High-Performance Model Compilation for Complex Behavioral Models

Daniel Platte
Qimonda AG
daniel.platte@qimonda.com

Christoph Knoth
Infineon Technologies AG
christoph.knoth@infineon.com

Ralf Sommer
Technische Universität Ilmenau
ralf.sommer@tu-ilmenau.de

Erich Barke
Leibniz University Hannover
eb@ims.uni-hannover.de

ABSTRACT

Automated modeling strategies for analog circuits are becoming more and more important. For complex analytic models, the simulation performance plays a crucial role. This paper presents a highly efficient model compilation based on symbolic analysis. A main feature is the efficient exploitation of advantageous equation structures (so-called sequential equations).

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 and can be easily parameterized.

The symbolic analysis toolbox *Analog Insydes* [1] provides a promising bottom-up modeling flow that is visualized in Fig. 1. It derives differential algebraic equations (DAE) based on an MNA formulation using linear as well as nonlinear symbolic device models from a given circuit. A nonlinear model reduction technique has been proven to efficiently reduce the high complexity of the DAEs [2]. Subsequent reformulation of the equations with regard to numerical methods is used to enhance the simulation performance [3]. Finally, a behavioral model in an analog hardware description language (AHDL) is generated from the reduced DAEs.

Despite efficient model reduction and optimization the simulation performance of these models is still unsatisfactory as was analyzed in detail in [4]. Unsimplified models with 100 % accuracy have been used for the performance analyses and have been compared to the corresponding netlist-based simulation (using the same simulator). Thus, comparable conclusions regarding the simulation performance have been drawn for different examples, simulators, as well as modeling languages. The resulting figure of merit is the "slow down" representing the simulation performance of a model normalized to the performance of its netlist-based original. As the models contain the same equations that have to be solved within the netlist-based simulation, the expectation value for the slow down would be one or slightly above. For the example of the operational amplifier uA741 the slow down significantly varies between 19 and 425 (for 4 widely used commercial simulators using VHDL-AMS or Verilog-A). This performance is unacceptably low and can hardly be compensated by model reduction without significant loss of accuracy.

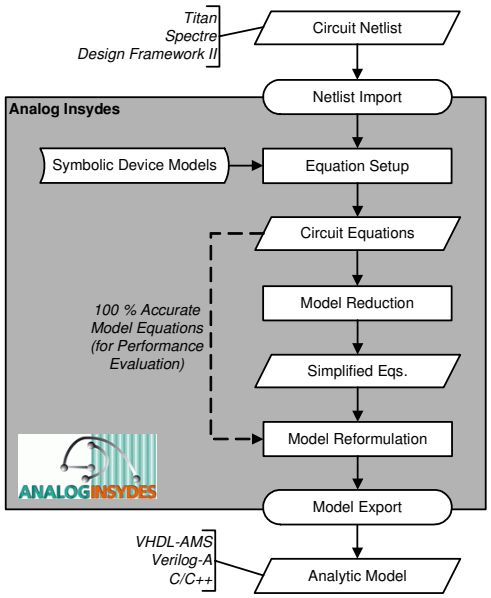

Figure. 1. Bottom-Up Modeling Flow

The aim of this work is to generate compiled models that are spezialized in efficiently dealing with high dimensional DAE systems. Thus, any kind of limitations and drawbacks posed by model compilers and AHDLs are avoided and the slow down is reduced to reasonable values between 1 and 5. The *Titan* simulator of Qimonda AG was chosen as simulation environment [5]. Its behavioral simulation is based on a subset of VHDL-AMS [6]. A new C-interface provides the opportunity to integrate user-specified compiled models. The model export of *Analog Insydes* was extended to directly generate C-based models for *Titan*. Similar approaches exist in the field of device model compilation [7, 8] (e.g. Verilog-A to CMI). However, device model compilation is typically more specific (e.g. requires admittance formulation) and relies on (manually) optimized models. In contrast to that, the model generation within *Analog Insydes* is applicable to general DAE systems.

