Symbolic Analysis of Large Analog Circuits with Determinant Decision Diagrams*

C.-J. Richard Shi Xiangdong Tan
Department of Electrical and Computer Engineering
University of Iowa, Iowa City, Iowa 52242

Abstract

Symbolic analog-circuit analysis has many applications, and is especially useful for analog synthesis and testability analysis. In this paper, we present a new approach to exact and canonical symbolic analysis by exploiting the **sparsity** *and* **sharing** *of product terms. It consists of representing the symbolic determinant of a circuit matrix by a graph—called determinant decision diagram (DDD)—and performing symbolic analysis by graph manipulations. We showed that DDD construction and DDD-based symbolic analysis can be performed in time complexity proportional to the number of DDD vertices. We described a vertex ordering heuristic, and showed that the number of DDD vertices can be quite small—usually orders-of-magnitude less than the number of product terms. The algorithm has been implemented. An order-of-magnitude improvement in both CPU time and memory usages over existing symbolic analyzers ISAAC and Maple-V has been observed for large analog circuits.*

1. Introduction

Symbolic analysis is to calculate the behavior or the characteristic of a circuit in terms of symbolic parameters. It offers many advantages than numerical simulation in many applications such as optimum topology selection, design space exploration, behavioral model generation, and fault detection [8]. However, symbolic analysis has not been widely used by analog designers, and was once judged as completely inefficient [13]. The root of the difficulty is that: the number of product terms in a symbolic expression may increase exponentially with the size of a circuit. For example, for a BiCMOS amplifier that has about 15 nodes and 25 devices (transistors, diodes, resistors and capacitors), the determinant of the circuit matrix contains more than 10^{11} product terms [19]. Any manipulation and evaluation of symbolic expressions will require CPU time at best linear in the number of terms, and therefore have both the time and space complexities exponential in the size of a circuit.

To cope with large analog circuits, modern symbolic analyzers [6] rely on two techniques—hierarchical decomposition and symbolic simplification—developed in the past decade. Hierarchical decomposition is to generate symbolic expressions in the nested instead of the expanded form [9, 16]. Symbolic simplification is to discard those insignificant terms based on the relative magnitudes of symbolic parameters and the frequency defined at some nominal design points or over some ranges. It can be performed before/during the generation of symbolic terms [3, 10, 15, 21] or after the generation [5, 7, 19]. Exploitation of these techniques has enabled the use of symbolic simulators in several university research projects [2, 6, 7, 20]. However, both techniques have some major deficiencies. Manipulation (other than evaluation) of a nested expression usually requires complicated and time-consuming procedures; e.g., sensitivity calculation in [12] and lazy approximation in [15]. On the other hand, simplified expressions only have a sufficient accuracy at some points or frequency ranges. Even worse, simplification often losses certain information, such as sensitivity with respect to parasitics, which is crucial for circuit optimization and testability analysis.

In this paper, we present a new approach to exact symbolic analysis, which is capable of analyzing analog integrated circuits substantially larger than those previously handled. It is based on the two observations on symbolic analysis of large analog circuits: (a) the circuit matrix is sparse and (b) a symbolic expression often shares many sub-expressions. Under the assumption that all the matrix elements are distinct, each product term involves a subset of all the symbolic parameters. Therefore, we adapt Zero-suppressed Binary Decision Diagrams (ZBDDs)—introduced originally for representing sparse subset systems [14]—to represent a symbolic determinant. This leads to a new

*This work is sponsored by U.S. Defense Advanced Research Projects Agency (DARPA) under grant number F33615-96-1-5601 from USAF, Wright Laboratory, Manufacturing Technology Directorate, and by Rockwell Semiconductor Systems.

0-89791-993-9/97 $10.00

interpreted graph, called Determinant Decision Diagram (DDD). The DDD representation has several advantages over both the expanded and arbitrarily nested forms of a symbolic expression. First, similar to the nested form, the DDD representation is compact for a large class of analog circuits. A ladder-structured network can be represented by graphs where the number of vertices (called the DDD size) grows linearly in the number of symbolic parameters. Second, similar to the expanded form, our representation is canonical, i.e., every determinant has a unique representation. Finally, evaluation and manipulation of symbolic determinants (such as sensitivity calculation and approximation) have time complexity proportional to the DDD size.

This paper is organized as follows: Section 2 introduces the background and basic notation for the rest of the paper. Section 3 presents the notion of determinant decision diagrams as an application of ZBDD to represent symbolic determinants. Section 4 describes an effective heuristic for ordering DDD vertices so that the resulting DDD has a smallest or near-smallest size. DDD-based algorithms for symbolic analysis and applications are described in Section 5. Experimental results are presented in Section 6. Section 7 concludes the paper.

