Efficient DDD–based Symbolic Analysis of Large Linear Analog Circuits

Wim Verhaegen
wim.verhaegen@esat.kuleuven.ac.be

Georges Gielen
georges.gielen@esat.kuleuven.ac.be

Katholieke Universiteit Leuven, Dept. ESAT–MICAS
Kasteelpark Arenberg 10, B–3001 Leuven, Belgium

ABSTRACT

A new technique for generating approximate symbolic expressions for network functions in linear(ized) analog circuits is presented. It is based on the compact determinant decision diagram (DDD) representation of the circuit. An implementation of a term generation algorithm is given and its performance is compared to a matroid-based algorithm. Experimental results indicate that our approach is the fastest reported algorithm so far for this application.

1. INTRODUCTION

The symbolic analysis of analog circuits has a long standing history [1, 2, 3]. Applications range from the calculation of symbolic expressions of network functions to the simplified behavioral modeling of the circuit in order to trade some accuracy loss for a substantial circuit simulation speed gain. From the designer's point of view, the symbolic expressions for the circuit's network functions are most interesting as they provide insight into many circuit characteristics, especially if they are simplified so that they only contain the dominant circuit parameters.

The purpose of linear symbolic analysis is thus to generate expressions in the form

$$\widetilde{H}(s) = \frac{\sum_0^{\widetilde{k}} \widetilde{a}_i s^i}{\sum_0^{\widetilde{l}} \widetilde{b}_j s^j} \quad (1)$$

which approximate an exact network function

$$H(s) = \frac{\sum_0^{k} a_i s^i}{\sum_0^{l} b_j s^j} \quad (2)$$

with $\widetilde{k} \leq k$, $\widetilde{l} \leq l$ and k and l the order of the numerator resp. denominator. The approximation must be as compact as possible, while respecting some error criteria:

$$g(H(s)) - \Delta g_- \leq g(\widetilde{H}(s)) \leq g(H(s)) + \Delta g_+ \quad (3)$$

with $s = 2j\pi f$ for $f_{min} \leq f \leq f_{max}$. Typical error control functions $g(\cdot)$ are magnitude $A_{db}(\cdot) = 20 \log \|\cdot\|$ and phase $\phi(\cdot) = \angle\cdot$.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

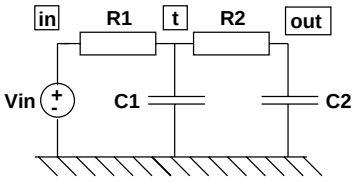

Figure 1: Illustrative Second-order RC circuit

Many tools have been developed for the automated generation of symbolic network functions, which becomes very cumbersome if not unfeasible to do by hand for increasing circuit sizes [3, 4, 5, 6, 7, 8, 9]. Most of these tools calculate the output of a given circuit in a simplified form based on relative size information of the circuit parameters. The resulting expressions are therefore only valid within a limited range of design points.

In this paper we present an alternative to the expression–based approach, based on an exact representation of determinants as a graph or determinant decision diagram (DDD), a concept which was introduced to the linear circuit analysis domain in [10, 11, 12]. Instead of calculating the circuit output explicitly, the Laplace expansion of the circuit matrix minor is modeled in a DDD, which is discussed in section 2. This setup needs to be done only once for a given circuit topology, and is similar to the MTDD–approach followed in [12]. In section 3 we present an additional feature of our approach: an efficient incremental term–generation algorithm. This term–generation tool makes DDD's a useful data structure for approximative symbolic analysis, and a contender in the symbolic–analysis field for the matroid–based approach, that is presented a.o. in [13]. Section 4 gives a comparison of experimental results for the DDD– based and matroid–based approached, and our new approach turns out to be the fastest term generation algorithm at present. Finally, conclusions are drawn in section 5.

2. DDD SETUP

The DDD representation used in this paper is based on the MNA representation of the linearized circuit. For a circuit with $n - m$ nodes and m independent voltage sources, the circuit equations are written as

$$\tilde{\mathbf{Y}}_{(n-1)\times(n-1)} \mathbf{V}_{n-1} = \tilde{\mathbf{I}}_{n-1} \quad (4)$$

with $\tilde{\mathbf{Y}}$ the modified admittance matrix, $\mathbf{V}$ the node voltage and source current vector and $\tilde{\mathbf{I}}$ the input source vector containing independent current and voltage sources.

