HIERARCHICAL SYMBOLIC ANALYSIS OF LARGE ANALOG CIRCUITS WITH DETERMINANT DECISION DIAGRAMS

Xiangdong Tan and C.-J. Richard Shi

Department of Electrical and Computer Engineering
University of Iowa, Iowa City, Iowa 52242, U.S.A.
Email: {xtan,cjshi}@eng.uiowa.edu

ABSTRACT

A new hierarchical approach is proposed to symbolic analysis of large analog circuits. The key idea is to use a graph-based representation, called Determinant Decision Diagram (DDD), to represent the symbolic determinant and cofactors of the MNA matrix for each subcircuit block. By exploiting the inherent sharing and sparsity of symbolic expressions, DDD is capable of representing a huge number of symbolic product terms in a canonical and highly-compact manner. Further, it enables cofactoring and sensitivity computation to be performed with time linear in the size of DDD. Experimental results have demonstrated that our method outperforms the best-known hierarchical symbolic analyzer SCAPP and even numerical simulator SPICE for small-signal AC analysis.

1. INTRODUCTION

Symbolic analysis is to calculate the behavior or the characteristic of a circuit in terms of symbolic parameters. It is important for many applications such as optimum topology selection, design space exploration, behavioral model generation, and fault detection [2]. However, symbolic analysis has not been widely used by analog designers. The root of the difficulty is apparently: the number of product terms in a symbolic expression may increase exponentially with the size of a circuit. Any manipulation and evaluation of symbolic expressions will require CPU time at best linear in the number of terms, and therefore have both the time and space complexities exponential in the size of a circuit.

One way to cope with the circuit-size limitation problem of symbolic analysis is by means of hierarchical decomposition. Hierarchical decomposition is to generate symbolic expressions in a nested form [3, 7]. There are two methods known as topological analysis method [7] and network approach [3]. Both are based on the *sequence-of-expressions* concept to obtain the transfer functions. Unfortunately, the number of expressions can grow rapidly so that even the compilation of the generated expressions could take enormous time. Manipulation (other than evaluation) of the resulting sequences of expressions is known to be complicated and often requires dedicated efforts, e.g., sensitivity calculation in [4] and lazy approximation in [6].

In this paper, we present a new hierarchical approach to exact symbolic analysis. It takes advantage of both hierarchical decomposition and a recently introduced graph-based representation for symbolic determinants called Determinant Decision Diagram [5]. By exploiting the *sparsity* and *sharing* among expressions, DDD can store a symbolic determinant compactly. For example, 1.01×10^8 symbolic product terms can be represented by a diagram with only 767 vertices. More importantlly, manipulations such as cofactoring and sensitivity can be performed in linear time in the size of DDD.

The rest of the paper is organized as follows: Section 2 provides an overview of the hierarchical symbolic analysis procedure. Sections 3 and 4 describe the DDD-based approach. Section 5 presents experimental results. Section 6 concludes the paper.

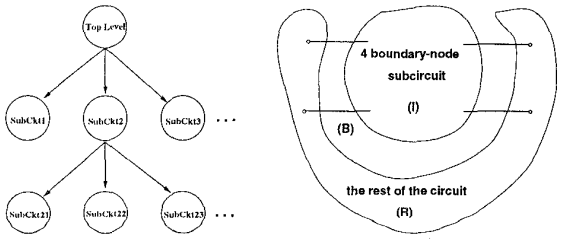
This work is sponsored by U.S. Defense Advanced Research Projects Agency (DARPA) under grant number F33615-96-1-5601 from the United States Air Force, Wright Laboratory, Manufacturing Technology Directorate.

2. OVERVIEW OF HIERARCHICAL ANALYSIS

For a linear(ized) time-invariant analog circuit, its system equation can be formulated by, for example, the modified nodal analysis (MNA) approach in the following general form [9]:

$$\begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v} \\ \mathbf{i} \end{bmatrix} = \begin{bmatrix} \mathbf{j} \\ \mathbf{e} \end{bmatrix}, \qquad (1)$$

where $\mathbf{v}$ is the vector of node voltage variables, $\mathbf{i}$ is the vector of the branch current variables, $\mathbf{A}$ is the modified nodal admittance matrix, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are contributions of the branch relationship, $\mathbf{j}$ represents the external current sources and $\mathbf{e}$ represents the independent voltage sources. We assume the presence of the predefined subcir-



(a) Circuit hierarchy. (b) Partition of a subcircuit.

