Compositional Behavioral Modeling of Analog Neural Networks

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*—This paper contributes to the automatic abstraction of analog circuits at transistor level. Specifically, this paper targets neuronal networks (NNs). As these circuits consist of millions of repeated neurons, simulation as well as verification routines are prohibitively time consuming. However, these netlists usually consist of repeated arrangements of neurons, which can be individually considered as subsystems. Starting with a neuron described as a Spice netlist, an abstraction methodology is presented that automatically generates an accurate behavioral model as a hybrid automaton (HA) in SystemC-AMS/Verilog-A while still preserving the internal voltages and currents of the subsystem. The abstracted model can replace the neuron in simulation as well as in verification routines with significant speedup factors while still achieving high accuracy.**

***Index Terms*—compositional abstraction, neuron, neuronal network, hybrid automaton, behavioral modeling, Verilog-A**

I. Introduction

Nowadays, NNs are gaining more importance due to their application spectrum. From applications including predication and association in autonomous driving, speech recognition, and process control, modern NNs are used even in the classification tasks in medicine. Thanks to the internet of things (IOT), the application spectrum of NNs is and will continue to increase. Even though NNs are widely applicable and user-friendly, the circuit simulation of such large circuits consisting of millions of neurons is a computational extensive task. Moreover, due to the state space explosion, a formal verification of such analog circuits is not feasible. In this contribution, we target these problems by presenting an automatic abstraction methodology capable of generating accurate behavioral models suitable for both, verification and simulation.

The abstraction of a single neuron, as described in Sec. III, is examined in this paper. Our methodology consists of two steps: sampling the Spice netlist and generating a HA from the sampled data. In Sec. IV the sampling process is examined, while in Sec. V the behavioral abstraction is described. A comparison between the abstract model and the Spice netlist is given in Sec. VI. Finally, a conclusion is stated in Sec. VII.

II. Previous Work

The abstraction of a neuron can speedup simulations routines and permit verification routines if the model is accurate enough. According to [1], formal verification of AMS circuits typically involves working on a higher level of abstraction, as this typically results in significant speedup factors. As NN consist of millions of neurons, an automated abstraction approach is mandatory. Different approach exist that generate behavioral models [2]–[5]. These techniques are not targeting hybrid automata and mainly improve the simulation speed. For high level continuous systems, methods modeling the analog circuit as a hybrid system are widely used [6]–[9]. These methods are able to handle up to tens of state variables, if the underlying locations use linear ordinary differential equations (ODEs) to describe the system behavior. To close the chain of proof at transistor level, HAs are usually not suitable as the ODEs become nonlinear differential algebraic equations (DAEs). In [10], an automated abstraction approach that generates accurate HAs from Spice netlists with BSIM 4 accuracy was presented. As the HAs have linear system descriptions, they are suited for compositional abstraction. In the following the applicability of this approach on a NN is examined by abstracting first a single neuron, followed by compositionally linking the abstracted models.

978-0-7381-4394-1/21/$31.00 ©2021 IEEE

III. Running Example: Neuron at Transistor Level

To demonstrate our approach, consider the single neuron shown in Fig. 1 with an adjustable ReLU activation function. The shown neuron is described in Spice with BSIM4 accuracy.

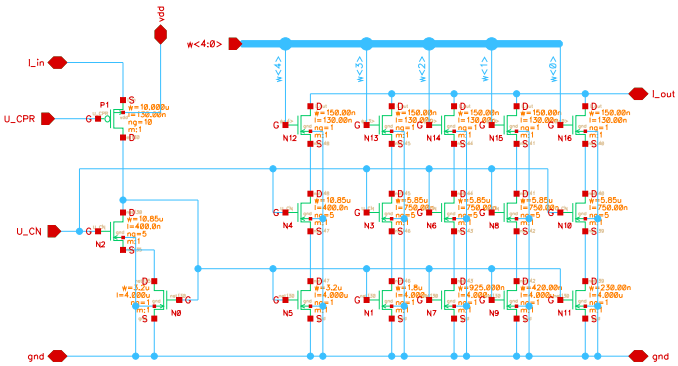


Fig. 1: Schematic of the neuron circuit on transistor level implementing a ReLU function.