For example, the circuit in Figure 1, which will be used for illustrating the presented algorithm, has the following MNA represen-

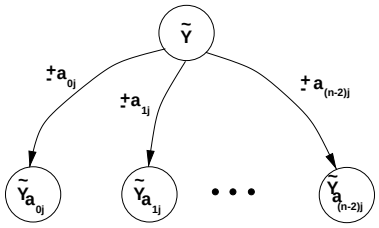

Figure 2: Graph representation of a step in the Laplace expansion

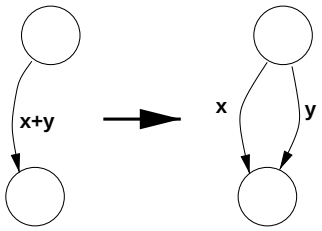

Figure 3: Substitution of matrix entries by its components in a DDD

tation:

$$\tilde{\mathbf{Y}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & sC_2 + \frac{1}{R_2} & \frac{-1}{R_2} & 0 \\ \frac{-1}{R_1} & \frac{-1}{R_2} & sC_1 + \frac{1}{R_1} + \frac{1}{R_2} & 0 \\ \frac{1}{R_1} & 0 & \frac{-1}{R_1} & -1 \end{bmatrix} \quad (5)$$

$$\mathbf{V} = \begin{bmatrix} v_{in} & v_{out} & v_t & i_{V_{in}} \end{bmatrix}^T \quad (6)$$

$$\tilde{\mathbf{I}} = \begin{bmatrix} V_{in} & 0 & 0 & 0 \end{bmatrix}^T \quad (7)$$

Starting from the circuit representation (4), any linearized network function can be calculated by applying Kramer's rule. This rule states that every unknown variable v_j of the node voltage vector $\mathbf{V}$ is the ratio of the determinants $\tilde{Y}_j$ and $\tilde{Y}$, where $\tilde{Y}_j$ is derived from $\tilde{Y}$ by substituting the j'th column by the source vector $\tilde{\mathbf{I}}$. The determinants themselves are calculated through Laplace expansion, which is explained below. Two DDD's are thus needed to represent a network function.

The Laplace expansion of a determinant $\tilde{Y}$ along a column is given by the expression

$$\tilde{Y} = \sum_{i=0}^{n-1} (-1)^{i+j} a_{ij} \tilde{Y}_{a_{ij}} \quad (8)$$

A similar expansion rule along a row is also possible, but we will limit ourselves to expansion along columns without loss of generality. The minors $\tilde{Y}_{a_{ij}}$ in equation (8) are also determinants and can be further expanded using the same rule. Recursive application would thus finally result in a flat expression for $\tilde{Y}$ containing only the basic entries of $\tilde{\mathbf{Y}}$.

Every expansion step described by equation (8) is graphically represented as shown in Figure 2, where each vertex corresponds to a minor and each edge to an entry of the admittance matrix. The recursive application of rule (8) leads to a nested graphical representation. When the Laplace expansion rule is executed recursively on a determinant, one finally ends up with only 1×1 minors, which evaluate to the only element they contain. For a translation to the graph representation, one can think of an additional Laplace expansion of the minor resulting in the value of the only element multiplied with a 0×0 minor, which value must then evidently equal 1. In the graph representation this is modeled by terminating the graph with a *1–vertex*.

There is one degree of freedom left in the ordering of the columns during the Laplace expansion. Two logical approaches are the *static column selection scheme*, in which the columns are ordered according to sparsity prior to the first column expansion, and the *greedy column selection scheme*, in which the most sparse column is selected prior to each column expansion. Both column selection schemes have been implemented and evaluated, and it turns out that the speedup of the greedy column selection always compensated the small overhead involved. We therefore use the greedy selection scheme for all examples shown in this paper.

Since the MNA circuit matrix is sparse in general, many entries a_{ij} in (8) equal 0, and the corresponding edges can be removed from the graphical representation. This can lead to vertices other than the 1–vertex with no outgoing edges. These vertices represent zero minors and are removed from the graph in a bottom–up recursive *zero–suppression* step. A slightly modified version of the algorithm with linear complexity described in [14] is used to this purpose. The zero–suppression leads to a compact graph, which lowers the execution time of many of the algorithms described below.

The DDD representation of the determinant is then obtained by one further operation demonstrated in Figure 3: an entry of the modified admittance matrix generally consists of a sum of components, which are either the symbolic admittance of a circuit component, an input source, or plus or minus one. In order to relate every edge in the graph representation to a single circuit element, source or constant, the entries are decomposed into their components, resulting in parallel edges.

