PAPER Special Issue on Special Section on "VLSI Design and CAD Algorithm"

Efficient DDD-based Interpretable Symbolic Characterization of Large Analog Circuits*

Sheldon X.-D. TAN[†a)] ***and*** **C.-J. Richard SHI**[††b)]**,**

SUMMARY A systematic and efficient approach is presented to generating simple yet accurate symbolic expressions for transfer functions and characteristics of large linear analog circuits. The approach is based on a compact determinant decision diagram (DDD) representation of exact transfer functions and characteristics. Several key tasks of generating interpretable symbolic expressions – DDD graph simplification, term de-cancellation, and dominant-term generation – are shown to be able to perform linearly by means of DDD graph operations. An efficient algorithm for generating dominant terms is presented based on the concepts of finding the k-shortest paths in a DDD graph. Experimental results show that our approach outperforms other start-of-the-art approaches, and is capable of generating interpretable expressions for typical analog blocks in minutes on modern computer workstations.

key words: *Circuit Simulation, Behavioral Modeling, Symbolic Analysis, Determinant Decision Diagrams.*

1. Introduction

Mixed-signal (analog and digital) systems are becoming increasingly important. While automation tools exist for digital circuits, analog design is still done manually and depends heavily on designers' experience. One of the challenging problems in analog design is that analog circuits typically have many characteristics, and they depend in a very complicated way on circuit, layout process and environment parameters. In this paper, we present a new approach to generating interpretable analytic expressions for small-signal characteristics of typical analog building blocks.

Previous attempts to generate interpretable expressions are based on various symbolic analysis methods to generate sum-of-product representations for network functions. This area has been studied extensively in 1960s-1980s [9]. The resulting approaches, however, are only feasible for very small circuits, since the number of expanded product terms grows exponentially with the size of a circuit, and resulting expressions become not interpretable by analog designers. Recently, various approximation schemes have been developed. Approximation after generation is reliable but it requires the expansion of product terms first [7], [15], [21]. Some improvement techniques based on nested expressions have been proposed [4], [16]. But they generally suffer symbolic term-cancellation and align-term problems. Approximation during generation extracts only significant product terms [5], [20], [22]. It is very fast, but has two major deficiencies: First, if accurate expressions are needed, then the complexity of the approach becomes exponential. Second, it works only for transfer functions. Other small-signal characteristics such as sensitivities, symbolic poles and zeros, cannot be extracted in general. At the same time, approximation before generations [8], [22] were proposed in which the complexity of a circuit is simplified before symbolic analysis methods are applied. In [3], a signal flow graph based approximation before generation method is proposed and demonstrated successfully to symbolic pole and zero generation.

In this paper, we present an efficient approach to deriving interpretable symbolic expressions based on determinant decision diagram (DDD) representations of circuit transfer functions [12], [14]. We show that how DDD graphs can be simplified, cancellation-free s-expanded (multi-root) DDDs can be constructed and dominant term generation can be performed elegantly. Since we start with exact symbolic expressions in DDD representations, our new approximation method has the reliability in the approximation-after-generation methods. The actual accuracy of simplified expressions mainly depends on the how errors are monitored and controlled during the approximation process. Since we have exact representations of circuit characteristics, our approach is not limited to any specific errors controlling mechanism and the errors of simplified expressions can be controlled to reach any required accuracy. Meanwhile, the new method also shows capability in approximation-before/after-generation for analyzing large analog circuits due to powerful DDD graphs. Experimental results show that our algorithm compares favorably with other state-of-the-art approximation methods [22].

Some preliminary results of this paper were presented [17], [18]. This paper is organized as follows.

Manuscript received February 14, 2003.
Manuscript revised June 7, 2003.
Final manuscript received November 7, 2003.

[†]Department of Electrical Engineering, University of California, Riverside, CA 92521, USA
[††]Department Of Electrical Engineering, University of Washington, Seattle, WA 98195 USA
a) E-mail: stan@ee.ucr.edu
b) E-mail: cjshi@ee.Washington.edu

*Some preliminary results of this paper appeared in *Proc. IEEE Asia South Pacific Design Automation Conference (ASP-DAC'03)* and *Proc. IEEE Design, Automation and Test in Europe (DATE'99)*. This work is partially funded by Univ. of California Academic Senate Research Funds and by DARPA NeoCAD program under Grant N66001-01-1-8920.

Section 2 reviews the concepts of DDDs and s-expanded DDDs. Section 3 presents a general DDD-based framework for deriving interpretable symbolic expressions. Experimental results are described in Section 4. Section 5 concludes the paper.

2. DDDs and s-Expanded DDDs

In this section, we provide a brief overview of the notion of determinant decision diagrams [12]. We review how an s-expanded DDD can be used to represent the symbolic coefficients of an s polynomial.

