Compact Representation and Efficient Generation of s-Expanded Symbolic Network Functions for Computer-Aided Analog Circuit Design

C.-J. Richard Shi, *Senior Member, IEEE,* and Xiang-Dong Tan, *Member, IEEE*

***Abstract*—A graph-based approach is presented for the generation of exact symbolic network functions in the form of rational polynomials of the complex frequency variable s for analog integrated circuits. The approach employs determinant decision diagrams (DDDs) to represent the determinant of a circuit matrix and its cofactors. A notion of *multiroot DDDs* is introduced, where each root represents a symbolic expression for an individual coefficient of the powers of s in the numerator and denominator of a network function, and multiple roots share their common subgraphs. A DDD-based algorithm is presented for generating s-expanded network functions. We prove theoretically and validate experimentally that the algorithm constructs in $O(kl|\mathrm{DDD}|)$ time an s-expanded DDD with no more than $kl|\mathrm{DDD}|$ vertices, where k is the degree of the denominator s polynomial, l is the maximum number of devices that connect to a circuit node, and $|\mathrm{DDD}|$ is the number of DDD vertices representing the circuit-matrix determinant. For a practical circuit, $|\mathrm{DDD}|$ is often many orders-of-magnitude less than the number of product terms. In contrast, previous approaches require the time and space complexities proportional to the number of product terms, which grows exponentially with the size of a circuit. Experimental results have demonstrated that the new approach can produce exact s-expanded-symbolic network functions for μA741 operational amplifiers in several CPU seconds on an UltraSparc-I workstation. The expressive power of multiroot s-expanded DDDs is so remarkable that in one instance, over 10^{35} symbolic product terms have been represented by a multiroot DDD with less than 17 K vertices. The compactness of DDDs is further demonstrated in the context of symbolic noise evaluation, where potentially many transfer functions, each being used for a noise source in the circuit, can be represented by a single DDD with the size comparable to that for a few transfer functions. This provides a powerful tool for solving many symbolic analysis problems such as deriving interpretable symbolic expressions, dominant pole/zero estimation, and analog testability analysis. We have also demonstrated that repetitive numerical evaluation with the derived s-expanded symbolic expressions for frequency-domain simulation and small-signal noise analysis can be much faster than SPICE-like simulators and the resulting expressions for a circuit block can be used as behavioral models for high-level simulation.**

Manuscript received December 22, 1999; revised October 31, 2000. This work was supported by the Defense Advanced Research Projects Agency, United States Air Force, Wright Laboratory, Manufacturing Technology Directorate, under Grant F33615-96-1-5601, by a National Science Foundation CAREER Award, and by Conexant Systems. This paper was presented in part at the IEEE Custom Integrated Circuits Conference, San Diego, CA, May 1998, and the Asia and South Pacific Design Automation Conference, Tokyo, Japan, January 2000. This paper was recommended by Associate Editor K. Mayaram.

C.-J. R. Shi is with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA (e-mail: cjshi@ee.washington.edu).

X.-D. Tan was with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA. He is now with Monterey Design Systems, Sunnyvale, CA 94089 USA.

Publisher Item Identifier S 0278-0070(01)04826-6.

***Index Terms*—Analog circuits, circuit design, determinant decision diagrams, network functions, noise evaluation, poles/zeros, small-signal analysis, symbolic circuit analysis.**

I. Introduction

IN THIS PAPER, we consider the problem of automatic generation of symbolic network functions for analog integrated circuits. The particular form of network functions we are interested in is rational polynomials in the complex frequency variable s, or simply *s-expanded,* written as

$$H(s) = \frac{F(s)}{G(s)} = \frac{\sum f_i(p_1, p_2, \ldots, p_m)s^i}{\sum g_i(p_1, p_2, \ldots, p_m)s^i} \tag{1}$$

where $F(s)$ and $G(s)$ are the numerator s polynomial and the denominator s polynomial, respectively, and coefficients $f_i(p_1, p_2, \ldots, p_m)$ and $g_i(p_1, p_2, \ldots, p_m)$ are symbolic functions in terms of symbolic circuit parameters $p_1, p_2, \ldots, p_m$ and do not contain the complex frequency variable s.

Network functions characterize the small-signal behavior of analog integrated circuits. Many circuit characteristics such as gains, input/output impedances, poles/zeros, common-mode rejection ratios (CMRRs), power-supply rejection ratio (PSRRs), and noise figures can be computed from network functions. Second-order effects such as distortion and nonlinearity can be estimated based on network functions of linearized circuits. To fully explore the power of symbolic circuit analysis in helping designers gain insight into the circuit behavior and in repetitive numerical evaluation, network functions in the s-expanded symbolic form are often required. Several examples are as follows.

1) Deriving interpretable symbolic expressions by approximation requires the expansion of a network function into the s-expanded form so that symbolic coefficients of s polynomial terms are equally approximated; otherwise, the resulting expressions may not be reliable [15], [17].
2) For circuits such as operational amplifiers, designers are interested in the first few dominant poles and zeros. Since dominant poles and zeros are well separated from other poles and zeros in a circuit, their expressions can be approximated as the ratios of the symbolic coefficients of two consecutive powers of s [13], [15].
3) Testability of linear analog circuits can be analyzed symbolically based on the sensitivities of individual coefficients of power s with respect to circuit parameters [7].

Symbolic testability analysis avoids inevitable round-off errors introduced in numerical testability analysis [21].

4) Even some large-signal behaviors, e.g., Elmore delay and other delay metrics for interconnect modeling of digital very large scale integrated (VLSI) circuits are related to the coefficients of the first few low-order s terms [9], [28], [35].
5) s-expanded symbolic network functions may be advantageous for repetitive numerical evaluation. For example, in the extreme case that s is the only symbolic variable, evaluation of an s polynomial for frequency-domain simulation can be much faster than solving repeatedly a set of circuit equations.

Despite the importance of s-expanded symbolic network functions, no efficient approaches exist for the automatic generation of such functions from a given circuit description. Traditional symbolic analysis techniques, either graph-based such as the signal-flow graph method [22] and tree enumeration method [8] or algebra-based such as determinant expansion [17] and parameter extraction [22] generate the denominators and numerators of network functions as the sum of complex product terms, where each *complex* product term is a product of several s polynomials, e.g., $(R_1 + sC_1)$ $(R_2 + sC_2)$ $(R_3 + sC_3)$. Not only may the number of complex product terms grow exponentially with the size of a circuit, but the expansion of one complex product into an s polynomial may lead to an exponential number of s-expanded terms, e.g., $R_1R_2R_3$ $+(R_1R_2C_3$ $+R_2R_3C_1$ $+R_1R_3C_2)s$ $+(R_1C_2C_3$ $+R_2C_1C_3$ $+R_3C_1C_2)s^2$ $+C_1C_2C_3s^3$. Therefore, traditional techniques are only applicable to very small circuits with a few nodes and devices.

Recently, various simplification schemes have been developed to find approximate symbolic expressions [13], [14], [17], [37]; however, it is well known that approximate expressions may not be adequate for complete circuit characterization such as symbolic pole-zero derivation and sensitivity computation [18]. Arbitrarily nested hierarchical expressions or sequences of expressions can be useful for circuit simulation [20], [32], but no efficient method exists to convert them into the s-expanded form. The interpolation method [36] can handle the case that all the circuit parameters are numbers and s is the only symbolic variable. In practice, it may suffer from numerical accuracy problems.

In this paper, a new approach is presented for the representation and generation of exact s-expanded symbolic network functions for analog integrated circuits. Based on Cramer's rule for solving systems of linear equations, the approach describes network functions in terms of the determinant and cofactors of a circuit matrix. We first exploit a shared multiroot determinant decision diagram (DDD) [29], [30] to represent the determinant and all the cofactors used to describe a network function, where each DDD-vertex label is a nonzero matrix entry. Since each matrix entry is an s polynomial under the modified nodal formulation [36] and s is implicitly contained in DDD labels, the resulting DDD is called a *complex DDD*. We then introduce a generalized notion of DDD to represent s-expanded network functions, where each root in a multiroot DDD represents the symbolic coefficient of a particular power of s and common subexpressions in the coefficients are shared in this *s-expanded DDD*. We present an efficient algorithm for constructing the s-expanded DDD from a complex DDD. We prove that the resulting s-expanded DDD has less than $kl|\mathrm{DDD}|$ vertices and can be constructed in time $O(kl|\mathrm{DDD}|)$, where k is the degree of the denominator s polynomial of a network function, l is the maximum number of devices that connect to a circuit node, and $|\mathrm{DDD}|$ is the number of vertices in the complex DDD. For practical circuits, $|\mathrm{DDD}|$ can be many orders of magnitude less than the number of complex product terms represented by a DDD. In an extreme case, for a circuit with an i-section ladder structure, $|\mathrm{DDD}|$ is $3i-2$, where the number of complex product terms is the $(i+1)$th Fibonacci number (exponential in i). Therefore, the proposed approach offers a significant improvement over previous approaches based on product-term generation.

We have implemented the proposed approach. Experimental results demonstrate that our program can produce exact s-expanded symbolic network functions for μA741 operational amplifiers in several CPU seconds on an UltraSparc-I workstation. We show that the use of generated symbolic expressions for frequency-domain simulation can achieve a significant speedup over SPICE [25].[1] We demonstrate that symbolic expressions for dominant pole/zeros can be estimated from s-expanded network functions.

We further apply DDD-based symbolic analysis to circuit-noise modeling and simulation. Noise behavior is an important characteristic of analog circuits, as it usually determines the fundamental limit of the circuit or system performance. Numerical noise analysis for analog circuits in direct current (dc) steady state can be carried out efficiently by using the adjoint method [27]. However, for system-level noise simulation, the adjoint method may not offer adequate efficiency and does not provide much insight into how noise is affected by circuit parameters. We observe that noise analysis amounts to computing a set of symbolic transfer functions, which share most subexpressions. Therefore, we propose to use a single multiroot DDD to represent all the symbolic transfer functions required for noise analysis. Experimental results from real analog circuits show that the computational cost for obtaining a number of transfer functions required by noise evaluation is comparable to that for computing a single transfer function. This leads to an efficient method of deriving symbolic noise expressions. Furthermore, we show that for common analog circuit blocks such as operational amplifiers, the resulting noise expressions can be used as noise models for higher level noise evaluation.

