Behavioral Synthesis of Analog Systems using Two-Layered Design Space Exploration*

Alex Doboli, Adrian Nunez-Aldana, Nagu Dhanwada, Sree Ganesan, and Ranga Vemuri
Laboratory for Digital Design Environments, Department of ECECS,
University of Cincinnati, Cincinnati, OH 45221
{adoboli, anunez, nagu, sganesan, ranga}@ececs.uc.edu

Abstract

This paper presents a novel approach for synthesis of analog systems from behavioral VHDL-AMS specifications. We implemented this approach in the VASE behavioral-synthesis tool. The synthesis process produces a netlist of electronic components that are selected from a component library and sized such that the overall area is minimized and the rest of the performance constraints such as power, slew-rate, bandwidth, etc. are met. The gap between system level specifications and implementations is bridged using a hierarchically-organized, design-space exploration methodology. Our methodology performs a two-layered synthesis, the first being architecture generation, and the other component synthesis and constraint transformation. For architecture generation we suggest a branch-and-bound algorithm, while component synthesis and constraint transformation use a Genetic Algorithm based heuristic method. Crucial to the success of our exploration methodology is a fast and accurate performance estimation engine that embeds technology process parameters, SPICE models for basic circuits and performance composition equations. We present a telecommunication application as an example to illustrate our synthesis methodology, and show that constraint-satisfying designs can be synthesized in a short time and with a reduced designer effort.

1 Introduction

The complex task of synthesizing analog and mixed analog-digital systems is typically approached using a hierarchically organized top-down design paradigm, across successive levels of abstraction [5] [11]. In the most general case, such a synthesis environment starts from specifications and design/performance constraints at the *system-level*, moves down to the *circuit-level*, and finally considers the *layout-level*. Recent research on analog synthesis exclusively targets circuit-synthesis and layout generation, which are design activities at lower levels of abstraction. Circuit synthesis [14] [19] [20] assumes a known circuit-topology, and searches for physical dimensions of transistors, so that circuit-level performance attributes, i.e. bandwidth, slew-rate, power, are optimized. Layout tools [6] perform cell placement and routing, with respect to physical constraints, i.e. parasitics, cross-talk, latch-up. However, for achieving automated analog-system synthesis or mixed-signal synthesis, bridging the gap between system and circuit level designs becomes of crucial importance. Traditionally, this task is known as *High Level Synthesis* (HLS).
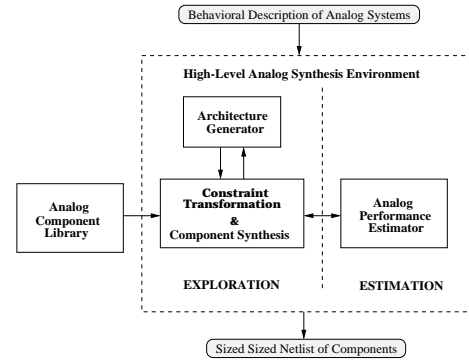
Figure 1: **Exploration and estimation based approach in VASE for analog behavioral synthesis**

The goal of HLS is the inferring of all required design elements, so that circuit synthesis can follow next. Three main synthesis tasks accomplish this goal: *architecture (system topology) generation*, *component synthesis*, and *constraint transformation* (from system-level to component-level). System-level design decisions such as selecting a particular system topology have a great impact on the overall performance characteristics but their quality is difficult to estimate, unless the design is completed until the physical layout stage. A straightforward synthesis approach would consider a flat, physical-level representation of the system, but this is un-realistic and cumbersome due to the immense design space to be explored. Thus, the essential challenge for any realistic synthesis tool is to make an effective compromise between the importance of the system-level design decisions and the large number of solution points which have to be analyzed. Basically, a HLS tool relies on two aspects for tackling this trade-off: (1) an effective mechanism for *design space exploration* and (2) a fast and accurate method for *estimating* the quality of the visited solution points. Moreover a practical approach for design space exploration requires a procedure for *design space pruning* that enables discarding of poor solutions early in the search, and a systematic way of *structuring and visiting the design space* for biasing the search, towards good solutions, aided by analog knowledge.

This paper presents the VASE (*VHDL-AMS Synthesis Environment*) methodology for synthesis of analog systems from VHDL-AMS [1] behavioral specifications. The result of the synthesis process is a sized net-list of electronic circuits such that the ASIC area is minimized and the rest of the system-level performance constraints, such as power, bandwidth, etc., are satisfied. VASE attempts to achieve this goal by performing a two-layered, optimization-based, design-space exploration, as depicted in Fig 1.

The top-most exploration step is the *Architecture Generator*, which generates alternate system topologies from the input specification. The *Constraint Transformation and Component Synthesis* step considers each system topology and obtains the constituent components, their topologies and their respective design/performance

constraints. For architecture generation, we suggest an optimal *branch-and-bound* [15] algorithm that enumerates all distinct alternatives for system topologies. This methodology is motivated by the fact that the system topology has a decisive impact on the overall performance, and also that the number of distinct alternatives tends to be small. Besides branch-and-bound permits effective design-space pruning by means of its bounding rules. Since their designated solution-spaces are inherently much larger, component constraint transformation and component synthesis rely on a Genetic Algorithm (GA) based heuristic method [13].

