Behavioral Modeling of Transistor-Level Circuits using Automatic Abstraction to Hybrid Automata

Ahmad Tarraf
Institute for Computer Science
Goethe University Frankfurt, Germany
tarraf@em.cs.uni-frankfurt.de

Lars Hedrich
Institute for Computer Science
Goethe University Frankfurt, Germany
hedrich@em.cs.uni-frankfurt.de

***Abstract*—Accurate abstracted behavioral modeling of analog circuits is still an open problem, especially when the abstraction process is automated. In this paper we present an automated abstraction technique of transistor level circuits with full SPICE accuracy alongside a significant simulation speed-up. The methodology computes a hybrid automaton which is transformed into a behavioral model in Verilog-A. The resulting hybrid automaton exhibits linear behavior as well as the technology dependent nonlinear e.g. limiting behavior. The accuracy and speed-up of the methodology is evaluated on several transistor level circuits ranging from simple operational amplifiers up to a complex industrial OTA-based Gm/C filter. Finally, we formally verify the equivalence between the generated model and the original circuit.**

***Index Terms*—abstraction, verification, hybrid automaton, Verilog-A, behavioral modeling**

I. Introduction

Automatic behavioral modeling of analog circuits is an open problem with a long research history [1]. A large speed-up factor of these behavioral models would be desired to support the complex simulation of at system or at least at module level. As the systems integrable on a chip become more complex and heterogeneous, the use of accurate behavioral models for analog signal processing and interfacing would enhance design, simulation and validation routines.

Due to the lack of automatic tools, most design groups manually write Verilog-A, VHDL-AMS or MAST models to perform module or system simulations. Even if a behavioral model is available, the succeeding problem is to proof the validity of this model. For example, due to increased demand in safety for autonomous driving applications, the need for a verifiable methodology of model generation is obvious. In this paper we want to tackle both parts of the problem:

- First, automatically generate a behavioral model in Verilog-A from a transistor-level netlist with SPICE BSIM accuracy.
- Second, compare the model against the netlist with a formal method (equivalence checking) including the calculation of the modeling error.

To tackle the first problem, we abstract the linear and nonlinear behavior of the transistor-level netlist to an hybrid automaton. Where each of the states of the automaton models the behavior of the system piecewise linear. We developed a methodology to generate a behavioral model in Verilog-A from these hybrid automaton. The model represents accurately the input-output behavior of the block. Additionally, every internal nodal voltage can be generated for debugging purposes. Moreover, the generated models are pin compatible, thus they can be easily used on top-level or module-level simulations. This comes also with the advantage of using the models to generate larger compositional ones. Hence, this approach aims to reduce the complexity of circuits, decrease simulation time and group systems by modeling the blocks of an analog signal processing chain in a compositional manner. An overview of our methodology is illustrated in Fig. 1.

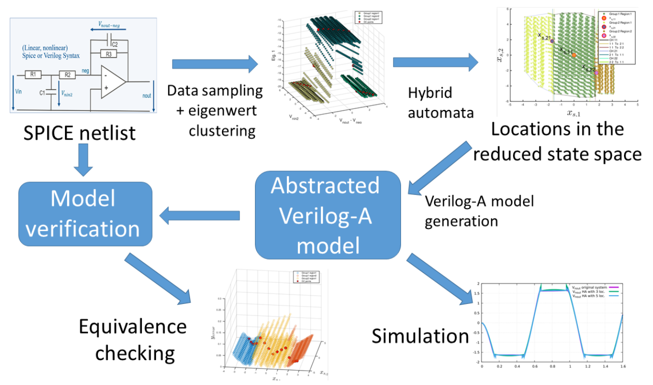
SPICE netlist
Data sampling + eigenwert clustering
Hybrid automata
Locations in the reduced state space
Verilog-A model generation
Abstracted Verilog-A model
Model verification
Equivalence checking
Simulation

Fig. 1. An overview of our approach. After the model generation, the model can be used for simulation or for verification against the original netlist.

The paper is structured as follows: First, we briefly present the state of the art in section II. Second, we explain the algorithm behind the automated model generation in section III. As illustrated in Fig. 1, we sample the state space (III-A), identify the locations of the hybrid automata (III-B and III-C), build the Dynamics (III-D) and finally establish the Verilog-A model (III-E). Third, some examples are handled in section IV-A, followed by a verification of a generated model against the original netlist in section IV-B. Finally, the conclusion is stated in section V.

