SPICE3 Modelica Library

Kristin Majetta Sandra Böhme Christoph Clauß Peter Schneider
Fraunhofer-Institute for Integrated Circuits, Design Automation Division
Zeunerstraße 38, 01069 Dresden, Germany
{Kristin.Majetta, Sandra.Boehme, Christoph.Clauss, Peter.Schneider}@eas.iis.fraunhofer.de

Abstract

Since the very beginning of the Modelica development ambitions for electronic simulation exist. The electronic simulator SPICE, the SPICE models and the SPICE netlists grew to a quasi standard in electronics simulation for the last 30 years. That is why the wish arose to have SPICE models available in Modelica. This paper deals with modeling the SPICE3 models in Modelica directly extracted from the original SPICE3 source code. This courses the problem of transforming the sequential simulator-internal model descriptions of SPICE to the declarative description from Modelica. To solve this problem a way was developed and tested for some SPICE3 semiconductor models. The actual library is presented and further plans are shown.

Keywords: SPICE, Modelica, SPICE3 library for Modelica, Semiconductor models, Electronic circuit simulation

1 Motivation

With starting the development of Modelica, models for electrical circuits were taken into consideration [1]. Since SPICE and its derivatives grew to a quasi standard in electronics simulation the SPICE models should become available within Modelica.

Beyond the Modelica standard library (MSL) two SPICE libraries were developed [2], the SPICELib and the BondLib. The SPICELib [3], which covers different complex MOSFET models, is a standalone library with its own connectors. The BondLib [4] bases on bond graphs. It offers different levels of models related to HSpice.

The reason for developing this SPICE3 library is to provide both the original Berkeley SPICE3 models and the SPICE netlist approach. Furthermore, some additions will be prepared to cover PSPICE models.

Since the Berkeley SPICE3 simulator is the only known electric circuit simulation program with open source code it offers the opportunity to extract models for implementation in Modelica. The SPICE3 library uses that way for SPICE3 semiconductor models.

In this paper the modelling steps are considered which are done starting with a C++-model library which was extracted from SPICE3 formerly. The SPICE3 library structure is presented as well as a circuit example.

2 SPICE3 models and netlists

The Berkeley SPICE3 (latest versions e5 or f4) is a general-purpose circuit simulation program which has built-in models both for general devices (resistors, capacitors, inductors, dependent and independent sources) and semiconductor devices (Diode, MOSFET, BJT,…). Some models are a collection of different single models (levels). Instead of adding new models the user is able to choose a large variety of parameters. Only sometimes a new model is added by the developer. The set of SPICE models is like a standard in circuit simulation.

Via a netlist the SPICE3 models are composed to a circuit to be simulated. The netlist contains the model instances, their actual parameters, and the connection nodes. In more detail SPICE3 netlists are described in the SPICE3 user's manual [5]. For many electric and electronic devices SPICE3 netlists are available. For the following inverter circuit figure 2 shows the SPICE3 netlist.

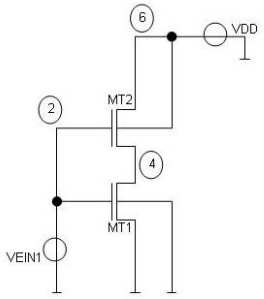

Fig. 1 MOSFET inverter circuit

```
 Simulation inverter circuit
 MT1  4 2 0 0 Tran_NMOS L=2u  W=5u
 MT2  6 2 4 6 Tran_PMOS
 VDD  6 0 5.0
 VEIN1 2 0 dc=0 sin(0 5 0.5)
 .model  Tran_NMOS NMOS (VT0 =0.7
 tox=8n lambda=3e-2)
 .model  Tran_PMOS PMOS (VT0 =-0.7)
 .tran 0.001 10
 plot v(6) v(2) v(4)
.end
```

Fig. 2 SPICE3 netlist of the inverter circuit

Within the semiconductor devices SPICE3 differentiates between technology parameters and device parameters. Device parameters can be chosen for every single model instance, e.g. the channel length of a transistor. Technology parameters which are specified in a model card (.model) are adjustable for more than one element simultaneously, e.g. the type of transistors.

3 Model extraction out of SPICE3

The SPICE3 internal models were extracted from the SPICE3 source code, and stored in a (commercial) C++ library [6] [7]. This library was intensively tested by including it as external model code to SPICE3, so it was possible to test the C++ models and the original SPICE3 models in parallel.

