Compositional Neural-Network Modeling of Complex Analog Circuits

Ramin M. Hasani*, Dieter Haerle**, Christian F. Baumgartner§, Alessio R. Lomuscio§ and Radu Grosu*
*Institute of Computer Engineering, Vienna University of Technology, Austria
(ramin.hasani, radu.grosu) @tuwien.ac.at
**KAI Kompetenzzentrum Automobil- und Industrieelektronik GmbH, Villach, Austria
dieter.haerle @k-ai.at
§Department of Computing, Imperial College London, UK
(c.baumgartner, a.lomuscio) @imperial.ac.uk

***Abstract*—We introduce CompNN, a compositional method for the construction of a neural-network (NN) capturing the dynamic behavior of a complex analog multiple-input multiple-output (MIMO) system. CompNN first learns for each input/output pair (i, j), a small-sized nonlinear auto-regressive neural network with exogenous input (NARX) representing the transfer-function h_{ij}. The training dataset is generated by varying input i of the MIMO, only. Then, for each output j, the transfer functions h_{ij} are combined by a time-delayed neural network (TDNN) layer, f_j. The training dataset for f_j is generated by varying all MIMO inputs. The final output is $f = (f_1, \ldots, f_n)$. The NNs parameters are learned using Levenberg-Marquardt back-propagation algorithm. We apply CompNN to learn an NN abstraction of a CMOS band-gap voltage-reference circuit (BGR). First, we learn the NARX NNs corresponding to trimming, load-jump and line-jump responses of the circuit. Then, we recompose the outputs by training the second layer TDNN structure. We demonstrate the performance of our learned NN in the transient simulation of the BGR by reducing the simulation-time by a factor of 17 compared to the transistor-level simulations. CompNN allows us to map particular parts of the NN to specific behavioral features of the BGR. To the best of our knowledge, CompNN is the first method to learn the NN of an analog integrated circuit (MIMO system) in a compositional fashion.**

I. Introduction

One challenging issue in the pre-silicon verification process of recently produced analog integrated circuits (IC)s is the development of high performance models for carrying out time-efficient simulations. Transistor-level fault simulations of a single analog IC can take up to one or two weeks to be completed. As a result, over the past years, several attempts to develop fast behavioral models of the analog ICs have been investigated. Examples include SystemC, Verilog HDL, Verilog AMS and Verilog-A models which in principle can realize very accurate models [1]–[4]. However, the development of such models is not automated, and the associated human effort is considerable [1]. Moreover, this approach is unlikely to scale up to large libraries of existing analog components. Another example is real number modeling (RNM). In this method, analog parts of a mixed-signal IC are functionally modeled by real values and they are used in top-level system on chip verification [5]. RNMs are fast and cover a large range of circuits. However, for analog circuits including continuous time feedbacks or detailed RC filter effects, it is not recommended [5]. Moreover, RNM is not appropriate to be employed for circuits that are sensitive to nonlinear input-output (I/O) impedance interaction.

In this paper we propose an alternative machine-learning approach for automatically deriving neural network (NN) abstractions of integrated circuits, up to a prescribed tolerance of the behavioral features. NN modeling of the electronic circuits has been recently used in electromagnetic compatibility (EMC) testing, where the authors modeled a band-gap reference circuit (BGR) by utilizing an echo-state neural network [6]. The developed NN model has shown a reasonable time performance in transient simulations; however, since the model is coded in Verilog-A, simulation speed-up is limited. In [7], authors used a novel nonlinear autoregressive neural-network with exogenous input (NARX) for modeling the power-up behavior of a BGR. They demonstrated attractive improvements in the time performance of the transient simulations of the analog circuit within the Cadence AMS simulator by using this NARX model.

In the present study, we employ a compositional approach for learning the Overall time-domain behavior of a complex multiple-input multiple-output (MIMO) system, CompNN. CompNN learns in a first step, for each input i and each output j a small-sized nonlinear auto-regressive NNs with exhogeneous inputs (NARX) representing the transfer-function f_{iJ} from i to j. The learning data-set for h_{ij} is generated by varying only input i of the MIMO system and keeping all the other inputs constant. In a second step, for each output j, the transfer functions h_{ij} learned in Step 1, one for each input i, are combined by a (possibly nonlinear) function f_j, which is learned by employing another NN layer. The training dataset in this case is generated by applying all the inputs at the same time to the MIMO system. Once we constructed f_j for each output j, the overall output function is obtained as $f = (f_1, \ldots, f_n)$. We evaluate our approach by modeling the main time-domain behavioral features of a CMOS band-gap voltage reference circuit. We initially extract such features from the BGR circuit by using our I/O decomposition method. Consequently, we define trimming, load jump and line jump as the main behavioral features of the circuit to be modeled. Individual small-sized NARX networks are designed and trained in order to model the BGR output responses. We recompose the

trained models by stacking a second layer network in a time-delayed neural network (TDNN) structure. The second layer is then trained in order to reproduce the output of the BGR. Such implementation provides us with an observable model where one can define a one-to-one mapping from specific behavioral features of the system to certain parts of the model. Finally, we employ our neural network model in a transient simulation of the BGR and evaluate its performance. This is done by utilizing a co-simulation approach between MATLAB and Cadence AMS Designer environment [8]. We demonstrate that by using such 2-layer neural network structure, we can achieve one order of magnitude speed-up in the transient simulations.

