ELECTRONIC CIRCUIT MODELING AND SIMULATION IN MODELICA

François E. Cellier[1], Christoph Clauß[2], Alfonso Urquía[3]

[1]ETH Zurich, Department of Computer Science,
ETH-Zentrum, CH-8029 Zurich, Switzerland
[2]Fraunhofer Institute of Integrated Circuits, Zeunerstr. 38,
D-01069 Dresden, Germany
[3]National University of Distance Education, Dept. of Computer Science and
Automatic Control, Juan del Rosal 16, 28040 Madrid, Spain

FCellier@Inf.ETHZ.CH (François Cellier)

Abstract

In recent years, three separate efforts took place at three different institutions to provide an electronic circuit modeling and simulation capability within the framework of Modelica, an object-oriented general-purpose environment for the modeling of physical systems. In order to be generally usable, no domain-specific knowledge is hard-coded into the Modelica software. Modelica only understands mathematics, not physics. Consequently, all domain-specific knowledge must be formulated as part of the model. Recent advances in symbolic algorithms and software technology have made it feasible to implement a full-fledged electronic circuit simulator in Modelica without making unacceptable sacrifices on the run-time efficiency of the resulting simulation code. What is being gained in the process is an improved transparency of the models that are being implemented, a significantly improved ease of maintainability and extensibility of the code, and a dramatically improved flexibility in combining electronic models with mechanical and thermal models. These are demands that industry now makes on a circuit simulator, demands that cannot easily be met using the traditional approach to electronic circuit simulation.

Keywords: Electronic circuit simulation, Object-oriented modeling, Modelica, Multi-energy modeling, Bond graph.

Presenting Author's biography

François E. Cellier received his BS degree in electrical engineering in 1972, his MS degree in automatic control in 1973, and his PhD degree in technical sciences in 1979, all from the Swiss Federal Institute of Technology (ETH) Zurich. Dr. Cellier worked at the University of Arizona as professor of Electrical and Computer Engineering from 1984 until 2005. He recently returned to his home country of Switzerland. Dr. Cellier's main scientific interests concern modeling and simulation methodologies, and the design of advanced software systems for simulation, computer aided modeling, and computer-aided design. Dr. Cellier has authored or co-authored more than 200 technical publications, and he has edited several books. He published a textbook on Continuous System Modeling in 1991 and a second textbook on Continuous System Simulation in 2006, both with Springer-Verlag, New York.

ISBN 978-3-901608-32-2

1 Introduction

Traditionally, analog electronic circuits were modeled and simulated using special-purpose electronic circuit simulators. Although a number of such simulation tools have been made available over the years, the one tool that has been most successful in conquering the market is Spice.

Spice operates on a so-called "net-list" that numbers the nodes of an electronic circuit and places the circuit elements in between these nodes. The net-list is interpreted by the circuit simulator. Due to some special characteristics of electronic circuits, it is possible to come up with a highly efficient implementation of a circuit simulator without need for compiling the circuit first.

A number of different software manufacturers compete for the market by offering different implementations of Spice. These implementations vary somewhat in details of their transistor models, and circuit models written for any one of these simulators may not run correctly on another due to differences in the set of device parameters offered by the different implementations.

There exist secondary markets for schematic capture programs, tools that are able to generate a Spice net-list from a graphical circuit description; for output visualization programs, tools that are able to interpret the output generated by Spice and represent them graphically in different formats; and for device parameter sets, tools that offer device parameter sets for a large number of different commercially available transistors.

Circuit simulators have always remained separate and distinct from the general-purpose dynamical systems modeling and simulation (M&S) tools.

The so-called *Continuous System Simulation Languages (CSSLs)* [1] that are based on state-space descriptions of dynamical systems have primarily been driven by the needs of control engineers. It is no accident that the state-space formalism for the mathematical description of dynamical systems was first proposed by a control engineer, R.E. Kalman [2]. It was further no accident that the *Society for Modeling and Simulation International* was originally founded by a group of aerospace engineers.

The electronic circuit simulators of the past remained outside of the mainstream CSSL development, because electronic circuit models do not lend themselves to manual transformation into a state-space form.

Only the advent of modern *object-oriented physical systems modeling languages (OOMLs)* [3] made it feasible to bridge the gap between circuit simulators on the one hand and general-purpose M&S software on the other.

2 CSSLs *vs.* OOMLs

The groundwork leading to today's OOMLs was laid by H. Elmqvist in his Ph.D. dissertation [4].

Dymola was designed as a declarative rather than procedural language based on equations rather than assignment statements. The computational causality of each equation, i.e., the variable for which each equation needs to be solved, is being determined in an automated fashion by the model compiler. Symbolic formula manipulation algorithms are invoked by the compiler to convert implicit (declarative) model specifications to an explicit (procedural) state-space form. The resulting simulation code can then be simulated by an off-the-shelf CSSL-type simulator. The semantic distance between the original implicit model specification and the resulting explicit simulation code may be quite large.

From a historical perspective, OOMLs can be viewed as an evolutionary extension of the former CSSLs. OOML compilers are simply a bit more advanced than CSSL compilers, with the equation handler replacing the former Macro handler.

Yet from a utilization point of view, the OOMLs represent a true revolution as they enable the modelers to treat all physical system models in exactly the same fashion. There is no longer any need to distinguish between a modeling environment for control systems and one for electronic circuits, for example.

Although the compilation of an OOML model of an electronic circuit may be a bit slower than the corresponding compilation using a conventional Spice implementation, the resulting simulation codes can be made equally efficient in execution speed.

Modelica [3] has meanwhile replaced *Dymola* [4] as the most widely used OOML. Whereas the Dymola language had been developed by Elmqvist and marketed by Dynasim as a proprietary code, Modelica has been designed by a standard committee and is non-proprietary. Anyone may develop a modeling compiler and an underlying simulation engine based on the Modelica language specification, and indeed, there already exist several implementations of Modelica.