2. Notations and Problem Statement

In this section, we introduce some basic notations and concepts that will be used in the rest of the paper.

2.1. Subset Systems and Zero-Suppressed Binary Decision Diagrams

Let V be a *set* of elements. The number of elements in V is called the *cardinality* of V, denoted by $|V|$. The set of all *subsets* of V is called the *power set* of V, denoted by 2^V. A subset X of the power set, written as $X \subseteq 2^V$, is called a *subset system* of V.

A subset system X of V can be decomposed with respect to an element v in V into two unique subset systems, X_v and $X_{\overline{v}}$, where X_v is the set of subsets of V belonging to X that contain v, from which v has been removed, and $X_{\overline{v}}$ is the set of subsets of V belonging to X that do not contain v. For instance, let $X = \{\{v_1, v_2\}, \{v_1, v_3, v_5\}, \{v_3, v_4, v_5\}\}$. Then we have $X_{v_1} = \{\{v_2\}, \{v_3, v_5\}\}$, and $X_{\overline{v}_1} = \{\{v_3, v_4, v_5\}\}$. This decomposition can be represented graphically by a decision *vertex*. It is labeled by v_1, and represents the subset system X. As illustrated in Fig. 1(a), the vertex has two outgoing edges: one points to X_{v_1} (called *1-edge*), and the other to $X_{\overline{v}_1}$ (called *0-edge*). We say that the edges are *originated* from the vertex. If X is recursively decomposed with respect to all the elements of V, one obtains a binary decision tree whose leaves are $\{\{\}\}$ and $\{\}$, respectively. For convenience, we denote leaf $\{\{\}\}$ by the 1-terminal, and $\{\}$ by the 0-terminal.

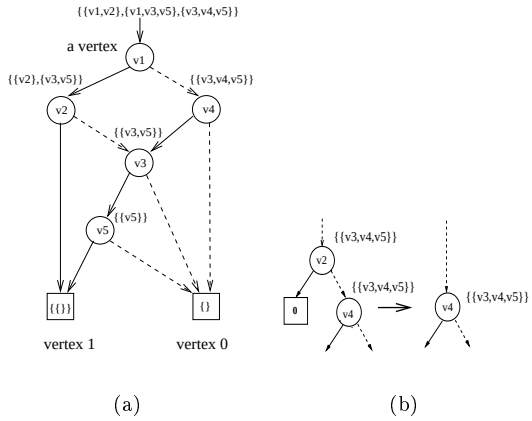

Figure 1: A ZBDD example.

When a subset system X is decomposed with respect to an element that does not appear in X, then its 1-edge points to the 0-terminal. This is illustrated in Fig. 1(b) for decomposing $X = \{\{v_3, v_4, v_5\}\}$ with respect to v_2. To make the diagram compact, Minato suggested the following *zero-suppression* rule for representing sets of *sparse* subsets: eliminate all the vertices whose 1-edges point to the 0-terminal vertex and use the subgraphs of the 0-edges, as shown in Fig. 1(b) [14]. A Zero-suppressed Binary Decision Diagram (ZBDD) is such a zero-suppressed graph with the following two rules due to Bryant [1]: *ordered*—all the elements of V, if one appears, will appear in a fixed order in all the paths of the graph, and *shared*—all equivalent subgraphs are shared. ZBDD is a canonical representation of a subset system, i.e., every subset system has a unique ZBDD representation under a given vertex ordering. For example, Fig. 1(a) is a unique ZBDD representation for the subset system $\{\{v_1, v_2\}, \{v_1, v_3, v_5\}, \{v_3, v_4, v_5\}\}$ with respect to ordering v_1, v_2, v_4, v_3 and v_5. For convenience, every non-terminal vertex is indexed by an integer number greater than those of its descendant vertices [14]. How all the non-terminal vertices are indexed is called *vertex ordering*.

A path from a non-terminal vertex to the 1-terminal is called *1-path*. It defines a subset of V. The subset consists of all the elements of V from which the 1-edges in the 1-path originate. The number of vertices in a ZBDD is called its *size*.