Figure 1: Model of hierarchical analysis.

cuits in the circuit hierarchy. The circuit hierarchy can be viewed as a rooted tree shown in Fig. 1(a). A circuit may have one or more subcircuits at each hierarchical level. Consider a subcircuit with some internal structure and terminals, as illustrated in Fig. 1(b). The circuit unknowns—the node-voltage variables $\mathbf{v}$ and branch-current variables $\mathbf{i}$ — can be partitioned into three disjoint groups

0-7803-4455-3/98/$10.00 © 1998 IEEE

$\mathbf{x}^I$, $\mathbf{x}^B$, and $\mathbf{x}^R$, where the sup-scripts *I, B, R* stand for, respectively, internal variables, boundary variables and the *rest* of variables. *Internal* variables are those local to the subcircuit, *boundary* variables (also called *tearing variables*) are those related to both the subcircuit and the rest of the circuit. Note that boundary variables also include those variables required as outputs. Then, the system-equation set (1) can be rewritten in the following form:

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

The basic idea that underlines all hierarchical analysis methods is to eliminate the number of equations and the number of variables from the equation-set above until a set of equations involving only the desired variables remains. The physical meaning of such elimination is to eliminate subcircuits internal to all input/output nodes in a circuit. So we call this *subcircuit elimination.* The resulting set of equations can be written as follows:

$$\begin{bmatrix} \mathbf{A}^{BB*} & \mathbf{A}^{BR} \\ \mathbf{A}^{RB} & \mathbf{A}^{RR} \end{bmatrix} \begin{bmatrix} \mathbf{x}^B \\ \mathbf{x}^R \end{bmatrix} = \begin{bmatrix} \mathbf{b}^{B*} \\ \mathbf{b}^R \end{bmatrix}, \quad (3)$$

where

$$\mathbf{A}^{BB*} = \mathbf{A}^{BB} - \mathbf{A}^{BI}(\mathbf{A}^{II})^{-1}\mathbf{A}^{IB}, \quad (4)$$

and

$$\mathbf{b}^{B*} = \mathbf{b}^B - \mathbf{A}^{BI}(\mathbf{A}^{II})^{-1}\mathbf{b}^I. \quad (5)$$

Subcircuit elimination can be performed for all the subcircuits by visiting the circuit hierarchy in a bottom-up fashion. Hierarchical *numerical* analysis performs subcircuit elimination numerically by partial LU decomposition [8]. Hierarchical *symbolic* analysis is to introduce intermediate variables to represent subcircuit elimination (4) and (5) using a *sequence of expressions* [3, 7]. However, expressions resulting for $\mathbf{A}^{BI}(\mathbf{A}^{II})^{-1}\mathbf{A}^{IB}$ and $\mathbf{A}^{BI}(\mathbf{A}^{II})^{-1}\mathbf{b}^I$ are usually very complicated, and the size of such expressions may grow rapidly even the compilation of the generated expressions could take a very long time.

3. BASIC IDEA

Let $\mathbf{A}$ be an $n \times n$ matrix. It may be denoted as $[a_{u,v}], u, v = 1, ..., n$. The *determinant* of matrix $\mathbf{A}$ is denoted by $det(\mathbf{A})$. According to linear algebra, the inverse of matrix $\mathbf{A}$ can be written as

$$\mathbf{A}^{-1} = \frac{1}{det(\mathbf{A})}[\Delta_{u,v}], \quad (6)$$

where

$$\Delta_{u,v} = (-1)^{u+v} det(\mathbf{A}_{a_{u,v}}).$$

and $[\Delta_{u,v}]$ is called the *adjoint* matrix of $\mathbf{A}$. $\Delta_{u,v}$ is the first-order *cofactor* of $det(\mathbf{A})$ with respect to $a_{u,v}$, $\mathbf{A}_{a_{u,v}}$ is the $(n-1) \times (n-1)$-matrix obtained from the matrix $\mathbf{A}$ by deleting row u and column v. Note that each entry in the adjoint matrix is a cofactor of the original matrix, and thus the adjoint matrix is a *full* (dense) matrix.

