A Decomposition-Based Symbolic Analysis Method for Analog Synthesis from Behavioral Specifications

Alex Doboli and Ranga Vemuri

Digital Design Environments Laboratory, Department of ECECS
University of Cincinnati, Cincinnati, OH 45221-0030

Email: { adoboli,ranga } ececs.uc.edu

Abstract This paper discusses a technique for symbolic analysis of large analog systems. The method exploits the hierarchical structure and uniformity of a system for producing **compact** symbolic expressions. The technique is based on decomposing a system through the method of **tearing** [5] as opposed to traditional symbolic methods that use nodal analysis. After producing the symbolic model of a system, a set of reduction rules is applied to the model. Reductions attempt to decrease the number of arithmetic operations performed for numerically evaluating the symbolic model. The discussed technique is useful for synthesis, inside an exploration-loop, as it avoids repeatedly computing the symbolic models.

1. INTRODUCTION

Symbolic analysis is the task of automatically deducing relationships between overall parameters of a system and parameters of the composing elements of the system [6] [3]. For example, a symbolic expression can describe the transfer-function of a filter in terms of parameters of its building elements, i.e. op amps, resistors, and capacitors. Analysis techniques include determinant-based methods and signal-flow graph methods [6]. Determinant-based methods use Cramer's rule for solving the set of linear equations implied by symbolic analysis [3]. Signal-flow graph methods represent a set of linear equations as a weighted graph, and use Mason's rule for solving the equations [6]. Symbolic analysis has a wide range of applications i.e. frequency response calculation, sensitivity analysis, hardware synthesis etc. This paper proposes on a new symbolic analysis technique for analog synthesis.

The main challenge for symbolic analysis is the exponential growth of the produced symbolic expressions (10^{11} terms for an op amp [4]). Current research considers two ways of handling this aspect: approximation and hierar-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35498-9_57](https://doi.org/10.1007/978-0-387-35498-9_57)

L. M. Silveira et al. (eds.), *VLSI: Systems on a Chip*

© IFIP International Federation for Information Processing 2000

Figure 1 Behavioral synthesis environment

chical methods. Approximation methods [12] retain only the significant terms of the symbolic expressions and eliminate the insignificant ones. The difficulty, however, lies in identifying what terms to eliminate, and what the resulting approximation error could be. Hierarchical methods [11] [4] decompose the global analysis problem into smaller sub-problems. The global expressions are described as sequences of sub-expressions for the sub-problems. An exponential number of operations is still needed for their numerical evaluation.

Our approach addresses the exponential complexity of symbolic analysis by considering specific aspects of the analyzed system. A system has a *structural hierarchy* and *uniformity* that can be exploited for more effective symbolic analysis. For example, a filter is typically composed of multiple stages of similar blocks connected in identical patterns. Such knowledge of hierarchy and uniformity can be used for (1) simplifying symbolic calculations, (2) reducing their memory size and (3) diminishing their derivation time.

This paper discusses a method for symbolic analysis of large analog systems. For an overall parameter of a system, the algorithm produces a *computational tree* that describes how its values depend on the parameters of the blocks composing the system. A computational tree (referred in this paper as *Analog Performance Tree*) is an uninterpreted variant of the closed-form, symbolic expressions that are produced by traditional methods [6] [3]. In our approach, computational trees are produced in a top-down manner by traversing the system hierarchy and without explicitly solving equations. The algorithm exploits the hierarchical structure and uniformity of a system for producing compact symbolic expressions and for diminishing the generation time of the symbolic trees. Once a tree is produced, a set of reduction rules is applied for simplifying the computational tree and decreasing the number of arithmetic operations performed for numerically evaluating the tree.

