Hierarchical Approach to Exact Symbolic Analysis of Large Analog Circuits *

Sheldon X.-D. Tan
Department of Electrical Engineering
University of California
Riverside, CA, 92521
stan@ee.ucr.edu

Weikun Guo
Department of Electrical Engineering
University of California
Riverside, CA, 92521
wguo@ee.ucr.edu

Zhenyu Qi
Department of Electrical Engineering
University of California
Riverside, CA, 92521
zhenyu@ee.ucr.edu

ABSTRACT

This paper provides a novel approach to exact symbolic analysis of very large analog circuits. The new method is based on determinant decision diagrams (DDDs) to represent symbolic product terms. But instead of constructing DDD graphs directly from a flat circuit matrix, the new method constructs DDD graphs in a hierarchical way based on hierarchically defined circuit structures. The resulting algorithm can analyze much larger analog circuits exactly than before. Theoretically, we show that exact symbolic expressions of a circuit are cancellation-free expressions when the circuit is analyzed hierarchically. Practically we propose a novel hierarchical DDD graph construction algorithm. Our experimental results show that very large analog circuits, which can't be analyzed exactly before like $\mu A725$ and other unstructured circuits up to 100 nodes, can be analyzed by the new approach for the first time. The new approach significantly improves the exact symbolic capacity and promises huge potentials for the new applications of symbolic analysis in analog circuit design automation.

Categories and Subject Descriptors

T5.2 [**Analog, mixed-signal, MEMS and/or RF design tools**]

General Terms

Algorithms, Design

Keywords

Behavioral Modeling, Symbolic Analysis, Circuit Simulation

1. INTRODUCTION

Symbolic analysis is to calculate the behavior or the characteristics of a circuit in terms of symbolic parameters. It can be used to derive parametric behavioral models of analog modules. As illustrated in [2], simple yet accurate symbolic expressions can also be interpretable by analog designers to gain the insight into circuit behavior, performance and stability, and are important for verification and synthesis applications in analog circuit design automation [3]. Symbolic analysis, however, suffers well-known circuit-size limitation problem as the number of product terms increases exponentially with the complexity of a circuit. Traditional symbolic analyzer can only analyze analog circuits with about 10 nodes [2]. With the introduction of the determinant decision diagram concept, symbolic approaches based on DDD graph have improved the exact symbolic analysis capacity, but the sizes of typical unstructured analog modules can be analyzed exactly are still less than 25-30 nodes in general [8].

In addition to the DDD-graph based approaches, modern symbolic analyzers employ the hierarchical decomposition [1] scheme to cope with very large analog blocks. Hierarchical decomposition is to generate symbolic expressions in the *sequence-of-expression* forms [4,9]. There are three methods known as topological analysis [9], network formulation [4] and DDD-based approach [12]. The first two methods are based on the symbolic Gaussian node elimination concept to obtain the circuit transfer functions in sequence-of-expression forms, the last one performs subcircuit reduction and generates *hierarchical DDD-graphs*, which can be viewed as a special form of sequence of expressions as some DDD nodes themselves represent symbolic expressions (DDD graphs) coming from subcircuits. The major drawback for all those hierarchical based approaches is that the symbolic manipulations, which are critical for behavioral modeling and approximations, other than evaluation, of the resulting sequences of expressions are known to be difficult and complicated [5], and often require dedicated efforts, e.g., sensitivity calculation in [6] and lazy approximation in [7].

In this paper, we present a novel approach to exact symbolic analysis based on DDD graphs. Our new approach is based on the observation that the exact symbolic expressions of a circuit are cancellation-free expressions during the hierarchical decomposition process of the same circuit [10]. We propose to perform a novel symbolic de-cancellation, which equivalently leads to hierarchical construction of flat DDD graphs. With flat DDD graphs, all the symbolic manipulations such as computing the s-expanded polynomials, finding dominant terms via shortest path search, cofactoring and sensitivity computation can be performed easily. Experimental results show that mesh-structured RC filter circuits with up to 100 nodes and tree-like structured analog circuits with more than 1000 nodes can be exactly analyzed for the first time. This promises huge potentials for the exact symbolic analysis and high accurate behavioral modeling of many analog blocks, whose sizes are typically less than 100 nodes.