Suppose that the number of internal variables is m, and the number of boundary variables is n. For practical circuits, n usually is small given a good partitioning. More importantly, $\mathbf{b}^I$ is a zero vector, $\mathbf{A}^{BI}$ and $\mathbf{A}^{IB}$ are very *sparse* submatrices. This implys that only a few of the first-order cofactors of A^{II} are needed. Applying (6) to (4), we have the expanded form of (4):

$$a^{BB*}_{u,v} = a^{BB}_{u,v} - \frac{1}{det(\mathbf{A}^{II})} \sum_{k_1,k_2=1}^{m} a^{BI}_{u,k_1} \Delta^{II}_{k_2,k_1} a^{IB}_{k_2,v}, u, v = 1, ..., n. \quad (7)$$

Note that we need first-order cofactors $\Delta^{II}_{k_2,k_1}$ only when a_{u,k_1} and $a_{k_2,v}$ are both non-zeros, and at same time a_{u,k_1} and $a_{k_2,v}$ are zero for most time due to sparsity of A^{BI} and A^{IB}. So the key problem of hierarchical symbolic analysis is how to represent symbolically the determinant $det(\mathbf{A}^{II})$ and *a few* of its first-order cofactors in a compact and efficient manner. We will show in the next section that this problem can be solved by using a recently-introduced graph representation called Determinant Decision Diagrams.

4. DETERMINANT DECISION DIAGRAMS FOR HIERARCHICAL SYMBOLIC ANALYSIS

Determinant Decision Diagrams is a canonical and compact graph-based representation of symbolic-matrix determinants [5]. It has enabled the exact symbolic analysis of such circuits like μA-741 operational amplifiers for the first time [5].

Formally, a determinant decision diagram *DDD* is a signed rooted directed acyclic graph with two terminal vertices, namely the 0-terminal vertex and the 1-terminal vertex. It has two outgoing edges, called 1-edge and 0-edge, pointing, respectively, to D_{a_i} and $D_{\overline{a}_i}$. A determinant graph having root vertex a_i denotes a matrix determinant D defined recursively as follows:

1. if a_i is the 1-terminal vertex, then $D = 1$,
2. if a_i is the 0-terminal vertex, then $D = 0$,
3. Otherwise(non-terminal vertex), $D = a_i \cdot s(a_i) \cdot D_{a_i} + D_{\overline{a}_i}$,

where $s(a_i)$ is a sign $\{+, -\}$ associated with each non-terminal vertex a_i. It can be determined recursively as follows.

1. Let $P(v)$ be the set of DDD vertices that originate the 1-edges in any path rooted at v to the 1-terminal. Then

$$s(v) = \prod_{x \in P(v)} sign(r(x) - r(v))\, sign(c(x) - c(v)), \quad (8)$$

   where $r(x)$ and $c(x)$ refer to the row and column indices of vertex x in the matrix, and $sign(u) = 1$ if $u > 0$, and $sign(u) = -1$ if $u < 0$.
2. If v has an edge pointing to the 1-terminal vertex, then $s(v) = +1$.

We have shown that DDD is a graph representation of the expansion of matrix determinant $det(\mathbf{A})$ in a way similar to binary decision diagrams (BDDs) for Shannon expansion of Boolean functions. Each vertex in a DDD represents a matrix expansion:

$$det(\mathbf{A}) = a_{r,c}(-1)^{r+c} det(\mathbf{A}_{a_{r,c}}) + det(\mathbf{A}_{\overline{a}_{r,c}}). \quad (9)$$

where $det(\mathbf{A})$ is the determinant represented by this vertex, the vertex pointed by its 1-edge represents its cofactor $(-1)^{r+c} det(\mathbf{A}_{a_{r,c}})$ with respect to $a_{r,c}$, and the vertex pointed by its 0-edge represents $det(\mathbf{A}_{\overline{a}_{r,c}})$, which is the *reminder* of $det(\mathbf{A})$ with respect to $a_{r,c}$.

For example, consider the following determinant

$$det(\mathbf{M}) = \begin{vmatrix} a & b & 0 & 0 \\ c & d & e & 0 \\ 0 & f & g & h \\ 0 & 0 & i & j \end{vmatrix} = adgj - adhi - aefj - bcgj + bchi. \quad (10)$$

Figure 2 illustrates the corresponding DDD representation under the expansion order: $a, c, b, d, f, e, g, i, h$ and j. Symbolic expressions represented by each vertex are also given near the vertices in the figure. In DDDs, each path from root vertex (a in our case) to 1-terminal is called *1-path*. Each 1-path uniquely define a product term which includes all vertices (symbols) from which the 1-edges in the 1-path originate. We note that subterms ad, gj, and hi appear in several product terms of the matrix determinant, and they are shared in the DDD representation.

