The PRAISE Approach for Accelerated Transient Analysis Applied to Wire Models*

Daniel Zaum, Stefan Hoelldampf,
Markus Olbrich and Erich Barke
Institute of Microelectronic Systems
Leibniz Universität Hannover, Germany
{zaum, hoelldampf, olbrich, eb}
@ims.uni-hannover.de

Ingmar Neumann
and Sebastian Schmidt
Continental Corporation
Division Chassis & Safety, Frankfurt a.M.
{ingmar.neumann, sebastian.schmidt}
@contiautomotive.com

ABSTRACT

Continuously shrinking design sizes and the integration of digital and analog blocks in a single IC are clearly identifiable trends in today's microelectronics industry. As both trends increase design complexity and concurrently make the outcome of manufacturing processes less predictable, manufacturing yield is potentially endangered. As a countermeasure, new methodologies for the simulation of mixed-signal-circuits are required. In this paper, we describe a new simulation kernel for the previously presented PRAISE methodology. It accelerates the transient simulation of analog mixed-signal systems by generating and employing abstract circuit models during runtime. We apply the methodology to wire models and discuss results and runtime behavior of different implementations. Furthermore, we present an automated XML-based approach at interfacing PRAISE with arbitrary simulation environments using SystemC.

1. INTRODUCTION

Continuously shrinking design sizes of microelectronic devices indisputably pose a major challenge for today's designers and their software tools. With the 28 nm node imminent, microelectronic systems keep increasing in complexity while the manufacturing process itself concurrently becomes more and more challenging. These ancillary conditions potentially endanger manufacturing yield and therefore drive a growing demand for a better simulation and verification performance prior to the tape-out of the first silicon wafer.

The design of chameleonic devices, integrating a greater variety of functionalities on a single IC (such as digital components, RF-circuits and power electronics), can be easily identified as a further demanding industry trend. The design of such mixed-signal-circuits or even Systems-On-Chips (SoCs) requires a design environment capable of simulating complex systems comprising both analog and digital components. Unfortunately, the explicitness of currently used HDLs such as Verilog or VHDL dramatically reduces simulation performance, especially in the context of global system verification of analog/mixed-signal circuits.

Rising the abstraction level in design capture and simulation by employing system description languages such as SystemC and SystemC-AMS [11] is among the most promising approaches to overcome the limitations of today's traditional HDLs [2]. Past studies have shown that an abstraction level beyond RTL is the enabler of fast global system verification. As the borders between analog and digital design blur, undoubtedly, design capture and simulation of mixed-signal-systems require a multitude of methodologies, each tailored to address a specific aspect of the system.

The creation of high-performance, yet accurate circuit or component models remains a challenging task for digital components and can pose insurmountable problems for certain analog blocks. There is evidence, that certain parts (especially analog components) will always have to be described in detail [3]. Such highly detailed component descriptions significantly slow down global system simulation. Since the conception of abstract models can be very time-consuming, it is desirable to automatically generate abstract models with reduced but adequate accuracy for these "bottleneck" components. In the past, several approaches to this end have been presented [6, 7, 8, 10].

We propose to generate extensively precomputed models of analog components to accelerate the co-simulation of digital parts and the critical analog blocks. Our approach, denominated as *Piecewise Rapid Analog Simulation Environment* (PRAISE) has been previously introduced in an automotive context [5, 12]. In this paper, we present an addition to our previously presented simulation methodology and discuss the performance of different implementations. We apply the PRAISE approach to exemplary wire models to prove it's feasibility in an area of application that differs from the previously considered automotive domain. Furthermore, our new XML-based interface to the SystemC language is presented.

*This work has been supported by the German Ministry of Education and Research (BMBF) within the project "AutoSUN" (Project ID 01M3178A). The content is the sole responsibility of the authors.