2. HIERARCHICAL CIRCUIT ANALYSIS AND SYMBOLIC CANCELLATION

Consider a subcircuit with some internal structures and terminals. The circuit unknowns—the node-voltage variables and branch-current variables—can be partitioned into three disjoint groups x^I, x^B, and x^R, where the superscripts *I, B, R* stand for, respectively, *internal* variables, *boundary* variables and the *rest* of variables. With

*The work is supported by UC Senate Research Funds and Cadence Design Systems Inc.

DAC 2004, June 7–11, 2004, San Diego, California, USA.

this, the system-equation set $\mathbf{Ax} = \mathbf{b}$, can be rewritten in the following form (Schur decomposition):

$$\begin{bmatrix} A^{II} & A^{IB} & 0 \\ A^{BI} & A^{BB} & A^{BR} \\ 0 & A^{RB} & A^{RR} \end{bmatrix} \begin{bmatrix} x^I \\ x^B \\ x^R \end{bmatrix} = \begin{bmatrix} b^I \\ b^B \\ b^R \end{bmatrix}. \quad (1)$$

The matrix, A^{II}, is the *internal* matrix associated with internal variable vector x^I.

Subcircuit suppression (Schur Decomposition) is to eliminate all the variables in x^I, and to transform (1) into the following reduced set of equations :

$$\begin{bmatrix} A^{BB} - A^{BI}(A^{II})^{-1}A^{IB} & A^{BR} \\ A^{RB} & A^{RR} \end{bmatrix} \begin{bmatrix} x^B \\ x^R \end{bmatrix} = \begin{bmatrix} b^B - A^{BI}(A^{II})^{-1}b^I \\ b^R \end{bmatrix}, \quad (2)$$

Suppose that the number of internal variables is m, and the number of boundary variables is l. Then each matrix elements in the modified A^{BB} and b^B can be written in the following expanded forms:

$$a^{BB*}_{u,v} = a^{BB}_{u,v} - \frac{1}{det(A^{II})} \sum_{k_1,k_2=1}^{m} a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} a^{IB}_{k_2,v}, \quad (3)$$

where $u,v = 1,...,l$ and $a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} a^{IB}_{k_2,v}/det(A^{II})$ is called *composite admittance* due to MNA formulation in the sequel, and

$$b^{B*}_{u} = b^{B}_{u} - \frac{1}{det(\mathbf{A}^{II})} \sum_{k_1,k_2=1}^{m} a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} b^{I}_{k_2}, \quad (4)$$

where $u = 1,...,l$ and $\Delta_{u,v}$ is the first-order *cofactor* of $det(\mathbf{A})$ with respect to $a_{u,v}$.

Notice that the modified submatrix A^{BB} is a full $l \times l$ matrix. The composite admittances $a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} a^{IB}_{k_2,v}/det(A^{II})$ in the modified A^{BB} submatrix will generate symbolic cancellations, if they are multiplied with other composite admittances when the reduced matrix is further reduced or final transfer functions are computed [10, 11]. Specifically, two kinds of symbolic cancellations will happen: *symbolic term* cancellation, where two product terms consisting of the composite admittances cancel out (sum equals to zero); *symbolic common-factor* cancellation, where the numerator and the denominator of the resulting product terms consisting of the composite admittances will have a symbolic common factor.

If there are at least two first-order cofactors exist in a product term, common-factor cancellation happens when all row and column indices of first-order cofactors in a product term are unique [11]. For common-factor cancellation, $det(A^{II})^m$, $m \geq 1$ will become a common factor if there are $m+1$ first-order cofactors in the involved product term [10]. Otherwise, term cancellation will happen where the product term will always cancel out with another product term. We call this *term-cancellation rule* in the sequel.

A general s-domain reduction algorithm was proposed in [10] where common-factor de-cancellation is carried out numerically in s-domain. In this paper, we show that both term and common-factor cancellations can be removed symbolically. Such de-cancellation process eventually leads to the hierarchical construction of exact DDD graphs.

3. THE HIERARCHICAL DDD GRAPH CONSTRUCTION ALGORITHM

In this section, we first show that cancellation-free expressions in the hierarchical circuit decomposition are the exact symbolic expressions obtained from the original flattened circuit, which lays the foundation for our hierarchical construction algorithm.

3.1 Cancellation-Free Expressions

We first have the following theoretical results without proof.

