An Approach to Analyze and Improve the Simulation Efficiency of Complex Behavioral Models

Daniel Platte
Infineon Technologies AG
daniel.platte@infineon.com

Ralf Sommer
Infineon Technologies AG
ralf.sommer@infineon.com

Erich Barke
University of Hannover/Germany
barke@ims.uni-hannover.de

ABSTRACT

Symbolic analysis [1] offers good opportunities for automated bottom-up generation of analytic models for nonlinear analog circuits. Unfortunately, the simulation performance for this kind of models is often a crucial issue. This paper focuses on detailed performance analyses of the behavioral simulation efficiency. Furthermore, promising approaches to improve the simulation performance by "simulator-friendly" formulation of the behavioral models and improvements of the simulation algorithms will be presented.

1. INTRODUCTION

Simulation performance is an important criterion for efficient verification of nonlinear analog circuits. A promising approach to reduce simulation effort is the application of bottom-up generated behavioral models. As manual modeling is time-consuming and error-prone an automated approach is highly desirable. Symbolic analysis turned out to be a flexible and efficient technique to derive accurate behavioral models from a circuit design. The resulting analytic models are of high accuracy, can be easily parameterized, and accurately represent the circuit's behavior also including higher order effects.

In this research, a model generation technique based on the symbolic analysis tool Analog Insydes [2] is applied. It supports a highly efficient model reduction technique. The derived models are based on Differential Algebraic Equations (DAEs) of exceptional high complexity - impossible to be set up manually. Unfortunately, the simulation performance of those models is not yet satisfactory. In many cases it is even lower than the performance of the original netlist-based simulation. The presented approach is based on the assumption that the performance suffers from missing consideration of the applied simulation algorithms as well as from the ability of simulators to deal with such complex behavioral models as efficiently as with netlist-based simulations.

In Section 2, detailed performance analyses for behavioral models will be presented to analyze possible shortcomings. Sections 3 and 4 propose approaches to enhance the simulation efficiency by changes to the behavioral simulator and by improved model formulation. Finally, Section 5 summarizes the presented research.

2. PERFORMANCE ANALYSES

Before presenting the analyses results the applied modeling technique and the conditions of the measurements will be discussed. To equitably analyze simulation performance a clearly defined reference and simulation environment is necessary. As the simulation performance of bottom-up generated models has to be analyzed, the most appropriate reference is the netlist-based simulation of the original circuit the model was derived from.

To make this comparison as accurate as possible a symbolic modeling technique was used to generate behavioral models containing DAEs similar to those of the netlist-based (transient) analysis (see Section 2.1 for details). Both netlist-based and behavioral simulation were evaluated under equal conditions to exactly measure the computational overhead of the behavioral simulation. All analyses have been performed with nonlinear dynamic models and transient simulations. The evaluation of the results relies on characteristic parameters of the simulations:

- Dimension of the linear equation system → Matrix size
- Sparsity of the Jacobian matrix → Number of nonzero elements to process
- CPU time for transient analysis → Over-all performance
- Profiling data → Detailed information on distribution of computational effort
- Number of time steps → Potential differences in time step control
- Total number of Newton iterations (of the transient analysis) → Convergence

These criteria provide a basis to benchmark behavioral vs. netlist-based simulations. Furthermore, profiling is very appropriate to identify possible bottlenecks and shortcomings in terms of simulation performance.

2.1. Modeling Flow

The motivation for the presented research is the astonishing

0-7803-9742-8/06/$20.00 © 2006 IEEE.

low performance of behavioral models generated by a promising modeling flow that will be briefly discussed in this subsection. The model generation process is based on the symbolic analysis tool Analog Insydes [2]. This powerful tool offers the functionality to automatically set up circuit equations for a circuit netlist and to use them as basis for a behavioral model. Symbolic device models (corresponding to the simulator's built-in device models) are used to make the strategy as accurate as the circuit simulation itself. The circuit equations are usually set up in an extended modified nodal analysis (MNA) for nonlinear equations. The resulting dynamic nonlinear equations contain the network equations in MNA (as used in most circuit simulators) as well as the nonlinear element relations resulting from symbolic device models.

Model reduction methods can be applied to reduce the complexity of the equations by term reduction techniques [3]. The benefit of this symbolic approximation technique is to ensure a user-specified accuracy. Hence, this is one of very few methods that allows satisfying a predefined accuracy of the resulting model. As the equations' complexity decreases while the resulting error increases with the degree of model reduction, it is up to the user to find a suitable trade-off between size and accuracy of the model. Experiments show that for reasonable error margins (5-10%) the complexity can be reduced very efficiently yielding a performance improvement of about factor 10 to 100.