Critical to the success of our exploration-based synthesis methodology is the accuracy of analog performance estimation. We use an *Analog Performance Estimator* (APE) to provide fast and accurate estimates of analog system performance at various levels of abstraction. The APE is a hierarchical analog estimation engine that contains performance models of analog components (SPICE models for basic circuits and performance composition equations at other levels of abstraction). We present the APE and how it is employed in the constraint transformation and component synthesis step, and in ranking the quality of the explored solutions.

The paper is organized as seven sections. Section 2 presents the related work, and highlights the contributions of this paper. Then, we discuss our analog synthesis approach in Section 3. Section 4 introduces the analog performance estimator that guides the exploration algorithm. In Section 5, the algorithms for architecture generation and component synthesis are described. Section 6 motivates the effectiveness of our synthesis flow by means of a case study. Finally, we provide our concluding remarks.

2   Related Work

Most of the research on analog-circuit synthesis performs either solution-space exploration [4] or constraint transformation [2] [16] as they tend to regard the two steps as being unrelated. Optimization based circuit synthesis assumes a known circuit-topology and searches for the transistor dimensions that optimize performance attributes. The developed tools start by producing circuit-performance models that relate performance attributes to design parameters. Various methods are suggested for this task. Simplified symbolic equations are calculated for performance parameters in [14]. [20] includes the Kirchhoff's laws into the cost function for the optimization algorithm and [17] relies on the square-law equations of the CMOS op amps. Next, the performance model is used by an optimization algorithm, i.e. simulated annealing, non-linear or geometric programming to find values for the physical design parameters. Finally, the quality of a solution point is evaluated through different simulation methods, i.e. symbolic simulation [14], general-purpose simulators (i.e.*SPICE*) [19], or approximate simulation [20]. Research that concentrates on constraint transformation is very limited. [2] suggests a semi-analytical approach that uses the circuit transfer function to relate performance attributes to the design parameters and parasitic effects. [16] uses interval analysis for circuit synthesis, defined as the task of partitioning design constraints together with a partitioning of a circuit into smaller blocks. Common to all mentioned approaches is the focus on circuit-level synthesis, without addressing system-level aspects. Second, they discuss either solution-space exploration or constraint transformation, although both tasks are compulsory in a complete synthesis tool.

In this paper, we propose an automated method for conducting high level synthesis at the analog system level. To the best of our knowledge, this is the first attempt to develop a complete methodology for analog synthesis from behavioral, system-level specifications. The main contributions of this paper are:

- We present a novel system-level synthesis approach that starts

```
ENTITY telephone IS                          IF (c1='1') USE
  PORT (                                        rvar == r1c;
     QUANTITY line: IN real; -- IS voltage    ELSE
     QUANTITY local: IN real; -- IS voltage     rvar == r1c + r2c;
     QUANTITY earph: OUT real;                END USE;
                        -- IS voltage         PROCESS (line'ABOVE(Vth)) IS
                        -- limited            BEGIN
                        -- drives 270 O at 285 mV peak   IF (line'ABOVE(Vth) = TRUE) THEN
   )                                               c1 <= '1';
END ENTITY;                                      ELSE
                                                   c1 <= '0';
ARCHITECTURE behavioral OF telephone IS          END IF;
  QUANTITY rvar: real;                         END PROCESS;
  SIGNAL c1: bit;
  earph == (Aline * line + Alocal * local) * rvar;   END ARCHITECTURE;
```

Figure 2: **VHDL-AMS specification for the receiver module**

from behavioral specifications and performs architecture generation, constraint transformation, and component synthesis.

- We discuss a *branch-and-bound* algorithm for system-level architecture generation. As opposed to other optimal methods, such as non-linear programming, *branch-and-bound* offers an improved potential for solution-space pruning. Space pruning is crucial for early discarding of infeasible solutions.

- We propose a genetic algorithm for component constraint transformation and component synthesis. Constraint transformation is also described as an exploration problem, an alternative approach to the analytical methods. Analytical methods are elegant and exact but require extensive analog expertise and tend to be complex even for simpler cases [4].

- The quality of the explored solutions is evaluated through fast and accurate analog performance estimation, instead of the traditional simulation approach [19] [20]. Inside an exploration loop, obtaining reasonably accurate performance values quickly is far more important than calculating them with a high accuracy.

3   The Synthesis Approach

With a motivating example, this section details our methods for architecture generation, constraint transformation, and component synthesis and presents the order which ties them into a meaningful synthesis-flow. Also, we discuss our notation for describing implementation-independent, behavioral specifications of analog systems.