In order to represent a network function, a second DDD representing $\tilde{Y}_j$ is constructed using the same rule. It is easily spotted that $\tilde{Y}_j$ has many entries in common with $\tilde{Y}$. In order to maximize the reuse of DDD vertices, we re–order the columns in $\tilde{Y}$ and $\tilde{Y}_j$ so that $j = 0$ and start the Laplace expansion along the 1st column (i.e. the column with index 0). The analyzed network function is then given by the ratio of the expressions represented by $DDD(\tilde{Y})$ and $DDD(\tilde{Y}_0)$.

For the example circuit of Figure 1, the DDD representation resulting from the Laplace expansion of the numerator and denominator determinant and subsequent zero suppression is given in Figure 4. The conductances in Figure 4 are derived from the resistances in Figure 1: $G_1 = 1/R_1$ and $G_2 = 1/R_2$.

3. TERM GENERATION ALGORITHM

In order to generate an approximate expression in the format (1) from the exact DDD, some additional graph transformations are needed. The terms for each coefficient $\widetilde{a}_i$ and $\widetilde{b}_j$ are to be generated in order of decreasing magnitude, and the number of generated terms per coefficient is controlled by an error control algorithm. An overview of common error control techniques is given in [15], so we will not go into the details here. A common characteristic of all these algorithms is that individual terms need to be generated per coefficient. DDD's constructed as explained in section 2 represent the numerator or denominator of a network function. A transformation of these DDD's to a DDD-representation of all coefficients is thus needed. The construction of these *Element-based Coefficient Diagrams* or ECD's, which resemble the multi–terminal decision diagram (MTDD) representation of [12], is explained below.

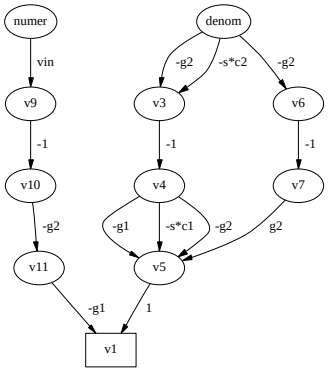

Figure 4: DDD representation of $v_{\mathbf{out}}$ for the example circuit

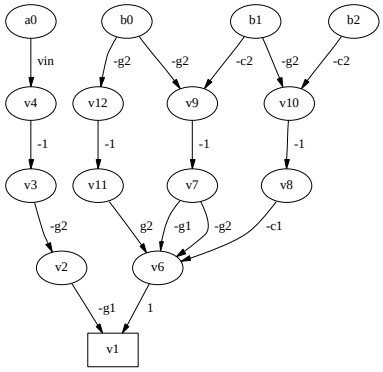

Figure 5: ECD representation of v_{out} for the example circuit

3.1 Setup of ECD representation

Before explaining the transformation algorithm, the following definitions are needed:

DEFINITION 1 (EDGE OF ORDER q). *A DDD edge whose value is of the form $K * s^q$ with K no function of s, is called an edge of order q.*

DEFINITION 2 (SUBVERTEX OF ORDER m). *A DDD vertex which value is the coefficient of s^m of a given DDD vertex, is the subvertex of order m of the latter vertex.*

The coefficient graphs are thus subvertices of the top vertex of the original DDD. They are derived from the original DDD through recursive application of the *subvertex construction rule*: the subvertex v^*_m of order m of a given DDD vertex v^* has the same edges as v^*, with this difference that an edge of order l points to the subvertex of order $m - l$ if the latter exists or is removed from the set of edges otherwise.

The ECD's for the denominator coefficients of the circuit in Figure 1 are shown in Figure 5. Note that the coefficient graphs are overlapping, resulting in a relatively low number of subvertices.

The time and space complexity of the ECD's is linear with the size of the original DDD. Their construction can be executed by applying the subvertex construction rule to each DDD vertex in a bottom-up sequence.

3.2 Symbolic term generation

Given that the root vertices of the ECD represent the coefficients a_i and b_j, we extract the dominant terms from each root vertex using the following algorithms. Each ECD edge is assigned a *next term index* **nti** which is initialized to 0, and a cached next term which is constructed with the algorithm given here in pseudo code:

```
Next-Available-Term(e) ::=
  term = Get-Term(e.toVertex,e.nti);
  if(term ≠ No-Term) {
    Update-Term(term,e) ;
    increment e.nti;
    return(term);
  }
  else return(No-Term);
```