2.2. Matrices, Determinants & Cofactors

Let $e = \{1, ..., n\}$ be a set of integers. Let A denote a set of m elements, called *symbolic parameters* or simply *symbols*, $\{a_1, ..., a_m\}$, where $1 \leq m \leq n^2$ and each symbol is labeled by a unique pair (r, c), where $r \in e$ and $c \in e$. Often, we write A as an $n \times n$ (square) *matrix*, denoted by $\mathbf{A}$, and use $a_{r,c}$ to denote the element of matrix $\mathbf{A}$ at row r and column c. We sometimes use $r(a)$ and $c(a)$ to denote, respectively, the row and column indices of element a.

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & ... & a_{1,n} \\ a_{2,1} & a_{2,2} & ... & a_{2,n} \\ ... & ... & ... & ... \\ a_{n,1} & a_{n,2} & ... & a_{n,n} \end{pmatrix}.$$

If $m = n^2$ the matrix is said to be *full*, otherwise *sparse*. The *determinant* of $\mathbf{A}$, denoted by $det(\mathbf{A})$, is defined by

$$\begin{vmatrix} a_{1,1} & a_{1,2} & ... & a_{1,n} \\ a_{2,1} & a_{2,2} & ... & a_{2,n} \\ ... & ... & ... & ... \\ a_{n,1} & a_{n,2} & ... & a_{n,n} \end{vmatrix} = \sum_{j_1 \neq ... \neq j_n} (-1)^p \cdot a_{1,j_1} ... a_{n,j_n}. \quad (1)$$

Here $(j_1, j_2, ..., j_n)$ is a permutation of e, and p is the number of permutations needed to make the sequence $(j_1, j_2, ..., j_n)$ monotonically increasing. The right hand side of Eq. (1) is a symbolic expression of $det(\mathbf{A})$ in the *expanded* form, more precisely, the *sum-of-product* form, where each *term* is an algebraic product of n symbolic parameters. We note that each symbol can be assigned a real or complex value for analog circuit simulation.

Let p, $p \subseteq e$, and q, $q \subseteq e$ such that $|p| = |q|$. The square matrix obtained from the matrix $\mathbf{A}$ by deleting those rows not in p and columns not in q forms a *submatrix* of $\mathbf{A}$, and is represented by $\mathbf{A}(p, q)$. It has dimension $|p|$ by $|q|$.

Let $a_{r,c}$ be the element of $\mathbf{A}$ at row r and column c. Let $\mathbf{A}_{a_{r,c}}$ be the $(n-1) \times (n-1)$-matrix obtained from the matrix $\mathbf{A}$ by deleting row r and column c, and let $\mathbf{A}_{\overline{a}_{r,c}}$ be the $n \times n$-matrix obtained from $\mathbf{A}$ by setting $a_{r,c} = 0$. Then, the determinant of matrix $\mathbf{A}$ can be *expanded* as below in a way similar to Shannon expansion for Boolean functions:

$$det(\mathbf{A}) = a_{r,c}(-1)^{r+c} det(\mathbf{A}_{a_{r,c}}) + det(\mathbf{A}_{\overline{a}_{r,c}}), \quad (2)$$

where $(-1)^{r+c} det(\mathbf{A}_{a_{r,c}})$ is referred to as the *cofactor* of $det(\mathbf{A})$ with respect to $a_{r,c}$, and $det(\mathbf{A}_{\overline{a}_{r,c}})$ as the *reminder* of $det(\mathbf{A})$ with respect to $a_{r,c}$. The determinant $det(\mathbf{A}_{a_{r,c}})$ is called the *minor* of $det(\mathbf{A})$ with respect to $a_{r,c}$.

2.3. Symbolic Analysis Problem

Consider a linear(ized) time-invariant analog circuit. Its system equation can be formulated by, for example, the modified nodal analysis (MNA) approach in the following general form [17]:

$$\mathbf{Tx} = \mathbf{w}. \quad (3)$$

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.

Symbolic analysis of analog circuits can be stated as the problem of solving the symbolic equation (3), i.e., to find a symbolic expression of any circuit unknown in terms of symbolic parameters in $\mathbf{T}$ and symbolic excitations expressed by $\mathbf{w}$. According to Cramer's rule, the kth component x_k of the unknown vector $\mathbf{x}$ is obtained as follows:

$$x_k = \frac{\sum_{i=1}^{n} w_i \ (-1)^{i+k} \ det(\mathbf{T}_{t_{i,k}})}{det(\mathbf{T})}. \quad (4)$$

Most symbolic simulators are targeted at finding various network functions, each being defined as the ratio of an output unknown from $\mathbf{x}$ to an input from $\mathbf{w}$. These are special cases of (4) or the ratios of the two expressions in the form of (4).