For explaining the VASE synthesis-flow, we use an analog telephone set as an illustrative example. A more detailed description of the telephone set is provided in Section 6. The telephone consists of a transmitter and receiver modules. The transmitter amplifies the incoming audio signal from the caller, and also matches the impedance of the line. The function of the receiver is to provide an audible output signal to the earphone of the telephone set. It amplifies incoming signals transmitted from the calling part and those produced locally by its own microphone amplifier and transmitter module. Besides, it automatically compensates losses introduced by different telephone line lengths.

Implementation-independent, behavioral specifications essentially describe the system functionality, and they are expressed in VHDL-AMS [1], in our synthesis environment. We identified a VHDL-AMS subset for synthesis, and we specified several real-life-inspired examples with our VHDL-AMS subset [8]. In the process, we learned that a declarative notation is needed for expressing properties of signals and ports. It is very common that ports must have a limited input/output impedance, which is a property of the system, and not a functional aspect described as an operational block. These annotations are essential in guiding the synthesis process. Figure 2 presents the annotated VHDL-AMS code for a simplified version of the receiver module (without the DTMF signal).
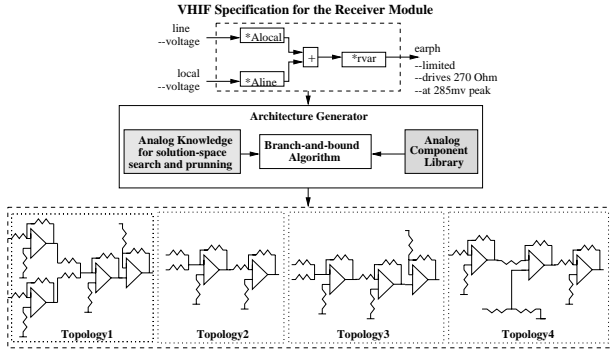
Figure 3: **Architecture generation**

Figure 4: **Constraint transformation**

We developed a compiler [10] that translates VHDL-AMS specifications into their equivalent VHIF representations, and these are the inputs for our synthesis environment. VHIF is a hierarchical intermediate format [9] that algorithmically represents continuous-time behavior as signal-flow graphs, that most naturally represent a system description at the level of components. Besides, the VHIF representation is designed to ensure that all its constituent elements can be implemented with components from an analog library [3]. Signal-flow graphs provide exact knowledge about the flow and processing of signals in a system. The output of each block in the graph depends only on its input signals and the functionality (operation) of the block. The top-most part of Figure 3 depicts the signal-flow graph obtained for the receiver module after compiling the input VHDL-AMS behavioral specification.

The Architecture Generator maps a VHIF representation, into netlists of circuits at the component level, which is the highest level of abstraction for system-level designs [12]. For producing all possible mappings, the *branch and bound* algorithm relies on VHIF representations for all components in the library [3]. The analog library contains topologies for some of the most commonly used analog circuits: operational amplifiers, filters, converters, etc. Besides the VHIF representations, the library also stores the constraints for input/output signals, impedances, etc. under which a circuit realizes its expected function. During the exploration, the algorithm also prunes the solution space and discards some infeasible solutions, without doing component synthesis and constraint transformation (eg. topologies with many opamps are immediately rejected by using knowledge about the minimum area of an opamp). Also, analog knowledge and common-sense rules can bias the exploration process towards better architectures. The exploration first considers topologies with a reduced number of op amps, as they tend to have a smaller area. Such "rules of thumb" do not influence the final topology, but they can increase the effectiveness of pruning, if they provide a good solution. Figure 3 illustrates the architecture generation step for the receiver module. Four different system-level topologies are generated for the signal flow graph representing the receiver module. The quality of each topology is estimated by instantiating components from the library and evaluating their performance using the APE.

For all components in an architecture produced by the Architecture Generator, the Constraint Transformation and Component Synthesis step fixes real circuit topologies and approximate transistor sizes, and then estimates the resulting performance. Concurrently, this step also transforms system-level constraints (i.e. area, power, gain, bandwidth) onto component design parameters (bias current, gain-w/l ratio, etc). Figure 4 exemplifies this step for *Topology 2* (the topology with minimum number of opamps) for the receiver module. The main features of this step are a GA based search engine, a component characterization method, and an analog
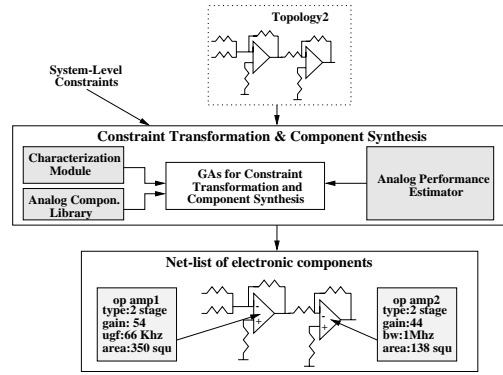
performance estimator. Both component characterization and constraint transformation are based on a directed interval representation of the relationships between design and performance parameters.