978-1-4244-1567-0/07/$25.00 © 2007 IEEE

2. ANALOG SIMULATION ALGORITHMS

SPICE-like simulators use Newton's method to iteratively calculate transient solutions for nonlinear dynamic circuits. To solve a system of DAEs $f(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = 0$, it is discretized over time by numerical integration methods with a variable time step (see [9] for details). Within each time step, the resulting nonlinear algebraic equations are iteratively linearized within Newton's method and solved by a linear solver.

$$\boldsymbol{J}(\boldsymbol{x}_n) \cdot \Delta\boldsymbol{x} = -f(\boldsymbol{x}_n) \text{ with } \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta\boldsymbol{x} \qquad (1)$$

Equation 1 shows the linear system where $\boldsymbol{J}$ is the Jacobian matrix, $\Delta\boldsymbol{x}$ the Newton correction, and $-f(\boldsymbol{x}_n)$ the residual (right-hand side). The Jacobian matrix is determined through partial derivation of the algebraic equations:

$$\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}. \qquad (2)$$

The actual solution $\boldsymbol{x}_{n+1}$ is determined from the previous solution $\boldsymbol{x}_n$ and the Newton correction $\Delta\boldsymbol{x}$. $\boldsymbol{x}_{n+1}$ is accepted as solution for a time step as soon as the convergence criteria are met. Most important are the residual test ($f(\boldsymbol{x}_{n+1}) \approx 0$) and the test of the Newton correction ($\Delta\boldsymbol{x} \approx 0$).

The network equations for circuits are set up via modified nodal analysis (MNA). The linear equation system is derived through "direct inspection" - all device models contribute their "stamp" to the Jacobian matrix $\boldsymbol{J}$ and the right-hand side. In behavioral simulation, the Jacobian matrix is determined from the equation set by symbolic derivation (Equation 2).

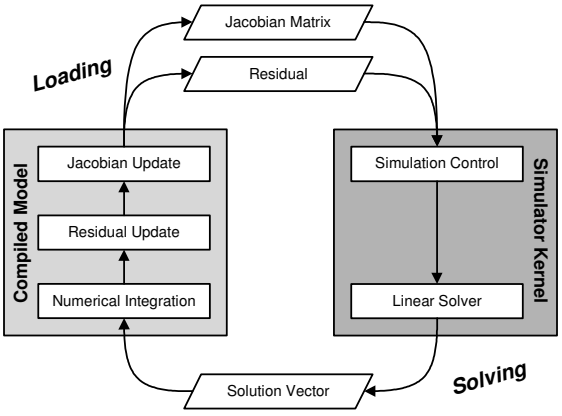

Figure. 2. Simulation Cycle

Fig. 2 visualizes the iterative solving process and the interaction between models and simulator kernel. The compiled model calculates its matrix entries and the residual vector for a solution vector provided by the simulator kernel. This process step is called loading. The achieved linear system is solved by a linear solver (using LU-factorization and forward-/backward-substitution) and yields a Newton correction (the solving process). The simulator kernel checks for convergence and - if necessary - computes a new solution vector.

The CPU time of the simulation scales linearly over the number of necessary Newton iterations, with the latter depending on the number of performed time steps. The solving performance is dominated by the dimension and the structure (sparsity, ordering) of the linear system. The performance of the loading process is related to both the evaluation performance for a large number of complex expressions as well as to the communication between model and simulator kernel. The latter is realized through a shared memory and is therefore determined by the amount of data to copy (which itself is influenced by dimension and sparsity).

The task of a model compiler is to convert an AHDL-based behavioral model into an efficient compiled model. This includes the generation of code for the following process steps:

- Structural information (connectivity, matrix structure)
- Calculation of Jacobian matrix and residual
- Parameter handling
- Initial values
- Numerical integration
- Prevention of numerical problems
- Definition of options and tolerances for the solver

3. MODEL COMPILATION

Due to the highly specialized application of compiling complex DAEs for an efficient simulation a new model compiler was developed. Especially the large number of equations used to result in performance problems when modeled with AHDLs and compiled with common model compilers. The new concept was to process the behavioral model similarly to a device model and to thereby achieve competitive performance to netlist-based simulations.