We use this method for behavioral synthesis of analog systems. The synthesis flow is depicted in Figure 1. The application to be synthesized is specified behaviorally by expressing its signal flows and processing. During synthesis, different structural net-lists of interconnected circuits are produced for the specification. For each net-list, a *cost function* (symbolic model) is derived by the described symbolic technique. The cost function relates system parameters, i.e. overall transfer function etc. to the composing block parameters, i.e. gains, bandwidth, impedances etc. It is used for performing optimization based trade-off exploration for the circuit parameters. However, all net-lists

Figure 2 Hierarchical description and Analog Performance Tree for a filter

share a common factor in that they realize the same signal-processing flow. This can be exploited in reducing both the memory size and set-up time for the symbolic expressions, and is the motivation behind this paper. The two gains are important as symbolic analysis is repeatedly performed during synthesis.

The rest of the paper is organized as follows. Section 2 describes the main features and advantages of our symbolic method by using a motivational example. In Section 3, we present the model of analog systems that is used for symbolic analysis. Section 4 focuses on our symbolic method. Experimental results are discussed in Section 5, followed by conclusions in Section 6.

2. MOTIVATING EXAMPLE

To give a brief insight on the addressed problem and the solution we propose, we present a fourth-order filter [11] as a motivating example. The middle part of Figure 2a sketches the filter specification represented at the *Signal-Flow Graph Level*. The fourth-order function of the filter is obtained by cascading two second-order stages. Each of the second-order blocks implements the same close-loop flow, as depicted in the middle part of Figur.2a. This way of describing the signal processing and flows of a system is called *Signal-Flow Graph (SFG)* [7]. We assume that the behavior of an analog system is described as an SFG and is the input for behavioral synthesis.

Behavioral synthesis now derives the best implementation for an SFG specification, so that, the described signal-path is accomplished, certain design/performances are met, and a cost function is optimized. During synthesis, different implementations are explored, and hence, the symbolic analysis part is repeatedly done. An observation is that all implementations will actually realize the same SFG described in the specification. We exploit this similarity to reduce the required time for finding the symbolic expressions of an implementation. Secondly, there are some common factors between parts of an implementation (i.e. *Stage 1* and *Stage 2* in Figur.2a). We intend to

take advantage of these similarities by analyzing only one of the stages, and re-using its symbolic model for the second stage. This shortens the set-up time of global symbolic expressions and also compacts the required memory.

For exemplifying the analysis technique, let's assume that we intend to calculate for different filter implementations their transfer-functions defined as $TF = \frac{v_o}{v_i}$, when $i_o = 0$. Values v_o and v_i are voltages at the input and output ports, and values i_i and i_o are the currents. This analysis is useful to check whether a real implementation is within an acceptable error range from the input specification.

For analysis, the overall system can be modeled as a two-port described by the symbolic matrix $[R_{ij}]_{2 \times 2}$ (Figure 1.2a), where:

$$\begin{aligned} v_o &= R_{11}v_i + R_{12}i_o \\ i_i &= R_{21}v_i + R_{22}i_o \end{aligned}$$

The filter transfer function is given by $TF = R_{11}$, and thus the symbolic formula of R_{11} is calculated. Note that there is no need for also computing R_{12} , R_{21} or R_{22} .

The goal is to exploit the invariance of the SFG "algorithm" for different implementations. We perform a *top-down* symbolic analysis by considering *generic* parameters for the blocks (matrix E for *Stage 1* and matrix F for *Stage 2*, where the nature of the matrix elements is not fixed). This differs from traditional methods that are bottom-up approaches, and use physical parameters like impedances, admittances, etc. [3] [4] [6]. However, if physical parameters are used in the analysis, then the obtained formulas are specific to an implementation.

The elements of the overall matrix $[R_{ij}]_{2 \times 2}$ relate to elements of matrices $[E_{ij}]_{2 \times 2}$ and $[F_{ij}]_{2 \times 2}$ according to:

$$\begin{aligned} R_{11} &= \frac{E_{11} + F_{11}(E_{12}E_{21} - E_{11}E_{22})}{1 - E_{22}F_{11}}, & R_{12} &= \frac{E_{12}F_{12}}{1 - E_{22}F_{11}} \\ R_{21} &= \frac{E_{21}F_{21}}{1 - E_{22}F_{11}}, & R_{22} &= \frac{F_{22} + E_{22}(F_{12}F_{21} - F_{11}F_{22})}{1 - E_{22}F_{11}} \end{aligned}$$

These relations were obtained by setting up the equations for *Stage 1* and *Stage 2* and eliminating the unknowns v_x and i_x .