The rest of the paper is organized as follows. In section II, we introduce our compositional approach for developing neural network models and define the case study. In Section III, we describe the NARX neural network architecture and identify the optimal quantity of components to be used in the neural network for each behavioral response. In Section IV, we explain the training process performed on the network and explore the performance of the designed models. Subsequently, in Section V, we train the second layer with the aim of merging the behavioral models into a single block. Finally, in Section VI, we employ a co-simulation approach for simulating our MATLAB/Simulink neural network model into Cadence Design environment and illustrate the performance of the network.

II. CompNN for MIMO system modeling

Let $I = \{i_1, i_2, ..., i_n\}$ be the vector of the defined inputs to a MIMO system, and $H = \{h_1, h_2, ..., h_m\}$ be the vector of nonlinear transfer functions delivering the corresponding output for each input exclusively, each output of the system is then constructed as $O = f(O_1, O_2, ..., O_m)$ where f, depending on the device under test (DUT), can be a linear or nonlinear function. As a result, we train small-sized neural networks for modeling each component of vector H and subsequently we estimate the function f for merging such components by a second layer NN. We call such compositional approach CompNN. CompNN provides us with the ability of mapping particular parts of the neural network model to specific behavioral features of the DUT and therefore having an observable model.

We demonstrate the performance of our method by developing a NN behavioral model of an analog integrated circuit: CMOS band-gap voltage reference circuit (BGR). A BGR outputs constant voltages (in our case 1V and 0.49V) regardless of possible variations caused by temperature change, power supply and load properties. Figure 1A depicts a symbolic representation of our BGR. The circuit is constructed from 50 transistors. We define the inputs to the system to be the power supply (V_{DD}) and three digital trimming inputs. A load-profile can be applied to the output-pin of the circuit (1V-Out). We therefore consider the load-profile as an input signal as well. Thus, the circuit realizes a multi-input single-output (MISO) dynamic system which is a particular case of a MIMO system.

Figure 1B shows the BGR behavioral representation where the circuit comprises several behavioral features such as:

Power-up – which is the activation of the power supply with several slopes and voltage levels.

Trimming inputs – which enables the circuit to generate 8 different stable outputs between 0.9 and 1.1 on its 1V-output pin. There are three digital trimming inputs.

Load jump – demonstrates the variations occurred on the output voltage when a current load is applied.

Line jump – models the response of the BGR when there is a line jump on the power supply of the circuit.

Features are consequently recomposed by the function f and create the output of the circuit.

In [7], authors employed a NARX NN for modeling the power-up behavior of the BGR. In this paper we model the rest of the decomposed features and thus complete the behavioral modeling of the circuit. We merge the behavioral features by approximating the function f using a second layer, time-delayed neural network.

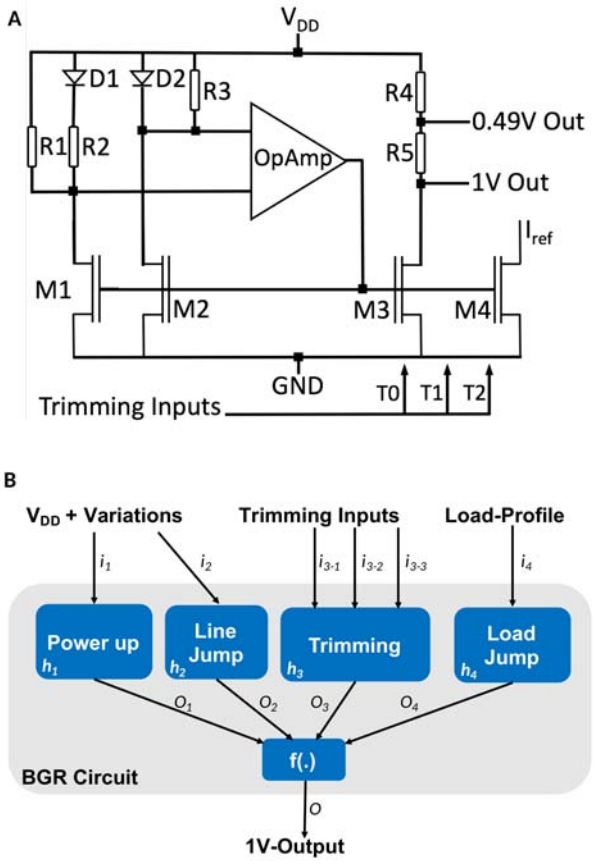

Fig. 1. CMOS band-gap voltage reference circuit (BGR) A) Symbolic representation of the circuit schematic. B) Behavioral representation of the circuit.

III. NARX Neural-network architecture

Although the transfer function of a BGR is in principle constant, this is in practice highly nonlinear. As a conse-

quence, modeling of the time-domain features requires powerful nonlinear system identification techniques and solutions. A nonlinear auto-regressive neural network with exogenous input (NARX NN) appears to be a suitable framework for deriving approximations, up to a prescribed, maximum error, of the BGR. It has been previously demonstrated that a recurrent nature of the NARX NN topology consisting of only seven neurons and three three-time input-and-output delay components is able to precisely reproduce the turn-on behavior of the circuit [7].