IV. Sampling the State Space with *Vera*

The target of the abstraction methodology is to build a HA with a finite set of locations as described later in Section V.

For that, the circuit is sampled numerically using *Vera* [11]. The starting point is a nonlinear netlist description in Spice at transistor level. Significant data, such as the nonlinear consistent operating point, the eigenvalues of the linearized system and their nonlinear evolution, and a local linearized system description for each sample point are computed by *Vera* during the sampling at transistor level. The circuit is described by a system of nonlinear differential-algebraic equations:

$$\boldsymbol{f}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), u(t)), \tag{1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ represents the vector of unknown voltages and currents in the original state space $\mathcal{S}_o$ of the system and $u(t)$ represents the input signal. For simplicity Eq. (1) is given for a SISO system. *Vera* steps through the state space as described in [11], linearizing the system locally and calculating the reachability of the sampled points of the overall nonlinear system. The result is a data set of sampled reachable points connected by a directed graph [12]. The sampled data additionally contains the operating points $\boldsymbol{x}_{DC} \in \mathbb{R}^n$, the conduction and capacitance matrices $\boldsymbol{G}$ and $\boldsymbol{C}$, and the input vector $\boldsymbol{b}$. The sampling results in the linearized system equation for each sample point:

$$\boldsymbol{C}\Delta\dot{\boldsymbol{x}} + \boldsymbol{G}\Delta\boldsymbol{x} = \boldsymbol{b}\Delta u \tag{2}$$

With $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_{DC}$ and $\Delta u = u - u_{DC}$. With the transformation matrices $\boldsymbol{F}$ and $\boldsymbol{E}$, the system is transformed to a Kronecker form. Note that $\boldsymbol{F}$ is calculated from the right eigenvectors of the generalized eigenvalue problem associated with Eq. (2), while $\boldsymbol{E}$ is a proper calculated matrix from the same problem. For that, first the state space vector $\boldsymbol{x} \in \mathbb{R}^n$ is transformed from the original state space $\mathcal{S}_o$ to the new state space $\mathcal{S}_s$ with the state space vector $\boldsymbol{x}_s \in \mathbb{R}^n$ conform to:

$$\Delta\boldsymbol{x} = \boldsymbol{F}\Delta\boldsymbol{x}_s \tag{3}$$

Substituting Eq. (3) in Eq. (2), and multiplying the equation with $\boldsymbol{E}$ yields the Kronecker form:

$$s\begin{bmatrix}\boldsymbol{I}_1 & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{N}\end{bmatrix}\Delta\boldsymbol{x}_s + \begin{bmatrix}-\boldsymbol{J} & \boldsymbol{0}\\ \boldsymbol{0} & -\boldsymbol{I}_2\end{bmatrix}\Delta\boldsymbol{x}_s = \boldsymbol{E}\boldsymbol{b}\Delta u \tag{4}$$

Where $\boldsymbol{I}_1 \in \mathbb{R}^{r\times r}$ and $\boldsymbol{I}_2 \in \mathbb{R}^{(n-r)\times(n-r)}$ are identity matrices, the matrix $\boldsymbol{J} \in \mathbb{R}^{r\times r}$ is in general in the Jordan normal form corresponding to the finite eigenvalues, and $\boldsymbol{N} \in \mathbb{R}^{(n-r)\times(n-r)}$ is a nilpotent matrix corresponding to the infinite eigenvalues from the underlying generalized eigenproblem. Hence, the initial system with n variables has r dynamic variables and $(n-r)$ algebraic ones. Moreover, η represents the order of nilpotency of $\boldsymbol{N}$. By performing a dominant pole reduction similar to [13] on the obtained equation, the large order resulting from parasitic poles is reduced to the functional needed. That is, the order is reduced from r to m. For simplicity, two assumptions are made: 1) the finite eigenvalues are distinct, and 2) the index of nilpotency is $\eta \leq 1$. Thus, the following equation is obtained:

$$\underbrace{s\begin{bmatrix}\boldsymbol{I}_\lambda & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}}_{\boldsymbol{ECF}}\begin{bmatrix}\Delta\boldsymbol{x}_{s,\lambda}\\ \Delta\boldsymbol{x}_{s,\infty}\end{bmatrix} + \underbrace{\begin{bmatrix}-\boldsymbol{\Lambda} & \boldsymbol{0}\\ \boldsymbol{0} & -\boldsymbol{I}_\infty\end{bmatrix}}_{\boldsymbol{EGF}}\begin{bmatrix}\Delta\boldsymbol{x}_{s,\lambda}\\ \Delta\boldsymbol{x}_{s,\infty}\end{bmatrix} = \underbrace{\begin{bmatrix}\tilde{\boldsymbol{b}}_\lambda\\ \tilde{\boldsymbol{b}}_\infty\end{bmatrix}}_{\boldsymbol{Eb}}\Delta u \tag{5}$$

With the identity matrices $\boldsymbol{I}_\lambda \in \mathbb{R}^{m\times m}$ and $\boldsymbol{I}_\infty \in \mathbb{R}^{(n-m)\times(n-m)}$, and the diagonal (or band-diagonal) matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{m\times m}$ filled with the m finite eigenvalues. Note that transformed vectors are marked with a tilde ($\tilde{}$). Eq. (5) can be split into a dynamic part with subscript (λ) and the state vector $\boldsymbol{x}_{s,\lambda} \in \mathbb{R}^m$ in the reduced state space $\mathcal{S}_\lambda$, and an algebraic part with subscript (∞) and the state vector $\boldsymbol{x}_{\infty,\lambda} \in \mathbb{R}^{(n-m)}$ in the $\mathcal{S}_\infty$ space. For simplicity, $\boldsymbol{x}_{s,\lambda}$ will be denoted as $\boldsymbol{x}_\lambda$. Hence, there exist a relationship between the three state spaces and their corresponding state space vectors given by:

$$\Delta\boldsymbol{x} = \begin{bmatrix}\boldsymbol{F}_\lambda & \boldsymbol{F}_\infty\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{x}_{s,\lambda}\\ \Delta\boldsymbol{x}_{s,\infty}\end{bmatrix} \tag{6}$$

This shows the strength of our approach: to calculate the state vector $\boldsymbol{x}$ using Eq. (6), only the differential part from Eq. (5) must be calculated. The algebraic part can be found using the second row of Eq. (5). However, Eq. (5) is only pointwise valid. Thus, the objective is to model in each location of the HA the system behavior using a generalized form of Eq. (5) valid for a set of sample points of the same location.

V. Automatic Behavioral Abstraction to HA

The abstraction approach aims to deploy an abstract model in the form of a HA. For that, the finite set of locations are first determined. In each location the current invariant, the guards to the neighbor locations, and the system behavior given by the linear state space representation are identified. Additionally, the starting location of the HA and the jump conditions associated with the guards are found. A formal definition of the HA is similar to [14], with some restrictions on the jumps and guards. We aim to deploy a deterministic HA, that is, the invariants are only used to find the guards, a transition occurs immediately once a guard becomes valid, and during simulation the HA is only in one location. In the following, the detailed construction process is examined.

A. Finding the Locations of the HA

After the sampling process, a data set is obtained containing:

(a) all the $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{x}_\lambda \in \mathbb{R}^m$ sampled values from the $\mathcal{S}_o$ and $\mathcal{S}_\lambda$ state spaces, respectively
(b) the transformation matrices $\boldsymbol{F} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{E} \in \mathbb{R}^{n\times n}$
(c) the eigenvalues of the linearized system after model order reduction $\boldsymbol{\Lambda} \in \mathbb{R}^{m\times m}$

Additionally, the data set contains as well the input vector $\boldsymbol{b} \in \mathbb{R}^n$ and the directed edges between the sampled points contained in a directed graph. For the neuron circuit from Fig. 1, the system has $n = 35$ nodal voltages and currents. Upon linearized, the system has a dynamic order of $r = 9$, with eigenvalues ranging from -8.68×10^1 to -2.6503×10^{15}, which is reduced by *Vera* to a reduced order of $m = 2$.