The C++ library includes the whole model pool of the semiconductor elements of SPICE3. For each element both a C++ file (*.cpp) and a header file (*.h) exist. The header file of each semiconductor element contains classes with data (parameter and internal data) and declaration of methods. In the C++ file the methods are coded.

Due to the object-oriented principle, a class hierarchy of model components was created. Central base classes contain such values and their methods that, according to SPICE3, are needed in nearly every model, e.g. the nominal temperature. Via inheritance of the base classes their values are provided to other classes. Each functionality that is needed more than one time is coded in a separate function. Consequently, a strongly structured hierarchy of classes was developed.

To simulate a model with the C++ library a SPICE3 typical system of equations is generated (initialization phase) and for each solution step the current data are loaded (simulation phase). For each device of the circuit, model specific methods that are called according to different aspects are supplied. These methods can be disposed under functional aspects as follows:

- Methods to analyse the source code
- Methods to create the linear system of equations
- Methods for instantiation the models and parameters
- Methods to calculate values of the linear equation system
- Methods to insert values into the system of equations

For each model parameter a variable exists, that is called “parameterValue” which gets the value of the particular parameter. For some parameters it is important to know whether they were set by the user or their default values were used. Depending on which case comes into effect, different formulas are used in the further calculation. Even if the value set by the user is the same as the default value, the simulation results differ in some cases. The information if a parameter is set is stored in a Boolean value “IsGiven” (true, if the parameter is set). The “IsGiven” value is analysed by different methods.

The semiconductor devices are modelled by means of a substitute circuit. In this process the different physical effects are allocated at any one time to a component of this circuit. For each of this effects different methods exist, that insert the currents and conductances that are calculated for the actual voltages at the pins, into the linear system of equations. Also equations are arranged for the internal nodes of the substitute circuit. For the calculation also internal values of the integration method are used, e.g. the actual time step size and the history of the calculated values.

In summary the C++ library of the SPICE3 semiconductor elements can be characterized as follows:

- The complete library is according to the semiconductors structured in classes, which contain data and methods.
- For each device methods exist, that achieve the necessary calculations.

- For each element of the SPICE3 netlist, the according classes are instantiated. The needed methods are called for every instance.
- The aim of these calls is to create a linear system of equations to calculate the solution like in SPICE3.
- The parameter handling is special because of the calculations in the initial phase that uses so called “IsGiven” values.
- Internal values of the integration methods are used.

The C++ library which was thoroughly tested is the base for creating SPICE3 models in Modelica.

4 Modelling steps towards Modelica

The C++ functions are constructed to calculate the currents of an equivalent circuit starting with given node voltages. The currents are used inside SPICE3 for filling a linear system of equations.

Starting with the C++ equivalent circuit a Modelica top level model (figure 3) is constructed with electrical pins for connecting. The components of the top level model represent special (e.g. semiconductor) effects (e.g. channel current). Using the pin voltages the components call in their algorithm part typically a hierarchy of functions for the calculation of currents.

There are several steps of modelling semiconductor devices [8], which are described in the following:

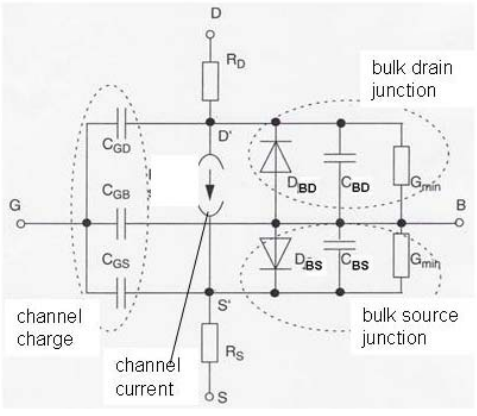

Fig. 3 MOS top level model

1. Construction of top level model

In Modelica every semiconductor device gets a so called top level model which calls the semiconductor functions and can be connected to other models via its connectors. This top level model is the semiconductor device component which will be applied by the user. As in SPICE3 the top level model is adjustable by choosing parameters. Within the top level model the branch currents are calculated using the existing voltages and parameters with the help of functions.