The remainder of the paper is organized as follows: Section 2 illustrates our approach for the automated generation of circuit models from given netlists and the appendant simulation methodology. Furthermore, our current implementations are presented and compared against each other. Section 3 details wire models as one possible domain of application of the PRAISE approach and exemplary circuits are presented. Section 4 describes the SystemC-based interfacing of our software with arbitrary simulation environments utilizing the SystemC system description language. In Section 5, transient simulation results of the previously presented circuits are outlined and simulation runtime is discussed. Section 6 concludes the paper.

2. PRAISE SIMULATION APPROACH

The PRAISE Simulation Approach is based on piecewise constant (PWC) excitations of a given circuit. This assumption facilitates a behavioral modeling using simple output functions such as

$$y(t) = \sum_{i=1}^{n} a_i e^{\lambda_i t} \tag{1}$$

for each variable to be observed. The coefficients a_i depend on both the initial state of the circuit and on the piecewise constant inputs. Inputs have to be specified as node voltages or branch currents due to the automatic addition of corresponding ideal sources for further model generation steps. Outputs also have to be provided as node voltages or branch currents in order to calculate the appropriate exponential output functions.

Currently, two different implementations exist. Firstly, there is a prototype $\mathfrak{I}_{symb}$ for evaluating the concepts of the approach. Computer algebra systems provide symbolic computations for finding the output functions. Secondly, there is an object oriented, high performance C++ implementation $\mathfrak{I}_{num}$, in which optimized libraries provide fast matrix operations. It furthermore features a SystemC interface for a straightforward integration into existing simulation environments.

Fig. 1 illustrates the general flow of the PRAISE approach. A netlist of the analog circuit is parsed and the obtained circuit equations are transformed to a state space representation. This representation is then transformed to circuit models during model compilation. These models are used by the actual circuit simulation. The details of the flow are described below.

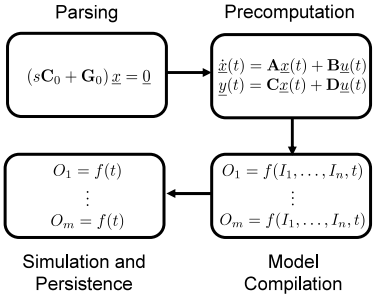

Figure 1: The PRAISE Simulation Flow

2.1 Parsing

In order to retrieve the system equations the *Modified Nodal Approach* (MNA) [4] is applied to the netlist of the circuit:

$$(s\mathbf{C}_0 + \mathbf{G}_0)\,\underline{x}_0 = \mathbf{B}_0\underline{u} \tag{2}$$

2.2 Precomputation

Our implementations $\mathfrak{I}_{symb}$ and $\mathfrak{I}_{num}$ pursue different strategies for transforming the circuit equations into the state space representation.

2.2.1 Transfer Functions

$\mathfrak{I}_{symb}$ calculates the transfer functions of the circuit:

$$H_{i,j}(s) = \frac{Y_i(s)}{U_j(s)} = \frac{a_m s^m + \cdots + a_1 s + a_0}{b_n s^n + \cdots + b_1 s + b_0} \tag{3}$$

In general a MIMO (multiple input multiple output) circuit results in the transfer matrix

$$\mathbf{H}(s) = \begin{bmatrix} H_{1,1}(s) & \cdots & H_{1,m}(s) \\ \vdots & \ddots & \vdots \\ H_{n,1}(s) & \cdots & H_{n,m}(s) \end{bmatrix} \tag{4}$$

The transfer matrix is then transformed to a canonical controllable form using a symbolic mathematical toolbox:

$$\begin{aligned} \dot{\underline{x}}(t) &= \mathbf{A}\underline{x}(t) + \mathbf{B}\underline{u}(t) \\ \underline{y}(t) &= \mathbf{C}\underline{x}(t) + \mathbf{D}\underline{u}(t) \end{aligned} \tag{5}$$