II. Previous Work

Behavioral modeling of analog circuits has been tackled for a long time [1]. Basic methods start with manual modeling in the frequency domain [2]. These equation based methods have been enhanced by direct symbolic simulation based methods for integrated nonlinear behavioral models [3] and piecewise linear models [4]. Moreover, the methods have been further developed to on-the-fly piecewise linear model generation with the possibility to use an event based simulation [5], [6]. However, these methods are mostly not fully automated or cannot handle SPICE netlist with full BSIM transistor accuracy. Other approaches use numeric approximations like radial basis functions with neural networks techniques [7] or general kernels models in support vector machines [8]. The latter paper uses also the order reduction techniques used in this paper, to bring the high order resulting from parasitic poles down to the functional needed order. Unfortunately, these data driven methods suffer from lower accuracy and difficulties in obtaining significant speed-ups. More digital oriented methods like [9] abstract the analog dynamic behavior with finite state machines. They can be executed directly in the digital domain but loose the analog dynamics like complex poles.

Besides the problem of model generation, model verification also plays a crucial role. This is tackled in this paper by using equivalence checking, where only few analogous approaches exist [10] [11].

III. Automatic Abstraction and Model Generation

A. Reachable State Space Sampling

The first step for the model generation is to sample the original netlist at transistor level with full BSIM accuracy. In order to keep the overall behavior, it is not sufficient to sample along some transient trajectories. Rather, the circuit has to be sampled in the reachable state space in an adequate manner. Thus, we use an algorithm to step through the state space [12], nonlinearly reducing the order like in [8] and calculating the reachability [12] of the sampled points. The result is a data set of sampled reachable points connected by a directed graph [13]. Additionally, the set contains the nonlinear consistent operating points $\vec{x}_P \in \mathbb{R}^n$, the conduction matrix $\underline{G}$ and the capacitance matrix $\underline{C}$, the input and output vectors $\vec{b}$ and $\vec{r}^T$, and the eigenvalues λ_i of the locally linearized system:

$$\begin{aligned} \underline{C} \cdot \dot{\vec{x}} + \underline{G} \cdot \vec{x} &= \vec{b} \cdot u \\ y &= \vec{r}^T \cdot \vec{x} \end{aligned} \tag{1}$$

This locally linear system is transformed via:

$$\begin{aligned} s \cdot \underline{E} \cdot \underline{C} \cdot \underline{F} \cdot \vec{x}_s + \underline{E} \cdot \underline{G} \cdot \underline{F} \cdot \vec{x}_s &= \underline{E} \cdot \vec{b} \cdot u \\ y &= \vec{r}^T \cdot \underline{F} \cdot \vec{x}_s \end{aligned} \tag{2}$$

to a Kronecker normal form with new state variables $\vec{x}_s \in \mathbb{R}^{n_r}$:

$$\begin{aligned} s \cdot \begin{bmatrix} \underline{I} & \underline{0} \\ \underline{0} & \underline{0} \end{bmatrix} \begin{bmatrix} \vec{x}_{s,Re} \\ \vec{x}_{s,\infty} \end{bmatrix} + \begin{bmatrix} \underline{\Lambda} & \underline{0} \\ \underline{0} & \underline{I} \end{bmatrix} \begin{bmatrix} \vec{x}_{s,Re} \\ \vec{x}_{s,\infty} \end{bmatrix} &= \begin{bmatrix} \tilde{\vec{b}}_{Re} \\ \tilde{\vec{b}}_{\infty} \end{bmatrix} \cdot u \\ y = \begin{bmatrix} \tilde{\vec{r}}^T_{Re} \tilde{\vec{r}}^T_{\infty} \end{bmatrix} \begin{bmatrix} \vec{x}_{s,Re} \\ \vec{x}_{s,\infty} \end{bmatrix} \end{aligned} \tag{3}$$

where $\underline{I}$ is the identity matrix, while $\underline{E}$ and $\underline{F}$ are the transformation matrices. $\underline{\Lambda}$ is a diagonal or band-diagonal matrix with numerically increasing generalized eigenvalues of the system. As indicated, transformed vectors are marked with a tilde (˜). Equation (3) is divided into a reduced part (subscript Re) and a neglected part (subscript ∞).

