

# Fast Time-Domain Simulation Through Combined Symbolic Analysis and Piecewise Linear Modeling

Hui Zhang and Alex Doboli

Department of Electrical and Computer Engineering  
Stony Brook University, Stony Brook, New York 11794  
{huizhang, adoboli}@ece.sunysb.edu

## ABSTRACT

This paper presents a method for fast time-domain simulation of analog systems with nonlinear parameters. Specifically, the paper focuses on  $\Sigma$ - $\Delta$  analog-to-digital converters (ADC). The method generates 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 (NN). As compared to existing behavioral simulation methods for  $\Sigma$ - $\Delta$  ADC, this technique is fully automated and more accurate. In our experiments, compiled-code simulation was about 100x faster than Spectre (numerical) simulation.

## 1. INTRODUCTION

Systems-on-chip (SOC) are emerging as the next showstopper in microelectronics. In spite of their commercial promise, designing mixed-signal SOC continues to be a challenging and expensive task, which demands expertise in many orthogonal areas. Also, there is limited CAD support to boost design productivity. It is fair to say that SOC design has a bottleneck in the steps of designing RF and analog IP cores, as well as integrating and verifying the final design. Existing research (please refer to [6] for an overview) offers remarkable solutions to synthesis of analog circuits, like opamps, operational transconductors (OTA), comparators, and so on. The next step is to tackle synthesis of complex analog and mixed-signal systems, like ADC, DAC, PLL, and transceivers. For this endeavor, however, existing analog and mixed-signal simulators [5,6,10], which are the core of analog synthesis [6], are still too slow to be used inside the SOC synthesis loop, experience numerous stability problems [5], and are unable to exploit the specifics of circuits and systems.

Vlach and Singhal [13] offer a comprehensive presentation of the fundamental simulation algorithms. For speeding-up simulation, behavioral models are used, so that irrelevant details are abstracted away. Please refer to [14] for the most recent advancements in behavioral modeling and simulation. Circuit models are of two kinds: *structural (physical)* and *mathematical models*. Structural models offer a qualitative insight into the circuit, but they do not give any quantitative perspective. Hence, for analog synthesis, structural models must be complemented by mathematical models, which express quantitative dependencies between the design parameters and performance attributes of a circuit. Physical modeling methods, in general, simplify a circuit to a reduced sub-circuit that includes only the dominant devices. Mathematical modeling includes linear and non-linear regression, Volterra series, Pade approximations, wavelet functions, and NN [14]. The most important limitations of existing modeling techniques include difficulties in handling nonlinear parameters and large circuits. Also, large amounts of sampling data are needed for modeling.

This paper presents a new approach to fast time-domain simulation of analog systems that contain nonlinear parameters. Without trading-off accuracy, the proposed technique achieves speed-ups of more than 100x as compared to Spectre (numerical) simulation by generating *compiled-code simulators*. Code generation relies on calculating symbolic expressions for the output voltages and

currents, and the state variables of a system. Code optimization identifies and eliminates all loop invariants [11], and propagates constant sub-expressions [11] present in the simulation loop. To avoid the large memory requirements specific to symbolic analysis, the suggested method exploits regularities of a net-list [3]. It is known that regularities are very efficient for AC modeling and simulation of linear systems [3]. Code generation uses detailed structural macromodels for the building blocks (OTA, opamp, and comparator), including non-idealities, like finite gain, poles and zeros, CMRR, phase margin, fall and rise time, and so on. This paper concentrates on  $\Sigma$ - $\Delta$  ADC [2] simulation as a case study.

Nonlinear parameters are described using PWL models. The paper presents a new algorithm for extracting PWL models from trained NN. NN are capable to learn any type of nonlinear mapping based on their well-known property of universal approximators [15]. The proposed method addresses the need of automatically creating PWL models [16]. The extraction algorithm approximates the sigmoidal functions of the intermediate neurons with an adaptively chosen number of linear segments. PWL models result for the nonlinear parameters through composition of the linearized neurons. NN training used sampled design points obtained with Analog Design Automation's Creative Genius v1.5 analog circuit synthesis tool.

Compared to other fast simulation methods for  $\Sigma$ - $\Delta$  ADC [5,10], our technique is fully automated, and uses detailed circuit models. Hence, it offers the benefit of more accurate simulation, and thus the advent of faster analog design closure. ADC simulation in [5,10] relies on behavioral models, which are known to be imprecise [5]. 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 first attempt of addressing compiled-code simulation for continuous-time systems with non-linear parameters. Existing compiled-code simulators [1,8,9] are for discrete and event-driven systems.