One of the main concepts is to take advantage of so-called sequential equations. The first order DAE system $f(\boldsymbol{y}, \dot{\boldsymbol{y}}, \boldsymbol{x}, \dot{\boldsymbol{x}}, t) = 0$ with

$\boldsymbol{y} \in R^n$	sequential variables
$\boldsymbol{x} \in R^m$	simultaneous variables
$f = (f_{seq}, f_{sim})$	
$f_{seq}: R^n \times R^m \times R \to R^n$	sequential equations
$f_{sim}: R^n \times R^m \times R \to R^m$	simultaneous equations

is a DAE system with sequential structure, if f_{seq} is of the form

$$y_i = f_{seq,i}(y_1 \ldots y_{i-1}, \dot{y}_1 \ldots \dot{y}_{i-1}, \boldsymbol{x}, \dot{\boldsymbol{x}}, t) \text{ for } i = 1 \ldots n.$$

The sequential equations are explicit for their corresponding sequential variable and ordered in such a way that they can be solved procedurally with a given solution of the simultaneous

variables. Hence, they do not need to be solved within the linear solver but will be calculated locally within the model. Thus, the dimension of the linear system can be reduced significantly leading to performance improvements. Furthermore, the direct solution of these explicit (nonlinear) equations enhances the convergence. The previous compiler for Titan did not support a modeling method to use sequential equations.

The bottom-up model generation is based on *Analog Insydes* and starts from a circuit netlist. As described above, the system of DAEs is set up. To ensure a fair performance comparison with the netlist-based simulation, no model reduction is applied. In [3], algorithms have been presented that optimize DAEs with respect to sequential equations. The optimizations maximize the number of sequential equations, perform common subexpression elimination, and remove redundancy (copy propagation) to speed-up the model evaluation without loss of accuracy. To take advantage of these optimizations the model compiler has to support sequential equations.

Afterwards, the equations are exported to a C-based model. This model is finally compiled with a standard C-compiler and the resulting library is dynamically linked to the simulator kernel. Still, optimizing the DAEs alone does not solve the performance problem. The focus of this paper is on the code generation to achieve optimal simulation performance. Data structures, efficiency of the algorithm's implementation, as well as code optimizations determine the performance of the resulting model to a large degree.

3.1. Newton's Method for Sequential DAEs

How can Newton's method actually take advantage of the sequential structure of a DAE system? Let $f(\boldsymbol{y}, \boldsymbol{x}, t) = 0$ be a DAE system with sequential structure. The vector of unknowns consists of the sequential variables $\boldsymbol{y}$ and the simultaneous variables $\boldsymbol{x}$. Setting up the linearized system yields the following equation system:

$$\boldsymbol{J} \cdot \Delta \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_{11} & \boldsymbol{J}_{12} \\ \boldsymbol{J}_{21} & \boldsymbol{J}_{22} \end{bmatrix} \cdot \Delta \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{x} \end{bmatrix} = - \begin{bmatrix} f_{seq}(\boldsymbol{y}, \boldsymbol{x}) \\ f_{sim}(\boldsymbol{y}, \boldsymbol{x}) \end{bmatrix} \tag{3}$$

Due to the lower diagonal structure of $\boldsymbol{J}_{11}$ resulting from the ordering and properties of sequential equations, it is possible to solve the sequential subsystem for $\Delta \boldsymbol{y}$ in an explicit form ($\boldsymbol{J}_{11}$ is regular due to its diagonal structure). The Newton correction for the sequential variables yields

$$\Delta \boldsymbol{y} = -\boldsymbol{J}_{11}^{-1}(f_{seq}(\boldsymbol{y}, \boldsymbol{x}) + \boldsymbol{J}_{12}\Delta \boldsymbol{x}) . \tag{4}$$

By substituting $\Delta \boldsymbol{y}$ into the simultaneous subsystem we achieve a linear system of reduced dimension that has to be solved for the Newton correction of the simultaneous variables $\Delta \boldsymbol{x}$:

$$-\boldsymbol{J}_{21}\boldsymbol{J}_{11}^{-1}(f_{seq}(\boldsymbol{y}, \boldsymbol{x}) + \boldsymbol{J}_{12}\Delta \boldsymbol{x}) + \boldsymbol{J}_{22}\Delta \boldsymbol{x} = -f_{sim}(\boldsymbol{y}, \boldsymbol{x}) \tag{5}$$

$$(\boldsymbol{J}_{22} - \boldsymbol{J}_{21}\boldsymbol{J}_{11}^{-1}\boldsymbol{J}_{12})\Delta \boldsymbol{x} = -f_{sim}(\boldsymbol{y}, \boldsymbol{x}) + \boldsymbol{J}_{21}\boldsymbol{J}_{11}^{-1} f_{seq}(\boldsymbol{y}, \boldsymbol{x}) \tag{6}$$