$\underline{F}$ can be computed from the right eigenvectors of the generalized eigenproblem, while $\underline{E}$ is a proper calculated matrix from the same problem. In order to reduce the system to a low order, the first n_r finite eigenvalues are used, while the remaining eigenvalues are neglected, thus handled like the infinity part of the system. In fact this realizes the nonlinear order reduction. Finally, the transformation matrices are split into two parts, which is later used in section III-C.:

$$\underline{F} = \left[\underline{F}_\lambda \underline{F}_\infty\right] \qquad \underline{E} = \begin{bmatrix} \underline{E}_\lambda \\ \underline{E}_\infty \end{bmatrix} \tag{4}$$

B. Linear Location Identification

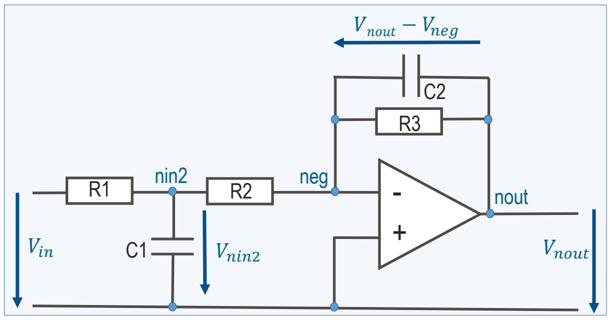

Fig. 2. A second order low-pass filter with a nonlinear limitation at 1.5V. The operational amplifier is a SPICE file consisting of 11 transistors in a 350nm CMOS technology [14] .

The n_r eigenvalues λ_i from each of the sampled points are used to identify the regions of the hybrid automata. This can be achieved in 2 steps. First, the eigenvalues are cluster into groups. The result of this clustering for the circuit shown in Fig. 2 is illustrated in Fig. 3. As shown, two groups can be identified. Note that this clustering is done with an extended k-means algorithm, and that the number of clusters is either determined by our algorithm or can be specified by the user for higher precision.

Second, a correction is done on this clustering in order to separate the regions which have the same eigenvalue, but belong to different spatial areas in the state space. For this example, our methodology separates group 2 into 2 separated regions, as the limiting behavior happens in both directions of the input voltage. This can be achieved by examining the connection graph of the state space. The result of this clustering is shown in Fig. 3. As illustrated, group 2 has been separated into two distinct regions. Thus, the data set consists of two groups of different eigenvalues, with one group having 2 distinct regions in the state space. The red points shown in Fig. 3, represent the DC operating points of the system. Note that these points where calculated for a input voltage ranging in the interval [-4,4] with a step size of 0.5V

Each of the colored data set in Fig. 3 represents a *location* of the generated hybrid automaton (HA). For each location of the HA, a representing DC point is chosen. For the example

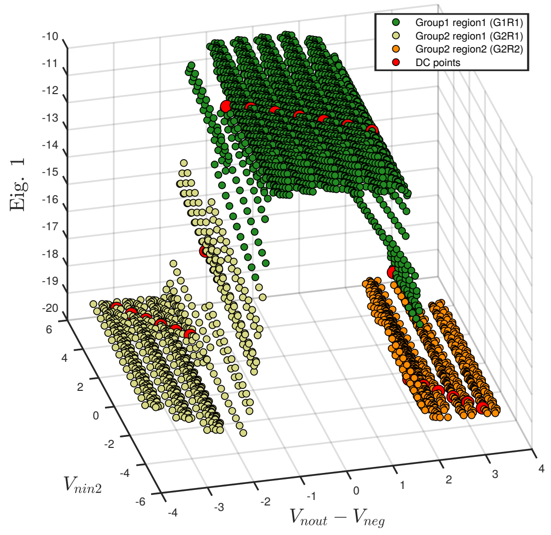

Fig. 3. State clustering into two groups. Group 2 has two regions (yellow and orange). Group 1 has only one region (dark green). As illustrated, the yellow and orange have nearly the same eigenvalues, but still represent 2 distinct regions. The red points represent the DC operating points. Note the missing dimension of the input voltage.

at hand, the representing DC point of G1R1 (group 1, region 1) is the one located at $V_{nin2} = 0$ and $V_{nout} - V_{neg} = 0$. The region G1R1 is referred to as the center region. The representing DC point for each of the other locations G2R1 and G2R2 can be chosen as the closest DC point with respect to location G1R1.

C. Calculating a State Space Representation of HA Locations

A location of the hybrid automata consists of an invariant, guards, reset and a state equation. Note that the invariants are only used to calculate the guards as seen later. All of these equations and variables should be calculated in the $\vec{x}_s$ domain. Thus, for further analysis the transformed state space ($\vec{x}_s$) for each of the locations of the HA has to be calculated. If we define the first linear system representation around the origin, the movement from one location to another will result in a jump of the state variables $\vec{x}_s$. This is due to the new coordinate system in the new location.