This paper is organized as six sections. Section 2 discusses the simulation methodology. Section 3 presents modeling of the ADC blocks. Next, system modeling is detailed, and simulation results are given in Section 5. Section 6 discusses our conclusions.

## 2. SIMULATION METHODOLOGY

Figure 1 presents the proposed circuit modeling method based on PWL models. It generates nonlinear mathematical models for the parameters of a structural circuit model. 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 3 details some of the 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 3 details some of the equations used in post-processing. The last step *automatically* produces PWL models for the nonlinear parameters. Section 3.C presents the proposed PWL modeling method.

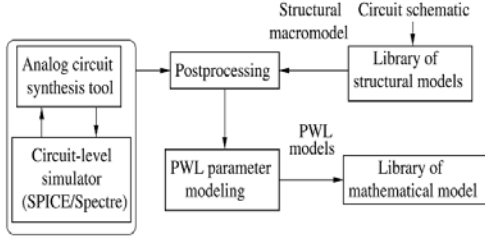


Figure 1: Circuit modeling methodology

Figure 2 shows the system simulation method, and the generic structure of cascaded  $\Sigma$ - $\Delta$  ADC [2]. The first step finds the structural regularities in the ADC architecture. Figure 5 shows the regularities for a 3<sup>rd</sup> order  $\Sigma$ - $\Delta$  ADC. Then, symbolic expressions are calculated for each of the found sub-structures (patterns) using a method that symbolically replaces time derivatives of state variables with their differences (based on Backward Euler Integration [13]). Section 4 discusses this step. The next step links the parameters in the symbolic expressions 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.

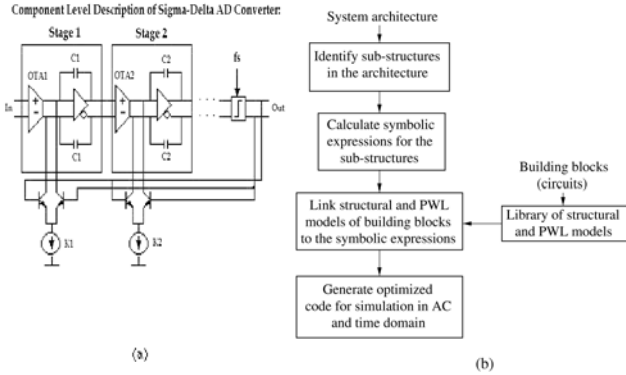


Figure 2:  $\Sigma$ - $\Delta$  ADC structure and system simulation method

### 3. CIRCUIT MODELING

This section presents the modeling of the building blocks in an ADC: OTA, opamp, and comparator circuits.

**A. OTA and opamp modeling.** For OTA modeling, we started from the macromodel proposed by Gomez *et al* [7]. 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 3 shows the 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* [7]: (1)  $V_{os}$  resulted directly through SPICE/Spectre simulation; (2)  $C_{cm} = 1 / (4 \pi |Z_{icm}(f_1)|)$ ; (3)  $C_d = 1 / (2\pi f_1 |Z_{idm}[Im]| f_1) - C_{cm}$ ; (4)  $R_d = Z_{idm}[Re](C_d + C_{cm}) / C_d^2$ ; (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)  $1 / (R_o C_o)$  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.C.

The opamp structural 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.

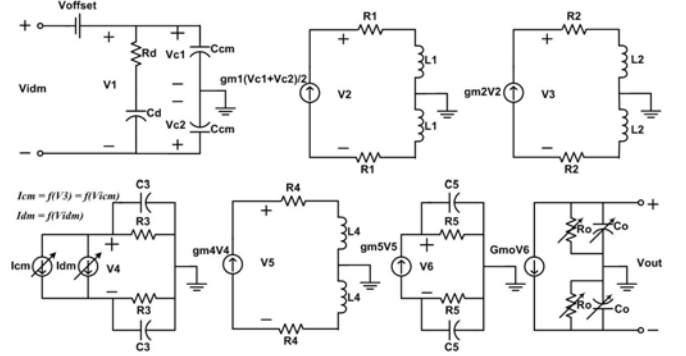


Figure 3: OTA structural macromodel

**B. Comparator modeling:** Figure 4 presents the comparator model. This structural model was based on the model by Moscovici [11]. The comparator model has the same input stage as the OTA and opamp models. The nonlinear  $G_m$  stage expresses the self-limiting behavior of the differential pair by using a hyperbolic tangent function.  $G_m$  is bounded to the range  $-I_{con}$  and  $I_{con}$ . As explained in [11], 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 2<sup>nd</sup> pole; (4)  $R_3 C_3$  is the 1<sup>st</sup> pole/ delay time; and (5)  $V_h, V_1$  are related to the minimum and maximum output voltages.

