Compiled Code Simulation of Analog and Mixed-Signal Systems Using Piecewise Linear Modeling of Nonlinear Parameters

Hui Zhang, Simona Doboli†, Hua Tang and Alex Doboli
Department of Electrical and Computer Engineering
State University of New York at Stony Brook, NY, 11794-2350
Email: {huizhang, adoboli}@ece.sunysb.edu
†Department of Computer Science
Hofstra University, Hempstead, NY, 11549
Email: Simona.Doboli@hofstra.edu

April 29, 2005

Abstract

This paper presents a method for fast time-domain simulation of analog systems with nonlinear parameters. Specifically, the paper focuses on $\Delta\Sigma$ analog-to-digital converters (ADC). The method creates compiled-code simulators based on symbolic analysis. Code is optimized using loop invariant elimination and constant folding. Circuits are described as structural macromodels. Non-linear parameters are expressed using piecewise linear (PWL) models. The paper presents a technique for automatically creating PWL models through model extraction from trained neural networks. As compared to existing behavioral simulation methods for $\Delta\Sigma$ ADC, this technique is more automated and accurate. In our experiments, compiled-code simulation was significantly faster than numerical simulation.

1 Introduction

One bottleneck of systems on chip (SOC) development is presently the design of RF and analog IP cores, as well as the integration and verification of final designs [18]. Existing research offers remarkable solutions to synthesis of analog circuits, like opamps, operational transconductors (OTA), comparators and so on. Please refer to [18] for a recent overview. Commercial tools, including Virtuoso NeoCircuit and Virtuoso NeoCell from Cadence and Circuit Explorer from Synopsys, are available for transistor sizing and layout design of analog circuits. Obviously, the next step is to tackle synthesis of more complex analog and mixed-signal systems, like analog to digital and digital to analog converters, phase-locked loop circuits and transceivers. For this endeavor, however, it has been reported that one of the main challenges is due to the large number of optimization variables that must be simultaneously tackled [9, 10, 23, 29, 47]. Existing analog and mixed-signal circuit simulators [48], which are the core of exploration-based synthesis, are still too slow for being used inside the SOC synthesis loop [15, 18, 33], experience numerous stability problems [15], and are unable to exploit the specifics of circuits and systems [50, 51]. Slow system simulation poses additional challenges to exploration-based analog synthesis, which needs a very long time to complete its search or might even not converge towards a constraint satisfying solution.

Behavioral models are used for speeding-up simulation and analysis inside the analog and mixed-signal system synthesis loop. For example, in the methodologies proposed by Dhanwada *et al* [9] and Vancoreland *et al* [47], behavioral models are used for most of the time to quickly find the performance of the analyzed designs. Periodically, detailed circuit simulation is performed to correct the inaccuracies introduced by the behavioral models. Behavioral circuit and system models are of two kinds: structural (physical) and mathematical models. Please refer to [1] for a presentation of the most recent advancements in behavioral

*Parts of this paper were published in the Proceedings of the 2003 International Joint Conference on Neural Networks (IJCNN) and 2004 Behavioral Modeling and Simulation Workshop (BMAS).

modeling and simulation. Structural modeling methods, in general, simplify a circuit to a reduced sub-circuit that includes only the dominant devices. Recently, several approaches are reported to automatize this process using the signal-path approach [26], root localization method [22] and behavioral model decoupling [49]. In spite of these promising results, structural models are obtained most of the time through manual analysis based on the designer's expertise and experience. Mathematical models capture quantitative relationships between the parameters and performances of a circuit. Mathematical modeling includes linear and non-linear regression, Volterra series, Pade approximations, wavelet functions, and neural networks [1]. Non-linear regression is traditionally used to produce mathematical models [7, 8, 21]. The most important limitations of existing structural and mathematical modeling techniques include difficulties in handling nonlinear parameters as well as in tackling large circuits. Also, large amounts of sampling data are needed for accurate and complete modeling.

In this paper, we propose a novel technique for fast simulation of analog systems with nonlinear parameters. We focus on time-domain simulation even though the method is valid for the frequency domain too. The main idea is to generate optimized code for a simulator that is customized to each individual system. The code generation methodology relies on calculating symbolic expressions for the output voltages and currents, and the state variables of a system. To avoid the large memory requirements specific to symbolic analysis, the suggested method exploits the structural regularities present in the topology of a system net-list [10]. Each kind of interconnection structure IST between two blocks b_i and b_j is captured as a separate C++ class $C_{IST}(b_i, b_j)$ with the related methods encapsulating the symbolic composition rule of the two blocks. All instances of structure IST present in the system topology are formulated as objects of class C_{IST}, where the identity of blocks b_i and b_j is set as the two blocks appearing in the structural instance. This representation is applied top-down to the entire system, so that blocks b correspond to building blocks and composed blocks. Code generation utilizes detailed structural macromodels for the building blocks, such as OTA, opamp, comparators and so on, including non-idealities, like finite gain, poles and zeros, CMRR, phase margin, fall and rise time. Nonlinear parameters are described using piecewise linear (PWL) models. Code optimization identifies and eliminates loop invariants, and propagates constant sub-expressions present in the simulation loop. As a case study, experiments present the simulation results for $\Delta\Sigma$ analog to digital converter (ADC) [4]. As compared to numerical simulation, the presented technique achieves significant simulation time reductions and has few stability problems, whereas losses in accuracy are minor.

The paper also discusses a new algorithm for extracting PWL models from trained neural networks (NN). NN are capable to learn any type of nonlinear mapping based on their well-known property of universal approximators [17]. The proposed method addresses the need of automatically creating PWL models for nonlinear parameters [27]. The model generation techniques starts with the training of a NN. A backpropagation algorithm is used for training until the desired accuracy is obtained at the output of the network. Next, a pruning method is applied to eliminate the neurons and weights with insignificant contributions. Then, the sigmoidal activation function of each hidden neuron is approximated with a PWL function with a variable number of segments. The number of segments, its limits and the linear approximation on each segment are automatically determined by a clustering algorithm. Finally, the PWL functions for the hidden neurons are composed together to generate the PWL functions of the model output. The regions - where each linear output model is active, are found by iteratively solving a linear system of inequalities and adjusting its limits.

Compared to other fast simulation methods for $\Delta\Sigma$ ADC [15, 33], our technique requires less designer input and uses more detailed circuit models. Hence, it offers the benefit of more accurate simulation, and thus, potentially, the advent of faster analog design closure. ADC simulation in [15, 33] relies on behavioral models, which are reported to be imprecise [15]. Also, the proposed simulation approach does not require extensive expertise in ADC or analog circuit design. Finally, our simulation technique belongs to the class of compiled-code simulators. To the best of our knowledge, this is the fist attempt to develop a general methodology for compiled-code simulation of continuous-time systems with non-linear parameters. Existing compiled-code simulators [2, 25, 32] are for discrete and event-driven systems. Being customized to a certain application, compiled-code simulators are much faster than their counter-part numerical simulators, and can be optimized to reduce numerical instability.

This paper is organized as six sections. Section 2 details the simulation methodology. Section 3 presents PWL modeling method for nonlinear parameters. Next, we discuss the structure of the C++ code of the customized simulators, and simulation results are given in Section 5. Finally, Section 6 summarizes our conclusions.

2 Proposed Simulation Methodology

Figure 1 shows the proposed compiled-code simulation methodology and also the structure of single-loop $\Delta\Sigma$ ADC [4] - used as illustrating examples in our paper. The methodology produces code for customized simulators generated from the system schematics used as input specifications. The program organization, as calling and called functions (described as C++ methods), reflects

