Analog Integrated Circuits and Signal Processing 3, 31–42 (1993)

Symbolic Analysis of Large-Scale Networks Using a Hierarchical Signal Flowgraph Approach

MARWAN M. HASSOUN AND KEVIN S. McCARVILLE

Department of Electrical Engineering and Computer Engineering, Iowa State University, Ames, IA 50011

Abstract. This paper presents an approach suitable for the symbolic analysis of large-scale active networks. The method depends on the partitioning of the network into smaller networks which are then symbolically analyzed noniteratively using the Mason's signal flowgraph models of each partition. The resulting solutions, which are reduced signal flowgraphs (transfer functions) for the subnetworks are then hierarchically combined to produce the final solution or solutions (transfer functions) for the entire system. The advantage of such an approach is the reduction in the number of symbolic terms compared to the conventional approaches, and the ability to analyze hybrid systems consisting of electrical and nonelectrical parts. The result of the analysis is a series of equations that have an upward hierarchical dependency on each other.

1. Introduction

Symbolic analysis of linear networks involves finding the relationship between an input and an output variable in the frequency domain in the form of a transfer function:

$$H(s, \mathbf{X}) = \frac{N(s, \mathbf{X})}{D(s, \mathbf{X})} \tag{1}$$

where $N(s, \mathbf{X})$ and $D(s, \mathbf{X})$ are polynomials in s and the symbolic network variables $\mathbf{X}$. The idea of symbolic analysis of large-scale networks has had severe limitations in the past. The algorithms have been limited to small size networks in the range of 50 nodes or less [1, 2]. The main reason for this limitation has been the exponential growth in the number of terms in equation (1). Two recently proposed methods [2, 3] have overcome this limitation by replacing the conventional transfer function in equation (1) by an alternate evaluation scheme called the *sequence of expressions* where the growth in number of terms is reduced to a linear or quadratic rate [2]. Also introduced is the idea of partitioning, signal flowgraph partitioning in [3] and direct network partitioning in [2, 4]. Although other symbolic analysis programs exist (e.g., ISSAC [5], ASAP [6], BRAINS [7], EASY [8]) comparisons in this paper are only made to the two methods mentioned earlier because they are hierarchical methods which utilize partitioning and a sequence of expressions.

This paper describes a method that makes use of both partitioning and a sequence of expressions to analyze large-scale networks. The method uses the Mason's signal flowgraph (MSFG) as the engine for the analysis mainly because it is a more general method of representing both electrical and nonelectrical systems versus the methods mentioned above that are more suitable for electrical networks. Further advantages of using the Mason's signal flowgraph are listed in Section 7.

The analysis process consists of the following parts:

1. *Partitioning of the network*: This can be done on one of two levels: on the network level by using circuit partitioning algorithms like the ones described in [4] and used herein, or can be done on the signal flowgraph level. The second would require the creation of the signal flowgraph for the entire network and then attempting the partitioning. Network partitioning is a natural choice for large circuits since they are inherently composed of hierarchically combined subcircuits. These subcircuits are natural partitions on the network level but not necessarily on a signal flow graph level. The discussion of a particular partitioning scheme is beyond the scope of this paper. Figure 15 shows a recursive hierarchical partitioning model.

2. *Analysis of the partitions*: By using Mason's signal flowgraph to symbolically analyze the partitions. The goal here is to produce a representation of the partition in terms of its inputs and outputs only. The inputs and outputs of a partition are defined as its tearing nodes, in addition to the input and output variables of the entire network. So, for a network with two inputs and two outputs, the final reduced signal flowgraph will be as illustrated in figure 1. All internal nodes of the MSGF have been suppressed using the MSGF reduction rules.

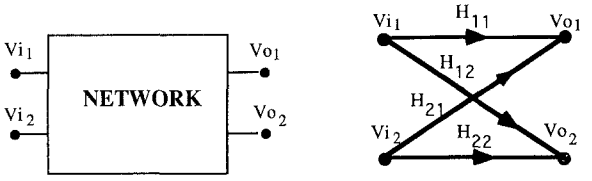


Fig. 1. Reduced signal flowgraph model.

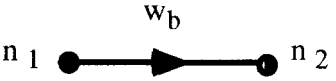


Fig. 2. Branch b.

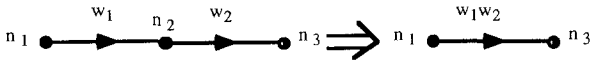


Fig. 3. Series suppression of node n_2.

Fig. 4. Parallel reduction of branches 1 and 2.

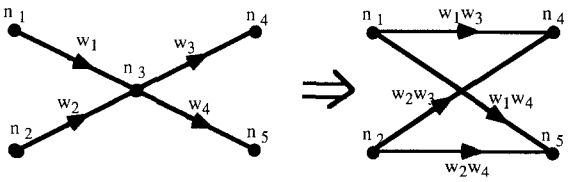


Fig. 5. General suppression of node n_3.

3. *Hierarchical recombination of the solutions*: A new partition is created by combining two partitions with at least one common tearing node. Each common tearing node becomes an internal node of the new partition and is reduced using the MSGF reduction rules (figure 1). The recombination process follows the partitioning model (figure 15) hierarchically and recursively up the tree.

2. MSGF Reduction Rules

The systematic node suppression rules utilized above for a signal flowgraph can be achieved by using the following four simple rules:

1. Series suppression of a node (figure 3)
   A series suppression is a specific case of the general suppression with only one branch entering and one branch exiting a node. A new branch is created from node 1 to node 3 with a reduced weight.
2. Parallel reduction of branches (figure 4)
   Any parallel branches are reduced to one remaining branch with a combined weight. A reduction of this type is performed on parallel feedback loops if any exist.
3. General suppression of a node (figure 5)
   A general suppression performs a series suppression from every branch entering to every branch leaving that node. Once all series suppressions for a given node are complete, the node and its incident branches are removed.
4. Self-loop reduction (figure 6)
   A self-loop adjusts the weights of all branches entering the node. The feedback branch is then deleted.
5. Suppression of a node with no outgoing branches
   This corresponds to an I/O variable not requested by the analysis. The suppression is done by simply deleting the node.