The four expressions depend on the parameters of the composing blocks and their connection style that is *series connection* for this example. However, they are not influenced by the nature of the block parameters. As matrices E_{ij} , and F_{ij} are generic, the four expressions can also be seen as functions that link for *any* series connection the overall symbolic parameters R_{ij} of the connection, to the parameters E_{ij} and F_{ij} of the composing blocks. Similar solving functions can be found for other popular connection styles, i.e. feed-back, feed-forward, divergent, etc. and which are discussed in more detail in Section 4. Solving functions are re-used in our technique for describing all connections of the same type. This reduces both the size of the overall expressions, and the set-up time as similar connections are handled only once.

Assume that solving functions *series*, *feed-back*, *feed-forward*, etc. relate the overall parameters of their connections to the parameters of the composing blocks. Then for a set of connected blocks, a *computational tree* named *Analog Performance Tree* (APT) can be built, describing how parameters relate to each other. Figure 2b depicts a set of connected blocks and Figure 2c shows the

APT for the transfer-function of the connection. *Internal nodes* describe the solving functions of the connection styles for the blocks. Various connections are also described in the first part of Section 4 and in Table 1. The parameters for each solving function relate to the composing blocks. The returned values are the parameters of the overall block. *Leaf nodes* describe the basic building blocks of an implementation, i.e. gain stages, integrators, RC stages, etc. Their parameters have a physical meaning, i.e. impedances, admittances, etc.

An APT is a symbolic expression in an unevaluated form. This is not a disadvantage as we are not interested in getting insight of a system, and where closed-form expressions would help. Besides, the tree structure of an APT allows easy simplifications or approximations.

In our technique, the computational tree is set-up by composing functions for the basic connection patterns, and not by solving equation sets like in traditional methods. This reduces the overall set-up time for the symbolic expressions. If basic connection styles cannot be recognized at one step then a decomposition of the system is performed. Decomposition is achieved through the principle of *tearing* [5]. Tearing separates a set of equations into sub-sets with less number of unknowns. Also, whenever similar block connections are identified (i.e. *Stage 1* and *Stage 2* in Figure 2a) only one of them is analyzed.

During synthesis, different implementations are explored for the blocks in an SFG. However, the entire APT tree remains unchanged, except the leaf nodes that are updated according to the chosen implementations for the blocks. This avoids repeated symbolic analysis inside the synthesis cycle.

3. SYSTEM MODEL FOR SYMBOLIC ANALYSIS

As seen from Section 1, the synthesis task bridges consecutive levels of abstraction: it starts with an abstract specification of a system, and goes down to hardware implementations by selecting circuit net-lists, building their symbolic model and performing the necessary parameter optimizations. For describing these synthesis tasks an appropriate representation model for an analog system is needed. The model must easily relate to the input specification. Also, it should allow the description of all relevant aspects for net-list generation, symbolic analysis, and parameter optimization.

Input specifications are described as SFGs (see Figure 2a). The net-list generation algorithm [1] also considers net-lists at the block-level. Thus, it is important that the model for symbolic analysis easily describes a hierarchy of connected blocks and how the overall performance parameters of the connected blocks relate to those of the composing blocks.

Our mathematical model for symbolic analysis captures (1) the signal processing and structural aspects of a system, and (2) the description of the computational path for calculating linear (linearized) performance values:

$$\boxed{\text{System} = \langle \text{SFG}, \text{APT} \rangle} \quad (1.1)$$

SFG is the *Signal-Flow Graph* used for describing the processing/structural attributes of a system, and *APT* is the *Analog Performance Tree*, that symbolically captures the computing path for performance attributes used in synthesis. This paper presents a method for setting-up the symbolic *APT* of a system when its *SFG* is known.