Some preliminary results of this paper have appeared in [31], [34]. The rest of the paper is organized as follows: after reviewing the concept of DDDs in Section II, we describe in Section III how all the symbolic coefficients of the power of s of a rational s polynomial can be elegantly represented by a multiroot DDD, called an s-expanded DDD. An efficient algorithm is presented in Section IV for deriving the s-expanded DDD from an original complex DDD. Section V describes several applications of s-expanded symbolic network functions. Section VI describes experimental results and we conclude in Section VII.

[1]SPICE3f5 from the University of California, Berkeley was used in this paper.

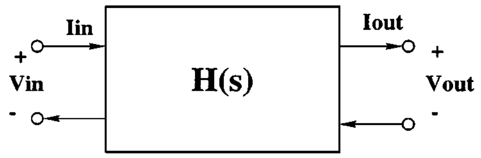

Fig. 1. Two-port network.

II. Preliminaries—Linear(ized) Circuit Analysis via DDD

Our approach is based on the theory of DDD for linear(ized) circuit analysis [29], [30]. In this section, we briefly review the basic DDD concepts used in this work. We assume that the reader is familiar with the fundamentals of linear algebra [6] and circuit analysis [36].

A. Linear Circuit Analysis

A linear(ized) analog circuit can be described by a system of linear equations written in the following matrix form using, for example, the modified nodal analysis (MNA) approach [36]

$$\mathbf{T}\mathbf{x} = \mathbf{w} \tag{2}$$

where the *circuit unknown vector* $\mathbf{x}$ may be composed of node voltages and branch currents and the *circuit matrix* $\mathbf{T}$ is usually large and *sparse*.

Network functions characterize how the voltages and/or currents associated with certain outputs change with certain input voltages and/or currents. For example, in the case of two-port networks as shown in Fig. 1, four common network functions are $V_{\text{out}}/V_{\text{in}}$, $I_{\text{out}}/I_{\text{in}}$, $V_{\text{out}}/I_{\text{in}}$, and $I_{\text{out}}/V_{\text{in}}$.

Network functions can be computed from (2) by applying Cramer's rule, which states that the jth component x_j of the unknown vector $\mathbf{x}$ can be obtained as follows:

$$x_j = \frac{\sum_i w_i (-1)^{i+j} \det(\mathbf{T}_{i,j})}{\det(\mathbf{T})} \tag{3}$$

where

$\det(\mathbf{T})$	*determinant* of matrix $\mathbf{T}$;
matrix $\mathbf{T}_{i,j}$	matrix $\mathbf{T}$ after removing row i and column j;
$(-1)^{i+j}\det(\mathbf{T}_{i,j})$	*first-order cofactor* of $\det(\mathbf{T})$ with respect to element $t_{i,j}$;
$t_{i,j}$	matrix entry at row i and column j.

If i and j are single digits, $\mathbf{T}_{i,j}$ will be simply written as $\mathbf{T}_{ij}$.

For example, consider a simple circuit shown in Fig. 2. Its system equations can be written as follows:

$$\begin{bmatrix} \frac{1}{R_1} + sC_1 + \frac{1}{R_2} & -\frac{1}{R_2} & 0 \\ -\frac{1}{R_2} & \frac{1}{R_2} + sC_2 + \frac{1}{R_3} & -\frac{1}{R_3} \\ 0 & -\frac{1}{R_3} & \frac{1}{R_3} + sC_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} I_{\text{in}} \\ 0 \\ 0 \end{bmatrix}.$$

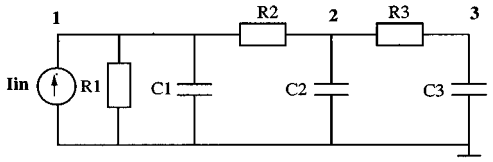

Fig. 2. Example circuit.

Let us denote this circuit matrix as $\mathbf{Y}$ and represent each matrix entry by a distinct symbol as follows: $A = (1/R_1) + sC_1 + (1/R_2)$, $B = -(1/R_2)$, $C = -(1/R_2)$, $D = (1/R_2) + sC_2 + (1/R_3)$, $E = -(1/R_3)$, $F = -(1/R_3)$, and $G = (1/R_3) + sC_3$. Then the input impedance can be written as follows:

$$R_{\text{in}} = \frac{v_1}{I_{\text{in}}} = \frac{\det(\mathbf{Y}_{11})}{\det(\mathbf{Y})} = \frac{DG - FE}{ADG - AFE - CBG}. \tag{4}$$

B. DDDs

Note that the denominators and numerators of network functions are composed of the determinant and cofactors of the circuit matrix $\mathbf{T}$. The root of the difficulty faced by traditional techniques for generating symbolic expressions of network functions is that the number of product terms in a matrix determinant, regardless of whether generated topologically via signal-flow graphs [22], tree enumeration [8], or algebraically via matrix-determinant expansion [17], is inherently exponential in the size of a matrix. Inspired by the success of binary decision diagrams (BDDs) for logic synthesis and formal verification [4], we introduced an implicit yet canonical graph representation called DDDs for the matrix determinants and cofactors [29], [30]. Similar to BDDs for Boolean functions, DDDs are capable of representing the determinants and cofactors resulting from practical circuit analysis with the number of vertices orders-of-magnitude less than that of product terms. Moreover, most symbolic analysis algorithms can be performed on a DDD with time complexity linear in the size of a DDD (i.e., the number of DDD vertices).

Formally, a DDD is a signed, rooted, directed acyclic graph with two terminal vertices, namely, the *zero-terminal* vertex and the *one-terminal* vertex. Each nonterminal vertex D is *labeled* by a symbol denoted by $D.label$ and a positive or a negative sign denoted by $D.sign$. It *originates* two outgoing edges, called *one-edge* (represented by a solid arrowed line in Fig. 3) and *zero-edge* (represented by a dotted arrowed line in Fig. 3) pointing to its two children $D.child1$ and $D.child0$, respectively. Each vertex D *represents* a symbolic expression $D.expr$ defined recursively as follows.

1) If D is the one-terminal vertex, then $D.expr = 1$.
2) If D is the zero-terminal vertex, then $D.expr = 0$.
3) If D is a nonterminal vertex, then $D.expr = D.sign * D.label * (D.child1).expr + (D.child0).expr$

where $(D.child1).expr$ and $(D.child0).expr$ represent symbolic expressions represented by the vertices $D.child1$ and $D.child0$, respectively. For example, it can be verified that Fig. 3 is a DDD representation of $\det(\mathbf{Y})$.

DDDs were originally introduced to represent symbolic matrix determinants [29], [30], where each vertex symbol is a nonzero matrix entry. Vertex D with matrix entry $t_{i,j}$ as its

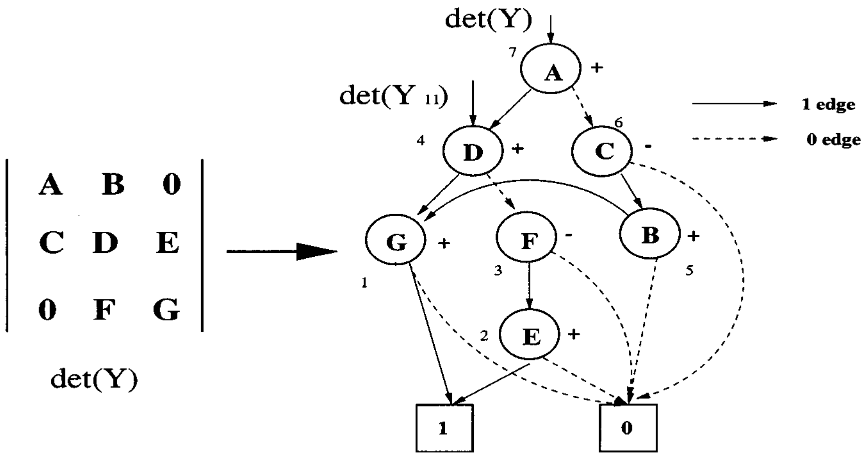

Fig. 3. Example of matrix determinant and its DDD representation.

label $D.label$ is a graphical representation of the following expansion of the determinant $\det(\mathbf{T})$ represented by D:

$$\det(\mathbf{T}) = t_{i,j}(-1)^{i+j}\det(\mathbf{T}_{i,j}) + \det(\mathbf{T}_{\overline{i,j}}) \tag{5}$$

where

$D.sign$	$(-1)^{i+j}$;
$D.child1$	$\det(\mathbf{T}_{i,j})$, which is the *minor* of $\det(\mathbf{T})$ with respect to $t_{i,j}$;
$D.child0$	$\det(\mathbf{T}_{\overline{i,j}})$, which is the *remainder* of $\det(\mathbf{T})$ with respect to $t_{i,j}$.

Matrix $\mathbf{T}_{\overline{i,j}}$ can be obtained from matrix $\mathbf{T}$ by setting entry $t_{i,j}$ to zero. We note that the DDD definition introduced in this paper can be used to represent symbolic expressions that may not necessarily correspond to matrix determinants. This observation is important for the introduction of s-expanded DDDs in Section III.

Given a vertex, a *one-path* is a path from the vertex to the one-terminal. A one-path represents a product of symbols that are labels of those vertices that *originate* all the one-edges along the one-path. For example, in Fig. 3, there exist three one-paths from the root vertex labeled by A, which represent three product terms: ADG, $A(-F)E$, and $(-C)BG$. It can be verified that the root vertex labeled by A represents the sum of these product terms.

DDDs can be viewed as an extension of Akers' BDDs for Boolean functions [1]. What makes the notion of DDD really useful are the four construction s inspired by the work of Bryant and Minato: 1) *zero-suppression* (Minato [23]): eliminate all the vertices whose one-edges point to the zero-terminal vertex and use the subgraphs of the zero-edges; 2) *uniqueness*: each label will appear no more than once in any one-path of the graph; 3) *ordered* (Bryant [3]): for all the one-paths, each label, if appears, will be in the same fixed order with respect to the other labels in that path; and 4) *shared* (Bryant [3]): all equivalent subgraphs are shared.

To facilitate the enforcement of the four rules above and the manipulation of a DDD, vertex label $D.label$ is implemented in a DDD as an index $D.index$ [5]. Further, indices are chosen to be consecutive integer numbers starting from one that satisfy the following requirement:

$$D.index > (D.child1).index$$

and

$$D.index > (D.child0).index. \tag{6}$$

The process of assigning an index to a DDD vertex label is called *vertex ordering*. A greedy vertex ordering heuristic that yields near minimal DDDs for representing the determinants of sparse matrices was suggested in [29], [30]. For example, the DDD in Fig. 3 is constructed with an optimal vertex order A, C, B, D, F, E, and G. Shown near each vertex is its corresponding integer index.