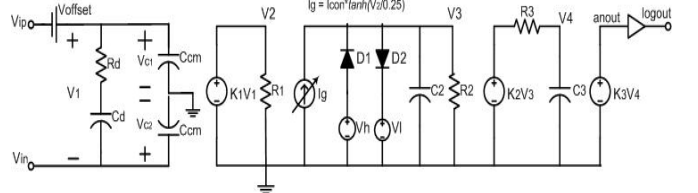


Figure 4: Comparator structural macromodel

**C. PWL Modeling of Nonlinear Parameters.** NN are capable to learn any type of nonlinear mapping based on their well-known property of universal approximators [15]. For system simulation, the implicit model embedded in an NN must be extracted as a symbolic relationship, so that the models of the building blocks can be composed together into the system model. NN cannot be directly composed. Most model extraction techniques were developed for classification, which is different from our problem (for a review see [17]). Recently two new techniques have been proposed to extract linear models for regression problems [18,19]. The main steps of the two extraction processes are as follows: A 2-layer NN is first trained and pruned. In [18] the activation function of each hidden neuron is then approximated with a fixed set of PWL functions - three or five. The input space is then split into a set of regions for each hidden neuron, such that an input point in one of the regions activates one PWL function. For each non-empty intersection of input regions - one for each hidden neuron - the output activation function can be expressed as a linear combination of the input variables. The method proposed in [19] differs from [18] in the way the linear models are generated, and

how the limits of the input regions for each valid model are determined. None of the methods chooses a different number of linear segments per each hidden neuron depending on their activation region. As a result, the PWL approximations are coarse.

The PWL extraction approach proposed here differs from [18,19] by approximating the activation function of each hidden neuron with a variable number of linear segments depending on the neuron's activation region. A clustering algorithm automatically detects the number of segments for each hidden neuron as well as its limits. This accuracy of the PWL approximation is thus superior to the one in [19].

**Problem definition:** The task is to approximate a nonlinear mapping represented by a trained feed-forward NN with a PWL mapping. We consider a three layer feed-forward network with  $N$  neurons in the input layer  $II$ ,  $H$  neurons in the hidden layer  $HH$ , and  $O$  neurons in the output layer  $OO$ . The weight matrix between the input and the hidden layer is  $W^{HH} = \{w_{ji}, j = 1..H; i = 1..N+1\}$ , 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  $W^{HO} = \{w_{kj}, k = 1..O; j = 1..H+1\}$  with  $w_{kj}$  the strength of the connection between output neuron  $k$  and hidden neuron  $j$ . Hidden neurons have a sigmoidal activation function, and the output neuron a linear one.

Extraction finds a set of  $L$  linear models of the following form:

$$LL = d^l_1 x_1 + d^l_2 x_2 + \dots + d^l_N x_N + d^l_{N+1},$$

$l = 1 \dots L$ ;  $d^l_{(\cdot)} \in \mathfrak{R}$ , where  $x_i$  is the output of a neuron in the  $II$  layer. The region in the input space where the  $l$ -th model is valid is defined by a set of linear constraints of the form:

$$CC^l = c^m_1 x_1 + c^m_2 x_2 + \dots + c^m_N x_N + d^m \{ \leq; \geq \} 0;$$

$m = 1..M^l$ ;  $c^m_i, d^m \in \mathfrak{R}$ .  $M^l$  is the number of constraints for model  $l$ .

Model  $l$  is *active*, if all constraints in  $CC^l$  are satisfied for a set of input values  $x_1, \dots, x_N$ , and *inactive*, if at least one of the constraints is violated. The input space region, which satisfies a constraint set  $CC^l$ , is called the *valid region* of model  $l$ . All constraint sets  $CC^l$  must satisfy the following two requirements:

- **Validity:** The valid regions of any pair of linear models must not intersect in any point in the input space:

$$CC^p \cap CC^r = \emptyset, \text{ for } p \neq r.$$

- **Minimality:** The set of constraints in  $CC^l$  is minimal. Thus, removing any constraint changes, the valid region of the model.

The first step of the proposed PWL model generation technique is the training of a neural network (NN) using the back-propagation algorithm until a desired accuracy is achieved on a test data set. Second, a pruning method eliminates insignificant weights and/or hidden neurons. Third, the sigmoidal activation function of each hidden neuron is approximated with a PWL function with a variable number of segments. A clustering algorithm automatically determines the number of segments, its limits, and the linear approximation on each segment. Finally, the PWL functions of 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. In this paper, we focused on clustering and PWL model extractions, as they are the more important components of the method. Additional details are given in [4].