To prevent this jump, we propose to shift the *state variables* by a constant at each location, in order to simulate the new system behavior in the new coordinate system. For this purpose, a operating point in the $\vec{x}_s$ domain has to be calculated. With the DC points in the $\vec{x}$ domain at hand, the DC shift $\vec{x}_{S,DC,k}$ for each location k in the $\vec{x}_s$ domain can be calculated by solving the following equation for $\vec{x}_{S,DC,k+1}$:

$$\begin{aligned}\underline{F}_{\lambda,k}\vec{x}_{s,DC,k+1} = \vec{x}_{DC,k+1} - \vec{x}_{DC,k} + \underline{F}_{\lambda,k}\vec{x}_{s,DC,k} \\ + \underline{F}_{\infty,k}\underline{E}_{\infty,k}\vec{b}(u_{DC,k} - u_{DC,k+1})\end{aligned} \tag{5}$$

where $u_{DC,k}$ is the input voltage corresponding to the DC point at location k. The calculation for each location is done in a recursive manner starting from the most inner one, the center location, outwards. From this location (k=0), the DC shift ($\vec{x}_{S,DC,1}$) of the next closest location (k=1) is than calculated and so forth. The DC points are labeled by the locations they belong to: $\vec{x}_{s,DC,GR}$ or simply $\vec{x}_{s,GR}$ in the figures. At this point, only the DC points in the $\vec{x}_s$ domain have been calculated. For determining the guards and invariants of the HA, we have to transform all points of the original $\vec{x}$ domain into the new $\vec{x}_S$ domain. This is done for each point i by solving the following equation for $\vec{x}_{s,i}$:

$$\begin{aligned}\underline{F}_{\lambda,k}\vec{x}_{s,i} = \vec{x}_i - \vec{x}_{DC,k} + \underline{F}_{\lambda,k}\vec{x}_{s,DC,k} \\ + \underline{F}_{\infty,k}\underline{E}_{\infty,k}\vec{b}(u_{DC,k} - u_i)\end{aligned} \tag{6}$$

Where i stands for an index of a point in the location k.

1) Invariants: Having transformed all points into x_s domain, the invariants of the locations of the hybrid automaton can be identified. This is done by enclosing the points of a given location with a convex hull (see Fig. 4 and Fig. 5). Our methodology allows the user to choose the type of the convex hull. We allow three different types of representations for the convex hulls: Polytope, interval and zonotope representation. The polytope representation is the most accurate one, while the remaining two overapproximate. Depending on the type of the convex hull chosen, the invariants overapproximate differently the data set.

2) Guards: From the edges of the convex hulls, the guards to the neighbor locations can be identified. This can be done by searching for the presence of points from the neighbor locations to the edges of the convex hulls. For the guards, three different types of representation exist: Halfspace, Interval and zonotope representation. While guards of types zonotope and interval overapproximates in most cases the transitions, guards of type halfspace make clear cuts. For reachability analysis the first might be more suitable. However, since we are interested in simulating the hybrid automaton in Verilog, this paper will focus on halfspaces. Note that the type of invariant used affects the guard calculations.

Beside the type of guards, one can also chose the number of guards between the locations. Either all guards can be selected, or the guards are reduce to one dominant one. The dominant guard encloses the highest number of points to the surrounding location. Note that the number of guards depends on the number of edges of the convex hull.

In Fig. 4 our methodology with an invariant of type polytope and guards of types halfspaces is illustrated for the circuit shown in Fig. 2. Note that only the dominant guards are considered. This setup is used throughout this paper. To compare this classification to a different region identification, Fig. 5 has been created using intervals as invariants and guards. As illustrated, the invariants of the locations are overapproximated, thus decreasing the number of available guards compared to the previous region identification. Note that the guards have a negligible thickness (0.01 for the example at hand).

D. Dynamics of the Hybrid Automata

Along with the guards, the operating points in both domains, the eigenvalues and eigenvectors, we are now able to specify

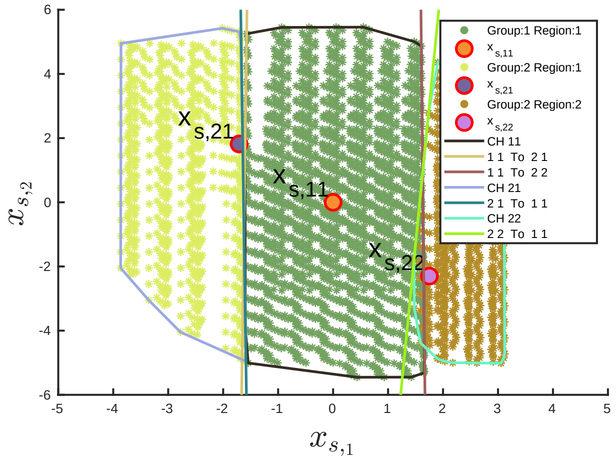

Fig. 4. State space of the example illustrated in Fig. 2 using polytopes as invariants and halfspaces as guards. The number of edges of the convex hulls at G1R1, G2R1 and G2R2 are 29, 10 and 19 respectively.