In this paper, we use the NARX architecture for modeling in addition the trimming, load jump and line jump behaviors of the BGR. The output of the network is constructed from the time-delayed components of the input signal $X(t)$ and output signal $Y(t)$, (see for example [9]):

$$Y(t) = f(X(t-1), X(t-2), ..., X(t-n_x), \\ Y(t-1), Y(t-2), ..., Y(t-n_y)). \quad (1)$$

The n_x and the n_y factors, define the input and output delays, that is, the number of discrete time steps within the input and the output histories that the component has to remember, in order to properly predict the next value of the output [10]. $n = n_x + n_y$ is the number of input nodes.

The size of the hidden layer is highly dependent on the number of the input nodes. There are several ad-hoc approaches for defining the appropriate number of hidden neurons. For instance one of the popular methods prescribes that the number of neurons within the hidden layer should be between the number of input nodes [11] and output nodes. We perform a grid search for choosing the optimal number of the delay components and hidden layer neurons [12]. A hyperparameter space (d, h), consists of two parameters representing the quantity of delay components d, and the number of hidden-layer neurons h. Parameter d is chosen from a set $D = \{1, 2, ..., 7\}$ and h from the set $H = \{1, 2, ..., 15\}$. The Levenberg-Marquardt back propagation is performing a parameter optimization where the error of the validation dataset for each architecture pair (d, h), is calculated in the course of the training process. We ultimately select the architecture pair which results in the least validation error. Table I depicts the optimal number of delay components and hidden neurons chosen for realization of individual BGR features. As the

TABLE I
NARX NETWORK ARCHITECTURE FOR EACH OF THE BGR BEHAVIORAL FEATURES

Features	♯ of delay components	♯ of hidden neurons
Trimming	3	10
Load Jump	3	7
Line Jump	3	7

output layer is a regressor, it comprises only one node.

The NARX architecture therefore, is designed for each behavioral task as shown in Figure 2. In this architecture, weighted input components synapse into the hidden layer nodes with an all-to-all connection topology. We evaluated

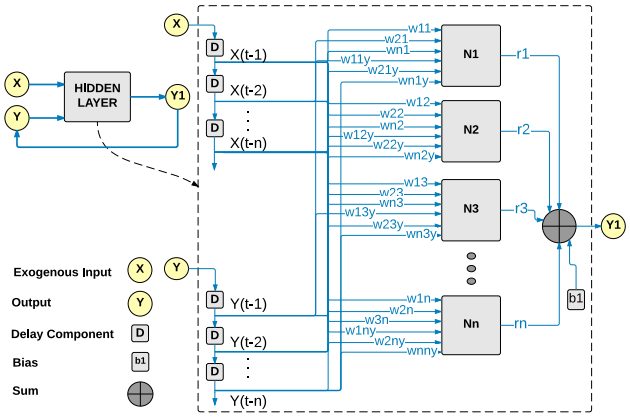

Fig. 2. NARX neural network architecture. Note that the network realizes a recurrent topology where the output is fed-back into the input layer and causes further refinements on the predicted output signal $Y1$.

TABLE II
TRANSIENT SIMULATIONS PERFORMED FOR THE TRAINING DATA COLLECTION PURPOSES

Simulation	Simulation Time	CPU time	Input	Output	♯ of samples
Trimming	100 μs	1.4 s	Trimming inputs	$Vout_{1V}$	695
Load Jump	540 μs	1.3 s	Load Profile	$Vout_{1V}$	433
Line Jump	200 μs	1 s	V_{DD}	$Vout_{1V}$	501

different activation functions such as (Elliot, logistic sigmoid and tanh) and achieved the best performance by using a hyperbolic-tangent activation-function:

$$H = tanh(\sum_{i,j=1}^{N} (w_{ij} X_i) + b_j), \quad (2)$$

where H is the output of the hidden layer, w_{ij} represents the synaptic weight of the input X_i, from input node i to hidden node j and b_j depicts the bias weight applied to the hidden neuron j. The output of the NARX network is constructed as a linear sum of the weighted Hidden layer outputs. The network is designed in MATLAB [13].

IV. TRAINING PROCESS AND NETWORK PERFORMANCE

In order to collect adequate training datasets for teaching the NARX networks for the three behavioral features of the BGR that is, trimming, load jump and line jump, we perform three transient simulations on the BGR by using the AMS simulator within the Cadence environment. Table II shows the details of the performed simulations and the collected datasets.