Modern *Dymola* [5] is one among several M&S environments that are based on the Modelica language specification. Dymola features a graphical user interface (GUI), a Modelica compiler, a CSSL-type simulation engine, a graphical postprocessor for visualization and animation of simulation results, and various additional tools for model manipulation, such as a linearization engine. This M&S environment is a commercial code marketed by Dynasim.

OpenModelica [6] is a free (open-source) implementation of a Modelica compiler. The code comes with a CSSL-type simulation engine. The code also offers a (simple) result visualization program, but many OpenModelica users prefer Matlab [7] for visualization of their simulation results.

ISBN 978-3-901608-32-2

MathModelica Lite [8] can be used as a GUI for OpenModelica.
Both Dymola and OpenModelica have their pros and cons. Industrial users may give a preference to Dymola, because the code is further developed, easier to use, offers more features, is more professionally maintained, and generates more run-time efficient simulation code. Academic users may prefer OpenModelica, because the code is free and because it is open-source, which allows them to develop additional tools and implement additional algorithms more easily than using Dymola.
This paper is based on Dymola, because its emphasis is on Modelica library design and not on the development of new algorithms.

3 Three Modelica libraries for analog electronic circuits

Three different and separate Modelica libraries for the description of analog electronic circuits have meanwhile been developed. These shall briefly be described.

3.1 The Modelica standard library

Modelica denotes not only a language definition, but also a library of models coded in the Modelica language. This Modelica library, which is in the public domain and which is supposed to run under any complete Modelica implementation, is called the *Modelica standard library*.
The Modelica standard library contains a sub-library featuring models of electronic circuit components, which itself contains two sub-libraries, one for digital (logic) circuit elements, the other for analog circuit components.
The electrical analog library [9] contains models of the basic passive electrical components (resistors, capacitors, and inductors), simple models of transistors and diodes, as well as models of voltage and current sources.
The transistor models of the standard library are considerably simpler than those provided in implementations of Spice. Consequently, these models aren't suitable for integrated analog circuit design.
Also, Spice implementations offer different analysis types, in particular, DC-analysis, AC-analysis, and transient analysis. Dymola is designed for transient analysis only.
On the other hand, the simplicity of the transistor models contained in the standard library also has its beauties. These models can be simulated very effectively, which allows the efficient simulation of large transistor circuits. Also, the simplicity of these models may be appealing to instructors, who wish to teach students the basics of analog circuit design.
Finally, the standard library also offers thermal analog circuit component models, which allows circuit designers to model and simulate heat storage and dissipation in electronic circuits, which cannot be done in Spice, and also, since electrical circuit models can be simulated together with mechanical system models, Dymola makes it possible to model and simulate mechatronic systems elegantly and efficiently.
A simple analog inverter circuit may serve to illustrate the modeling and simulation of electronic circuits using the Modelica standard library. The circuit diagram is shown in Fig. 1.

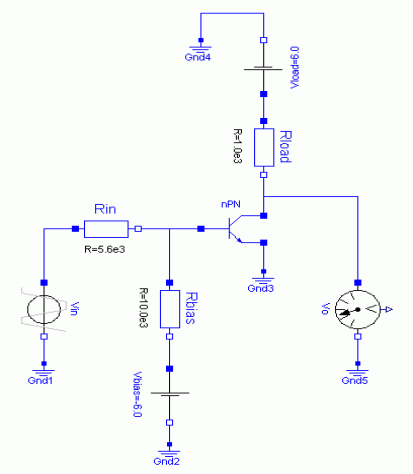

Fig. 1 Circuit diagram of analog inverter circuit

The circuit diagram looks essentially the same as a corresponding circuit diagram drawn using any schematic capture program designed to be used as a preprocessor to Spice. The circuit contains a single bipolar junction transistor (BJT).
The BJT model offered in the Modelica standard library is a simple equation model as shown in Fig. 2.

```
equation
  ExMin = exp(EMin);
  ExMax = exp(EMax);
  vbc = B.v - C.v;
  vbe = B.v - E.v;
  qbk = 1 - vbc*Vak;

  ibc = if (vbc/Vt < EMin) then Is*(ExMin*(vbc/Vt - EMin + 1) - 1) + vbc*Gbc else
          if (vbc/Vt > EMax) then Is*(ExMax*(vbc/Vt - EMax + 1) - 1) + vbc*
    Gbc else Is*(exp(vbc/Vt) - 1) + vbc*Gbc;
  ibe = if (vbe/Vt < EMin) then Is*(ExMin*(vbe/Vt - EMin + 1) - 1) + vbe*Gbe else
          if (vbe/Vt > EMax) then Is*(ExMax*(vbe/Vt - EMax + 1) - 1) + vbe*
    Gbe else Is*(exp(vbe/Vt) - 1) + vbe*Gbe;
  Capcjc = if (vbc/Phic > 0) then Cjc*(1 + Mc*vbc/Phic) else Cjc*pow(1 - vbc
    /Phic, -Mc);
  Capcje = if (vbe/Phie > 0) then Cje*(1 + Me*vbe/Phie) else Cje*pow(1 - vbe
    /Phie, -Me);
  cbc = if (vbc/Vt < EMin) then Taur*Is/Vt*ExMin*(vbc/Vt - EMin + 1) +
    Capcjc else if (vbc/Vt > EMax) then Taur*Is/Vt*ExMax*(vbc/Vt - EMax + 1)
     + Capcjc else Taur*Is/Vt*exp(vbc/Vt) + Capcjc;
  cbe = if (vbe/Vt < EMin) then Tauf*Is/Vt*ExMin*(vbe/Vt - EMin + 1) +
    Capcje else if (vbe/Vt > EMax) then Tauf*Is/Vt*ExMax*(vbe/Vt - EMax + 1)
     + Capcje else Tauf*Is/Vt*exp(vbe/Vt) + Capcje;
  C.i = (ibe - ibc)*qbk - ibc/Br - cbc*der(vbc) + Ccs*der(C.v);
  B.i = ibe/Bf + ibc/Br + cbc*der(vbc) + cbe*der(vbe);
  E.i = -B.i - C.i + Ccs*der(C.v);
end NPN;
```

Fig. 2 BJT model of Modelica standard library

Contrary to Spice, which "hard" encodes its device models either in Fortran or C (depending on the version), Modelica "soft" encodes the entire domain knowledge of its models in the Modelica language itself. The Modelica compiler only encapsulates mathematical knowledge. Consequently, Modelica models are considerably easier to understand, maintain, and upgrade than Spice models.