2.2.2 Direct Transformation

The implementation $\mathfrak{I}_{num}$ directly transforms the circuit equations into a state space representation using an algorithm similar to [9]. The voltages and branch currents to be observed are specified by the output matrix $\mathbf{D}_0$ in

$$\underline{y}_0(t) = \mathbf{D}_0\underline{x}_0(t) \tag{6}$$

The algorithm identifies the state variables of the system and transforms the equation system using row and column operations:

$$s\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{G}_{11} & \mathbf{G}_{12} \\ \mathbf{G}_{21} & \mathbf{G}_{22} \end{bmatrix}\begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{01} \\ \mathbf{B}_{02} \end{bmatrix}\underline{u} \tag{7}$$

The variable vector $\underline{x}_0$ is now separated into $\underline{x}_1$ consisting of the state variables and $\underline{x}_2$ holding excessive variables. The vector $\underline{x}_2$ mostly represents dependencies between resistive elements which can be expressed as a function of $\underline{x}_1$. The reduced equation system is then transformed to the state space representation of the form (5). The voltages and currents to be observed during simulation are selected by appropriate matrices $\mathbf{C}$ and $\mathbf{D}$. Calculating the coefficients a_i is the essential task of the PRAISE approach. Therefore, we create a Jordan normal form of the state space representation by applying the transformation

$$\underline{x}(t) = \mathbf{T}\tilde{\underline{x}}(t) \tag{8}$$

The transformation matrix $\mathbf{T}$ consists of the eigenvectors of the reduced system. Eventually the new state space representation results in

$$\begin{aligned} \dot{\tilde{\underline{x}}}(t) &= \tilde{\mathbf{A}}\tilde{\underline{x}}(t) + \tilde{\mathbf{B}}\underline{u}(t) \\ \underline{y}(t) &= \tilde{\mathbf{C}}\tilde{\underline{x}}(t) + \mathbf{D}\underline{u}(t) \end{aligned} \tag{9}$$

where $\tilde{\mathbf{A}}$ is a diagonal matrix with all eigenvalues on the main diagonal.

2.3 Model Compilation

The time domain solution of the state space representation is

$$\begin{aligned} \underline{\tilde{x}}(t) &= e^{\tilde{\mathbf{A}}t}\underline{\tilde{x}}_0 + \int_0^t e^{\tilde{\mathbf{A}}(t-\tau)}\tilde{\mathbf{B}}\underline{u}(\tau)d\tau \\ \underline{y}(t) &= \tilde{\mathbf{C}}e^{\tilde{\mathbf{A}}t}\underline{\tilde{x}}_0 + \tilde{\mathbf{C}}\int_0^t e^{\tilde{\mathbf{A}}(t-\tau)}\tilde{\mathbf{B}}\underline{u}(\tau)d\tau + \mathbf{D}\underline{u}(t) \\ \underline{y}(t) &= \tilde{\mathbf{C}}\underline{\tilde{x}}(t) + \mathbf{D}\underline{u}(t) \end{aligned} \tag{10}$$

with the initial values $\underline{\tilde{x}}_0$ and the inputs $\underline{u}(t)$ of the system. The matrix exponential $e^{\mathbf{A}t}$ is defined as

$$e^{\tilde{\mathbf{A}}t} = \sum_{k=0}^{\infty} \frac{\tilde{\mathbf{A}}^k t^k}{k!} = \mathbf{I} + \tilde{\mathbf{A}}t + \frac{\tilde{\mathbf{A}}^2 t^2}{2!} + \cdots \tag{11}$$

Calculating the infinite sum is both symbolically and numerically difficult. In order to avoid this computation, we exploit the fact that the diagonal matrix $\tilde{\mathbf{A}}$ facilitates the simple relationship

$$e^{\tilde{\mathbf{A}}t} = e^{\mathrm{diag}(\lambda_i)} = e^{\begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}} = \begin{bmatrix} e^{\lambda_1} & & 0 \\ & \ddots & \\ 0 & & e^{\lambda_n} \end{bmatrix} \tag{12}$$

and therefore each entry of the result matrix can be computed independently. Considering the diagonal matrix $\tilde{\mathbf{A}}$ and the piecewise constant inputs, the time domain solution can be simplified to

$$\begin{aligned} \underline{\tilde{x}}(t) &= e^{\tilde{\mathbf{A}}t}\left(\underline{\tilde{x}}_0 + \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{B}}\underline{u}(t)\right) - \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{B}}\underline{u}(t) \\ \underline{y}(t) &= \tilde{\mathbf{C}}\underline{\tilde{x}}(t) + \mathbf{D}\underline{u}(t) \end{aligned} \tag{13}$$

At this point of the flow the circuit model is compiled and can be used for simulation.