Figure 3 Representation of the model for symbolic analysis

SFG is the quint-tuple:

$$SFG = \langle Wires, Blocks, Edges, Inputs, Outputs \rangle \quad (1.2)$$

Each *wire* $\in Wires$ represents an ordered pair of symbolic values (v, i) , meaning that the two values are interdependent and they always appear in conjunction. *Wires* models continuous-time signals, v being a voltage and i a current.

Blocks is the set of all operational blocks that realize the signal processing of a system. Each block has a set of ports for connecting it to other blocks by using wires. Also, each block has a behavior defined in terms of the values v and i of the wires connected at its ports.

The behavior of a block with m ports can be modeled as a set of m linear equations with m knowns and m unknowns. Equations for composed blocks are similar to the input-output equations of a linear system [7]. For basic blocks (blocks that correspond to electric circuits), equations are derived from the nodal equations of the block. Knowns and unknowns are the i and v values of the m wires at the block ports. Depending on how the sets of knowns and unknowns are selected, the symbolic coefficients in the equations have different meanings. For example, if the v values of wires are considered to be knowns, and the i values unknowns then a block with m ports is described by the equation set:

$$\begin{aligned} i_1 &= s_{11}v_1 + s_{12}v_2 + \cdots + s_{1n}v_m \\ i_2 &= s_{21}v_1 + s_{22}v_2 + \cdots + s_{2n}v_m \\ &\vdots \\ i_m &= s_{m1}v_1 + s_{m2}v_2 + \cdots + s_{mn}v_m \end{aligned}$$

Coefficients s_{ij} are symbolic expressions and they describe admittances. The matrix $[s_{ij}]_{m \times m}$ is called *block matrix*.

A basic block describes either an active circuit or an RC network. Thus, coefficients for the basic blocks have a physical meaning as they are transfer-functions, impedances, admittances. They are expressed as Laplace transforms for capturing frequency dependencies. Also, these elements depend only on the type of the circuit and can be stored in a library. Kirchhoff's laws are satisfied by block matrices.

The block model is similar to the *port network model* in network theory [8]. It has the advantage that the block entities for symbolic analysis are easily identifiable with the specification entities and those of the net-list generation step, as they also consider a block level. Other representations, i.e. Mason's signal-flow graphs [7] do not offer any structural information on how their elements relate to the composing blocks of a design. However, such knowledge is important for structural synthesis i.e. net-list generation.

Edges is the function:

$$\boxed{Edges : (Blocks, N) \times (Blocks, N) \rightarrow Wires} \quad (1.3)$$

$Edges((block_i, I), (block_j, J))$ is either the wire that interconnects port I of $block_i$ to port J of $block_j$, or $\emptyset$ if the two ports are not linked together.

Inputs and *Outputs* are the sets of all input and output ports of the SFG.

A representation of the model is showed in Figure 3a and Figure 3b.

For the building block it presents its block matrix. Input ports are small white squares and output ports are small black squares.

An *Analog Performance Tree* (APT) describes the computational path for calculating linear (linearized) attributes of an SFG. It indicates how attributes for higher-level blocks are calculated using parameters of its composing blocks and composition functions that depend on the connection style. APT can be used to describe linear attributes, i.e AC small-signal parameters: overall transfer functions for implementations, input/output impedances etc.

An APT has two types of nodes. (1) *Leaf nodes* that describe basic blocks of an SFG (blocks that are not further decomposed). Leaf nodes contain equations for relating block parameters to their design parameters. These equations are not necessarily linear and describe elements, i.e. DC biasing, manufacturability aspects. (2) *Internal nodes* that describe the connections of leaf nodes or internal nodes. They correspond to series, feed-back, feed-forward etc. connections. Depending on the connection, an internal node is annotated with a composition (solving) function that links overall parameters to the parameters of composing blocks. Figure 2c depicts an APT.

System Decomposition and Modeling of Block Interactions by Tearing

Assume that two blocks are interconnected as in Figure 4. Knowns for the block matrices are bordered by a square.