ISBN 978-3-901608-32-2

In Dymola, each object (model) is represented by four different entities. It consists of an iconic (graphical) representation that is used to allow the model to be graphically invoked at a hierarchically higher level; a diagram (graphical) representation that is used to represent the internal structure of the model by an interconnected set of sub-models; an equation (textual) representation that allows the model to be described using (implicitly formulated) equations; and a documentation (hypertext) representation that enables the modeler to describe what the model is supposed to be doing.
In the above example, the inverter circuit is described using the diagram layer, whereas the BJT model is described using the equation layer.
The inverter circuit model is now converted to state-space form by the Modelica compiler. This process is shown in Fig. 3.

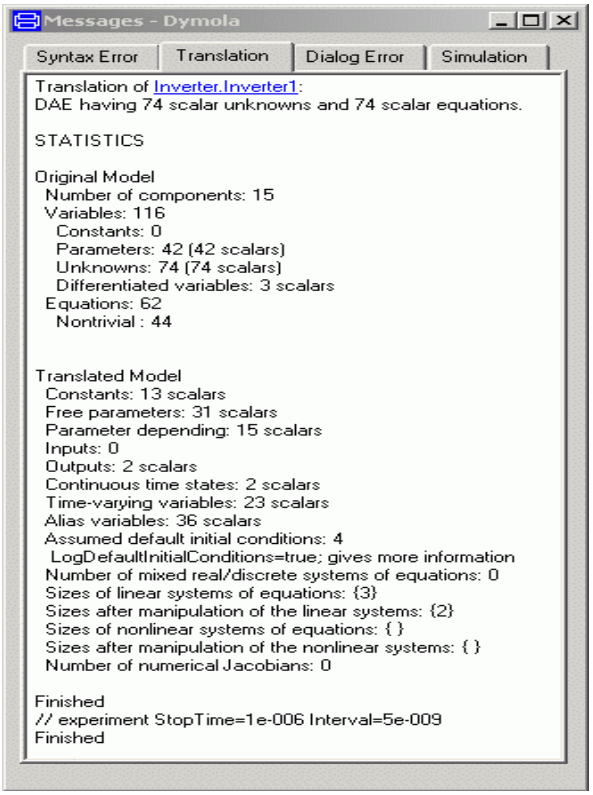

Fig. 3 Translation of analog inverter circuit model

The original (implicit) DAE model contains 74 equations in 74 unknowns. The translated (explicit) ODE model contains 2 state variables and 23 algebraic variables. The remaining equations were trivial (connection) equations that were eliminated by the compiler.
The model can now be simulated. This process is shown in Fig. 4.
The Dymola simulator uses by default the DASSL numerical ODE solver. The entire simulation required 0.04 sec for its execution.
The simulation results are shown in Fig 5.

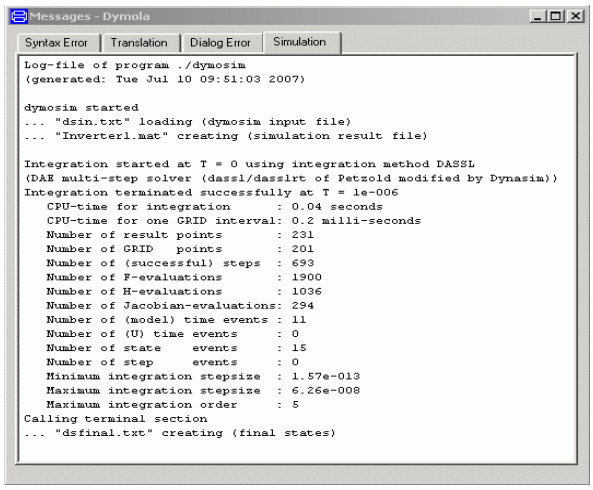

Fig. 4 Simulation of analog inverter circuit model

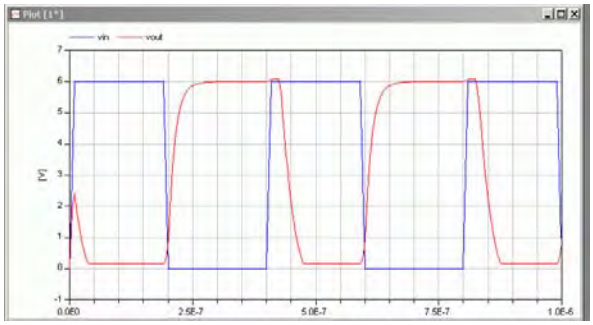

Fig. 5 Results of analog inverter circuit simulation

During the initial period, the circuit undergoes a transient, because the model didn't start out in a trimmed state. In Spice, the user would normally precede the transient analysis by computing a DC OP point, which would eliminate the initial transient from the simulation results.