3. The Hierarchical Algorithm (MASSAP)

The implementation of the flowgraph analysis algorithm was named MASSAP (Mason's symbolic analysis program). MASSAP is part of a more general symbolic analysis program called SCAPP (symbolic circuit analysis program with partitioning) [2, 9]. SCAPP gives the user the option of performing an automatic partitioning on either the signal flowgraph level or the network level. It also accepts user defined partitions (subcircuits) and treats them as natural partitions for the network. The package also gives a choice between two methods of analysis: signal flowgraph analysis (MASSAP) or direct network analysis based on a modified nodal approach [2, 9]. Section 6 illustrates comparisons between the two methods.

3.1. Program Operation

The operation of the program starts by parsing the input deck and creating the data structures to completely describe the circuit. If requested, the circuit is partitioned into subcircuits.

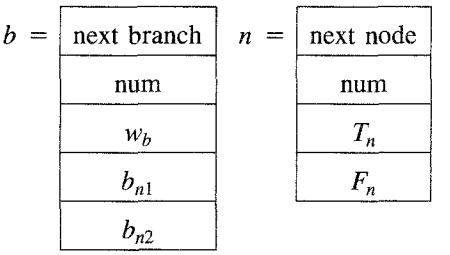
Subcircuit analysis begins by visiting all the nodes included in the subcircuit. For all nodes in the subcircuit, any parallel or feedback reductions are performed, and for all internal nodes any general reductions are also performed. These operations produce new branch

structures. The reduced nodes and old branches data structures are then removed and rather than their memory deallocated from the program, they are stored in a linked list of deleted elements. This list is then reused for new structures that need to be created by the program. So MASSAP has a form of local memory management in order to reduce the execution time due to costly system memory allocation and deallocation calls. Operations are performed in the order stated above because parallel and self-loop reductions are computationally less expensive than series reductions. The external nodes are then revisited to reduce any self-loops created by the deletion of internal flowgraph nodes.

The main variables of the algorithm are defined as follows:

- $\mathbf{N}_i$ = set of signal flowgraph nodes (variables) in partition i
- $\mathbf{B}_i$ = set of signal flowgraph branches in partition i
- $\mathbf{EN}_i$ = set of signal flowgraph external nodes (variables) in partition i
- $\mathbf{IN}_i$ = set of signal flowgraph internal nodes (variables) in partitioned circuit i
- $\mathbf{TN}$ = set of signal flowgraph tearing nodes (variables)
- $\mathbf{V}$ = set of signal flowgraph input and output nodes (variables)
- w_b = weight associated with branch b
- T_n = set of braches entering signal flowgraph node n
- F_n = set of branches exiting signal flowgraph node n

Each branch and node (figure 2) is described as follows, respectively:



Being an external node is determined by a direct check in the list of tearing nodes **TN**. **TN** is defined as

$$\mathbf{TN} = \left\{ \left[\bigcap_{i=1}^{p} \mathbf{EN}_i \right] \cup \mathbf{V} \right\} \tag{2}$$

If node n does exist in **TN**, then the node is not to be suppressed. If, however, the node is not in **TN**, the node is scheduled to be suppressed.

The program concludes execution by recombining the solutions for the partitions into an overall system solution. The algorithm is recursive, highly parallelizable, and suitable for implementation on multiprocessor machines. Each partition can be analyzed independent of the other partitions and is only visited once (non-iterative). This reduces to a minimum any interprocessor communication, and in turn the large overhead usually associated with the use of multiprocessor machines.

The general algorithm can be illustrated as follows:

```
main {
    MSFG_analyze (Network);
    suppress (Network);                    /* Delete any internal nodes left */
}   /* End of main procedure */

MSFG_analyze (Network) {                   /* Binary partition into P_left & P_right and perform
                                              the analysis */
    partition (Network);                   /* Partitions P_left & P_right */

    if (P_left is a leaf cell)
        suppress (P_left);                 /* Suppress internal nodes for this terminal block */
    else MSFG_analyze (P_left);            /* Recursively call MSFG_analyze */

    if (P_right is a leaf cell)
        suppress (P_right);                /* Suppress internal nodes for this terminal block */
    else MSFG_analyze (P_left);            /* Recursively call MSFG_analyze */

    combine (P_left,P_right);              /* Produce merged MSGF by concatenating the reduced
                                              MSFGs and generate the new Network */

    return (Network);
} /* End of MSGF analysis */
```

```
suppress (P,TN) {                          /* Suppress all internal nodes for a partition P */
    foreach (n ∈ N_i) {                    /* All nodes */
      /* A parallel suppression (figure 4) */
      foreach (b_1 ∈ T_n){                 /* For all incoming branches */
          foreach (b_2 ∈ T_n && node1(b_1) = node2(b_2)){ /* For all parallel branches */
              w_b1 = w_b1 + w_b2;          /* w_b1 = weight of 1st parallel branch */
              delete(b_2);
          }
      }
          /* reduce all self loops (figure 6) */
      foreach (b ∈ T_n && node1(b) = node1(b)){        /* A self-loop found */
          foreach (b_in ∈ T_n && b_in ≠ b){            /* For all incoming branches */
              w_bin = w_bin / (1 − w_b);
          }
      }
          /* A series and general suppression (figures 3 and 5) */
      foreach (b_in ∈ T_n) {
          foreach (b_out ∈ F_n) {
              create new branch b;
              w_bnew = w_bin * w_bout;
          }
      }
      delete all branches attached to node n;
      delete(node n);
    } /* End of variable suppression for this partition */
} /* End of suppression for this partition */
```

3.2. *The Input Deck*

MASSAP requires a SPICE-like input deck with additional partitioning statements. A flowgraph branch is input exactly as a circuit branch is using SPICE. The direction of flow is taken from the first node given to the second. An ''.output'' statement outlines the input and the output nodes. Currently a graphics interface is being built onto the package which includes menu driven options in addition to schematic capture capabilities.

3.3. *The Output File*

The program output is a sequence of hierarchically dependent expressions in the form of an evaluatable programming module. The program gives an option of choosing a C function or a FORTRAN subroutine as an output. As each internal node is deleted, a sequence of expressions term is created from the weights of the reduced branches and assigned a new symbol. The new symbol is then used in the analysis rather than the cumbersome expression it represents. The resulting output contains one branch for each input–output node combination (figure 1).