Note that $(-1)^{i+k} det(\mathbf{T}_{t_{i,k}})$ in (4) is the cofactor of $det(\mathbf{T})$ with respect to element $t_{i,k}$ of matrix $\mathbf{T}$ at row i and column k. Therefore, the core issue in analog symbolic analysis is how to find symbolic expressions of $det(\mathbf{T})$ and the cofactors of $det(\mathbf{T})$. In the rest of the paper, we focus on how to represent a symbolic determinant (Sections 3 and 4), and how to compute, manipulate, and evaluate a symbolic determinant (Section 5). For the simplicity, we assume that all the entries in $\mathbf{T}$ are distinct. This assumption has been used in previous symbolic analysis approaches; methods have been proposed to formulate the symbolic equations to meet (or closely meet) this assumption [7, 17].

3. Representation of Symbolic Matrix Determinants

In this section, we apply the notation of zero-suppressed binary decision diagram (ZBDD) to represent a symbolic matrix determinant. A key observation is that the circuit matrix is sparse, and many times, a symbolic expression may share many sub-expressions. For example, consider the following determinant

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

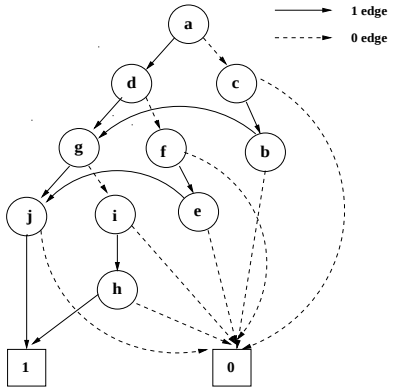

Figure 2: A ZBDD representing $\{adgi, adhi, afej, cbgj, cbhi\}$.

We note that subterms ad, gj, and hi appear in several product terms, and each product term involves a subset (four) out of ten symbolic parameters. Therefore, we view each symbolic product term as a subset, and use a ZBDD to represent the subset system composed of all the subsets each corresponding to a product term. Figure 2 illustrates the corresponding ZBDD representing all the subsets involved in $det(\mathbf{M})$ under ordering $a > c > b > d > f > e > g > i > h > j$. It can be seen that subterms ad, gj, and ih have been shared in the ZBDD representation.

We can view the resulting ZBDD as a graphical representation of recursive application of determinant expansion formula (2) in Section 2.2 with the expansion order $a, c, b, d, f, e, g, i, h, j$. Each vertex is labeled with the matrix entry with respect to which the determinant is expanded, and it represents all the subsets contained in the corresponding submatrix determinant. The 1-edge points to the vertex representing all the subsets contained in the cofactor of the current expansion, and 0-edge points to the vertex representing all the subsets contained in the reminder.

To embed the signs of the product terms of a symbolic determinant into its corresponding ZBDD, we consider one step of matrix expansion with respect to $a_{r,c}$ as defined by (2). The sign is $(-1)^{r+c}$. Note that r and c are, respectively, the row and column indices of the element $a_{r,c}$ in the submatrix, say $\mathbf{A}'$, before this step of expansion. Let the *absolute* row and column indices of the element $a_{r,c}$ in the original matrix $\mathbf{A}$ before any expansion be $r(a_{r,c})$ and $c(a_{r,c})$, respectively. Then, we observe that $r + c$ is equal to the number of rows in $\mathbf{A}'$ with absolute indices less than $r(a_{r,c})$ plus the number of columns in $\mathbf{A}'$ with absolute indices less than $c(a_{r,c})$. We also note that all the rows and columns in $\mathbf{A}'$ except that of $a_{r,c}$ are represented in the subgraph rooted at the vertex pointed to by the 1-edge of vertex $a_{r,c}$. Therefore, the sign of a non-terminal vertex v, denoted by $s(v)$, can be defined recursively as follows:

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

$$s(v) = \prod_{x \in P(v)} sign(r(x) - r(v))\ sign(c(x) - c(v)), \tag{5}$$

where $r(x)$ and $c(x)$ refer to the absolute row and column indices of vertex x in the original matrix, and u is an integer so that

$$sign(u) = \begin{cases} 1 & \text{if } u > 0, \\ -1 & \text{if } u < 0. \end{cases}$$

2. If v has an edge pointing to the 1-terminal vertex, then $s(v) = +1$.

This is called the *sign rule*. For example, in Fig. 3, vertices c, f, and i are assigned a negative sign $(-)$ by using the sign rule.