Tearing Rule [5]: Wire (v_x, i_x) can be torn apart so that v_x is a known and i_x an unknown for the first block. For the second block, i_x is a known and v_x an unknown.

The other way of assigning the values v_x and i_x as knowns-unknowns is also possible. The only restriction is that, when selecting the block matrices for the basic blocks, the known- unknown character assigned by tearing to the values v and i of the torn wires, must be reflected.

Tearing decomposes a system into smaller sub-systems. It is a key concept for our technique of building APTs, as it allows identification of connection styles i.e. series, feed-back, etc. After handling the sub-systems their APTs have to be re-composed for finding APTs for the system parameters. Assume that v_i and v_j are the knowns in Figure 1.4. After we wrote equations for unknowns $i_i \dots i_k$ and i_x for block 1 and unknowns $i_j \dots i_k$ and v_x for

Figure 4 Decomposition through wire tearing

block 2, we eliminated unknowns v_x and i_x from the equation set. Following expressions resulted for unknowns i_k of block 1 and block 2 depending only on knowns v_i and v_j . The parameters for the re-composed system are:

$$i_k = \Sigma(t_{ki} + \frac{\beta\gamma_k r_i}{1-\alpha\beta})v_i + \Sigma(\frac{p_j\gamma_k}{1-\alpha\beta})v_j, \text{ for block 1.} \quad (1.4)$$

$$i_k = \Sigma(\frac{r_i\theta_k}{1-\alpha\beta})v_i + \Sigma(q_{kj} + \frac{\alpha\theta_k p_j}{1-\alpha\beta})v_j, \text{ for block 2.} \quad (1.5)$$

The equations show how the overall parameters of the re-composed system depend on those of the two sub-systems. They can be easily coded into a computer program for hierarchical analysis of a system.

4. TOP-DOWN SYMBOLIC ANALYSIS METHOD FOR BEHAVIORAL SYNTHESIS

4.1 SOLVING FUNCTIONS FOR LINEAR SYSTEMS

As already seen in Section 2, our symbolic analysis algorithm constructs an APT that relates a linear parameter of the overall system to the parameters of its basic blocks. The structure of the APT remains constant during the entire synthesis algorithm as an APT reflects the invariant control (SFG) algorithm of a system. The synthesis flow instantiates different physical blocks for the leaf blocks of an APT. Thus, the block matrices of the leaves have to be correspondingly updated so that the overall-system parameters relate to the physical parameters of the leaves.

It is worth mentioning that the APT could be set-up by using only tearings and re-compositions. However, we defined a library of popular connection styles and their solving functions so that it avoids repeated tearing and re-composition for the same connections. Solving functions relate parameters of the overall connection to the parameters of its composing blocks. The algorithm for APT construction repeatedly decomposes a system through tearing so that *basic* connection styles (patterns) i.e. series, feed-back, feed-forward, divergent, etc. are identified. After the entire system is decomposed into basic connection patterns, the symbolic expressions for the sub-systems are re-composed to eliminate the tearing effects.

Selecting a library of solving functions should consider not only how often their related connection patterns are used, but also the size of their symbolic expressions. For a set of connected blocks, the size of the overall symbolic expressions increases as the number of unknowns and symbolic parameters is bigger. For example, for a connection type characterized by 9 equations with 9 unknowns and 24 symbolic parameters, the overall symbolic expressions in

	# of blocks	# of unknowns	# of symb. param.	# of knowns	# of operations			Description of connection
					Addit.	Multipl.	Total	
1	2	4	8	2	7	17	24	Series connection
2	2	6	13	4	20	29	49	Series connection 2
3	2	7	25	3	105	233	338	Series connection 3
4	3	7	12	3	23	53	76	Convergent connection
5	2	6	8	2	30	55	85	Feed-back connection 1
6	2	5	13	2	43	131	174	Feed-back connection 2
7	2	5	13	2	14	23	37	Connection of teared wires

Table 1 Set of connection patterns and their symbolic expressions

unoptimized form implied 264 additions and 714 additions. This result shows that for connected blocks with 24 symbolic parameters the size of the symbolic expressions tends to be very large.