**Clustering Algorithm:** The activation function of hidden neurons is the sigmoidal function  $\phi(x) = 1/(1+e^{-\lambda x})$ , with  $0 < \lambda \leq 1$ . The weighted sum inputs into a hidden neuron and into an output neuron are respectively  $h_j = \sum_{i=1}^N w_{ji} x_i$ , and  $h_k = \sum_{j=1}^H w_{kj} y_j$ .  $x_i$  is the output of the input neuron  $i$ . The output of the hidden neuron  $j$  is  $y_j = \phi(h_j)$ , and that of the output neuron  $y_k$  is  $y_k = h_k$ .

The clustering algorithm approximates the nonlinear sigmoidal activation function of each hidden neuron with a group of PWL functions. The clustering process determines the number of linear regions as well as their limits. The idea is to group input points - sampled from the input region of interest - that correspond to the same slope of the activation function. The main feature of the clustering algorithm is the stopping criteria, which ends the algorithm when an optimal number of clusters are found.

The first step consists in finding the activation values of each hidden neuron by using all the available input data points ( $x_n$ ) to evaluate the weighted sum  $h_j$ , and the output  $y_j$  of the sigmoidal function  $y_j = \phi(h_j(x_n))$ . Then, the output points ( $y_j$ ) are sorted in ascending order, and only distinct activation points are selected for clustering -  $N_i$ . The clustering algorithm is a modified agglomerative clustering technique [17]. First, a linear segment passing through each pair of consecutive output values is defined by computing its slope and intercept. The distance between two such segments is defined as the cosine of the angle between them

$$d_{\cos}(c_{r1}, c_{r2}) = (v_{r1} v_{r2}^T) / (|v_{r1}| |v_{r2}|)$$

$c_{r(\cdot)}$  are the indices of two segments or clusters, and  $v_{r(\cdot)}$  are the coordinates of the vector through the two points of the initial clusters.

The clustering starts with a number of clusters equal to the number of linear segments between consecutive output points. It then iteratively attempts to merge the closest pair of clusters until a stopping criteria is reached. The criteria to stop merging is:

$$J(t) = N_{c(t)} / (N_i - 1) + 1 / N_i \sum_{n=1}^{N_i} |y_{ij}(x_n) - y_j(x_n)|$$

$N_{c(t)}$  is the number of clusters at step  $t$ ,  $y_{ij}(x_n)$  is the linear output for input point  $x_n$ , and  $y_j(x_n)$  is the original sigmoidal output. The first term of the above relation penalizes a large number of clusters, while the second term penalizes a large linearization error. At the beginning of clustering, 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 becomes more important in the sum, and at one point the values of  $J(t)$  go up. At that moment, clustering stops. The resulting number of clusters determines the number of linear regions for hidden neuron  $j$ .

The linear output  $y_{ij}(x_n)$  is computed as follows. Each cluster has two limiting points, and a linear segment that passes through them. The slope ( $a_{cr}$ ) and intercept ( $b_{cr}$ ) of the linear segment, which goes through the limits of  $c_r$ -th cluster, are computed. The linear output of point  $x_n$  - within the limits of cluster  $c_r$  - is  $y_{ij}(x_n) = a_{cr} x_n + b_{cr}$ .

The closest pair of clusters at each step in the algorithm is defined as follows: The distance between any two adjacent clusters  $c_{r1}$  and  $c_{r2}$  is measured as:

$$d(c_{r1}, c_{r2}) = \max(d_{\cos}(k_1, k_2) + 1/(n_{r1} + n_{r2}) \sum |y_{ij}(x_k) - y_j(x_k)|, \{k, k_1, k_2 \in c_{r1} \cap c_{r2}\})$$

$k_{(\cdot)}$  is the index of a segment defined at Step 0 of the algorithm, which is now part of either  $c_{r1}$  or  $c_{r2}$  clusters at step  $t$ . 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  $c_{r1}$  and  $c_{r2}$ . The pair of clusters with the minimum distance  $d(c_{r1}, c_{r2})$  is merged.

After each merging the value of the criterion function  $J(t)$  is re-evaluated, and if it is bigger than the value at the previous step  $J(t-1)$  the algorithm stops. The results of the algorithm are: the number of clusters -  $N_c^j$  - for the activation function of the hidden neuron  $j$ , the coordinates in the input space of the upper and lower bounds of each cluster, and the slope and intercept of each cluster.

The slope and intercept are obtained from the linear segment that goes through the limiting points of each cluster. The resulting linear segments cover all the activation values of the hidden neuron, and any two adjacent segments overlap only in one point.