Determinant Decision Diagrams [12] are compact and canonical graph-based representation of determinants. The concept is best illustrated using a simple RC filter circuit shown in Fig. 1. Its system equations can be written as

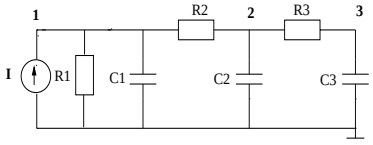

Fig. 1 A simple RC circuit.

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

We view each entry in the circuit matrix as one distinct symbol, and rewrite its system determinant in the left-hand side of Fig. 2. Then its DDD representation is shown in the right-hand side. Please refer to [12] for the formal definition of a DDD graph.

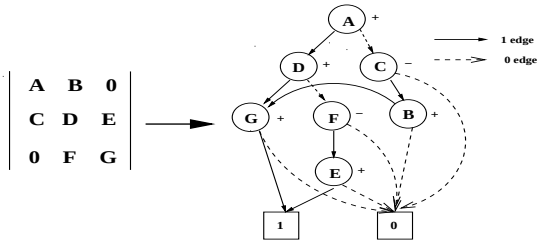

Fig. 2 A matrix determinant and its DDD.

A *1-path* in a DDD corresponds a product term in the original DDD, which is defined as a path from the root vertex (A in our example) to the 1-terminal including all symbolic symbols and signs of the vertices that originate all the 1-edges along the 1-path. In our example, there exist three 1-paths representing three product terms: ADG, $-AFE$ and $-CBG$. The root vertex represents the sum of these product terms. Size of a DDD is the number of DDD vertices, denoted by $|DDD|$. Note that the size of a DDD depends on the size of circuits in a very complicated way. $|DDD|$ is a linear function of the circuit size for ladder circuits and it may grow super-linearly in general with sizes of circuits [12].

To exploit the DDD to derive circuit characteristics, we need to directly represent circuit parameters not matrix entries. To this end, s-expanded DDDs are introduced [14]. Consider again the circuit in Fig. 1 and its system determinant. Let us introduce a unique symbol for each circuit parameter in its admittance form. Specifically, we introduce $a = \frac{1}{R_1}$, $b = f = \frac{1}{R_2}$, $d = e = -\frac{1}{R_2}$ $g = k = \frac{1}{R_3}$, $i = j = -\frac{1}{R_3}$, $C_1 = c, h = C_2, l = C_3$. Then the circuit matrix can be rewritten as

$$\begin{bmatrix} a+b+cs & d & 0 \\ e & f+g+hs & i \\ 0 & j & k+ls \end{bmatrix}$$

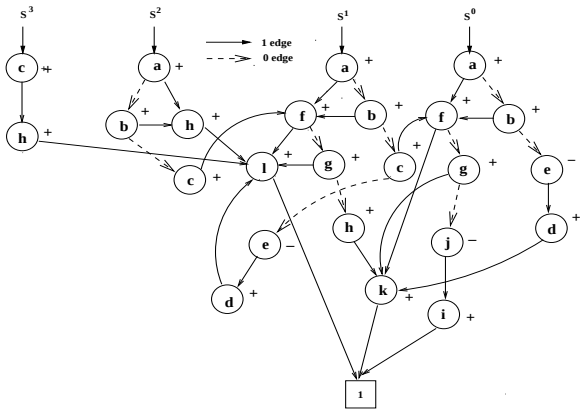

Fig. 3 An s-expanded DDD.

The original 3 product terms will be expanded to 23 product terms in different powers of s. We can represent these product terms nicely using a slight extension of the original DDD, as shown in Fig. 3. This DDD has exactly the same properties as the original DDD except that there are four roots representing coefficients of s^0, s^1, s^2, s^3. Each DDD root represents a symbolic expression of a coefficient in the corresponding s polynomial. Each such DDD is called a *coefficient* DDD, and the resulting DDD is a *multi-root* DDD. The original DDD in which s is contained in some vertices is called *complex* DDD.

The s-expanded DDD can be constructed from the complex DDD in linear time in the size of the original complex DDD [13], [14]. In Fig. 4, we present a version of the algorithm called CoeffConst. CoeffConst takes a complex DDD rooted at D and returns its corresponding coefficient DDD, where,

- *D.child1* and *D.child0* denote, respectively the vertices pointed to by the 1-edge and 0-edge of vertex D.
- Let each circuit node be connected by at most m devices. Then each matrix entry can be a sum of at most m individual elements, each denoted by $D.x_i$ can be of type s^0 or s^1 based on the Modified