3.2 SPICELib

Beside from the Modelica standard library, a number of Modelica users have written libraries of their own. Some of these libraries are commercial libraries, whereas others are free libraries.
SPICELib [10] is one such free library. It implements a subset of component models of PSpice [11]. PSpice is one of the most widely used dialects of Spice. Consequently, many modelers are familiar with the device parameters featured in PSpice, and there have been identified PSpice parameter sets for most commercially available electronic devices.
SPICELib follows the traditional Spice philosophy of offering separate models for the different analysis types, i.e., one SPICELib component model contains multiple models, one for DC analysis, another for AC analysis, and a third for transient analysis.
As a consequence of this decision, the SPICELib connectors are incompatible with the connectors of the standard electrical library, and therefore, SPICELib and standard component models can unfortunately not be mixed.

ISBN 978-3-901608-32-2

Just as in the case of the standard library, the SPICELib component models have been implemented using the equation layer. However, these models are now considerably more complex, and therefore, they are less easy to understand and maintain. On the upside, these are full-fledged Spice models that can be used for integrated circuit design. Also, the similarity between PSpice and SPICELib and the availability of separate models for the different analysis types makes it particularly convenient for experienced PSpice users to switch to Modelica if they so choose.
However at the current time, our demonstration circuit cannot yet be simulated in SPICELib, because as of now, SPICELib offers MOSFET models only. The BJT and JFET models have not yet been released.

3.3 BondLib

BondLib [12] is another free Modelica library that is built on bond graph technology. Bond graphs [13] offer domain-independent graphical modeling. Bond graphs represent the most basic graphical modeling paradigm that is still fully object-oriented. By mapping higher-level models first down to a bond graph layer, the need for modeling using the equation layer can be dramatically reduced, which enhances the understandability of the models and simplifies their maintenance.
BondLib contains an electrical analog sub-library that is quite similar in appearance to the corresponding sub-library of the Modelica standard library. However, each of the component models is further graphically decomposed down to a bond graph layer.
Contrary to the standard library, BondLib contains also true Spice models. Its Spice models are however based on HSpice [14], another dialect of Spice with device parameter sets that differ somewhat from those used in PSpice. HSpice and BondLib offer different levels of complexity of their device models. Higher level models are more complex, offer more device parameters, and consequently are slower in their execution.
Just like the standard library, BondLib contains transient analysis models only. For this reason, the electrical BondLib connectors are compatible with the electrical connectors of the standard library, and the models of the two libraries can be freely mixed.
Fig. 6 shows our simple demonstration circuit, now modeled using BondLib component models. The model looks quite similar to that of Fig. 1. However in BondLib, the BJT models contain an additional (green) port denoting the substrate, and all device models contain an additional (red) thermal port.
Fig. 7 shows the internal representation of the BJT model. The BJT model is composed of an internal BJT model plus the collector, emitter, and (non-linear) base connection resistances, the base-substrate junction diode (in the case of a laterally diffused BJT), and a non-linear stray capacitance placed between the external base and the internal collector nodes.

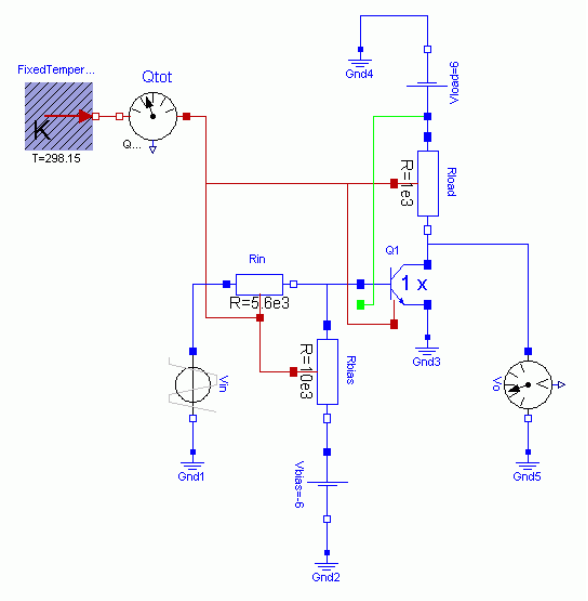

Fig. 6 Analog inverter model coded in BondLib

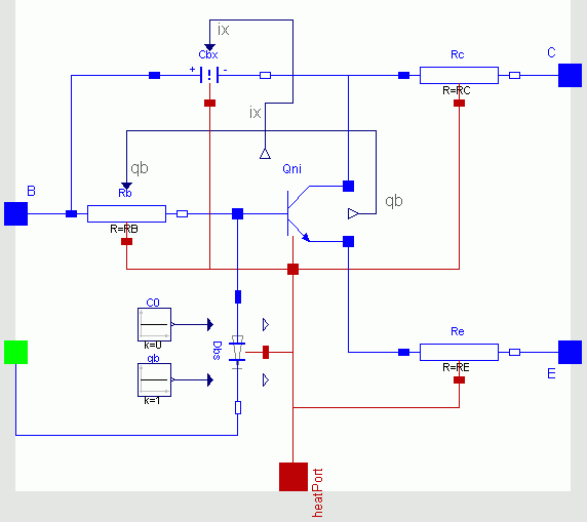

Fig. 7 Internal representation of BondLib BJT model

Fig. 8 shows the internal representation of the internal BJT model.

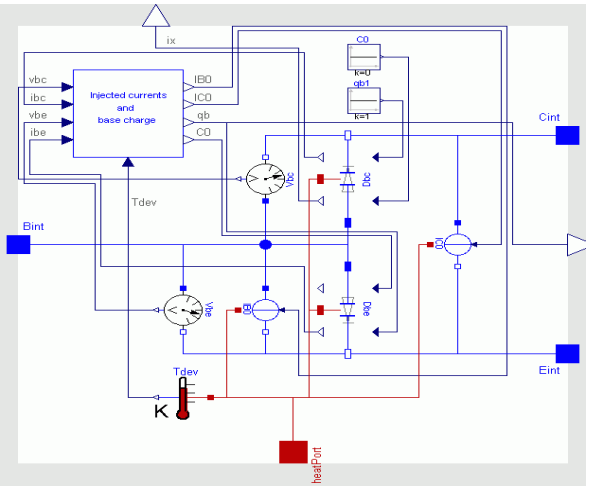

Fig. 8 Internal representation of internal BJT model

Fig. 8 shows the typical replacement circuit of the BJT consisting of two junction diodes and two non-linear current sources.
Fig. 9 shows the internal representation of the base-collector junction diode.