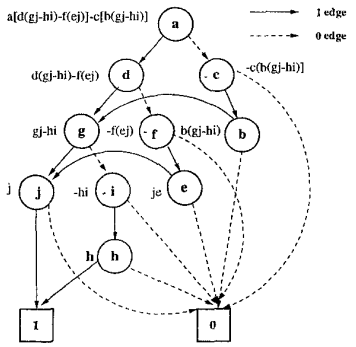


Figure 2: A determinant decision diagram for matrix **M**.

A key issue is that how to find a suitable expansion order such that the DDD has as few vertices as possible. This is called DDD vertex ordering. An efficient heuristic has been proposed in [5], which gives the optimal order for ladder-structured circuits and usually good order for practical analog circuits.

5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The proposed method has been implemented in a symbolic analyzer that can read the circuit description in the SPICE format. The analysis is performed by depth-first traversal of the circuit hierarchical tree shown in Fig. 1(a). At each node, circuit equation set is built and its subcircuits are suppressed using equation (7). Then the analysis is moved upwardly. At the top level of the circuit hierarchy, symbolic transfer function is derived using Cramer's rule and all the symbolic determinants and cofactors generated are represented by DDDs.

The results from two examples are presented. The first example is an active low-pass filter circuit shown in Fig. 3. It has four identical subcircuits, named $X1$ to $X4$. Figure 4 shows the detailed structure of a subcircuit. Each subcircuit contains two Opamps. We have tested our program on two different implementation of Opamps: a linear model of 741 Opamp circuit shown in Fig. 5(a) and a miller-compensated two-stage Opamp circuit shown in Fig. 5(b). For the miller-compensated Opamp circuit, all the MOS transistors are replaced by their corresponding small-signal models at the DC operating point computed by SPICE. The AC analysis is performed by depth-first traversals of all the DDD vertices used to represent all symbolic expressions at each frequency point. The numerical value of the determinant is obtained when its root is reached. Since each DDD vertex only needs one visit for each frequency point, the time complexity of the DDD-based numerical evaluation is directly proportional to the DDD size.

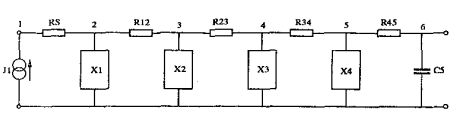


Figure 3: An active low-pass filter.

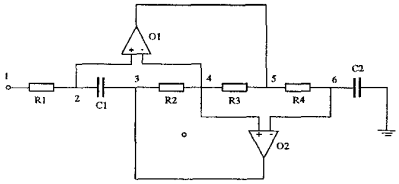
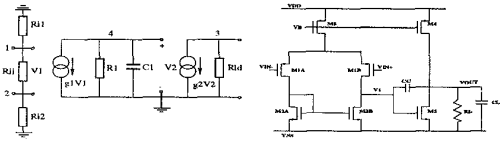


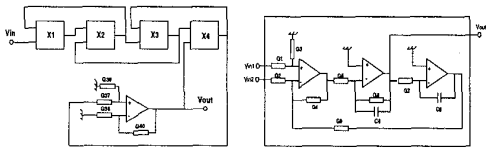
Figure 4: An FDNR subcircuit.



(a) Linear model of 741 Opamp.

(b) Miller-compensated two-stage Opamp.

Figure 5: Two implementations of an Opamp.



(a) The partitioned RC band-pass filter.

(b) The subcircuit in band-pass filter.

Figure 6: An active RC band-pass filter

The second example is a band-pass filter circuit (Fig. 6(a)) which was widely used for hierarchical analysis [3, 7]. It consists of four topologically identical subcircuits $X1$ to $X4$ shown in Fig. 6(b). For each Opamp in the subcircuits, we also test two implementations in Fig. 5(a) and Fig. 5(b), respectively. We have conducted two sets of experiments on a SUN SPARCstation 5 with 32M memory. We first compare our program with SPICE on repetitive numerical evaluation. For each circuit, 1000 frequency points are simulated. The results are summarized in Table 1, where *#term* is the actual number of distinct product terms generated, $|DDD|$ is the number of DDD vertices used to represent all the symbolic expressions. Columns 4, 5, and 6 list, respectively, the CPU time in

seconds used by the proposed DDD-based method, SPICE, and the speedup of the proposed method over SPICE for each test circuit.

From Table 1, we can see that the proposed DDD-based method

Table 1: Comparison against SPICE in numerical evaluation.

circuit	#term	|DDD|	DDD	Spice	Speedup
LP(linear)	23	43	1.07	6.58	6.15
LP(miller)	67	58	1.56	14.82	9.50
BP(linear)	562	228	3.07	9.70	3.16
BP(miller)	622	243	3.72	17.76	4.77

LP is for low-pass filter and BP is for band-pass filter.
linear or *miller* denotes the corresponding Opamp model used.

outperforms SPICE for all the test cases. Further, the speedup increases with the size of the circuit.