Nodal Analysis (MNA) formulation. For example, for a matrix entry $a + b + cs$, we have $m = 3$, $D.x_1 = a, D.x_2 = b$, and $D.x_3 = c$, and their types are s^0, s^0, and s^1.
- CoeffUnion(P_1, P_2) computes the union of two coefficient DDDs, P_1 and P_2.
- CoeffMultiply(P, v) computes the product of coefficient DDD P and coefficient DDD vertex v.
- $P * s$ increments the power of s in coefficient DDD P.

```
CoeffConst(D)
1   if ( D = 0 or D = 1)
2       return NULL
3   L0 = CoeffConst(D.child0)
4   L1 = CoeffConst(D.child1)
5   Presult = NULL
6   for i = 1 to m do
7       if (type(D.xi) = s^0)
8           Pg = CoeffMultiply(L1,D.xi)
9           Presult = CoeffUnion(Pg, Presult)
10      if (type(D.xi) = s^1)
11          Pc = CoeffMultiply(L1 * s,D.xi)
12          Presult = CoeffUnion(Pc, Presult)
13  return CoeffUnion(Presult, L0)
```

Fig. 4 Coefficient DDD construction.

3. A DDD-based Framework for Symbolic Expression Simplification

Using modified nodal analysis (MNA), a linear(ized) analog circuit can be described by a system of linear equations in the following general form: $\mathbf{Tx} = \mathbf{w}$, where the *circuit unknown vector* $\mathbf{x}$ may be composed of node voltages and branch currents, and the *circuit matrix* $\mathbf{T}$ is a modified symbolic admittance matrix.

We consider linear circuit modeling as finding network transfer functions, sensitivity expressions, symbolic pole and zero expressions. According to Cramer's rule, the kth component x_k of the unknown vector $\mathbf{x}$ is obtained as follows:

$$x_k = \frac{\sum_{i=1}^{n} w_i \ (-1)^{i+k} \ det(\mathbf{T}_{t_{i,k}})}{det(\mathbf{T})}. \tag{1}$$

Note that the numerator and the denominator are only composed of the determinant and cofactors of the circuit matrix, which can be represented effectively using DDDs. With this, simplification of symbolic expressions for parametric circuit modeling can be performed by DDD manipulations:

- A network transfer function is defined as the ratio of an output unknown from $\mathbf{x}$ over an input from $\mathbf{w}$. This can be represented as the ratio of two complex DDDs, or two s-expanded DDDs.
- Under the assumption that poles are far away from each other, the root splitting method can be applied to find symbolic expressions for poles and zeros [6]. In this case, a pole or zero can be represented as the ratio of two consecutive coefficient DDDs.
- Sensitivity of a DDD with respect to a circuit parameter amounts to finding a DDD cofactor. Sensitivities of general circuit modeling characteristics with respect to circuit parameters can be represented as expressions of DDDs and their cofactors.

After we obtain the exact DDD representations of linear circuit characteristics, the key task is how to simplify a DDD or a ratio of two DDDs.

Simplification of symbolic expressions consists of the following tasks: 1) *discarding insignificant terms*, 2) *constructing cancellation-free expressions* and 3) *generation of dominant terms*. In this section, we show that all these tasks can be performed efficiently by means of DDD graph manipulations. Moreover, since we begin with the exact and compact DDD representation of linear circuit characteristics, all the error-controlling mechanisms [2] such as monitoring response magnitudes/phases, poles/zeros, can be effectively implemented. This is only feasible previously in approximation-after-generation procedures, which work only for small analog circuits [7], [21].

3.1 Discarding Insignificant Terms

Discarding insignificant terms is to delete those terms not significant to the characteristics of interest. It consists of two methods: *device elimination* and *node contraction*.

Consider a transfer function written in the following form:

$$f(p) = \frac{N}{D} = \frac{pN_p + N_{\overline{p}}}{pD_p + D_{\overline{p}}}, \tag{2}$$

where p is a circuit element in the admittance form, N_p (D_p) is the sum of all the product terms in N (D) containing p from which p is removed, and $N_{\overline{p}}$ ($D_{\overline{p}}$) is the set of product terms in N (D) not containing p.

There are two scenarios where p can be eliminated from both N and D. First, if both $D_{\overline{p}}$ and $N_{\overline{p}}$ are compared to N and D, then f can be simplified by $f' = \frac{N_{\overline{p}}}{D_{\overline{p}}}$. This step removes those devices that are not significant for the characteristics of interest. It is called *device elimination*. Second, if both pN_p and pD_p dominate N and D, then f can be simplified by $f' = \frac{N_p}{D_p}$. This is called *node contraction*. This is the *node contraction* step. During those two steps, we make sure that the errors of the simplified expression ($f' = \frac{N_{\overline{p}}}{D_{\overline{p}}}$ or $f' = \frac{N_p}{D_p}$) are bounded by some error bounds over the frequency range of interests.