Finally, the behavioral model can be generated from the DAEs by using Analog Insydes' model export function. It is able to generate several AHDLs (VHDL-AMS, Verilog-A, etc.) and hence supports the creation of models for almost every behavioral simulator.

To achieve behavioral models that are one-to-one comparable to netlist-based models no model reduction was applied in this research. All models are 100% accurate, fully pin compatible, and have been automatically derived from a circuit netlist. For lack of space in this paper, neither schematics and waveforms nor equation sets and Jacobian matrices of the presented experiments can be presented.

2.2. Simulation Environment

By assuring these preconditions, the problems of calculating a transient response for the netlist-based as well as the behavioral simulation are (nearly) equivalent. "By nature", the SPICE-like topological way of solving the problem is highly efficient and is therefore also the (theoretical) optimum for the behavioral simulation performance under the described circumstances. As in the behavioral simulation the nonlinear element relations have to be solved in almost the same way as the network equations, a "smart preprocessing" as usually done by device models to simplify the solving process is impossible.

The algorithmic differences between both simulation types are mainly related to the assembly of the Jacobian matrix and the right-hand side (RHS) - the so-called loading process. During loading, the linearized equation system that is needed for the Newton iteration is derived. Furthermore, variable tolerances and differences in the time-step control have to be taken into account.

Ensuring equal simulation environments for the experiments will allow a direct comparison of the results for the netlist-based and behavioral simulations:

- Same simulator, testbench, transient inputs and analysis
- Default simulator options (if not mentioned)
- CPU time of each experiment measured on the same processor (arithmetic mean value of 10 simulations to eliminate load variation)
- All characteristics extracted from simulator protocol files

The analyses have been repeated with various common simulators. In this publication, the focus will be on Infineon's in-house-simulator TITAN [4,5]. This SPICE-like analog circuit simulator supports behavioral simulation using a subset of VHDL-AMS. It provides excellent information within its log files containing all characteristics mentioned above. Hence, TITAN is very well suited for such detailed analyses.

For some experiments, results of Cadence's AMS Designer and Spectre will also be presented. Unfortunately, they do not provide much details for our analyses except dimension, CPU time and number of performed time steps. The number of Newton iterations as well as a brief profiling (loading, solving) would be very valuable for interpreting the simulation performance.

It is not in the scope of this paper to draw a comparison among the used simulators. Results simulated with different simulators must not be absolutely compared as it is not possible to assure equal conditions among different simulators. The presented "variety" of simulators should solely indicate that the results are not TITAN specific.

Table 1. Overview of the Example Circuits

EXAMPLES	Transistors	Device Model	Eqs.	Pars.
opamp741	26	Gummel-Poon	368	564
cfcamp	19	MOS Level 1	220	518
dflipflop	14	BSIM3v3	974	1285

2.3. Example Circuits

For the presentation of analysis results different examples will be used. Table 1 gives a brief overview of the complexity of the example circuits and their corresponding behavioral models derived by Analog Insydes. The number of equations is the total number of model equations when using an extended MNA to set up the circuit equations. All behavioral models were derived by using symbolic equivalents of the device models used in circuit simulation. A fully symbolic

BSIM3 model was used to model the *dflipflop* example.

A short summary of the circuit designs and their testbenches:

- *opamp741* - the well-known μA741 operational amplifier in degenerative feedback (unity gain buffer), pulse wave input voltage (220 kHz frequency, 200 mV amplitude, 2 MV/s slew rate)
- *cfcamp* - a complementary folded-cascode operational amplifier in degenerative feedback (unity gain buffer), sinus input voltage (10 MHz frequency, 1V amplitude)
- *dflipflop* - a D flip-flop (NAND) with clock signal (100 MHz frequency, 1.5 V amplitude, 15 GV/s slew rate), input stimuli trigger all states

2.4. Netlist-Based vs. Behavioral Simulation

To motivate the research, a comparison of the netlist-based (circuit) and the behavioral simulation (model) of the *opamp741* and *cfcamp* circuits (simulated with TITAN) will be presented in this subsection. The statistic data of the according simulations is given in Table 2 for the *opamp741* and Table 3 for the *cfcamp* example. Both tables contain the dimension of the linear system, the sparsity of the Jacobian matrix as well as the number of performed time steps and Newton iterations (during transient analysis). Furthermore, the CPU time of the transient analysis and its distribution for loading (setup of Jacobian / RHS) and solving (LU-factorization and forward-/backward-substitution) are given.

