Mathematical and Computer Modelling of Dynamical Systems
Vol. 11, No. 1, March 2005, 43 – 60

Taylor & Francis
Taylor & Francis Group

Design of *SPICE*Lib: A Modelica library for modeling and analysis of electric circuits

A. URQUIA, C. MARTIN and S. DORMIDO

Department Informática y Automática, E.T.S. de Ingeniería Informática, U.N.E.D., Juan del Rosal 16, 28040 Madrid, Spain

*SPICE*Lib is an object-oriented model library, written in the Modelica language, that implements some of the modeling and analysis capabilities of the circuit simulator PSpice. A novel approach has been adopted in the *SPICE*Lib design. It arises from considering that the reasons behind the success of PSpice include: the quality of the device models, the variety of supported analyses and the good performance of the numerical simulation. As a consequence, *SPICE*Lib is conceived to mimic not only the PSpice device models, but in addition PSpice capability to perform a variety of circuit analyses and the PSpice algorithms to calculate the circuit bias point, which is the most problematic analysis from the numerical standpoint.

The fundamental hypotheses and the architecture of *SPICE*Lib library are discussed, in addition to the modeling of the supported analyses and devices. A case study is fully developed, in order to illustrate *SPICE*Lib use and validation. *SPICE*Lib version 1.1 (release October 2003) is free software, and it can be retrieved from the website: http://www.modelica.org/Conference2003/papers.shtml.

Keywords: Circuit simulation: Modelica; Object-oriented modelling languages; Hybrid models; Differential-algebraic equations.

1. Introduction

Modelica is a freely available, object-oriented modeling language, which has been designed by the developers of the object-oriented modeling languages Allan, Dymola, NMF, ObjectMath, Omola, SIDOPS + and Smile, and a number of modeling practitioners in different fields. The Modelica language is intended to serve as a standard format for the model external representation, so that models arising in different domains can be exchanged between tools and users [1].

As a general-purpose modeling language, Modelica has the important advantage of allowing the physical modeling of multi-domain systems, supporting several formalisms [2]: ordinary differential equations (ODE), differential-algebraic equations (DAE), bond graphs, finite state automata, Petri nets, etc.

Correspondence to: {aurquia,carla,sdormido}@dia.uned.es

Mathematical and Computer Modelling of Dynamical Systems
ISSN 1387-3954 print/ISSN 1744-5051 online
http://www.tandf.co.uk/journals
DOI: 10.1080/13873950500052488

However, one of the main obstacles to spread the use of Modelica throughout the industrial and academic environments was the lack of truly-reusable model libraries written in the Modelica modeling language. The recognition of this situation drives the strong interest within the Modelica modelers community in developing well-tested and well-documented libraries of reusable models in different application domains (see [1] for a relation of the available libraries).

The Modelica Standard Library [1] has an electrical sub-library, developed by Clauß et al. [3], which contains models for transient analysis of some simple analog components, such as independent and controlled sources, passive and semiconductor devices, and ideal elements. The electric circuits, built up connecting these analog device models, are capable of interacting with the components of other sub-libraries of the Modelica Standard Library.

The approach adopted by the *SPICE*Lib designers is different. It arises from considering that the reasons behind the success of PSpice [4], as an essential computer-aid for circuit design and analysis, include:

1. **The quality of the device models.** PSpice implements different levels of detail in the device description. For instance, OrCAD PSpice version 9. provides seven different models of the MOSFET device [4].
2. **The variety of supported analyses.** They include operating point (.OP), DC sweep (.DC), AC sweep (.AC), transient (.TRAN), Fourier (.FOUR), Monte Carlo (.MC) and noise (.NOISE) analyses [4].
3. **The good performance of the numerical simulation.** It is achieved by [4,5,6]: (1) imposing that all the device equations must be continuous; (2) re-designing those device models "problematic" from the numerical point of view, in order to facilitate their numerical solution; and (3) implementing different algorithms for the bias point calculation, which is the most problematic analysis from the numerical standpoint.

As a consequence, *SPICE*Lib is conceived to mimic not only the PSpice device models, but in addition its capability to perform a variety of circuit analyses and the PSpice algorithms to calculate the bias point of the circuit. *SPICE*Lib version 1.1 implements the following [7,8]:

- Device models, such as independent and controlled sources, passive devices (resistor, capacitor and inductor), and semiconductor devices (PSpice PN-junction diode and LEVEL1 n-channel and p-channel MOSFET). In addition, IC1 and IC2 pseudo-components have been implemented for setting the initial conditions.
- Three types of circuit analysis: operating point calculation (.OP), AC sweep (.AC) and transient (.TRAN).
- The three algorithms supported by PSpice for bias point calculation and, in addition, a fourth algorithm proposed by Cellier [9].

In addition, *SPICE*Lib is well suited to build the electric part of multi-domain physical models. The circuit models composed using *SPICE*Lib can be interfaced to models built using other Modelica libraries. However, in this case, only the transient analysis of the multi-domain model is supported.

The design and implementation of *SPICE*Lib is a long-term, academic project. Its main goal is to develop open-source models of the PSpice devices and analyses, so that

they can be freely reviewed, modified and extended. As *SPICE*Lib is completely written in the Modelica language, this greatly facilitates its code comprehension, maintenance and extension.

Four key points of the *SPICE*Lib design are discussed in this contribution: the fundamental modeling hypotheses, the library architecture, the modeling of the circuit analyses, and some relevant aspects of the device "atomic"-models (i.e., voltage and current sources, resistor, capacitor and inductor models). These atomic-models are of paramount importance: in addition to being used in the modeling of user-defined electric circuits, they are the only components of the *SPICE*Lib semiconductor-device models. The modeling of semiconductor devices is not addressed in this manuscript (it is discussed in [8]).

A complete case study, including circuit modeling, analysis and validation of the results is used to illustrate *SPICE*Lib use, and device and analysis modeling. *SPICE*Lib validation is accomplished by comparing the analysis results obtained with *SPICE*Lib and OrCAD PSpice version 9.1 [4]. Finally, the modeling of multi-domain system for transient analysis, by combined use of *SPICE*Lib and other Modelica libraries, is discussed.

Additional examples of circuit modeling, analysis and validation can be found in [7,8], and a tutorial description of *SPICE*Lib use in [8]. The *SPICE*Lib accompanying documentation contains a detailed description of model implementation. It can be downloaded, together with *SPICE*Lib 1.1 source code, from the website: http://www.modelica.org/Conference2003/papers.shtml.

2. Fundamental modeling hypotheses

To support the implementation of the OP, AC and TRAN circuit analyses, *SPICE*Lib device models contain three different device descriptions: static, AC small-signal and large-signal. Each of these three descriptions has its own set of equations and variables, and its own contribution to the device-model interface (the details can be found in [7,8]). As a consequence, when connecting *SPICE*Lib device models to compose a circuit model, these three descriptions of the circuit (i.e., static, AC small-signal and large-signal) are obtained.