THEOREM 1. *For the reduced matrix shown in Eq.(2), the numerator of a cancellation-free symbolic rational expression of the determinant of the reduced matrix is the exact the symbolic expression obtained from the determinant of the original matrix.*

In short, if we can manage to remove the cancellations during the symbolic Schur decomposition, we can obtain the exact symbolic expressions from the original circuit matrix. This leads to our hierarchical DDD construction algorithm shown in the next subsection.

3.2 Symbolic De-cancellation Strategy

The basic idea is to construct cancellation-free DDDs for one subcircuit at a time from the hierarchically defined DDD structures. We call such a process *expansion* of a subcircuit. The construction proceeds until all the subcircuits are expanded.

For any product term in the reduced matrix in Eq.(2), if the number of first-order cofactors are larger than one, both term and common-factor cancellation may happen. According to the term-cancellation rule [11], if any two first-order cofactors share the same row or column indices in the subcircuit matrix A^{II}, this term will cancel out with another term. Otherwise, this product (along with other product terms) will generate a common factor between the numerator and the denominator.

This suggests an efficient symbolic de-cancellation strategy based determinant decision diagrams [8]. We illustrate the symbolic de-cancellation process using the following example. Consider a 5×5 matrix shown below:

$$\begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} \begin{bmatrix} a_{11} & 0 & x_{13} & 0 & 0 \\ 0 & a_{22} & 0 & x_{24} & 0 \\ x_{31} & 0 & a & 0 & 0 \\ 0 & x_{42} & 0 & b & e \\ 0 & 0 & 0 & f & d \end{bmatrix} \quad (5)$$

After reducing variables v_1 and v_2, we have a reduced 3×3 matrix:

$$\begin{matrix} v_3 \\ v_4 \\ v_5 \end{matrix} \begin{bmatrix} a+t_1 & 0 & \\ 0 & b+t_2 & e \\ 0 & f & d \end{bmatrix} \quad (6)$$

where $t_1 = x_{13}\Delta_{11}x_{31}/det(A^{II})$, $t_2 = x_{24}\Delta_{22}x_{42}/det(A^{II})$ and $det(A^{II}) = a_{11}a_{22}$. The corresponding DDD representation of the determinant of the reduced matrix is shown in Fig. 1

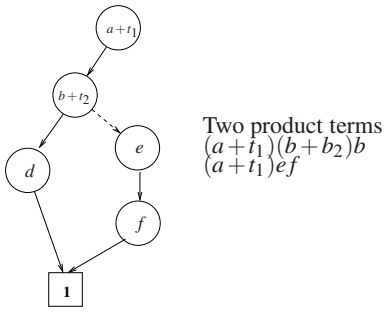

Figure 1: The DDD graph for a 3×3 determinant.

The idea of de-cancellation is to build a number of *partial* DDD (*PDDD*) graphs from one DDD graph such that each of the partial DDD graph is cancellation-free with respect to one given subcircuit. For this propose, we need to remove all the nodes which have composite admittances from the give subcircuit. Since the depth of some DDD graphs will be reduced, we have to find the missing DDD nodes in the reduced subcircuits and associated boundary nodes. All the missing DDD node for one PDDD graph will become another DDD graph again, which is called complementary DDD (*compDDD*) graph with respect to the current PDDD graph. The de-cancellation process is better illustrated by Fig. 2.

In this figure, we construct four cancellation-free PDDD graphs out of the original DDD graph (subfigure (a)-(d)). Each of the

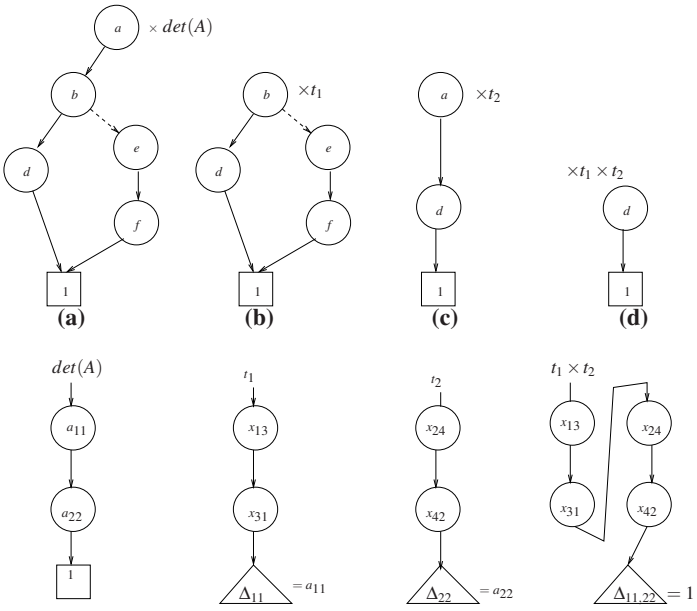

Figure 2: The illustration of partial DDD graphs and complementary DDD graphs.