This requires in turn that a term can be requested of an ECD vertex. The terms for a vertex are stored in a **generated-terms** list and are generated using the following set of algorithms:

```
Get-Term(v,index) ::=
  while(not exists v.generated-terms[index]) {
    term = Generate-Next-Term(v);
    if(term == No-Term) {
      return(No-Term) ;
    }
  }
  return(v.generated-terms[index]) ;
```

```
Generate-Next-Term(v) ::=
  if(v == One-Vertex
    AND v.generated-terms is empty) {
    push(v.generated-terms, One-Term) ;
  }
  else {
  START-SEARCH:
    find em ∈ v.outedges such that
      ∀ e ∈ v.outedges :
        | em.term | ≥ | e.term | ;
    tm = em.term ;
    if(tm == No-Term) {
      return(No-Term) ;
    }
    else if(tm == Invalid-Term) {
      Remove-Invalid-Path(em)
      goto START-SEARCH
    }
    else {
      push(v.generated-terms, tm) ;
      return(tm) ;
    }
  }
```

The mutual recursion between these algorithms makes that we descend the ECD with one edge at the time until we arrive at the terminal vertex. By adding caching for generated terms that are not immediately used to ***Next-Available-Term*** (which is not explicitly shown in the pseudo code), only the terms generated along the followed path need to be updated. Since the depth of the ECD equals the depth of the original DDD, i.e. n, the generation of a single symbolic term is of complexity $O(n)$.

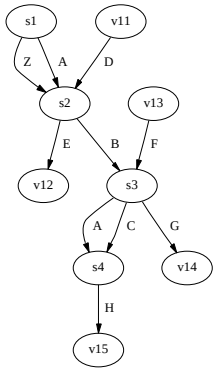

Figure 6: Illustratory part of an ECD before restructuring

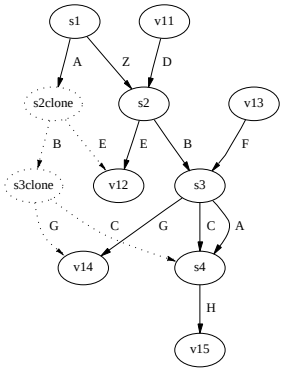

Figure 7: Illustratory part of an ECD after restructuring. The subpath A-B-A has been removed.

However, the generated terms are not always non–cancelling. This is already the case for the small example circuit. A short inspection of its ECD (Figure 5) reveals that the terms G_2^2 and $-G_2^2$ will be generated for coefficient b_0. The terms cancel out in the final result, but they both need to be generated, which results in a waste of memory and CPU time. This problem becomes serious for large circuits, where those cancelling terms form a major part of all generated terms, which severely slows down the generation of non–cancelling terms.

The cancelling–terms problem however can be detected early, as they always contain a square of a circuit element when using the MNA circuit representation. We therefore mark a term as nonvalid as soon as we find a square in the Update-Term() function invoked from Next-Available-Term():

```
Update-Term(term,e) ::=
  if(e is numeric) {
    multiply term.numcoeff with value(e) ;
  }
  else { // e is symbolic
    if(term contains symbol(e)) {
      term = Invalid-Term ;
    }
    else {
      add symbol(e) to term ;
    }
  }
```

In the following section we show a method that avoids the generation of most cancelling terms as soon as the above condition pops up. The use of this method will thus reduce the complexity of generating a non–cancelling symbolic term back to $O(n)$.

3.3 Optimization of the ECD representation

The reason for the generation of the cancelling terms is that two edges carrying the same symbol are present in a sequence of edges (or subpath) of the ECD. Their presence renders each term corresponding to a path containing the subpath as cancelling. So it is only logical that when we find such a subpath running into a cancelling term, we eliminate any further term generation along that subpath.

It is however not possible to just remove the invalid subpath from the ECD. This would also remove paths which hook into the subpath after the first invalidating edge or split of the subpath before the last invalidating edge. In order to keep those paths, the ECD vertices along the path and their outgoing edges need to be duplicated, and some edges need to be reconnected or removed.

This is illustrated with the ECD part shown in Figure 6. The ECD contains the invalid subpath $A - B - A$ running from vertex $s1$ over $s2$ and $s3$ to $s4$. The vertices $v11$ to $v15$ have been added to illustrate the interaction with the rest of the ECD. Figure 7 shows the same ECD part after restructuring. The subpath has been cut at the 1st index and the cut edge (A) has been reconnected to a path consisting of vertices cloned of $s2$ and $s3$. $s2clone$ and $s3clone$ have identical outedges as $s2$ and $s3$, except of course for the A-edge of $s3clone$, which has been removed. Comparison of these two graphs reveals that they are identical, except for the path $A - B - A - H$ which is no longer present in the restructured ECD.