While the on-going circuit analysis does not require the use of a circuit description, *SPICE*Lib sets its bias conditions to zero (i.e., the description is "disabled"). Therefore, the equations of this description can be trivially solved, and the simulation performance is not unnecessarily degraded. For instance, *SPICE*Lib disables the large-signal description of the circuit by setting to zero the value of the independent sources and the energy stored in capacitors and inductors.

*SPICE*Lib implementation of the OP, AC and TRAN analyses require the combined use of the three circuit descriptions:

- *SPICE*Lib uses the static description of the circuit to calculate its bias point at the beginning of the AC sweep analysis. The obtained voltage and current values allow calculating the parameter values of the AC small-signal (i.e., linearized at the operating point) description. Then, *SPICE*Lib disables the static formulation of the circuit, and it completes the AC sweep analysis using the AC small-signal description.
- *SPICE*Lib model of transient analysis constitutes a second example. *SPICE*Lib supports two different initialization conditions of the circuit for transient

analysis: performing the bias point calculation or skipping the bias point calculation. If the bias point needs to be calculated, then *SPICE*Lib obtains the operating point of the circuit static description. These calculated values are imposed to be the initial condition for transient analysis of the circuit large-signal description. *SPICE*Lib device models are designed to allow this transfer of information from the static to the large-signal description. Once the large-signal description of the circuit has been initialized, it is used to simulate the circuit transient behavior, and the static description is disabled.

- *SPICE*Lib implementation of Cellier's algorithm [9] for bias point calculation ("dynamic model ramping" in *SPICE*Lib terminology) is a third example. In this case, the initial condition to iterate the static description is calculated by simulating the large-signal description of the circuit. *SPICE*Lib implementation of this algorithm requires the combined use of the circuit large-signal and static descriptions.

*SPICE*Lib analysis models are formulated in terms of equations, and they are completely written in the Modelica language: the implementation of algorithms for bias point calculation, in addition to the OP, AC and TRAN analyses, do not include calls to external functions written in any programming language (C, Fortran, etc.).

Information is transmitted, from the analysis models to the device models, by variables common to both model types. These variables are called *control-signals* in *SPICE*Lib terminology, and they are modeled taking advantage of the Modelica capability to describe physical fields [10,11]: control-signals are *inner* variables of the analysis models and *outer* variables of the device models.

Analysis models describe the sequence of control-signal value transitions required to perform the analyses. Control-signals trigger instantaneous changes in the device model state-variables and in the mathematical structure of the device descriptions. These changes facilitate setting the initial conditions of the analyses, enabling and disabling the different circuit descriptions, and the information exchange among them. A list of *SPICE*Lib control-signals, and a description of their function, can be found in [7].

3. *SPICE*Lib architecture

The *SPICE*Lib sub-libraries ("packages" in Modelica terminology) have been organized in order to facilitate their use and maintenance. The package hierarchy is shown in Figure 1a (the modeling environment is Dymola [12]). The *SPICE*Lib architecture has a clear separation between those libraries to be used by *SPICE*Lib users and those libraries to be used only by *SPICE*Lib designers [8]:

- The *SPICE*Lib models describing the structure and behavior of parts and analyses are gathered in the *SPICE*Lib.*src* package (see Figure 1a). This sub-library is intended to be used and modified only by *SPICE*Lib designers. Its documentation is oriented to the designers, explaining implementation details which are not of interest to *SPICE*Lib users.
- Two sub-libraries are defined, *SPICE*Lib.*parts* and *SPICE*Lib.*analyses* (see Figure 1a), in order to gather the models that *SPICE*Lib users need to compose and analyze their circuits. The models of these two sub-libraries are sub-classes of a reduced set of *SPICE*Lib.*src* library models, inheriting the structure and the

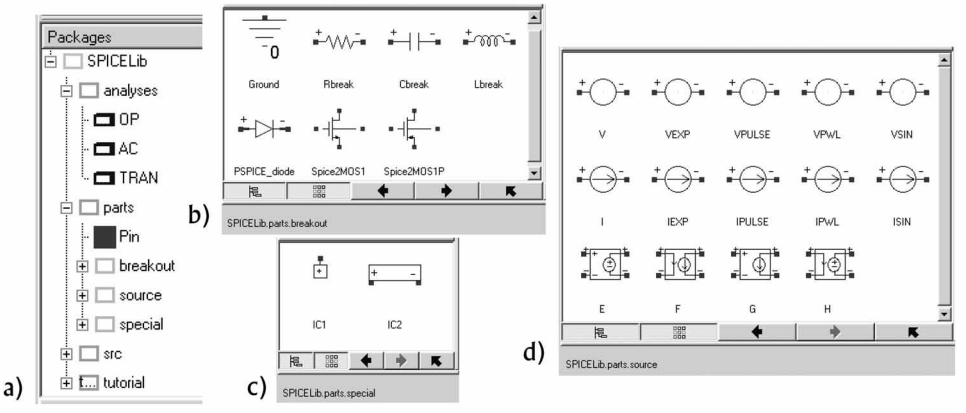


Figure 1. (a) *SPICE*Lib sub-libraries; (b), (c) and (d) Device models in *SPICELib.parts* sub-library.

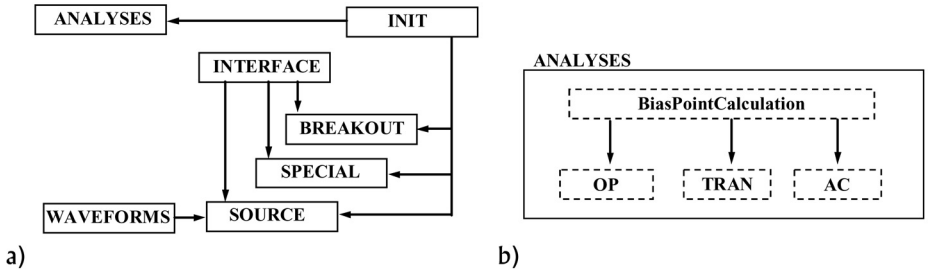


Figure 2. (a) Sub-libraries of *SPICELib.src* package; (b) Models in *SPICELib.src.ANALYSES* package.

behavior, and adding the documentation oriented to the library users. The *SPICE*Lib.*parts* package is composed of three sub-libraries: *breakout* (see Figure 1b), *special* (see Figure 1c) and *source* (see Figure 1d).

- The *SPICE*Lib.*tutorial* package (see Figure 1a) contains some tutorial examples, explaining in detail how to use the *parts*-library models to compose the circuit schematic, and how to use the *analyses*-library models to analyze this previously defined circuit.