The end of the second step of the linear model extraction method consists in expressing the limits in the input space, for which each linear region of a hidden neuron is active. The limits are specified as a set of linear constraints. For example, for neuron  $j$ , linear region  $r$ , the set of constraints ( $CC^r$ ) is:

$$C^{j1} = \sum_{i=1}^N w_{ji} x_i \leq M_r, \text{ and } C^{j2} = \sum_{i=1}^N w_{ji} x_i \geq m_r$$

$$x_1 \leq M_1, x_1 \geq m_1 \dots x_N \leq M_N, x_N \geq m_N$$

where  $m_r$  and  $M_r$  are the minimum and maximum values of the linear function in region  $r$ ,  $m_i$  and  $M_i$  are the limits of each input variable as they result from the clustering process.

**Extraction of the PWL models:** Once the activation function of the hidden neurons is approximated by a PWL mapping, the next step consists in finding the valid combinations of linear regions for the hidden neurons. Such a combination is given by a set of indices, where each index is the active linear region of a hidden neuron:

$$OO^p = \{r^1, r^2, \dots, r^H\}, r^p \in \{1, 2, \dots, N_c(j)\},$$

$$\text{and } p = 1 \dots N_c(I)N_c(2) \dots N_c(H),$$

$N_c(j)$  is the number of linear regions of hidden neuron  $j$ . Each combination is a region in the input space given by the intersection of the constraint sets  $CC^p = CC^{I,r^1} \cap CC^{2,r^2} \dots \cap CC^{H,r^H}$ . Valid combinations are those for which the set of constraint in  $CC^p$  defines a non-empty region in the input space.

The constraints are placed in the set  $CC^p$  in an iterative process as follows: first, the set of constraints from the first hidden neuron ( $CC^{1,r^1}$ ) is added to  $CC^p$ , then each constraint from the subsequent sets  $CC^j, j=2 \dots H$  is checked for similarity against all constraints already in  $CC^p$ . If a new constraint is similar to one already in  $CC^p$  then the intersection between them is placed in  $CC^p$ , otherwise the new constraint is added to  $CC^p$ . Two inequality constraints are similar, if they have equal coefficients in the same input variables and the same inequality type. For example,  $x_1 \leq 3.0$ , and  $x_1 \leq 2.0$  are similar, and the intersection between them is  $x_1 \leq 2.0$ . In this way the number of constraints in  $CC^p$  is minimal.

Next, the sets of constraints  $CC^p$  are checked for validity, and their limits are refined. The validity of  $CC^p$  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  $CC^p$  is non-empty, and therefore the combination is valid.

The goal of refining the limits of the constraints in each valid set  $CC^p$  is to eliminate redundancy in the constraint limits. The limits of each constraint in the set  $CC^p$  are adjusted iteratively using the linear optimizer. At each step, a constraint becomes the objective function, and a minimization or 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 returned solution is more restrictive. The adjusting procedure stops when none of the constraint limits undergoes any changes.

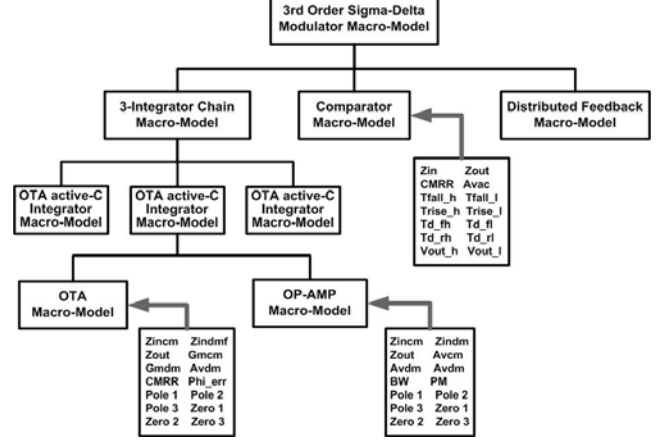
For each valid combination region defined by  $CC^p$  the output of the NN is expressed as a linear combination in the input variables:

$$y^l = a_1^l x_1 + a_2^l x_2 + \dots a_N^l x_N + a_{N+1}^l.$$

Coefficients  $a_i^l$  are functions of the weights of the NN, 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  $CC^p$  are the results of extraction.

The validity requirement (that the intersection between two sets of constraints  $CC^{p1}$  and  $CC^{p2}$  is the empty set) is always true. The

reason is that the set of constraints of each linear model ( $CC^p$ ) is obtained by intersecting the  $CC^r$  constraints for each hidden neuron. Each  $CC^r$  corresponds to a linear region of a hidden neuron. Any two  $CC^r$  of the same hidden neuron do not intersect because the  $N_c(j)$  linear regions defined by clustering do not overlap. Two constraint sets  $CC^{p1}$  and  $CC^{p2}$  do not intersect because at least one hidden neuron must be in a different region.