The next task is to find the finite set of locations $loc \in Loc$, where the sampled point exhibit similar dynamic behaviors. In order to find the locations of the HA, a group identification is performed that clusters points with similar eigenvalues into the same groups. Our algorithm uses an extended version of k-means clustering, where the number of clusters is either determined by the algorithm using the silhouette coefficient or

Copyright © AJSE Publishers

provided by the user for a higher accuracy. For the neuron from Fig. 1 the result of the clustering is illustrated in Fig. 2. As

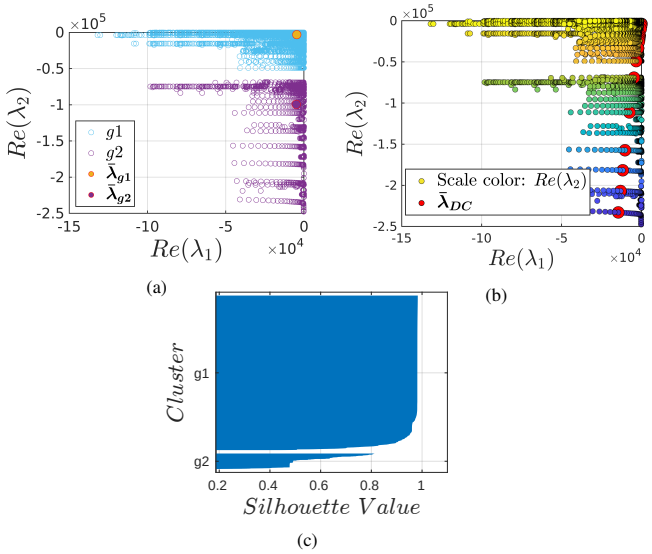


Fig. 2: In (a) the clustered eigenvalues of the linearized system are shown, while (b) presents the unclustered eigenvalues. (c) presents the corresponding Silhouette coefficients of the data points.

illustrated, two groups $g1$ and $g2$ were identified. According to Fig. 3b, the system is well clustered, especially for $g1$.

The second step required to find the locations of the HA, is the region identification. The region identification splits the obtained groups into regions. This is necessary in case several disjoint portions in the $\mathcal{S}_\lambda$ space belong to the same group. However, as Fig. 3 shows, this is not the case for the current example. In general, by using the directed graph obtained by the sampling process, or by using a clustering algorithm such as dbScan on the obtained groups, regions can be found.

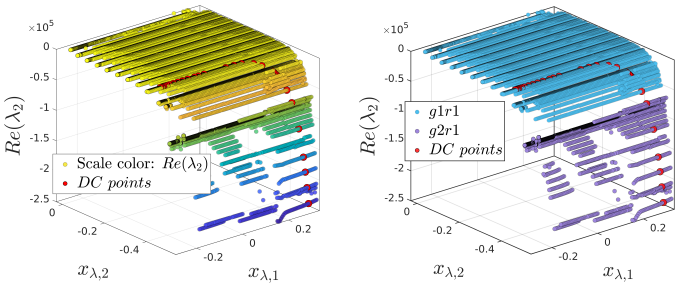


Fig. 3: $\mathcal{S}_\lambda$ space illustrated against the real part of the second eigenvalue of the linearized system. The large red points show the DC points. In (a) the initial unclustered data set is illustrated, while (b) presents the result of the region identification.

Together, the group and region identification compose the location identification. With gj denoting the j^{th} group and rk the k^{th} region, a location $loc_i \in Loc$ of the HA is denoted as:

$$loc_i = gjrk \tag{7}$$

For the current example, all point in a group gj belong to the same region $r1$. Hence, the locations of the HA are $Loc = \{g1r1, g2r1\}$ corresponding to a linear location $g1r1$ modeling the firing behavior of the neuron, and a blocking location $g2r1$ where the neuron is inactive.

B. Invariants and Guards of the HA

With the HA's locations at hand, the invariants and guards are identified next. As the data set sampled by *Vera* is distributed in accordance to the identified locations, the invariants (inv_{gjrk}) are simply found by enclosing the sampled points in the $\mathcal{S}_\lambda$ space with polytopes, zonotopes, or interval hulls. For the current application, polytopes are used (see Fig. 4). As we restricted the scope of this paper to deterministic HAs, invariants are only used to find the guards. Note that optionally, the model can issue warnings once the HA leaves an invariant.