The architecture of the *SPICE*Lib.*src* package is outlined in Figure 2a (further details can be found in [7]). An arrow from A-package to B-package means that B-package models inherit from A-package models. The *INIT* package contains the declaration of the control-signals, other global variables and the global parameters. These declarations are inherited by the analysis models of the *ANALYSES* package, and by the device models of *BREAKOUT* (passive and semiconductor device models), *SOURCE* (source models) and *SPECIAL* (pseudo-component models) packages. The device models of these three packages inherit their interface definitions (i.e., the description of the device pins) from the *INTERFACE* package. The *WAVEFORMS* package contains the models of the waveforms used to define the transient behavior of the independent sources.

The *SPICE*Lib.*src.ANALYSES* package contains the models of the OP, AC and TRAN analyses. Bias point calculation is a part of the OP and AC analyses, and it is an option of the TRAN analysis. Therefore, *SPICE*Lib algorithms for bias point

calculation are modeled in a separate *partial model*, called *BiasPointCalculation*, which is inherited by the *OP, AC* and *TRAN* models (see Figure 2b).

4. Circuit modeling and analysis with *SPICE*Lib

Circuit analysis using *SPICE*Lib requires two pieces of information: (1) the circuit-schematic model, defined instantiating and connecting the required *SPICE*Lib device models; and (2) the analysis specification, defined instantiating the correspondent *SPICE*Lib analysis model. These *SPICE*Lib analysis models contain a parameter whose value specifies the name of the circuit model to analyze. The user needs to specify this parameter value as a part of the analysis definition.

The steps to building the circuit model and defining the analysis, using *SPICE*Lib with the Dymola modeling environment [12], are the following [7,8]:

Step 1. Define a new *partial* model: the circuit schematic. Drag and drop the required circuit components from *SPICE*Lib.*parts* sub-library to the model window, connect them and set the value of the device model parameters.
Step 2. Define a new model: the analysis. Drag and drop the required analysis model from *SPICE*Lib.*analyses* sub-library to the model window, and set the value of the analysis parameters. One of this parameters is the name of the circuit model defined in Step 1.
Step 3. Simulate the analysis model defined in Step 2.

5. Setting initial conditions

*SPICE*Lib supports the same procedures as PSpice for setting the initial operating point of the circuit [7]: (1) IC1 and IC2 pseudo-components (also called IC symbols); and (2) IC property of capacitors and inductors. These ways of specifying the initial conditions substitute the model initialization procedures of Modelica (which are discussed in [13]).

The IC property allows the association of the initial condition with a device, while IC symbols (see Figure 1c) allow the association to be with a circuit node or a node pair. IC1 is a one-pin symbol that allows setting the initial voltage on a circuit node. IC2 is a two-pin symbol that allows setting the initial voltage between two nodes. The capacitor IC property sets the initial voltage-drop across the capacitor. The inductor IC property sets the initial current through the inductor.

The IC symbols clamp the voltage for the entire bias point calculation. *SPICE*Lib attaches a voltage source with a 0.0002 ohm series resistance (*R_EPS*) at each circuit node to which an IC1 symbol is connected, and between the two nodes to which an IC2 symbol is connected. The *SPICE*Lib static description of IC1 symbol is represented in Figure 3a. The implementation of IC2 symbol is analogous: the ground connection in IC1 model is replaced by a second pin: n (−).

The capacitor IC property is implemented by *SPICE*Lib using an IC2 symbol in parallel with the capacitor (see Figure 3b). The implementation of the inductor IC property is analogous: *SPICE*Lib attaches a current source with a 1 Gohm parallel resistance (*R_BIG*) in series with the inductor (see Figure 3c).

Asterisks in Figure 3 (i.e., (*) and (**)) represent control-signals. The arrow from the control-signal indicates the device controlled by the control-signal. For instance, while

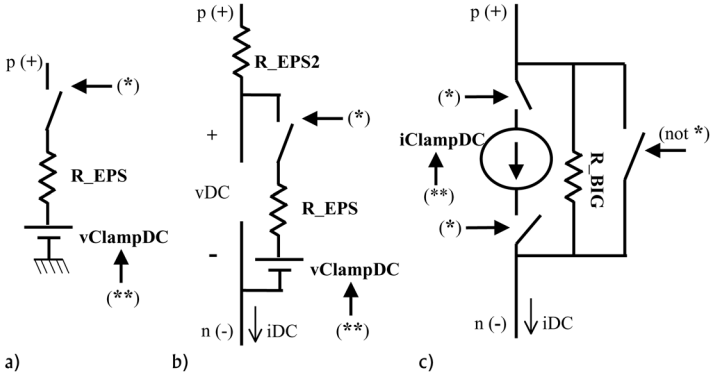


Figure 3. Static descriptions of: (a) IC1 symbol; (b) *Cbreak* capacitor; (c) *Lbreak* inductor.

the control-signal represented by (*) equals true, the switches are closed, and while (*) equals false they are open. In addition, the value of (**) control-signal sets the value of the voltage and current sources. The implementation details are discussed in [7,8].

6. Bias point calculation

*SPICE*Lib implements four algorithms for calculating the bias point of the circuit [7]. Only three of these are supported by PSpice [4,5] and the fourth algorithm is ''dynamic model ramping'' [9]. *SPICE*Lib allows the user to choose among these four algorithms in order to perform the OP analysis, calculate the operating point prior to the AC sweep analysis, and evaluate the steady-state initial condition of the transient analysis. The algorithm selection is made by setting the value of a parameter of the *BiasPointCalculation* model [7]. If the selected algorithm fails to converge, then the analysis run aborts, and the user has to re-start the analysis run, setting different values of the analysis model parameters or selecting a different algorithm.

Two control-signals, *biasPoint* and *biasPointCalculated* [7], are used to synchronize the bias point calculation with other analysis operations. The *OP, AC* or the *TRAN* model triggers the transition of the *biasPoint* control-signal from false to true when the calculation of the bias point is required to start. This change in *biasPoint* control-signal triggers the execution of the selected algorithm, which is carried out by the *BiasPointCalculation* model.

Once the algorithm execution is completed, *BiasPointCalculation* model saves the bias point values to a text file and changes the value of *biasPointCalculated* control-signal from false to true. This change indicates to the *OP, AC* or *TRAN* model that the bias point calculation is finished.

OP analysis requires no more actions in addition to the calculation of the bias point. As a consequence, when the *biasPointCalculated* control-signal becomes true, the OP analysis switches the value of *terminate* control-signal from false to true. The action associated to this event is a call to a Modelica function (*terminate* Modelica function [11]), which forces the simulation end.

AC and TRAN analyses contain more steps, in addition to the bias point calculation. When the *biasPointCalculated* control-signal becomes true, these analysis models start the next analysis step. Once all the steps of the analysis are complete, the

analysis model (i.e., *AC* or *TRAN* model) changes the value of *terminate* control-signal from false to true.

The fundamentals of the *SPICE*Lib algorithms for the bias point calculation are discussed next. The implementation details of these algorithms can be found in [7].