4 Analog Performance Estimation

The analog performance estimator [18] accepts design parameters (bias current etc) of an analog circuit along with it's topology and determines the performance parameters (area, UGF, slew rate, etc) of the circuit with the anticipated sizes of the circuit elements. The APE is structured hierarchically, and contains technology process parameters, SPICE models of the circuit elements, and performance composition equations for determining the performance of the circuit at different levels of abstraction. These levels include basic circuit elements (MOS transistors, resistors, capacitors), simple analog circuits (current mirrors, differential amplifiers etc), operational amplifiers in various configurations and analog library cells.

The estimation and sizing are performed bottom up starting from the transistor level through to the system level. At each level in the hierarchy, the current design parameters are obtained from the parent level and are further decomposed into the parameters for the sub-components. This eventually leads to the rough sizing of the transistors and generation of the performance estimates. The estimation at different levels in the APE is as following:

- CMOS **transistor level :** is the lowest level in the hierarchy of APE. The transistor is sized based on its DC operating point and the fabrication process parameters. In a CMOS transistor level, the parameters specify the electrical and process characteristics of the devices. Using these parameters and the transistor SPICE models, the large signal and small signal models of a CMOS transistor, capacitor and diode can be evaluated.

- **Basic Circuit Level :** The elements in this level include DC-bias voltages, current sources, gain amplifiers, output buffers, differential amplifiers, and differential-single ended converters. This level contains several topologies for each component, e.g. a current source can be implemented as a Cascode or a Wilson topology. APE contains a set of symbolic equations which relate the performance of the components to the circuit topology. The small signal characteristics of the transistors and the symbolic equations of the circuit are used to estimate the performance parameters of this circuit. For instance, the typical performance parameters estimated for a differential amplifier are A_{dm}, CMRR, gate area, UGF, Zout, DCpower and slew rate.

Table 1 compares the APE estimations and SPICE simulation results of some basic components after being sized according

Topology	Gate Area μ^2		UGF MHz		DC Power mW		Gain		CMRR dB		Current μA	
	est	sim	est	sim	est	sim	est	sim	est	sim	est	sim
CurrMirr	165.7	165.6	-	-	.5	.56	-	-	-	-	100	97.9
Wilson	383.3	383.1	-	-	.5	.58	-	-	-	-	100	106
GainNMOS	101	101	15.7	15.2	.6	.82	-8.5	-8.0	-	-	-	-
GainCMOS	101	101	26.7	30.3	.62	.8	-19.0	-16.9	-	-	-	-
GainCMOSH	101	101	8.3	8.7	.23	.22	-5.1	-5.3	-	-	-	-
Follower	123.4	123.4	-	-	.5	.64	.8	.81	-	-	100	128
DiffNMOS	47.2	45.8	6.0	6.5	.5	.49	-10	-10.2	78	78.5	1	1.02
DiffCMOS	15.8	15.8	4.4	4.6	.5	.49	1000	1055	128	-	1	1.02

Note 1: - not applicable for the topology

Table 1: **Estimation vs SPICE simulation for basic circuits**

Ckt	Power mW		Adm		UGF MHz		Ibias μA		Zout $Kohm$		Gate Area μ^2		CMRR dB		SR $V/\mu S$	
	est	sim	est	sim	est	sim	est	sim	est	sim	est	sim	est	sim	est	sim
OA1	.29	.28	206	223	1.3	2.1	1	.9	1	.9	4885.7	4884.4	129.7	135.3	0.1	0.1
OA2	.17	.19	374	380	8.0	13.7	2	1.9	1	.9	2379.6	2376.2	141.8	157.4	0.2	0.2
OA3	.15	.16	167	170	12.4	9.8	1.5	1.4	2	1.8	1010.8	1010.8	133.8	125.1	.15	.16
OA4	.24	.29	514	489	2.6	4.0	1	1.1	-	-	696.9	696.9	98.6	100.8	4.5	4.3

Note 1: OA1, OA2, OA3 topology: Wilson, DiffCMOS, Output Buffer. OA4 topology: Mirror, DiffCMOS

Table 2: **Estimation vs SPICE simulation of opamp's**

to user-specified requirements in terms of voltage for DC voltage sources, current for current sources, gain for gain stages etc. This table shows that the models used in the APE are reasonably accurate. [1]

- **Operational Amplifiers :** This level consists of topologies of operational amplifiers. Each stage in the operational amplifier is composed of basic circuit elements. The performance of the operational amplifier is evaluated using the attributes of these basic circuit elements. For example, the UGF of the operational amplifier is computed as a function of gain and UGF of the differential amplifier and the gain stage, and the compensator capacitor.

  Table 2 compares the performance estimation and SPICE simulation results of several operational amplifiers after being sized. In each case, Adm and UGF values were specified by the user along with the specification of an opamp topology. Again these results show that the opamp models used in APE are reasonably accurate.