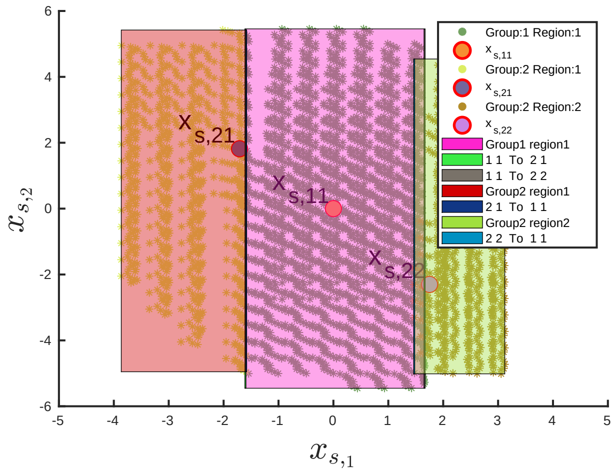

Fig. 5. State space of the example illustrated in Fig. 2 using intervals as invariants and guards.

the dynamics of the hybrid automaton of our running example from Fig. 2 as follows:

$$\dot{\vec{x}}_{s,LOC} = \underline{\Lambda} \cdot \Delta\vec{x}_{s,LOC} + \underline{E}_{\lambda} \cdot \vec{b} \cdot \Delta u_{LOC} \tag{7}$$

The index 'LOC' in (7) represents a location of the hybrid automaton in the $\vec{x}_s$ domain. The system matrix $\underline{\Lambda}$ contains the n_r eigenvalues (see eq. (3)) and is in block diagonal form. It is important to notice that the equations of the hybrid automaton at each location are linearized around the DC points in the $\vec{x}_s$ domain, as indicated by the term Δ. Thus, $\Delta\vec{x}_{s,LOC} = \vec{x}_{s,LOC} - \vec{x}_{s,DC,LOC}$. This means that after taking a guard from location G1R1 to location G2R1 for example, the DC points in the $\vec{x}_s$ domain change from $\vec{x}_{s,DC,G1R1}$ to $\vec{x}_{s,DC,G2R1}$ as illustrated in Fig. 4.

For the example at hand, the locations of the hybrid automata at G1R1 and G2R1 are represented in equations (8) and (9), respectively. For G2R2, the state equation would be similar to (9) with the difference, that $u_{DC,LOC}$ and $\vec{x}_{s,DC,LOC}$ vary. These equations are later transformed into separated row equations for the Verilog-A simulation.

$$\dot{\vec{x}}_{s,G1R1} = \begin{bmatrix} -10.4 & 0 \\ 0 & -1099.7 \end{bmatrix} \Delta\vec{x}_{s,G1R1} + \begin{bmatrix} -8.89 \\ 999.9 \end{bmatrix} \cdot \Delta u \tag{8}$$

$$\dot{\vec{x}}_{s,G2R1} = \begin{bmatrix} -18.6 & 0 \\ 0 & -1100.3 \end{bmatrix} \Delta\vec{x}_{s,G2R1} + \begin{bmatrix} -9.51 \\ 999.1 \end{bmatrix} \cdot \Delta u \tag{9}$$

such that:

$$\begin{aligned} \vec{x}_{s,DC,G1R1} &= \begin{bmatrix} 0 & 0 \end{bmatrix}^T & u_{DC,G1R1} &= 0 \\ \vec{x}_{s,DC,G2R1} &= \begin{bmatrix} -1.71 & 1.81 \end{bmatrix}^T & u_{DC,G2R1} &= 2 \end{aligned}$$

As stated earlier, there are different approaches to represent the guards. Regardless of the representation, the guards are transformed into inequalities at the given locations. For example, the guards at location G1R1 to G2R1 and G2R2 are represented in the following 2 inequalities. Note that the subindex 'DC' has been omitted.

$$\begin{aligned} 0.42(x_{s,1} - x_{s,1,G1R1}) + 3.13 \cdot 10^{-4}(x_{s,2} - x_{s,2,G1R1}) &< -0.71 \\ -0.92(x_{s,1} - x_{s,1,G1R1}) + 2.78 \cdot 10^{-3}(x_{s,2} - x_{s,2,G1R1}) &< -1.81 \end{aligned}$$