To find the guards, the edges (facets in general) of the invariants are examined for the presence of points from the neighbor locations. In case the number of sampled points is above a specified tolerance value, the edge is taken as a guard. In general, guards can be modeled as halfspaces, polytopes, zonotopes, or interval hulls. For this paper, guards are always modeled as halfspaces. The l^{th} guard from the current location loc_i to the target location loc_t is denoted as:

$$grd_l : loc_i \rightarrow loc_t \qquad | \ loc_i, loc_t \in Loc \tag{8}$$

For the neuron from Fig. 1, the result of the abstraction approach is illustrated in Fig. 4. Note that $g1r1$ represents the starting location, as it contains the reference point $\boldsymbol{x}_\lambda = \mathbf{0}$.

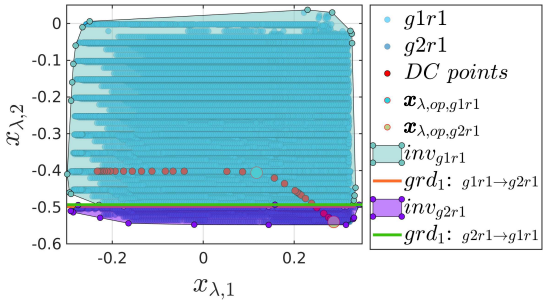


Fig. 4: $\mathcal{S}_\lambda$ state space of the HA with two locations.

C. Dynamics of the HA

Next, the dynamic system behavior in each location of the HA is described using the linear state space representation. For that, the operating point in each location is first identified. From the set of DC points (red points in Fig. 4), suitable operating points are chosen ($\boldsymbol{x}_{\lambda,op,gjrk}$). With these operating points, the system behavior in a location $loc_i \in Loc$ of the HA is described as:

$$\Delta\dot{\boldsymbol{x}}_\lambda = \boldsymbol{A}_{loc_i}\Delta\boldsymbol{x}_\lambda + \boldsymbol{B}_{loc_i}\Delta u\,, \tag{9}$$

such that:

$$\Delta\boldsymbol{x}_\lambda = \boldsymbol{x}_\lambda - \boldsymbol{x}_{\lambda,op,loc_i} \qquad \Delta u = u - u_{op,loc_i} \tag{10a}$$

$$\boldsymbol{A}_{loc_i} = \bar{\boldsymbol{\Lambda}}_{loc_i} \qquad \boldsymbol{B}_{loc_i} = \bar{\boldsymbol{E}}_{\lambda,loc_i}\bar{\boldsymbol{b}}_{\lambda,loc_i} \tag{10b}$$

Matrices with subscript λ correspond to sub-matrices belonging to $\boldsymbol{x}_\lambda$ (see Eq. (5)). $\boldsymbol{A}$ and $\boldsymbol{B}$ are computed from the mean values of the matrices of the sampled points belonging to the same location. For the running example, as the poles are always real, $\bar{\boldsymbol{\Lambda}}_{loc_i}$ is a diagonal matrix with the mean of the eigenvalues for each location (see different colored dots in Fig. 2a).

Copyright © AJSE Publishers

Using Eq. (9), $\boldsymbol{x}_\lambda$ is calculated in the $\mathcal{S}_\lambda$ space. In order to obtain the values of the nodal voltages and currents in the original state space $\mathcal{S}_o$, $\boldsymbol{x}$ must be calculated. This is done by a back-transformation obtained by combining Eqs. (5, 6):

$$\boldsymbol{x} = \boldsymbol{x}_{op,loc_i} + \bar{\boldsymbol{F}}_{\lambda,loc_i}\Delta\boldsymbol{x}_\lambda - \bar{\boldsymbol{F}}_{\infty,loc_i}\bar{\boldsymbol{E}}_{\infty,loc_i}\bar{\boldsymbol{b}}_{\infty,loc_i}\Delta u \quad (11)$$

D. Model deployment in Verilog-A/SystemC-AMS

The last modeling step is the deployment of the HA in the target language. For the Verilog-A models, the system behavior can be described using nodal equations and the *ddt* operator, while the SystemC-AMS models make use of the Time-Data-Flow model of computation (TDF-MoC) and use the TDF solver for the state space description (sca_tdf::sca_ss). In both cases, the guards are realized using if conditions. Once a guard becomes valid, the HA performs a location transition activating the jump condition by changing the operating points in Eq. (10a). In the new location, the system behavior is described by the corresponding matrices in Eqs. (9, 11).