With this, the number of elements in each product term is decreased by one. Different from [22], which

considers each individual device p, we consider a group of devices connected to a particular circuit node. This idea has been proved to be more effective, since for such circuits as Opamps and OTAs, many devices in the bias circuitry can be eliminated without affecting small-signal behaviors of circuitry in signal paths.

Both device elimination and node contraction are performed on complex DDDs and involve mainly two basic DDD operations: *Cofactor* and *Remainder*, which are to compute the cofactor and remainder of a DDD representation with respect to a symbolic parameter. These operations take liner time in the DDD size [12].

3.2 Construction of Cancellation-Free s-Expanded DDDs

Canceling terms arise from the use of the Modified Nodal Analysis (MNA) formulation and device matching in analog circuits. For instance, consider the s-expanded DDD in Fig. 3. Since $g = k = \frac{1}{R_3}$ and $i = j = -\frac{1}{R_3}$, term $agks^0$ cancels term $-ajis^0$ in the coefficient DDD of s^0. To ease our discussion, we refer it to as *symbolic cancellation* if terms are canceling each other symbolically or *numerical cancellation* if terms are canceling each other numerically, i.e., only if their normal numeric values are used.

Removing canceling terms is useful for enhancing expression interpretability, the efficiency and accuracy of numerical evaluation. It also facilitates efficient dominant term generation, since most of generated terms can be canceling terms. Our experimental results show that up to 70-90 percent product terms can be canceling terms for a typical analog circuit.

In this subsection, we focus on building symbolic cancellation-free s-expanded DDDs. This is important since the MNA formulation leads naturally to a lot of canceling terms. In practice, removing canceling terms has been known to be a difficult problem in determinant-based symbolic analysis methods [7], [11]. Numerical cancellation can be taken care of during the dominant term generation step.

We consider several MNA matrix patterns shown in Fig. 5.

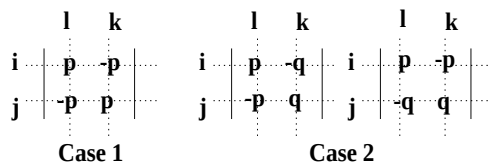

Fig. 5 Matrix patterns causing term cancellation.

Where matrix element p and q can be $1/R_1$, sC_1 stamped from device resistor R_1 and capacitor C_1 respectively in MNA formulation. Notice that case 1 may come from the rectangular stamps of a floating resistor in the nodal admittance formulation (in this case, $l = i$ and $k = j$) and case 2 may come from stamps of a voltage controlled current source and one RC component.

Let L_1, L_2, L_3, L_4 denote, respectively, unique DDD symbols at four rectangular positions (i, l), (i, k), (j, l), (j, k) in the matrix. For case 1 in Fig. 5, $L_1 = p$, $L_2 = -p$, $L_3 = -p$ and $L_4 = p$. Then it is easy to show the following lemma:

Lemma 1: For each product term containing L_1 and L_4, there exist a corresponding canceling product term containing L_2 and L_3.

Canceling terms caused by matrix pattern cases 1 and 2 can be removed efficiently from a DDD by using basic DDD operations: first perform two cofactoring operations with respect to either L_1 and L_4 or L_2 and L_3; then multiply the obtained DDD with both L_1, L_4, and L_2, L_3 respectively to obtain the complete canceling terms; finally subtract all the canceling terms from the original DDDs.

Canceling terms can be removed more efficiently during the construction of s-expanded DDDs from original complex DDDs [14]. To build cancellation-free s-expanded DDDs from DDDs, we build a canceling label list $CL(L_x)$ for each label L_x such that $L_x > L_y, \forall L_y, L_y \in CL(L_x)$. With this, we first loop through all the devices and build the CL for each unique symbol. Then we call function CoeffConst to construct the s-expanded DDDs. The pseudo code for cancellation-free CoeffMultiply in CoeffConst is shown in Fig. 6, where lines 2 to 3 are used not to generate canceling terms.

```
CoeffMulplty(P, D.x)
1  for i = 0 to p − 1 do
2      for each L_y ∈ CL(D.x)
3          P[i] = Remainder(P[i], L_y)
4      P[i] = Multiply(P[i], D.x)
5  return P
```

Fig. 6 Cancellation-free CoeffMultiply.

For this example the resulting cancellation-free DDD shown in Fig. 7 is smaller than the original s-expanded DDD in Fig. 3. Our experimental results, however, show that the cancellation-free s-expanded DDDs can be larger than the normal s-expanded DDDs [17].

3.3 Generation of Dominant Terms

Many small-signal characteristics are dominated by a small number of product terms called *significant* or *dominant* terms. In our framework, the extraction of significant product terms can be transformed to the problem of finding k shortest paths in a DDD graph.