From Cramer's rule (3), a network function can be described in terms of the determinant and some (first-order) cofactors of a circuit matrix. To represent a network function, we propose to construct the DDDs for the determinant and cofactors using the same vertex order with all the common subgraphs shared. This leads to one *shared DDD with multiple roots*, where each root represents either the determinant or a cofactor of the circuit matrix. For example, Fig. 3 is actually a shared DDD representation of $\det(\mathbf{Y})$ and $\det(\mathbf{Y}_{11})$ required in (4). A *root* here can be a vertex with no incoming edges (root in the graph-theoretic sense) or any vertex that represents a cofactor of interest; they are marked by new "dangling" edges in the diagram.

We note that DDD vertices are labeled by matrix entries and each entry is in general an s polynomial whose coefficients may contain symbolic parameters (e.g., $G = (1/R_3) + sC_3$). Since the complex frequency variable s is implicitly contained in vertex labels, a DDD that represents the determinant and cofactors of the circuit matrix of a dynamic circuit and uses nonzero matrix entries as its labels is referred to as a *complex* DDD [31].

III. s-Expanded Symbolic Representation

To motivate our method of deriving s-expanded symbolic network functions, we consider how to expand the symbolic ex-

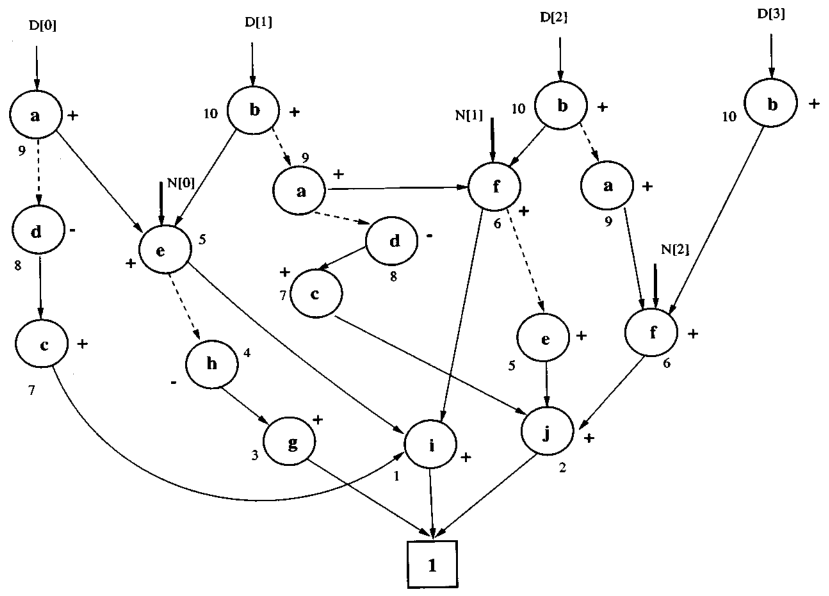

Fig. 4. s-expanded DDD.

pression in (4) to the s-expanded form. We rewrite the circuit matrix so that each symbolic entry is in the s-expanded form

$$\begin{bmatrix} a+bs & c & 0 \\ d & e+fs & g \\ 0 & h & i+js \end{bmatrix} \tag{7}$$

where

$$a=\frac{1}{R_1}+\frac{1}{R_2},\quad b=C_1,$$
$$c=d=-\frac{1}{R_2},\quad e=\frac{1}{R_2}+\frac{1}{R_3},$$
$$f=C_2,\quad g=h=-\frac{1}{R_3},\quad i=\frac{1}{R_3},$$

and

$$j=C_3.$$

Expanding the three product terms, we have

$$ADG=(a+bs)(e+fs)(i+js)\rightarrow\begin{cases} +aeis^0 & +aejs^1 \\ +afis^1 & +beis^1 \\ +afjs^2 & +bejs^2 \\ +bfis^2 & +bfjs^3 \end{cases}$$

$$AFE=(a+sb)(h)(g)\rightarrow\begin{cases} ahgs^0 \\ bhgs^1 \end{cases}$$

$$CBG=(d)(c)(i+js)\rightarrow\begin{cases} dcis^0 \\ dcjs^1 \end{cases}$$

With this, (4) can be rewritten as

$$R_{\text{in}}=\frac{\sum_{i=0}^{2}N[i]s^i}{\sum_{i=0}^{3}D[i]s^i} \tag{8}$$

with

$$\begin{aligned} D[0]&=aei-ahg-dci \\ D[1]&=aej+afi+bei-bhg-dcj \\ D[2]&=afj+bej+bfi \\ D[3]&=bfj \\ N[0]&=ei-hg \\ N[1]&=ej+fi \\ N[2]&=fj. \end{aligned}$$

Now we are ready to extend the notion of DDD to represent s-expanded symbolic network functions. Let $N[i]$ and $D[i]$ denote, respectively, the symbolic expressions of the coefficients of power s^i in the numerator and denominator of a rational polynomial function $f(s)$ of s

$$f(s)=\frac{\sum_{i}N[i]s^i}{\sum_{i}D[i]s^i}.$$

An *s-expanded DDD* is a multiroot shared DDD, where each root defines a DDD—called *coefficient DDD* [31]—that represents $N[i]$ or $D[i]$ and common subgraphs among all the coefficient DDDs are shared.

For example, Fig. 4 shows a seven-root s-expanded DDD that represents R_{in} in (8). The DDD is constructed based on the DDD definition and four construction rules with vertex ordering b, a, d, c, f, e, h, g, j, and i. We can see that this representation exploits the sharing among different coefficients in both the denominator and numerator of a rational polynomial function. In Fig. 4, 18 nonterminal vertices are used. In comparison, without exploiting the sharing, a straightforward representation of 17 product terms in both the denominator and the numerator would require 46 vertices (exploiting the sparsity) and 170 vertices (without exploiting the sparsity).

TABLE I
EXAMPLE OF VERTEX ORDERING FOR s-EXPANSION

complex-DDD-vertex index i	1		2	3	4		5	6	7	
complex-DDD-vertex label	G		E	F	D		B	C	A	
number of s-expanded symbols in each entry m_i	2		1	1	2		1	1	2	
s-expanded-DDD-vertex label $c_{i,j}s^{p_{i,j}}$	i	js	g	h	e	fs	c	d	a	bs
j	1	2	1	1	1	2	1	1	1	2
s-expanded-DDD-vertex index	1	2	3	4	5	6	7	8	9	10

IV. VERTEX ORDERING AND CONSTRUCTION OF s-EXPANDED DDDS

In this section, we present an elegant algorithm for constructing s-expanded DDDs that represent symbolic network functions. The algorithm is *implicit* in the sense that it constructs an s-expanded DDD from the complex DDD, not from the expanded symbolic expressions such as (8). The explicit approach is computationally prohibitive for a large circuit since the number of complex product terms can be exponential in the size of a circuit and further the expansion of a complex product term may lead to an exponential number of terms (e.g., *ADG* in our example leads to eight terms). In contrast, we show that our implicit algorithm constructs, in time $O(kl|\mathrm{DDD}|)$, an s-expanded DDD with no more than $kl|\mathrm{DDD}|$ vertices, where k is the degree of the denominator s polynomial of the transfer function, l is the maximum number of devices that connect to a circuit node, and $|\mathrm{DDD}|$ is the size of the complex DDD.

Similar to a complex DDD, the size of an s-expanded DDD depends crucially on the ordering of s-expanded DDD vertices. In this section, we first present such a vertex ordering heuristic s-ORDER that attempts to minimize the number of s-expanded DDD vertices. Our construction algorithm is then described in two steps: vertex-expansion s-EXPAND and s-extraction s-EXTRACT. We note that the algorithm presented originally in [31] for constructing s-expanded DDDs can apply to any vertex ordering. With the proposed vertex ordering heuristic s-ORDER, the construction described here is simpler, more elegant, and its time and space complexities become easier to analyze (Theorem 1).

A. Vertex Ordering

Let the vertices of a complex DDD be indexed from one to m. Let the symbol that corresponds to index i be v_i. We consider the general case that entry v_i in the circuit matrix, i.e., the label of a complex DDD vertex with index i, can be further represented as a sum of m_i terms

$$v_i = \sum_{j=1}^{m_i} c_{i,j}s^{p_{i,j}} \tag{9}$$

where

s	complex frequency variable;
$p_{i,j}$	integer representing the power of s;
$c_{i,j}$	symbolic coefficient of power $s^{p_{i,j}}$.

For example, for $v_1 = A = (1/R_1) + (1/R_2) + C_1 s$, we have $i = 1$, $m_i = 3$, $c_{1,1} = (1/R_1)$, $c_{1,2} = (1/R_2)$, $c_{1,3} = C_1$, $p_{1,1} = p_{1,2} = 0$, and $p_{1,3} = 1$. In general, if the MNA formulation is used, $p_{i,j}$ can be zero or one. For the applications where only s expansion is of interest, we can lump those symbolic coefficients with the same $s^{p_{i,j}}$ together. Then, (9) reduces to

$$v_i = \sum_{j=l_i}^{u_i} c_{i,j}s^j$$

where l_i and u_i denote the lowest and the highest powers of s, respectively, such that $u_i - l_i + 1 = m_i$. Note that $m_i \le l$ and l are the maximum number of devices that connect to a circuit node. For example, we can represent $A = (1/R_1) + (1/R_2) + C_1 s$ as $A = a + bs$ with $a = (1/R_1) + (1/R_2)$ and $b = C_1$. We choose to consider the more general case (9) here so that the algorithm presented in this section is directly applicable to other applications such as deriving interpretable symbolic expressions [33] and analog testability analysis [26], where certain individual circuit elements (e.g., R_1, R_2) are of interest and should not be lumped together.

To ensure the label *uniqueness* for an s-expanded DDD, it is sufficient to assign each term in (9) a unique label. Thus, a total of $m' = \sum_{i=1}^{m} m_i$ labels will be required for the s-expanded DDD. For our motivational example, ten labels a to j are used as shown in (7).

With this symbol labeling, vertex ordering for constructing s-expanded DDDs amounts to assigning an index to each term in (9). We observe that the original complex DDD was constructed with such a vertex ordering (i) that exploits as much subgraph sharing as possible. For our motivational example, indices for labels A to G as shown in Fig. 3 were chosen based on a heuristic in [29], [30] that minimizes the number of complex DDD vertices. Therefore, we would like to choose a vertex ordering for the s-expanded DDD so that the s-expanded DDD inherits as much sharing as possible from the original complex DDD. This motivates us to assign term $c_{i,j}s^{p_{i,j}}$ the following index:

$$\sum_{x=1}^{i-1} m_x + j. \tag{10}$$

We refer to the process of vertex ordering using this scheme as operation s-ORDER. Table I illustrates the use of s-ORDER for our motivational example, where rows 1 and 2 are complex-DDD indices and labels as shown in Fig. 3, rows 3 to 5 describe m_i, j, and $c_{i,j}s^{p_{i,j}}$ for each complex-DDD label and the last row is the resulting s-expanded DDD indices.