6.1 *"Static model iteration" algorithm*

PSpice first tries to solve the static formulation of the circuit using the Newton-Raphson algorithm. "Static model iteration" algorithm constitutes the *SPICE*Lib implementation of this approach. The time-evolution of the control-signals implementing this algorithm are represented in Figure 4. When the bias point calculation starts (i.e., *biasPoint* control-signal becomes true), *BiasPointCalculation* model triggers the changes in the control-signals required to: (1) enable the static formulation of the circuit (*ctrl_DC* control-signal); (2) clamp the voltages and currents to their initial values (*ctrl_IC_clampDC* control-signal, which was represented by an asterisk in Figure 3); and (3) save the calculated bias point in a text file (*ctrl_log_DC* control-signal). In addition, the *BiasPointCalculation* model triggers the change of the *biasPointCalculated* control-signal from false to true (see Figure 4). The *CLOCK* time-interval shown in Figure 4 is a model parameter representing the time elapsed between two consecutive control-signal transitions.

Therefore, the "static model iteration" algorithm leaves the solution of the static circuit completely in charge of the modeling environment solver. In this case, the user should provide the initial values to iterate the static circuit model, using the experiment-definition capabilities of Modelica and the modeling environment [12]. *SPICE*Lib 1.1 does not support the PSpice NODESET1 and NODESET2 symbols, which are intended to provide an initial guess for the Newton-Raphson algorithm [4].

A parameter of the *BiasPointCalculation* model allows the user to specify the required detail level at saving the results [7]. In addition, the device atomic-models contain a parameter in order to classify the circuit devices into two types: those whose variables have to be saved always, and those whose variables have to be saved only in special cases. By default, *SPICE*Lib considers that the atomic-models composing the semiconductor device models belong to this second group.

Example 1. The model of the AC to DC octupler circuit shown in Figure 5 has been built connecting the SPICELib.parts package components. This circuit creates a DC output at eight times the AC value [14]. The results of OP analysis using "static model iteration" algorithm are compared in Figure 6 with the results obtained using OrCAD PSpice. The percentage error is smaller than 0.02%.

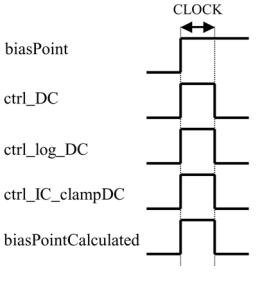


Figure 4. *SPICE*Lib control-signals implementing the "static model iteration" algorithm.

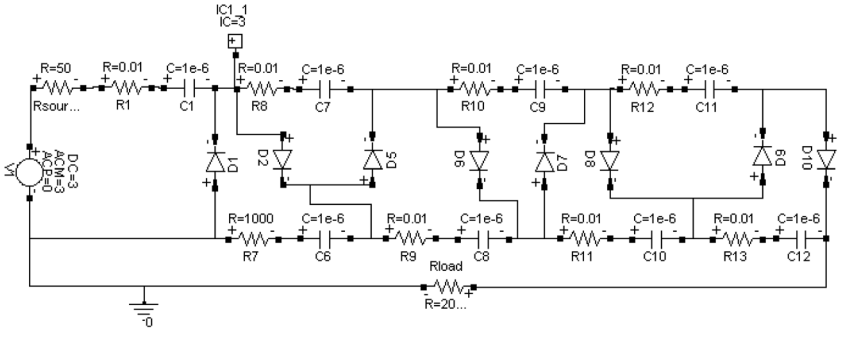
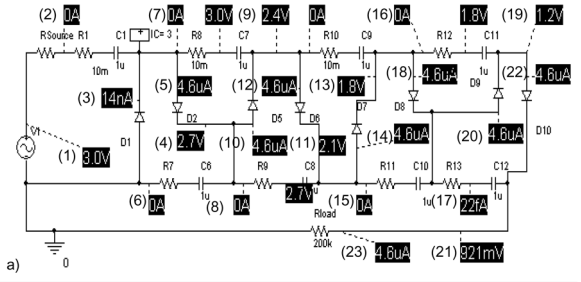


Figure 5. *SPICE*Lib model of an AC to DC octupler circuit.



b)

OP1_V1_vDC(1e-010) = 3	(1)
OP1_Rsource_iDC(1e-010) = 0	(2)
OP1_D1_p_iDC(1e-010) = -1.429465907437056e-008	(3)
OP1_C6_Rx_0EPS2_n_vDC(1e-010) = 2.703067158530661	(4)
OP1_D2_p_iDC(1e-010) = 4.607350580285652e-006	(5)
OP1_R7_iDC(1e-010) = 0	(6)
OP1_R8_iDC(1e-010) = -7.95560370441178e-038	(7)
OP1_R9_iDC(1e-010) = -3.08706197604614e-030	(8)
OP1_C7_Rx_0EPS2_n_vDC(1e-010) = 2.406134317954946	(9)
OP1_D5_p_iDC(1e-010) = 4.607350580285652e-006	(10)
OP1_C8_Rx_0EPS2_n_vDC(1e-010) = 2.109201477334306	(11)
OP1_D6_p_iDC(1e-010) = 4.607350580285652e-006	(12)
OP1_C9_Rx_0EPS2_n_vDC(1e-010) = 1.812268636966913	(13)
OP1_D7_p_iDC(1e-010) = 4.607350580285652e-006	(14)
OP1_R11_iDC(1e-010) = -1.632107143697598e-028	(15)
OP1_R12_iDC(1e-010) = -1.087486358618267e-028	(16)
OP1_R13_iDC(1e-010) = -9.99361960342605e-031	(17)
OP1_D8_p_iDC(1e-010) = 4.607350580285652e-006	(18)
OP1_C11_Rx_0EPS2_n_vDC(1e-010) = 1.218402956312039	(19)
OP1_D9_p_iDC(1e-010) = 4.607350580285652e-006	(20)
OP1_C12_Rx_0EPS2_n_vDC(1e-010) = 0.9214701160571304	(21)
OP1_D10_p_iDC(1e-010) = 4.607350580285652e-006	(22)
OP1_Rload_iDC(1e-010) = 4.607350580285652e-006	(23)

Figure 6. OP analysis results: (a) Using OrCAD PSpice; (b) Using *SPICE*Lib.

6.2 *"GMIN stepping" algorithm*

If the Newton-Raphson algorithm does not converge and "GMIN stepping" is enabled, then PSpice applies the "GMIN stepping" algorithm. It attempts to facilitate the convergence of the Newton-Raphson algorithm by modifying the value of the circuit GMIN conductance [4,5].

GMIN conductance is included in the PSpice (and *SPICE*Lib) models of PN-junction diodes and other semiconductor devices containing PN-junctions. It is intended to address the convergence problems experienced when the voltage-drop needs to be calculated from the diode constitutive relation, and the diode is operating

in the reverse-bias region [5]. In this region of operation, the current through the diode is virtually constant (i.e., independent of the voltage). In other words, the diode conductance (i.e., the slope of the device's I-V plot) is very close to zero. As a consequence, the Newton-Raphson algorithm significantly overshoots the correct solution voltage, and a large number of algorithm iterations are required to work back to the correct solution voltage [5]. Worse yet, if the conductance value reaches zero, the next Newton-Raphson iteration will cause a floating divide-by-zero error.