3.4. *Special Considerations*

1. It is better to reduce feedback paths before all the nodes in the forward path of the loop are deleted. This is not a requirement for the success of the algorithm, however, it will have a significant impact on the resultant sequence of expressions. Example 3 and table 2 illustrate the impact of the reduction node ordering on the sequence of expressions.

It is possible to guarantee that all nodes in a feedback paths are reduced first by one of two ways:

a. Implementing a check before eliminating each node. A search for any path from that node to itself is sufficient to identify a feedback loop. If such a path is found, the processing of that node is assigned a lowest priority position in the reduction queue.

b. Listing all feedback paths first in the input deck. This will guarantee that all feedback nodes are placed

in a higher position in the reduction queue. This is equivalent to labeling the feedback paths or the feedback nodes as such, which is another possible alternative.

2. Since Laplace transforms and signal flowgraphs are only capable of characterizing and analyzing linear networks, operational amplifiers circuits can be analyzed only if they are in a linear configuration. Therefore, since there are only a few general forms of such configurations (see figure 9 and equations (5)–(8)), it is best to include them in a macro library. Any occurrence of a linear operational amplifier network would be replaced by its approriate signal flowgraph model.

Ideal operational amplifier reduction rules are: Recursively reduce all op amp feedback loops. The recursiveness is necessary due to the fact that an op amp circuit can exist in the feedback path of another op amp configuration (figure 8). This would require reducing op amp circuit A and replacing it by its equivalent op amp model before continuing with the reduction of op amp circuit B. Also, all linear operation amplifier configuration signal flowgraphs are derived from a library rather than reanalyzing at every op amp instance in the circuit. A general form for a linear op amp circuit is shown in figure 7. The resulting signal flowgraph is also shown in figure 7, where

$$\mathbf{H}_{54} = \frac{V_4}{V_5} = -\frac{y_2}{y_4} \tag{3}$$

$$\mathbf{H}_{14} = \frac{V_4}{V_1} = \frac{y_1(y_2 + y_4)}{y_4(y_1 + y_3)} \tag{4}$$

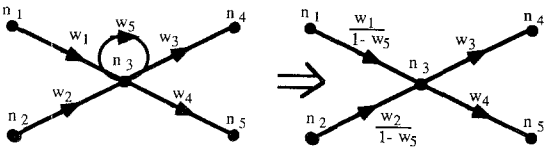
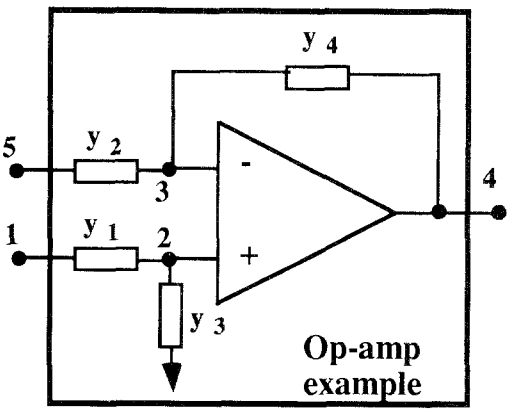


Fig. 6. General reduction of a self-loop.

For each additional input V_{i-} connected to the negative terminal of the op amp through an admittance y_{i-}, the expression associated with it is given by

$$\mathbf{H}(V_o, V_{i-}) = \frac{V_o}{V_{i-}} = -\frac{y_{i-}}{y_f} \tag{5}$$

where y_f is the feedback admittance from the op amp output node to the negative input terminal. Also for each additional input connected to the positive terminal of the op amp through an admittance y_{i+} the expression associated with it is given by



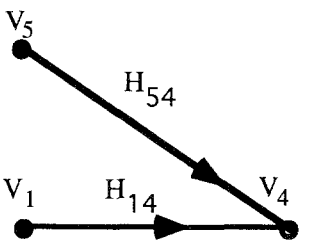


Fig. 7. Operational amplifier example.

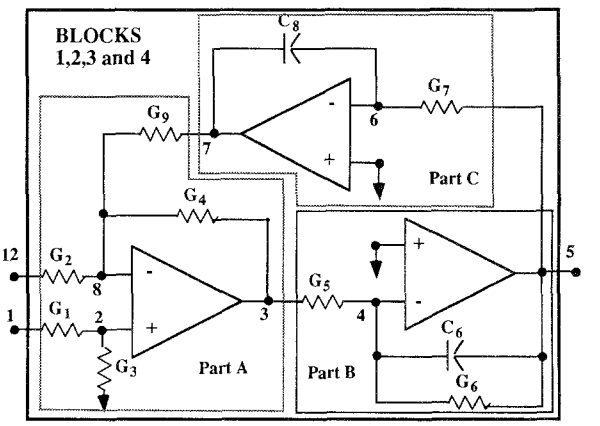


Fig. 8. Building block of the bandpass filter.

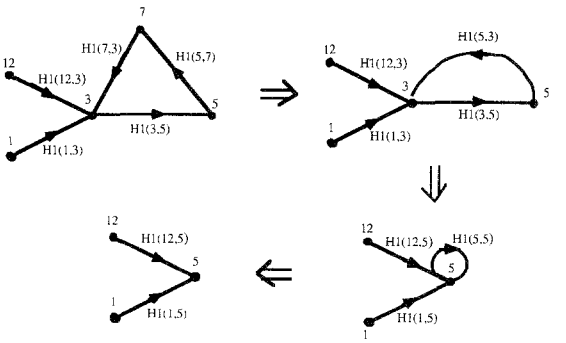


Fig. 9. Signal flowgraph reduction process on Example 4.1.

$$\mathbf{H}(V_o, V_{i+}) = \frac{V_o}{V_{i+}} = \frac{y_{i+}(\Sigma y_{i-} + y_f)}{y_f(\Sigma y_{i+} + y_g)} \quad (6)$$

where Σy_{i-} is the sum of all input admittances connected to the negative terminal of the op amp and y^g is the admittance between the positive terminal and the low reference voltage. Usually in active filter design y_g is chosen such that

$$y_g = \Sigma y_{i-} + y_f - \Sigma y_{i+} \quad (7)$$

which makes equation (6) look like

$$\mathbf{H}(V_o, V_{i+}) = \frac{V_o}{V_{i+}} = \frac{y_{i+}}{y_f} \quad (8)$$

and reduces the number of operations in the output expressions needed to model the general linear op amp circuit. However for the sake of generality, equations (5) and (6) are used in solving the examples in this paper to produce the figures in tables 2 and 3.