B. Operation s-EXPAND

DDD operation s-EXPAND is to replace a complex DDD vertex into a cluster of s-expanded DDD vertices. As illustrated in Fig. 5, a complex DDD vertex D with label $v_i = \sum_{j=1}^{m_i} c_{i,j}s^{p_{i,j}}$ is expanded into m_i new DDD vertices,

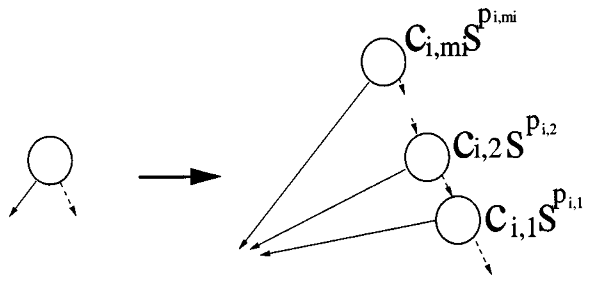

Fig. 5. Illustration of the expansion of a complex DDD vertex.

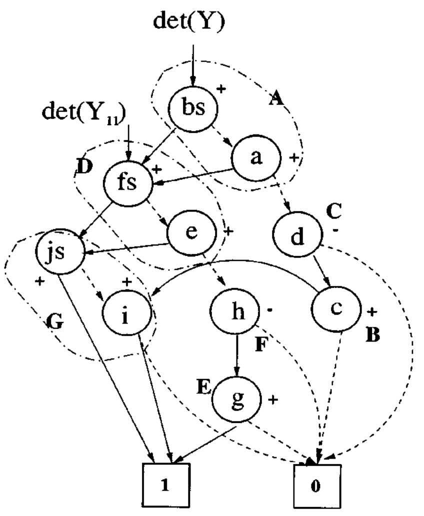

Fig. 6. Illustration of vertex expansion.

labeled by $c_{i,j}s^{p_{i,j}}$, $j = 1, \ldots, m_i$. The one-edges of all these m_i new DDD vertices point to $D.child1$. The zero-edge of DDD vertex $c_{i,j}s^{p_{i,j}}$ points to DDD vertex $c_{i,j-1}s^{p_{i,j-1}}$ and the zero-edge of the last DDD vertex $c_{i,1}s^{p_{i,1}}$ points to $D.child0$. For each new DDD vertex, its sign is the same as that of the original complex DDD vertex.

For example, Fig. 6 shows the result of applying operation s-EXPAND to the DDD in Fig. 3. It can be seen that the new DDD in Fig. 6 preserves all the sharing in the original complex DDD.

We say that DDD1 *represents* DDD2 if and only if for any vertex in DDD2, there exists a vertex in DDD1 that represents exactly the same symbolic expression.

Proposition 1: For a DDD that has |DDD| vertices and each DDD vertex label i can be expanded as the sum of m_i distinct new labels, operations s-EXPAND and s-ORDER construct a new DDD with $\sum_{i=1}^{|DDD|} m_i$ vertices that represents the original DDD.

Proof: First, it can be seen that all the new DDD vertices satisfy the following requirement:

$$D.index > (D.child1).index$$

and

$$D.index > (D.child0).index.$$

Hence, the resulting graph by applying s-EXPAND and s-ORDER satisfies DDD rules 2(uniqueness) and 3 (ordered). Followed from the original DDD, two other DDD rules—1 (zero-suppression) and 4 (shared)—are satisfied, too. Thus, the new graph is indeed a valid DDD.

Now we show that the new DDD represents the original DDD. From the definition of DDDs, the original complex DDD vertex D with index i represents the following symbolic expression:

$$D.label * D.sign * (D.child1).Expr + (D.child0).Expr.$$

As illustrated in Fig. 5, this vertex is expanded into m_i vertices D_j, $j = 1, \ldots, m_i$. The root of the expanded DDD subgraph created by operation s-EXPAND(D) defines the following symbolic expression:

$$\begin{aligned}
&(D.child0).Expr + D_1.label * D_1.sign * (D.child1).Expr \\
&\quad + D_2.label * D_2.sign * (D.child1).Expr \\
&\quad + \cdots \\
&\quad + D_{m_i}.label * D_{m_i}.sign * (D.child1).Expr \\
&\quad = (D.child0).Expr \\
&\quad + (D_1.label + D_2.label + \cdots + D_{m_i}.label) \\
&\quad * D.sign * (D.child1).Expr \\
&\quad = (D.child0).Expr + D.label * D.sign \\
&\quad * (D.child1).Expr.
\end{aligned}$$

This is exactly what the original DDD vertex D represents. □

We note that s-EXPAND invokes a local replacement operation on each complex DDD vertex and none of the original graph structure is changed. Therefore, the new DDD inherits all the sharing from the original DDD.

C. Operation s-EXTRACT

After vertex expansion, each DDD vertex D is labeled by $c_{i,j}s^{p_{i,j}}$. For the convenience of description, we refer to such a DDD as a *term DDD*. To construct s-expanded DDDs, we need to extract $s^{p_{i,j}}$ out from each vertex and keep only $c_{i,j}$ as the label. We use $D.label$ to denote $c_{i,j}$ and $D.spower$ to denote $p_{i,j}$. We also assume that the s polynomial represented by vertex D can be written in the following expanded form:

$$\sum_i P[i]s^i$$

where $P[i]$ is either a symbolic coefficient expression or zero. Suppose that we have derived s-expanded DDDs for $D.child1$ and $D.child0$; let them be denoted by $P1$ and $P0$, respectively. Then

$$P[i] = D.label * D.sign * P1[i - D.spower] + P0[i].$$

Thus, $P[i]$ can be created by a new DDD vertex with label $D.label$, sign $D.sign$, its one-edge pointing to $P1[i - D.spower]$, and its zero-edge pointing to $P0[i]$.

For example, Fig. 7 illustrates this idea for the cases that $D.spower$ can only be -1, 1, and 0. Note that under the MNA formulation, $D.spower$ is 1 or 0.

The complete algorithm s-EXTRACT is described in Fig. 8, where GETVERTEX ($D.index$, $D.sign$, $P0$, $P1$) is a DDD operation that returns a DDD vertex with index $D.index$, sign $D.sign$, $P1$ as its one-child, and $P0$ as its zero-child [29], [30]. Note that GETVERTEX ($D.index$, $D.sign$, *NULL*, *NULL*)

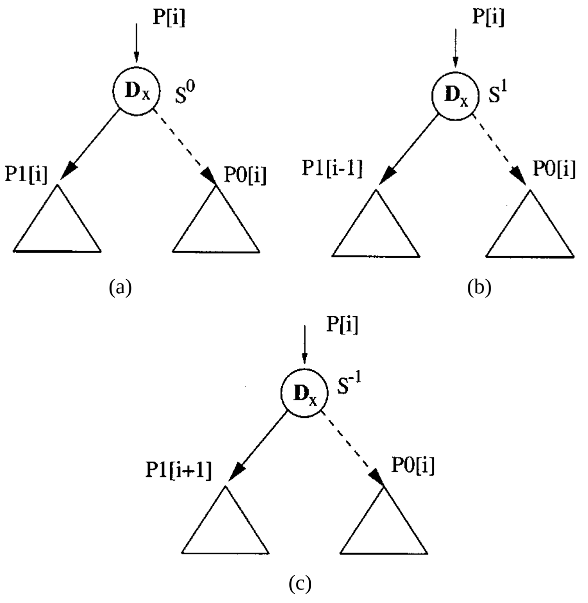

Fig. 7. Illustration of s extraction. Admittance of (a) resistance, (b) capacitance, and (c) inductance.

returns NULL, and GETVERTEX ($D.index$, $D.sign$, $P0$, *NULL*) returns $P0$ (zero-suppression). If a vertex with the same $D.index$, $D.sign$, $P0$, and $P1$ exists already, GETVERTEX simply returns that vertex; this implements subgraph sharing. To simplify the description of the algorithm, we use L and U to denote the lowest and highest powers of s in the DDD and we assume that all the vertices represent the polynomials with the same lowest and highest powers of s. In the implementation, each vertex has its actual lowest and highest powers of s; they are calculated bottom up from terminals. In Fig. 8, VERTEXZERO and VERTEXONE denote the zero-terminal and the one-terminal, respectively.

Fig. 9 illustrates the application of s-EXTRACT to the DDD in Fig. 6. Ten major steps, each resulting from applying s-EXTRACT to a DDD vertex, are shown. The procedure is executed bottom-up. For example, at level 1, vertex i is s-extracted; at level 2, vertex js is s-extracted; and at level 10, vertex bs is s-extracted. It can be seen that the s-expanded DDD shown in Fig. 9 is the same as the s-expanded DDD shown in Fig. 4 constructed directly from the expanded expressions.

We have the following result.

Proposition 2: Given a term DDD with $|\text{DDD}|$ vertices, operation s-EXTRACT constructs in time $O(k|\text{DDD}|)$ an s-expanded DDD with no more than $k|\text{DDD}|$ vertices, where k is the maximum degree of the s polynomials that the term DDD represents.

Proof: s-EXTRACT performs a depth-first search on a given DDD. Each DDD vertex will be visited just once and s-EXTRACT will be called $|\text{DDD}|$ times.

Consider each time when s-EXTRACT is called. Let the maximum power of its representing polynomial function be p_i. Then precisely p_i s-expanded vertices will be created.

The total number of s-expanded DDD vertices is thus

$$\sum_{i=1}^{|\text{DDD}|} p_i \leq \sum_{i=1}^{|\text{DDD}|} k = k|\text{DDD}|.$$

Since a hash table is used to keep track of the previous results, if we do not count the hashing time (as commonly done in the area of decision diagram research [5], [23]), the time complexity of the algorithm is, thus, $O(k|\text{DDD}|)$. □

From Propositions 1 and 2, and noting that $\sum_{i=1}^{|\text{DDD}|} m_i \leq \sum_{i=1}^{|\text{DDD}|} l = l \cdot |\text{DDD}|$, where l is the maximum number of devices that connect to a circuit node and s-ORDER takes time $\sum_{i=1}^{|\text{DDD}|} m_i$, we have the following main result.