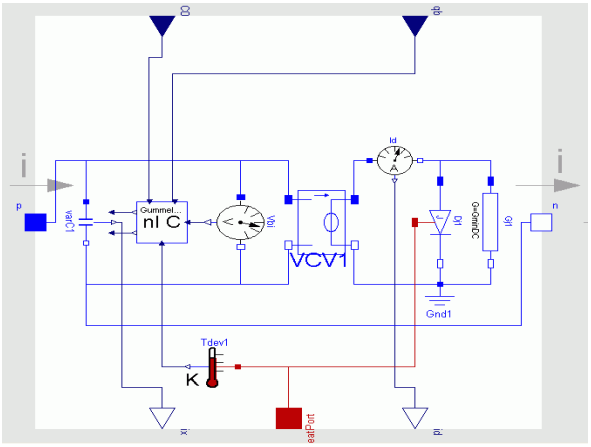

Fig. 9 BondLib model of base-collector junction diode

It consists of the typical non-linear Gummel-Poon junction capacitance, the true diode characteristic, and a (numerically motivated) stray conductance.
At the next lower (still graphical) level are the bond graph device models. Below that level are small equation models implementing the bond graph primitives.
Fig. 10 shows the translation of the BondLib analog inverter circuit.

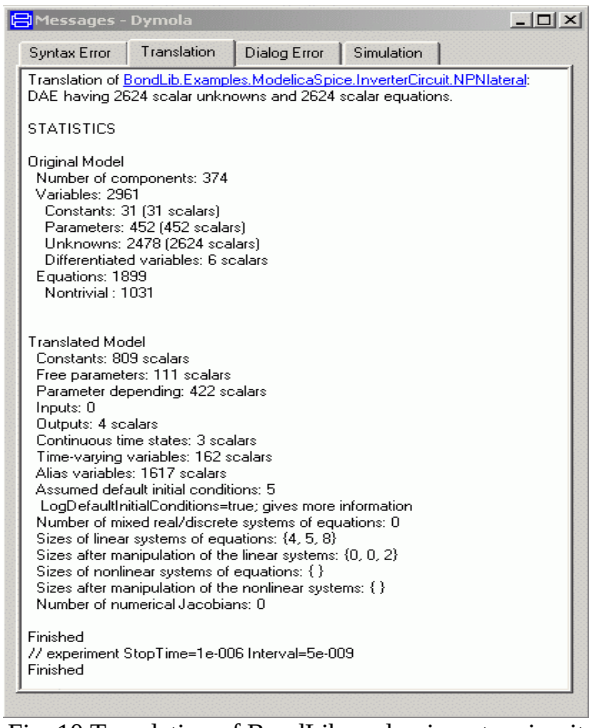

Fig. 10 Translation of BondLib analog inverter circuit

The model contains initially 2624 equations in 2624 unknowns. However, most of those equations are trivial connection equations that are being eliminated in the process of compilation. The simulation model contains 3 state variables and 162 algebraic variables. It is about as simple as can be expected of such a complex model.
Each of the three junction diodes contains one junction capacitance leading to one state. The external stray capacitance is a dependent capacitance that does not lead to an additional state variable. It gets eliminated in the compilation process by symbolic index reduction.
The standard BJT model contains only two state variables, because it doesn't include a model of the substrate.
The compilation of the BondLib model is evidently slower than that of the standard model, but Dymola is very efficient in its compilation. Both compilations consume only splits of a second.
Fig. 11 shows the simulation of the BondLib analog inverter circuit.

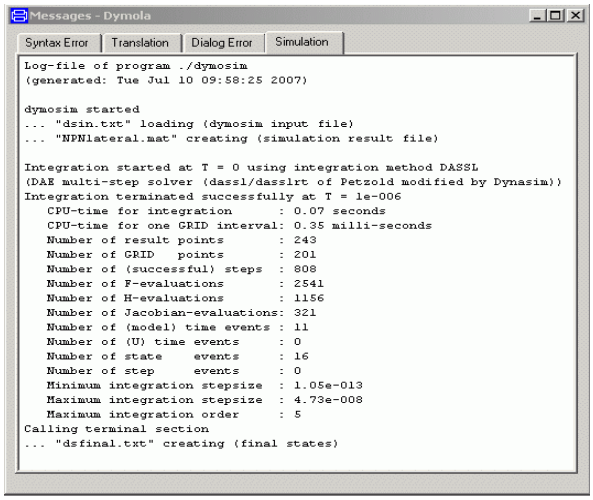

Fig. 11 Simulation of BondLib analog inverter circuit

Also the simulation is slower than that of the equivalent circuit using the standard library, which is to be expected since the model is considerably more complex. However, the slow-down of the simulation is less than a factor of 2, which is quite acceptable.
Fig. 12 shows the simulation results obtained for this model.

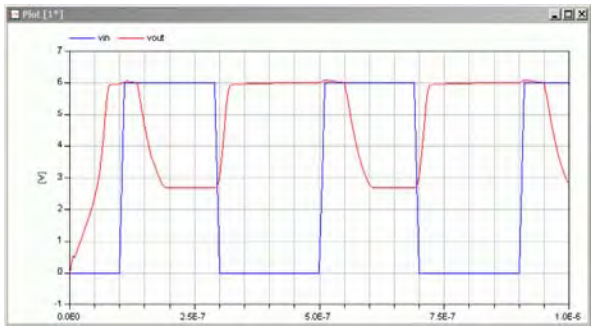

Fig. 12 Results of BondLib analog inverter simulation

The simulation results are quite different from those obtained using the standard library for two reasons.