3.5. *Algorithm Complexity*

There are two measures to the performance of the recursive algorithm outlined above: (1) the program running time complexity measured by the number of steps the algorithm has to perform and (2) the quality of the results measured in terms of the number of operations produced in the sequence of expressions (number of additions, subtractions, multiplications, and divisions).

First the program complexity is addressed using a network with b branches, n nodes, and p partitions. An average number of nodes per subcircuit is given by

$$n_p = \frac{n}{p} \quad (9)$$

Since this algorithm seeks to deal with real problems, a perfectly valid assumption is that a real network has an upper bound on the number of branches incident and leaving each node (excluding the reference nodes). A general survey of practical (real) circuits showed that an average of six branches are connected to a given circuit node. Therefore, for this discussion, the average number of branches connected to a node is assumed to be a constant. Alternatively, this assumption states that each vertex in the flowgraph has an average of k branches connected to it, $k/2$ entering and $k/2$ leaving. Since each branch has two nodes, this gives rise to the equation

$$b = \frac{nk}{2} \quad (10)$$

The function *suppress*(P, TN) is the heart of the algorithm. The analysis is done by addressing the components of this function:

1. *Parallel reduction.* Since a nested search through all branches entering the node is performed, the order of complexity of this reduction is given by

$$\frac{k}{2}\frac{k}{2}n = \frac{k^2}{2}n \Rightarrow O(n) \quad (11)$$

2. *General series reduction.* Here a nested search through all the branches connected to this node is needed, hence the order of complexity of this reduction is derived from

$$\frac{k}{2}\frac{k}{2}n = \frac{k^2}{2}n \Rightarrow O(n) \quad (12)$$

3. *Self-loop reduction.* Self-loops results from feedback paths in the signal flowgraph. For a given network, a constant number feedback paths, f, is assumed to exist. The worst-case situation is when every node develops a self-loop through the reduction of the feedback path. This gives rise to a complexity of

$$\frac{k}{2}\frac{k}{2}f \Rightarrow O(f) \Rightarrow O(n) \quad \text{(worst case)} \quad (13)$$

The order of complexity of *suppress*( ) is given by adding equations (11), (12), and (13) which yields the following figure for real networks

$$3\frac{k^2}{2}n \Rightarrow O(n) \quad (14)$$

The complexity of the whole algorithm can be expressed in terms of the following reoccurrence equation:

$$F(n) = F\left(\frac{n}{2}\right) + 3\frac{k^2}{2}\frac{n}{p} \quad (15)$$

MASSAP uses a binary partitioning algoritm to decompose the input network or flowgraph [4]. The complexity of the partitioning algorithm however, it not included in the current complexity analysis. The reason is that most circuits are already prepartitioned due to the fact that they are composed of building blocks during their design phase. These building blocks are considered natural partitions and eliminate the need for applying the partitioning function.

The function *suppress*( ) is initiated when the size of each partition (subcircuit) reaches n/p nodes. Since the internal nodes n_i are suppressed the number of nodes processed at the next function cell level is $2(n/p - n_i)$ which is the number of tearing variables (external) to the subcircuits. It is expected that $n_i > n/p$. This actually is a necessity for the efficient execution of the algorithm. Also for practical circuit building blocks, the number of I/O variables will always be less than the number of internal nodes. Therefore, on the average, the number of nodes that *suppress*( ) will handle will always be a constant proportional to $2(n/p - n_i)$. Another feature of real circuits is the fact that n and p are directly proportional to each other, that is a bigger circuit will always have more partitions to it (building blocks). Also, automatic partitioners usually continue until a maximum p or a minimum n/p is met. Hence the final average asymptotic time complexity of the algorithm is given by solving the reoccurence equation (15). The result is

$$O(n \log n) \tag{16}$$

To study the quality of the results that the algorithm produces, a close inspection of the above analysis will yield a direct correspondence between the number of operations that need to be performed in every reduction and the number of operations resulting in the sequence of expressions. In general terms each parallel, series or self-loop reduction at a node will produce

0 multiplications and $k/2$ additions
for parallel reduction (17)

$k^2/2$ multiplications and 0 additions
for general reduction (18)

$k/2$ multiplications and 1 addition
for self-loop reduction (19)

Equations (17), (18), and (19) show that the number of operations that will result in the sequence of expressions will exhibit only a quadratic growth with respect to the size of the circuit measured by the number of nodes in it.

4. Examples

Example 4.1. (Bandpass filter terminal block). Example 1 (figure 8) is the analysis of a subcircuit which is a building block of example 3. The ideal operational amplifiers have been reduced to flowgraph form as shown in figure 9. Since the block has 2 input nodes (1 and 12) and one output node (5), the resultant signal flowgraph consists of 2 branches representing the transfer functions $H(12, 5)$ and $H(1, 5)$.

Example 4.2. (Dual-biquad). Consider the dual-biquad amplifier example shown in figure 10 [10]. The output sequence of expressions is

```
w1 = −g4/(g1 − sC1)
w2 = −g2/sC2
w3 = −g5/g6
w4 = −g4/g3
w5 = −g8/g11
w6 = −g7/g10
w7 = −g9/g10
H0E1(1,6) = w1*w2
H0E2(1,8) = w1*w5
H0E3(1,11) = w1*w6
H0E4(10,6) = w4*w2
H0E5(10,8) = w4*w5
H0E6(10,11) = w4*w6
H0E7(1,10) = H0E1(1,6)*w3
H0E8(10,10) = H0E4(10,6)*w3
H0E9(1,10) = H0E7(1,10)/(1−H0E8(10,10))
H0E10(1,8) = H0E9(1,10)*H0E5(10,8)
H0E11(1,11) = H0E9(1,10)*H0E6(10,11)
H0E12(1,8) = H0E2(1,8)+H0E10(1,8)
H0E13(1,11) = H0E3(1,11)+H0E11(1,11)+w7
```

Example 4.3. (Bandpass filter). Figure 14 is a full implementation of a bandpass filter [3]. Figure 15 shows the hierarchical partitioning and analysis tree model and figures 7, 12, and 13 show the subcircuits of the partitioned network. By using partitioning, the computational efficiency of the flowgraph is improved. Table 1 shows the results of the analysis and table 2 shows a comparison for this example with SCAPP [2] and with the Coates Graph hierarchical analysis method presented in [3]. The results are shown with and without network partitioning. The table also illustrates the effect of the node reduction order on the complexity of the output expressions.