We aim to train a specific NARX network for each of the behavioral features where we use the input data as the exogenous input to the neural network and the output data as the target values to be learned. In order to gain high precision in the training process, we use the network in a feedforward topology in which the input of this topology consists of the original inputs and outputs, plus all the delayed inputs and outputs, up to their maximum input and output delays, respectively [7]. A Levenberg-Marquardt (LM) back-propagation algorithm is employed for training each network [14]. The LM learning

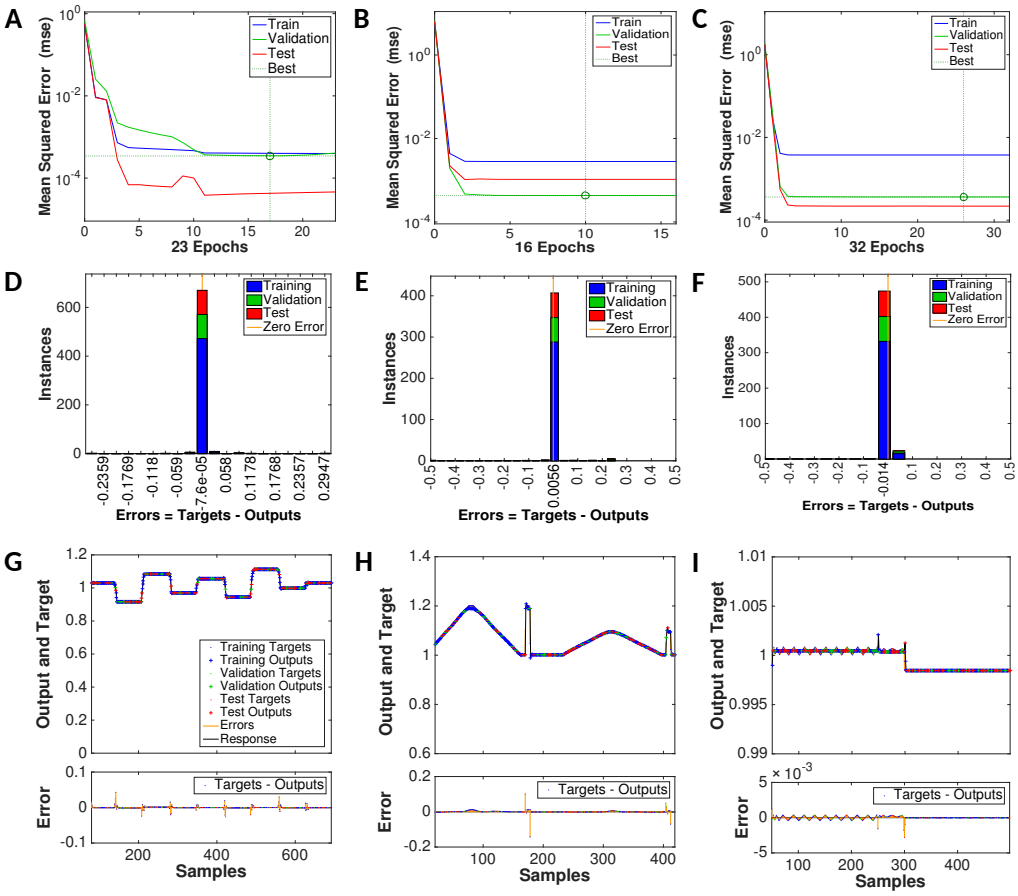

Fig. 3. Network performance of the trimming, load jump and line jump NARX behavioral models. A, B and C display the performance of the NARX neural network model of trimming, load jump and line jump, respectively, throughout the training process. The MSE is reduced drastically by each training step. In all three cases, the process terminated as soon as the validation dataset error stopped descending after 6 consequent epochs. D, E and F shows the error histogram of training samples for the NARX model of trimming, load jump and line jump behavior, respectively. Note that most of the instances' error are close to the zero error line for each case. G, H and I represent the output of the band-gap circuit together with its neural network response for trimming, load jump and line jump behaviors, respectively. They also show the generated output error per sample.

method, which is a modified version of the Gauss-Newton training algorithm, results in fast convergence of the gradient to its minimum since it is unaccompanied by calculation of Hessian matrix. We initially define a cost function as follows:

$$E(w,b) = \frac{1}{2}\sum_{k\in K}(f(w,b)_k - t_k)^2, \tag{3}$$

where $E(w,b)$ stands for the error rate as a function of the weight w, and bias values b, $f(w,b)_k$ is the output generated by the neural network and t_k is the target outputs. We then try to minimize the error function for each training iteration with respect to the synaptic weights. $\triangle w$ which is calculated by the LM method and it is given by:

$$\triangle w = [J^T(w)J(w) + \eta I]^{-1} J^T(w)(f(w) - t), \tag{4}$$

Accordingly, the updated value of the weights is computed as:

$$w_{new} = w + \triangle w. \tag{5}$$

where $J(w)$ is the Jacobian matrix comprising the first-order derivatives of the error function with respect to weight values. Parameter η is the key to the fast convergence [15]. When this parameter is zero, the LM method realizes the common Gauss-Newton algorithm. If η increases throughout the training process, it is multiplied by an $\eta_{increase}$ value. On the contrary, when a training step results in a decrease of the value of η, its value gets reduced by a $\eta_{decrease}$ value. As a result, the cost function moves in a fast way towards the error reduction within each training epoch. The parameters' initial values and descriptions employed within the LM training algorithm are summarized in the Table III.