Basing on the DDD graphs, we develop an efficient algorithms find to k dominant terms in a decreasing

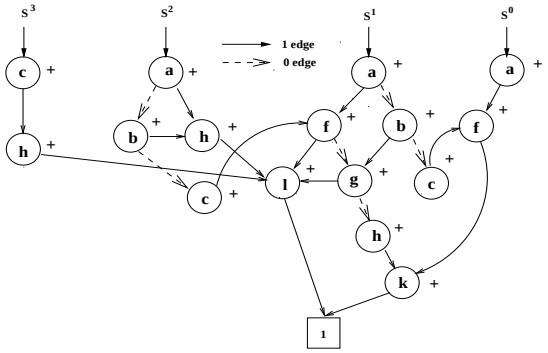

Fig. 7 Cancellation-free multi-root DDD.

order. The algorithm is based on the observation that the k dominant product terms can be transformed to find the k shortest paths in a DDD graph.

We first need to introduce the notion of path weight in DDDs.

Definition 1: The cost of a path in a DDD is defined to be the total cost of the edges along the path where each *0-edge* costs 0 and each *1-edge* costs $-log|a_i|$, and $|a_i|$ denotes the numerical value of the DDD vertex a_i that originates the corresponding 1-edge.

We can show the following result:

Lemma 2: The most significant product term in a symbolic determinant D corresponds to the minimum cost (shortest) path in the corresponding DDD between the root and the 1-terminal.

The shortest path in a coefficient DDD, which is a DAG, can be found by depth-first search in time $O(V + E)$, where V is the number of DDD vertices $|DDD|$ and E is number of edges [1]. For DDDs, $E = 2|DDD|$, thus the shortest path based algorithm takes time $O(|DDD|)$.

A nice property of DDD is that after we find the shortest path from a DDD, we can *subtract* it from the DDD using a basic DDD operation [12], and then we can find the next shortest path in the resulting DDD. Then we have the following result:

Lemma 3: The time complexity of finding k shortest paths is

$$O(k|DDD| + n\frac{k(k-1))}{2}), \tag{3}$$

where n is the depth of the DDD graph.

Proof: It was shown in [18] the number of new DDD vertices created in *subtraction* operation is bounded by n and the time complexity of the function is $O(n)$, where n is the depth of the DDD graph. As a result, we can find the k shortest paths in time $O(k|DDD| + n(\sum_{i=1}^{k-1} i))$, which actually is Eq.(3).

This procedure can be performed on any s-expanded DDD structures. Error controlling is carried out by enumerating the dominant terms from all the coefficient DDDs simultaneously according to certain criteria, until the generated terms, coming from different coefficient DDDs, well approximate the exact expressions in terms of magnitudes and phases.

Following the same strategy in [20], our approach also handles numerical cancellation. Since numerical canceling terms are extracted one after another, they can be eliminated by examining two consecutive terms.

We notice that Verhaegen and Gielen [19] proposed a dynamic programming based dominant term generation algorithm on the basis of one variant of DDD structures. But unfortunately, their DDD structure does not satisfy the canonicity [18] — a crucial property that enables many linear time DDD graph operations [10]. As a result, their DDD graph can't be used with the k-shortest path algorithm as *subtraction* operations cannot be performed. On the other hand, if we maintain the canonicity in the dynamic programming algorithm, the method is not applicable to all the DDD graphs (most important one is the cancellation-free s-expanded DDD graphs) [18].

4. Experimental Results

The proposed approach has been implemented and tested on a number of practical analog circuits. For each circuit, DC analysis is first carried out using SPICE and our program reads in small-signal element values from the SPICE output. For the completeness, the small signal models used for bipolar and MOS transistors are described in Figs. 8 and 9, respectively. The algorithms described in [12], [14] are used to construct complex DDDs and s-expanded DDDs.

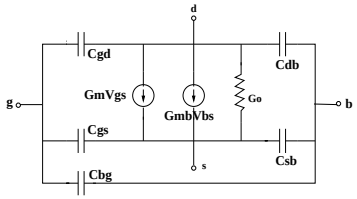

Fig. 8 MOSFET small signal model.

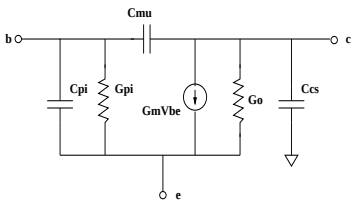

Fig. 9 BJT (Bipolar Junction Transistor) small signal model.

Table 1 summarizes the experimental results for three integrated circuit examples *TwoStage*, *Cascode*, and $\mu A741$ with schematics shown, respectively, in Fig. 10, Fig. 11, and Fig. 12. Row 3 to row 9 describe the statistics of each circuit, the size of the complex

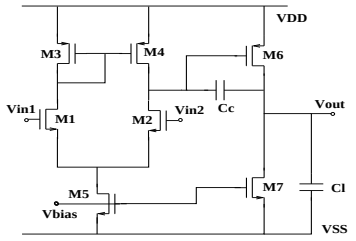

Fig. 10 Simplified two-stage CMOS opamp [6].