Theorem 1: Given a complex DDD with $|\text{DDD}|$ vertices, its corresponding s-expanded DDD can be constructed by applying DDD operations s-ORDER, s-EXPAND, and s-EXTRACT in time $O(kl|\text{DDD}|)$ with no more than $kl|\text{DDD}|$ vertices, where k is the maximum degree of the s polynomial that the complex DDD represents and l is the maximum number of devices that connect to a circuit node.

We can make the following remarks on algorithm s-EXTRACT and Theorem 1.

1) For a network function, the degree of its denominator s polynomial is always no less than the degree of its numerator s polynomial. Hence, for a complex DDD that represents a network function, k is the degree of the denominator s polynomial.
2) Under the MNA formulation as far as s-expansion is concerned, we can rewrite each matrix entry in the MNA circuit matrix in one of the following three forms: a, bs, or $a+bs$, where a and b represent the symbolic coefficients of s^0 and s^1 and represent the lumped effect of those devices connected to a circuit node. We call this *compact symbol representation*. Then, m_i is bounded by two, i.e., $l=2$. For comparison, the case where each term in the MNA matrix is represented as a distinct symbol is referred to as *full symbol representation*.
3) In the extreme case that all circuit parameters are numbers and s is the only symbolic variable, algorithm s-EXTRACT provides an alternative to the numerical interpolation method [36]. In this case, operation GETVERTEX is replaced by simple numerical calculation. The algorithm takes time $O(kl|\text{DDD}|)$ with $l=2$.

V. APPLICATIONS OF s-EXPANDED SYMBOLIC NETWORK FUNCTIONS

In this section, we describe several applications of s-expanded symbolic network functions in the design of analog integrated circuits. We note that the algorithm described in previous section is sufficiently general to be applicable to many other applications. For example, by keeping those circuit-element symbols of interest (instead of merging them as for s-expansion), operations s-ORDER, s-EXPAND, and s-EXTRACT are the foundation for deriving interpretable symbolic expressions [33] and for analog-testability analysis [26].

A. Small-Signal Behavioral Modeling

An analog integrated circuit can be linearized at its operating point and its small-signal behavior can be analyzed by solving repeatedly (2) with $\mathbf{T} = \mathbf{G} + s\mathbf{C}$ and $s = 2\pi \mathbf{j} f$ [36]. This is generally a fast procedure since for each frequency point f, only one sparse LU factorization and substitution are required. How-

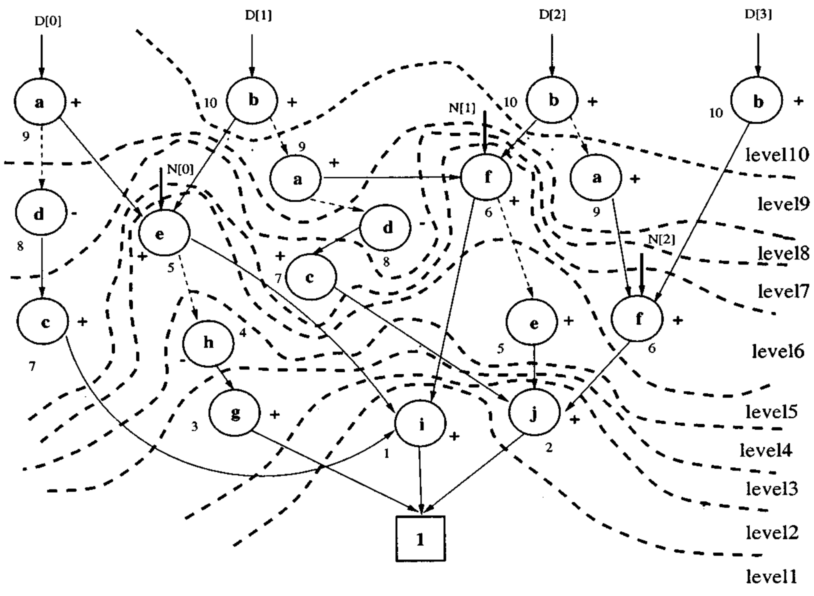

Fig. 8. Derivation of s-expanded DDDs by s extraction.

```
s-Extract(D)
1  if (D = 0)
2      for i= L to U do
3          P[i] = VertexZero
4  else if (D = 1)
5      for i= L to U do
6          P[i] = VertexZero
6      P[0] = VertexOne
7  else
8      P0 ← s-Extract(D.child0)
9      P1 ← s-Extract(D.child1)
10     for i= L to U do
11         P[i] ← GetVertex(D.index, D.sign, P0[i], P1[i − D.spower])
12 return P
```

Fig. 9. Illustration of operation s-EXTRACT.

ever, as in the context of design optimization or Monte Carlo simulation for test generation, the procedure has to be performed hundreds of thousands times; this can be time consuming.

With the derived s-expanded expressions, the exact behavior can be obtained by substituting numerical values for symbols other than the complex frequency variable s. The resulting network transfer functions are rational polynomials of s with numerical coefficients. Evaluation of these polynomials can be extremely fast. In the context of statistical design, parameters with variations can be kept as symbols while other parameters are substituted as numbers. In the application of analog testability analysis or fault simulation, only potentially faulty circuit parameters are treated as symbols. These lead to mixed symbolic and numerical expressions.

B. Symbolic Pole/Zero Estimation

Symbolic expressions of poles and zeros of network functions can help circuits designers gain the insight on the circuit frequency-domain behavior and stability. It is unknown how a symbolic expression can be derived for an arbitrary pole or zero in the network functions. However, if a pole or zero is well separated from the others, an approximated symbolic expression can be derived from the s-expanded network functions by root splitting ([16, Chapter 4],).

Consider the following s polynomial:

$$f(s) = a_0 s^n + \cdots + a_{n-1}s + a_n = 0.$$

Under the assumption that the root s_k is well separated from the other roots, i.e.,

$$|s_1| \leq \cdots \leq |s_{k-1}| \ll |s_k| \ll |s_{k+1}| \leq \cdots \leq |s_n|.$$

Then, the root s_k can be approximated as

$$s_k = -a_{n-k+1}/a_{n-k}.$$

Since roots for an up-to-fourth order polynomial can be obtained analytically, this method can be generalized to derive approximate symbolic expressions for a cluster of up-to-four poles or zeros as long as clusters are well separated from each other [11], [12].

C. Symbolic Noise Evaluation

Noise in integrated circuits is caused by some random physical phenomena, which lead to small current and voltage fluctuations within circuit devices. The most important noise sources in integrated circuit devices are *thermal noise*, *shot noise*, and *flicker noise* [24]. In general, they are modeled as current sources or voltage sources associated with various devices in the circuit. Each of these devices may include several noise sources due to different physical phenomena. The widely used noise models for resistors, bipolar junction transistors (BJTs), and metal–oxide–semiconductor (MOS) transistors can be found in [25].

Mathematically, noise is characterized in terms of the power spectral density in the frequency domain. The integration of the noise power spectral density over frequency gives the total noise power. Since all the noise sources are uncorrelated, their contributions at an output can be calculated separately. Let $H_p(s)$ denote the transfer function from noise source I_p^2 to an output.

Then, the noise voltage in the root mean square (rms) form at the output is given by

$$V_{\mathrm{out}}(s) = \sqrt{\sum_{p=1} |H_p(s)|^2 I_p^2(f)}. \tag{11}$$

$V_{\mathrm{out}}^2(s)/\Delta f$ is called the noise spectral density function, where $s = 2\pi \mathbf{j} f$, $\mathbf{j}$ is the imaginary number and f is the real frequency variable. With this, computing the noise spectral density function of an output amounts to symbolic computation and manipulation of a set of transfer functions with different inputs and the same output [17], [18].

We illustrate this multifunction computation process on the simple resistance–capacitance (*RC*) filter circuit in Fig. 2. We assume that the device noise is modeled by three noise current sources I_{R_1}, I_{R_2}, and I_{R_3} connected in parallel with resistors R_1, R_2, and R_3, respectively. The system of circuit equations has been formulated in Section II. If we view each entry of the circuit matrix as one distinct symbol, the resulting system determinant and its DDD representation are shown in Fig. 3. Let us consider the input-referred voltage noise at node 1. Then, the three transfer functions associated with three noise sources can be written as

$$H_{R_1}(s) = \frac{v_1}{I_{R_1}} = \frac{(-1)^{(1+1)}\det(Y_{11})}{\det(Y)}$$

$$H_{R_2}(s) = \frac{v_1}{I_{R_2}} = \frac{(-1)^{(1+1)}\det(Y_{11}) - (-1)^{(1+2)}\det(Y_{12})}{\det(Y)}$$

$$H_{R_3}(s) = \frac{v_1}{I_{R_3}} = \frac{(-1)^{(1+2)}\det(Y_{12}) - (-1)^{(1+3)}\det(Y_{13})}{\det(Y)}.$$

Note that in addition to the system determinant $\det(Y)$, we need three minors of $\det(Y)$: $\det(Y_{11}) = DG - FE$, $\det(Y_{12}) = BG$, and $\det(Y_{13}) = BE$. In fact, minors $\det(Y_{11})$ and $\det(Y_{12})$ already exist in $\det(Y)$, as shown in Fig. 10. To represent $\det(Y_{13})$, we need one extra DDD vertex. So, we end up with only eight vertices to represent all the three transfer functions required for representing the input-referred noise spectral density for this *RC* filter circuit.

After all the transfer functions required for a noise spectral density are computed and represented by DDDs, they are transformed into the s-expanded form using s-expanded DDDs and each of them takes on the following form:

$$H_p(s) = \frac{\sum_{i=0}^{m} N_p[i]s^i}{\sum_{i=0}^{n} D_p[i]s^i} \tag{12}$$

where $N_p[i]$, $i = 1, \ldots, m$ and $D_p[i]$, $i = 1, \ldots, n$ are represented by coefficient DDDs. Substituting (12) into (11), we obtain the noise spectral density at output port o as

$$V_o^2(f)/\Delta f = \sum_{p=1} \left(\left| \frac{\sum_{i=0}^{m} N_p[i](2\pi \mathbf{j} f)^i}{\sum_{i=0}^{n} D_p[i](2\pi \mathbf{j} f)^i} \right|^2 I_p^2(f) \right) /\Delta f. \tag{13}$$

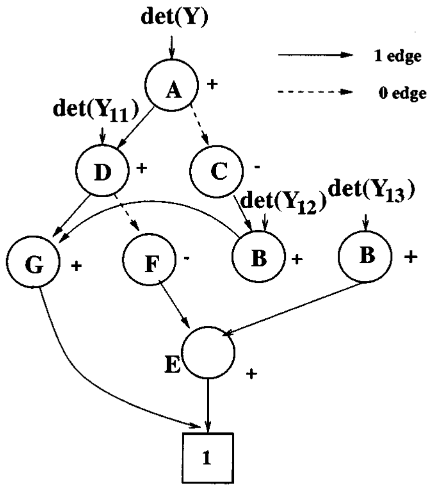

Fig. 10. DDD representing noise transfer functions.