- **Analog Module Level :** This library of analog modules forms the fourth level in the APE. Each of the modules are built using the operational amplifiers, elements from the basic circuit level, transistors, resistors and capacitors. This level consists of elements like inverting/non-inverting amplifiers, integrators, adders, sample-and-hold circuits, A-D, D-A converters, etc. The performance of these components is estimated using the operational amplifier attributes and the equations relating the ideal behavior of the component with the non-ideal characteristics of the operational amplifier.

- **System Level :** At this level, the APE estimates the performance of a system comprised of blocks in one of the basic configurations (cascaded, split, join) given the performances of the individual blocks in the system. The netlist is represented as a signal flow graph, where each node is an estimated element from the analog module level. APE evaluates the performance of the signal flow graph using the Mason's rule.

Above this is a wrapper that accepts an arbitrary system level net-list, partitions it into basic configurations and generates a performance estimator specific to the given system netlist. The APE also has built into it some rules that detect the cases where transistors in the design go out of saturation, or when some basic conditions governing the functionality of the circuit have been violated.

The APE, through several examples, has been proven to be fast and accurate. Components at the analog module level are estimated in tenths of seconds with maximum estimation error of 10% when compared with SPICE. This makes APE suitable for use within our exploration-based methodology.

---

[1] Estimated area deviates slightly from the area of the simulated circuit due to the fact the estimated transistor sizes may contain fractional dimensions not permitted by the layout design rules. These sizes are rounded up to the nearest dimension allowed by the design rules before performing simulations. All area numbers denote active transistor areas, with no routing taken into account.

5 Design Space Exploration

This section presents the design-space exploration phase of our behavioral synthesis methodology. The exploration phase consists of two distinct parts: (1) *architecture generation* and (2) *constraint transformation and component synthesis*. The goal of the exploration phase is to map a set of VHIF signal-flow graphs for a system into a net-list of sized components such that all performance constraints are satisfied and the total ASIC area is minimized.

5.1 Architecture Generator

The algorithm for architecture generation contemplates different component-level mappings for a VHIF representation, while attempting to achieve its goal of minimizing the ASIC area. At the level of the system topology, the goal of area minimization is addressed by analyzing two possibilities of *hardware sharing*: (1) between blocks in different signal paths, and (2) between blocks of the same signal-flow path. Blocks in distinct signal paths can share the same component, if they have identical inputs, and perform the same operation. Blocks of the same signal-flow path can share a component, if the component implements the overall functionality of the blocks. Any optimal algorithm must analyze all possible mappings, as the two sharing options can conflict each other. Although the problem of architecture generation is NP-hard [15], we decided to solve it optimally by using a *branch-and-bound* algorithm [15]. VHIF representations for real-life examples tend to be of small or medium size, so that it is practical to map them optimally. Besides, an optimal algorithm can be used as a reference for designing future mapping heuristics.

The algorithm for the Architecture Generator is depicted in Figure 5. It maps the signal-flow graph denoted by variable *signal-flow* into the minimum area net-list indicated by variable *net-list*. Variable *opamp nr* represents the number of op amps in a partial mapping. As *branch-and-bound* is a popular algorithm [15], we show only the three elements, that are specific to our problem:

- *Branching rule*: marked with ◇ in Figure 5, describes how distinct mapping solutions are produced for a partial solution-point. It distinguishes all VHIF block-structures, pointed by variable *sub-graph*, with *cur block* as their output block, and which are directly mappable to library components [3]. Besides, the branching rule contemplates two kinds of transformations, in a signal-flow graph. *Functional transformations* replace a particular block structure with a distinct, but semantically equivalent structure, i.e. for improving bandwidth, an op amp is replaced by a chain of two op amps with lower gains [12], or two non-inverting amplifiers are substituted for/by two inverting amplifiers [12], etc. Transformations pertaining to circuit *interfacing* introduce additional circuits, i.e. follower circuits [12], or various input/output stages [12], etc., for diminishing loading/coupling effects among interconnected components.

- *Bounding rule*: marked with □ in Figure 5, eliminates a partial solution, if it finds that the minimum area, which can result, is greater than the area of the best solution found so

procedure *mapping (signal-flow, cur block, op amp nr)* **is**

◇**forall** *sub-graph* ∈ *signal-flow*, that have *cur block* as output block **and** are
  mappable to one library-component;
  in decreasing order of the number of blocks in *sub-graph* **do**
   if sharing is possible **and** library-component for *sub-graph* exists in *net-list* **then**
    make interconnections for *sub-graph* in *net-list*;
    if *signal-flow* was completely mapped **then**
     ● call GA for constraint transformation & component synthesis, and
      save solution if it is best so far;
    else
     signal = select an input signal of *sub-graph*;
     mapping (*signal-flow*, block ∈ *signal-flow* with output *signal*, *opamp nr*);
    end if
   end if
  □**if** (*opamp nr* + nr of opamps for *sub-graph*) * *MinArea* < *current best* **then**
   allocate hardw. component for *sub-graph*, and add it to *net-list*;
   if *signal-flow* was completely mapped **then**
    ● call GA for constraint transformation & component synthesis, and
     save solution if it is best so far;
   else
    signal = select an input signal of *sub-graph*;
    mapping (*signal-flow*, block ∈ *signal-flow* with output *signal*,
     opamp nr + nr of opamps for *sub-graph*);
   end if
  end if
 end for
end procedure

Figure 5: **Algorithm for architecture generation**

far (variable *current best*). The minimum area for a partial mapping is estimated using value *MinArea*, the minimum area of an op amp (with transistors sized to the minimum dimensions).