For starting the training process, the collected samples are randomly divided into three data subsets consisting of:

- Training set (70%): This dataset is employed during the training process.
- Validation set (15%): This dataset is used for generalization and validation purposes. It also plays a role in the termination of the training process.
- Test set (15%): This dataset provides an additional evaluation test after the training phase. It is not deployed

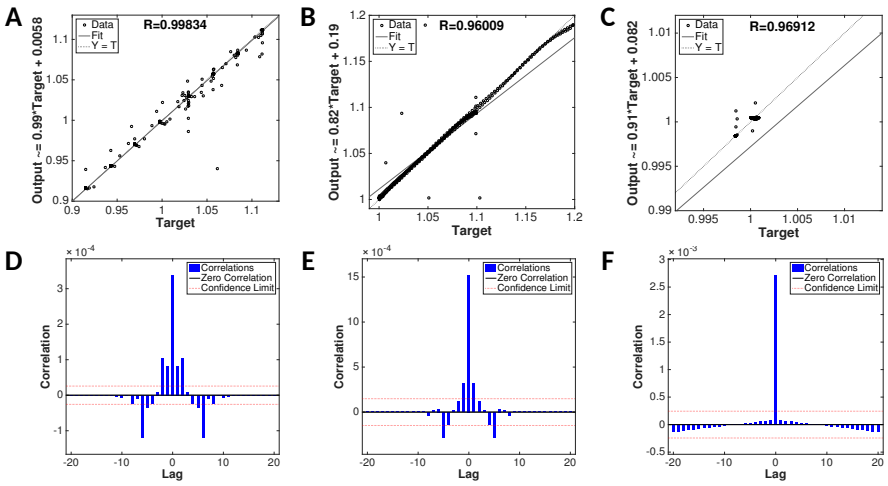

Fig. 4. Linear regression and error auto-correlation function (ACF) representation of the NARX behavioral models. A, B and C show the regression analysis which is performed on the behavioral features, respectively for the trimming, load jump and the line jump. On the left-hand side axes of each regression plot the fitting line function of the NARX output and the selected target values is computed. Note that R stands for the regression coefficient. D, E and F demonstrate the error ACF calculated for our NARX models. blue bars represent the correlation distribution of the lagged errors and the red lines are the 95% confidence bounds (limit lines are located at an error correlation correspond to $\pm 2 \times$ $standard\ error$ (SE)). For an ideal model, the error ACF will be a single bar at the lag zero while for a reliable model most of the lagged error components are located within the confidence boundaries.

TABLE III
LM TRAINING ALGORITHM PARAMETERS

Parameter Name	**Initial Value**	**Description**
max_epochs	1000	Maximum number of training iterations
$ideal\_error\_value$	0	Ideal error rate
$max\_refinement$	6	Maximum validation error descending failure
$min\_cost\_function$	10^{-7}	Cost function minimum Value
eta	0.001	η initial value
$eta\_decrease$	0.1	η decrease factor
$eta\_increase$	10	η increase factor
max_eta	10^{10}	η Maximum η
max_time	inf	Maximum training time

during the learning process.

The training process terminates as soon as one of the conditions mentioned below occurs:

- No further refinement of the validation-dataset error-function is observed after $max\_refinement$ consequent training epochs.
- The cost function is minimized to $ideal\_error\_value$.
- η goes higher than max_η.
- The maximum number of training iterations is reached.
- The time of the training exceeds its maximum value.
- The Error function drops below $min\_cost\_function$.

Figure 3 illustrates the training performance of the three NARX networks together with their corresponding error histogram. In all cases, the training process is concluded when no further reduction on the validation dataset error is noticed after six sequential training iteration. Moreover, it is observed that within trimming, load jump and line jump, over 95% of the training samples have an average error of 7×10^{-5}, 5.6×10^{-3} and 1.4×10^{-3}, respectively. The time-series responses of the trimming, load jump and line jump models, during the training process are plotted in the Figure 3 G, H and I, correspondingly. Note that the NARX networks precisely follow their target values.

In case of Trimming network, an input consisting of various trimming sets is applied as the training dataset network. The output varies around 1V whenever the trimming values toggle to different configuration. Note that the 1V output of the BGR is modeled in this work. In case of the load-jump network, two different current load profiles are separately applied to the 1V and 0.49V output of the BGR. Since the 0.49V output of the BGR is constructed from a resistor devision on the 1V output pin, we expect to observe the voltage change caused by the load applied to the 0.49V output on the 1V pin. Therefore, as input to the load jump network we take both load profiles into account. Finally, for the line jump, small variations on the power supply of the circuit are considered. In the ideal situation, we expect to see no change on the output. However, the output slightly varies as it is shown in Figure 3I. The figure shows small fluctuations around 1V in the order of 10^{-3} due to a power supply variation of 10%. We notice that the line jump network imitates the behavior of the target values with a decent accuracy.