**Figure 5: Structural patterns in a 3<sup>rd</sup> order cascaded  $\Sigma$ - $\Delta$  ADC**

#### 4. SYSTEM SIMULATION

Figure 2(b) presents the methodology for compiled-code system simulation. Code generation is based on *symbolic composition* of the circuit macromodels depending on the structural patterns that link them together. The generated code describes the sequence of steps for calculating output, state and internal variables (i.e. voltages and currents) over time. For  $\Sigma$ - $\Delta$  ADC, the output voltage is computed, and used to find typical ADC performance figures, like SNR and DR [2]. Code is also generated to select - during simulation - the correct PWL region of non-linear devices.

The first step in the methodology identifies the structural patterns that connect the building blocks (circuits) together. The partitioning-based algorithm proposed in [3] can be employed for this step. Figure 5 shows the structural patterns found in the 3<sup>rd</sup> order cascaded  $\Sigma$ - $\Delta$  ADC. For example, 3-Integrator Chain Macromodel was obtained through composition of the three structurally identical macromodels for the OTA active-C integrator. Similar structural patterns can be identified for higher order  $\Sigma$ - $\Delta$  ADC, or for ADC of different topologies.

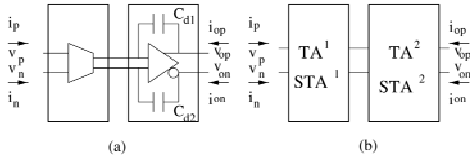
At the system-level, structural patterns compose two or more blocks having their behavior described as symbolic relationships between input, output, and state variables. Blocks are either *basic blocks* (OTA, opamp, and comparator circuits), or *composed blocks*, which correspond to previous composition steps (like integrators, ADC stages etc). Symbolic equations were formulated for each structural pattern by formulating Kirchhoff's laws for the interconnected blocks. After symbolically solving these equations, a set of mathematical expressions resulted for relating over time the unknown signals (voltages, currents, and charges) of the blocks to circuit parameters, and the known signals. Then, these expressions were encoded as C++ functions, and optimized for fast execution. In our experience, the most time effective optimizations were propagation of common expressions, and elimination of loop invariants [12]. The system model for simulation was obtained through composition of the symbolic expressions for the patterns.

**Basic blocks.** We used the OTA in Figure 3 as an example for sketching the finding of symbolic formula that link input, output, and state variables of a basic block. The OTA circuit has a 4-port model (2 ports for across voltage  $V_{idm}$  and 2 ports for across voltage  $V_{out}$ ) characterized by the symbolic expression:

$$I = TA \times V + STA,$$

$I = [I_{ip}, I_{im}, I_{op}, I_{on}]^t$  and  $V = [V_{ip}, V_{im}, V_{op}, V_{on}]^t$ . Matrix  $TA$ , called *function sub-matrix*, has its symbolic entries determined by the values of the components in the structural circuit macromodel. Matrix  $STA$ , named the *state sub-matrix*, relates to the state variables and the previous state of the circuit. For OTA, we assumed voltages  $V_i$  and  $V_o$  as *known*, and currents  $I_i$  and  $I_o$  as *unknowns*. The opposite reasoning would have been also correct. Symbols  $TA_{ij}$  and  $STA_i$  were obtained by symbolically solving the nodal equations of the OTA structural macromodel, and replacing the derivatives of state variables with their differences according to Euler Backward Integration formula [13]:  $\dot{x} = (x(t)-x(t-1))/h$ , 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 example, symbol  $TA_{11} = C_c/h + C_{cd}/(h + C_d R_d)$ , and  $TA_{31} = C_d(V_{os} - V_{cd}(t-1))/(h + C_d R_d) + C_{cm}(V_{os} - V_{cm1}(t-1))/h$ . Note that the symbol  $TA_{31}$  depends on the circuit parameters, as well as state values at the previous time moment. Symbolic function and state sub-matrices were calculated for all building blocks, and stored in the circuit library.