The previous discussion motivates two aspects: (1) The library of solving functions has to be designed so that the number of symbolic parameters is not very high. If a connection pattern involves more parameters, then a decomposition through tearing is preferred. (2) As expression sizes grow rapidly, optimizations i.e. finding common sub-expressions, factorization are very important. This also motivates the feasibility of using a library of solving functions, where each function is carefully optimized.

In our experiments, we used the solving functions for the connection patterns in Table 1. Functions have similar forms to those in Section 2, and were too big to be introduced in the paper. Different tearing situations are handled by patterns 1, 2 and 7, while the rest of the patterns correspond to connection styles that we often found for systems. Connection patterns 2, 3 and 6 were previously exemplified in Figure 2b. Table columns have the following meaning: Column 2 indicates the number of blocks in a connection. Columns 3, 4 and 5 show the number of unknowns, symbolic parameters and knowns of the equation sets for a connection. Columns 6, 7 and 8 describe the size of the solving functions expressed as number of additions and multiplications.

4.2 ALGORITHM FOR BUILDING THE APT OF A SYSTEM

The algorithm for building the APT traverses the hierarchical specification in a top-down fashion and infers symbolic expressions, without explicitly solving a system of equations as in traditional symbolic methods. If required, wires are torn so that already existing patterns are reused. By this strategy, both the required memory and the set-up time of an APT are reduced.

The algorithm for building the APT of a system is depicted in Figure 5. It starts with the top-most abstraction level, and traverses top-down the overall hierarchy. First, it verifies if blocks are connected in one of the known connection styles (line 1 in Figure 5). These styles are either a library pattern or a previously handled connection pattern. If the connection style is known then a new node in the APT is produced (line 2 in Figure 5). Its *function field*

314 **procedure** *built_APT* (*current_hierarchy_level* in a system) **is**
 (1) **if** blocks of *current_hierarchy_level* are connected in a
 library pattern or a previously handled connection pattern **then**
 (2) Build a new APT node, and label it with the function of the
 already handled pattern. Actual function parameters are
 parameters of the blocks that form *current_hierarchy_level*;
 else
 (3) Identify a set of already handled patterns so that each block
 in *current_hierarchy_level* belongs to only one pattern;
 (4) Tear the wires that do not belong to any identified pattern;
 (5) Produce APT nodes for the teared wires. Label them with the
 functions that describe the re-connection of torn wires;
 end if;
 (6) **for each** *block* $\in$ *current_hierarchy_level* **do**
 (7) **if** (*block* has a hierarchical structure) **then**
 (8) **call** *built_APT* (*block*) for finding the parameters of *block*;
 end if;
 end for;
end procedure *built_APT*.

Figure 5 Algorithm for building the APT

points to the solving function of the already encountered pattern. Actual parameters for the function are parameters of the blocks forming the connection. Thus, uniformity inside a system is considered by re-using solving functions for similar patterns. As shown by experimental results in Section 5, compact APT with less nodes result. If the connection style was not previously encountered then a number of wires are torn, so that each block belongs to a known connection pattern and no block appears in two patterns (lines 3,4,5 in Figure 5). APT nodes are produced to reflect the tearing process (their functions correspond to pattern 7 in Table 1). After processing the current hierarchy level and checking if any of the blocks has a hierarchical structure, the algorithm continues by recursively creating APTs for these blocks (line 8 in Figure 5).

It is worth mentioning that the algorithm builds APT nodes only for the parameters required in the symbolic analysis. For example, assume that only parameter R_{11} of the overall block matrix is needed and not parameter R_{12} . Obviously, no computations for parameter R_{12} should be performed by an analysis method. This requirement can be easily accomplished in a top-down method, where at each step the required parameters can be identified. Only the required parameters are expanded during the traversal of the hierarchy. In a bottom-up method, knowledge about the usefulness of certain computations is hard to establish. This is also an advantage of our method over traditional bottom-up approaches.