Table 2. Performance Measurements opamp741 (TITAN)

OPAMP741	Dim., Spar.	Time Steps	Iterations	CPU Time		
				Load	Solve	Total
Circuit	58, 90%	1670	3753	0.25 s 81%	0.06 s 19%	0.4 s
Model	381, 98.8%	1391	4856	51.0 s 61%	32.0 s 38%	83.8 s

As can be seen from Table 2, the dimension of the behavioral simulation is much higher than for the circuit as the device model internal equations are contained in the behavioral model too. However, the Jacobian matrix is extremely sparse (98.8%). The time step control of the netlist-based simulation performs 17% more time steps and needs 2.25 iterations per time step at average. The convergence of the behavioral simulation is slightly worse (3.5 iterations per time step), but does not indicate serious convergence problems. The performance of the behavioral simulation compared to the netlist-based simulation is significantly lower by a factor of 210. Moreover, the profiling indicates a remarkable different distribution of the computational effort spent for loading and solving.

For the *cfcamp* (cf. Table 3), there is no difference within the time step control. The behavioral simulation takes 16% more iterations (also not indicating serious convergence problems). The ratio between both performances is 116 with a similar difference of the loading/solving distribution as already noticed in the *opamp741* example. Both analyses are typical for examples of similar complexity. The following analyses are intended to identify potential bottle-necks.

Table 3. Performance Measurements cfcamp (TITAN)

CFCAMP	Dim., Spar.	Time Steps	Iterations	CPU Time		
				Load	Solve	Total
Circuit	23, 71%	508	1515	0.07 s 87.5%	0.01 s 12.5%	0.09 s
Model	233, 98.4%	509	1815	7.25 s 69%	3.13 s 30%	10.5 s

2.5. Analysis of Linear Solving

The TITAN simulator provides different linear solvers that can optionally be applied. For behavioral simulation, three solvers shall be mentioned and compared:

- A *dense* solver - no sparse handling, very robust (default)
- A *sparse* solver - very efficient, but not as robust as the *dense* solver (as no specialized pivoting strategy is used)
- The *MUMPS* solver [6] - a multifrontal parallel sparse solver with dynamic pivoting

By default, the *dense* solver was used for behavioral models as they were usually of low dimension but numerically critical to handle. Hence, this robust but for high dimensions disadvantageous solver was used. To compare the solvers in terms of robustness and performance, Table 4 lists the characteristics of the behavioral simulation for the *opamp741* with different solvers.

Table 4. Analysis of Different Linear Solvers (TITAN)

RESULTS	Solver	Iterations	CPU Time		
			Load	Solve	Total
opamp741	*Dense*	4856	67.9 s	50.4 s	118.3 s
	Sparse	6267	88.0 s	0.8 s	88.8 s
	MUMPS	4856	72.1 s	7.6 s	79.7 s

All simulations took the same number of time steps. The number of iterations needed indicates that the sparse solver is disadvantageous in terms of convergence/robustness. Despite the high number of iterations, it speeds-up the solving by a factor of 63. The loading time increases proportionally to the number of iterations. For the overall performance, the *MUMPS* solver turned out to be even more efficient. It prevents convergence problems by dynamic pivoting strategies and thereby reduces the total CPU time by 33%. This reduces the number of needed iterations, but increases the effort for solving (as pivoting is also accounted). The statistic shows that a sparse solver is essential for behavioral simulations of higher dimension models.

2.6. Analysis of Loading Process

Two experiments have been used to analyze the loading performance in behavioral simulation. To eliminate any side-effects and to only measure the performance in terms of loading and linear solving two types of linear networks were set up as netlists and according behavioral model:

- *Chain network* of resistors with each node additionally connected to ground → tridiagonal Jacobian matrix, high sparsity
- *Complete networks* with each node connected to each other by a resistor → fully populated Jacobian matrix, no sparsity

As the networks are static linear, no dynamic effects or linearization issues are influencing the simulation. The Jacobian matrices of netlist and behavioral simulation are identical. For the complete networks, all benefits from sparse handling strategies are disabled as the matrices are fully populated. By varying the number of nodes (20 to 100) of both network types, the scaling of the simulation performance over dimension and sparsity can be observed. In spite of the linear nature of the problems, transient simulations with limited step size have been performed to achieve a high number of iterations (for accurate CPU measurements).