This problem is addressed by PSpice (and consequently by *SPICE*Lib) including a shunt resistor with conductance GMIN in the diode model [4,5]. Consequently, the diode conductance under reverse-bias conditions is equal to GMIN. A GMIN resistor has a by-default conductance value of GMIN = 10^{-12} mhos [4]. However, the GMIN value should be set as large as possible without affecting the accuracy of the simulation output. Indeed, the larger the GMIN value, the faster the Newton-Raphson algorithm will converge to a solution. Recommendations to choose a GMIN value appropriate to a particular circuit are provided in [5].

The *SPICE*Lib implementation of "GMIN stepping" algorithm is represented in Figure 7. It takes advantage of the device equation continuity with respect to GMIN parameter. When *biasPoint* control-signal becomes true, the *BiasPointCalculation* model: (1) enables the static formulation of the circuit (by changing the value of *ctrl_DC* control-signal from false to true); (2) clamps the voltages and currents to their initial values (by switching from false to true *ctrl_IC_clampDC* control-signal); and (3) sets a large value of GMIN: initially 10^{10} times the nominal value. In the *SPICE*Lib implementation (see Figure 7), *scaleGMIN* control-signal is the scale factor applied to GMIN.

If a solution is found at this setting, the *BiasPointCalculation* model reduces GMIN by a factor of 10 and a new solution is found. This continues until GMIN is back to the nominal value, or the repeated cycle fails to converge. Once GMIN has reached the nominal value (i.e. the value of *scaleGMIN* control-signal equals one), then the *BiasPointCalculation* model saves the results in a text file (i.e., *ctrl_log_DC* control-signal is switched to true), and finally it changes the value of the *biasPointCalculated* control-signal from false to true.

Example 2. The OP analysis of the circuit shown in Figure 5 is performed using *SPICE*Lib "GMIN stepping" algorithm. The evolution of the decimal logarithm of the ratio between GMIN and the nominal value of GMIN is shown in Figure 8a. The

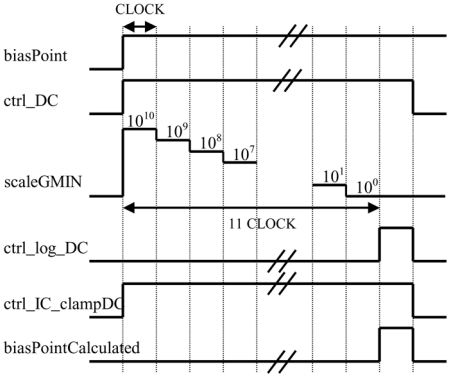


Figure 7. *SPICE*Lib control-signals implementing the "GMIN stepping" algorithm.

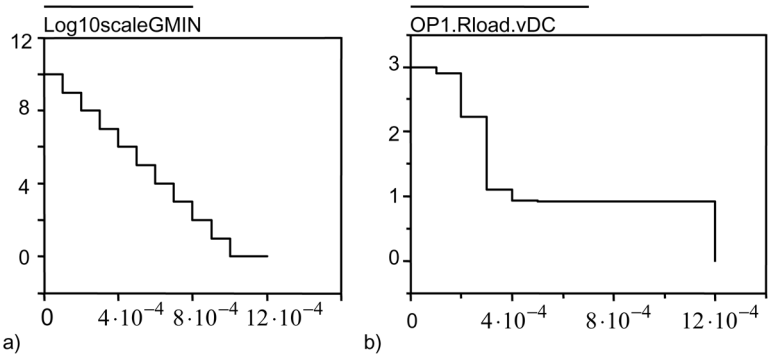


Figure 8. OP analysis applying "GMIN stepping" algorithm: (a) log10 $\left(\frac{\text{GMIN}}{\text{nominal GMIN}}\right)$; (b) Voltage drop across the load resistor.

BiasPointCalculation model reduces this ratio by a factor of ten every 10^{-4} seconds (this time value is a model parameter). When the nominal value of GMIN is reached (in this case, at time equals 10^{-3} seconds), the *BiasPointCalculation* model saves the analysis results in a text file. These are equal to the results shown in Figure 6b. The evolution of the voltage drop across the load resistor is shown in Figure 8b. The *BiasPointCalculation* model triggers the simulation end at time equals 12×10^{-4} seconds.

6.3 *"Static model ramping" algorithm*

If the "GMIN stepping" algorithm fails to converge or it is not enabled, then PSpice tries to solve the static circuit ramping the independent sources from almost zero to their desired initial value [4]. The process relies heavily on the continuity of PSpice device models with respect to the power supplies. First, PSpice cuts back the power supplies of the static description to almost zero (0.001%), so that all the non-linearities are turned off. In this situation, the static description of the circuit is linear, and a solution can be found. The initial condition of this first step is zero for all voltages. Then, PSpice works its power supplies back up to 100% using a variable step size [4].

The "static model ramping" algorithm constitutes the *SPICE*Lib implementation of this approach. When the *biasPoint* control-signal becomes true, the *BiasPointCalculation* model triggers the changes in the control-signals required to ramp up the value of the independent sources and the value of the IC clamping sources (i.e., *vClampDC* and *iClampDC* in Figure 3), from zero to their desired initial values. The *SPICE*Lib models of independent sources support this capability. Finally, the *BiasPointCalculation* model saves the results in a text file, and it switches the value of the *biasPointCalculated* control-signal from false to true. The implementation details can be found in [7].

Example 3. The OP analysis of the circuit shown in Figure 5 is performed again, this time applying the "static model ramping" algorithm. The *BiasPointCalculation* model ramps the power supplies and the IC symbols and properties from zero up to their nominal values. The ramping of the IC-symbol value and the evolution of the voltage drop across the load resistor are plotted in Figure 9a. Once the nominal value is reached (in this case, at time equals 10^{-4} seconds, the *BiasPointCalculation* model saves the bias point in a text file. The obtained results are equal to the ones shown in Figure 6b.

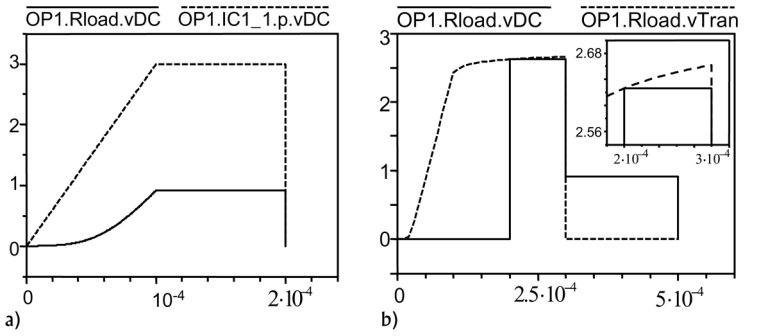


Figure 9. OP analysis applying: (a) "Static model ramping" algorithm; (b) "Dynamic model ramping" algorithm.