VI. Results

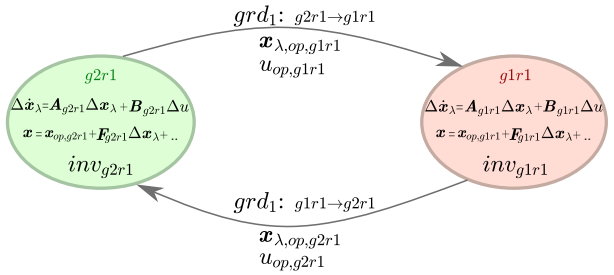


Fig. 5: Generated HA with two locations: a linear location ($g2r1$) and a nonlinear limiting location ($g1r1$).

As shown in Fig. 5, an abstract model in the form of a HA was generated in the previous sections from the circuit of a single neuron (see Fig. 1). The HA hast two locations: $g1r1$ and $g2r1$. During simulation, the HA starts from the center location $g1r1$ and computes the dynamic behavior using Eq. (9) with the corresponding values for Eq. (10) at $g1r1$. During each time step, the nodal voltages and currents (x values in $\mathcal{S}_o$) are calculated using the back-transformation Eq. (11) with the values of the matrices and operating points corresponding to $g1r1$. In case the guard $grd_1 : g1r1 \rightarrow g2r1$ becomes valid, that is:

$$\begin{aligned} 0 * \Delta\boldsymbol{x}_{\lambda,1} + 0.1650 * \Delta\boldsymbol{x}_{\lambda,2} < -0.01275 \\ \boldsymbol{x}_{\lambda,2} - (-0.4207) < -0.0773 \end{aligned} \quad (12)$$

the HA performs a transition to the location $g2r1$. Once the HA is in location $g2r1$, the corresponding values are used in Eqs. (9, 11). In case the guard of a location stays invalid during a simulation, the HA stays in the current location till the simulation time has elapsed. In the following, several examples are handled that use all this model, but with different arrangements and number of instances. All simulation were performed on a Linux PC with a four core i5-7300HQ CPU at 2.50 GHZ and 16 GB of RAM.

First, the generated HA (in Verilog-A) is compared against its original Spice netlist. The result is shown in Fig. 6. The last row of this figure shows the input voltage provided to both systems. Both models are simulated for 1 s with a timestep of 0.1 ms. As observed, the output voltages (second row) of both systems are nearly identical. Only immediately after/before a location transition of the HA, as indicated by the vertical purple lines, the voltages deviate. This is also shown in the first row of Fig. 6, which illustrates the output difference δ_y between the systems.

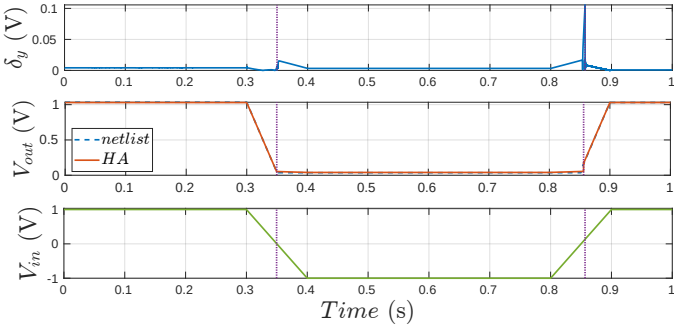


Fig. 6: Result of simulating the generated HA and the Spice netlist. The vertical lines indicated a location transition of the HA.

The detailed results are given in Table I (ex.1). Note that the HA can as well reconstruct internal voltages, however, this is skipped here.

A second example (ex.2) uses the same neuron in a configuration shown in Fig. 7. Three instances of the previous

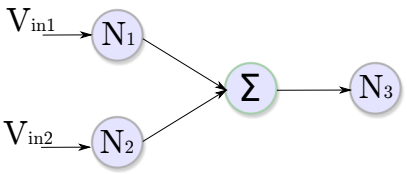


Fig. 7: Simple neuronal network consisting of three neurons.

generated HA are used to build the circuit. The circuit is compared to a Spice circuit with three neurons. For an input voltage as shown in the third row of Fig. 8, the result behave similar to the previous example for the same simulation conditions.