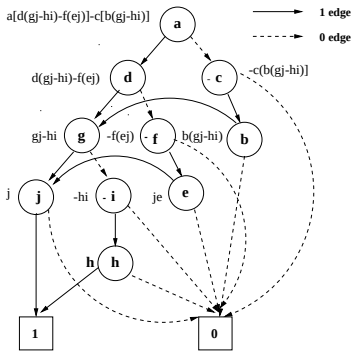

Figure 3: A determinant decision diagram for $\mathbf{M}$.

It can be verified that all 1-paths originating from the same vertex yield the same sign, and the product of all the signs in a path is exactly the sign of the corresponding product term. For example, consider the 1-path *acbgih* in Fig. 3. The vertices that originate all the 1-edges are c, b, i, h, their corresponding signs are $-$, $+$, $-$ and $+$, respectively. Their product is $+$. This is the sign of the symbolic product term *cbih*.

Formally, let $\mathbf{A}$ be an $n \times n$ sparse matrix with a set of distinct m symbolic parameters $\{a_1, ..., a_m\}$, where $1 \leq m \leq n^2$. Each symbolic parameter a_i is associated with a unique pair $r(a_i)$ and $c(a_i)$, which denote, respectively, the row index and column index of a_i. A *determinant decision diagram* is a signed rooted directed acyclic graph with two terminal vertices, namely the 0-terminal vertex and the 1-terminal vertex. Each non-terminal vertex a_i is associated with

a sign, $s(a_i)$, determined by the sign rule defined by (5). It has two outgoing edges, called 1-edge and 0-edge, pointing respectively to D_{a_i} and $D_{\overline{a}_i}$. A determinant decision graph having root vertex a_i denotes a matrix determinant D defined recursively as

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

Here $s(a_i)D_{a_i}$ is the *cofactor* of D with respect to a_i, D_{a_i} is the *minor* of D with respect to a_i, $D_{\overline{a}_i}$ is the *reminder* of D with respect to a_i, and operations are algebraic multiplications and additions.

4. Vertex Ordering

A key problem in many decision diagram applications is how to select a vertex ordering, since the size of the resulting decision diagram strongly depends on the chosen ordering. In this section, we describe an efficient heuristic for selecting a good vertex ordering.

Suppose that $\mathbf{A}$ is an $n \times n$ matrix with m non-zero elements (entries, or symbols). The vertex ordering problem is how to *label* all the nonzero elements in $\mathbf{A}$ using integers 1 to m so that the resulting DDD constructed with the chosen order has a small size. As stated in Section 2.1, those elements labeled by smaller integers will appear close to the leaves, or the bottom, of the DDD, and the root is labeled by m.

We propose a vertex ordering heuristic, based on the refinement of a well-known strategy for Laplace expansion of a sparse matrix [7]. The basic idea is to label those columns or rows with fewer nonzero elements with larger indices. Elements in those dense rows and columns will be labeled using small indices. Intuitively, this strategy increases the possibility of DDD subgraph sharing. Figure 4 describes the proposed heuristic MATRIX_LABELING($\mathbf{A}(p,q)$) for labeling all the elements in $\mathbf{A}(p,q)$. In the algorithm, $\mathbf{A}(p-\{i\}, q-\{j\})$ denotes the matrix obtained from $\mathbf{A}(p,q)$ by removing row i and column j. We keep a global counter k. Initially k is set to 1, and $p = q = e$.

We have shown that the heuristic is optimal for a class of circuit matrices, called *band matrices*. Band matrices are matrices that have only non-zero elements at positions (i,i), $(i-1,i)$ and $(i+1,i)$. The MNA matrices for ladder networks—an important class of structures in analog design—are band matrices. For an n-by-n band matrix, the number of DDD vertices is equal to the number of non-zero elements in the matrix, that is $3n-2$, whereas the number of expanded product terms is $F(n+1)$, an $(i+1)$-th Fibonacci number. Each product term involves n symbols. To store all the product terms without considering sharing, the memory requirement is proportional to $nF(n+1)$. In Fig. 5, the number of DDD vertices is plotted against the number of product terms. Note that any symbolic manipulation using the expended form would have time complexity at best proportional to $nF(n+1)$.

```
MATRIX_LABELING(A(p, q))
1  select column j (row i) that has least non-zeros
2  s ← all the rows i (columns j) so that a_{i,j} ≠ 0
3  for all row i (column j) in s from the one
       with least non-zeros
4      if exist unlabeled elements in A(p − {i}, q − {j})
5          MATRIX_LABELING(A(p − {i}, q − {j}))
6  for all row i (column j) in s from the one
       with most non-zeros
7      if a_{i,j} has not been labeled
8          label a_{i,j} by k and then increment k
```

Figure 4: A DDD vertex ordering heuristic.