2.4 Simulation

Substituting piecewise constant input segments

$$\underline{u}(t) = \text{const},\ 0 \le t < T \tag{14}$$

into (13) yields the corresponding output functions depending on the respective initial circuit state and the current piecewise constant input segments $\underline{u}(t)$.

The general software architecture of our C++ implementation $\mathfrak{I}_{num}$ is shown in Fig. 2. It primarily consists of two parts, the *Transformation* package on the left side and the *Representation* package at the center. The former performs transformations and conversions on the different data structures of the Representation package. Starting at the bottom, the circuit netlist is swayed between both packages until the output functions are calculated at the top. The *Library* package provides equations for nonlinear elements as piecewise-linear structures. The *Tools* packages comprises all auxiliary libraries including BLAS and LAPACK [1] for matrix operations.

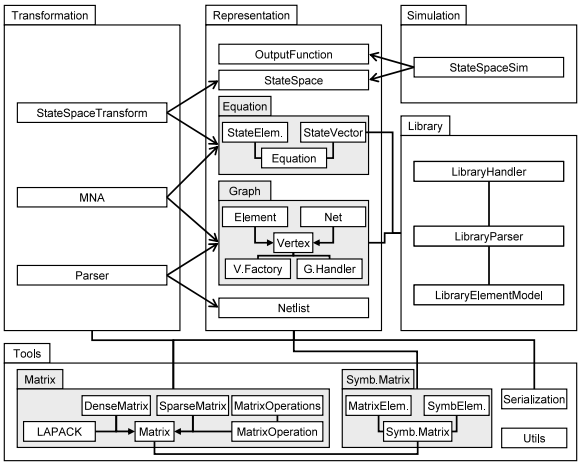

Figure 2: PRAISE software architecture

3. APPLICATION TO WIRE MODELS

Interconnects, being used for the transportation of signals between distant points on a chip, often exhibit malicious electrical properties. Such so called parasitics introduce undesired effects (e.g. delay and dispersion), which can be derived from Maxwell's equations. As solving Maxwell's equations is very time-consuming, wire models consisting of resistors, capacitors and inductors are employed. State-of-the-art delay calculation methods are based on model order reduction (MOR) of lumped RCL networks.

In order to evaluate the performance of our approach applied to such wire models, we employ two scalable circuit examples, described below. The RCL-circuit $\mathfrak{C}_{rcl}^N$ is shown in Fig. 3. Throughout the paper we use the parameter N

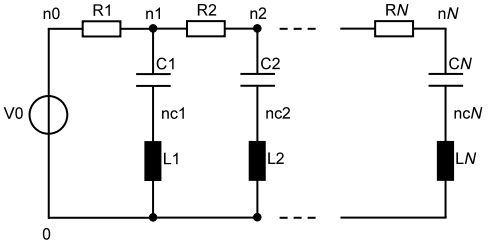

Figure 3: Dynamically Generated RCL-Circuit $\mathfrak{C}_{rcl}^N$

to denominate the size of the circuit $\mathfrak{C}_x^N$. $\mathfrak{C}_{rcl}^N$ is of the order

$$2^N \tag{15}$$

with

$$3N + 1 \tag{16}$$

elements. As RC-models often deliver sufficient results while allowing for reduced simulation times, as a second example we will use the RC-circuit $\mathfrak{C}_{rc}^N$ depicted in Fig. 4. This circuit is of the order