Firstly, the BondLib BJT model contains (in accordance with Spice philosophy) the base, collector, and emitter connection resistances, whereas the BJT model of the standard library does not. Those would have to be added externally. Their influence should not be neglected. Especially the base resistor is quite large (around 1000 Ohm).
Secondly, BondLib offers some support for DC analysis, which the standard library does not. Traditionally in Spice, a DC OP point is computed by Newton iteration. BondLib doesn't do that, but instead computes a DC OP point by ramping up the sources.
If all active devices are initially switched off and if all sources are initially at zero, the only physically plausible solution is for all voltages and currents to be zero. Thus, we can start with this solution, ramp up all sources to their initial values, and keep them there for a while, before the transient analysis starts.
BondLib offers ramping sources that allow the user to specify a ramping and a settling time for each one of them.

4 AC analysis in BondLib

We shall now analyze a different circuit, namely a high-gain operational amplifier circuit. The overall model is shown in Fig. 13. The inverter itself is shown in Fig. 14.

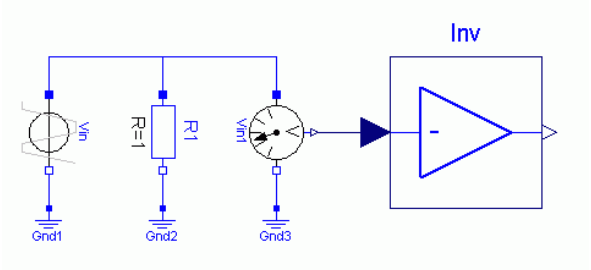

Fig. 13 Overall operational amplifier inverter circuit

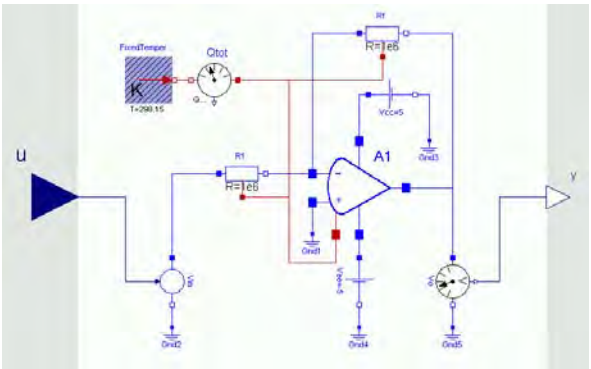

Fig. 14 High-gain operational amplifier circuit

The internal representation of the operational amplifier is shown in Fig. 15. It consists of a dozen BJTs.

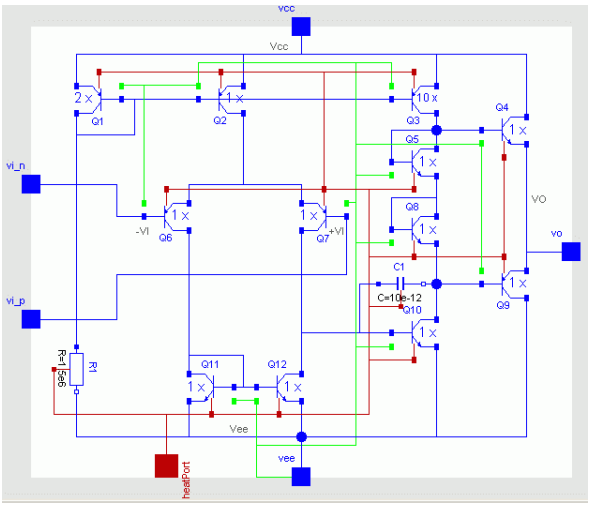

Fig. 15 Model of high-gain operational amplifier

The circuit can be translated and simulated. Fig. 16 shows the translation process, whereas Fig. 17 shows the simulation.

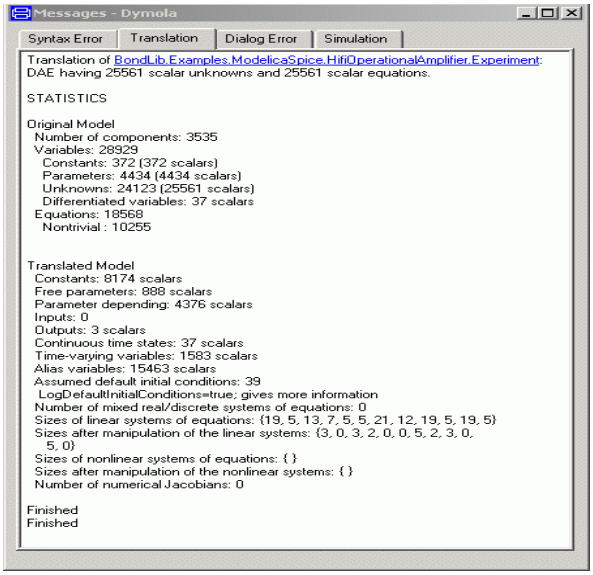

Fig. 16 Translation of operational amplifier circuit

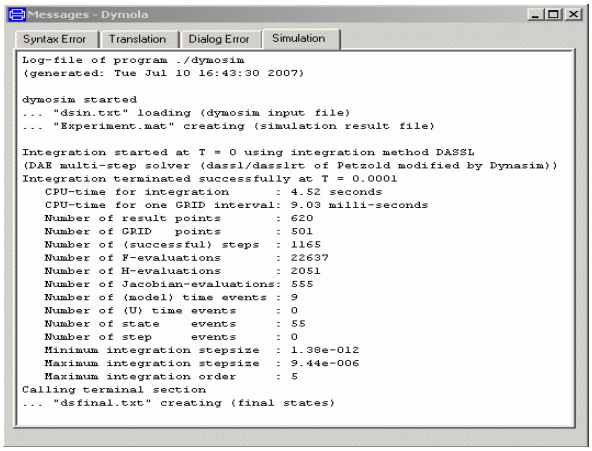

Fig. 17 Simulation of operational amplifier circuit

Here we started out with 25561 equations. The compilation required a few seconds. After the compilation, we ended up with 37 state variables (12 transistors with 3 states each plus one external capacitor) and 1583 algebraic variables. The simulation required 4.52 sec of execution time.
Fig. 18 shows the simulation results.