As the sequential variables $\boldsymbol{y}$ can be determined via direct solution of the sequential equations with the previous solution vector of $\boldsymbol{x}$ through

$$y_i = f_{seq,\,i}(y_1 \ldots y_{i-1}, \boldsymbol{x}) \text{ for } i = 1 \ldots n , \tag{7}$$

their residual $f_{seq}(\boldsymbol{y}, \boldsymbol{x})$ is zero. Hence the residual of the reduced system can easily be determined with the knowledge of $\boldsymbol{y}$. Consequently, Newton's method has to be applied to solve

$$\boldsymbol{J}'\Delta \boldsymbol{x} = -f_{sim}(\boldsymbol{x}) \tag{8}$$

with the reduced Jacobian matrix $\boldsymbol{J}'$ obtained by the Schur complement:

$$\boldsymbol{J}' = \boldsymbol{J}_{22} - \boldsymbol{J}_{21}\boldsymbol{J}_{11}^{-1}\boldsymbol{J}_{12} \tag{9}$$

$\boldsymbol{J}'$ represents the Jacobian matrix for the simultaneous subsystem taking into account additional contributions to the original Jacobian matrix resulting from the evaluation of the chain rule for the sequential equations.

3.2. Schur Complement

Calculating the Schur complement (refer to Equation 9) is based on structural matrices. Within these matrices, all nonzero entries of the original matrix are represented by a reference element (row/column index). The operator $struct(\,)$ applied to a matrix containing symbolic expressions returns the corresponding structural matrix.

There are several solutions to evaluate the chain rule from the submatrices $\boldsymbol{J}_{11}, \boldsymbol{J}_{12}, \boldsymbol{J}_{21}, \boldsymbol{J}_{22}$ of the complete Jacobian matrix $\boldsymbol{J}$. The resulting $\boldsymbol{J}'$ matrix is the same for all approaches. Nevertheless, the implementation significantly determines their efficiency:

- **Symbolic Schur Complement:** All sequential variables within the simultaneous equations are substituted by their determining sequential equations. Subsequently, a fully symbolic $\boldsymbol{J}'$ matrix is set up by symbolic derivation. This approach results in an expression set of tremendous complexity with a high degree of redundancy. The complexity of the expressions alone disables this procedure even for low dimensional DAE systems.
- **Numeric Schur Complement:** The full Jacobian matrix $\boldsymbol{J}$ is set up by symbolic derivation during the model compilation (as for both following methods). After numerically evaluating this matrix within each iteration, the reduced Jacobian $\boldsymbol{J}'$ is calculated from (9). This approach requires a matrix inversion and two matrix multiplications without

exploiting the structural knowledge of the problem (high sparsity, lower diagonal structure).

- **Semi-Symbolic Schur Complement:** This method combines the strengths of both numeric and symbolic processing. Therefore, the structural Jacobian matrix $\boldsymbol{S} = struct(\boldsymbol{J})$ is set up and the Schur complement is symbolically calculated from $\boldsymbol{S}$ to derive $\boldsymbol{S}'$. Finally, evaluating $\boldsymbol{J}$ and $\boldsymbol{S}'$ results in the reduced Jacobian matrix $\boldsymbol{J}'$. Hence, the Schur complement takes advantage of structural zeros within $\boldsymbol{J}$. Unfortunately, the necessary symbolic matrix operations still result in extraordinary high complexity (disabling the approach starting from medium size matrices).
- **Semi-Symbolic Schur Complement by an Elimination Method:** In order to reduce the complexity of the previous approach, a method with low redundancy (no repetition of performed calculations) and maximal usage of structural properties of the matrices (exploitation of structural zeros and ones) is proposed. Instead of calculating the Schur complement based on $\boldsymbol{S}$, the submatrix $\boldsymbol{S}_{21} = struct(\boldsymbol{J}_{21})$ is eliminated with the diagonal elements of submatrix $\boldsymbol{S}_{11} = struct(\boldsymbol{J}_{11})$ that are structurally one. The performed elimination steps are "recorded" and represent a procedurally evaluated transformation to calculate $\boldsymbol{S}'$ from the submatrices of $\boldsymbol{S}$. Thus, the high complexity of a fully symbolic $\boldsymbol{S}'$ matrix can be avoided. The transformation process has the advantage of containing a large number of very simple expressions (instead of a small number of highly complex expressions for the previous method).