E. Hybrid Automata in Verilog-A

Having computed the HA, the system can now be reformulated in Verilog-A. As indicated earlier, the invariants are neglected as the guards are sufficient to use. Therefore, the guards are wrapped in if statements and used for the identification of the current location. Furthermore, each location of the hybrid automaton contains a set of variables specific to it. These variables include each element of the eigenvectors, the operating points in both domains, the input vector $\vec{b}$ and the elements of the system matrix indicated in (7). Additionally, in order to obtain the n node voltages and currents in the original $\vec{x}$ domain, a back transformation from the $\vec{x}_s$ space must be performed. This can be done with (10), where k indicates the location of the system:

$$\vec{x} = \underline{F}_{\lambda,k}\vec{V}(x_s) + \vec{x}_{DC,k} + \underline{F}_{\infty,k}\underline{E}_{\infty,k}\vec{b}(u - u_{DC,k}) \tag{10}$$

All these equations are automatically generated into a Verilog-A behavioral model with n_r ddt operators and n_{LOC} locations. The inputs and output variables are electrical nodes.

IV. Experimental Results

A. Examples

The experimental results are performed on a Intel Core I5-7300HQ CPU @ 2.50GHz with 16GB ram using the SPICE simulator Gnucap [15] together with the Verilog-A compiler ADMSXml [16]. The circuit from Fig. 2 is compared against the behavioral Verilog-A model with 3 locations as previously elaborated. The results of a transient simulation of this comparison is shown in Fig. 6. As illustrated, there are some deviations during the switching from the linear to the limiting mode. However, by using a HA with 5 locations this error can be significantly reduced. In both cases, due to the forward linear description of the HA in Verilog-A, the

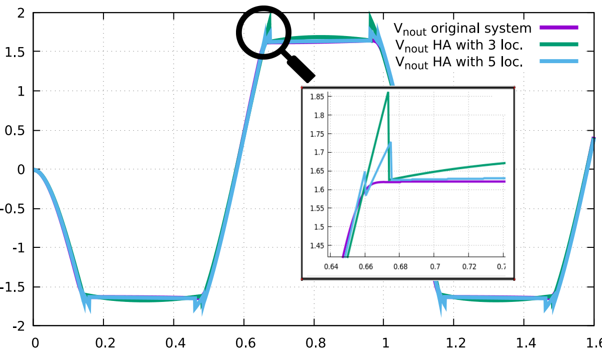

Fig. 6. SPICE simulation results of the original netlist versus an hybrid automaton with 3 and 5 locations respectively. The input voltage to the circuit is a sin wave with an amplitude of 5V and a frequency of 1 Hz. The output voltage differs depending on the modeling effort

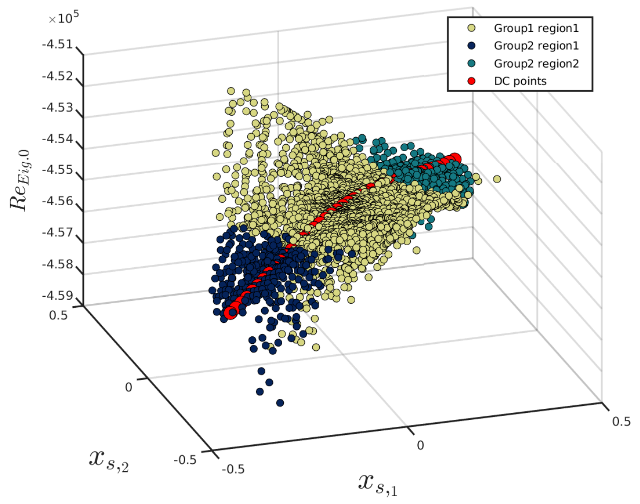

Fig. 7. Real part of the first eigenvalue plotted versus the shifted and transformed state space of the gm/C filter. The different locations of the HA are plotted in different colors. The DC operating points are plotted in red.

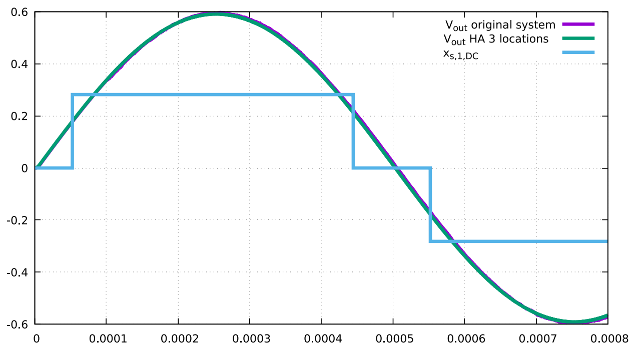

Fig. 8. Simulation results of the netlist versus the generated HA with 3 locations of the gm/C filter. The input of both systems is a sine signal with an amplitude and offset at 1.52V and a frequency at 1 KHz.