**Figure 6: Composition rule for the  $\Sigma$ - $\Delta$  stage**

*Composed blocks.* Symbolic composition rules (SCR) were found for each structural pattern in a system. 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 6 exemplifies the finding of the symbolic function and state sub-matrices for a  $\Sigma$ - $\Delta$  OTA Active-C Integrator Macromodel (see Figure 5). The stage consists of the OTA macrocell linked to the opamp-C macrocell through two links (Figure 6(a)). The OTA is modeled by the symbolic function sub-matrix  $TA^1_{4 \times 4}$ , and by the symbolic state sub-matrix  $STA^1_{4 \times 15}$ . There are 15 state variables in the OTA structural macromodel. The symbolic function sub-matrix  $TA^2_{4 \times 4}$  and the symbolic state sub-matrix  $STA^2_{4 \times 17}$  describe the opamp-C macrocell. The opamp-C macrocell has 17 state variables. Figure 6(b) presents that the SCR for the stage includes symbolic functional sub-matrix  $TA^c$  and state sub-matrix  $STA^c$ . For example, assuming that  $V_i$  and  $I_i$  (DAC currents) are known, and that  $I_i$  and  $V_o$  are unknown, then symbol

$$TA^c_{11} = [(TA^2_{22} - TA^1_{33})(TA^2_{11}TA^1_{11} - TA^1_{14}TA^2_{41} + TA^1_{11}TA^2_{44}) + (TA^2_{21} + TA^1_{34})(TA^2_{12}TA^1_{11} - TA^1_{13}TA^2_{41} + TA^1_{11}TA^2_{43}) + TA^1_{31} (TA^1_{13} (TA^2_{11} + TA^1_{44}) - TA^1_{14}(TA^2_{12} + TA^1_{33}))] / [(TA^2_{12} + TA^1_{43})(TA^2_{41} + TA^1_{34}) - (TA^2_{11} + TA^1_{33})(TA^2_{11} + TA^1_{44})].$$

Similar expressions describe all parameters  $TA^c_{ij}$  and  $STA^c_{ij}$ .

Once the symbolic function and state sub-matrices were calculated for each basic and composed block, C++ 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, including elements  $TA_{11}$  and  $TA_{31}$ . 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 [12]) saved significant amount of computations during the time-domain simulation process.

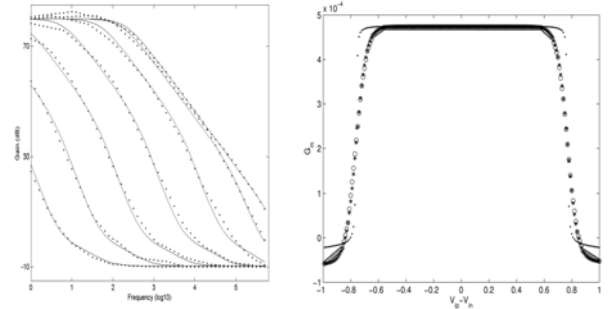
The time-domain simulation algorithm implements a loop for the time range to be simulated. The time increment is  $h$ , the parameter also used by Backward Euler Integration formula. At each time

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. The C++ code for the symbolic expressions is used. 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 through 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 most of the time. Only the parameters of the state sub-matrices must be updated for each new time step to capture the dynamics of state variables. Function sub-matrices change only if they include non-linear devices that change their current PWL region. This observation is important to speed-up simulation, 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 [12]. In our experience, this code optimization greatly reduced the total simulation time of a converter.

## 5. EXPERIMENTS

**A. PWL extraction.** For validation, the proposed PWL model extraction is applied to model the amplitude frequency response of an OTA [19] for different layout parasitic levels. The data was obtained using SPICE simulations of the analog circuit sampled in a large number of frequency and parasitic values. 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 are 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, eight weights were eliminated and one hidden neuron. The clustering method was applied to each hidden neuron. Two of the hidden neurons have 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.



**Figure 7: Results of the PWL model extraction method**

From a total of 1296 combinations of linear regions of the hidden neurons, only 136 had a non-empty solution set. For each valid combination, a linear model of the network output was computed. The result of the PWL extraction method is presented in the left part of Figure 7. The dotted plot represents the PWL model output, while the line represents the true values simulated with SPICE. Each curve corresponds to a different layout parasitics. It can be seen that the PWL approximation is very accurate. Previous approaches to extract linear models from trained NN [18, 19] (using a fixed number of segments for each hidden neuron) do not have the same accuracy. The right part of Figure 7 shows the model extraction results for the OTA  $G_m$  as a function of  $V_{idm}$  at the OTA input ports. 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 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.

**B.  $\Sigma$ - $\Delta$  ADC simulation.** Table 1 compares simulation time for Spectre simulator and the proposed symbolic method. The table relates the speed-up of the proposed method as a function of the ADC complexity (order). Results are shown for 1<sup>st</sup> to 5<sup>th</sup> order cascaded  $\Sigma$ - $\Delta$  ADC. The resulting speed-up is significant, it varies between 90 and 144 times. Please note that these speed-ups were obtained without 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 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 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.