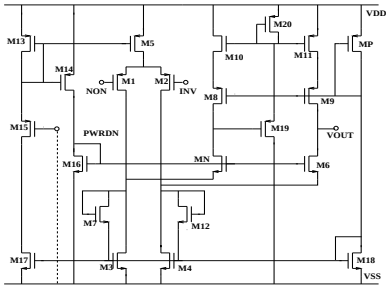

Fig. 11 CMOS cascode opamp.

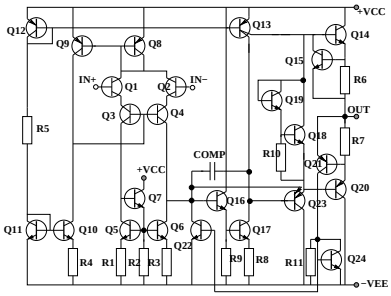

Fig. 12 Bipolar $\mu A741$ opamp.

DDD, and the size of multi-root DDDs used to represent s-expanded circuit transfer functions. We can observe that DDD is highly compact, and the number of vertices is many orders of magnitude less than the number of product terms. For example, the denominator of the s-expanded DDD for $\mu 741$ has $6.40 * 10^{19}$ product terms, but the entire DDD to represent both the denominator and numerator contains only 297117 vertices (this is for the complete and exact transfer function)!

Now we apply the proposed simplification algorithms to derive interpretable symbolic expressions for transfer functions and poles. In each simplification step, we monitor both magnitude and phase of the simplified expressions to control the accumulated error within a given frequency range. We perform device removal and node contraction based on the complex DDD representation. The results are summarized in row 10 to row 12 of Table 1. We observe that this step removes devices that do not affect the small-signal characteristics of the circuits (mainly the bias circuitry not in signal paths) and reduces significantly the size of a complex DDD.

Next, we construct multi-root DDDs from the simplified complex DDDs and remove canceling terms at the same time. Note that even after device elimination and node constriction, the number of product terms by the matrix determinant method is still in the range of millions. For comparison, we also present the multi-root DDDs with all the canceling terms in row 13. The results for multi-root DDDs with canceling terms removed are shown in row 14 and row 15. We see that over 70-90 percent terms are canceling terms. We also notice, however, that elimination of canceling terms may not necessarily reduce the size of DDDs, since some sharing in the original DDD may be destroyed.

After this, we suppress those insignificant high order coefficients. The results are shown in row 16 to row 17. This reduces the size of each DDD by about 10 percent.

Finally, we extract significant terms from the resulting s-expanded DDDs. During this step, we monitor both magnitude and phase of the simplified expressions to ascertain that the accumulated error are bounded. Note that we use different errors bounds at different simplification stages to guide the overall approximation. But in the end, we will make sure that total accumulated errors are within the user specified error bounds. For *TwoStage*, *Cascode* and $\mu A741$, the number of product terms in the final simplified transfer functions (including both the numerator and denominator) are respectively 5, 81, and 73. The whole simplification process takes 12.1 seconds in a Sun Ultra-I workstation for *Cascode*, and 58.1 seconds for $\mu A741$ when we sample two points per frequency decade.

In Fig. 13, Fig. 14, Fig. 15 and Fig. 16, we plot the voltage gain and phase responses using both the exact and simplified expressions. The results from Rainier [22] for *Cascode* and $\mu A741$ are also plotted for comparison, where Rainier's expressions for *Cascode* and $\mu A741$ contain, respectively, 784 and 89 product terms. Those figures show that the new algorithm is able to obtain more accurate results with smaller number of product terms (shown in Table 1) compared with Rainer.

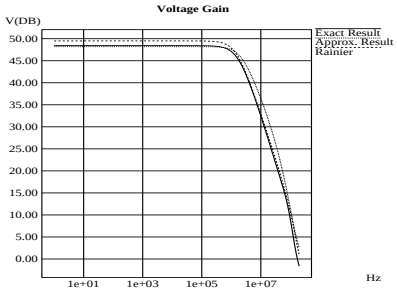

Fig. 13 Accuracy comparison for *Cascode* in magnitude.

Fig. 17 and Fig. 18 show the distributions of number of product terms with respect to different powers of s in the exact symbolic expression, the expression after elimination of insignificant terms (with canceling

Table 1 Statistics of simplified symbolic expressions.