Example 4.4. (Ladder network). Examples 4a and 4b show comparisons for ladder networks of 10 and 20 nodes, respectively. The results of the signal flowgraph analysis and comparisons with other methods are presented in table 3. The computational count increases linearly with the number of circuit elements. The flowgraph method multiplication and addition counts are comparable to those from the network approach [2]. Although there is not a one-to-one mapping from circuit

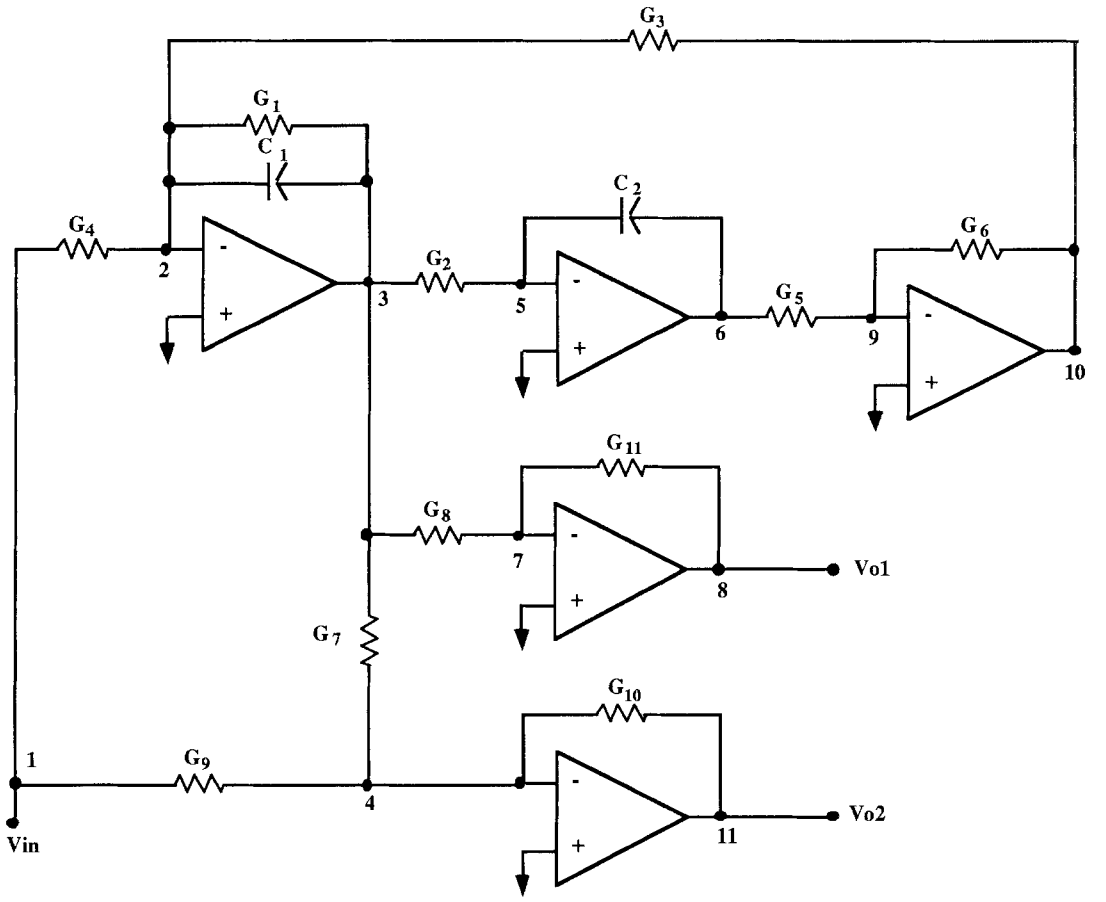


Fig. 10. Dual-biquad amplifier for Example 4.2 [10].

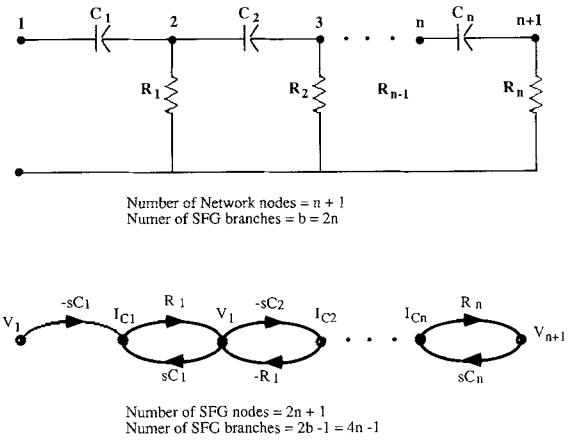


Fig. 11. Ladder network signal flowgraph model.

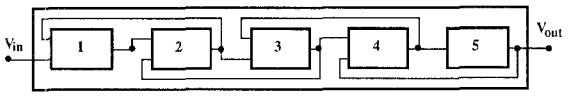


Fig. 12. Interconnections of the partitioned bandpass filter.

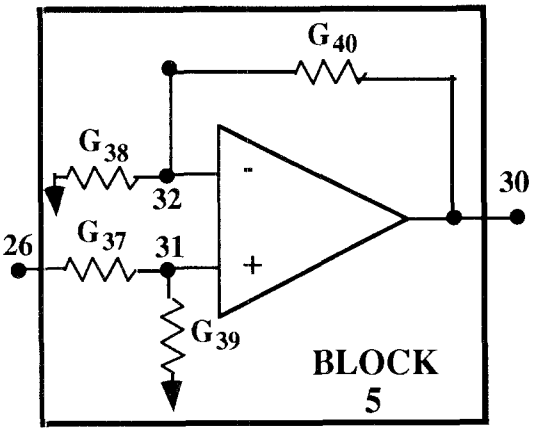


Fig. 13. Block 5 of the bandpass filter (Example 4.3.).

nodes to flowgraph vertices, for a ladder network with n nodes and b branches the number of flowgraph vertices is $2n + 1$ and the number of flowgraph branches is $4n - 1$, the node reduction rules are simpler for the flowgraph approach.