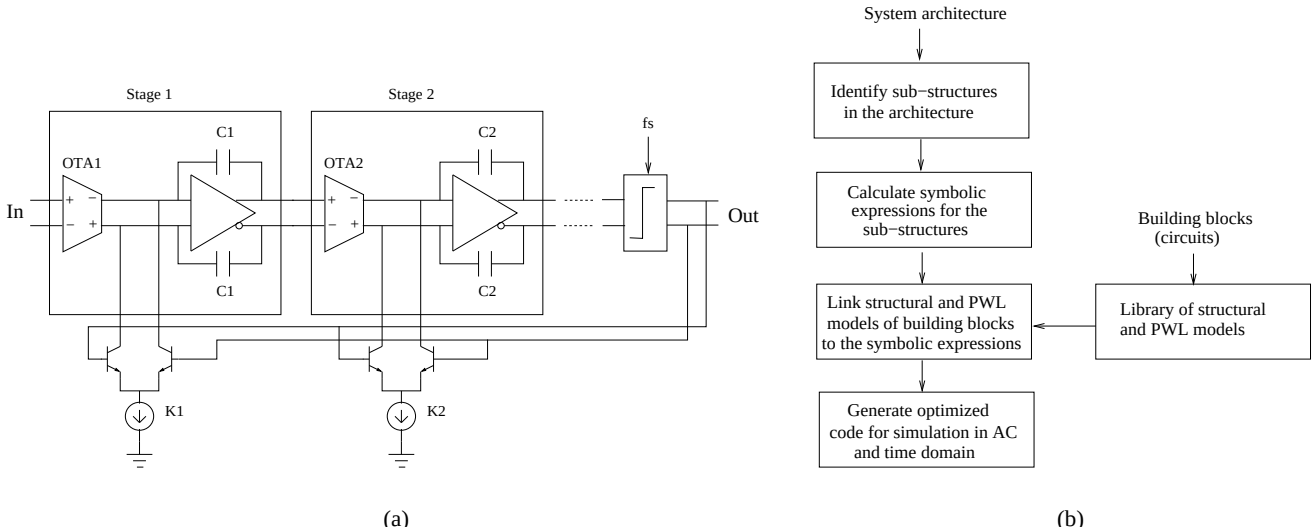

Figure 1: $\Delta\Sigma$ **ADC structure and system simulation method**

the hierarchical structure of a system. Produced methods include variables corresponding to the state variables of the system, as well as instructions for computing the values of output and state variables (i.e. voltages and currents) at each time and frequency instance. Instructions describe symbolic compositions of circuit macromodels depending on the structural patterns that link them together. Additional code is generated for post-processing the output values to compute the required performance metrics. For example for $\Delta\Sigma$ ADC, the output voltage is used to find typical performance figures, like signal-to-noise ratio (SNR) and dynamic range (DR) [4]. Finally, code is also generated to select - during simulation - the correct PWL regions of the non-linear devices.

Figure 1(b) details the methodology. The first step identifies the structural patterns that connect building blocks (circuits). Structural patterns compose two or more blocks and have their behavior described as symbolic relationships between input, output, and state variables. Several similar structural patterns can be identified for complex systems, and Figure 2 shows the structural patterns found in the 3rd order single-loop $\Delta\Sigma$ ADC. The 3-integrator chain macromodel is obtained through composition of the three structurally identical macromodels for the OTA active-C integrator. Then, symbolic expressions are found for each of the identified patterns using a method that symbolically replaces time derivatives of state variables with their differences. Next, the parameters in symbolic expressions are linked to the building block parameters. Finally, optimized code is generated for time-domain simulation, including code for selecting the correct linear segment in the PWL models.

At the system level, each analog block's behavior is described as symbolic relationships between input, output, and state variables. Symbolic equations are formulated by setting-up Kirchhoff's current and voltage laws. After symbolically solving these equations, a set of mathematical expressions result for relating unknown voltages and currents of the blocks to the known signals and circuit parameters. Non-linearities are converted to PWL models using the method presented in Section III. Consequently, following equation describes the behavior of a block:

$$F_V V + F_I I + S = 0 \tag{1}$$

We called matrices $\mathbf{F_V}$ and $\mathbf{F_I}$ *function sub-matrices* and matrix $\mathbf{S}$ *state sub-matrix* respectively.

The entries of the function sub-matrices are determined by the values of the components. The state sub-matrix is related to the state variables and the previous state of the circuit. This accounts for energy storage components, such as capacitors and inductors. For example, the OTA circuit in Figure 4 has a 4-port model (two ports are for across voltage V_{idm} and two ports are for across voltage V_{out}) characterized by following symbolic expression:

$$\begin{pmatrix} I_{ip} \\ I_{in} \\ I_{op} \\ I_{on} \end{pmatrix} = \begin{pmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{pmatrix} \begin{pmatrix} V_{ip} \\ V_{in} \\ V_{op} \\ V_{on} \end{pmatrix} + \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} \tag{2}$$

For defining this expression, we assumed that voltages V_i and V_o are known, and currents I_i and I_o are unknowns. The opposite reasoning would have been also correct.

Symbols f_{ij} and s_i were obtained by symbolically solving the nodal equations of the OTA model, and replacing the derivates of

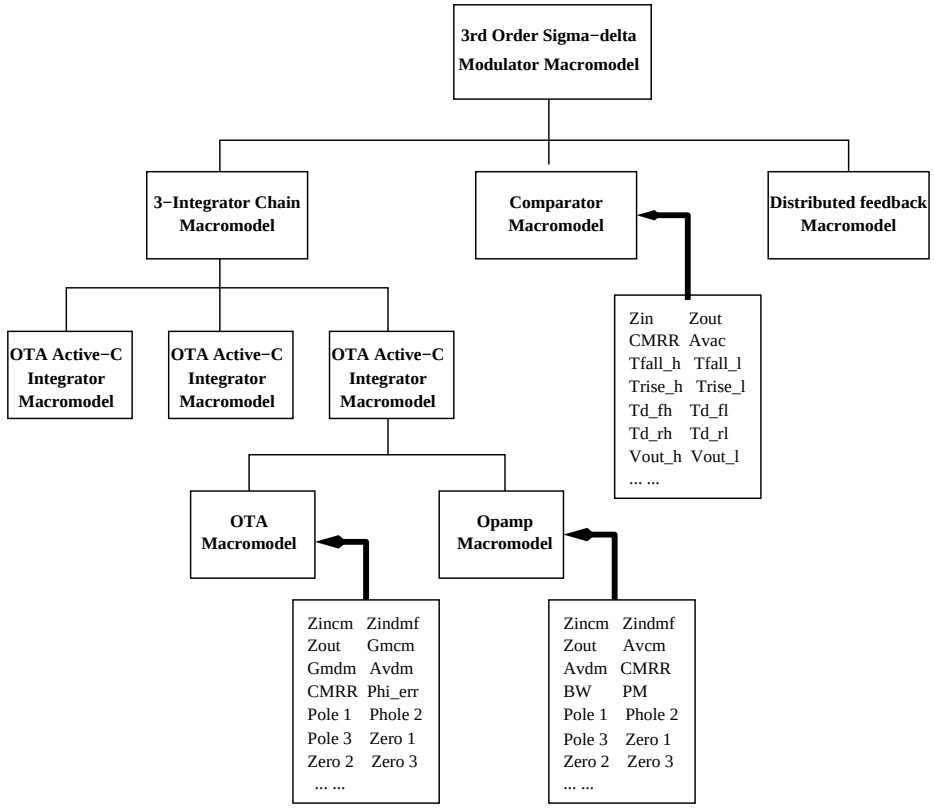

Figure 2: **Structural patterns in a 3rd order single-loop $\Delta\Sigma$ ADC**

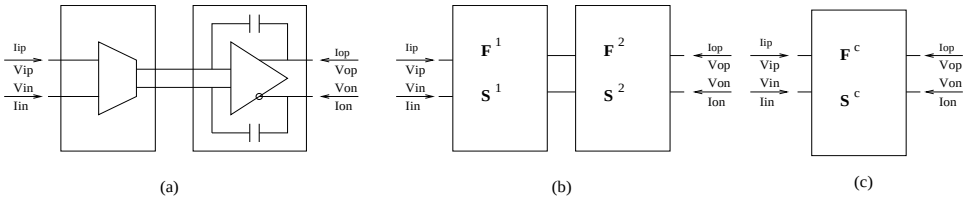

Figure 3: **Structure composition rules for the $\Sigma - \Delta$ stage**

state variables with their differences according to Euler Backward Integration formula [48]:

$$\delta x = \frac{x(t) - x(t-1)}{h}, \tag{3}$$

where $x(t)$ and $x(t-1)$ are the state values at the current and previous time moments, and h is the discretization step. For the OTA circuit in Figure 4, following two symbols were obtained

$$f_{11} = \frac{C_c}{h} + \frac{C_{cd}}{h + C_d\ R_d} \qquad \text{and} \qquad f_{31} = \frac{C_d\,(V_{os} - V_{cd}\,(t-1))}{h + c_d R_d} + \frac{C_{cm}\,(V_{os} - V_{cm1}\,(t-1))}{h}$$

All parameters relate to the structural model. Note that symbol f_{31} depends on the circuit parameters as well as the state values at the previous time moment. Function and state sub-matrices were calculated for all building blocks, and stored in a library.

In some cases, an analog block might behave independently. Then, the outputs of the block are only determined by its inputs and state variables. However, more frequently, analog blocks are influenced by their connected blocks. For example, the load of a transconductor block might change the performance of the block it is connected to. In this case, the equations of that block might not be solvable without knowing all other blocks connected to it, because the number of equations is less than the number of unknown variables. For example, in the OTA example, the values of I_i and I_o can be found only when both V_i and V_o are known.