PDDD graph has a unique complementary DDD graph, which is also displayed at the bottom part of each subfigure. If we multiply each PDDD graph with its compDDD graph and add (union operation) all the resulting DDD graphs together, the resulting DDD graph, which have six product terms, will be the exact DDD graph that we construct directly from matrix in Eq.(5).

So the critical issues left is how to construct the PDDDs and their compDDDs from a given DDD graph. The construction process actually is the symbolic de-cancellation process. Specifically, we need to consider the following two situations: (a) If all the first-order cofactors have unique row and column indices, then common-factor cancellation will take place. As a result, the compDDD will consist of a higher-order cofactor predicted by Theorem (2), as shown in the $t_1 \times t_2$ case in Fig. 2 (subgraph (d)). The higher-order cofactor is the cofactor obtained from A^{II} when all the rows and columns of the boundary admittances (like x_{ij} in the Fig. 2) are removed. (b) If there are two first-order cofactors sharing the same row or column indices in A^{II}, term cancellation will happen. That means that the PDDD and its corresponding compDDD should be removed. In summary, we only keep compDDD, whose first-order cofactors in any path in the compDDD have unique row and column indices.

We have the following theoretical result regarding the maximum number of PDDDs (compDDDs) constructed from a DDD graph without proof.

THEOREM 2. *If the size of* $\mathbf{A}^{II}$ *is m and size of* $\mathbf{A}^{BB}$ *is l, then the maximum number of PDDD graphs is bounded by* $\sum_{i=0}^{p}(C_{l,i})^2(C_{m,i})^2$, *where* $C_{n,k} = \frac{n!}{k!(n-k)!}$ *and* $p = min(l,m)$.

Theorem 2 tells us that the number of PDDDs can be limited by either number of boundary nodes or internal nodes of a subcircuit. So for very strongly coupled circuits, we can chose smaller subcircuit sizes to control the number of PDDDs.

3.3 The Hierarchical Construction Algorithm

Both PDDDs and their compDDDs can be constructed by traversing the given DDD graph in a bottom fashion. We *expand* one subcircuit at a time from the top to the bottom in a breath first search (BFS) order until all the subcircuits are expanded. For each PDDD (structure), we maintain four index sets: the row index set (*PRowSet*) and the column index set (*PColSet*) for the DDD node in the partial DDD graph and the row index set (*CRowSet*) and the column index set (*CColSet*) for the DDD node in the compDDD graph in submatrix A^{II}. We also keep track of two DDD graphs in each PDDD structure: *PartDDD*, which is the partial DDD graph for the PDDD and *compDDD*, which is the complementary DDD graph for this PDDD. A PDDD list will be created at each DDD node when we visit each DDD in a bottom up way. The whole hierarchical construction algorithm is shown in Fig. 3.

```
HIERCONST (P, subckt)
For all the subckts, visit each subckt in a BFS order {
  D_res = P;
  for subckt i, do {
    build its internal determinant det(A_i^II);
    D_res = CONSTPDDDR(D_res, subckt);
    Multiply each PDDD with its compDDDs;
    Add all the resulting DDD graphs into one DDD graph;
  }
}
CONSTPDDDR(P, subckt)
if(P = one or has been visited)
  return;
if(P.index < parent_min(P))
  return;
CONSTPDDDLISTR(P.child1, subckt);
CONSTPDDDLISTR(P.child0, subckt);
L_1 = MULTIPLYPDDDLIST(P, PDDDList(P.child1), subckt);
L_res = UNIONPDDDLIST(L_1, PDDDList(P.child0));
PDDDList(P) = L_res;
MULTIPLYPDDDLIST(P, PL_1, subckt)
if (P has device admittances or composite elements
which is not created from the given subckt)
  L_res = Multiplication of P with each PDDD in the PL_1;
if (P have composite admittance which is created from
the given subckt)
  for each composite admittance (CA){
    for each PDDD T_i in PL_1, do {
      r = row of the first-order cofactor in the CA;
      c = column of the first-order cofactor in the CA;
      if (r ∉ T_i's CRowSet and c ∉ T'_i s's CColSet){
        Add r and c into T_i's CRowSet and CColSet respectively;
        Add the row and column indices of P into T_i's PRowSet
        and PColSet respectively;
      } else {
        Remove the T_i from PL_1;
      }
    }
  }
```