5. General Discussion

The reduction processes of a Mason's signal flowgraph using the above suppression procedures has been considered a complex process for a digital computer [1, 11]. The main problem is the growth in the number of terms involved when the reduction is applied in order to get a compact signal flowgraph. This problem is solved here by using several aids:

1. There is no advantage to attempting to produce a single expression for the final symbolic transfer function for large network. No possible insight could be gained from inspecting a transfer function with hundreds of terms in the numerator and denominator. The only possible insight is through numerical evaluation of the function and a plot of the transfer characteristics. Using a hierarchically dependent sequence of expression [2, 3] rather than a single expression to represent the transfer function eliminates the handicap.
2. Computer memory requirement for analysis of large-scale networks can be reduced drastically by using a sequence of expressions method.

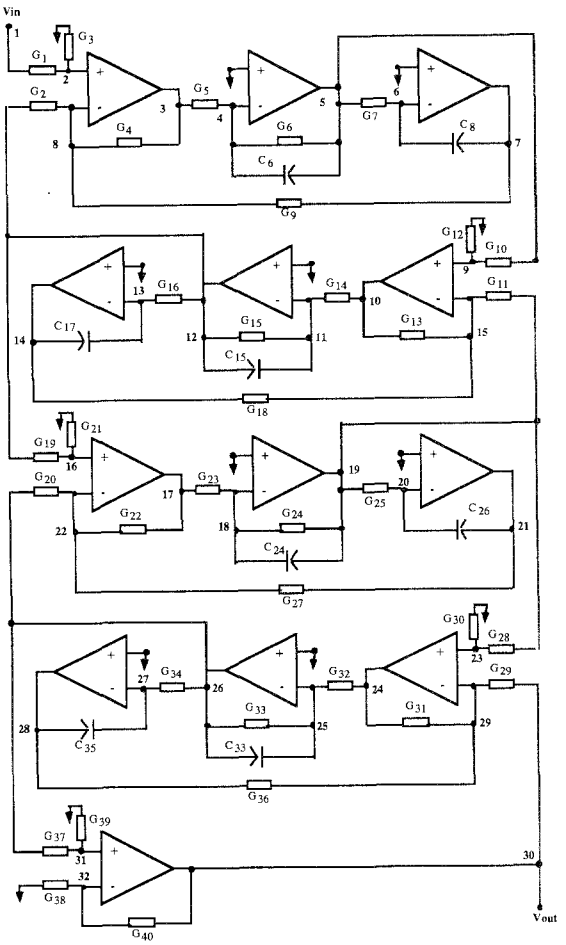


Fig. 14. Bandpass filter of example 3 [3].

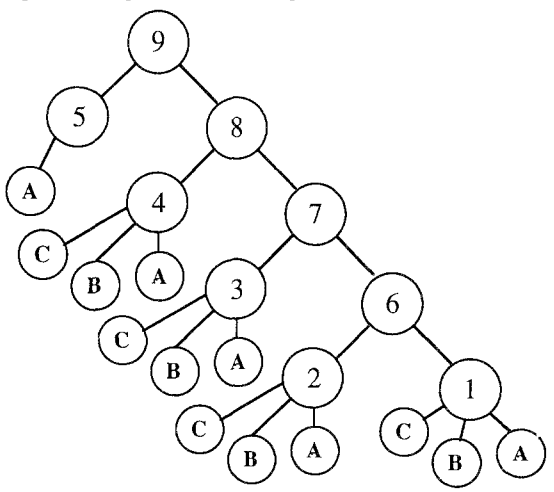


Fig. 15. Hierarchical model for bandpass filter (Example 4.3).

3. The use of network partitioning or graph partitioning to reduce the overall complexity of the algorithm needed to analyze large-scale networks.

Table 1. Symbolic analysis results of Example 4.3.