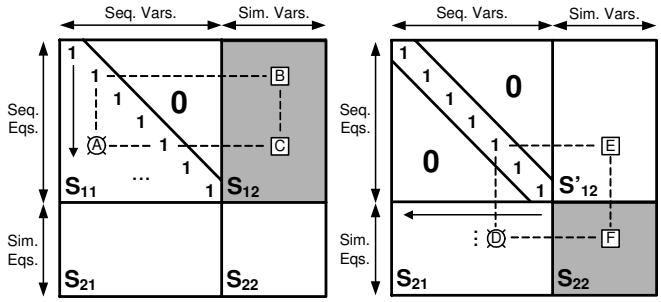

Figure. 3. Elimination Process

Due to its advantageous properties, the latter approach was chosen. Fig. 3 shows the proposed elimination method. The Schur complement is calculated from a two-stage elimination process. Within the first step, the submatrix $\boldsymbol{S}_{11}$ is columnwise eliminated using the diagonal elements to achieve an identity matrix. The elimination starts with the first column. During the processing, the submatrix $\boldsymbol{S}_{12}$ is gradually altered into $\boldsymbol{S}'_{12}$. Fig. 3 (left) exemplarily shows a single step of the proposed elimination method. To eliminate the entry A, an elimination step is added to the transformation rules:

$$C = C - AB \tag{10}$$

During the elimination, fill-ins (former structural zero elements) are generated within the submatrix $\boldsymbol{S}_{12}$. Within a second step, the entries of the submatrix $\boldsymbol{S}_{21}$ are eliminated using the derived identity matrix. Fig. 3 (right) shows the elimination scheme for this stage. The elimination is performed rowwise and results in the desired chain rule contributions within submatrix $\boldsymbol{S}_{22}$. Finally, the gray part of the matrix in Fig. 3 (right) is provided to the simulator as reduced Jacobian matrix. The elimination typically results in a large number of transformation steps to perform. Still, a single elimination step is of very limited complexity. The transformation process itself can be effectively optimized by the C-compiler.

3.3. Optimization

Further enhancements to the performance of the model have been achieved through strategies known from compiler design - constant propagation, constant folding, copy propagation, and preevaluation of loop-invariant expressions (see [10] for details on compiler design). As the optimizations are already carried out on a relatively high abstraction level (the DAEs), they are much more efficient than relying on the C-compiler's code optimizations. Previous analyses made it obvious that commercial model compilers do not sufficiently apply such optimizations [3].

Especially the common subexpression elimination - applied to both symbolic Jacobian matrix and function evaluation - yields very good speed-ups (20 to 40 %). The algorithm recognizes reoccuring subexpressions and replaces them by an additional sequential variable and equation. Furthermore, it is highly desirable to handle as many equations as possible sequentially. A recognition algorithm scans the set of simultaneous equations in order to recognize additional sequential equations. Thus, the dimension of the simultaneous subsystem is efficiently reduced. Please refer to [3] for details on both algorithms.

Constant propagation identifies constants assigned to a variable and removes the corresponding variable by replacing all its occurences with the constant's value (saving evaluation time and memory). As the Jacobian matrix $\boldsymbol{J}$ contains a relatively large number of constant nonzero entries (especially ones) and simple expressions consisting only of parameters (e.g. 1/R), those entries have been recursively propagated into the transformation rules of the Schur complement. Combined with constant folding (preevaluation of expressions containing only constants) the Schur complement could be simplified efficiently.

Finally, the preevaluation of loop-invariant expressions saves unnecessary evaluations within a loop. This strategy is adapted as so-called preloading within simulators. The constant entries of the reduced Jacobian matrix $\boldsymbol{J}'$ are loaded only once within the initialization of the model instead of

being processed repeatedly within each iteration. Furthermore, sub expressions that consist only of parameters and constants are recognized and evaluated within the initialization phase. Thus, the computational effort and the number of necessary memory accesses within each iteration is further reduced.