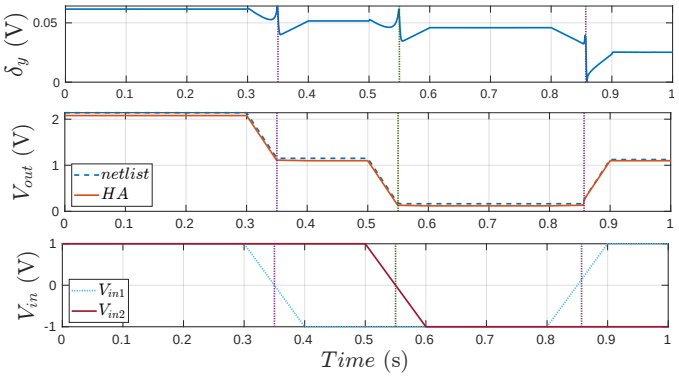


Fig. 8: Comparison of the simulation results from ex.2 (see Fig. 7).

The third example (ex.3) uses the 2-1 configuration from Fig. 7 several times to build-up a NN with 15 neurons in

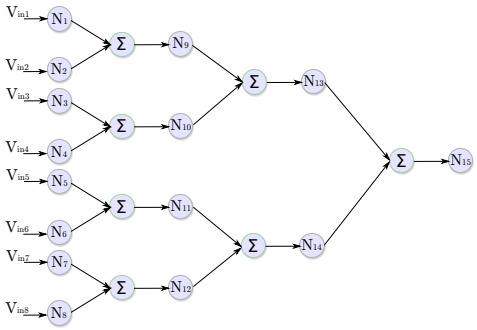


Fig. 9: Simple neuronal network consisting of 15 neurons.

an 8-4-2-1 configuration as shown in Fig. 9. The circuit is simulated for 2 s with a timestep of 0.1 ms. The eight inputs of the system switch 0.1 s apart, as illustrated in the last row of Fig. 10. As shown in the second row of Fig. 10, the output voltage V_{out} of the 15 combined HAs is similar to the voltage of the Spice netlist with 15 instances of the same neuron. The detailed result is presented in Table I.

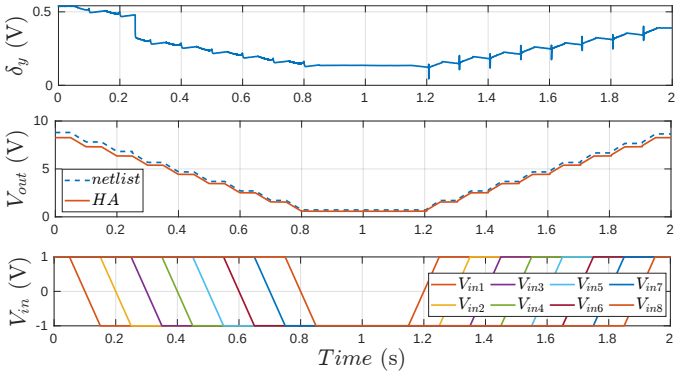


Fig. 10: Comparison of the simulation results from ex.3 (see Fig. 9).

A forth example (ex.4), using a 16-8-4-2-1 configuration, is presented in Table I. This example was simulated with 16 input firing 50 ms apart in a similar fashion to the previous example for a simulation time of 2 s with a timestep of 0.1 ms. Summing up this section, as Table I presents, by using the

TABLE I: Comparison of the Spice netlists and the generated HAs

Model		complexity	Time steps	Runtime (s)	Speedup	$\hat{\delta}_{y,r}$ (%)
ex.1	Netlist	18 tr	10032	3.55	-	-
	HA	-	10034	0.17	20.88	10.56
ex.2	Netlist	54 tr	10045	6.54	-	-
	HA	-	10053	0.25	26.16	3.04
ex.3	Netlist	270 tr	20134	75.16	-	-
	HA	-	20214	2.38	31.57	6.32
ex.4	Netlist	558 tr	20246	253.84	-	-
	HA	-	20404	4.63	54.82	5.9

tr stands for transistors and $\hat{\delta}_{y,r}$ stands for the maximum value of $\delta_{y,r}$, the relative output error normed over the output range

HAs significant speedups are obtained while maintaining an adequate output deviation compared to the Spice netlists.