Block Number	New Symbol	Value of New Symbol
Terminal block 1	$H1(1,3)$	$g_1(g_2+g_4)/g_4(g_1+g_3)$
	$H1(12,2)$	$-g_2/g_4$
	$H1(7,3)$	$-g_9/g_4$
	$H1(3,5)$	$-g_5/(g_6+sc_6)$
	$H1(5,7)$	$-g_7/sc_8$
Reduce node 7	$H1(5,3)$	$H1(5,7)\ H1(7,3)$
Reduce node 3	$H1(12,5)$	$H1(12,3)H1(3,5)$
	$H1(1,5)$	$H1(1,3)H1(3,5)$
	$H1(5,5)$	$H1(5,3)H1(3,5)$
Reduce self-loop at node 5	$H1(12,5)$	$H1(12,5)/(1-H1(5,5))$
	$H1(1,5)$	$H1(1,5)/(1-H1(5,5))$
Terminal block 2	$H2(5,10)$	$g_{10}(g_{11}+g_{13})/g_{13}(g_{10}+g_{12})$
	$H2(19,10)$	$-g_{11}/g_{13}$
	$H2(14,10)$	$-g_{18}/g_{13}$
	$H2(10,12)$	$-g_{14}/(g_{15}+sc_{15})$
	$H2(12,14)$	$-g_{16}/sc_{17}$
Reduce node 14	$H1(12,10)$	$H1(12,14)H1(14,10)$
Reduce node 10	$H1(19,12)$	$H1(19,10)H1(10,12)$
	$H1(5,12)$	$H1(5,10)H1(10,12)$
	$H1(12,12)$	$H1(12,10)H1(10,12)$
Reduce self-loop at node 12	$H1(19,12)$	$H1(19,12)/(1-H1(12,12))$
	$H1(5,12)$	$H1(5,12)/(1-H1(12,12))$
Terminal block 3	$H3(12,17)$	$g_{19}(g_{20}+g_{22})/g_{22}(g_{19}+g_{21})$
	$H3(26,17)$	$-g_{20}/g_{22}$
	$H3(21,17)$	$-g_{27}/g_{22}$
	$H3(17,19)$	$-g_{23}/(g_{24}+sc_{24})$
	$H3(19,21)$	$-g_{25}/sc_{26}$
Reduce node 21	$H1(19,17)$	$H1(19,21)H1(21,17)$
Reduce node 17	$H1(26,19)$	$H1(26,17)H1(17,19)$
	$H1(12,19)$	$H1(12,17)H1(17,19)$
	$H1(19,19)$	$H1(19,17)H1(17,19)$
Reduce self-loop at node 19	$H1(26,19)$	$H1(26,19)/(1-H1(19,19))$
	$H1(12,19)$	$H1(12,19)/(1-H1(19,19))$
Terminal block 4	$H4(19,24)$	$g_{28}(g_{29}+g_{31})/g_{31}(g_{28}+g_{30})$
	$H4(30,24)$	$-g_{29}/g_{31}$
	$H4(28,24)$	$-g_{36}/g_{31}$
	$H4(24,26)$	$-g_{32}/(g_{33}+sc_{33})$
	$H4(26,28)$	$-g_{34}/sc_{35}$
Reduce node 28	$H1(26,24)$	$H1(26,28)H1(28,24)$
Reduce node 24	$H1(30,26)$	$H1(30,24)H1(24,26)$
	$H1(19,26)$	$H1(19,24)H1(24,26)$
	$H1(26,26)$	$H1(26,24)H1(24,26)$
Reduce self-loop at node 26	$H1(30,26)$	$H1(30,26)/(1-H1(26,26))$
	$H1(19,26)$	$H1(19,26)/(1-H1(26,26))$
Terminal block 5	$H5(26,30)$	$g_{37}(g_{38}+g_{40})/g_{40}(g_{37}+g_{39})$
Middle block 6	$H6(1,12)$	$H1(1,5)H2(5,12)$
	$H6(12,12)$	$H1(12,5)H2(5,12)$
	$H6(1,12)$	$H6(1,12)/(1-H6(12,12)$
	$H6(19,12)$	$H2(19,12)/(1-H6(12,12))$
Middle block 7	$H7(1,19)$	$H6(1,12)H3(12,19)$
	$H7(19,19)$	$H6(19,12)H3(12,19)$
	$H7(1,19)$	$H7(1,19)/(1-H7(19,19))$
	$H7(26,19)$	$H3(26,19)/(1-H7(19,19))$
Middle block 8	$H8(1,26)$	$H7(1,19)H4(19,26)$
	$H8(26,26)$	$H7(30,26)H4(19,26)$
	$H8(1,26)$	$H8(1,26)/(1-H8(26,26)$
	$H8(30,26)$	$H4(30,26)/(1-H8(26,26))$
Middle block 9 (final)	$H9(1,30)$	$H8(1,26)H5(26,30)$
	$H9(30,30)$	$H8(30,26)H5(26,30)$
	$H9(1,30)$	$H9(1,30)/(1-H9(30,30)$

Table 2. Comparison of bandpass filter with other methods.[a]

Partitioning Status	Analysis Method	Example 4.3. Bandpass [3] Mults	Adds	Eqs
With partitioning	Coates (flowgraph) [3]	63	30	25
	SCAPP (network) [2]	48	27	56
	MASSAP (flowgraph)	70	29	60
No partitioning	Coates (flowgraph) [3]	*b*	*b*	*b*
	SCAPP (network) [2]	79	35	88
	MASSAP (flowgraph) (feedback nodes reduced first)	89	38	85
	MASSAP (flowgraph) (feedback nodes reduced last)	94	40	92

[a]Number of multiplications include divisions, and the number of additions include subtractions.
[b]Data is not available for the unpartitioned case.

4. The use of symbolic solutions facilitates the use of noniterative hierarchical procedures for combining partitioned solutions into an overall system solution. This application is ideally suited for implementation on parallel processor machines.
5. Building standard symbolic signal flowgraph solution librarires for commonly used cells. This would eliminate the need for the repeated analysis of such cells. Even within one network, once a subnetwork that is instantiated more than once is analyzed, no further analysis of the other instances is required.

The reason the algorithm is suitable for active operational networks is the one-to-one correspondence between the circuit and its signal flowgraph representations. For a passive network, the number of nodes and branches in the signal flowgraph is more than in the original circuit (figure 11). This results in more suppression steps and thus more computational time. However the method works for both passive and active networks and the comparisons in the next section shows its suitability to the task.

6. Comparisons with Other Symbolic Methods

Symbolic analysis has been performed on the bandpass filter (figure 14 and example 3) using the two methods previously proposed for symbolic analysis of large-scale circuits [2, 3]. This particular filter has been chosen because both hierarchical methods showed detailed results of their analysis of the filter. The results of using the hierarchical Mason flowgraph approach proposed in this paper are shown in table 1. The comparison with the other two methods is shown in table 3. The hierarchical model for the process is shown in

Table 3. Examples 4.1, 4.2, and 4.4 comparison with SCAPP[a].

Analysis Method	Example 4.1 Basic Cell			Example 4.2 Biquad [10]			Example 4.4 Ladder Network (10 nodes)			Example 4.4 Ladder Network (20 nodes		
	Mults	Adds	Eqs	Mults	Adds	Eqs	Mults	Adds	Eqs	Mults	Adds	Eqs
SCAPP [2]	16	7	23	16	6	22	48	34	54	102	70	107
MASSAP	13	5	11	16	5	18	67	33	67	100	49	100

[a]Number of multiplications include divisions, and number of additions include subtractions.

figure 15. Parts A, B and C, are simply the blocks already analyzed in figure 7 (part A, which is a summer is slightly different with an extra input terminal).

Table 3 also illustrates comparisons with the hierarchical network approach used in SCAPP for all the examples presented in this paper.