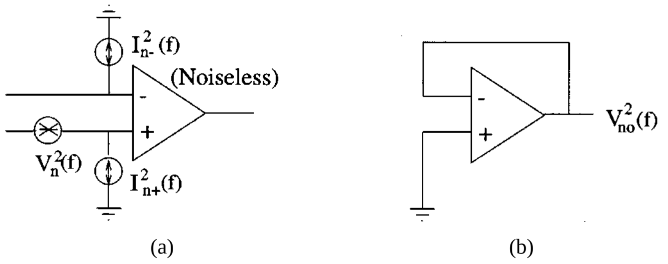

Fig. 11. Input-referred noise model for an operational amplifier. (a) Noise model for opamps. (b) $V_n^2(f) = V_{no}^2(f)$ configuration.

We note that any noise source $I_p^2(f)$ in aforementioned circuit devices is either a constant or a reciprocal function of f, i.e., c/f, where c is a constant. Hence, $|H_p(f)|^2 I_p^2(f)$ is still a rational function of f. Therefore, the computation above can be performed in two ways. First, squaring operation on a transfer function can be implemented as a product of two DDDs. This is a purely symbolic approach. Second, the numerical values of the coefficients in each transfer function are computed by evaluating coefficient DDDs using DDD_EVALUATE operation [29], [30]. This leads to an expression with only the frequency variable f as the symbol, This expression can be used for repetitive noise evaluation or noise modeling for high-level noise simulation.

D. Input-Referred Block Noise Modeling for High-Level Simulation

Any noisy two-port circuit can be modeled by a noiseless circuit with two generally correlated input noise sources: a series voltage source and a parallel current source [19]. The noise of an operational amplifier circuit can be modeled by three noise sources as shown in Fig. 11(a). For an operational amplifier with the MOS field-effect transistor (MOSFET) input stage that operates at not very high frequencies, the two current noise sources $I_{n-}^2(f)$ and $I_{n+}^2(f)$ can be ignored.

The *input-referred* noise voltage source $V_n^2(f)$ can be generated automatically by DDD-based symbolic noise evaluation described in Section V-C on the circuit configuration shown in Fig. 11(b). Then, the generated $V_n^2(f)$ combined with a noiseless operational amplifier circuit can be used as an operational

TABLE II
STATISTICS ON COMPLEX DDD AND s-EXPANDED DDD CONSTRUCTION

circuit	mx_size	#nonzero	Complex DDD				s-expanded DDD					
			#numP	#denP	$\|DDD\|$	CPU (s)	#numP	#denP	$\|DDD\|$	CPU (s)	deg(num)	deg(den)
miller	6	21	9	13	51	0.02	108	252	220	< 0.01	6	6
butter	7	19	1	13	22	0.01	1	239	120	< 0.01	1	12
rclad7	8	22	1	21	26	0.01	1	172	72	< 0.01	1	6
rlctest	9	37	80	120	152	0.04	140	234	259	0.01	3	4
ccstest	9	36	56	120	97	0.03	100	234	173	0.02	3	4
vcstest	12	46	56	64	70	0.04	56	64	70	< 0.01	1	1
rclad21	22	64	1	17711	84	0.05	1	123009	168	0.01	1	6
Cascode	14	76	42154	79643	1994	3.17	9.42×10^6	1.52×10^7	18570	0.88	14	14
$\mu A741$	23	90	10970	108032	6654	2.96	7.77×10^9	9.31×10^{10}	99844	5.08	23	23
bigtst	32	112	164168	1.6×10^7	951	0.89	1.91×10^8	6.67×10^{10}	13431	0.58	18	21
rclad100	101	301	1	5.7×10^{20}	398	1.14	1	1.0×10^{35}	16767	0.65	1	85
rctreeA	40	119	3560	7.1×10^7	208	0.21	1.2×10^7	3.8×10^{14}	5040	0.19	20	39
rctreeB	53	158	1.4×10^5	3.0×10^{10}	299	0.44	1.6×10^{10}	3.8×10^{19}	9478	0.35	28	52

amplifier model for noise evaluation of circuits that contain operational amplifiers as subcircuits.

VI. Experimental Results

The proposed algorithms have been implemented in a prototype symbolic analyzer for analog integrated circuits. The program reads in the circuit description in the SPICE format and uses SPICE to perform dc operating point analysis and to create the small-signal models. Next the system of circuit equations are set up based on the MNA formulation. With each nonzero entry in the MNA circuit matrix viewed as a distinct symbol, the complex DDD that represents a symbolic network function of interest is constructed using algorithm DDD_OF_MATRIX from [29] and [30]. Then, an s-expanded DDD is constructed using the algorithms presented in this paper. From the s-expanded DDD, frequency-domain simulation is performed by first evaluating the coefficient DDDs and then calculating the response by substituting just the frequency variable. Symbolic expressions of dominant poles and zeros are derived based on the root splitting method. Small-signal noise analysis is also supported.

The program has been tested on a set of benchmark circuits, including various filter circuits, MOS, and bipolar operational amplifiers [30]. Statistics on these circuits, their complex DDD construction, and s-expanded DDD construction are reported in Table II. For each *circuit*, columns 2 and 3 give the size of its MNA circuit matrix mx_size and the number of nonzeros $\#nonzero$ in the matrix. Statistics on complex DDD construction and s-expanded DDD construction are collected in columns 4 to 7 and columns 8 to 13, respectively, where $\#numP$ is the number of product terms in the numerator of the transfer function, $\#denP$ is the number of product terms in the denominator of the transfer function, and $|\text{DDD}|$ is the size (number of vertices) of the DDD representing both the numerator and the denominator of the transfer function. The CPU time reported is in seconds on an Ultra-SPARC-I workstation with 167-MHz clock rate. For complex DDDs, it is the time of constructing complex DDDs from circuit matrices. For s-expanded DDDs, it is the time of constructing s-expanded DDDs from complex DDDs.

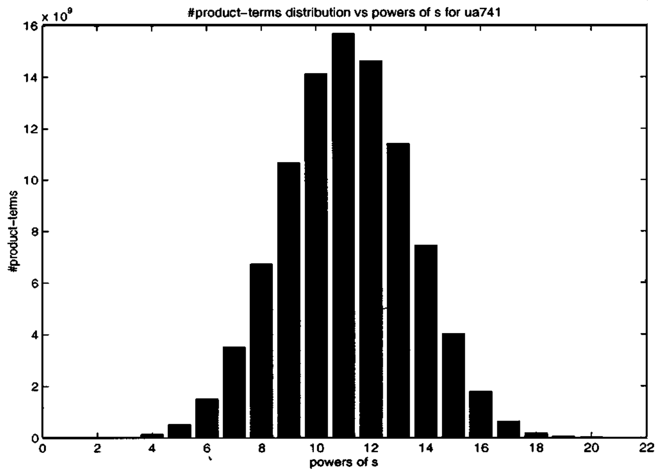

Fig. 12. Product term distribution of μA741 over power of s.

The last two columns deg(num) and deg(den) denote the degrees of the numerator polynomial and the denominator polynomial, respectively. Note that k in Theorem 1 equals deg(den).

From Table II, we can make several observations.

1) For large circuits, product terms grow dramatically with the size of a circuit, while the DDD sizes grow modestly.
2) For large circuits, s-expansion, i.e., expanding complex products terms into an s polynomial, can lead to an exponential number of s-expanded product terms. For example, the denominator of the transfer function for μA741 has 108 032 complex product terms. Expansion of these terms leads to 7.77×10^{10} s-expanded product terms. In Fig. 12, we draw the distribution of the number of product terms over the power of s. We can see that the numbers of product terms in the coefficients of middle powers of s increase rapidly.

If each term in an MNA matrix entry is represented as a distinct symbol (*full symbol representation*), the number of expanded product terms increases much more dramatically. For μA741, a matrix entry typically has three or four distinct symbols (three or four devices connected to a node) with nine as its maximum. Using the

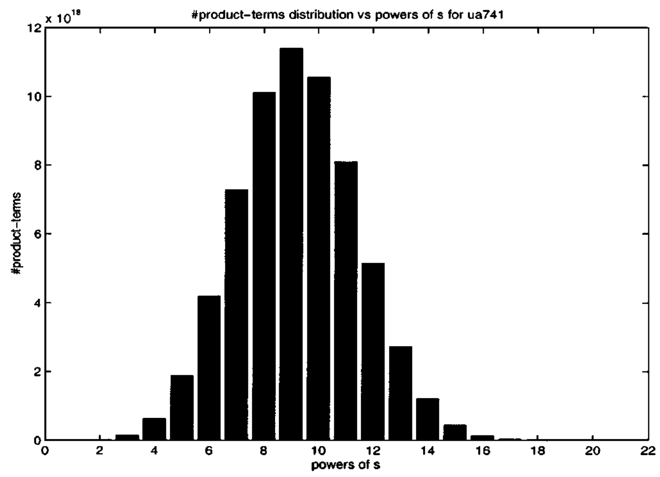

Fig. 13. Product-term distribution of μA741 over power of s for full symbol representation.

full symbol representation, Fig. 13 shows the distribution of the number of product terms over the power of s for the denominator of the μA741 transfer function. We can see a more than nine orders of magnitude increase in the number of product terms in comparison with the results in Fig. 12.

3) Despite the rapid growth of s-expanded product terms, the sizes of s-expanded DDDs and the CPU time in constructing these s-expanded DDDs grow very modestly. For μA741, the s-expanded transfer function with 7.77 $\times 10^9$ product terms in the numerator and 9.31×10^{10} product terms in the denominator can be compactly represented by a DDD with only 99 844 vertices. Even if for the full symbol representation, where the number of product terms grows by nine orders of magnitude, the number of DDD vertices increases only to 297 115 (about three times of 99 844). The construction of s-expanded DDDs takes only a few CPU seconds on an Ultra-SPARC-I workstation. This demonstrates the superior expressive power of s-expanded DDDs and the efficiency of s-expanded-DDD based algorithms for symbolic analysis.