$$N \tag{17}$$

and contains

$$2N + 1 \tag{18}$$

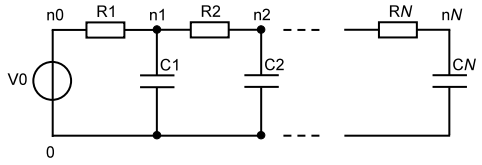

Figure 4: Dynamically Generated RC-Circuit $\mathfrak{C}_{rc}^{N}$

elements. Due to the linearity of RC and RCL wire models, our approach requires only one, yet complex, precomputation step. Hence, only one circuit state exists which is highly beneficial, since our approach in this case yields a closed form representation for each monitored circuit variable.

4. SYSTEMC SIMULATION KERNEL

In recent years, SystemC has been established as one of the most broadly accepted system description languages for behavioral modeling and the emerging transaction level design methodology (TLM). Anyhow, due to the huge spectrum of possible applications, various publications reveal a variety of interpretations, rendering TLM to remain an ambiguous term. The success of SystemC is certainly to be credited with the fact, that SystemC is not only a description language but beyond that a stand-alone simulation kernel. Hence, the choice of SystemC as an interface language to other simulation environments enables us to deliver precompiled executable models of analog blocks.

As our circuit descriptions represent the complete circuit and potentially contain precomputed terms for every node voltage and branch current in the circuit, a further abstraction layer has to be added in order to create interfaceable circuit models. Primarily, the sources serving as inputs to the module and the output values to be monitored during simulation have to be specified. Additionally, a module name is specified. For coupling with event-driven simulators it is possible to specify a signal input called stepclock. This signal will then be used to trigger the computation of output values during runtime. We chose an XML-based format as shown below:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE praiseSpec SYSTEM "praise.dtd">
<module name="valveControl">
```

Inputs, as described in Section 2, are characterized by an identifier of the form

$$\{\mathbf{V}|\mathbf{I}\}, \mathfrak{n}_a, \mathfrak{n}_b \tag{19}$$

with $a, b \in \{0, \ldots, n\}$ whereas $\mathbf{V}$ or $\mathbf{I}$ denominate a voltage or a current between the two given nodes $\mathfrak{n}_a$ and $\mathfrak{n}_b$. A SystemC data type for each value (such as double, float, int, bool) can be appointed. Input values are analogously defined:

```
<inputs>
 <input name="V,n_1,n_2" type="double"/>
 <input name="I,n_15,n_21" type="double"/>
</inputs>
```

```
<outputs>
 <output name="V,n_12,n_18" type="double"/>
 <output name="V,n_27,n_39" type="double"/>
</outputs>
```

The module declaration ends with the optional specification of the signal name for time synchronization with discrete event simulators:

```
<stepclk name="slck"/>

</module>
```

Fig. 5 depicts the mandatory steps for the creation of a SystemC interface layer around the precomputed models. Given a netlist and the XML input specification described

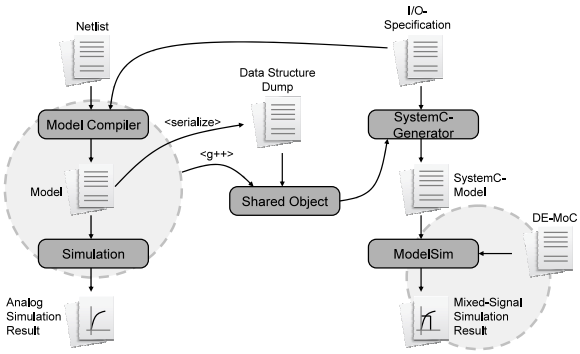

Figure 5: Generation of the SystemC-Interface

above, our model compiler precomputes and stores output equations for the circuit variables. The resulting numerical model can be used to perform a stand-alone model-based simulation of the considered analog block, as described in Section 2. In order to trigger the evaluation of the model from a SystemC wrapper file, two steps are necessary: firstly, the simulation kernel has to be compiled as a dynamically linkable shared object and secondly, the precomputed output functions have to be stored on mass storage media.

4.1 Persistence of Circuit Models

We compile the simulation kernel as a shared object, which guarantees platform independence, since several target platforms are possible. Storage of the precomputed data is performed using the serialization technique. The term serialization describes the process of converting an object into a sequence of bits. This technique can be used to create a semantically identical clone of the original object. However, for many complex objects, such as those that make extensive use of references, this process is not straightforward. As pointer objects are too fragile to save, a step called unswizzling is required. Runtime and resulting file sizes of this automated process are depicted in Table 1. The circuit size N refers to the corresponding parameter of the automatically generated netlists, as described in Section 3. It becomes obvious, that this approach of storing the precomputed data structures excels with respect to performance while yielding models of acceptable size. The measurements presented in Table 1 are valid for linear circuits. As nonlinear circuits require the precomputation of multiple circuit

Table 1: Serialization Runtime and File Sizes for $\mathfrak{C}_{rcl}^N$

Size N	Plain ($byte$)	Zipped ($byte$)	Read (s)	Write (s)
100	$46k$	$8.6k$	0.0	0.01
200	$92k$	$17k$	0.01	0.01
250	$115k$	$21k$	0.01	0.01
300	$139k$	$25k$	0.01	0.01
400	$185k$	$32k$	0.01	0.01
500	$231k$	$40k$	0.02	0.01
1000	$463k$	$77k$	0.04	0.02

states, file sizes will increase. We expect 500 MB model files and read/write times of about 1 s for large nonlinear circuits containing about 200 elements.

4.2 Generation of SystemC Wrapper

Given the I/O-specification file described above, we automatically generate a SystemC file with the required input and output definitions. This file includes the previously compiled shared simulation object and hence evaluates the stored models when input value changes are detected. An exemplary output of the SystemC wrapper generator is listed below:

```
#include <vector>
#include "systemc.h"
#include "SimulationCore.h"

SC_MODULE (Example) {

  sc_in<int>  a;
  sc_in<int>  b;
  sc_out<int> c;

  void simulate() {
    std::vector<int> inputs_vector(2);
    inputs_vector[0] = a;
    inputs_vector[1] = b;
    std::vector<int> outputs_vector(1);
    SimulationCore sC;
    sC.simulate(inputs_vector, outputs_vector);
    c.write(outputs_vector[0]);
  }