- *Sequencing rule*: is a heuristic rule, which decides the order in which branching alternatives are traversed. A good sequencing rule can dramatically improve the speed of the overall algorithm, as the bounding rule becomes very efficient, if a high-quality solution is found early. Our sequencing rule first considers branching alternatives, which map a higher number of blocks to one library component, in its attempt to find early a mapping with less op amps. Besides, the algorithm first analyzes the case, where blocks in *sub-graph* share existing components in the net-list, and then maps *sub-graph* to its dedicated hardware component.

The algorithm calls (marked with ● in Figure 5) the GA for the constraint transformation and component synthesis, which calculate approximate performance attributes (power, UGF, slew rate) and hardware area by instantiating op amps with precise circuit topologies, and sizing their transistors. Component selection and constraint transformation is discussed in the next subsection.

5.2 Constraint Transformation and Component Synthesis

The aim of the Constraint Transformation and Component Synthesis step is to instantiate circuit topologies, and compute component design parameters (which correspond to a sizing solution) that satisfy the overall system level constraints. A straightforward approach would simultaneously explore circuit topologies and design parameters, and estimate (using the APE) their impact on the system-level performance. However, through experiments we learned that this approach results in unacceptably long exploration times. Also the quality of the search was not very good. Therefore, we adopted a hierarchical exploration method, that is depicted in Figure 6. The top-most GA distributes the system-level performance constraints onto component performance constraints for the
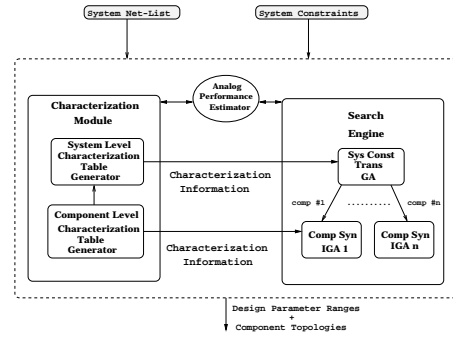

Figure 6: **Constraint transformation and component synthesis**

components. Next, the lower GAs instantiate circuit topologies and find circuit design parameters, so that the previously determined circuit-level performance constraints are met.

System-Level Constraint Transformation We describe the system constraint transformation as a *constraint satisfaction problem* that computes performance values for the components such that system constraints are satisfied. Also the method has to guarantee that the resulting parameter values are realistic for the circuits present in our component library. Our constraint transformation procedure includes (1) a system characterization table for guaranteeing the feasibility of the produced performance values and (2) a GA-based search engine for design space exploration. Besides, the method relies on performance composition equations of the APE to relate circuit performance attributes to system performance values.

System characterization tables provide to the GA-based search engine the analog knowledge required for guaranteeing that certain circuit performance attributes are realistic for the existing library components. For example the system level table of an opamp in a cascaded system of opamps, would give information about how the performance parameters of the opamp (power, UGF, etc) affect the overall system performance (area, power, band width etc). The tables are organized based on the concept of *directed intervals* [7]. Each entry consists of a system performance interval, component performance interval, and a direction attribute that gives the way in which the system performance changes with the component performance increasing in it's range. The tables are dynamically created by calling a Characterization Table Generator. Basically, the characterization method samples the design parameter and component performance space at different points to generate information about the space that could be used by the search engine. Although dynamic in nature, the characterization method takes only a small amount of time and does not involve moving through the entire search space.

The *system constraint transformation* GA accepts a system netlist produced by the Architecture Generator, system constraints and the system characterization table, and computes component performance values that satisfy the system constraints. After initialization, the GA repeats the steps of selection, crossover, mutation and replacement till convergence is reached. The selection step picks the two best solutions from this set. This step makes use of a cost function to evaluate the solution quality. The crossover operator combines two solution to produce a new solution, while mutation operator perturbs the selected solution. Finally, the replacement method replaces the two worst solutions with the newly generated solution. Once the convergence condition has been reached the solution, the solution with best cost function value represents a solution to constraint transformation problem.