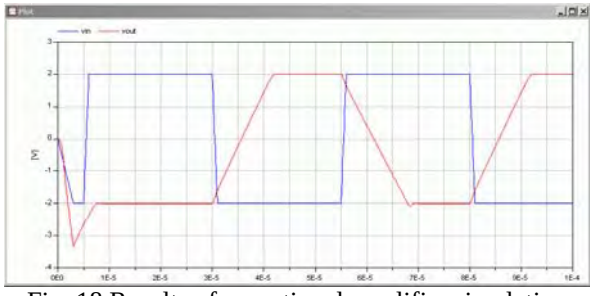

Fig. 18 Results of operational amplifier simulation

We now wish to perform an AC analysis. To this end, we require an input/output model that needs to be linearized around a suitable steady state point.
The final state of the transient simulation is a good steady state point. It is being stored by Dymola automatically in the file `dsfinal.txt`.
The input/output model to be linearized is the model of Fig. 14. Hence we need to quickly return to the modeling window, select the model of the operational amplifier circuit, and return to the simulation window.
We now need to read in the final values of the transient analysis as initial states. This is done in the simulation pull-down menu using the command *Continue/ImportInitial…* that asks you to select the file to be read, which is the file `dsfinal.txt`. Next we need to linearize the circuit around its initial state. This is also done in the simulation pull-down menu using the command *Linearize*. This command stores the linearized model in the file `dslin.mat`.
The actual AC analysis is not performed in Dymola itself, but rather using the control systems toolbox of Matlab. To this end, we execute the file `OpAmp.m` shown in Fig. 19.

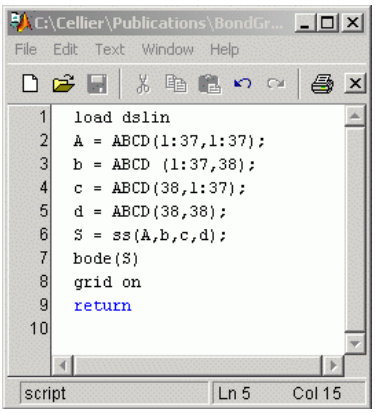

Fig. 19 Matlab code to perform AC analysis

The results of the analysis are shown in Fig. 20.

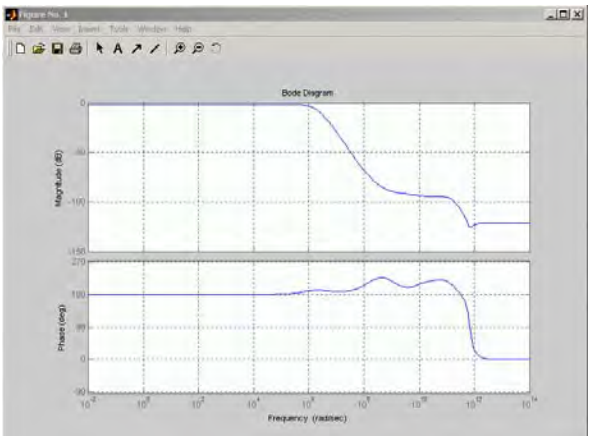

Fig. 20 Results of AC analysis of OpAmp circuit

The operational amplifier has a bandwidth of roughly 1 MHz. Up to that frequency both the amplitude and phase characteristics are beautifully flat.
The approach described here is not specific to BondLib. The same approach could have been applied just as easily when using the standard library for modeling the high-gain operational amplifier circuit. However, the approach is specific to the Dymola M&S environment. There is nothing in the specification of the Modelica language that supports linearization. Hence other implementations of Modelica either won't support this feature at all, or at least, they won't support this feature in the same fashion.
The proposed approach is more object-oriented than that advocated in SPICELib, because the same model can be used to perform all three analysis types. This simplifies the maintenance of the code. However, the approach implemented in SPICELib is more user-friendly, because it enables modelers to perform AC analysis with a single command just like in Spice.

5 Advantages of using OOMLs for modeling electronic circuits

Using OOMLs for modeling and simulation of electronic circuits offers a number of striking advantages over the use of Spice. These shall be enumerated now.

5.1 Mechatronics

Ever more frequently, industry now demands the co-simulation of electronic and mechanical subsystems. Car manufacturers want to be able to simulate the electronic control circuitry of their engines together with the engines themselves, for example. The field has become so prominent that it even received a name of its own: *mechatronics*.
In Spice, mechatronic systems cannot be simulated because Spice doesn't allow modeling systems other than analog electronic circuits. Using an OOML, such as Modelica, M&S of mechatronic systems is natural and highly intuitive.

5.2 Thermal analysis

With the ever increasing density and speed of integrated electronic circuits, heat dissipation becomes a major issue.
Whereas Spice allows simulating an electronic circuit at different temperature values, it doesn't provide for a feature to compute the heat produced by a circuit and from there determine its operational temperature.
Using an OOML and especially using one that is based on bond graph technology, performing thermal analysis of electronic circuits is a breeze.