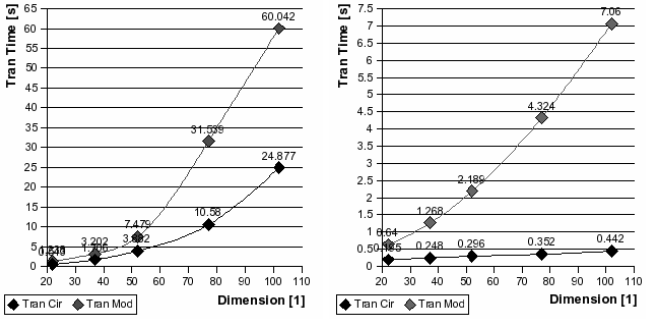

Figure 1 - CPU Time for Complete Networks (left) and Chain Networks (right) in TITAN Simulation

Fig. 1 shows the CPU time needed for the transient analysis over the dimension of the linear systems. As expected, for the complete networks (left figure), both circuit (black) and behavioral (gray) simulation show a quadratic complexity. Although both problems are equivalent, the behavioral simulation is 2-3 times slower than its netlist-based counterpart (for dimensions 20 to 100, rising trend). The profiling data reveals, that the overhead is caused by the loading process (5-10 times slower) whereas the linear solver has the same performance.

The right chart of Fig. 1 contains the measurements for the chain networks. While the sparse mechanisms effectively reduce the complexity for the circuit simulation (linear complexity), the CPU time of the behavioral simulation still shows a quadratic increase over dimension. As typical behavioral models are of relatively low dimension, no sparse loading was realized for TITAN's behavioral simulation. Section 3.1 provides a solution for this shortcoming.

Using Spectre for the same experiments showed that a sparse loading strategy is applied. Nevertheless, the performance of the behavioral simulation is 3-5 times (for chain networks) and 5-7 times (for complete networks) slower than the corresponding netlist-based simulation.

There is no doubt, that some computational overhead will remain in behavioral simulations as the strategy has to be general whereas loading routines in device models are very specialized and therefore highly efficient. Nevertheless, it is most likely that the behavioral simulation could be improved by a more efficient processing during loading.

3. SIMULATOR-RELATED ISSUES

Within this section, some recent improvements of the loading process of the TITAN simulator as well as an approach for a sequential solving strategy will be presented.

3.1. Sparse Loading

For complex behavioral models resulting in Jacobian matrices of high dimension but low ratio of nonzero entries, sparse loading becomes a serious issue (see Section 2.6). Especially the handling of the Jacobian matrix (storage, evaluation, copying) turned out to be worth an increased effort to exploit sparsity. TITAN generates a fully symbolic Jacobian matrix by automatic derivation of the model's equations. This results in a high number of very complex expressions to determine the Jacobian's nonzeros. The CPU time is not dominated by evaluation of those highly complex nonlinear expressions. In fact, cache misses dramatically slowed down the performance during expression evaluation. These cache effects are caused by low data locality. The latter can be improved by sparse data structures.

The realization of a so-called coordinate data structure (a very basic sparse matrix format) and corresponding changes in the processing of the Jacobian matrix effectively increased the loading performance. Fig. 2 (left chart) shows the repeated experiment with chain networks applying the new sparse loading technique. The behavioral simulation (gray) is of linear complexity now and has the same overhead of 2-3 times as already analyzed in the complete networks experiment before (cf. Fig. 1, left chart). Fig. 2 (right chart) shows the behavioral simulations for the complete networks with (gray) and without (black) sparse loading. As sparse loading has no effect for the complete networks (no sparsity), the difference between both graphs represents the additional overhead by initializing and processing the sparse data structure - which is negligible low. Hence, the improved loading algorithm should be of advantage even for Jacobians with very low sparsity and may therefore be used as default loading

mechanism.