simulator does not react on the edges with strongly reduced time steps. Moreover, these edges can be reduced or even eliminated by several techniques: increasing the number of locations as mentioned, increasing the order of the system to include the input, shifting the guards to the operating points (used in the next example) or by expanding the convex hull. As a second example, we abstracted an industrial full differential second order gm/C filter. The netlist of this filter consists of 38 nodes and 42 transistors. The hybrid automaton generated by our approach consists of 3 locations as shown in Fig. 7. The circuit has 2 dominant state variables implying 2 eigenvalues. These eigenvalues are complex and vary as illustrated in Fig. 7. Note that only the real part of the first eigenvalue is illustrated. The simulation results of the generated Verilog-A model versus the original transistor netlist are shown in Fig. 8. Additionally, the first coordinate of the shifted state $x_{S,1,DC,LOC}$ is plotted. A change in this value indicates that the HA has changed the location.

A summary of all computed example circuits along with some properties and performances is illustrated in Table I. Note that the number of time steps during transient simulation is comparable between the circuit and the model indicating no problem with location switching in the model. Moreover, the low number of Newton iterations indicates a faster convergence, which is expected from a piecewise linear behavior. The speed up is significant and varies between 30 for smaller transistor netlist up to 130 for bigger netlists. Furthermore, we expect even bigger speedups for more sophisticated netlists on modern process nodes. Note that the modeling time, which is the time that the algorithm spends from III-B to III-E, is reasonable for mid size circuits. As indicated in Table I, the models show a favorable accuracy compared with the original netlists.

B. Formal Verification of the Generated Models

To close this tool chain, we formally verify the generated models. This is done by using an equivalence checking tool [12]. In Fig. 9 the deviations of the output voltage in the reachable state space between the original transistor netlist and the generated model for the first example in section IV-A are illustrated. As observed, the closer the system behavior is to the selected sample points for linearization, the more accurate the results are. At the far borders of the state space, the behavior deviates from the original model. But as stated, their are approaches which can improve this behavior e.g. increase the number of locations. Additional improvement could be optimizing the used eigenvalues and the guards' positions. The resulting errors are in column *Accuracy* in Table I.

V. Conclusion

In this paper, we have presented an approach to automatically generate abstracted Verilog-A models based on netlists with full BSIM accuracy. Different settings can be applied to this abstraction, resulting in either more abstracted or preciser models. To analyze the accuracy, the generated models are verified by equivalence checking. The methodology key improvement is the significant speed up of the simulation time

TABLE I
COMPARISON OF THE TRANSISTOR-LEVEL CIRCUITS WITH THE PRESENTED BEHAVIORAL MODELS.

Circuit	Model	Complexity	Sample points	Sampling + Data read time[s]	Modeling time[s]	Locations	Accuracy†	Newton iterations	Time steps	Run time[s]	Speed up
Differential mode Gm-C filter	Netlist	11 OTAs, 46 trans., 38 nodes	-	-	-	-	-	50204	10007	17.41	-
	HA	916 lines 5 nodes	17997	372.2+127.4	6.05	3	0.81%	39221	10006	0.14	124.3
Lowpass based on 2-stage OTA	Netlist	1 OTA, 11 trans., 19 nodes	-	-	-	-	-	100187	20009	13.31	-
	HA	403 lines 3 nodes	312	0.34+1.68	1.51	3	3.6%	61496	20006	0.24	55.45
Second order low-pass filter	Netlist	1 OTA, 11 trans., 19 nodes	-	-	-	-	-	62572	12012	6.06	-
	HA	217 lines 4 nodes	4017	43.12+10.97	4.46	3	15.2%	58202	12057	0.20	30.29
	HA	765 lines 4 nodes	4017	43.12+10.97	6.04	5	6.63%	58673	12121	0.21	28.85

† Maximum of the relative dynamic and the output error from equivalence checking. The output error is the same as y_{error} from Fig. 9

while maintaining an accurate model and having an full automatic approach. Moreover, due to the fact that the generated models are pin compatible they are suitable candidates for simulation and formal verification on higher abstraction levels.