The physical values that are calculated in the C++ semiconductor devices are prepared to be inserted into the linear system of equations like in the SPICE3 simulator. Such a system of equations cannot be addressed in Modelica usually. Only the relation between voltage and current at the interfaces of the model is of interest (terminal behaviour [9]). The voltages at the pins, that are the results of the simulation algorithm, are gripped and given to functions that calculate currents and other values.

The top level model that can be connected and provided with parameters is extracted from the functionality in C++ (figure 3).

2. Parameter handling

The behaviour of a transistor is determined by its parameters significantly. Parameters are e.g. the physical dimension, the temperature or the oxide thickness. Before the simulation the Boolean value “IsGiven” is analysed, which gives the information whether a parameter was set by the user its default value is used.

In the C++ library the parameters are handled as a string. If a parameter is needed when calling a method, the string is searched for a value of the parameter. This way is also possible in Modelica, but it is not usual. In Modelica all parameters are provided in a parameter list, where the user can adjust the parameters. It is desirable that in the Modelica concept a possibility exists that decides whether a parameter is set by the user (“IsGiven”) or not. Unfortunately such a possibility does not exist yet. That is why a temporary solution was chosen. The default value of parameters whose “IsGiven” value is of importance, is set to a very big negative value (-1e40), because that values does make no sense as a normal parameter value. Afterwards it is checked if the value of a parameter is not equal to -1e40. In that case it is assumed that the parameter was given by the user and consequently “IsGiven” is true. Otherwise the parameter gets it default value. This solution is only preliminary and will be improved as soon as Modelica delivers the necessary possibilities. The example

in figure 4 shows the parameter handling with the parameter phi.

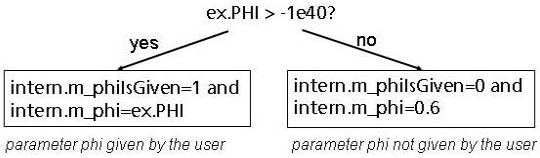

Fig. 4 parameter handling

As described in section two the parameters in SPICE3 are divided into two groups, device and technology parameters. In Modelica the device parameters are part of the semiconductor model. The technology parameters are collected in a record. This record is a parameter for all semiconductor devices. This courses that also the technology parameters can be adjusted in every single model separately which is not intended in SPICE. But in some cases it could make sense.

Furthermore the record with the technology parameters is available in the highest level of the circuit. Every semiconductor device gets the record as a parameter. So the components of the record can be adjusted in a global way for each device in the circuit. Another possibility to provide the technology record as global is to define a model in the circuit level that inherits the properties of the MOSFET where the desired parameters are unchangeably included. Both possibilities force the user to work within the source code. For untrained users it would be better to work in the graphical modus of Dymola and giving each single semiconductor device parameter its value by clicking on the device and inserting the parameter value in the prepared list.

3. Transformation of C++ library data structure

In the C++ source library the data are concentrated in classes and located in the according header file of the semiconductor elements. For each parameter a variable "parameterValue" exists that gets the particular value of the parameter. In Modelica the parameters are concentrated in records because these are the equivalent classes to the C++ classes with the parameters. Records were developed in Modelica to collect and administrate data and to instantiate data all at once. Inside the records the data get their default values. With a function call all data that are located inside a record can be accessed. Parameters that are needed for more than on model are collected in a higher level record which is inherited to the lower level records of the single models (figure 5).

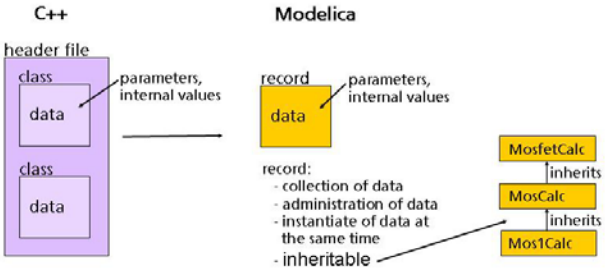

Fig. 5 Transformation of C++ data structure

4. Transformation of C++ library methods

The C++ library of the semiconductor elements of SPICE3 contains beyond parameters and variables that are concentrated in classes, also of a huge number of methods that need to be transformed. Within the transformation it is important, that the structure of the C++ library also remains in Modelica with the aim to recognise the C++ code.