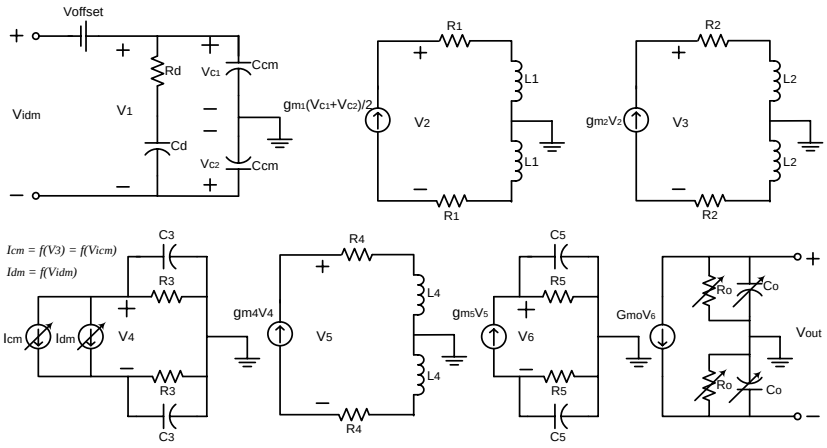

Figure 4: **OTA structural macromodel**

When we simulated the signal flow of the system, we knew only V_i. Another two equations including variable V_o were needed to symbolically solve for the unknowns. These equations were set-up using the blocks connected the output ports of the OTA.

Symbolic composition rules (SCR) are computed for each structural pattern in the system topology. SCR relate the symbolic function and state sub-matrices of a composed block to those of its composing blocks. SCR are calculated using the definition of the block sub-matrices, and constraining that voltages and currents at the connecting links are the same. The symbolic elements of the composed function and state sub-matrices are found after eliminating the currents and voltages at the common links from the equation set.

Figure 3 exemplifies the process for finding the symbolic function and state sub-matrices for a $\Sigma\Delta$ stage. The stage consists of the OTA macrocell linked to the opamp-C macrocell through two links as shown in Figure 3(a). The OTA is modeled by symbolic functional sub-matrix $F^1_{4\mathbf{X}4}$ and by the symbolic state sub-matrix $S^1_{4\mathbf{X}1}$. There are 15 state variables in the OTA structural macromodel. The symbolic functional sub-matrix $F^2_{4\mathbf{X}4}$ and the symbolic state sub-matrix $S^2_{4\mathbf{X}1}$ describe the opamp-C macrocell. The opamp-C macrocell has 17 state variables. Figure 3(b) presents that the composed pattern for the stage includes the symbolic function sub-matrix F^c and state sub-matrix S^c. For example, assuming that V_i and I_f (the DAC currents) are known, and that I_i and V_o are unknown, then

$$\begin{aligned} f^c_{11} \;=\; & [\left(f^2_{22}-f^1_{33}\right)\left(f^2_{11}f^1_{11}-f^1_{14}f^2_{41}+f^1_{11}f^2_{44}\right)+\left(f^2_{21}+f^1_{34}\right)\left(f^2_{12}f^1_{11}-f^1_{13}f^2_{41}+f^1_{11}f^2_{43}\right) \\ & +f^1_{31}\left(f^1_{13}\left(f^2_{11}+f^1_{44}\right)-f^1_{14}\left(f^2_{12}+f^1_{33}\right)\right)]/\left(\left(f^2_{12}+f^1_{43}\right)\left(f^2_{41}+f^1_{34}\right)-\left(f^2_{11}+f^1_{33}\right)\left(f^2_{11}+f^1_{44}\right)\right) \end{aligned}$$

All parameters f^c_{ij} and s^c_{ij} are described by similar expressions.

Once the symbolic function and state sub-matrices were calculated for each basic and composed block, code was generated. The code is a sequence of assignment statements for numerically calculating the elements in the sub-matrices. Code generation carefully identified any redundant sub-expressions. For example, for the OTA model, sub-expression $h + C_d R_d$ was identified as being common to all matrix elements. Hence, the sub-expression was isolated as a new variable and re-used in all instances. This code optimization (similar to constant folding in compiler theory [35]) saved significant amount of computations during the time-domain simulation process.

The time-domain simulation algorithm implements a loop for the entire time range to be simulated. The time increment is h, the parameter also used by Backward Euler Integration formula. At each time instance, the algorithm calculates only a subset of all voltages and currents in an ADC netlist. The subset includes output signals, state variables, and the voltages and currents relevant to the nonlinear devices. For nonlinear devices, the simulation algorithm must also identify the correct PWL region. The identification step first calculates the voltages and currents through the nonlinear devices assuming that PWL regions for the current time instance remain the same as those for the previous time moment. If the assumption is incorrect then the algorithm re-iterates the calculating of the voltages and currents trough the nonlinear devices by assuming the closest PWL regions, and so on. The iteration process stops when the closest feasible PWL regions were found.

Inside the time-domain simulation loop, the elements of the function sub-matrices remain constant. Only the parameters of the state sub-matrices must be updated for each new time step to capture the dynamics of state variables. This observation is very

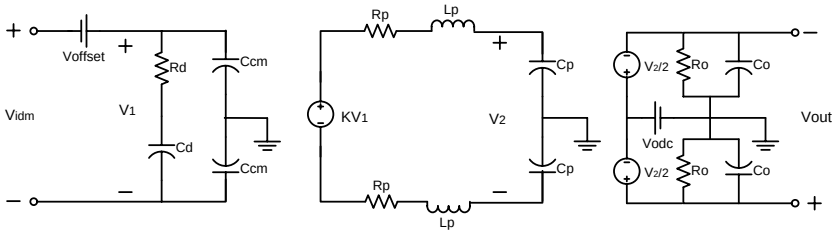

Figure 5: **Opamp structural macromodel**

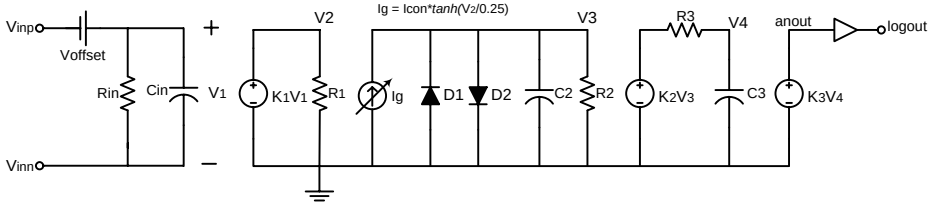

Figure 6: **Comparator structural macromodel**

important to speed-up execution because a large number of computations can be moved outside the simulation loop. This optimization process is similar to removing loop invariants in a compiler [35].

Blocks are either basic blocks (OTA, opamp, and comparator circuits), or composed blocks, which correspond to previous composition steps (like integrators, ADC stages etc). For OTA modeling, we started from the macromodel proposed by Gomez *et al* [20]. We extended the model to fully differential mode (DM) by duplicating the single end stage, the common mode stage, the intermediate and output stages, and the dominant pole stage. Figure 4 shows the obtained model.

Next step related the device parameters in the macromodel to the data collected through SPICE and Spectre simulation during analog circuit synthesis. We used the relationships proposed by Gomez et al [20]:
(1) V_{os} resulted directly through SPICE/Spectre simulation;
(2) $C_{cm} = \frac{1}{(4\pi|Z_{icm}(f_1)|)}$;
(3) $C_d = \frac{1}{(2\pi f_1 \ |Z_{idm}[Im](f_1))} - C_{cm}$;
(4) $R_d = Z_{idm}[Re]\frac{(C_d+C_m)^2}{C_d^2}$;
(5) C_3 depends on the position of the first pole (given by SPICE simulation);
(6) L_4 relates to the dominant zero (offered by SPICE simulation);
(7) $\frac{1}{(R_o C_o)}$ is the frequency of the dominant pole;
(8) R_o results from SPICE/Spectre simulation, directly;
(9) R_1, R_2, L_1 and L_2 are determined by common mode zeros.
Currents I_{dm} and I_{cm} depend nonlinearly with voltages V_{icm} and V_{idm}. Nonlinear dependencies were expressed as PWL models obtained through model extraction from NN. The extraction method was discussed in Section 3.

Figure 5 shows the opamp structural model. The model includes three stages. (1) The input stage of the fully differential opamp model is the same as the input stage of OTA model. (2) The intermediate stage describes the two dominant poles in differential mode. (3) In the output stage, we added a dc bias voltage to the differential output voltages. The bias voltage is needed for transient analysis. Figure 6 presents the comparator model. This structural model was based on the model by Moscovici [34]. The comparator model has the same input stage as the OTA and opamp models. The nonlinear Gm stage expresses the self-limiting behavior of the differential pair by using a hyperbolic tangent function. Gm is bounded to the range $-I_{con}$ and I_{con}. As explained in [34], the two diodes specify a certain time for the slew limited mode of the circuit. The I-V characteristic of the diodes is expressed as PWL functions using the proposed model extraction technique. $R_3\ C_3$ and $R_2\ C_2$ are the two poles of the comparator, and $R_3\ C_3$ is the delay time for large input overdrive voltages. Following relationships were used to relate the macromodel parameters to the data collected using SPICE/Spectre simulation during circuit synthesis:

(1) R_d, C_d, and C_{cm} are obtained using the same formulas as for the OTA and opamp input stages;
(2) product $K_1\ R_1$ was set to 1;
(3) $R_2\ C_2$ corresponds to the 2nd pole;
(4) $R_3\ C_3$ is the 1st pole/ delay time;
(5) V_h, V_l are related to the minimum and maximum output voltages.