There are plenty of features that could not be covered within this publication due to limited space. For the sake of high efficiency and high robustness special attention was payed to the following aspects:

- **Tolerances -** to assure adequate accuracy.
- **Limiting functions** - to avoid numerical problems (e.g. floating point exceptions) for nonlinear functions.
- **Initial values -** to enhance DC convergence and avoid numerical problems (e.g. division by zero).
- **Sparse data structures and algorithms -** to assure good performance for high dimensional sparse systems.
- **Data locality -** to reduce the cache miss rate and thereby enhance performance.
- **Structural information and checks -** to enable pivoting strategies and guarantee solvability.

4. RESULTS

To evaluate the simulation performance of the compiled models, four analog blocks have been modeled without application of model reduction using the described flow:

- *cfcamp* – a complementary folded-cascode operational amplifier in degenerative feedback (unity gain buffer), pulse input voltage (500 kHz frequency, 1.3 V amplitude, 900 GV/s slew rate, 450 mV DC), 19 BSIM3 instances
- *multiplier* – a small circuit "multiplying" two input voltages, 8 Gummel-Poon instances
- *nand2* – a NAND gate, input stimuli trigger all states (1.5 V supply voltage, 1.5 GV/s slew rate), 4 BSIM3 instances
- *opamp741* – the uA741 operational amplifier in degenerative feedback (unity gain buffer), pulse wave input voltage (220 kHz frequency, 200 mV amplitude, 2 MV/s slew rate), 26 Gummel-Poon instances

These models are an exemplary selection of typical nonlinear applications for the presented modeling method. The biggest example (*cfcamp*) consists of 19 transistors modeled with the fully symbolic BSIM3 model resulting in the enormously complex model of ~1300 equations. As the focus is on perfomance measurements only, no waveforms will be shown. As no model reduction was applied, the models' accuracies are still 100 %, resulting in identical simulation results.

The comparison is based on unsimplified models of different optimization levels and compared to the netlist-based simulation (Cir.) as reference:

- Simultaneous Model - All equations are modeled simultaneously and solved by the linear solver (Sim.)
- Sequential Model - Sequential equations defined within the symbolic device models are locally solved (Seq.)
- Optimized Model - An optimized sequential model formulation [3] using local solving (Opt.)

The simulation settings and model equations have been set up carefully to ensure equal conditions for the compared simulations. The results in Table 1 show the number of sequential and simultaneous model equations. Note that the sequential equations are solved locally whereas the simultaneous equations require the application of an iterative solving method. The average number of Newton iterations per time step is a convergence criterion (2 is pretty much "perfect convergence"). Finally, the overall performance is indicated by the CPU time spent for the transient analysis and compared to the circuit simulation by the slow down (CPU time normalized to the netlist-based reference simulation).

Example		Seq. Eqs.	Sim. Eqs.	Iter/ Step	T_{CPU}	Slow Down
bsim3	Cir.	n/a	(4)	1.94	0.154 s	1
	Opt.	68	4	1.94	0.208 s	1.35
	Seq.	68	8	2.61	0.414 s	2.69
	Sim.	0	72	3.11	1.209	7.85
cfcamp (BSIM3)	Cir.	n/a	(16)	2.1	1.076 s	1
	Opt.	1261	22	2.9	7.984 s	7.4
	Seq.	1205	78	2.9	30.193 s	28.0
multiplier (Gummel Poon)	Cir.	n/a	(23)	2.0	0.344 s	1
	Opt.	70	20	2.90	0.488 s	1.42
	Seq.	48	42	2.95	0.764 s	2.22
	Sim.	0	90	2.97	1.3 s	3.78
opamp741 (Gummel Poon)	Cir.	n/a	(53)	2.25	1.050 s	1
	Opt.	246	63	3.05	1.802 s	1.72
	Seq.	177	132	3.05	3.001 s	2.86
	Sim.	0	309	3.10	6.452 s	6.14
nand2 (BSIM3)	Cir.	n/a	(6)	2.09	0.314 s	1
	Opt.	270	6	3.21	1.271 s	4.05
	Seq.	254	22	3.25	2.72 s	8.66

Table 1. Performance Results

As Fig. 4 shows, the simulation performance has been significantly enhanced by locally solving as many equations as possible. This leads to a reduced dimension of the linear system that has to be solved by the simulator kernel. In fact, the optimized systems have almost the same dimension (number of simultaneous equations) as the netlist-based simulation - which is an important prerequisite for competitive perfomance. Solving behavioral models of such high complexity