4) In fact, experimental results in Table II validates Theorem 1 for the case of compact symbol representation ($l = 2$). Fig. 14 shows that the actual size of an s-expanded DDD is clearly bounded by $2k|\mathrm{DDD}|$.

With s-expanded symbolic expressions, frequency-domain simulation can be performed rapidly. This has been demonstrated by experimental results in Table III. For each *circuit*, *s-expanded_Eval* is the CPU time used for calculating the numerical values of coefficient DDDs in the resulting s-expanded DDDs and s-expanded DDD is the CPU time used for frequency-domain simulation, i.e., numerical evaluation of the resulting s-polynomials over 1000 frequency points. The CPU time used by SPICE is described in column 4 (*Spice*). We see that an order-of-magnitude speedup has been achieved and furthermore the speedup usually increases with the size of a circuit. We note that after DDD construction, DDD-based frequency-domain simulation amounts to s polynomial evaluation. We used here a very naive nonoptimized implementation

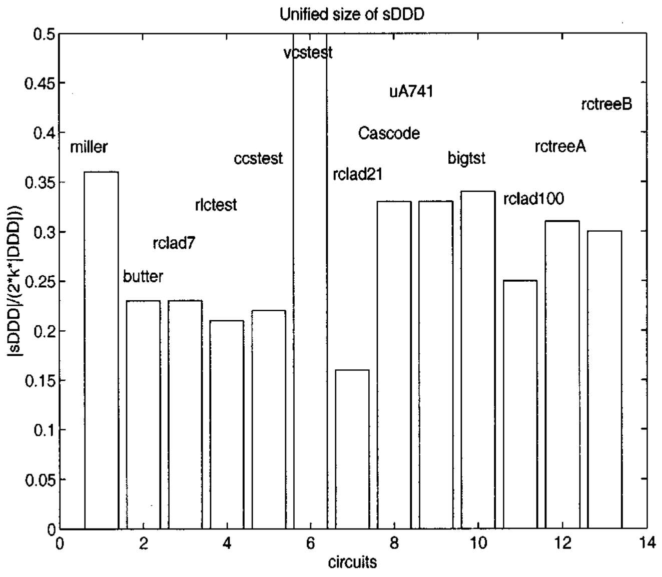

Fig. 14. Sizes of s-expanded DDDs versus sizes of the complex DDDs.

of s polynomial evaluation; an optimized implementation of s polynomial evaluation can be much faster.

To illustrate symbolic pole/zero estimation, we consider a simplified two-stage MOS operational amplifier *TwoStage* taken from [16]. Its schematic is shown in Fig. 15. The transfer function of interest is the open-loop voltage gain from input node $V_{\mathrm{in}2}$ (positive input) to output node V_{out} with node $V_{\mathrm{in}1}$ shorted to the ground. The exact values of three poles are described in Table IV.

Since three poles are far away from each other, the pole splitting method can be used to find the symbolic expressions for these poles. The simplified expressions for the three poles obtained by our program are as follows:

$$\text{pole } 1 = -\frac{(g_{d6}+g_{d7})(g_{d1}+g_{d3})}{g_{m6}c_C}$$
$$\text{pole } 2 = -\frac{g_{m6}}{c_L}$$
$$\text{pole } 3 = -\frac{g_{m3}}{c_{gs3}+c_{gs4}}.$$

Their numerical values agree with the results in Table IV.

Next, we evaluate the proposed DDD-based approach for the noise evaluation of analog integrated circuits. Our program first generates a single shared DDD that represents all the transfer functions required for the representation of the output noise power spectral density. To compare with SPICE, the numerical values of noise sources from SPICE are taken and fed into our program and the values of the output noise power spectral density are then evaluated. Table V summarizes the experimental results for three circuits *miller*, *Cascode*, and *μA741*. *System* $|\mathrm{DDD}|$ is the number of complex DDD vertices used to represent the determinant of the MNA circuit matrix. *Total* $|\mathrm{DDD}|$ is the total number of complex DDD vertices used to represent all the required transfer functions in the noise spectral density function. Column *construction* is the CPU time used to construct a shared DDD representing all the noise transfer functions for

TABLE III
COMPARISON AGAINST SPICE IN NUMERICAL EVALUATION

circuit	s-expanded Eval (s)	s-expanded DDD (s)	Spice (s)	speedup
miller	< 0.01	0.02	0.32	16.0
butter	< 0.01	0.03	0.54	18.0
rclad7	0.01	0.01	0.21	21.0
rlctest	0.01	0.03	0.44	14.7
ccstest	0.01	0.02	0.62	31.0
vcstest	< 0.01	0.02	0.52	26.0
rclad21	0.01	0.03	0.51	16.7
Cascode	0.52	0.03	1.16	38.6
$\mu A741$	3.32	0.05	2.40	48.0
bigtst	0.58	0.04	1.94	48.5
rclad100	4.57	0.13	2.14	13.2
rctreeA	0.64	0.07	0.72	10.2
rctreeB	1.46	0.1	0.94	12.3

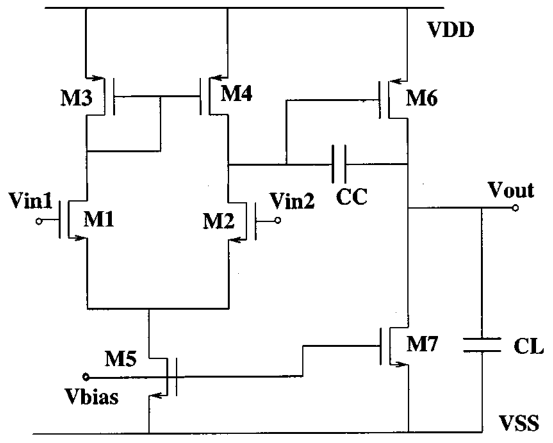

Fig. 15. Simplified two-stage CMOS operational amplifier [16].

TABLE IV
POLES OF CIRCUIT *TWOSTAGE*

poles	$-373.7Hz$	$-0.88MHz$	$-16.8MHz$

each circuit after the system DDD is built. Max f is the highest power of frequency f in both the numerator and denominator of the derived spectral density function. Column DDD is the CPU time used to evaluate the obtained spectral density function for 10 000 frequency points. Column *SPICE* is the CPU time used by SPICE to perform noise analysis over the same number of frequency points.

Finally, we perform system-level noise simulation of a state variable filter circuit shown in Fig. 16. It consists of four identical operational amplifiers, which in turn are implemented by a CMOS Cascode operational amplifier *Cascode*. To compare with SPICE, all the current noise sources in circuit devices were taken from SPICE and fed into our program.

With the proposed DDD-based method, the exact voltage noise spectral density function is first computed. It is then used as the input-referred noise generator as shown in Fig. 11 together with noiseless *Cascode* to perform noise analysis on the state-variable filter circuit. Fig. 17 shows the noise spectral densities of the state variable filter circuit computed by SPICE and by the proposed DDD-based method. It can be seen that the results are identical. We note that the noise spectral density calculation from the resulting symbolic expression took 0.03 s, while SPICE noise analysis took 1.69 s.

VII. CONCLUSION

Symbolic network functions in the s-expanded form provide a complete small-signal characterization of analog integrated circuits. With symbolic expressions, designers can gain insight about how circuit parameters affect the circuit behavior and create more innovative circuit architectures, while computers can use symbolic expressions as behavioral models to speed up repetitive numerical evaluation for what–if analysis and design optimization. Unfortunately, symbolic analysis was known to be computationally prohibitive and has been regarded as one of the hardest problems. Despite many years of research, symbolic analysis techniques are either applicable only to very small circuits or rely on simplification, which often ignores what could be really important [18].

This paper introduced a new graph-based representation—multiroot DDDs—for s-expanded network functions. It was built on the progress and implementation experience from BDDs for logic synthesis and formal verification [4]. Experimental results with a prototype symbolic analyzer utilizing the proposed approach have shown that the exact s-expanded symbolic transfer functions for analog integrated circuits such as μA741 operational amplifiers can be generated in a few CPU seconds on modern computer workstations. The expressive power of multiroot s-expanded DDDs is so remarkable that for the first time, over 10^{35} product terms have been successfully represented by a multiroot DDD with less than 17 K vertices. Such a representation compactness is achieved by exploring the expression sharing among both the numerator and the denominator s coefficients of symbolic network functions. It is made possible by a vertex ordering heuristic and an implicit construction algorithm for s-expanded DDDs developed in this paper. We proved theoretically as well as validated experimentally both the space and time complexities of multiroot s-expanded DDD construction. We demonstrated the advantages of applying s-expanded DDDs to frequency-domain simulation, pole/zero estimation, as well as noise evaluation. The remarkable compactness of DDDs is further demonstrated in the context of symbolic noise evaluation,

TABLE V
STATISTICS OF NOISE ANALYSIS FOR ANALOG INTEGRATED CIRCUITS

Circuits	#noise-sources	System $\|DDD\|$	Total $\|DDD\|$	construction (s)	Max f	DDD (s)	Spice (s)
miller	8	12	17	0.01	8	0.35	6.17
Cascode	44	1406	3601	0.61	28	1.39	29.29
$\mu A741$	65	2357	18491	8.16	44	2.87	70.70

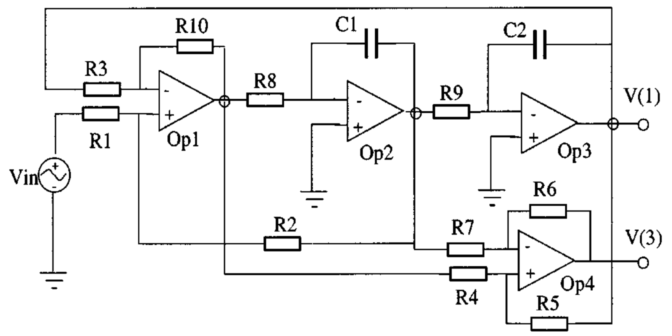

Fig. 16. State-variable filter circuit.