3 PWL Modeling of Nonlinear Parameters

Figure 7 presents the proposed circuit modeling method based on PWL models. The method takes the circuit schematic as input. First, the structural macromodel of the circuit is retrieved from a library of manually built models. Section 2 introduced some of the used models. Then, sampling data is collected for creating PWL performance models. Transistors in the circuit schematic are sized with a circuit synthesis tool, and simulation data on the circuit behavior is collected using transistor level simulation (using simulators like SPICE or Spectre). This is a one-time process, as simulation data is stored in a database. Next, post-processing links the simulation data to the parameters in the structural model. Section 2 detailed some of the equations used in post-processing. The last step automatically produces PWL models for the nonlinear parameters.

Feed-forward multilayer neural networks (NN) are a very powerful black-box modeling technique. They can approximate a large class of nonlinear functions based on their well known property of universal approximators [17]. But the theory just states the existence of a NN model. It does not offer any hints on how to find the optimal model given a finite sample of data. Numerous methods have been proposed for training a given NN and/or constructing a NN by pruning or adding neurons. In general, we can assume that a good NN model can be found using any of these techniques. The main disadvantage of NN modeling starts from here: The knowledge embedded in the models - expressed as a set of layers with neurons and weights connecting them - cannot be directly interpreted. Extensive work was done to extract the knowledge in a NN model in a symbolic form (i.e. if-then rules, decision trees, PWL) [5, 16, 44, 43, 40] (see [13] for a recent review). Most techniques were developed for classification problems [43, 5, 16, 46, 38, 39], while few for regression or function approximation problems [40, 3]. Some extraction methods start with a general purpose feedforward trained NN [43, 40, 3, 5], while others extract rules from special type NNs (e.g. boolean inputs [30, 24], product hidden units [37]).

Our method extracts PWL models from trained general purpose feed-forward NNs. Typically, any PWL extraction method from a trained neural network has to accomplish the following two tasks: (a) find a PWL approximation of the nonlinear activation function of each hidden neuron in the NN, and (b) extract the set of PWL models for the input output function. For (b), the common idea is to partition an input region into subregions, such that the output of each subregion is given by a unique linear equation. The main limitation is the exponential scaling of the number of PWL models with the number of hidden neurons. To overcome this complexity, [40] used pruning of the NN. For (a), a number of methods have been proposed. In [40], the activation function of each hidden neuron is decomposed in a fixed number of linear segments, either three or five. The limits of the segments are found using an optimization method to minimize the error between the linear and nonlinear functions. In the method proposed in [11], the activation function of each hidden neuron is split into three linear segments: one with a constant output of zero, one with a non-constant linear dependence and one with a constant output of one. The main advantage of this method consists in a smaller number of linear output models that are generated, at the expense of reduced accuracy.

Neither of the two above methods considers an adaptive modeling of each of the hidden neuron's activation functions. Each hidden neuron covers a different region of the activation function, each with a different degree of non-linearity. Thus, the number of linear segments that should cover each hidden neuron activation function for a similar accuracy varies. A new approach is presented here that extracts a variable number of linear segments for each hidden neuron, depending on each neuron's active region. The number of linear segments for each hidden neuron as well as its limits are automatically detected by a novel clustering algorithm [12].

3.1 Extraction Problem

Given a trained feed-forward NN, the problem is to extract a set of PWL models that approximates well the NN mapping. We limit ourselves to a three layer NN, though the method can be easily extended to more hidden layers. The NN has an input layer $\mathcal{I}$, a hidden layer $\mathcal{H}$ and an output layer $\mathcal{O}$. The outcome of the extraction method is a set of L linear models each of the following form: $\mathcal{L} = \{a^l_1 x_1 + a^l_2 x_2 + \ldots + a^l_N x_N + a^l_{N+1},\ l = 1 \ldots L,\ a^l_{(.)} \in \Re\}$, where N is the number of input variables and x_i is the value of an $\mathcal{I}$ layer neuron. The region in the input space where the lth model is valid is defined by a set of linear constraints of

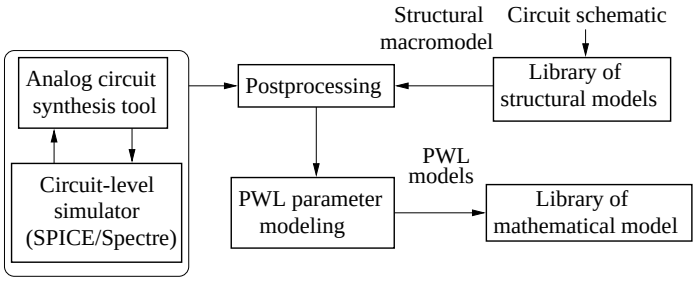

Figure 7: **Circuit modeling methodology**

the following form: $\mathcal{C}^l = \{c_1^m x_1 + c_2^m x_2 + \ldots + c_I^m x_N \ \{\leq, \geq\} \ d^m, \ m = 1 \ldots M^l, \ \ c_i^m, \ d^m \in \Re\}$, where M^l is the number of constraints for model l.

The model l is active if all the constraints in $\mathcal{C}^l$ are satisfied for a set of input values: $\{x_1, \ldots, x_I\}$ and inactive if at least one of the constraints is violated. The region in the input space where a constraint set $\mathcal{C}^l$ is satisfied is called the valid region of model l. All constraint sets $\mathcal{C}^l$ must satisfy the following requirements:

1. The valid regions of any pair of linear models must not intersect in any point in the input space: $\mathcal{C}^p \cap \mathcal{C}^r = \emptyset$, for $p \neq r$.
2. The set of constraints in $\mathcal{C}^l$ is minimal. By removing any constraint, the valid region for model l changes.
3. Constraints for a model define the smallest region in the input space, thus the interval defined by each constraint in a set $\mathcal{C}^l$ is the smallest possible.

3.2 Extraction Method

The following notations pertain to the trained NN model: N input neurons in the input layer $\mathcal{I}$, H hidden neurons in the hidden layer $\mathcal{H}$ and O output neurons in the output layer $\mathcal{O}$. The weight matrix between the input and the hidden layer is $W^{IH} = \{w_{ji}, j = 1 \ldots H, \ i = 1 \ldots N+1\}$, where w_{ji} is the weight of the connection between input neuron i and hidden neuron j. The input layer and the hidden layer are both augmented with a bias neuron with a constant output of one. The weight matrix between the hidden and the output layer is $W^{HO} = \{w_{kj}, k = 1 \ldots O, \ j = 1 \ldots H+1\}$ with w_{kj} the strength of the connection between output neuron k and hidden neuron j.

The activation function of the hidden neurons is the sigmoidal: $\phi(x) = \frac{1}{1+exp(-\lambda x)}$, with $0 < \lambda \leq 1$. The weighted sum input into a hidden neuron and into an output neuron are respectively:

$$h_j = \sum_{i=1}^{N} w_{ji} x_i, \quad h_k = \sum_{j=1}^{H} w_{kj} y_j. \tag{4}$$

where x_i is the output of the input neuron i. The output of the hidden neuron j is: $y_j = \phi(h_j)$, and the output of the output neuron y_k is: $y_k = h_k$.

The starting point of the extraction algorithm is a trained NN. The NN is first trained using a back-propagation type algorithm [36] and then pruned. Training and pruning are detailed in [12].

The steps of the extraction algorithm are: (a) Extract a PWL approximation of the activation function of each hidden neuron, and (b) Find all subregions with their PWL model and valid region constraints. For (a), a novel clustering algorithm is detailed in the next section.

3.2.1 Clustering Algorithm

The purpose of the clustering algorithm is to find a minimal number of linear segments for each hidden neuron such that the error between the linear approximation and the nonlinear activation function is small. Our algorithm belongs to the class of agglomerative clustering algorithms [14]. The new feature of our clustering algorithm is the stopping criteria which allows the algorithm to stop when an optimal number of clusters is found.