The implementation of the restructuring algorithm ***Remove-Invalid-Path***, which is invoked from ***Next-Available-Term***, is not too difficult and therefore omitted here.

The ECD restructuring in itself is not a very CPU intensive task, and does not need to be applied too often, as will be illustrated by the experimental results in section 4. The overhead induced by the restructurings is therefore negligible when compared to the considerable speedup that is achieved for large circuits in this way.

4. EXPERIMENTAL RESULTS

The ECD–based symbolic term generation algorithm has been implemented in C++ and evaluated on an Ultra–SPARC 30 workstation. The results are shown in Table 1, and are compared to the results of SADAMAC, a matroid–based term generation tool which uses the algorithm described in [13]. The following symbols are used in Table 1:

- n : the number of circuit nodes;
- b : the number of circuit branches (or small signal elements);
- $S(\cdot)$: the setup time for a data structure [secs];
- $V(\cdot)$: the number of vertices in a graph;
- T : the number of terms generated for this experiment;
- R : the number of ECD restructurings during term generation;
- TS : the term generation speed [terms/sec];
- TS^* : the term generation speed achieved by SADAMAC [terms/sec].

Table 1: Experimental results of the ECD-based term generation, compared to the matroid-based SADAMAC

Circuit	n	b	S(DDD)	V(DDD)	S(ECD)	V(ECD)	T	R	TS	TS^*
maze005	5	12	< 0.01	38	0.01	38	673	33	2493	15000
maze009	9	38	0.12	522	0.04	522	2002	27	1944	NA
maze013	13	80	3.42	8206	0.93	8206	2002	13	674	NA
maze017	17	138	92.59	131090	23.05	131090	2002	13	36	NA
ladder010	11	21	< 0.01	36	< 0.01	144	6280	88	2418	2631
ladder030	31	61	0.04	96	0.02	1024	28119	166	1555	63
ladder050	51	101	0.08	156	0.06	2704	48199	226	993	91
ladder070	71	141	0.18	216	0.12	5184	68279	286	679	85
highspeed	7	49	< 0.01	31	0.01	119	9359	41	2463	454
bicmos-miller	10	32	0.01	115	0.02	483	14894	1164	2092	346
bicmos-ota	16	69	0.27	1329	0.24	7223	31031	1885	1511	191
741	25	118	32.90	105206	28.41	912995	100	NA	25	10

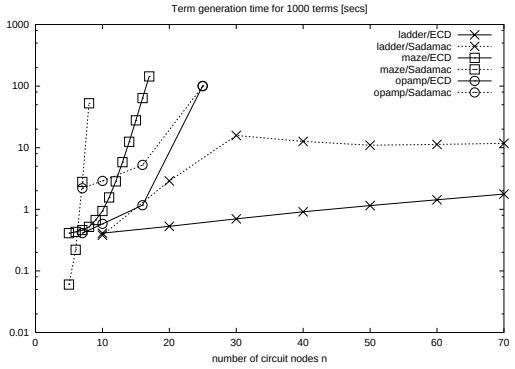

Figure 8: Comparison of the performance of ECD-based and matroid-based symbolic term generation

Table 1 includes results for

- maze circuits, which are networks with an impedance between each pair of nodes;
- RC ladder circuits, like the illustrative circuit ladder002 in Figure 1. These circuits have tridiagonal matrices, which renders the DDD setup a linear function of n as is explained in [11];
- and opamp–like circuits of increasing complexity.

Note that for some maze circuits, SADAMAC did not generate any results. Term generation was limited to a fixed number of terms per coefficient, with an exception for the complex 741 circuit, for which only a few coefficients were generated.

Note also that the setup time of the DDD and ECD graphs is very large in comparison with the time needed to generate a term. This is explained by the fact that the DDD setup time is proportional to the number of minors and thus has complexity $O(n!)$, while the term generation time is proportional to the depth of the ECD, which equals n. For a performance evaluation of a typical symbolic term generation which is limited to a few thousand terms, one must thus take into account S(DDD) and S(ECD) as well as TS.

In order to compare the ECD–based and matroid-based symbolic term generation methods, the total generation times for 1000 terms including setup overhead are plotted in Figure 8. SADAMAC performs horribly on the maze circuits — it even does not generate any results for maze circuits with more than 8 nodes — while our approach generates results in a reasonable time up to a maze circuit with 17 nodes. Our approach also makes better use of the intrinsic linear complexity of the ladder circuit, performing a factor of 10 better on ladder circuits with over 20 nodes. For the opamp circuits, our approach is a factor of 4 faster than SADAMAC, with the exception of the 741 circuit where the performance is only equal due to the DDD setup time. Keep in mind however that the overhead per term decreases drastically when more terms need to be generated, which is likely to be the case for the 741 circuit.