The cost function and the GA operators that are specific to our constraint transformation problem are discussed next.
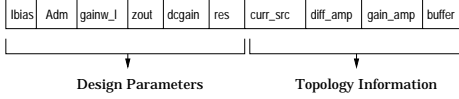
Ibias	Adm	gainw_l	zout	dcgain	res	curr_src	diff_amp	gain_amp	buffer

Design Parameters　　　　　Topology Information

Figure 7: **Solution Representation for comp syn GA**

The *cost function* to be minimized is

$$\frac{1}{N} * \sum_{i=1}^{N} W_i . \mathcal{F}_i$$

where N represents the number of specifications, W_i is the weight associated with that performance specification and $\mathcal{F}_i$ is defined as:

$$\mathcal{F}_i = \begin{cases} 0 & if\ P_{i\_est}\ satisfies\ P_{i\_constraint} \\ \frac{P_{i\_est} - P_{i\_constraint}}{P_{i\_constraint}} & otherwise \end{cases}$$

P_{i_est} is the value for the performance parameter in the current solution, and $P_{i\_constraint}$ is the user specified constraint on that performance parameter. Such a cost function is typical of GAs that handle multiple constraints. During the evolution process, there might be infeasible solutions generated. i.e the design parameters might be assigned values that might lead to a circuit that does not work. Such conditions are detected during performance estimation of the individual components, and the resulting infeasible solutions are removed from the current population by imposing a heavy penalty on them.

Besides the traditional GA operators [13], non-uniform mutation, uniform crossover and uniform selection, our GA for constraint transformation also uses a set of *Directed Interval based Operators* (DIO). These operators to some extent act as local optimization methods, and help in intelligently focusing the search process, once a promising region has been discovered. Therefore, in our GA we start out with the traditional operators of non-uniform mutation and uniform crossover [13], and after some evolution we switch to the DIOs. Also, DIOs are exclusively defined using the system characterization tables, which embed analog knowledge about the components in the library. Thus by their nature, these operators implicitly guarantee that the resulting circuit performance constraints are also realizable with the existing components.

We defined two types of DIOs: mutation and crossover. *Mutation* compares the resulting performance parameters with the user defined constraints to identify those performance constraints that are being violated in the current solution. Then, the characterization table information is used to select which component performance parameter is to be changed and in what direction. Similar to the mutation operator, *crossover* individually evaluates the two parents, and identifies the constraints that are violated. The system characterization table is looked up, and the component performance parameters that correspond to the satisfied performance constraints

Block	Specifications
Echo Cancel	BW = 300Hz-3600Hz, Tx attenuation = -14dB
Dialing Attn	BW = 300Hz-3600Hz, DTMF attenuation = -40dB
Receiver Gain	BW = 300Hz-3600Hz, Rx Gain = 6dB
Rx Zout	BW = 300Hz-3600Hz, Zout = 270 Ω
Current-Voltage	20mA < input Current < 40mA
Transmit Gain	BW = 300Hz-3600Hz, Tx Gain = 40dB
DTMF Gain	BW = 300Hz-3600Hz, Gain = 0dB
Tx Zout	BW = 300Hz-3600Hz, Zout = 600 Ω

Table 3: **Telephone system specification**

are combined together to form one child. The remaining parameters are combined together to form the second child.

Component Synthesis　The component synthesis GA (see Figure 6) is almost identical to the constraint transformation GA. The role of the component synthesis GA is to compute component design parameter values and select component topologies that satisfy the performance constraints generated by the constraint transformation GA. The *cost function* and DIOs are the same as the one used in the constraint transformation GA, but for component performance values being used instead of system performances. However, it uses static *component characterization tables* [7], that are produced only once for each component.

The specific solution representation used by the component synthesis GA is depicted in Figure 7. This has two parts, the first representing the component design parameter values, and the second topology information. Each value in the topology part of the representation indicates the type of topology to be selected from library. Also, each component may have more than one entry in the topology part of the array if that component has sub-components having different topologies.

6　Experiments

This section presents a design example, an analog telephone set, synthesized using our methodology. We have shown that a realistic constraint-satisfying system can be successfully synthesized by our methodology with minimum designer effort in a reasonable amount of time. The importance of design space exploration as well as constraint transformation and component synthesis is also illustrated by the example.

The telephone should work in a reduced audio bandwidth (300 Hz to 3600 Hz). The transmitter amplifies the incoming audio signal from the caller. The communication is full duplex, sent in two wires. Two-to-four wire transformation is performed by the receiver. The receiver reconstructs an incoming signal by subtracting the local transmitted signal (voice and dial tone) from the line. The incoming signal is amplified and reproduced in a telephone speaker. The receiver gain is automatically controlled by the quality of the line, which is quantized by the dc current level of the line. The output impedance of the transmitter is matched to the intrinsic impedance. Table 3 shows the telephone system constraints.

The telephone system was synthesized using the methodology presented in this paper. First, the behavioral VHDL-AMS specifications (see Section 3) were compiled to produce the VHIF signal flow graph. Figure 8 shows the signal flow graph of the system. The architecture generator's branch and bound algorithm produced several alternative mappings for the signal flow graph. For each mapping, constraint transformation and component synthesis then created several topologies. Figure 9 shows four different topologies generated for the receiver and two for the transmitter.