(~1300 eqs.) could only be achieved by efficient processing and local solving. Even more, a performance quite close to the theoretical optimum of a slow down of one has been reached (1.4 to 7.4). Compared to the performance that was previously achieved with models realized in VHDL-AMS or Verilog-A, the CPU-time was reduced by factors of at least 10 (e.g. *opamp741*, from slow down 19 to 1.72). In the past, simulation of behavioral models based on BSIM3 was utterly impossible due to lacking consideration of sequential equations and the resulting numerical problems. Furthermore, a slow down of 7.4 for the highly complex *cfcamp* model (compared to previously 64 using a Verilog-A model [11]), is a great achievement. The remaining overhead can easily be compensated by efficient model reduction, which has yielded a speed up of factor 40 with a 10 % accuracy of the output voltage for this example [11].

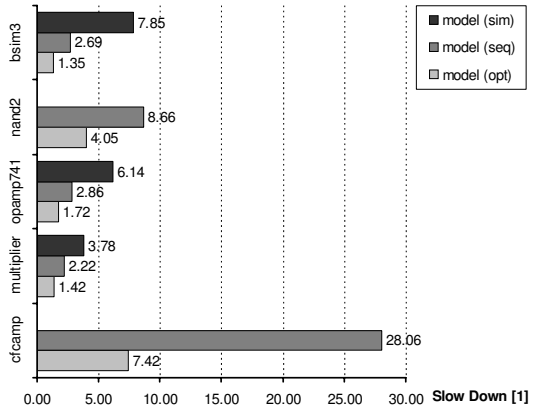

Figure. 4. Comparison of Slow Down Factors

5. CONCLUSIONS

The presented work describes the integration of a high-performance model compilation into the symbolic analysis toolbox *Analog Insydes*. It enables the future usage of the tool for bottom-up modeling purposes with competitive simulation efficiency. The compiler generates specialized C-based models for a compiled model interface of the *Titan* simulator. Compared to AHDL-based models, a significant improvement of the simulation performance by at least one order of magnitude was achieved.

Especially the efficient handling of sequential equations led to a major speed-up resulting from a local solving method and a significantly reduced dimension of the equation system that has to be solved by iterative methods. Furthermore, the convergence of the highly nonlinear and complex DAE systems was improved by the same means.

The compiler has been proven to be capable of handling models of up to 1300 equations based on a fully symbolic BSIM3 model. By combining the presented model generation with the nonlinear model reduction, a fully automated bottom-up modeling method to generate high-performance analytic models is available.

Future aspects of the modeling flow include an extension of the general model compilation strategy for other simulator interfaces (e.g. Cadence's CMI). Furthermore, a user-friendly integration into the Cadence Design Framework II is planned.

REFERENCES

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

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

[3] 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

[4] D. Platte, R. Sommer, E. Barke, "An Approach to Analyze and Improve the Simulation Efficiency of Complex Behavioral Models", IEEE International Behavioral Modeling and Simulation Workshop (BMAS), San Jose/USA, Sept. 2006

[5] 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

[6] 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

[7] V. Chaudhary, M. Francis, W. Zheng, A. Mantooth, L. Lemaitre, "Automatic generation of compact semiconductor device models using Paragon and ADMS", IEEE International Behavioral Modeling and Simulation Workshop (BMAS), San Jose/ USA, Oct. 2004

[8] 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 Circuit Conference, Sept. 2003

[9] J. Vlach, K. Singhal, "Computer Methods for Circuit Analysis and Design", Van Nostrand Reinhold, 1994

[10] A. Aho, M. Lam, R. Sethi, J. D. Ullman, "Compilers: Principles, Techniques, & Tools", 2nd Edition, Pearson Education Inc., Boston, 2007

[11] R. Sommer, D. Platte, J. Broz, A. Dreyer, T. Halfmann, E. Barke, "Automatic Nonlinear Behavioral Model Generation using Sequential Equation Structure", International Workshop on Symbolic Methods and Applications to Circuit and Design, Florence/Italy, Oct. 2006

ACKNOWLEDGEMENTS

This work has been supported by the German Ministry of Education and Research (BMBF) within the project "Verifikation analoger Schaltungen" (VeronA), 01 M 3079.

The research has been done while the authors Daniel Platte and Christoph Knoth have been with Infineon Technologies AG, Munich/Germany. We thank the *Titan* department of Qimonda AG for the excellent support, cooperation, and many fruitful discussions.