The input data for the algorithm is the of set of activation values for each hidden neuron. Since the neural network is already trained, artificial input data can be generated in addition to training data, such that the whole active region of the activation function of each hidden neuron is covered well. The data set for hidden neuron j is: $\mathcal{D}_j = \{(h_j(n), y_j(n)),\ \ n = 1,\ \ldots\ , P_j\}$. Each set $\mathcal{D}_j$ is ordered by the values of $y_j(n)$ (the sigmoidal activation function is monotone). The ordered set $\mathcal{D}_j$ is transformed into a set of linear segments that pass through each consecutive pair of points in $\mathcal{D}_j$. Each linear segment is defined by slope, intercept and its limits. The initial set of segments for hidden neuron j is: $\mathcal{S}_j = \{s_k = (p_k, i_k),\ \ B_k = ((h_j(l), y_j(l)),\ (h_j(u), y_j(u))),\ \ k = 1,\ \ldots,\ P_j - 1\}$, where p_k is the slope, i_k is the intercept, B_k are the limits of s_k. The clustering algorithm starts with the set $\mathcal{S}_j$ for each hidden neuron. The idea behind clustering is to group segments (i.e. activation points) with similar slope values. The distance function between two segments $(s_x\,, s_y)$ in a set $\mathcal{S}_j$ is the cosine of the angle between them.

Clustering algorithm is repeated for each hidden neuron. For hidden neuron j the agglomerative clustering method starts with a number of clusters equal to the number of linear segments. It then iteratively attempts at merging the closest pair of clusters together until a stopping criteria is reached. The criteria to stop merging is:

$$J(t) = \frac{R_j(t)}{(P_j - 1)} + \frac{1}{P_j}\sum_{n=1}^{P_j} |y_{lj}(n) - y_j(n)| \tag{5}$$

where $R_j(t)$ is the number of clusters at step t, $y_{lj}(n)$ is the linear output. The first term of relation 5 penalizes a large number of clusters, while the second term penalizes a large linearization error. At start, the linearization error is zero and the penalty for the number of clusters is one: $J(0) = 1$. As merging of closest clusters continues, the first term goes down, while the second term goes up. Therefore, at the beginning, the values of the criterion function $J(t)$ decrease while the penalty for a large number of clusters dominates compared to the linearization error. As merging progresses, the linearization error starts to become more important in the sum, and at one point the values of $J(t)$ go up. At that moment clustering stops. The number of clusters at the end represents the number of linear regions for hidden neuron j.

At any point in the algorithm there are $R_j(t)$ clusters defined as: $r_z(t) = \{s_z^a,\ \ s_z^a \in \mathcal{S}_j,\ \ a = 1,\ \ldots\ N_z(t)\}$, with $N_z(t)$ the number of segments in cluster $r_z(t)$. Only clusters that have a common limit point are eligible for merging. The distance between two adjacent clusters $r_x(t)$ and $r_y(t)$ is given by:

$$\begin{aligned} d(r_x(t), r_y(t)) \quad &= \quad \max_{a,b \in r_x(t) \cup r_y(t)} (d_{cos}(s_x^a, s_y^b)) + \\ &+ \quad \frac{1}{N_x(t) + N_y(t)} \sum_{k \in r_x(t) \cup r_y(t)} |y_{lj}(k) - y_j(k)| \end{aligned} \tag{6}$$

where $d_{cos}(.)$is the cosine distance function between a pair of linear segments one from each cluster. The first term - the maximum cosine distance between any pairs of segments in the two clusters - is a measure of how closely oriented are the segments in the two clusters - while the second term is the average of the absolute linearization error that would be introduced by merging clusters $r_x(t)$ and $r_y(t)$. The pair of clusters with the minimum distance $d(r_x(t), r_y(t))$ is merged at each step of the algorithm.

After each merging, the stopping criterion $J(t)$ is evaluated and if it is went up compared to the previous step, the clustering stops. The results of the algorithm are: number of clusters (R_j) for each hidden neuron j (each cluster corresponds to a linear segment) the coordinates in the input space of the upper and lower bounds of each cluster and the slope and intercept of the linear segment.

The limits of each linear segment are specified as a set of linear constraints. For example, for neuron j, linear segment z, the set of constraints ($\mathcal{C}_j^z$) is:

$$C_j^z = \left\{ \begin{array}{rcl|rcl} \sum_{i=1}^N w_{ji} x_i & \leq & M_z, & \sum_{i=1}^N w_{ji} x_i & \geq & m_z \\ x_1 & \leq & M_1, & x_1 & \geq & m_1 \\ \ldots & & & \ldots & & \\ x_N & \leq & M_N, & x_N & \geq & m_N \end{array} \right. \tag{7}$$

where m_z and M_z are the minimum and maximum values of the linear function in region z, m_i and M_i are the limits of each input variable.

3.2.2 Extraction of PWL Models

Given the set of linear segments that cover the active region of each hidden neuron, the final step of the extraction method consists in finding the PWL models $\mathcal{L}$ and their valid regions. Each valid region is characterized by a combination of linear segments, one for each hidden neuron. Each such combination uniquely defines a linear model $\mathcal{L}$. Its valid region is given by a non-empty

```
struct node { double v, i; };
void main (void) {
    struct node internal signals;

    for all blocks in the architecture do
        create an instance of the class corresponding to the block type;

    for all symbolic composition rules (SCR) in the architecture do
        create an instance of the class corresponding to the SCR type;

    connect all created objects to inputs, outputs and internal signals;
    set initial state of the system;
    for all blocks in the architecture do
        calculate the constant function sub-matrices of the block;

    for the time interval of interest at successive time instances with time step h do
        simulate the system (current time instance);
};
```

Figure 8: **Structure of the C++ program for system simulation**

intersection of constraint sets ($\mathcal{C}_j^z$) for each hidden neuron. The algorithm is exponential: The maximum number of valid regions and of PWL models is: $R_{max} = R_1 R_2 \dots R_H$. That is why pruning the NN before extraction is very important.

For each combination of linear segments, a set of constraints $\mathcal{C}^l$ is build iteratively. First, the constraints for the first hidden neuron are included, then one by one, the constraints from the sets of constraints for the rest of the hidden neurons. The purpose of this iterative process is to eliminate redundant constraints. It just checks for identical linear constraints formulas, and if it finds such, then it changes the limits of the constraint already in the set $\mathcal{C}^l$, without adding a new one. A source for identical constraints are the limits of the input variables that appear in each set $\mathcal{C}_j^z$ (see relation 7). In this way, an invalid region can also be detected early (i.e. empty intersection of identical constraints).

Once a rough set of constraints is obtained, the constraints in $\mathcal{C}^l$ are checked for validity and their limits refined using the validity interval analysis (VIA) [43, 31]. The validity of $\mathcal{C}^l$ is checked by a linear programming solver with the first constraint chosen as objective function, the optimization type - minimization (for $\geq$) or maximization (for $\leq$), and the rest of the inequalities as constraints. If the linear solver returns an acceptable solution then the input region defined by the $\mathcal{C}^l$ is non-empty, and therefore the combination is valid.

The goal of refining the limits of the constraints in each valid set $\mathcal{C}^l$ is to eliminate redundancy in constraint limits (see requirement 3 in Section 3.1). The limits of each constraint in the set $\mathcal{C}^l$ are adjusted iteratively using the linear optimizer: at each step, a constraint becomes the objective function and a minimization/maximization is done depending on the inequality type of the constraint, with the rest of the inequalities as constraints. The limit of the optimized constraint is adjusted if the solution returned by the solver is more restrictive. The procedure for adjusting the limits stops when none of the constraint limits undergoes any changes. The convergence of the VIA method for single layer networks (which is the case here: input and hidden layer) is assured within one step [31].

For each valid combination region defined by $\mathcal{C}^l$ the output of the network is expressed as a linear combination in the input variables: $y^l = a_1^l x_1 + a_2^l x2 + \dots a_N^l x_N + a_{N+1}^l$. The coefficients a_i^l are functions of the weights of the network and of the slopes and intercepts of the linear regions of the hidden neurons determined in the clustering algorithm. The set of linear models defined by coefficients a_i^l, together with the set of constraints of the valid combinations $\mathcal{C}^l$ represent the result of the extraction method.

4 C++ Code for Customized Simulation

Figure 8 presents the pseudocode of the C++ *main* program of a simulator customized to a given system. The program includes local variables defined for all internal nodes of the system. Each variable includes a voltage and a current component. Then, objects are instantiated for each block of the system depending on the kind of the block. These classes embed the structural model of the circuit including the symbolic composition rules (SCR) to compute its function and state sub-matrices. For example, similar objects are created for all identical OTA circuits using the class defined for these OTA in Figure 9. Similar classes exist for op amp

```
class OTA {
    public:
        friend class OTA_ActiveC_Integrator;
        OTA (parameter list);
        set_input(input signals);
        simulate();
        get_output();
    protected:
        Node in_p, in_n; // input ports
        Node out_p, out_n; // output ports
        double F[4][4]; // function sub-matrix
        double S[4]; // state sub-matrix
        set_initial_state();
        update_state();
    private:
        calculate_function_matrix();
        calculate_state_matrix();
};
```

Figure 9: **C++ code for OTA class**

circuits, comparators and so on. Then, objects are instantiated for each block composition depending on the specific partitioning of the system architecture. For the $\Delta\Sigma$ ADC in Figure 1(a) and its decomposition shown in Figure 2, all active OTA-C integrator models are obtained through using the same composition rule of the OTA objects and the objects describing capacitors. Objects are connected through the variables describing inputs, outputs and nodes. Next, the system is initialized by setting all its state variables to their initial values. Finally, the code for time domain simulation calls the system simulation method at successive time instances that are separated by h, the time integration step.