VII. Conclusion

In this paper, the automatic model abstraction of a neuron with BSIM4 accuracy was examined. As the results show, the obtained model exhibits a significant speedup factor compared to the Spice netlist while maintaining acceptable deviations. Moreover, the speedup scales with the increasing complexity of the circuit, making this approach favorable for the attraction of NNs. Future work will involve controlling the error during the transition of the location and abstracting larger NNs. Additionally, parameter variations of the circuit will be handled with enhanced system descriptions utilizing range arithmetic. As the main target is the formal verification of NNs, future work will as well focus on the hierarchical verification of these circuits and their exhibited behavior.

VIII. Acknowledgment

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

References

[1] G. Gielen, N. Xama, K. Ganesan, and S. Mitra, "Review of methodologies for pre- and post-silicon analog verification in mixed-signal SOCs," in *2019 Design, Automation Test in Europe (DATE)*, 2019.

[2] W. Zheng, Y. Feng, X. Huang, and H. Mantooth, "Ascend: Automatic bottom-up behavioral modeling tool for analog circuits," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium On*, pp. 5186–5189 Vol. 5, May 2005.

[3] C. Borchers, "Symbolic Behavioral Model Generation of Nonlinear Analog Circuits," *IEEE Transactions on Circuits and Systems II: Analog & Digital Signal Processing*, vol. 45, no. 10, pp. 1362–1371, 1998.

[4] J.-Y. Chen, S.-W. Wang, C.-H. Lin, C.-N. Liu, Y.-J. Lin, M.-J. Lee, Y.-L. Luo, and S.-Y. Kao, "Automatic behavioral model generator for mixed-signal circuits based on structure recognition and auto-calibration," in *2015 International SoC Design Conference (ISOCC)*, Nov. 2015.

[5] L.-Y. Song, C. Wang, C.-N. J. Liu, Y.-J. Lin, M.-J. Lee, Y.-L. Lo, and S.-Y. Kao, "Non-regression approach for the behavioral model generator in mixed-signal system verification," in *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct. 2017.

[6] T. Dang, A. Donzé, and O. Maler, "Verification of analog and mixed-signal circuits using hybrid system techniques," in *Formal Methods in Computer-Aided Design*, pp. 21–36, Springer, 2004.

[7] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Computer Aided Verification*, pp. 379–395, Springer, 2011.

[8] M. Fränzle, H. Hungar, C. Schmitt, B. Wirtz, B. Becker, W. Damm, Martin, E.-R. Olderog, A. Podelski, and R. Wilhelm, "HLang: Compositional Representation of Hybrid Systems via Predicates," Reports of SFB/TR 14 AVACS 20, SFB/TR 14 AVACS, July 2007.

[9] M. Althoff, A. Rajhans, B. H. Krogh, S. Yaldiz, X. Li, and L. Pileggi, "Formal verification of phase-locked loops using reachability analysis and continuization," *Communications of the ACM*, vol. 56, no. 10, pp. 97–104, 2013.

[10] A. Tarraf and L. Hedrich, "Behavioral Modeling of Transistor-Level Circuits using Automatic Abstractionto Hybrid Automata," in *Design Automation and Testin Europe (DATE)*, (Florence), 2019.

[11] 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.

[12] S. Steinhorst and L. Hedrich, "Trajectory-directed discrete state space modeling for formal verification of nonlinear analog circuits," in *International Conference on Computer-Aided Design (ICCAD)*, 2012.

[13] J. Phillips, J. Afonso, A. Oliveira, and L. M. Silveira, "Analog macromodeling using kernel methods," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2003.

[14] O. Stursberg and B. H. Krogh, "Efficient representation and computation of reachable sets for hybrid systems," in *Hybrid Systems: Computation and Control* (O. Maler and A. Pnueli, eds.), (Berlin, Heidelberg), pp. 482–497, Springer Berlin Heidelberg, 2003.

Copyright © AJSE Publishers