Each semiconductor element in the C++ library becomes to a top level model in Modelica. Inside the top level model functions are called, that calculate both the parameters and the currents at the pins of the model. These functions need to be extracted from C++ and transformed to Modelica. In the C++ library a hierarchy of classes exists where often more than one method calculate one physical effect. Like in a tree structure one method calls another method that itself also calls another method and so on.

The transformation starts with the transfer of the name of the C++ method to the according Modelica function. That function has to be included into a package that has the name of the C++ class where the appropriate method came from. In the second step the parameters and values that are concentrated in classes in C++ are transformed to Modelica into records. In the third step the function text that changes the values in the classes respectively the records has to be directly red of the C++ code and transformed to Modelica where the original C++ names are used. Within that step the C++ code is included into the Modelica code as annotation to recognise the C++ code (figure 6).

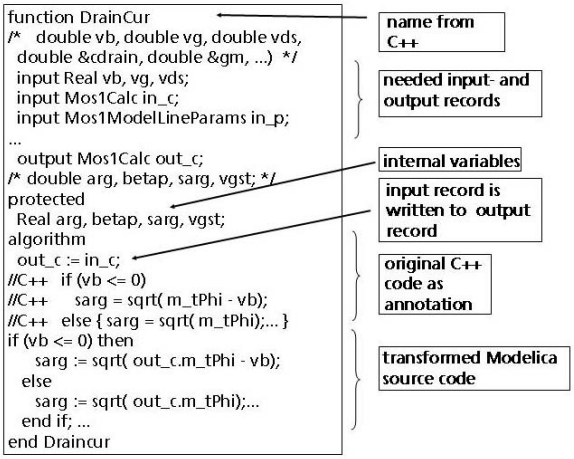

Fig. 6 Transformation of a function

5. Code revision

After a SPICE3 model was transformed into Modelica the source code is checked again with the aim to make is more effective. One point is to include the Modelica operator "smooth". Within this all conditions (if) are checked to find out if it is continuous, also in higher derivations. In that case "smooth" avoids the not needed breaks of the analogue simulation algorithm. With this approach the performance of the simulation can be increased very much.

It also has to be checked if methods were transformed to Modelica that are actually not needed, to simplify the Modelica code.

The system of equations that is built in SPICE3/C++ is not used in Modelica as well as internal values of the integration method that is in connection to the SPICE3 solution algorithm. The calculation of the Jacobians that is done in SPICE3/C++ is also not used in Modelica. It was tried to ensure to transform only the functional aspects of the models to Modelica. In this way a mixture between model equations and numerical solution algorithms like in SPICE3 is avoided.

5 Structure of SPICE3 library

The current SPICE3 library contains the packages Basic, Interfaces, Semiconductors, Sources, Examples, Repository and Additionals (as can be seen in figure 7).

Fig. 7 SPICE3 library overview

The package Basic contains basic elements like resistor, capacitance, inductivity and controlled sources. In the package Sources there are the voltage- and current sources transformed from SPICE3. The package examples include some example circuits, to help the user getting a feeling of the behavior of the library and their elements.

Only the semiconductor models are written using the converted C++ library. The packages Semiconductor and Repository are related to each other very closely. In the package Repository the semiconductor devices and their model cards from SPICE3 are modeled. The necessary function and records are also in this package. This package is not for user access. The semiconductor package contains clearly arranged the offered semiconductor devices and their model card records for easy usage. The user should take the models out of this package. Via inheritance these models are connected to the repository. That's why the user does not have to access to the repository directly.

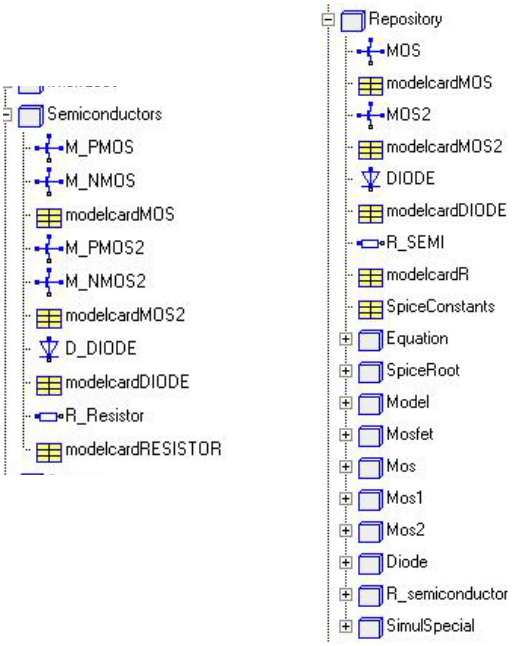

Fig. 8 Packages Semiconductor and Repository

The package Additionals contains the polynomial sources like they are available in SPICE2 or PSPICE. Other models that are not from SPICE3 can be collected here.