4.3 ANALOG PERFORMANCE TREE REDUCTION

Reduction rules attempt to decrease the number of arithmetic operations performed for numerically evaluating a symbolic APT by executing simplifications i.e. addition of value 0, subtraction of two identical expressions, multiplication with values 0 and 1, etc. Reducing the number of operations is important for a practical synthesis tool, as numerical evaluations of an APT are repeatedly performed inside the synthesis loop. Besides, the algorithm for

Figure 6 Reduction of the APT for ladder structure

APT building does not consider what the actual block parameters represent (i.e. impedances, admittances, etc.). This information is not available at the time the APT is constructed. During synthesis, blocks are instantiated with physical components. Simplifications of an APT can result as a consequence of block parameters having similar values.

We refer to the symbolic expression of the input resistance of the RC ladder in Figure 6 to motivate how reduction rules can be effective in APT simplifications. The input resistance is defined as $R_{in} = \frac{v_i}{i_i} = \frac{1}{R_{11}}$, where R_{11} pertains to the equivalent block matrix of the ladder. Figure 1.6 depicts the block matrices after mapping each block to electronic components. The selected knowns/unknowns for each block are also indicated in the figure (unknowns appear on the left hand side of the matrix equations). The APT for R_{11} is represented in the right part of the figure. Due to specific values i.e. $A_{12} = A_{21} = 1, A_{22} = 0$ the APT can be reduced, the simplified sub-trees being indicated as bordered.

5. EXPERIMENTAL RESULTS AND DISCUSSION

Experiments of our symbolic method were carried out on a set of five examples. The observed elements were: the memory size for the symbolic APTs, the number and types of the solving functions that were shared among different connections and the CPU time for setting-up the APTs. Experiments were run on a SUN Sparc 5 workstation. The five examples are: *ladder7* an RC ladder with 7 stages [10], *ladder21* an RC ladder with 21 stages [10], *filter4* a 4th order low-pass filter network [11] [4], *filter8* an 8th order band-pass filter [11], and *cascode op amp* the small-signal model for a cascode op amp [2] for which we manually built a hierarchical description.

The results of our experiments are summarized in Table 2 Columns 2 and 3 indicate the sizes of the analyzed systems (circuits) in terms of their

Example	# of unknowns	# of symbols	Required memory		Function sharings			CPU Time (ms)
			# of nodes if no sharing	# of nodes with sharing	Total	Same param.	Distinct param.	
ladder7	7	17	108	52	4	2	1	9.26
ladder21	21	59	503	62	25	11	14	14.35
filter4	42	74	1115	115	66	65	1	106.15
filter8	72	136	910	102	44	15	29	50.86
cascode op amp	36	64	120	65	4	4	-	16.45

Table 2 Experimental results for the symbolic analysis method

number of unknowns and the number of symbolic parameters in their block matrices. The sizes of the resulting APTs without any function sharings are indicated in Column 4. Column 5 shows the APT sizes if solving functions were shared among similar connections. Columns 6, 7, and 8 show the nature of sharings. Column 6 describes how many solving function were re-used. Column 7 indicates the number of cases where the same block matrix coefficient of a complex connection was re-used. Column 8 presents the number of situations where solving functions were re-used for connections involving different blocks. Finally, column 9 shows the CPU execution time.

Comparing Columns 4 and 5, we see significant reductions of APT sizes resulting from re-using solving functions for the connections. Column 5 shows that APTs tend to be small even for systems with a larger number of unknowns. The reason is that for any encountered connection type only one replica of its solving function is produced. For structures built by cascading similar stages, the method is very effective, as additional APT nodes are required only to indicate how the actual block parameters are passed to the solving functions for the connection of a stage. The growth of APT sizes is linear with the number of parameters. Columns 6, 7 and 8 indicate that for small systems, i.e. *ladder7* and *cascode op amp*, sharings of solving functions correspond to the block matrix parameters of a connection that is re-used in several places. For bigger systems i.e. *ladder21* and *filter8*, a significant number of solving functions were shared among connections with distinct blocks. Column 9 shows that execution times were small even for larger examples.