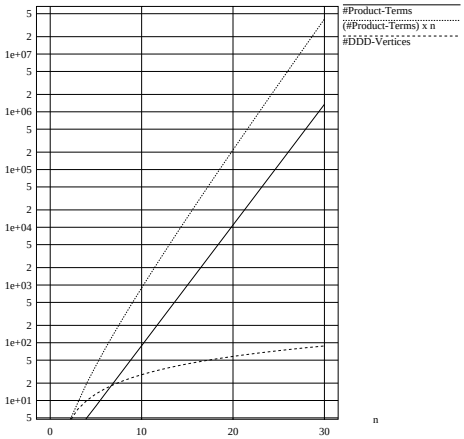

Figure 5: Band-matrix DDD sizes vs numbers of terms.

5. Construction and Manipulation of Determinant Decision Diagrams

In this section, we show that, using determinant decision diagrams, algorithms needed for symbolic analysis and its applications can be performed with the time complexity proportional to the size of the diagrams being manipulated, *not the number of 1-paths in the diagrams, i.e., product terms in the symbolic expressions.* Hence, as long as the determinants of interest can be represented by reasonably small graphs, our algorithms are quite efficient.

Table 1: Summary of Basic Operations.

Operation	Result
VertexOne()	return 1
VertexZero()	return 0
Cofactor(D,s)	return the cofactor of D wrt s
Reminder(D,s)	return the reminder of D wrt s
Multiply(D,s)	return $s \times D$
DDD_of_Matrix(**A**)	construct DDD for matrix **A**
Evaluate(D)	return the value of D

```
Cofactor(D, s)
1  if (D.top < s) return 0
2  if (D.top = s) return D_1
3  if (D.top > s) return GetVertex(D.top,
          Cofactor(D_1, s), Cofactor(D_0, s))

Reminder(D, s)
1  if (D.top < s) return D
2  if (D.top = s) return D_0
3  if (D.top > s) return GetVertex(D.top,
          Reminder(D_1, s), Reminder(D_0, s))

Multiply(D, s)
1  if (D.top < s) return GetVertex(s, D, 0)
2  if (D.top = s) return GetVertex(s, D_1, D_0)
3  if (D.top > s) return GetVertex(D.top,
          Multiply(D_1, s), Multiply(D_0, s))

DDD_of_Matrix(A(p, q))
1  if (A = 0) return VertexZero()
2  if (A = 1) return VertexOne()
3  let s be a non-zero element at row i and column j
4      return GetVertex(s,
          DDD_of_Matrix(A(p - {i}, q - {j})),
          DDD_of_Matrix(A|_{s=0}))

Evaluate(D)
1  if (D = 0) return 0
2  if (D = 1) return 1
3  return Evaluate(D_0) +
          s(D) * D.top * Evaluate(D_1)
```

A basic set of operations on matrix determinants and their implementations are summarized in Table 1. Most operations are simple extensions of subset operations introduced by Minato on ZBDDs [14]. For the clarity of the description, the steps for computing the signs associated with DDD vertices, using the sign rule defined in Section 3, are not shown. In the implementation, GetVertex$(D.top, D_1, D_0)$ is to generate (or copy) a vertex for a symbol $D.top$ and two subgraphs D_1 and D_0. A hash table is used to keep each vertex unique, and vertex elimination and sharing are managed mainly by GetVertex. A cache is used to memorize the results of recent operations, and it is referred to by every recursive call. This avoids duplicate executions for equivalent subgraphs.

Evaluation: Given a determinant decision diagram pointed to by D and a set of numerical values for all the symbolic parameters, Evaluate(D) computes the numerical value of the corresponding matrix determinant. Evaluate(D) naturally exploits subexpression sharing in a symbolic expression, and has time complexity *linear* in the size of the diagram.

Construction: Let $\mathbf{A}(e, e)$ be an n-by-n symbolic matrix, where e is a set of integers from 1 to n. DDD_of_Matrix$(\mathbf{A}(p, q))$ constructs a determinant decision diagram of a submatrix of **A** with row set $p \subseteq e$ and column set $q \subseteq e$ such that $|p| = |q|$ for a given ordering of symbolic parameters. It can be viewed as a generalized Laplace expansion procedure of matrix determinants. In line 3 of the procedure DDD_of_Matrix, a non-zero element s is picked up, and the matrix is expanded. Due to the canonical property of DDD, s can be any non-zero matrix element, and the resulting DDD is always the same. However, the most efficient construction is to use the element with the largest integer label in line 3.