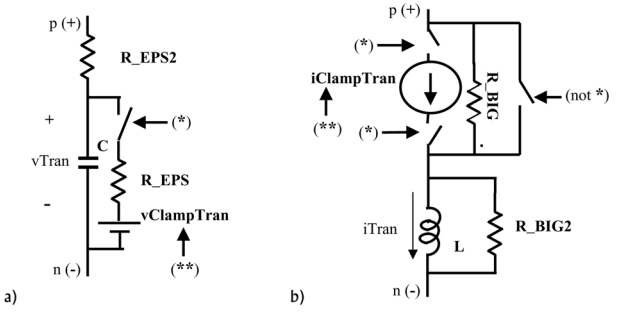


Figure 10. Large-signal descriptions of: (a) *Cbreak* capacitor; (b) *Lbreak* inductor.

6.4 *"Dynamic model ramping" algorithm*

This algorithm for the bias point calculation was proposed by F. Cellier in [9]. In this case, the initial condition to iterate the static description of the circuit is obtained by simulating its large-signal description. The independent sources are ramped from zero to their desired initial value, and these values are held for some time to allow the node voltages to stabilize. These voltages, obtained by simulating the circuit large-signal description, are used as the initial condition to iterate the circuit static description.

In order to implement this algorithm, IC-like clamping circuits have been included in the large-signal description of *SPICE*Lib capacitors and inductors (see Figure 10). The *SPICE*Lib implementation of the algorithm is as follows. When the *biasPoint* control-signal becomes true, the *BiasPointCalculation* model triggers the changes in the control-signals required to perform a simulation of the large-signal description of the circuit, ramping the independent sources and the initial conditions of the circuit (*vClampTran* and *iClampTran* in Figure 10) from zero to their desired initial values. These values are held for some time to allow the circuit to stabilize.

Next, the *BiasPointCalculation* model enables the static description of the circuit, with the power supplies disconnected, and the node voltages of the large-signal description are transferred to the static description. The *BiasPointCalculation* model accomplishes this by attaching a voltage source with a 0.0002 ohm series resistance to each node of the static circuit. Next, the power supplies of the static description are connected to the circuit: the equations of the static description of the circuit are iterated, using as initial condition the bias point calculated from the large-signal

description simulation. Finally, the *BiasPointCalculation* model saves the results in a text file and switches the *BiasPointCalculated* control-signal from false to true. The implementation details can be found in [7].

Example 4. The operating point of the circuit shown in Figure 5 is calculated applying ''dynamic model ramping'' algorithm (see Figure 9b). First, a transient analysis is performed: the power supplies and the IC symbols and properties are ramped from zero up to their nominal values. In this case, the ramping takes 10^{-4} seconds (this time value is a model parameter). Then, these values are held during another 10^{-4} seconds to allow the circuit to stabilize. The dotted line in Figure 9b represents the evolution of the voltage drop across the load obtained from solving the circuit large-signal description.

At time equals 2×10^{-4} seconds, the large-signal description voltages are transferred to the static description. The continuous line in Figure 9b represents the evolution of the voltage drop across the load resistor calculated from the static description of the circuit. 10^{-4} seconds later (at time equals 3×10^{-4}), the power supplies of the static description are connected: the circuit static formulation is solved. In the same instant, the large-signal formulation of the circuit is disabled. The OP analysis results are logged out at *time* = 4×10^{-4} seconds, and *terminate* control-signal is switched from false to true at *time* = 5×10^{-4} seconds.

7. AC sweep analysis

The AC sweep analysis starts with the bias point calculation: the *AC* model changes the value of the *biasPoint* control-signal from false to true (see Figure 11). When the *biasPointCalculated* control-signal becomes true, the *AC* model switches *ctrl_AC* control-signal from false to true. This switching triggers the calculation of the small-signal model parameters from the voltage and current values at the bias point. Next, *AC* model ramps the frequency value (*freq* variable in Figure 11) from its starting value up to its final value. *SPICE*Lib supports two types of frequency sweep: linear and logarithmic by decades.

Every time *ctrl_log_AC* control-signal changes from false to true, the solution of the circuit small-signal description is saved in a text file (see Figure 11). The transitions in the value of this control-signal are synchronized with the frequency sweep, so that the

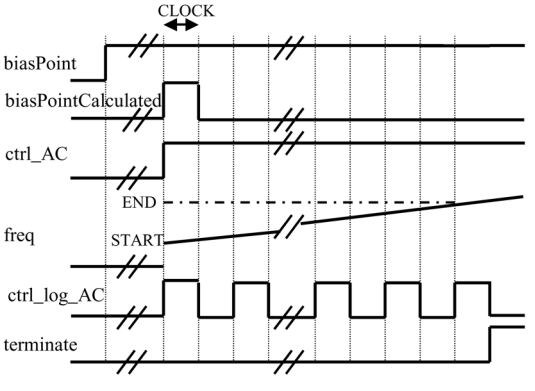


Figure 11. *SPICE*Lib control-signals implementing the AC-sweep analysis.

analysis results are obtained at the selected frequency values. When the final frequency is reached, the *AC* model changes the value of *terminate* control-signal from false to true. Further details of the *SPICE*Lib AC sweep analysis can be found in [7].

Example 5. *SPICE*Lib is used to perform the AC sweep analysis of the circuit shown in Figure 5. The integration algorithm is Dassl, with a tolerance equal to 10^{-4}. The results are shown in Figures 12a and 12b, and they are compared with the results obtained using OrCAD PSpice (see Figures 12c and 12d). However, a significant improvement in the error is not achieved by reducing to 10^{-7} the Dassl tolerance.

In general, the analysis CPU-time using *SPICE*Lib is greater than using OrCAD PSpice. *SPICE*Lib performs a continuous sweep from the initial value of the frequency up to its final value (see the *freq* variable in Figure 11). As a consequence, although the analysis results are saved to a text file only at the frequencies selected by the user, the AC small-signal formulation is solved for the intermediate values of the frequency (as determined by the step-size of the DAE-solver), and the obtained results are saved to another file (as determined by the communication interval of the DAE-solver). This second file contains much more detailed information, and it is intended to be used as a data source for plotting the voltages and currents of the AC small-signal description as continuous functions of the frequency.

8. Transient analysis

*SPICE*Lib implements the two same procedures for transient analysis initialization as PSpice: bias point calculation and skipping the bias point calculation. The *SPICE*Lib user can select the initialization procedure and, if required, the algorithm for the bias point calculation to be used, among the four algorithms supported by *SPICE*Lib.