Furthermore, a linear regression is performed at the output

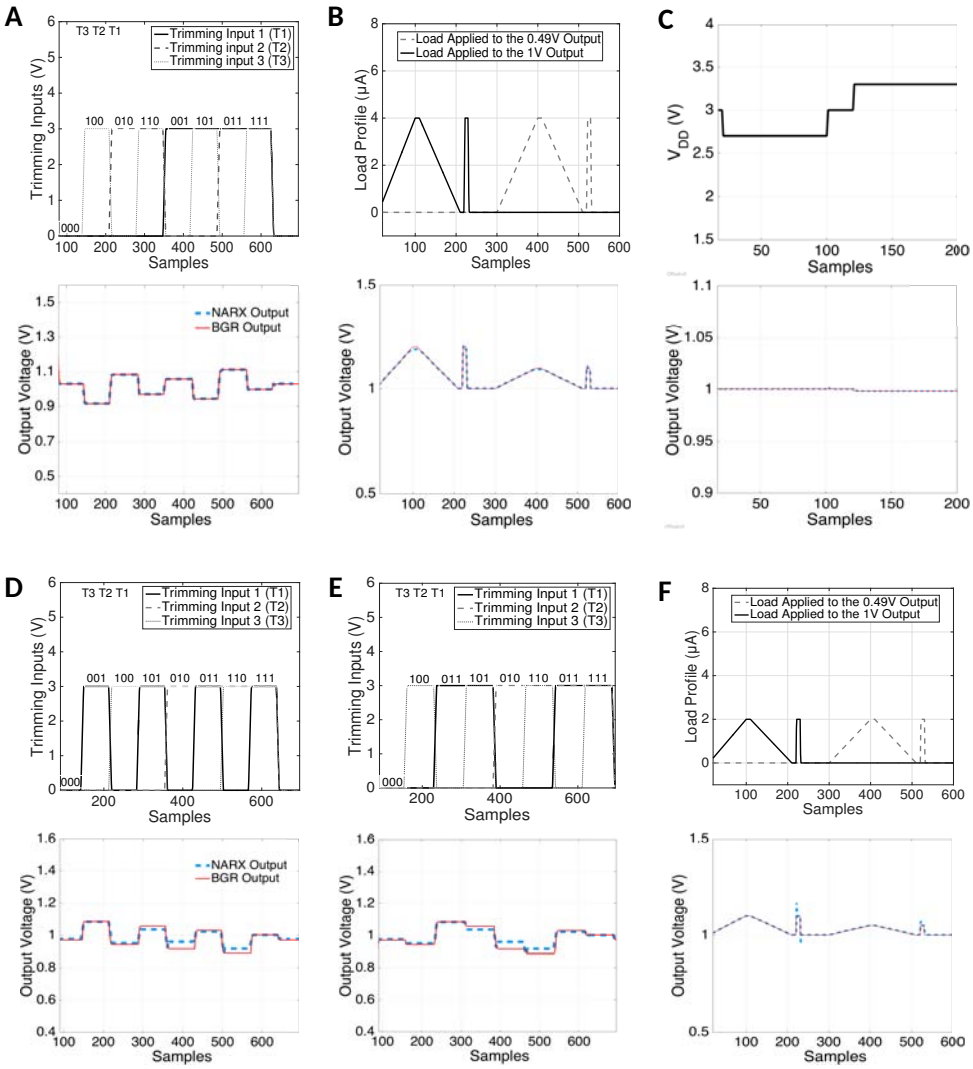

Fig. 5. Time-response of the trained neural networks. A, B and C represent the input and output response of the NARX networks resembling trimming, load jump and line jump, after the training process where a simulink block of the network is generated. Training input data is applied to the network and its corresponding output is recorded. In B, we applied two load profiles, one to the 0.49V output and the other one to the 1V output. Since the 0.49V output of the BGR is created by using a resistor devision on the 1V output pin, at the output of the 1V pin we see the effect of the load connected to the 0.49V, as well. D and E depict two different input sets that are applied to the trained trimming neural network, in order to check the behavior of the NARX network in case of input patterns unalike the training input pattern. The same is checked for the load jump network in F. Note that the network generated a reasonable response in case of dissimilar input patterns in both cases.

layer of the neural network. The regression performance of the NARX network for each individual behavioral feature is shown in the Figures 4A-C. the regression coefficients R, are calculated to be close enough to $R = 1$ which is the case of ideal model. Moreover, the fitting-line function between the output of the NARX and target values are computed for each network.

In order to assess the efficiency of the network and the training process, we calculate the error auto-correlation function (ACF) in each case. The ACF explains how the output errors are correlated in time [16]. Let the output error time-series, $e(t)$, be the difference between the generated output of the NARX network, $Y(t)$, and the target values, $T(t)$, $e(t) = Y(t) - T(t)$. The error correlation rate for the lag i, $\hat{\rho}_i$, is computed as follows:

$$\hat{\rho}_i = \frac{\sum_{t=i+1}^{T}(e_t - \hat{e})(e_{t-i} - \hat{e})}{\sum_{t=1}^{T}(e_t - \hat{e})^2}, \tag{6}$$

where T is the number of lags in time, which in our case is set to 20 and $\hat{e}$ stands for the average of the output error time-series. Ideally, the AFC comprises a single bar at the lag zero and the correlation rates of the other lagged-error components are zero. For a reliable model we set a 95% confidence limit equal to $\pm 2SE_\rho$, where SE is the standard error for checking the importance of the i^{th} lag for the autocorrelation, $\hat{\rho}_i$, and