6 Example

In this section a Modelica model of the inverter circuit shown in figure 1 is developed. The following two approaches are important.

Graphical composition

The SPICE3 library models are composed and connected with the graphical possibilities of the simulator. Figure 9 shows such a circuit (Dymola).

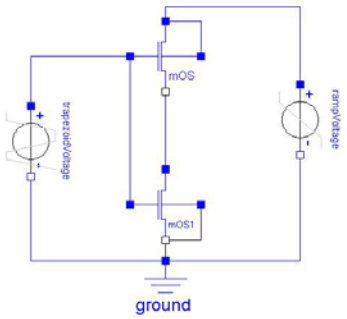

Figure 9: Graphically composed inverter circuit

Textual composition

Starting with the SPICE3 netlist (figure 2) the Modelica inverter model can be generated directly without using graphical information. This feature is important, because the SPICE3 netlists that exist for many circuits, modules and complex circuit elements, should also be available in Modelica. In the following example of two inverters, a way of transforming is shown. First of all the two source codes are opposed to each other.

SPICE3	Modelica
inverter	model inverter
	Spice3.Basic.Ground g;
Mp1 11 1 13 11 + MPmos	Spice3…M Mp1(mtype=true, M(GAMMA=0.37));
Mn1 13 1 0 0 + MNmos	Spice3…M Mn1(M(LAMBDA=0.02, GAMMA=0.37));
Vgate 1 0 PULSE + (0 5 2s 1s)	Spice3…V_pulse vgate(V1=0, V2=5, TD=2, TR=1);
Vdrain 11 0 + PULSE(0 5 0s + 1s)	Spice3…V_pulse vdrain(V1=0, V2=5, TD=0, TR=1);
	Spice3.Interfaces.Pin p_in, p_out;
.model MPmos PMOS + (gamma=0.37)	protected
	Spice3.Interfaces.Pin n0, n1, n11, n13;
.model MNmos NMOS + (gamma=0.37 + lambda=0.02)	equation
	connect(p_in, n1); connect(p_out, n13);
.tran 0.01 5	connect(g.p, n0);
.end	connect(vdrain.n,n0); connect(vdrain.p,n11);

	connect(Mp1.NB,n11);
	connect(Mp1.ND, n11);
	connect(Mp1.NG, n1);
	connect(Mp1.NS, n13);
	connect(Mn1.NB,n0);
	connect(Mn1.ND, n13);
	connect(Mn1.NG, n1);
	connect(Mn1.NS, n0);

	end inverter;

Fig. 10 Inverter model in SPICE and Modelica

The creation of the Modelica texts requires the following steps:

1. The obligate name of the Modelica model can be derived from the first line in the SPICE3 netlist.

2. It is necessary to create entities of each circuit element of the SPICE3 netlist and to provide them with parameters, e.g. the SPICE3 line

```
Vdrain 11 0 PULSE(0 5 0 1)
```

is in Modelica

```
V_pulse vdrain(V1=0,V2=5,TD=0,TR=1);
```

3. For each node number in SPICE an internal pin has to be created in Modelica, e.g. for the node number 2 in SPICE, the Modelica line would be:

```
protected Spice3.Interfaces.Pin n2;
```

The "n" is necessary because in Modelica a single number is not a name.

4. According to the netlist the internal pins have to be connected to the circuit element, e.g.

```
connect (Mp1.ND, n11);
```

5. In the last step the external connectors have to be created and connected to the according internal connectors, e.g.

```
Spice3.Interfaces.Pin p_in, p_out;
connect(p_in, n1); connect(p_out, n2);
```

Concerning the semiconductor elements the model cards have to be transformed to Modelica. Two ways seem to be possible.