We compare our method with SCAPP—a best-known hierarchical symbolic analyzer [3]. We construct the test circuits by cascading, respectively, the first 1, 2, 3 and 4 subcircuit blocks(Fig. 6(a)). The Opamp subcircuit is implemented by the miller-compensated Opamp circuit. SCAPP uses the flatten circuit description, and exploits an automatic optimized partitioning strategy[1].

The results are summarized in Table 2. Columns 1 and 2 list, respectively, the number of subcircuits cascaded for each test case, and the total number of node voltage and branch current variables in the flatten circuits. Columns 3 and 4 describe the number of DDD vertices, and the number of product terms represented. Columns 5 and 6 give the numbers of additions and multiplications used in the expressions generated by SCAPP. Note that the number of additions and multiplications used by the proposed DDD-based method is exactly the number of DDD vertices. From Table 2,

Table 2: Comparison against SCAPP.

#subckt	#var	DDD-based		SCAPP	
		#vertices	#terms	#add	#mul
1	23	146	500	556	274
2	39	369	2.60×10^4	1339	854
3	55	568	1.60×10^6	1772	1074
4	71	767	1.01×10^8	2479	1639

we can observe that the DDD-based representation is much more compact than the sequence-of-expressions representation used in SCAPP. Note that the number of DDD vertices is about 2-4 times less than the number of expressions in SCAPP, and the storage of each expression takes much more space than that for one DDD vertex. We also see that the DDD size grows almost linearly in the circuit size, despite that the number of product terms grow exponentially.

Table 3 shows the statistics of using the DDD-based symbolic method and SCAPP for repetitive numerical evaluation. For SCAPP, we compile the generated symbolic expressions and then execute the code for simulation. For each test case, we report in Columns 2 and 3 the CPU time required to construct the DDD and then the CPU time taken for simulation from the constructed DDD. For SCAPP, we report respectively in Columns 4, 5, and 6 the CPU time for SCAPP to generate the sequence-of-expressions (*analy*), the compilation time (*comp*), and the actual simulation time (*sim*). The last two columns give the matrix setup time and simulation time used by SPICE. We can see from Table 3 that the proposed DDD-based method is more efficient than both SCAPP and SPICE. Although it may take less time for SCAPP to obtain the sequence of expressions, it takes a much longer time to compile

[1] We have not exploited optimal partitioning yet.

Table 3: Comparison against SCAPP and SPICE in CPU time.

#subckt	DDD-based		SCAPP			SPICE	
	const.	sim.	analy.	comp.	sim.	setup	sim.
1	0.37	2.09	0.81	13.1	2.60	1.10	5.34
2	1.01	4.75	2.09	33.3	7.49	2.70	8.98
3	2.42	6.91	3.69	44.2	10.37	3.12	15.58
4	12.75	9.19	5.54	64.7	12.06	3.42	22.10

the generated expressions. We also note that our program is interpreted, while SCAPP is compiled. We have not exploited optimal partitioning, while SCAPP already did. Therefore, we expect that the compiled and optimized version of our DDD-based method could offer even more improvement over SCAPP.

6. CONCLUSIONS

A new hierarchical method for symbolic analysis of large analog circuit has been presented and implemented. Experimental results have shown that the proposed method compares very favorably with the best-known symbolic analyzer SCAPP and numerical simulator SPICE for small-signal AC analysis.

Acknowledgment: The authors wish to thank Prof. G. Gielen of KUL for several helpful discussions on symbolic analysis and Prof. M. Hassoun of Iowa State University for making SCAPP code available to us.

7. REFERENCES

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

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

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

[4] P. M. Lin, "Sensitivity analysis of large linear networks using symbolic program", pp. 1145–1148 in *Proc. IEEE Int. Symp. Circuits and Systems*, 1992.

[5] C. J. Richard Shi and X. Tan, "Symbolic analysis of large analog circuits with determinant decision diagrams", in *Proc. IEEE Int. Conf. Computer Aided Design (ICCAD)*,Nov., 1997.

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

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

[8] M. Vlach, "LU decomposition algorithms for parallel and vector computation", pp. 37–64 in *Analog Methods for Computer-Aided Circuit Analysis and Diagnosis*, T. Ozawa (ed.), Marcel Dekker, New York, 1988.

[9] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design*, Van Nostrand Reinhold, New York, 1994.