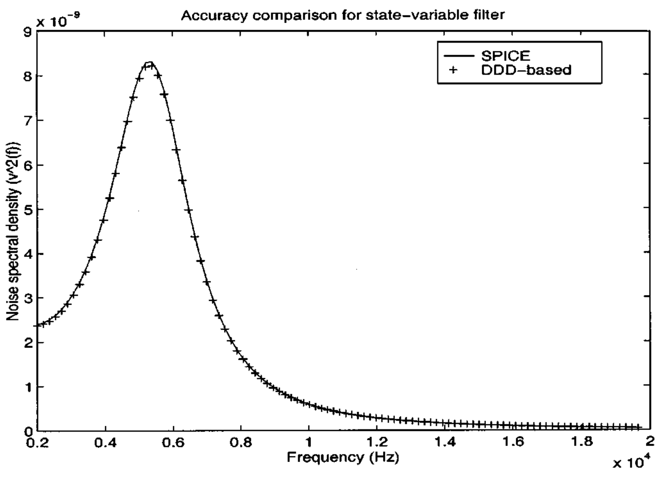

Fig. 17. Noise spectral densities computed by SPICE and DDD.

where multiple transfer functions each being used for a noise source in the circuit can be represented by a DDD with the size comparable to that for a single or a few transfer functions.

We have demonstrated that repetitive numerical evaluation with the derived s-expanded symbolic expressions for frequency-domain simulation and small-signal noise analysis can be much faster than SPICE and the resulting expressions for a circuit block can be used as behavioral model for high-level simulation. We note that in practice, SPICE and SPICE-like simulators are very fast already for frequency-domain simulation and small-signal noise analysis. Therefore, the potential impact of this paper's contribution will mainly be in the area of deriving interpretable symbolic expressions for analog circuit characterization and in those applications where numerical analysis is not viable. Some preliminary results on the further applications of the algorithm described in the paper were presented in [33] for deriving interpretable symbolic expression and in [26] for analog testability analysis.

ACKNOWLEDGMENT

The authors would like to thank Prof. D. Allstot at the University of Washington and Prof. K. Mayaram at Oregon State University for helpful discussions on circuit noise modeling and the four anonymous reviewers for their careful reviews of this paper and for their comments that improved the presentation of this paper.

REFERENCES

[1] S. B. Akers, "Binary decision diagrams," *IEEE Trans. Comput.*, vol. C-27, pp. 509–516, June 1976.
[2] P. E. Aldersona and P. M. Lin, "Computer generation of symbolic network functions—A new theory and implementation," *IEEE Trans. Circuit Theory*, vol. CT-20, pp. 48–51, Jan. 1973.
[3] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 677–691, Aug. 1986.
[4] ——, "Binary decision diagrams and beyond: Enabling technologies for formal verification," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 236–243.
[5] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. 27th IEEE/ACM Design Automation Conf.*, June 1990, pp. 40–45.
[6] J. G. Broida and S. G. Williamson, *A Comprehensive Introduction to Linear Algebra*. Reading, MA: Addison-Wesley, 1989.
[7] M. Carmassi, M. Catelani, G. Iuculano, A. Liberatore, S. Manetti, and M. Marini, "Analog network testability measurement: A symbolic formulation approach," *IEEE Trans. Instrumen. Meas.*, vol. 40, pp. 930–935, Dec. 1991.
[8] W. K. Chen, *Applied Graph Theory*. Amsterdam, The Netherlands: North-Holland, 1971.
[9] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *J. Appl. Phys.*, vol. 19, pp. 55–63, 1948.
[10] P. Feldman and R. W. Freund, "Circuit noise evaluation by Pade approximation based model-reduction techniques," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Jan. 1997, pp. 132–138.
[11] F. V. Fernández, A. Rodríguez-Vázquez, and J. L. Huertas, "A tool for symbolic analysis of analog integrated circuits including pole/zero extraction," in *Proc. Eur. Conf. Circuit Theory and Design*, 1991, pp. 752–761.
[12] ——, "Interactive AC modeling and characterization of analog circuits via symbolic analysis," *J. Analog Integr. Circuits Signal Process.*, vol. 1, no. 3, pp. 183–208, Nov. 1991.
[13] ——, "Formula approximation for flat and hierarchical symbolic analysis," *J. Analog Integr. Circuits Signal Process.*, vol. 3, no. 1, pp. 43–58, Jan. 1993.
[14] F. V. Fernández and A. Rodríguez-Vázquez, "Symbolic analysis tools—the state of the art," in *Proc. IEEE Int. Symp. Circuits Systems*, June 1996, pp. 798–801.
[15] ——, "Symbolic formula approximation," in *Symbolic Analysis Techniques: Applications to Analog Design Automation*, F. V. Fernández, A. Rodríguez-Vázquez, J. L. Huertas, and G. Gielen, Eds. Piscataway, NJ: IEEE, 1998, ch. 6, pp. 141–178.
[16] H. Floberg, *Symbolic Analysis in Analog Integrated Circuit Design*. Norwell, MA: Kluwer, 1997.
[17] G. Gielen and W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits*. Norwell, MA: Kluwer, 1991.
[18] G. Gielen, P. Wambacq, and W. Sansen, "Symbolic analysis methods and applications for analog circuits: A tutorial overview," *Proc. IEEE*, vol. 82, pp. 287–304, Feb. 1994.

[19] P. R. Gray and R. G. Meyer, *Analysis and Design of Analog Integrated Circuits*, 3rd ed. New York: Wiley, 1993.

[20] M. M. Hassoun and P. M. Lin, “A hierarchical network approach to symbolic analysis of large scale networks,” *IEEE Trans. Circuits Syst.*, vol. 42, pp. 201–211, Apr. 1995.

[21] G. Iuculano, A. Liberatore, S. Manetti, and M. Marini, “Multifrequency measurement of testability with application to large linear analog systems,” *IEEE Trans. Circuits Syst.*, vol. 23, pp. 644–648, June 1986.

[22] P. M. Lin, *Symbolic Network Analysis*. Amsterdam, The Netherlands: Elsevier, 1991.

[23] S. Minato, “Zero-suppressed BDDs for set manipulation in combinatorial problems,” in *Proc. 30th IEEE/ACM Design Automation Conf.*, Dallas, TX, June 1993, pp. 272–277.

[24] C. D. Motchenbacher and J. A. Connelly, *Low-Noise Electronic System Design*. New York: Wiley, 1993.

[25] L. W. Nagel, “SPICE2: A Computer Program to Simulate Semiconductor Circuits,” Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. California, Berkeley, CA, May 1975.

[26] T. Pi and C.-J. Shi, “Testability analysis of analog circuits via determinant decision diagrams,” *IEICE Trans. Fundam. Electron., Commun., Comput. Sci.*, vol. E83-A, no. 12, Dec. 2000.

[27] R. Rohrer, L. Nagel, R. Mayer, and L. Weber, “Computationally efficient electronic-circuit noise calculations,” *IEEE J. Solid-State Circuits*, vol. SC-6, pp. 204–213, Aug. 1971.

[28] C.-J. Shi, “Analysis, sensitivity and macromodeling of the Elmore delay in linear networks for performance-driven VLSI design,” *Int. J. Electron.*, vol. 75, no. 3, pp. 467–484, Sept. 1993.

[29] C.-J. Shi and X.-D. Tan, “Symbolic analysis of large analog circuits with determinant decision diagrams,” in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1997, pp. 366–373.

[30] ——, “Canonical symbolic analysis of large analog circuits with determinant decision diagrams,” *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1–18, Jan. 2000.

[31] ——, “Efficient derivation of exact s-expanded symbolic expressions for behavioral modeling of analog circuits,” in *Proc. IEEE Custom Integrated Circuits Conference*, May 1998, pp. 463–466.

[32] J. Starzyk and A. Konczykowska, “Flowgraph analysis of large electronic networks,” *IEEE Trans. Circuits Syst.*, vol. 33, no. CAS-3, pp. 302–315, Mar. 1986.

[33] X.-D. Tan and C.-J. Shi, “Interpretable symbolic small-signal characterization of large analog circuits using determinant decision diagrams,” in *Proc. Design, Automation, Test Eur.*, Munich, Germany, Mar. 1999, pp. 10–13.

[34] ——, “Symbolic circuit-noise analysis via determinant decision diagrams,” in *Proc. Asia South Pacific Design Automation Conf.*, Tokyo, Japan, Jan. 2000, pp. 283–287.

[35] J. Vlach, J. Barby, A. Vanelli, I. Talkhan, and C.-J. Shi, “Group delay as an estimate of delay in logic,” *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 949–953, July 1991.

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

[37] Q. Yu and C. Sechen, “A unified approach to the approximate symbolic analysis of large analog circuits,” *IEEE Trans. Circuits Syst. I*, vol. 43, pp. 656–669, Aug. 1996.

C.-J. Richard Shi (M’91–SM’99) received the B.S. and M.S. degrees in electrical engineering from Fudan University, Shanghai, China, in 1985 and 1987, respectively, and the M.A.Sc. degree in electrical engineering and the Ph.D. degree in computer science from the University of Waterloo, Waterloo, ON, Canada, in 1991 and 1994, respectively.

He is currently an Assistant Professor of Electrical Engineering at the University of Washington, Seattle. He participated in the standardization of IEEE std 1076.1-1999 (VHDL-AMS) for the description and simulation of mixed-signal circuits and systems. His research interests include several aspects of the computer-aided design and test of integrated circuits and systems, with particular emphasis on analog/mixed-signal and deep-submicrometer circuit modeling, simulation, and design automation.

Dr. Shi founded the IEEE/ACM International Workshop on Behavioral Modeling and has served on the technical program committees of several international conferences. He received a Best Paper Award from the IEEE/ACM Design Automation Conference, a Best Paper Award from the IEEE VLSI Test Symposium, a National Science Foundation CAREER Award, and a Doctoral Prize from the Natural Science and Engineering Research Council of Canada. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II, ANALOG AND DIGITAL SIGNAL PROCESSING.

Xiang-Dong Tan (S’96–M’99) received the B.S. and M.S. degrees in electrical engineering from Fudan University, Shanghai, China, in 1992 and 1995, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Iowa, Iowa City, in 1999.

He is currently a Member of Technical Staff in Monterey Design Systems. He worked with Rockwell Semiconductor Systems in the summer of 1997, and worked with Avant! Corporation in the summer of 1998. He was a Research Assistant in the Department of Electrical Engineering, University of Washington, Seattle, from September 1998 to April 1999. His current research interests include very large scale integration (VLSI) physical design automation, symbolic analysis of large analog circuits, layout optimization for performance, timing, power, and clock tree synthesis.

Dr. Tan received a Best Paper Award from the 1999 IEEE/ACM Design Automation Conference in 1999 and the First-Place Student Poster Award from the 1999 Spring Meeting of the Center for Design of Analog Digital Integrated Circuits (CDADIC). He received a Best Graduate Award in 1992 and a number of Excellent College Student Scholarships from 1988–1992, all from Fudan University.