Separate record

The records of the technology parameters MPmos and MNmos are instances of the record model card in the model inverter for each transistor (Mp1, Mp2,…).

```
model inverter
  parameter …modelcardMOS Pmos(GAMMA=0.37);
  parameter …modelcardMOS Nmos(LAMBDA=0.02,
                               GAMMA=0.37);
  Spice3.Basic.Ground g;
  Spice3…MOS Mp1(mtype=1,modelcard=MPmos);
  Spice3…MOS Mp2(mtype=1,modelcard=MPmos);
  Spice3…MOS Mn1(modelcard=MNmos);
  Spice3…MOS Mn2(modelcard=MNmos);
  …
end inverter;
```

Extended model

For each technology parameter set a separate model is created. This model extends the transistor M that was defined in Modelica. Within this way the needed technology parameters are given.

```
model inverter
 model MPmos
  Spice3.Semiconductors.modelcardMOS M
                (GAMMA=0.37);
  extends Spice3…MOS(final type=1,
                             modelcard=M);
 end MPmos;
 model MNmos
  Spice3.Semiconductors.modelcardMOS M
             (LAMBDA=0.02, GAMMA=0.37);
  extends Spice3…MOS(final mtype=0,
                             modelcard=M);
 end MNmos;
  Spice3.Basic.Ground g;
  MPmos Mp1;
  MPmos Mp2;
  MNmos Mn1;
  MNmos Mn2;
  …
end inverter;
```

With the help of these two possibilities the user can give many transistors the same technology parameters like it can be done in SPICE3.

The textual composition could be done automatically by a special translator. The aim is to have such a translator in the future, maybe in the Modelica language.

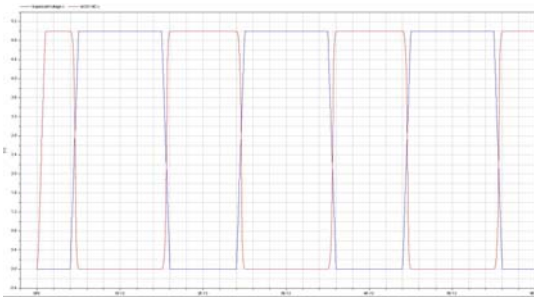

Fig. 11 Inverter simulation result

The result of the Dymola simulation of the inverter circuit is in accordance with the SPICE3 simulation result.

7 Test and Comparison

To verify the transformed models several different test steps were arranged. It is important that the Modelica library is in accordance with SPICE3. Since the C++ library was tested very intensively it can be assumed that it is correct. That is why SPICE3 as well as the C++ library are the base of the tests.

The C++ code was included to the Modelica code as comment. This allows the visual comparison of the source codes.

Single values of currents or other variables (e.g. capacitances) are compared between the Modelica simulation and the simulation of the C++ model library. This approach is very complex and time consuming. Therefore it is only done when the reason of known differences has to be found out.

The terminal behavior is compared between Modelica and SPICE3. Therefore single semiconductor devices are connected to voltage sources to calculate the current-voltage characteristics.

In the next step complex circuits are created with several semiconductor elements and the results are compared between SPICE3 and Modelica. Such circuits are the base for a collection of circuits for regression tests, which are maintained to ensure the correctness of the library in future.

A comparison between the Spice3 library for Modelica and the BondLib in Dymola showed that the two libraries have nearly the same results and performance. For the comparison three circuits were used (NAND, NOR, double Inverter). The following table 1 shows the results in detail:

Tab. 1 Comparison between BondLib and Spice3 Library for Modelica

	NAND		NOR		Inverter	
	SPICE3lib	BONDlib	SPICE3lib	BONDlib	Spice3lib	BONDlib
Before translating						
scalar unknowns	873	10.677	873	10.673	870	10.860
variables	1.157	12.136	1.157	12.132	1.152	12.315
After translating						
parameter depending	8	2.005	8	2032	8	2.030
time-varying variables	826	688	826	687	824	687

As it can be seen in the table 1, the Spice3 library has much less variables then the BondLib before translation of the model. After the model has been translated, the BondLib has little less variables than the Spice3 library. This shows that the simplification algorithms of Dymola work better for the BondLib.

For the double Inverter circuit the output voltage of the original SPICE3 simulator, the BondLib and the Spice3 library for Modelica are shown in the following figures 12/13.