Cofactor and Sensitivity: Cofactor(D, s) is to compute the cofactor of a symbolic determinant D represented by a DDD with respect to symbolic parameter s. It is perhaps the most important operation in symbolic analysis of analog circuits. For example, the sensitivity of D with respect to s is exactly its cofactor with respect to s. The network functions can be obtained by first computing some cofactors, and then combining these cofactors using Cramer's rule.

6. Experimental Results

We have tested the proposed algorithms on a set of circuits varying from RLC filters to bipolar and MOS integrated circuits, which includes

- *filter*, a cascade bandpass active filter [8],
- *miller*, a two-stage miller compensated MOS opamp [7],
- $\mu A741$, a bipolar opamp containing 26 transistors and 11 resistors, with the schematic in Fig. 6,
- *cascode*, a CMOS cascode opamp containing 22 transistors with the schematic in Fig. 7,
- *ladder7, ladder10, ladder21*, 7, 10 and 21 section cascade resistive ladder networks,
- some RLC filters, named *butter*, *rlctest*, *vcstest*, *ccstest* and *big*.

For nonlinear integrated circuits, DC analyses are performed using SPICE, and the resulting small signal models are used for symbolic analysis.

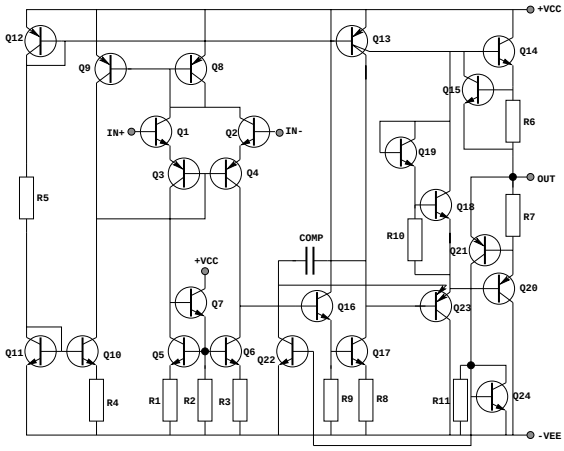

Figure 6: Circuit schematic of bipolar $\mu A741$.

Table 2 describes the statistics of all the test circuits and the resulting DDD sizes. Columns 2 to 6 describe respectively the number of nodes in each circuit, the number of non-zero elements in each circuit matrix, the number of product terms in each symbolic determinant, and the number of vertices in the resulting DDD representation without and with the use of the ordering heuristic described in Fig. 4. For each circuit, the number of DDD vertices without ordering is the average of that of ten randomly generated orderings.

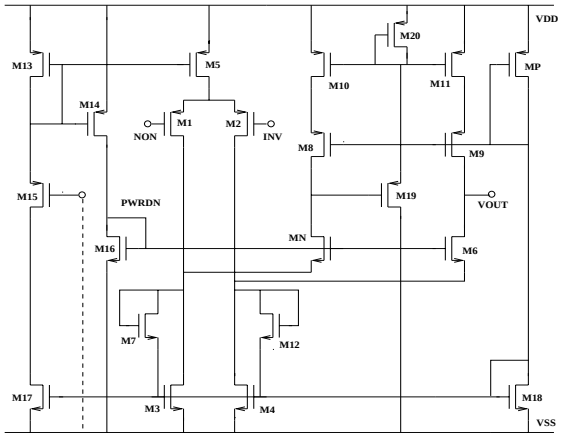

Figure 7: Circuit schematic of MOS cascode Opamp.

From Table 2, we can make several observations:

1. Ordering leads to DDDs significantly smaller than that of non-ordering. For ladder networks *ladder7*, *ladder10*, *ladder21*, the numbers of DDD vertices are exactly the numbers of non-zero matrix elements.

Table 2: DDD sizes with/without vertex ordering.

ckt name	#nod	#ele	#term	#vertex (no-order)	#vertex (order)
miller	7	23	37	109	55
butter	8	19	21	102	19
ladder7	9	22	34	135	22
rlctest	10	37	352	877	130
ccstest	10	35	260	746	109
ladder10	11	31	144	637	31
vcstest	11	37	402	1024	104
cascode	15	77	232363	372579	2110
filter	20	46	82	939	52
ladder21	23	64	28657	92783	64
$\mu A741$	24	89	374884	$-*$	7431
big	33	101	7.31×10^6	–	905

-*: out of memory.