#row	Circuit	Twostage		Cascode		$\mu A741$	
2		Num	Den	Num	Den	Num	Den
3	#lumped devices	14		84		111	
4	#nodes in the circuit	5		14		24	
5	#terms in exact complex DDDs	1	3	9437	28218	381526	$3.82 * 10^6$
6	#vertices in exact complex DDDs	12		1406		9572	
7	#terms in exact multi-root DDDs	16	74	$2.57 * 10^{10}$	$3.60 * 10^{11}$	$8.98 * 10^{16}$	$6.40 * 10^{19}$
8	#vertices in exact multi-root DDDs	52		18109		297117	
9	Highest power of s in exact expressions	3	3	12	12	22	22
10	#devices eliminated	4		34		66	
11	#nodes contracted in complex DDDs	1		3		9	
12	#terms after contraction in complex DDDs	1	2	85	107	75	576
13	#terms after expansion in multi-root DDDs	2	14	$4.28 * 10^6$	$8.04 * 10^6$	103360	$2.05 * 10^7$
14	#terms after expansion(decan) in multi-root DDDs	2	12	$2.00 * 10^5$	$2.44 * 10^6$	20730	$1.08 * 10^6$
15	#vertices after expansion(decan) in multi-root DDDs	17		2705		3886	
16	Highest power of s after expansion	1	2	7	9	9	10
17	Highest power of s after suppression	1	2	7	8	6	8
18	#terms in final expression	2	3	19	62	12	59
19	Highest power of s in final expression	1	2	4	5	3	4

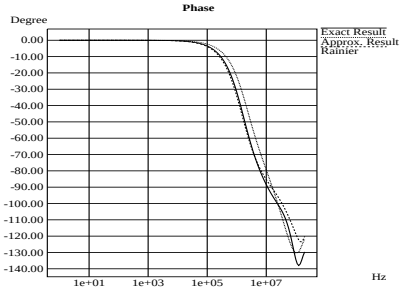

Fig. 14 Accuracy comparison for *Cascode* in phase.

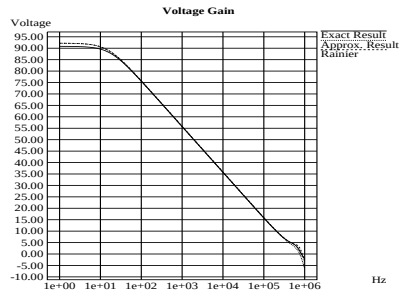

Fig. 15 Accuracy comparison for $\mu A741$ in magnitude.

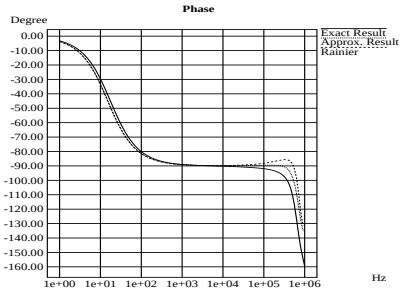

Fig. 16 Accuracy comparison for $\mu A741$ in phase.

terms) and the expression after removing all the canceling terms in the denominator of transfer functions for *Cascode* and $\mu A741$.

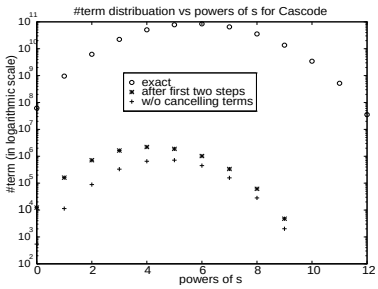

Fig. 17 #terms distribution vs powers of s for *Cascode*.

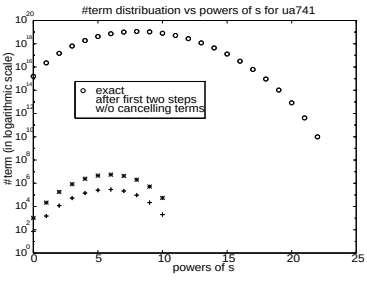

Fig. 18 #terms distribution vs powers of s for $\mu A741$.

The simplified voltage gain for *TwoStage* given by our program is:

$$\frac{g_{m2}g_{m6} + s^1(g_{m2}C_C)}{\frac{1}{(r_{o2}+r_{o4})(r_{o6}+r_{o7})} - s^1(g_{m6}C_C) + s^2(c_{db2} + c_{db4} + c_{gs6} + C_L)C_C}$$

For *TwoStage*, Table 2 shows the exact values of three zeros and three poles.

Table 2 Poles and zeros for opamp *TwoStage*.

poles	$-1.68 * 10^7$	$-8.80 * 10^5$	-373.74
zeros	$3.18 * 10^9$	$-1.68 * 10^7$	$1.11 * 10^7$

Since three poles are far away from each other, the pole splitting method can be used to find their symbolic

expressions. The resulting expression of the third pole based on DDD manipulations is as follows:

$$-\frac{g_{m3}}{c_{gd1}+c_{db1}+c_{gs3}+c_{db3}+c_{gs4}} = -1.68 * 10^7.$$

This agrees with the exact third pole described in Table 2. We note that, however, if we apply the pole splitting method directly on the simplified expression of the transfer function as done in many previous approaches, we cannot obtain the third pole. This is because the third pole and second zero cancel each other, and the third pole information is lost during expression simplification.