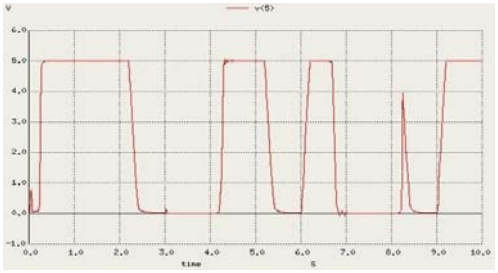

Fig. 12 Output voltage original SPICE3

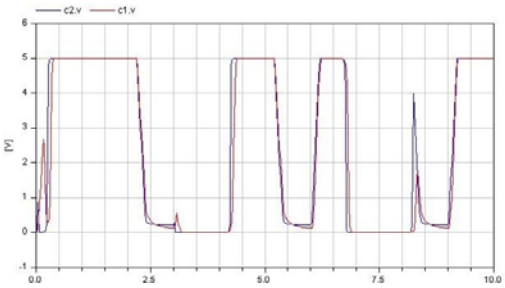

Fig 13 Output voltage BondLib and Spice3 library for Modelica

Each figure shows the output voltage of the second inverter. In figure 12 the result of the original SPICE3 simulator is shown. The results of the three simulators are nearly the same.

8 Conclusions

In this paper a concept was described to transform the procedural implemented SPICE3 models, which are directly extracted from the original SPICE3 source code, to declarative described models for Modelica. Therefore a list of modeling steps was elaborated and applied to transform several semiconductor devices from SPICE3 to Modelica whereas the parameter handling was focused on. The result is a SPICE3 library for Modelica which contains the general devices and first semiconductor devices.

A disadvantage of the Spice3 library compared with the Bondlib is that the Spice3 library has no heatport. At the moment it is possible to simulate with a fixed parameter “Temp”. It has to be figured out how this parameter can be made variable and time dependent in the future.

Further steps for the improvement of the SPICE3 library are:

- developing a method for automatically transforming SPICE3 netlists to Modelica
- increasing the performance of the Modelica models (e.g. application of the smooth operator)
- parameter treatment ("IsGiven") has to be simplified
- adding SI units to the Modelica models
- large number of tests
- testing large circuits (many devices)
- inclusion of further SPICE3 models
- intensively testing, comparison to SPICE3
- inclusion of some PSPICE model features
- comparison with existing electronic libraries
- adding a heatport

Acknowledgement

This research was founded by the European ITEA2, projects EUROSYSLIB and MODELISAR.

References

[1] Clauß, C.; Leitner, T.; Schneider, A.; Schwarz, P.: *Modelling of electronic circuits with Modelica.* Proc. Modelica Workshop, Lund, Sweden, Oct. 2000, 3-11.

[2] Cellier, F.E.; Clauß, C.; Urquia, A.: *Electronic circuit modeling and simulation in Modelica.* EUROSIM 2007, Ljubljana, Slovenia, 9.-13. Sept. 2007.

[3] Urquia, A.; Martin, C.; Dormido, S.: *Design of SPICELib: a Modelica Library for modeling and analysis of electric circuits.* Mathematical and Computer Modelling of Dynamical Systems, 11(1)2005, 43-60.

[4] Cellier, F.E.; Nebot, A.: *The Modelica bond graph library.* Proc. 4th Int. Modelica Conference, Hamburg-Harburg, Germany, 1, 2005, 57-65.

[5] SPICE Version 3e Users Manual, 1991

[6] Leitner, T.: *Entwicklung simulatorunabhängiger Modelle für Halbleiter-Bauelemente mit objektorientierten Methoden.* Chemnitz, Technische Universität, Diss., 1999.

[7] Leitner, T.: *A new approach for semiconductor models basing on SPICE model equations.* Proc. ECS'97, Bratislava, Slovakia, 4./5. 9. 1997, 119-123.

[8] Majetta, K.: *Entwicklung und prototypische Umsetzung eines Konzeptes für eine Modelica-Bibliothek von SPICE-Halbleiterbauelementen und Erarbeitung einer Teststrategie.* Dresden, Berufsakademie Sachsen, Dipl., 2008.

[9] Clauss, C.; Haase, J.; Kurth, G.; Schwarz, P.: *Extended Admittance Description of Nonlinear n-Poles.* AEÜ, Vol. 49 (1995) 2, 91-97.