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

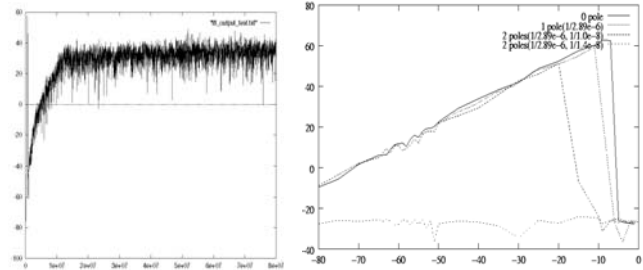
$\Sigma\Delta$ ADC order	Spectre (s)	Symbolic (s)	Speed-up
1	507.1	3.5	144.88
2	533.9	5.88	90.79
3	852.3	8.24	103.43
4	1284.9	10.69	120.19
5	1752.0	12.91	135.70

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 8 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 models including poles and zeros, rather than ideal models. In the right part of Figure 8, 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. Existing behavioral simulation methods for  $\Sigma$ - $\Delta$  ADC (like [5] and [10]) 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. The paper focuses on  $\Sigma$ - $\Delta$  ADC as a case study. Using compiled-code simulation, the proposed technique achieves speed-ups of more than 100x as compared to Spectre simulation. Simulation accuracy is not affected. 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. ADC 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. As compared to existing behavioral simulation methods for  $\Sigma$ - $\Delta$  ADC, this technique is fully automated and uses more accurate circuit models.



**Figure 8: SNR and DR plots for  $\Sigma\Delta$  ADC**

## REFERENCES

- [1] R. Bryant *et al*, "COSMOS: A Compiled Simulator for MOS Circuits", *Proc. DAC*, 1987, pp. 9-16.
- [2] J. Cherry, W. M. Snelgrove, "Continuous-Time Delta-Sigma Modulators for High-Speed A/D Conversion", *Kluwer*, 2000.
- [3] Doboli *et al*, "A Regularity-based Hierarchical Symbolic Analysis Method for Large-scale Analog Networks", *IEEE Trans. Circuits & Systems- II*, No 11, pp. 1054-1068, 2001.
- [4] S. Doboli, A. Doboli, "Piecewise-Linear Modeling of Analog Circuits using Trained Feed-Forward Neural Networks and Adaptive Clustering of Hidden Neurons", *Joint Conf. Neural Networks*, 2003.
- [5] K. Franken, G. Gielen, "A High-Level Simulation and Synthesis Environment for  $\Delta\Sigma$  Modulators", *IEEE Trans. CAD*, No. 8, pp. 1049-1061, 2003.
- [6] G. Gielen *et al*, "Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits", *Proc. of IEEE*, Vol. 88, Dec. 2000.
- [7] G. J. Gomez *et al*, "A Generic Parameterizable CMOS OTA Macromodel", *IEEE Trans. Circuits & Systems-I*, 1995.
- [8] D. Lewis, "A Hierarchical Compiled Code Event-Driven Logic Simulator", *IEEE Trans. CADICS*, Vol. 10, pp. 726-737. 1991.
- [9] P. Maurer, "Event Driven Simulation Without Loops or Conditions", *Proc. ICCAD*, 2000.
- [10] F. Medeiro, "Top-Down Design of High Performance Sigma-Delta Modulators", *Kluwer*, 1998.
- [11] A. Moscovici, "High Speed A/D converters - understanding Data Converters through SPICE", *Kluwer*, 1999.
- [12] S. Muchnik, "Advanced Compiler Design and Implementation", *Morgan Kaufmann*, 1997.
- [13] J. Vlach, K. Singhal, "Computer methods for circuit analysis and design", *New York: Van Nostrand Reinhold*, 1983.
- [14] IEEE Trans. on CADICS, Special Issue on Behavioral Modeling and Simulation of Mixed-Signal/Mixed-Technology Circuits and Systems, Vol. 22, February 2003.
- [15] K. Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks", pp. 183-190, 1988.
- [16] D. Leenaerts, W. Van Bokhoven, "Piecewise-Linear Modeling and Simulation", *Kluwer*, 1998.
- [17] 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
- [18] R. Setiono *et al*, "Extraction of rules from artificial neural networks for nonlinear regression", *IEEE Trans. Neural Networks*, 13(3), pp. 564-577, 2002.
- [19] S. Doboli, G. Gonthoskar, A. Doboli, "Modeling of Analog Circuits based on Model Extraction from Trained Neural Networks", *Proc. of IEEE Intl. Workshop on Behavioral Modeling and Simulation*, 2002