The *SPICE*Lib transient analysis can be performed on circuit models, composed of *SPICE*Lib.*parts* sub-library components, and also on multi-domain models, built by

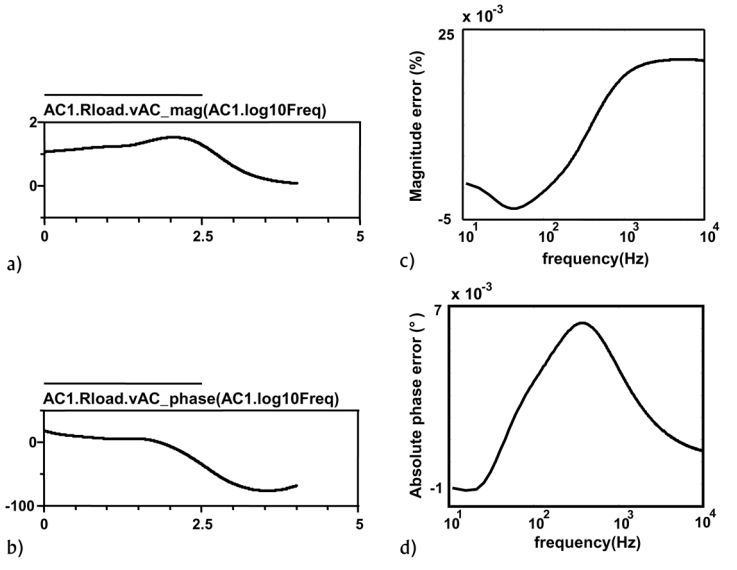


Figure 12. (a) and (b) *SPICE*Lib AC sweep analysis; (c) and (d) Difference between *SPICE*Lib and PSpice AC sweep analyses.

combined use of *SPICE*Lib and other Modelica libraries. These topics are discussed next.

8.1 *Initialization with bias point calculation*

In this case, the initial bias for transient analysis is the DC bias-point of the circuit, calculated setting the independent sources to their respective transient-analysis initial values (which, in general, are different from their DC values [4]). The IC-symbols and the IC-properties can be used to complete the specification of this initial bias point.

When the transient analysis starts, the *TRAN* model switches the *biasPoint* control-signal value from false to true (see Figure 13a). This change triggers the execution of the selected algorithm for the bias point calculation. *SPICE*Lib models of independent sources have been designed to support the different initialization strategies. They allow setting (or ramping up from zero to) their DC value or the initial value of their transient waveform. Control-signals, whose values are set by the *BiasPointCalculation* model, determine the behavior of the independent sources during the bias point calculation (see [7] for further details).

Once the execution of the selected bias point algorithm is completed, the *BiasPointCalculation* model changes the value of the *biasPointCalculated* control-signal from false to true. Then, the *TRAN* model: (1) enables the large-signal description of the circuit (by switching *ctrl_Tran* control-signal); and (2) transfers the bias point values from the static description of the circuit to its large-signal description (by switching *ctrl_CBREAK_Tran2DC* control-signal). This is carried out by initializing the state variables of the circuit large-signal description to the calculated DC voltage and current values. Next, the *TRAN* model starts the time-sweep, using the large-signal description of the circuit.

When the time-sweep starts, the value of the Modelica *time* variable can be different from zero. This is the case when the ''GMIN stepping'', ''static model ramping'' or the ''dynamic model ramping'' algorithm is applied for calculating the bias point. For this reason, a variable is defined in the *TRAN* model to measure the elapsed simulated-time during the time-sweep: *TIME*. Once the stop-time is reached (i.e., *TIME* equals *TRAN_STOP_TIME*), the *TRAN* model changes the value of *terminate* control-signal from false to true.

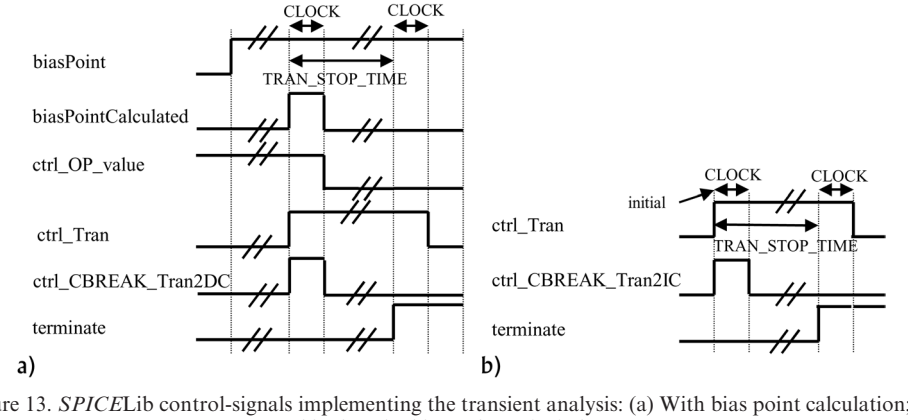


Figure 13. *SPICE*Lib control-signals implementing the transient analysis: (a) With bias point calculation; (b) Skipping the bias point calculation.

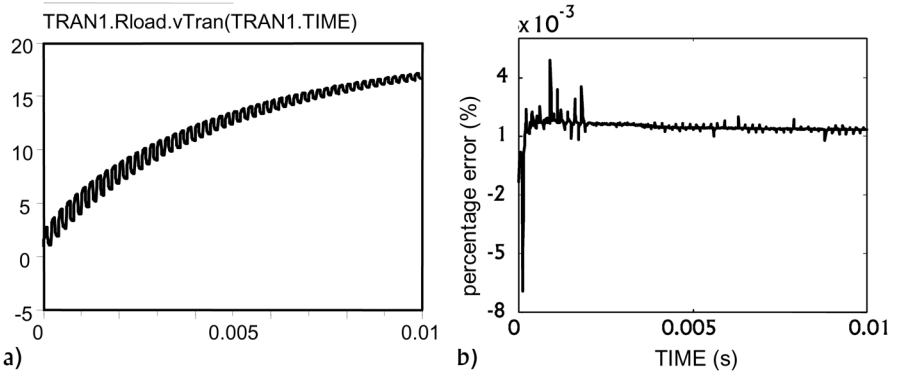


Figure 14. (a) *SPICE*Lib transient analysis with bias point calculation; (b) Difference between *SPICE*Lib and PSpice transient analyses.

Example 6. A transient analysis of the circuit shown in Figure 5 has been performed using *SPICE*Lib. In this case, the IC symbol has been removed from the circuit. The voltage source stimulus is a pulse waveform with the following parameters: $v1 = -3$ (initial voltage), $v2 = 3$ (pulsed voltage), $TD = 0$ (delay), $TR = TF = 10^{-6}$ (rise and fall time), $PW = 10^{-4}$ (pulse width), $PER = 2 \times 10^{-4}$ (period). The time evolution of the voltage drop across the load resistor is shown in Figure 14a. The DAE-solver used in this *SPICE*Lib analysis is Dassl, with a tolerance value equal to 10^{-7}.