By experimenting with our reduction rules, we observed that they are effective for connections between physical blocks. These blocks have similar symbolic terms, and hence simplification, i.e. term cancelations were more probable. At higher levels, expressions for block parameters are more abstract and tend to have less common elements.

As opposed to other decomposition methods [4] that require a partitioning of the overall model, our technique uses information on hierarchy available in a specification. The technique can be adapted to work in conjunction with a partitioning algorithm that finds connection patterns in a system. Similar to BDD-like representations [10], APTs are a compact representation. BDD-like representations avoid creating any arithmetic operators because of their semantics. APTs require extra memory as they explicitly indicate the arithmetic operations. However, they gain by using higher-order functions for describing complex connections. Similar to [9], our technique is *lazy* as it computes only

sub-expressions that are required for the overall parameters. Lazy computation is natural in a top-down approach, where the elements to be calculated are known and propagated to the lower levels.

6. CONCLUSION

This paper describes a novel symbolic analysis technique that exploits the hierarchy and uniformity of a system for producing compact symbolic expressions for system parameters. Symbolic expressions are built in a top-down fashion by applying the *tearing* decomposition method. The algorithm inspects the hierarchical structure of a system, and tears apart wires so that predefined block connection patterns are recognized. The method is useful for synthesis as it avoids repeated recomputing of symbolic models inside an exploration-loop.

We identified more directions for future work. (1) In the current approach, if more wires have to be torn then their tearing order is arbitrarily selected. We plan to analyze the impact of tearing order on produced symbolic expressions. (2) The approach we used for expression reduction was not very effective for connections at the higher levels of the hierarchy. This limitation is critical for strongly connected blocks as it requires many tearing and re-connection steps.

9. ACKNOWLEDGMENT

This work was sponsored by the USAF, Air Force Research Laboratories, Wright Patterson Air Force Base under contract number F33615-96-C-1911.

8. REFERENCES

- [1] A. Doboli *et al*, "Behavioral Synthesis of Analog Systems using Two-Layered Design Space Exploration", *Proc. of DAC*, 1999, pp.951-957.
- [2] R. Geiger, P. Allen, N. Strader, "VLSI Design Techniques for Analog and Digital Circuits", McGraw-Hill, 1990.
- [3] G. Gielen, W. Sansen, "Symbolic Analysis for Automated Design of Analog Integrated Circuits", Kluwer Academic Publishers, 1991.
- [4] M. Hassoun, P.-M. Lin, "Hierarchical Network Approach to Symbolic Analysis", *IEEE Trans. on Circuits and Systems*, April 1995, pp.200-211.
- [5] G. Kron, "Diakoptics - The Piecewise Solution of Large-scale Systems", Mac Donald & Co., London, 1963.
- [6] P.-M. Lin, "Symbolic Network Analysis", Elsevier, 1991.
- [7] K. Ogata, "Modern Control Engineering", Prentice-Hall, 1990.
- [8] T. Schubert, E. Kim, "Active and non-linear electronics", J. Wiley, 1996.
- [9] S. Seda *et al*, "Lazy-Expansion Symbolic Expression Approximation in SYNAP", *Proc. of ICCAD*, 1992, pp.310-317.
- [10] R. Shi, X. Tan, "Symbolic Analysis of Large Analog Circuits with Determinant Decision Diagrams", *Proc. of ICCAD*, 1997, pp.366-373.
- [11] J. Starzyk, A. Konczykowska, "Flowgraph Analysis of Large Electronic Networks", *IEEE Trans. on Circuits and Systems*, March 1986, pp.302-315.
- [12] Q. Yu, C. Sechen, "A Unified Approach to the Approximate Symbolic Analysis of Large Analog Integrated Circuits", *IEEE Trans. on Circuits and Systems*, August 1996, pp.656-669.