  SC_CTOR(Example) {
    SC_METHOD(simulate);
    sensitive << a;
    sensitive << b;
  }
};
```

In this simple example, two input values are passed to the simulation kernel (linked as shared object), are then evaluated and the result is passed on as output of the module.

5. RESULTS

Fig. 6 and Fig. 7 depict the simulation results of two exemplary circuits. All data presented in this section has been generated on a 16 core system with AMD 8220 processors at 2.8 GHz. For the sake of clarity the two relatively simple circuits $\mathfrak{C}_{rc}^2$ and $\mathfrak{C}_{rcl}^3$ containing 5 and 10 elements respectively, are employed. Nevertheless, $\mathfrak{C}_{rcl}^3$ already represents a circuit of the 6th order. Correctness of the presented simulation results has been proven by verification against the results of a traditional analog simulator. Table 2 and Table 3 show the runtime of the model generation and the consecutive simulation step using our symbolic implementation $\mathfrak{I}_{symb}$. As expected, simulation runtime scales approximately linearly,

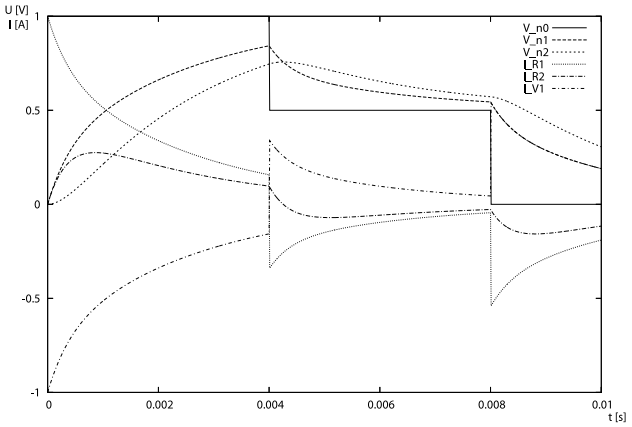

Figure 6: Simulation Result for Circuit $\mathfrak{C}_{rc}^2$

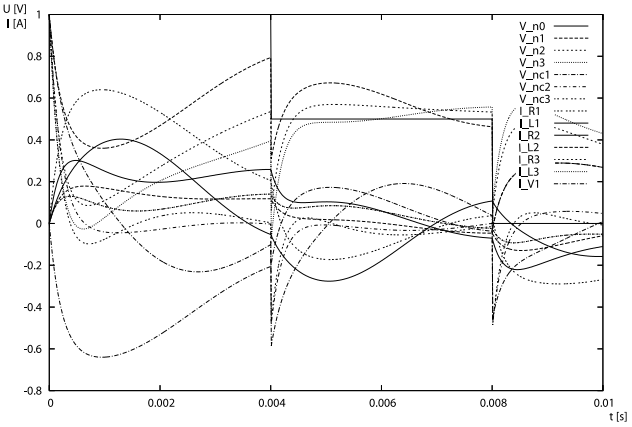

Figure 7: Simulation Result for Circuit $\mathfrak{C}_{rcl}^3$

when model size increases. This implementation certainly does not excel with respect to absolute runtime, but serves to show the feasibility of the approach. Due to the use of symbolic math toolboxes, we are confident that the runtime behavior of this implementation can be further optimized.

Table 4 and 5 show the model generation runtimes of the

Table 2: Model Generation and Simulation Runtime for $\mathfrak{C}_{rcl}^N$ with implementation $\mathfrak{I}_{symb}$

Size N	Model Generation (s)	Simulation (s)	Total (s)
1	1.11	0.83	1.94
2	1.74	1.50	3.24
3	2.70	2.55	5.25
4	4.67	3.67	8.34
5	5.48	5.53	11.01
6	7.51	7.85	15.36

Table 3: Model Generation and Simulation Runtime for $\mathfrak{C}_{rc}^N$ with implementation $\mathfrak{I}_{symb}$

Size N	Model Generation (s)	Simulation (s)	Total (s)
1	0.93	0.40	1.33
2	1.04	0.69	1.73
3	1.32	0.83	2.15
4	1.60	1.02	2.62
5	2.11	1.28	3.39
6	2.47	1.58	4.05

high-performance implementation $\mathfrak{I}_{num}$ for very large circuits $\mathfrak{C}_{rc}^{N}$ and $\mathfrak{C}_{rcl}^{N}$ up to a size of $N = 1000$. The largest circuit, $\mathfrak{C}_{rcl}^{1000}$, of the 2000^{th} order already exceeds the limits of realistic circuit examples. The tables present the constituent of the model generation runtime in detail. Runtimes of the parsing and MNA steps for circuit equation formulation are given as well as the time-consuming state space transformation step and the runtime of the eigenvector computation, which can be considered a major bottleneck of the procedure.

Table 4: Model Generation Runtime for $\mathfrak{C}_{rc}^{N}$ with implementation $\mathfrak{I}_{num}$

Size N	Parsing (s)	MNA (s)	$\vec{v}_{eig}$ (s)	State Space (s)	Total (s)
100	0.01	0.01	0.05	0.08	0.13
200	0.01	0.02	0.43	0.63	0.72
250	0.01	0.04	0.84	1.38	1.50
300	0.01	0.05	1.52	2.74	2.92
400	0.02	0.09	4.11	7.10	7.27
500	0.03	0.14	8.24	13.61	13.94
1000	0.10	0.56	49.42	97.09	98.01

Table 5: Model Generation Runtime for $\mathfrak{C}_{rcl}^{N}$ with implementation $\mathfrak{I}_{num}$

Size N	Parsing (s)	MNA (s)	$\vec{v}_{eig}$ (s)	State Space (s)	Total (s)
100	0.0	0.05	0.32	0.68	0.86
200	0.01	0.18	2.99	7.33	7.85
250	0.02	0.27	6.10	13.71	14.29
300	0.02	0.44	9.65	24.19	24.85
400	0.04	0.73	23.72	58.56	59.66
500	0.06	1.14	48.27	118.05	119.81
1000	0.29	6.32	432.25	1015.44	1020.09

6. CONCLUSION AND OUTLOOK