It is thus striking that our approach outperforms SADAMAC, while the intrinsic DDD setup complexity of $O(n!)$ is larger than the in [13] reported complexity for generating a term using the matroid-based algorithm, which is $O(Kbn^3)$. This is not caused by the sparsity of the used circuits — our experiments have shown that this reduces the ECD complexity only slightly — but by an intrinsic deficiency of the matroid based approach. The 3–matroid intersections are found by generating the elements of a 2–matroid intersection and checking whether they are also an element of the third matroid. The acceptance ratio of this final test can be small, which renders the matroid–based algorithm less efficient.

It can therefore be concluded that our algorithm is the most efficient symbolic term generation algorithm presented so far.

5. CONCLUSIONS

A new term generation technique for linear symbolic analysis based on the determinant decision diagram (DDD) data structure has been presented. The DDD representation has the advantage that several manipulations are of complexity proportional to the depth or size of the DDD. A cancellation–free per–coefficient symbolic term generation algorithm that employs the low DDD manipulation complexity has been implemented. Experimental results indicate that this algorithm performs better than the fastest matroid–based algorithm reported previously, which indicates that DDD-based term generation is possibly the best solution in the field of linear(ized) symbolic analysis of analog circuits.

6. REFERENCES

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

[2] P. M. Lin, *Symbolic Network Analysis*. Elsevier, 1991.

[3] F. V. F. Fernandéz, A. Rodríguez-Vázquez, J. L. Huertas, and G. G. E. Gielen, *Symbolic analysis techniques – Applications to Analog Design Automation*. IEEE Press, 1998.

[4] G. G. E. Gielen, H. Walscharts, and W. Sansen, "ISAAC: a symbolic simulator for analog integrated circuits," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1587–1597, Dec. 1989.

[5] S. Seda, M. Degrauwe, and W. Fichtner, “Lazy-expansion symbolic expression approximation in SYNAP,” in *Proceedings IEEE/ACM International Conference on Computer Aided Design*, pp. 310–317, 1988.

[6] R. Sommer, E. Hennig, G. Dröge, and E.-H. Horneber, “Equation-based symbolic approximation by matrix reduction with quantitative error prediction,” *Alta Frequenza-Rivista di Elettronica*, vol. 6, Dec. 1993.

[7] M. M. Hassoun and K. S. McCarville, “Symbolic analysis of large-scale networks using a hierarchical signal flowgraph approach,” *J. Analog Integrated Circuits and Signal Processing*, no. 3, pp. 31–42, 1993.

[8] F. V. F. Fernandéz and A. Rodríguez-Vázquez, “Interactive AC modeling and characterization of analog circuits via symbolic analysis,” *J. Analog Integrated Circuits and Signal Processing*, vol. 1, pp. 183–208, Nov. 1991.

[9] M. Pierzchała and B. Rodanski, “Symbolic analysis of large-scale networks by circuit reduction to a two-port,” in *Proceedings Symbolic Methods an Applications in Circuit Design Workshop*, Oct. 1996.

[10] C.-J. Shi and X. Tan, “Symbolic analysis of large analog circuits with determinant decision diagrams,” in *Proceedings IEEE/ACM International Conference on Computer Aided Design*, Nov. 1997.

[11] C.-J. Shi and X. Tan, “Canonical symbolic analysis of large analog circuits with determinant decision diagrams,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 1–18, Jan. 2000.

[12] T. Pi and C.-J. Shi, “Multi-terminal determinant decision diagrams: A new approach to semi-symbolic analysis of analog integrated circuits,” in *Proceedings Design Automation Conference*, pp. 19–22, June 2000.

[13] P. Dobrovolny, “Design tool for symbolic analysis based on the matroid intersection theory,” in *Proceedings IEEE International Symposium on Circuits and Systems*, 1998.

[14] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, pp. 677–691, 1986.

[15] W. Daems, W. Verhaegen, P. Wambacq, G. G. E. Gielen, and W. Sansen, “Evaluation of error-control strategies for the linear symbolic analysis of analog integrated circuits,” *IEEE Transactions on Circuits and Systems-I*, vol. 46, pp. 594–606, May 1999.