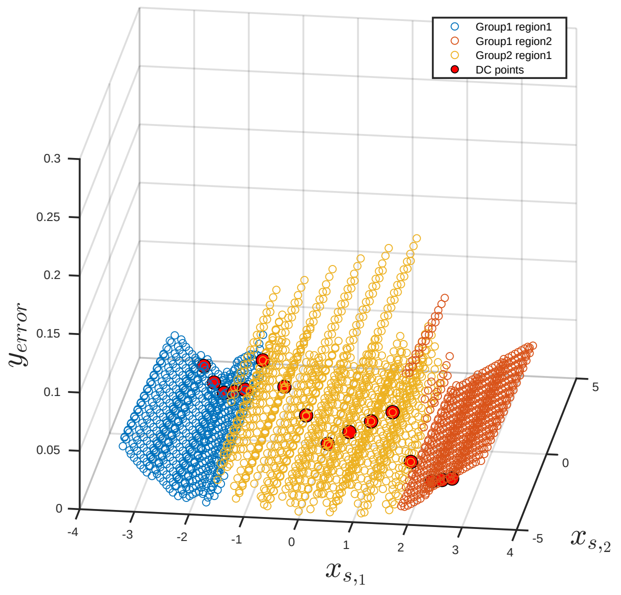

Fig. 9. Output deviation (y_{error}) between the generated model, with three locations, versus the original netlist for the example form Section IV-A. The three used DC points for modeling exhibit the lowest errors.

VI. ACKNOWLEDGMENT

This paper presents result of the project faveAC funded by the DFG under the project number 286525601.

REFERENCES

[1] G. G. Gielen and J. R. Phillips, “Simulation and modeling for analog and mixed-signal integrated circuits,” *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology*, pp. 455–477, 2016.
[2] B. Antao and F. El-Turky, “Automatic Analog Model Generation for Behavioral Simulation,” *CICC: Custom Integrated Circuit Conference*, pp. 12.2.1–12.2.4, 1992.
[3] C. Borchers, L. Hedrich, and E. Barke, “Equation-Based Behavioral Model Generation for Nonlinear +Analog Circuits,” in *DAC: Design Automation Conference*, pp. 236–239, 3–7 June 1996.
[4] F. Fernandez, B. Perez-Verdu, and A. Rodriguez-Vazquez, “Behavioral Modeling of PWL Analog Circuits Using Symbolic Analysis,” *ISCAS ’98: IEEE International Symposium on Circuits and Systems*, vol. VI, pp. VI–17 – VI–20, 1998.
[5] L. Näthke, V. Burkhay, L. Hedrich, and E. Barke, “Hierarchical automatic behavioral model generation of nonlinear analog circuits based on nonlinear symbolic techniques,” in *DATE: Design Automation and Test in Europe*, pp. 442–447, 2004.
[6] D. Zaum, S. Hoelldampf, M. Olbrich, E. Barke, I. Neumann, and S. Schmidt, “The PRAISE approach for accelerated transient analysis applied to wire models,” in *BMAS: Behavioral Modeling and Simulation Workshop*, pp. 120–125, 2009.
[7] M. Isaksson, D. Wisell, and D. Ronnow, “Wide-band dynamic modeling of power amplifiers using radial-basis function neural networks,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 53, no. 11, pp. 3422–3428, 2005.
[8] J. Phillips, J. Afonso, A. Oliveira, and L. M. Silveira, “Analog macromodeling using kernel methods,” in *ICCAD: International Conference on Computer-aided design*, p. 446, 2003.
[9] A. V. Karthik, S. Ray, P. Nuzzo, A. Mishchenko, R. K. Brayton, and J. Roychowdhury, “ABCD-NL: Approximating continuous non-linear dynamical systems using purely Boolean models for analog/mixed-signal verification.,” in *ASP-DAC*, pp. 250–255, 2014.
[10] A. Singh and P. Li, “On behavioral model equivalence checking for large analog/mixed signal systems,” in *ICCAD*, pp. 55–61, 2010.
[11] S. Steinhorst and L. Hedrich, “Equivalence checking of nonlinear analog circuits for hierarchical AMS System Verification,” in *VLSI-SoC: VLSI and System-on-Chip*, pp. 135–140, IEEE, 2012.
[12] S. Steinhorst and L. Hedrich, “Advanced methods for equivalence checking of analog circuits with strong nonlinearities,” *Formal Methods in System Design*, vol. 36, no. 2, pp. 131–147, 2010.
[13] S. Steinhorst and L. Hedrich, “Trajectory-directed discrete state space modeling for formal verification of nonlinear analog circuits,” in *ICCAD: International Conference on Computer-Aided Design*, pp. 202–209, 2012.
[14] “Ams hitkit - Process Design Kit (PDK) for Cadence Tools, Circuit Simulation, Device Models.” http://asic.ams.com/hitkit/index.html.
[15] A. Davis, “An overview of algorithms in Gnucap,” in *University/Government/Industry Microelectronics Symp.*, pp. 360–361, 2003.
[16] ADMS Automatic Device Model Synthesizer, “https://sourceforge.net/projects/mot-adms/.”