Figure 9 presents the pseudocode for the C++ class corresponding to OTA circuits. Any other circuit class has a similar structure. Protected variables correspond to the two input nodes representing the differential inputs, the two output nodes for the differential output of the OTA circuit, and the arrays used to store the values for the circuit function and state sub-matrices. Finally, the class methods set the numerical values of the model components, like the values of capacitors, resistors and constant current and voltage sources, initialize the state sub-matrix, calculate the state and function sub-matrix elements, update the value of the state sub-matrix for the next instance of time, and simulate the OTA circuit over time. Classes for SCR have similar structures.

The methods for computing state and function sub-matrices as well as for updating the state are optimized by sharing common sub-expressions. Common sub-expression sharing is a popular compiler technique for reducing program execution time [35]. In our case, the function sub-matrix F of the OTA class, for example, has elements F_{31} and F_{32} defined by the following symbolic expressions:

$$F_{31} = \frac{8hG_mG_{m4}G_{m5}G_{mo}(L_4+hR_4)R_5}{(h+C_3R_3)(h+C_5R_5)} \text{ and } F_{32} = \frac{-8hG_mG_{m4}G_{m5}G_{mo}(L_4+hR_4)R_5}{(h+C_3R_3)(h+C_5R_5)}.$$

Then, the simulation code calculates the numerical value of element F_{31} and then assigns its negative value to element F_{32} without actually computing the expression represented by the symbolic formula of F_{32}. Similarly, the state sub-matrix S includes elements S_{11} and S_{12} defined by the following expressions

$$S_{11} = C_d\frac{V_{os}-V_{cd0}}{h+C_dR_d} + C_{cm}\frac{V_{os}-V_{cm0}}{h} \text{ and } S_{12} = -C_d\frac{V_{os}-V_{cd0}}{h+C_dR_d} + C_{cm}\frac{V_{cm2}}{h}.$$

The C++ code includes the temporary variables $temp_1 = \frac{C_{cm}}{h}$ and $temp_2 = C_d\frac{V_{os}-V_{cd0}}{h+C_dR_d}$. The two temporary variables are first calculated, and their values are re-used for finding the two matrix elements. Common expression sharing is very efficient, especially for code that is repeatedly executed inside a loop statement, like the simulation loop. Method $update\ state\ matrix$ - called at each new time instance of the simulated time range, computes sub-expressions, such as $\frac{h}{h+C_dR_d}$, $\frac{hG_mR_3}{h+C_3R_3}$, and so on. These sub-expressions are constant, and thus represent invariants for the simulation loop [35]. Invariants are extracted outside the simulation loop, and executed only once before the loop execution starts. As experiments show, loop invariants elimination is an important source for speeding-up simulation speed - especially for the methods of CSR classes.

Figure 10 shows the pseudocode of the system simulation algorithm. At each time instance, the algorithm identifies the current

```
void simulate the system (current time instance) {
    while current PWL segments are not correct do
        for all blocks following the top-down structure of the architecture do
            calculate the port signals of the blocks;
        for all blocks in the architecture do
            find the corresponding PWL segments and check if they are correct;
            if PWL segments are not correct then update PWL segments;

    for all blocks following the top-down structure of the architecture do
        update the state matrix S;
};
```

Figure 10: **Pseudocode of the system simulation method**

PWL region for the nonlinear devices. The identification step first calculates the voltages and currents through the nonlinear devices by traversing top-down the structure hierarchy of the system and assuming that the PWL regions for the current time instance remain the same as those for the previous time moment. If this assumption is incorrect then the algorithm re-iterates the calculation of the voltages and currents after switching to the closest PWL region, and so on. The iteration process stops when the closest feasible PWL region was found. The nonlinear component PWL checking and adjusting process are inside each basic module. They are called before state sub-matrices are updated. Changing the current PWL region of a module also result in updating the function sub-matrices F_i of this modules and all composed modules that contain this module.

5 Experiments

The first part of this section presents experimental results for PWL modeling of circuit nonlinearities. The second part discusses our case study for compiled-code simulation of $\Delta\Sigma$ modulators.

5.1 PWL model extraction

As an initial modeling experiment, the method for linear model extraction was applied to model the amplitude frequency response of an analog transconductance amplifier (OTA) [28] for different parasitic levels. The data was obtained using SPICE simulation of the analog circuit sampled in a large number of frequency and parasitic values. The two inputs - frequency (f) and parasitic (c) - and the output - the gain - were first normalized. A three layer neural network with $I = 2$ inputs and $H = 7$ hidden neurons was trained, such that the performance on both training and testing data were very good. A larger number of hidden neurons did not improve the approximation. The trained NN was then pruned. From the initial set of weights between the input and hidden neurons ($(I + 1)H = 21$) eight weights were eliminated. Because one of the hidden neurons gets disconnected completely from the input layer, it was removed and the network retrained. Figure 11 shows the SPICE values of the gain compared to the unscaled output of the pruned NN for seven parasitic values. The NN output approximates very well the real values.

The clustering algorithm was then applied to each hidden neuron. Two of the hidden neurons had constant outputs given by the bias weight - the weights to the input variables were all pruned. The rest of the hidden neurons were clustered into 6, 9, 3 and 8 clusters respectively. Figure 12 shows the linear segments obtained by clustering for the hidden neuron with 6 clusters.

From a total of 1,296 combinations of linear regions of the hidden neurons, only 136 have a non-empty solution set. For each valid combination, a linear model of the network output was computed. The result of the piecewise linear extraction method is presented in Figure 12. The dotted plot represents the piecewise linear model output, while the line represents the true values simulated with SPICE. It can be seen that the piecewise linear approximation is very accurate. Previous approaches to extract linear models from trained neural networks [40] and [11] using a fixed number of segments for each hidden neuron do not have the same accuracy. In this case the activation function of each hidden neuron is split into three linear regions, where the output is either zero, linear dependence or one. The accuracy of the extracted model is much worse than that obtained with the present method.

A second experiment focused on PWL model extraction for the OTA transconductance gm as a function of voltage V_{idm} at the OTA differential input ports. The extraction results are shown in Figure 13. Spectre simulation data was shown with dots, output of the NN with stars, and output of the PWL model with circles. The trained NN had 3 hidden neurons, from which one was

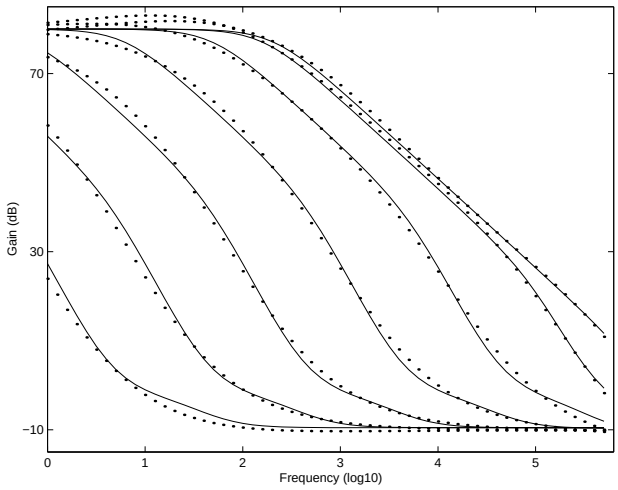

Figure 11: **The gain frequency response. The pruned neural network output is represented with dots and the SPICE values with lines. Each curve corresponds to a different value of the parasitic.**

ADC order	Spectre (sec)	Compiled code (sec)	Speed up
1	507.1	3.5	144.88
2	533.8	5.88	90.79
3	852.3	8.24	103.43
4	1284.9	10.69	120.19
5	1752.0	12.91	135.70

Table 1: **Simulation time for symbolic method vs. Spectre**

eliminated after pruning. Clustering extracted 7 and 8 linear segments from each hidden neuron. Only 14 linear models out of a total of 56 had solutions. As the figure shows, the accuracy of the extracted PWL model is very good compared to the trained NN.