5.3 Software upgrades

One of the authors of this paper used to work for years as a consultant to BurrBrown, a Tucson-based company specialized in those years in the design of high-gain analog amplifiers, like the one shown in this paper. To this end, he maintained their version of HSpice, called BBSpice.
Once he was asked to add a Zener diode model to the code. It took him roughly two full weeks of work to implement the model in the code. On another occasion, he was asked to provide an automated ramping feature, because the built-in DC analysis frequently did not converge on their analog circuits. Implementing this feature in the code required almost a full month of work.
Using a language like Modelica, similar requests can be fully handled in a few hours each due to the fact that Modelica stores its domain-specific knowledge in the model, rather than in the compiler.

5.4 Teaching

Whereas Spice can be used to test circuits that have been designed, Spice cannot be used easily to teach how transistor models work.
The reason is that the transistor models in Spice are black-box models that can only be looked at from the outside.
It is true that many Spice manuals show the equivalent replacement circuits on which the transistor models are based, but the student has no access to the internal signals of these replacement circuits.
Making use of an OOML, especially when embracing the approach that BondLib took, the student can actually view the replacement circuits and can dig into them to any level that he or she so chooses, even down to the bond graph models at them bottom of the graphical model hierarchy.
This makes it possible to use BondLib effectively and efficiently for teaching the basics of semiconductor technology.

6 Summary

Modelica has become an excellent vehicle for communicating knowledge about dynamical systems among scientists and engineers. Models can be freely interchanged and may be conveniently combined for simulation of mixed energy systems.
Ever more researchers are making use of the language for many of their M&S needs. An international Modelica conference [15] is being held in frequent intervals, where researchers can present their newest models and exchange information amongst each other. At the last of these events, close to 200 people were in attendance.
The Modelica standard library [16] has grown impressively over the years. It contains sub-libraries for electrical, mechanical, and thermal systems, as well as a block diagram library for the description of control systems, a state graph library for the description of discrete systems, a media library for modeling systems with convective flows, as well as a mathematical library for frequently used mathematical functions, and a physical library for universal constants, commonly used measurement units, and conversions between them. The standard library represents easily the largest vault of freely available public domain information concerning dynamical systems. It is envisaged to become the most important one-place-shop-all tool for physical systems modelers around the globe.
Also available from the Modelica website are growing numbers of free libraries, including SPICELib and BondLib that are maintained outside the standard library. These libraries are expected to undergo frequent modifications and enhancements. For example, BondLib shall soon be enhanced by a sub-library for translational and rotational 1D mechanical systems, whereas SPICELib shall soon add models for BJTs. Once a free library has matured and stabilized, it may be considered by the Modelica standard committee for inclusion as a sub-library within the standard library.
Finally, there exist a number of commercial libraries that their producers consider too valuable to be given away for free.
This paper has shown that the OOML approach to physical system modeling has been able to break down the former barriers among hitherto separate and disjoint tools for describing particular classes of physical systems, such as electronic circuits. Modelica models of analog electronic circuits can be and have been developed that are as robust and as efficient as the specialized Spice codes of the past, and yet, much more needs to be done.
Until now, we have concentrated our efforts on analog circuits primarily. Although the standard library already contains a digital electronic library as well, that library is still rudimentary. It cannot compete with commercial tools, like LogicWorks [17] for example, that consider the fan-in and fan-out of digital logic components, allow the simulation of propagation delays across gates, and contain models for TTL and ECL chips currently on the market.
There is also a definite need for the simulation of mixed analog and digital circuits, such as switched power converter circuits, that neither Spice nor LogicWorks can currently handle.

ISBN 978-3-901608-32-2

7 References

[1] D.C. Augustin, M.S. Fineberg, B.B. Johnson, R.N. Linebarger, F.J. Sansom, and J.C. Strauss. The SCi Continuous System Simulation Language (CSSL). *Simulation*, 9:281-303, 1967.

[2] R.E. Kalman. Mathematical description of dynamical systems. *SIAM J Control*, 1:152-192, 1963.

[3] H. Elmqvist, S.E. Mattsson, and M. Otter. Modelica – a language for physical system modeling, visualization and interaction. *Proc. IEEE Intl. Symp. Computer Aided Control System Design*, Kohala Coast, HI, 630-639, 1999.

[4] H. Elmqvist. A structured model language for large continuous systems. Ph.D. dissertation, Dept. of Automatic Control, Lund Institute of Technology, Sweden, 1978.

[5] D. Brück, H. Elmqvist, H. Olsson, and S.E. Mattsson. Dymola for multi-engineering modeling and simulation. *Proc. 2nd Intl. Modelica Conf.*, Oberpfaffenhofen, Germany, 55.1-55.8, 2002.

[6] A. Pop, P. Fritzson, A. Remar, E. Jagudin, and D. Akhvlediani. OpenModelica development environment with Eclipse integration for browsing, modeling, and debugging. *Proc. 5th Intl. Modelica Conf.*, Vienna, Austria, 2:459-465, 2006.

[7] Mathworks. *Using Matlab version 7*. The Mathworks Inc., Natick, MA, 2005.

[8] http://www.mathcore.com/products/mathmodelica/lite/.

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

[10] A. Urquía, C. Martín, and S. Dormido. Design of SPICELib: a Modelica library for modeling and analysis of electric circuits. *Mathematical and Computer Modelling of Dynamical Systems*, 11(1):43-60, 2005.

[11] J.G. Tront. PSpice for basic circuit analysis. McGraw-Hill, 2005.

[12] F.E. Cellier and À. Nebot. The Modelica bond graph library. *Proc. 4th Intl. Modelica Conf.*, Hamburg-Harburg, Germany, 1:57-65, 2005.

[13] F.E. Cellier. Continuous system modeling. Springer-Verlag, 1991.

[14] Synopsys. *HSpice simulation and analysis user guide*. Synopsys Inc., Mountain View, CA, 2003.

[15] http://www.modelica.org/events/modelica2008/modelica2008announcement/view.

[16] http://www.modelica.org/libraries.

[17] http://www.capilano.com/LogicWorks/.