We have presented the PRAISE approach to the accelerated transient analysis of mixed-signal systems by generating precomputed circuit models of the crucial analog blocks during simulation runtime. The approach was applied to the simulation of wire models. The methodology has proven to work with the expected accuracy in this new domain of application by verifying our results against a traditional analog simulator. In addition, an extended simulation methodology, exploiting certain characteristics of the matrix exponential function, has been presented. Runtime behavior of two implementations, differing in the usage of symbolic and numerical expressions, has been discussed in detail. An XML-based approach for interfacing our implementations to other simulation environments via SystemC has been detailed.

We plan to build up a database of non-linear element models tailored to the automotive domain. Multiple models will allow a trade-off between simulation speed and precision. Due to the superior performance of numerical methods, our final implementation will be based completely on the C++ language and will offer a SystemC interface for effortless integration within other simulation environments.

7. REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 3 edition, 1999.

[2] R. Bergamaschi, S. Bhattacharya, R. Wagner, C. Fellenz, M. Muhlada, F. White, J.-M. Daveau, and W. Lee. Automating the design of SoCs using cores. *IEEE Design and Test of Computers*, 18(5):32–45, Sept. 2001.

[3] G. Gielen and R. Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1854, Dec. 2000.

[4] C.-W. Ho, A. Ruehli, and P. Brennan. The modified nodal approach to network analysis. *IEEE Transactions on Circuits and Systems*, 22(6):504–509, June 1975.

[5] S. Hoelldampf, D. Zaum, M. Olbrich, I. Neumann, S. Schmidt, and E. Barke. Methodologies for high-level modelling and evaluation in the automotive domain. In *Specification, Verification and Design Languages, 2008. FDL 2008. Forum on*, pages 73–77, Sept. 2008.

[6] X. Li, X. Zeng, D. Zhou, X. Ling, and W. Cai. Behavioral modeling for analog system-level simulation by wavelet collocation method. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 50(6):299–314, June 2003.

[7] J. A. Martinez, T. P. Kurzweg, S. P. Levitan, P. J. Marchand, and D. M. Chiarulli. Mixed-technology system-level simulation. *Analog Integrated Circuits and Signal Processing*, 29(1–2):127–149, Oct. 2001.

[8] C. Meise and C. Grimm. A SystemC based case study of a sensor application using the BeCom modeling methodology for virtual prototyping. In *SBCCI '04: Proceedings of the 17th Symposium on Integrated Circuits and System Design*, pages 242–247, Sept. 2004.

[9] S. Natarajan. A systematic method for obtaining state equations using MNA. *Circuits, Devices and Systems, IEE Proceedings G*, 138(3):341–346, June 1991.

[10] L. Nathke, V. Burkhay, L. Hedrich, and E. Barke. Hierarchical automatic behavioral model generation of nonlinear analog circuits based on nonlinear symbolic techniques. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 442–447, Feb. 2004.

[11] A. Vachoux, C. Grimm, and K. Einwich. Extending SystemC to support mixed discrete-continuous system modeling and simulation. *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, 5:5166–5169, May 2005.

[12] D. Zaum, S. Hoelldampf, M. Olbrich, I. Neumann, S. Schmidt, and E. Barke. Accelerating transient analysis using runtime-generated abstract circuit models. In *International Workshop on Symbolic and Numerical Methods, Modeling and Applications to Circuit Design, 2008 (SMMACD 2008)*, Oct. 2008.