We also developed piecewise-linear models for the current-voltage relationships at the terminals of popular analog building blocks, such as differential stages, current mirrors, output stages etc. For example, the output stage was modeled as the dependence of the output current to the input voltages and the widths of the two transistors. Then, the simulation behavior of complex analog circuits were obtained using the extracted, piecewise linear models. The piecewise linear models were composed by using Kirchhoff's and Ohm's laws to obtain the current-voltage behavior at the terminals of the complex circuits. This behavior is useful for efficient AC and time-domain simulation. Due to its event-driven nature, this simulation is much more efficient than continuous time simulation. This is very important for analog circuit synthesis as it is well known that the large simulation time is the bottleneck of the synthesis cycle.

5.2 System simulation

For $\Delta\Sigma$ ADC simulation, Table 1 compares the simulation times for Spectre simulator and for the proposed compiled-code simulation method. The table relates the speed-up of the proposed method as a function of the ADC complexity (order). Results are shown for 1st to 5th order single loop $\Delta\Sigma$ ADC. The resulting speed-up is significant, it varies between 90 and 144 times. Please note that these speed-ups were obtained without actually affecting the accuracy of simulation. The same netlist (composed of circuit macrocells) was simulated in both cases. The huge speed-up is due to employing customized compiled-code simulation, and the two optimizations for the expressions of function and state sub-matrix elements (propagation of common sub-expressions and elimination of loop invariants from inside the time-domain simulation loop). The symbolic method shows a linear increase of the simulation time with the order, thus the number of state variables. This is explained by the fact that only state variables are recomputed inside the loop. The time complexity of the numerical simulator grows at a much faster rate. We expect that the speed up will grow with the order of the ADC.

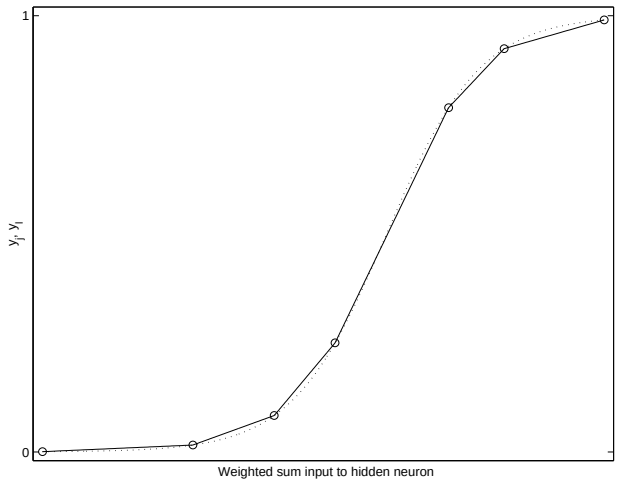

Figure 12: **An example of the results of the clustering algorithm. The limits of the clusters are plotted with circles (o). The piecewise linear approximation (y_l) is represented by lines, and the sigmoidal function values (y_j)with dotted line.**

The second experiment studied the importance of circuit non-idealities (like poles, zeros, input and output impedances etc) on the accuracy of ADC simulation. Figure 14 shows the signal to noise ratio (SNR) and dynamic range (DR) plots for the ADC. The maximum SNR is 64dB and DR is 67dB. Similar values resulted through Spectre simulation. This motivates the correctness of the symbolic method. The figure also shows the importance of using detailed circuit models, such as circuit models including poles and zeros, rather than ideal models. In the right part of Figure 14, the three plots with dotted lines correspond to simulations, which used circuit macromodels with one pole and two poles. In the first two cases, the system still worked as an ADC, but the SNR went down by about 5dB and 13dB, and the DR by about 4dB and 12dB respectively due to the poles. In the third case, the poles prevented the system from a correct functioning. This example argues that using detailed circuit models is compulsory. Handling various non-idealities is much easier using the proposed symbolic technique, in which system modeling is fully automated. Some of the existing behavioral simulation methods for $\Delta\Sigma$ ADC (like [15] and [33]) require extensive designer expertise to develop the models, and are cumbersome, if more non-ideal elements were to be considered.

6 Conclusions

This paper presents a novel method for fast time-domain simulation of analog systems with nonlinear parameters. Code generation relies on calculating symbolic expressions for the output and state variables, as well as the voltages and currents of nonlinear devices in a system. Code optimization identifies and eliminates from the simulation loop all loop invariants, and propagates constant sub-expressions. System simulation uses detailed structural macromodels for the building blocks including non-idealities, like finite gain, poles, zeros, CMRR, phase margin, fall/rise time etc. Nonlinear parameters are expressed using PWL models extracted from trained NN. Using a modified clustering method, the method automatically determines the best number of linear regions to approximate each hidden neuron activation function. The adaptive clustering improves the accuracy of extracted PWL models over other extraction approaches. The paper presents $\Delta\Sigma$ ADC simulation as a case study. Using compiled-code simulation, the proposed technique achieves important speed-ups as compared to Spectre simulation. Simulation accuracy is not affected. As compared to existing behavioral simulation methods for $\Delta\Sigma$ ADC, this technique is more automated and uses more accurate circuit models.

References

[1] IEEE Transactions on CADICS, Special Issue on Behavioral Modeling and Simulation of Mixed-Signal/Mixed-Technology Circuits and Systems, Vol. 22, February 2003.

[2] R. Bryant et al, "COSMOS: A Compiled Simulator for MOS Circuits", *Proc. Design Automation Conference*, 1987, pp. 9-16.

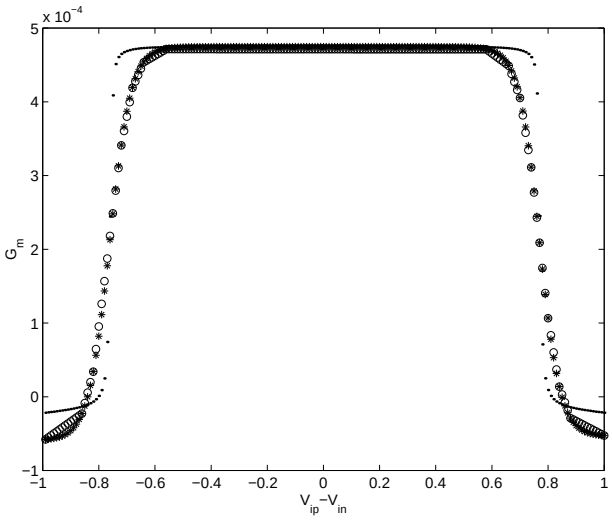

Figure 13: **Extracted PWL model for OTA gm**

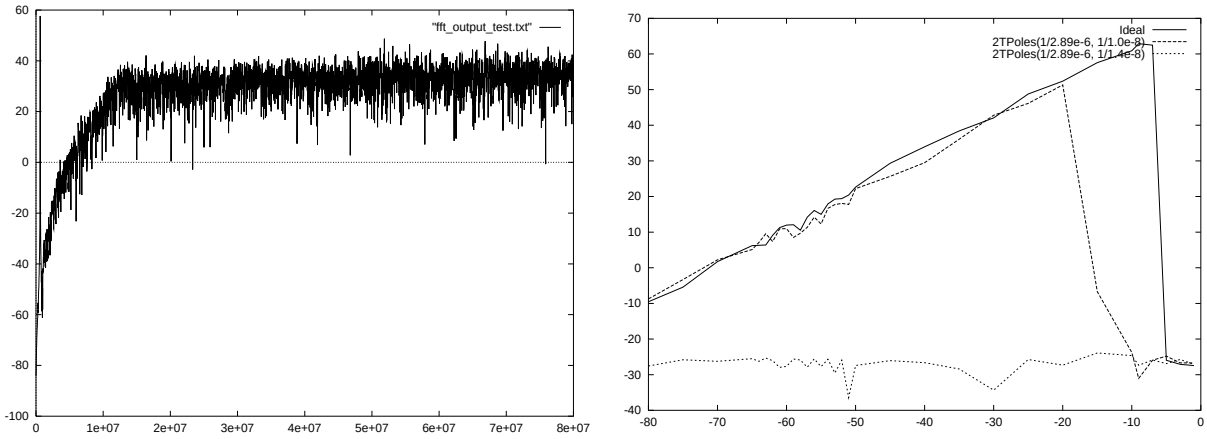

Figure 14: **SNR and DR plots for $\Delta\Sigma$ ADC**

[3] Z.-Q. Chen, S.-F. Chen and Z.-H. Zhou, "A statistics based approach for extracting priority rules from trained neural networks", *Proc. of Int. Joint Conf. on Neural Networks*, IJCNN'2000.

[4] J. Cherry, W. M. Snelgrove, "Continuous-Time Delta-Sigma Modulators for High-Speed A/D Conversion", *Kluwer*, 2000.

[5] M.W. Craven, J.W. Shavlik, "Using sampling and queries to extract rules from trained neural networks", *Proc. of the Eleventh International Conference on Machine Learning*, *Morgan Kaufmann*, San Mateo, CA, 1994.