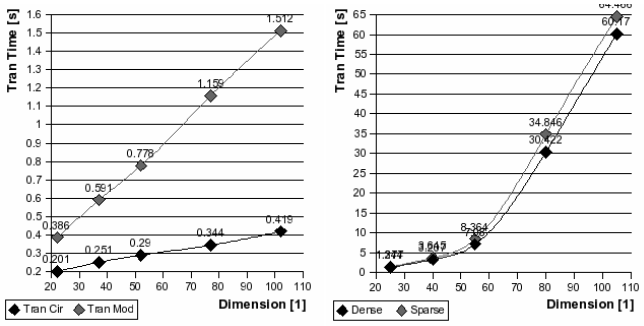

Figure 2 - CPU Time for Chain Networks (left) and Complete Networks (right) in TITAN with Sparse Loading

3.2. Sequential Handling

In common circuit simulators, a major portion of the equations contained in a device model are solved internally in the device model code to compose a compact stamp that is inserted (via MNA) into the simulator's Jacobian matrix. The model-internal equations are presolved in a procedural manner without application of iterative methods. The main intention of this smart handling is to keep the linear equation system as small as possible as the complexity for solving it is dominated by its dimension (and sparsity). To preprocess behavioral models in a similar way and thereby reduce the dimension of the linear system, sequential equations are used. Sequential equations are basically a set of DAEs that can be explicitly solved for their sequential variables. Additionally, each equation may only depend on simultaneous variables (the remaining iteratively treated unknowns) and previously determined sequential variables. This condition results in a lower diagonal block within the Jacobian matrix that might be exploited for improved solving.

So far, TITAN did not support a modeling strategy to consider sequential equations (neither *simultaneous procedural statements* nor VHDL-AMS *functions*). In Spectre and AMS Designer, Verilog-A *procedural assignments* are well-suited to represent sequential equations. Current research is addressing the topic of developing a sequential handling strategy for TITAN. Therefore, a Schur-like reduction method is applied to reduce the dimension of the stamp. The matrix reduction is setup within a prototypical model compiler. This strategy benefits from the high sparsity of the matrix and a cache-efficient processing during the reduction. Applying the Schur-complement to the sequential block results in a correction that is added to the simultaneous block of the Jacobian matrix. Basically, this correction represents the effect of the chain rule, that would have been applied to sequential equations during a (potential) direct setup of the Jacobian for the simultaneous equations. The derived reduced matrix is subsequently loaded and solved by a standard Newton Raphson method. It has the dimension of the number of simultaneous equations and is typically still relatively sparse. By applying such simulation methods, a high number of sequential equations has only a secondary effect on the simulation performance that is primary dominated by the dimension of the linear system and its sparsity.

The speed-up by applying a sequential handling is supposed to be (roughly) proportional to the reduction of the dimension (plus an additional effort for the Schur-complement). First experiments indicate, that the performance can be significantly increased.

4. ADVANCED MODELING STRATEGIES

This section presents some strategies for automatic reformulation of DAEs with respect to simulation performance. The presented algorithms have been integrated into the Analog Insydes modeling flow. Optimizations of similar type have already been successfully applied in automatic device model compilation (e.g. [7]). In this (much more specialized) application, simulator-specific device model code is generated from an AHDL-based model implementation. All presented simulation results were derived with Spectre or AMS Designer as the concepts rely on a sequential handling during simulation (which is currently not supported within TITAN).

4.1. Recognition of Sequential Equations

As stated in Section 3.2, it is most worthwhile in terms of performance and robustness to process as many of the models' equations sequentially as possible. Therefore, sequential equations should be identified during the modeling process and be adequately modeled.

Table 5. Recognition Results for Sequential Identification

RESULTS	*Model*	*Seq. Eqs.*	*Sim. Eqs.*	*Reduction*
opamp741	Orig.	182	186	-14%
	Opt.	208	160	
dflipflop	Orig.	864	110	-44%
	Opt.	912	62	

In Analog Insydes, the information about equations to be handled sequentially is already coded in the symbolic device models. Thereby, a major portion of the nonlinear element relations are set up as sequential equations. E.g. for an instance of the symbolic BSIM3v3 model, the setup of the circuit equations results in 54 sequential and only 6 simultaneous equations. To enable the usage of sequential equations also for general DAEs (e.g. not set up from netlists) an algorithm to identify sequential equations from general DAEs was developed [8]. The algorithm partitions a set of DAEs and its corresponding variables into sequential and simultaneous DAEs. It works on the basis of dependency matrices (indicating the dependent variables of each equation) and their reordering to be compliant with the sequential structure. Optionally, the identification algorithm is able to keep exist-

ing sequential equations (from device models) and identify further sequential equations.

Table 6. Performance Results for Examples with Sequential Identification (Spectre / AMS Designer)

RESULTS	CPU (Orig.)	CPU (Opt.)	Speed-Up
opamp741	9.2 s	8.0 s	1.15
dflipflop	10.5 s	7.2 s	1.45

Table 5 and Table 6 summarize the identification results and their effect on the simulation performance (improvement of 15% to 45%). A simulation without sequential equations - as performed with the TITAN simulator - was not possible due to severe convergence problems.