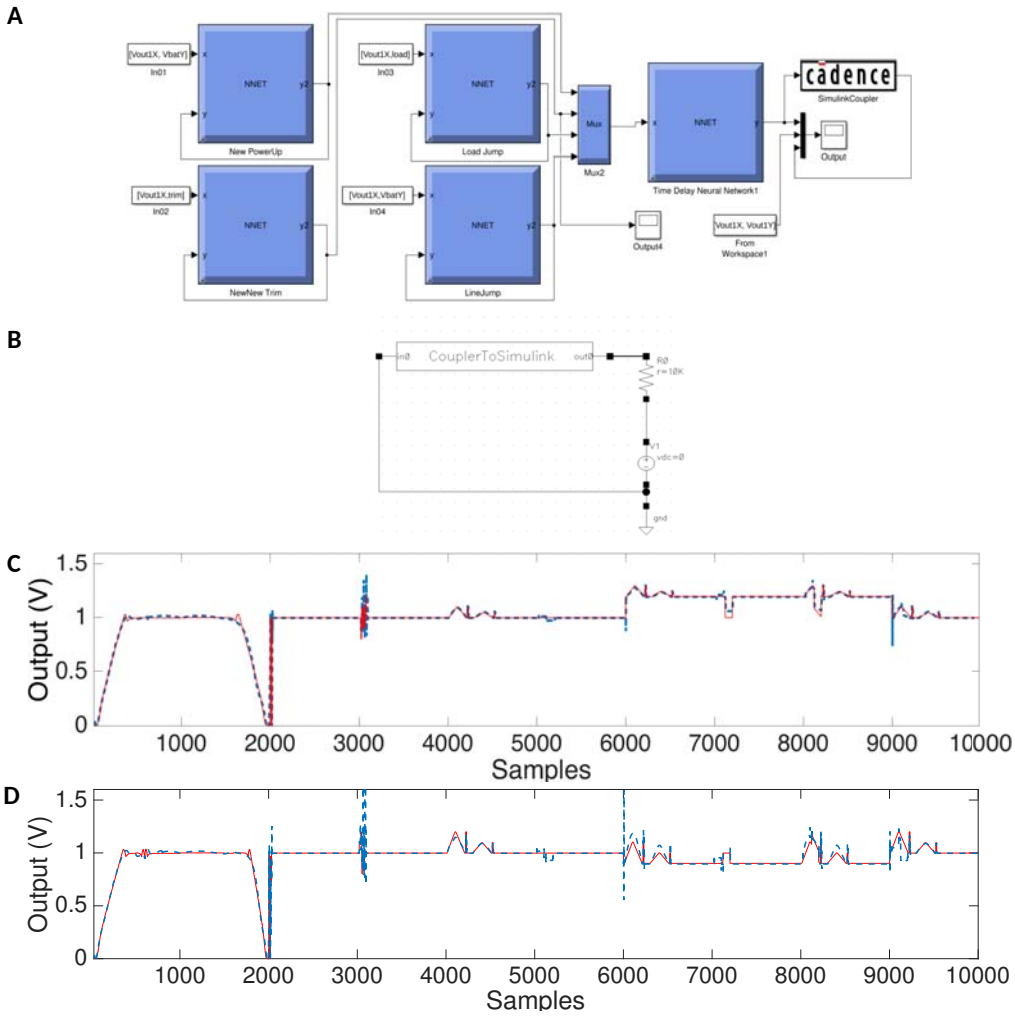

Fig. 6. Two-layer neural network structure. A) Four NARX behavioral models are fed into the second layer network. B) Cadence schematic environment prepared to perform the co-simulation of Simulink model in Cadence AMS Desinger C) Response of the the BGR (solid red line) and its model (dashed blue line) to the training data. D) Response of the circuit and the model to the test pattern.

it is roughly calculated as follows:

$$SE_\rho = \sqrt{\frac{(1 + 2\sum_{j=1}^{i-j} \hat{\rho}_i{}^2)}{T}}. \quad (7)$$

Figures 4D-F show the error ACF plots for our trimming, load jump and line jump networks, respectively. The horizontal red lines are the 95% confidence bounds. Note that in all cases most of the error autocorrelation samples are within the confidence limits. This underlines the accuracy of the model.

Furthermore, in order to observe the behavior of the trained NARX models after the training process, we perform validation simulations by applying training datasets and datasets different from the training sets to the network. Figures 5A-F show the applied input profiles together with the time response of the networks, trimming, load jump and the line jump. We observe that the neural network's output reasonably follows its target values in all cases. Based on the specification of our BGR, the acceptable error-rate at the 1V-output is 5%. Our neural network models generates a response in case of different input datasets (Figures 5D-F), which satisfy such condition.

Note that once the training process is terminated, the simulation time of the trained neural network is very fast. The CPU time recorded by MATLAB to perform our validation simulations is on average in the range of some milliseconds. Our learned models show improvements in the time performance by a factor of 17 when compared to their analog counterparts, during transient simulations. We experimentally verify such results in the following.