2. For a small circuit, the number of DDD vertices may be greater than the number of product terms. This is not surprising, since the number of DDD vertices without exploiting sharing would be the number of product terms times the number of symbolic parameters in each product term.
3. For a large circuit, the number of DDD vertices can be several orders of magnitude smaller than the number of product terms.
4. Further, the difference becomes more dramatic with the increase of the circuit size.

We compare our program with *ISAAC* [7] and *Maple-V* [4]. *ISAAC* is a well-known special-purpose symbolic analyzer designed for analog integrated circuits. *Maple-V* is a general-purpose mathematic package capable of solving linear equations symbolically. To use *Maple-V*, we use our program to set up the circuit equations and then feed the circuit matrices to *Maple*. All data are obtained using a SUNsparc 20 with 32M memory. The results are described in Table 3. We can observe that our program runs significantly faster than both *ISAAC* and *Maple-V*, and uses much less memory. For slightly large circuits, both *ISAAC* and *Maple-V* ran out of memory.

Table 3: The DDD algorithm vs ISAAC and Maple-V.

ckt	ISAAC		Maple-V		DDD	
	CPU(s)	Mem	CPU(s)	Mem	CPU(s)	Mem
butter	0.61	493k	0.03	10.9k	0.033	8.1K
ladder7	1.05	980k	0.63	250k	0.033	8.1k
miller	0.49	640k	0.6	250k	0.066	24.5k
rlctest	$-*$	–	17.3	15.2M	0.10	49.1k
vcstest	–	–	99.9	20.1M	0.10	73.1k
ladder21	–	–	–	–	0.066	32.7k
cascode	–	–	–	–	7.62	4.9M
$\mu A741$	–	–	–	–	9.13	5.2M
big	–	–	–	–	4.01	2.8M

–*: out of memory.

To verify the correctness of our symbolic analysis algorithms, we have implemented a frequency-domain circuit simulator based on DDD evaluation. For all the test circuits, our simulator produced the exactly same results as SPICE did.

7. Conclusions

In this paper, a graph representation, called determinant decision diagrams (DDDs), for symbolic matrix determinants is proposed and symbolic analysis algorithms for analog circuits are presented. Unlike previous approaches based on either the expanded form or the nested form representations of symbolic expressions, DDD-based symbolic analysis exploits the sparsity and sharing in a canonical manner. With an efficient vertex ordering heuristic, we showed that the number of vertices in a DDD is quite small—usually several orders of magnitude smaller than the number of product terms in the expanded form. This enables the exact analysis of such large analog circuits as $\mu A741$ for the first time. In contrast to the nested form, our representation is unique. Symbolic analysis algorithms such as determinant evaluation, driving cofactor computation, network function construction, and sensitivity calculation can be performed in time complexity proportional to the number of DDD vertices.

References

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation", *IEEE Trans. Computer*, pp. 677–691, 1986.

[2] R. Carmassi, M. Catelani, G. Iuculano, A. Liberatore, S. Manetti, and Marini, "Analog network testability measurement: a symbolic formulation approach", *IEEE Trans. Instrumentation and Measurement*, vol. 40, pp. 930–935, Dec. 1991.

[3] S.-M. Chang, J.-F. MacKey, and G. M. Wierzba, "Matrix reduction and numerical approximation during computation techniques for symbolic analog circuit analysis", pp. 1153–1156 in *Proc. IEEE Int. Symp. Circuits and Systems*, 1992.

[4] B. W. Char, *et. al.*, *Maple V: Language Reference Manual*, Springer-Verlag, 1991.

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

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

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

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

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

[10] Jer-Jaw Hsu and Carl Sechen, "DC small signal symbolic analysis of large analog integrated circuits", *IEEE Trans. Circuits and Systems*, vol. 41, no. 12, pp. 817–828, Dec. 1994.

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

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

[13] W. J. McCalla and D. O. Pederson, "Elements of computer-aided analysis", *IEEE Trans. Circuit Theory*, vol. 18, pp. 14–26, 1971.

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

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

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

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

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

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

[20] Z. You, E. Sánchez-Sinencio, and J. P. de Gyvez, "Analog system-level fault diagnosis based on a symbolic method in the frequency domain", *IEEE Trans. Instrumentation and Measurement*, vol. 44, no. 1, pp. 28–35, Jan. 1995.

[21] 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, August 1996.