4.2. Common Subexpression Elimination

Another promising strategy is Common Subexpression Elimination (CSE) - an optimization technique known from compiler design. It is used to prevent (unnecessary) multiple evaluation of identical subexpressions by prior evaluation and substitution with a temporary variable. This technique can also be successfully applied to DAEs (cf. [8] for details).

Table 7. Recognition Results for CSE

RESULTS	Model	Seq. Eqs.	Sim. Eqs.	Recognition
opamp741	Orig.	206	160	+104
	Opt.	312	160	
dflipflop	Orig.	912	62	+604
	Opt.	1516	62	

After the recognition of common subexpressions within the DAEs, the according expressions are assigned to additional sequential variables. All occurrences of the subexpression within the DAEs are consequently substituted by the new sequential variable. As the additional sequential equations do not have any negative effect on the dimension of the resulting linear system and multiple evaluation is prevented, the evaluation of the behavioral model can be significantly accelerated.

Table 8. Performance Results for Examples with CSE (Spectre / AMS Designer)

RESULTS	CPU (Orig.)	CPU (Opt.)	Speed-Up
opamp741	8.0 s	7.3 s	1.09
dflipflop	7.2 s	3.5 s	2

The CSE algorithm extracts a significant number of common subexpressions (see Table 7) yielding a performance increase of 9% to 100% (see Table 8).

5. CONCLUSIONS

In this publication, a detailed comparison of the performance of netlist-based and behavioral simulations has been presented. The analyses indicate an insufficient simulation performance of complex behavioral models generated by the applied symbolic analysis technique. Detailed performance analyses show that the simulation efficiency suffers from suboptimal model formulation as well as from shortcomings in the behavioral simulators due to the extraordinary high complexity of the generated models.

Approaches for (lossless) improvements of the behavioral simulation performance by “simulator-friendly” modeling and enhancements of the simulation algorithms have been presented. Amongst others, automatic reformulation methods have been developed. The strategy reduces the dimension of the linear system by using sequential equations and preventing multiple evaluation of common subexpressions. On the simulator side, an efficient sparse loading was implemented and a prototypical approach for sequential handling in TITAN has been briefly discussed. The efficiency of the recent developments have been presented by exemplary circuits. By combining the discussed and future approaches together with an efficient model reduction strategy (as already available in Analog Insydes), a significant speed-up of the behavioral simulation in comparison to the circuit simulation is most likely. Hence, the used modeling flow may obtain increasing acceptance for automatic bottom-up modeling.

Future aspects of this research will primarily address the development of a new simulation strategy for TITAN to take advantage of sequential equations in behavioral models.

REFERENCES

[1] R. Sommer, E. Hennig, T. Halfmann, T. Wichmann, “Symbolic Modeling and Analysis of Analog Integrated Circuits“, Proc. of the European Conference on Circuit Theory and Design, 1999

[2] Analog Insydes: www.analog-insydes.de

[3] T. Wichmann, M. Thole, “Computer Aided Generation of Analytic Models for Nonlinear Function Blocks”, 10th Intern. Workshop PATMOS, Sep 2000

[4] U. Feldmann, R. Schultz, U. A. Wever, H. Wriedt, Q. Zheng, “Algorithms for Modern Circuit Simulation”, Arch. Elektron. & Uebertragungstech., Vol. 46, pp. 274-285, No. 4, 1992

[5] T. Schneider, J. Mades, M. Glesner, A. Windisch, W. Ecker, “An Open VHDL-AMS Simulation Framework”, Proc. of the IEEE/ACM International Workshop on Behavioral Modeling and Simulation, pp. 89-94, Oct 2000

[6] MUMPS: A Multifrontal Massively Parallel Sparse Direct Solver: http://graal.ens-lyon.fr/MUMPS

[7] B. Wan, B. P. Hu, L. Zhou, C.-J. R. Shi, “MCAST: An abstract-syntax-tree based model compiler for circuit simulation”, Proc. of the IEEE Custom Integrated Circuits Conference, Sep 2003

[8] D. Platte, S. Jing, R. Sommer, E. Barke, “Using Sequential Equations to Improve Efficiency and Robustness of Analog Behavioral Models“, Forum on Specification and Design Languages, Sep 2006