This analysis result is compared in Figure 14b with the result obtained using ORCAD Pspice, with a maximum step-size of 10^{-7}. In this example, the error exhibits a significant dependence of the Dassl tolerance: the maximum error decreases from 3% to 8×10^{-3}% by reducing the Dassl tolerance from 10^{-4} to 10^{-7}. The CPU-time of the *SPICE*Lib transient analysis (with a Dassl tolerance of 10^{-7}) is approximately two times greater than the PSpice CPU-time (with a maximum step-size of 10^{-7}).

8.2 *Initialization by skipping the bias point calculation*

In this case, IC-symbols are ignored. The initial bias of the large-signal description is fully determined by the IC-property of capacitors and inductors, which set the initial value of the description state variables. When the transient analysis starts, the *TRAN* model enables the large-signal description of the circuit: (1) independent sources are set to the initial value of their waveforms; and (2) the voltage across the capacitors and the current through the inductors is initialized according to their IC-property values. These two actions are accomplished by switching the value of *ctrl_Tran* and *ctrl_CBREAK_Tran2IC* control-signals from false to true (see Figure 13b). Next, the time-sweep is performed until the stop-time (*TRAN_STOP_TIME*) is reached. Then, the *TRAN* model sets *terminate* control-signal to true, which forces the analysis to end.

8.3 *Transient analysis of multi-domain models*

*SPICE*Lib transient analysis can be performed on Modelica multi-domain models in any of the following two cases. Case 1: the bias point calculation is skipped. Case 2: the bias point is calculated using the "static model iteration" algorithm. In both cases, the

time-sweep starts at the beginning of the transient analysis. Also, the actions triggered by the control-signals, which affect only the "*SPICE*Lib part" of the model, do not interfere with the analysis of the complete model.

In Case 1, only one description is enabled during the analysis in the *SPICE*Lib part of the model: the circuit large-signal description. The static and the AC small-signal descriptions of the circuit are disabled. As a consequence, the components interfacing between the *SPICE*Lib part and the non-*SPICE*Lib part of the model have to: (1) establish the required connections among the variables of the circuit large-signal description and the variables of the non-*SPICE*Lib part of the model; and (2) impose trivial boundary conditions (if required) to the static and the AC small-signal description of the circuit, in order to obtain a complete model with the same number of equations and unknown variables.

In Case 2, the static and the large-signal descriptions of the circuit are enabled during the analysis. The AC small-signal description is disabled. The interface components have to: (1) establish the required connections among the variables of the circuit large-signal description and the variables of the non-*SPICE*Lib part of the model; (2) impose adequate boundary conditions to the circuit static formulation (if required), in order to allow the bias point calculation; and (3) impose trivial boundary conditions to the AC small-signal description of the circuit (if required), in order to obtain a complete model with the same number of equations and unknown variables.

The procedure to build the multi-domain model and to define the transient analysis, using the Dymola modeling environment, is analogous to the one described in Section 4:

Step 1. Define a new *partial* model: the multi-domain model. Drag and drop the required components from the *SPICE*Lib.*parts* sub-library and from the other Modelica libraries to the model window, connect them and set the value of the model parameters.
Step 2. Define a new model: the transient analysis. Drag and drop the *TRAN* model from the *SPICE*Lib.*analyses* sub-library to the model window, and set the value of the analysis parameters. One of these parameters is the name of the model defined in Step 1. Other parameters define whether the bias point calculation is skipped or performed, and the algorithm to apply for bias point calculation.
Step 3. Simulate the analysis model defined in Step 2.

9. Conclusions

*SPICE*Lib is an object-oriented model library that implements some of the modeling and analysis capabilities of the circuit simulator PSpice. *SPICE*Lib contains device and analysis models. Device analog models include passive and semiconductor devices, independent and controlled sources, and IC-symbols. Three analysis models have been implemented: operating point (OP), AC sweep (AC) and transient (TRAN); in addition to four algorithms for bias point calculation. The *SPICE*Lib device and analysis models, and the *SPICE*Lib algorithms for bias point calculation, are completely written in the Modelica modeling language.

The fundamental hypotheses and the architecture of the *SPICE*Lib library have been discussed, in addition to the modeling of the circuit analyses and the device atomic-models. A case study has been fully developed, in order to illustrate the *SPICE*Lib use and validation.

Acknowledgements

The authors wish to thank the referees of this paper for their constructive comments.

References

[1] Web site of Modelica Association: http://www.modelica.org/.
[2] Aström, K. J., Elmqvist, H. and Mattsson, S.E., 1998, Evolution of Continuous-Time Modeling and Simulation. *Proceedings of the 12th ESM*, Manchester, UK. pp. 16–19. (Retrieved from the website: http://www.modelica.org/).
[3] Clauß, C., Schneider, A., Leitner, T. and Schwarz, P. 2000, Modelling of Electrical Circuits with Modelica. *Proceedings of the Modelica Workshop*, Lund, Sweden. (Retrieved from the website: http://www.modelica.org/).
[4] OrCAD, Inc. 1999, OrCAD PSpice A/D. *Reference Guide and User's Guide*. OrCAD, Inc.
[5] Kielkowski, R.M., 1998, *Inside SPICE* (New York: McGraw-Hill). 2nd edition.
[6] Massobrio, G. and Antognetti P., 1993, *Semiconductor Device Modeling with SPICE* (New York: McGraw-Hill).
[7] Urquia, A. and Dormido, S., 2002, DC, AC Small-Signal and Transient Analysis of Level 1 N-Channel MOSFET with Modelica. *Proceedings of the 2nd International Modelica Conference*, March, Oberpfaffenhofen, Germany, 99–108. (It can be downloaded from the website: http://www.modelica.org/).
[8] Martin, C., Urquia, A. and Dormido, S., 2003, *SPICE*Lib – Modeling and Analysis of Electric Circuits with Modelica. *Proceedings of the 3rd International Modelica Conference*, November, Linkoping, Sweden, 161–170. (It can be downloaded from the website: http://www.modelica.org/).
[9] Cellier, F.E., 1991, *Continuous System Modeling* (New York: Springer-Verlag).
[10] Modelica Association. 2000, Modelica, Tutorial, Version 1.4. (Retrieved from the website: http://www.modelica.org/).
[11] Modelica Association. 2002, Modelica, Language Specification, Version 2.0. (Retrieved from the website: http://www.modelica.org/).
[12] Elmqvist, H., Brück, D., Mattson, S.E., Olsson, H. and Otter, M., 2002, Dymola. User's Manual. Version 5.0a. Dynasim AB.
[13] Mattsson, S.E., Elmqvist, H., Otter, M. and Olsson, H., 2002, Initialization of Hybrid Differential-Algebraic Equations in Modelica 2. *Proceedings of the 2nd International Modelica Conference*, March, Oberpfaffenhofen, Germany, 9–15. (It can be downloaded from the website: http://www.modelica.org/).
[14] Sandler, S.M. and Analytical Engineering, Inc. 2004, *The SPICE Handbook of 50 Basic Circuits*. (Retrieved from the website: http://www.pcbcafe.com/BOOKS/SpiceHandBook/).