[6] M.W. Craven, J.W. Shavlik, "Extracting tree-structured representations of trained neural networks", In D. Touretzsky, M. Mozer, M. Hasselmo, eds., *Advances in Neural Information Processing Systems*, vo. 8. pp. 24-30, 1996.

[7] W. Daems, G. Gielen, W. Sansen, "Simulation-Based Automatic Generation of Signomial and Posynomial Performance Models for Analog Integrated Circuits", *Proc. International Conference on Computer-Aided Design*, 2001.

[8] W. Daems, G. Gielen, W. Sansen, "An Efficient Optimization-Based Technique to Generate Posynomial Performance Models for Analog Integrated Circuits", *Proc. Design Automation Conference*, 2002, pp. 431-436.

[9] N. R. Dhanwada, A. Nunez, R. Vemuri, "Hierarchical Constraint Transformation using Directed Interval Search for Analog System Synthesis", *Proc. Design, Automation and Test in Europe Conference*, 1999, pp. 328-335.

[10] A. Doboli, R. Vemuri, "A Regularity-based Hierarchical Symbolic Analysis Method for Large-scale Analog Networks", *IEEE Trans. Circuits & Systems - Part II*, No 11, pp. 1054-1068, 2001.

[11] S. Doboli, G. Gothoskar, A. Doboli, "Piecewise Linear Modeling of Analog Circuits based on Model Extraction from Trained Neural Networks", *Proc. IEEE International Workshop on Behavioral Modeling and Simulation*, 2002.

[12] S. Doboli, A. Doboli, "Piecewise-Linear Modeling of Analog Circuits using Trained Feed-Forward Neural Networks and Adaptive Clustering of Hidden Neurons", *Proc. International Joint Conference on Neural Networks (IJCNN)*, 2003.

[13] W. Duch, R. Setiono, J. Zurada, "Computational intelligence methods for rule-based data understandin", *Proceedings of the IEEE*, Vol. 92, No. 5, pages 771-805, 2004.

[14] R.O. Duda, P.E. Hart, "Pattern classification and scene analysis", *Wiley*, 1973.

[15] K. Franken, G. Gielen, "A High-Level Simulation and Synthesis Environment for $\Delta\Sigma$ Modulators", *IEEE Transactions on CADICS*, No. 8, pp. 1049-1061, 2003.

[16] L.M. Fu, "Rule generation from neural networks", *IEEE Transactions Syst., Man., Cybern.*, 28:1114–1124, 1994.

[17] K. Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks", pp. 183-190, 1988.

[18] G. Gielen, R. Rutenbar, "Computer Aided Design of Analog and Mixed-signal Integrated Circuits", *Proc. of IEEE*, vol 88, No 12, Dec 2000, pp. 1825-1852.

[19] M. Golea, "On the complexity of rule-extraction from neural networks and network-querying", *Proc. Rule Extraction From Trained Artificial Neural Networks Workshop*, AISB 1996.

[20] G. J. Gomez et al, "A Generic Parameterizable CMOS OTA Macromodel", *IEEE Transactions on Circuits & Systems - Part I*, 1995.

[21] R. Harjani, J. Shao, "Feasibility and Performance Region Modeling of Analog and Digital Circuits", *Analog Integrated Circuits and Signal Processing*, Kluwer, 1996.

[22] X. Huang et al, "Modeling Nonlinear Dynamics in Analog Circuits via Root Localization", *IEEE Transactions on CADICS*, No 7, pp. 895-907, 2003.

[23] M. Krasnicki, R. Phelps, R. Rutenbar, R. Carley, "MAELSTROM: Efficient Simulation-Based Synthesis for Custom Analog Cells", *Proc. Design Automation Conference*, 1999, pp.945-950.

[24] R. Krishnan, "A systematic method for decompositional rule extraction from neural networks", *Proc. NIPS'97 Rule Extraction from Trained Artificial Neural Networks Workshop*, pp 38-45, 1996.

[25] D. Lewis, "A Hierarchical Compiled Code Event-Driven Logic Simulator", *IEEE Transactions on CADICS*, Vol. 10, pp. 726-737. 1991.

[26] F. Leyn et al, "Analog Small-signal Modeling. Behavioral Signal- Path Modeling for Analog Integrated Circuits", *IEEE Transactions on Circuits & Systems - Part II*, No. 7, pp. 701-711, 2001.

[27] D. Leenaerts, W. van Bokhoven, "Piecewise Linear Modeling and Analysis", *Kluwer*, 1998.

[28] K. Laker, W. Sansen, "Design of Analog Integrated Circuits and Systems", *McGraw Hill*, 1994.

[29] H. Liu, A. Singhee, R. Rutenbar, R. Carley, "Remembrance of Circuit Past: Macromodeling by Data Mining in Large Analog Design Spaces", *Proc. Design Automation Conference*, 2002, pp. 437-442.

[30] F. Maire, "A partial order for the M-of-N rule extraction algorithm", *IEEE Trans. Neural Networks*, vol. 8, pp. 1542-1544, 1997.

[31] F. Maire, "On the convergence of validity interval analysis", *IEEE Trans. Neural Networks*, vol. 11, No. 3, 802-808, 2000.

[32] P. Maurer, "Event Driven Simulation Without Loops or Conditions", *Proc. International Conference on Computer Aided Design*, 2000.

[33] F. Medeiro, "Top-Down Design of High Performance Sigma-Delta Modulators", *Kluwer*, 1998.

[34] A. Moscovici, "High Speed A/D converters - understanding Data Converters through SPICE", *Kluwer*, 1999.

[35] S. Muchnik, "Advanced Compiler Design and Implementation", *Morgan Kaufmann*, 1997.

[36] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning internal representations by error propagation". In D.E. Rumelhart and J.l. McClelland, editors, *Parallel Distributed Processing*, volume I+II, MIT Press, 1986.

[37] K. Saito, R. Nakano, "Law discovery using neural networks", *Proc. NIPS'97 Rule Extraction From Trained Artificial Neural Networks*, vol. 8, pp. 62-69, 1996.

[38] R. Setiono, "Extracting rules from neural networks by pruning and hidden unit splitting", *Neural Comput.*, vol. 9, pp. 205-225, 1997.

[39] R. Setiono, W.K. Leow, "FERNN: An algorithm for Fast Extraction of Rules from Neural Networks", *Journal of Applied Intelligence*, 2000, Vol. 12, No. 1/2, pages 15-25.

[40] R. Setiono, W.K. Leow, J.M. Zurada, "Extraction of rules from artificial neural networks for nonlinear regression", *IEEE Trans. Neural Networks*, 13(3):564–577, 2002.

[41] A.Tickle et al, "The truth will come to light: directions and challenges in extracting knowledge embedded within trained artificial neural networks", *IEEE Trans. Neural Networks*, 9(6), pp. 1057–1068, 1998.

[42] P.R. Thie, "An introduction to linear programming and game theory", *John Wiley & Sons*, 1988.

[43] S. Thrun, "Extracting rules from artificial neural networks with distributed representations", In D. Touretzky G. Tesauro and T. Leen, editors, *Advances in Neural Processing Systems*, volume 7. MIT Press, 1995.

[44] A.B. Tickle, R. Andrews, M. Golea, J. Diederich, "The truth will come to light: directions and challenges in extracting knowledge embedded within trained artificial neural networks", *IEEE Trans. Neural Networks*, 9(6):1057–1068, 1998.

[45] G. Towell, J. W. Shavlik, "The extraction of refined rules from knowledge-based neural networks", *Machine Learning*, vol. 131, pp. 71-101, 1993.

[46] H. Tsukimoto, "Extracting rules from trained neural networks", *IEEE Trans. Neural Networks*, vol. 11, no. 2, pp. 377-389, 2000.

[47] P. Vancorenland *et al*, "A Layout-Aware Synthesis Methodology for RF Circuits", *Proc. International Conference on Computer Aided Design*, 2001, pp. 358-362.

[48] J. Vlach, K. Singhal, "Computer methods for circuit analysis and design", *Van Nostrand Reinhold*, 1983.

[49] Y. Wei, A. Doboli, "Systematic Development of Analog Circuit Structural Macromodels through Behavioral Model Decoupling", *Proceedings of Design Automation Conference (DAC)*, 2005

[50] H. Zhang, A. Doboli, S. Doboli, "Fast Time-Domain Simulation Through Combined Symbolic Analysis and Piecewise Linear Modeling", *Proc. of Behavioral Modeling and Simulation Workshop (BMAS)*, 2004.

[51] H. Zhang, A. Doboli, "Fast Time Domain Symbolic Simulation for Synthesis of $\Sigma\Delta$ Analog-Digital Converters", *Proc. of International Symposium on Circuits and Systems (ISCAS)*, 2004.