V. Recomposition Function: A Time-Delayed Neural Network Layer

In this section we select a recomposition function f, as described in Section II, for combining behavioral models of the BGR including the power-up behavior. By using the LM back-propagation algorithm we train a time-delayed neural network (TDNN) comprised of three input delay elements and 200

hidden-layer neurons, to be able to take the generated output of the four pre-trained NARX models and to predict the correct 1V-output pin of the BGR. The structure is selected with the same approach as that of NARX models. Figure 6A represents the structure of the two-layer network. The network response to the training and test dataset is shown in Figure 6B and 6C, respectively. Matlab CPU time for executing the simulation of the network is approximately $50ms$.

VI. Co-simulation of MATLAB/SIMULINK models and analog design environment

Here we utilize the Cadence AMS Designer/MATLAB co-simulation interface in order to evaluate the performance of the designed neural network model within the Analog Design Environment (ADE) of Cadence software, where we execute analog IC's fault simulations [8]. Inside the co-simulation platform, a coupling module is provided in order to link Simulink and Cadence schematics environments. Figure 6A and 6B show the simulation environments in Simulink and Cadence schematics respectively. We apply inputs to the neural network block in Simulink and simultaneously run a transient simulation in the Cadence ADE. Figure 6C and 6D depict the results of the co-simulation in case of training input dataset and test input dataset, correspondingly. The total CPU time for such transient simulations is calculated as 1.07s while the same simulation of the transistor-level BGR takes 17.8s to be completed. As a results, we gain a simulation speed-up by a factor of 17.

VII. Conclusions

We employed a new neural network modeling approach for complex MIMO systems (CompNN). We modeled individual I/O behavioral functions of the system by training NARX neural networks. We then merged the overall behavioral features by training a second layer TDNN. CompNN enabled us to define a one-to-one mapping from specific behavioral features of the system to certain parts of the model. We illustrated the performance of our modeling approach by designing behavioral NN models for a CMOS band-gap voltage reference circuit. Individual, small-sized NARX networks were designed and trained to imitate the trimming, load jump and line jump responses of the BGR. Such pre-trained networks together with the power-up behavior, were fed into a second time-delayed network in order to generate a single block representing the BGR.

The performance of the instructed networks were qualitatively and quantitatively analyzed by carrying out linear regression analysis, computing the error auto-correlation function and calculating the error histogram for each model. We confirmed the level of generalization and the accuracy of such predictive neural networks by illustrating the output response of the models to various input patterns different from the training patterns. We subsequently created a single neural network block by adding the second layer for merging the behavioral features and training the network. Finally we employed the designed network in a transient simulation and achieved sensible enhancement in the time performance of the simulation.

For future work, we intend to exploit our NARX models in the verification of analog integrated circuits, where the instantaneous response of the network together with its high level of accuracy results in significant improvements in the performance of the pre-silicon analog fault simulations.

Acknowledgments

We would like to thank Infineon for training, mentoring and provision of the tool landscape. This work was jointly funded by the Austrian Research Promotion Agency (FFG, Project No. 854247) and the Carinthian Economic Promotion Fund (KWF, contract KWF-1521/28101/40388). Part of this research work was carried out while the first author was visiting Imperial College London in 2016.

References

[1] R. Narayanan, N. Abbasi, M. Zaki, G. Al Sammane, and S. Tahar, "On the simulation performance of contemporary ams hardware description languages," in *2008 International Conference on Microelectronics*. IEEE, 2008, pp. 361–364.

[2] M. Shokrolah-Shirazi and S. G. Miremadi, "Fpga-based fault injection into synthesizable verilog hdl models," in *Secure System Integration and Reliability Improvement, 2008. SSIRI'08. Second International Conference on*. IEEE, 2008, pp. 143–149.

[3] F. Pêcheux, C. Lallement, and A. Vachoux, "Vhdl-ams and verilog-ams as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE transactions on Computer-Aided design of integrated Circuits and Systems*, vol. 24, no. 2, pp. 204–225, 2005.

[4] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816–2823, 2006.

[5] S. Balasubramanian and P. Hardee, "Solutions for mixed-signal soc verification using real number models," *Cadence Design Systems*, 2013.

[6] M. Magerl, C. Stockreiter, O. Eisenberger, R. Minixhofer, and A. Baric, "Building interchangeable black-box models of integrated circuits for emc simulations," in *Electromagnetic Compatibility of Integrated Circuits (EMC Compo), 2015 10th International Workshop on the*. IEEE, 2015, pp. 258–263.

[7] R. M. Hasani, D. Haerle, and R. Grosu, "Efficient modeling of complex analog integrated circuits using neural networks," in *2016 12th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*. IEEE, 2016, pp. 1–4.

[8] Cadence. Cadence Virtuoso AMS Designer Simulator, cosimulation of mixed-signal systems with matlab and simulink. [Online]. Available: http://www.mathworks.com/products/

[9] H. T. Siegelmann, B. G. Horne, and C. L. Giles, "Computational capabilities of recurrent narx neural networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 27, no. 2, pp. 208–215, 1997.

[10] S. A. Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

[11] J. Heaton, *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.

[12] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[13] H. Demuth, M. Beale, and M. Hagan, "Neural network toolbox 8.4," *Users guide*, 2015.

[14] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[15] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[16] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.