7. Conclusions

The advantages of the Mason's flowgraph method implemented in MASSAP and presented herein are:

1. The simplicity of its reduction rules in order to obtain the I/O transfer function sequence of expressions.
2. The number of operations and expressions resulting are comparable to the other methods proposed for the analysis of large-scale networks. This is due to (a) the use of a sequence of expressions which exhibits only a quadratic growth rate for general circuits and sometimes a linear growth rate (ladder networks) in the number of terms as the network size increases, and (b) the use of network partitioning which reduces the overall complexity of the analysis algorithm.
3. Symbolic libraries can be built for common subsystems with a reduced flowgraph already produced.
4. The ability to include any current or voltage variable as part of the analysis without any extra or special considerations [Lin76].
5. Another major advantage of this hierarachical flowgraph approach is the ability to simulate integrated systems. That is, systems with electrical and non-electrical components. The only requirements on the components of the system is that they be representable in flowgraph format. Mechanical and chemical systems are commonly represented by a Mason's flowgraph and interconnections of such components with electrical networks can be simulated using the method herein.
6. The existence of division in the sequence of expression. This eliminates underflow and overflow problems and the need for normalization during the numerical evaluation process of the sequence of expressions. This is also the case in the network approach [2] but is not the case in the Coates flowgraph approach [3].
7. The number of terms in the flowgraph analysis of active networks is lower than that for general networks due to the one-to-one correspondence between the actual circuit nodes and the nodes of its flowgraph model.

In general this method works very well for circuits with standard building blocks, active or passive. The fact that these building blocks can be considered as natural partitions to the circuit and their analysis results loaded into the analysis directly eliminates the overhead associated with building the signal flowgraph for the network.

References

1. P.M. Lin, "A survey of applications of symbolic network functions," *IEEE Trans. Circuit Theory*, Vol. CT-20, pp. 732–737, 1973.
2. M.M. Hassoun and P.M. Lin, "An efficient network approach to symbolic simulation of large-scale circuits," in *Proc. IEEE Int. Symp. Circuits Sys.*, pp. 806–809, 1989.
3. J.A. Starzyk and A. Konczykowska, "Flowgraph analysis of large electronic networks," *IEEE Trans. Circuit Sys.*, Vol. CAS-33, pp. 302–315, 1986.
4. M.M. Hassoun, and P.M. Lin, "An efficient partitioning algorithm for large-scale circuits," in *Proc. IEEE Int. Symp. Circuits Sys.*, pp. 2405–2408, 1990.
5. G. Gielen, H. Walscharts, and W. Sansen, "ISSAC: A symbolic simulator for analog integrated circuits," *IEEE J. Solid-State Circuits*, Vol. SC-24, pp. 1587–1597, 1989.
6. F.V. Fernandez, A. Rodriguez-Vazquez, and J.L. Huertas, "An advanced symbolic analyzer for the automatic generation of analog circuit design equations," in *Proc. IEEE Int. Symp. Circuits Sys.*, Singapore, pp. 810–813, 1991.
7. G. Di Domenico and S.J. Seda et. al., "BRAINS: a symbolic solver for electronic circuits," *Int. Workshop Symb. Methods Appl. Circuit Design*, 1991.
8. R. Sommer, "EASY—an experimental analog design system framework," *Int. Workshop Sym. Methods Appl. Circuit Design*, 1991.

9. M. M. Hassoun, "Symbolic analysis of large-scale networks," Ph.D. dissertation, Purdue University Engineering Library, West Lafayette, IN, 1988.
10. W.-K. Chen, *Passive and Active Filters Theory and Implementations*, Wiley: New York, 1986.
11. L.O. Chua and P.M. Lin, *Computer Aided Analysis of Electronic Circuits—Algorithms and Computational Techniques*, Prentice-Hall: Englewood Cliffs, NJ, 1975.
P.R. Adby, "Tree enumeration from the incidence matrix," *Int. J. Elec. Eng. Educ.*, Vol. 24, pp. 183–187, 1987.
G. Gielen and W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits*, Kluwer: Boston, 1991.
M. M. Hassoun, "Hierarchical symbolic analysis of large-scale systems using a Mason's signal flowgraph model," in *Proc. IEEE Int. Symp. on Circuits and Sys.*, pp. 802–805, 1991.
A. Konczykowska and M. Bon, "Symbolic simulation for efficient repetitive analysis and artificial intelligence techniques in CAD," in *Proc IEEE Int. Symp. Circuits Sys.*, pp. 802–805, 1989.
P.-M. Lin, *Symbolic Network Analysis*, Elsevier: Amsterdam, Netherlands, 1991.
S.J. Mason, "Feedback theory—some properties of signal flowgraphs," *Proc. IRE*, Vol. 41, pp. 1144–1156, 1953.
S.J. Mason, "Feedback theory—further properties of signal flowgraphs," in *Proc. IRE*, Vol. 44, pp. 920–926, 1956.
R.R. Mielke, "A new signal flowgraph formulation of symbolic network functions," *IEEE Trans. Circuits Sys.*, Vol. CAS-25, pp. 334–340, 1978.
M. Sagawa and H. Kitazawa, "Symbolic network analysis of linear networks using parameter extraction proc.," *Trans. IECE Japan*, Vol. E 60, No. 8, Abs, p. 414, 1980.
K. Singhal and J. Vlach, "Generation of immetance functions in symbolic form for lumped distributed active networks," *IEEE Trans. Circuits Sys.*, Vol. 21, pp. 57–67, 1974.
K. Singhal and J. Vlach, "Symbolic analysis of analog and digital circuits," *IEEE Trans. Circuits Sys.*, Vol. 24, pp. 598–609, 1977.
R. Sommer, "EASY—an experimental analog design system framework," *Int. Workshop Sym. Methods Appl. Circuit Dseign*, 1991.
J.A. Starzyk and A. Konczykowska, "Flowgraph analysis of large electronic networks," *IEEE Trans. Circuits Sys.*, Vol. CAS-33, pp. 302–315, 1986.

Kevin McCarville was born in Stillwater, Minnesota on April 4, 1968. He completed a B.S. degree at South Dakota State University in 1991 and is completing a M.S. Degree at Iowa State University both in Electrical Engineering.

He has been a research and teaching assistant since 1991 at Iowa State University. His research interests include symbolic circuit simulation particularly in the area of active device modeling, VLSI design, and electronic circuits.

Marwan M. Nadim Hassoun received his B.S. degree in Electrical Engineering with high honors from South Dakota University in 1983 and the M.S. and Ph.D. degrees in Electrical Engineering from Purdue University in 1984 and 1988. In 1985 he was a sofware development engineer with Hewlett-Packard Company in Santa Clara, California. In 1988 he joined the department of Electrical Engineering and Computer Engineering at Iowa State Unviersity. His current research interests are in symbolic analysis of VLSI circuits and compuer-aided design algorithms for VLSI.