC) Standard Models", available from: https://si2.org/standard-models/. [Accessed April 2021].

[15] S. Jahn and M.E. Brinson, "Interactive compact device modelling using Qucs equation-defined devices", International Journal of Numerical Modelling: Devices and Fields, pp. 335-349, 2008. DOI:10.1002/jnm.676.

[16] M.E. Brinson and V. Kuznetsov, "A new approach to compact semiconductor device modelling with Qucs Verilog-A module synthesis", International Journal of Numerical Modelling: Devices and Fields, pp. 1-19. ISSN 1099-1204, 2016. DOI/10.1002/jnm.2166.