5. Conclusions

An efficient approach is proposed for automatic generation of interpretable expressions for ac characteristics of large linear analog circuits. Unlike simplification-during-generation approaches, the proposed approach works on a complete and exact representation of transfer functions. It can monitor the errors in the magnitude, phase, poles and zeros of simplified expressions, and thus provides a reliable and robust simplification scheme. The proposed approach is based on a compact but canonical representation of symbolic expressions, and has the time and space complexity many-orders-of-magnitude less than the simplification-after-generation approaches. It provides a powerful framework for deriving not only interpretable network functions, but also interpretable expressions for other small-signal characteristics such as poles, zeros and sensitivities. Experimental results indicate that the proposed algorithm outperforms Rainier [22] in driving interpretable expressions.

References

[1] T. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts 1990.

[2] W. Daems, W. Verhaegen, P. Wambacq, G. Gielen, and W. Sansen, "Evaluation of error-control strategies for the linear symbolic analysis of analog integrated circuits", *IEEE Trans. Circuits and Systems*, vol. 46, No. 5, pp. 594-606, May 1999.

[3] W. Daems, G. Gielen, and W. Sansen, "Circuit simplification for the symbolic analysis of analog integrated circuits", *IEEE Trans. Computer-Aided Design*, vol. 21, No. 4, pp. 395-407, April 2002.

[4] F. V. Fernández, J.D. Martín, A. Rodríguez-Vázquez and J. L. Huertas, "On simplification techniques for symbolic analysis of analog integrated circuits", in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 1149–1152, 1992.

[5] F. V. Fernández, P. Wambacq, G. Gielen, A. Rodríguez-Vázquez, and W. Sansen, "Symbolic analysis of large analog integrated circuits by approximation during expression generation," in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 25–28, 1994.

[6] H. Floberg, *Symbolic Analysis in Analog Integrated Circuit Design*, Kluwer Academic Publishers, Massachusetts, 1997.

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

[8] J.-J. Hsu and C. Sechen, "DC small signal symbolic analysis of large analog integrated circuits", *IEEE Trans. Circuits and Systems-I: Fundamental*, vol. 41, no. 12, pp. 817–828, Dec. 1994.

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

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

[11] P. Sannuti and N. N. Puri, "Symbolic network analysis—an algebraic formulation", *IEEE Trans. Circuits and Systems*, vol. 27, no. 8, pp. 679–687, Aug. 1980.

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

[13] C.-J. Shi and X.-D. Tan, "Efficient derivation of exact s-expanded symbolic expressions for behavioral modeling of analog circuits", in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, pp. 463–466, 1998.

[14] C.-J. Shi and X.-D. Tan, "Compact representation and efficient generation of s-expanded symbolic network functions for computer-aided analog circuit design", *IEEE Trans. Computer-Aided Design*, vol. 20, No. 7, pp. 813-827, July 2001.

[15] S. J. Seda, M. G. R. Degrauwe and W. Fichtner, "A symbolic analysis tool for analog circuit design automation," in *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD)*, pp. 488–491, 1988.

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

[17] X.-D. Tan and C.-J. Shi, "Interpretable symbolic small-signal characterization of large analog circuits using determinant decision diagrams", in *Proc. Design, Automation and Test in Europe (DATE'99)*, pp. 448–453, Munich, Germany, Mar. 10-13, 1999.

[18] S. X.-D. Tan, C.-J. Shi,"Efficient DDD-based term generation algorithm for analog circuit behavioral modeling", in *Proc. Asia South Pacific Design Automation Conference (ASP-DAC'03)*, pp.789-794, Kitakyushu, Japan, Jan. 2003.

[19] W. Verhaegen and G. Gielen, "Efficient DDD-based symbolic analysis of large linear analog circuits", in *Proc. ACM/IEEE 38th Design Automation Conference (DAC)*, pp. 139–144, Las Vegas, June 2001.

[20] P. Wambacq, G. Gielen and W. Sansen, "A cancellation-free algorithm for the symbolic simulation of large analog circuits", in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 1157–1160, May 1992.

[21] P. Wambacq, G. Gielen and W. Sansen, "A new reliable approximation method for expanded symbolic network functions", in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 584–587, 1996.

[22] Q. Yu and C. Sechen, "A unified approach to the approximate symbolic analysis of large analog integrated circuits", *IEEE Trans. Circuits and Systems*, vol. 43, no. 8, pp. 656–669, Aug. 1996.