Figure 3: Hierarchical DDD construction algorithm

UNIONPDDDLIST(L_1, PDDDList(*P.child*0)) performs the union of two lists of PDDDs. Two PDDDs can be unioned, if their PRowSet, PColSet, CRowSet and CColSet are same. DDD graph Union operations are done for both its PartDDD and compDDD graph also.

The complete traversal of the given DDD graph rooted at P actually can be avoided if we know where the composite admittances from a given subcircuit are located in the resulting D_{res}. For a subcircuit, we notice that its composite admittances are created in its parent circuits. If we know the minimum DDD index in its parent circuits, we can stop the further search when the DDD index just visited is smaller than the minimum DDD index [1]. This reflects the statement – *if(P.index < parent_min(P)) return* – in function CONSTPDDDR(). In this case, the resulting PDDD list has just one PDDD and its PartDDD is just the original DDD graph rooted at the current DDD node.

4. EXPERIMENTAL RESULTS

The proposed algorithm has been implemented. A number of analog circuits ranging from large Opamp circuits, active filter to

[1]In DDD implementation, ordering is enforced such that parent's index is always larger its children indices.

Table 1: Construction statistics on the different analog circuits.

Circuit	#nodes	$\|DDD\|$	#path	#IntNode	CPU Time(hier)	CPU time(flat)
CMOS_Opamp	14	8803/1891	79643	5	2.22	0.44
$\mu A741$	24	2314/4985	108032	10	0.26	0.61
$\mu A725$	32	11373	1.21×10^8	5	4.22	–
act_fliter	34	68362	5.10×10^7	-	8.43	–
p5x20x2	100	20402	5.53×10^{20}	10	2.91	–
p5x20x3	100	112913	7.3×10^{20}	10	40.6	–
p5x20x4	100	392326	2.10×10^{21}	10	202.63	–
4bit_bus	133	442605	3.81×10^{11}	13	46.03	–
p10x100	1000	67840	N/A	50	13.72	–
p10x150	1500	276340	N/A	50	73.66	–

mesh-structured RC filter circuits and magnetically coupled RLCM bus circuits have been simulated. All results are collected on a Linux workstation with dual 1.6Ghz AMD Athlon MP 2000+ CPUs and 1GB memory.

We first test our program on a number of small analog circuits for which the non-hierarchical method can be still used for the construction. Both algorithms give the same DDDs in terms of number terms, but different DDD sizes. Table 1 summarizes the results. In this table, $|DDD|$ is the number of DDD nodes for denominator of a transfer function. If two numbers $n1/n2$ are given, $n1$ is the DDD size using the hierarchical method, $n2$ is the DDD size using the flat method. $\#path$ is the number of product terms in the denominator of the transfer function and $\#IntNode$ is the number of internal nodes allowed for any subcircuit during the clustering. $CPUtime(hier)$ gives the CPU time using the hierarchical construction method and $CPUtime(flat)$ gives the CPU time using flat DDD construction method. In general, we find out that the new construction method will leads to larger DDD size than non-hierarchical method in general. This is a not surprise as ordering is done for each subcircuit separately. Although in the subcircuit level, it follows the minimum degree ordering.

Then we test a number of circuits that can't be analyzed by non-hierarchical method. The most noticeable one is $\mu A725$ Opamp circuit, which has 32 nodes as shown in Fig. 4. By using the new approach, it take less than 5 seconds to finish the construction. The number of internal nodes for each subcircuit is set to 5. This is the first time that analog circuits like this size has been exactly analyzed symbolically. Then, we perform the hierarchical DDD

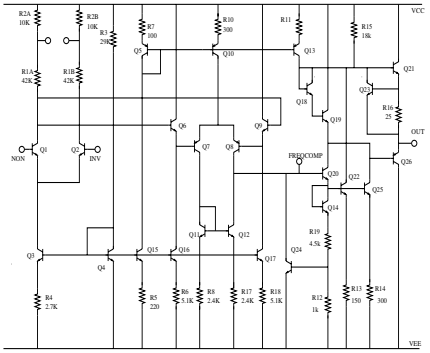

Figure 4: The $\mu A725$ operational amplifier circuit.

construction on a number of mesh-structured RC circuits. Circuits $p5x20x*$ have 5 rows and each row is a RC ladder circuit with 20 RC segments. The last digit is the number of the vertical lines for connecting the 5 ladder circuits at evenly distributed points. From Table 1, as the number of vertical lines increases, so do the number of product terms and construction time.

For tree-like structured circuits $p10x100$ and $p10x150$, which have 1000 nodes, the new algorithm constructs the DDD graphs in very short time. Actually we can analyze almost arbitrarily large tree-structured circuits due to their localized nature. N/A here means the number of paths is out of numerical range of floating point numbers in the computer.

We notice also that as the construction process continues, the DDD sizes grow larger, it will slow down the construction process as in the worse case we need to visit very node once in the existing DDD graph to build a new one for each subcircuit. So the time complexity of the construction process depends on the final DDD sizes and the hierarchical structure of the circuit. The actual sizes of circuits can be analyzed are subject to the structure of the circuits, the computer memory and the efficiency of DDD graph operations (with or without using garbage collection to improve memory efficiency).

5. CONCLUSION

In this paper, we have proposed a novel approach to exact symbolic analysis of very large analog circuits. We proposed novel hierarchical DDD graph construction algorithm based on symbolic de-cancellation process. Theoretically, we show that exact symbolic expressions of a circuit are cancellation-free expressions in the hierarchical decomposition of the same circuit. Practically, we developed a novel symbolic de-cancellation algorithm for hierarchical DDD graph constructions. Our experimental results show that very large analog circuits, which can't be analyzed exactly before like $\mu A725$ and some large mesh-structured RC filter circuits up to 100 nodes and tree-like structured circuits up to 1000 nodes, can analyzed by the new approach for the first time. The new approach has substantially improved exact symbolic simulation capacity.

6. REFERENCES

[1] F. V. Fernández and A. Rodríguez-Vázquez, "Symbolic analysis tools—the state of the art," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, 1996, pp. 798–801.

[2] G. Gielen and W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits*. Kluwer Academic Publishers, 1991.

[3] G. Gielen, P. Wambacq, and W. Sansen, "Symbolic analysis methods and applications for analog circuits: A tutorial overview," *Proc. of IEEE*, vol. 82, no. 2, pp. 287–304, Feb. 1994.

[4] M. M. Hassoun and P. M. Lin, "A hierarchical network approach to symbolic analysis of large scale networks," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 4, pp. 201–211, April 1995.

[5] P. M. Lin, *Symbolic Network Analysis*. Elsevier Science Publishers B.V., 1991.

[6] ——, "Sensitivity analysis of large linear networks using symbolic program," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, 1992, pp. 1145–1148.

[7] S. J. Seda, M. G. R. Degrauwe, and W. Fichtner, "Lazy-expansion symbolic expression approximation in synap," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, Nov. 1992, pp. 310–317.

[8] C.-J. Shi and X.-D. Tan, "Canonical symbolic analysis of large analog circuits with determinant decision diagrams," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 1, pp. 1–18, Jan. 2000.

[9] J. A. Starzky and A. Konczykowska, "Flowgraph analysis of large electronic networks," *IEEE Trans. on Circuits and Systems*, vol. 33, no. 3, pp. 302–315, March 1986.

[10] S. X.-D. Tan, "A general s-domain hierarchical network reduction algorithm," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, 2003, pp. 650–657.

[11] S. X.-D. Tan, Z. Qi, and H. Li, "Hierarchical modeling and simulation of large analog circuits," in *Proc. European Design and Test Conf. (DATE)*, 2004.

[12] X.-D. Tan and C.-J. Shi, "Hierarchical symbolic analysis of large analog circuits via determinant decision diagrams," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 4, pp. 401–412, April 2000.