Each of the topologies is then evaluated in order to determine the best topology. Table 4 shows the evaluation results for the receiver and transmitter topologies. It is seen that each sized circuit meets the design constraints. Also, note the reasonable execution times to
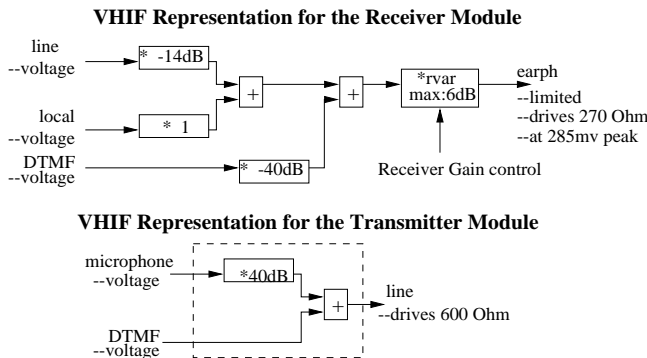
VHIF Representation for the Receiver Module

VHIF Representation for the Transmitter Module

Figure 8: **Signal-flow graph of a telephone system**
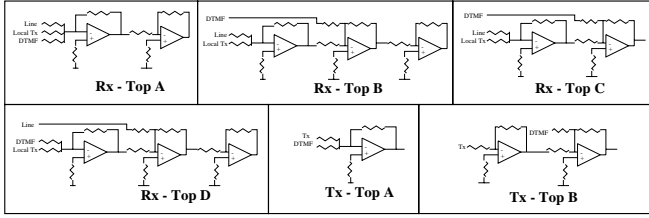
Figure 9: **Receiver and Transmitter topologies**

Parameter	Rx-Top A	Rx-Top B	Rx-Top C	Rx-Top D	Tx- Top A	Tx-Top B
DC Gain (dB)	5.7	5.6	5.5	5.9	40	39.7
UGF (KHz)	18.3	17.0	12.7	52.8	912.3	671.6
F3dB (KHz)	11.2	10.8	8.0	31.6	9.3	10.7
Area (μ^2)	96,695	120,047	96,070	119,400	107,500	44,955
Power (mW)	1.96	2.92	1.96	2.92	0.963	1.96
CPU Time (sec)	670.59	1021.67	702.16	1065.73	363.57	639.02

Table 4: **Evaluation of the topologies**

size and evaluate the topologies. The global system objectives are silicon area and power consumption; hence the receiver topology Rx-Top C and transmitter topology Tx-Top B are selected.

The final designs produced were then simulated using HSPICE to verify their performance. Figure 10 shows the HSPICE simulation results of the transmitter and receiver. The top-left one is the simulated input line, which carries a transmitted (500 Hz), received (2 KHz) and dialing tone signals. The output of the receiver, shown on the top-right, eliminates the transmitted and dialing tone signals and amplifies the calling part (2 KHz). The figures on the bottom left and right show the frequency response of the receiver and transmitter respectively.

A second set of experiments were performed on the telephone system, this time a band-width constraint being added to the goal of minimizing area. Our synthesis methodology selected Rx-Top D for the receiver in this case. This illustrates why the architecture generation step is so important in exploring various mappings. For instance, in this case, Rx-Top A is an obvious mapping; it has two components and smaller area. On the other hand, Rx-Top D has three components; it compromises area for better bandwidth.

7 Conclusions and Future Work

This paper presents a novel methodology for analog-system synthesis from behavioral VHDL-AMS specifications. We implemented this methodology in the VASE behavioral-synthesis tool. The synthesis process produces a sized net-list of electronic circuits such that overall ASIC area is minimized and other imposed system-level
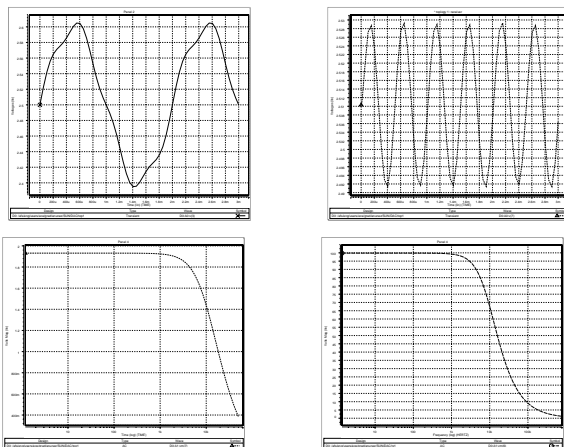

Figure 10: **Receiver and Transmitter HSPICE simulations**

performance constraints, such as power, bandwidth, are satisfied. The gap between system-level specifications and implementations is bridged through design-space exploration, which performs three synthesis tasks: architecture generation, constraint transformation and component synthesis. Crucial to the exploration algorithms is a fast and accurate analog performance estimation engine. The application of our synthesis methodology for a telecommunication application shows that realistic designs can be synthesized in short time and with reduced designer-effort.

Despite the good quality of our synthesis results, there are several directions for future work that may improve or extend the presented work. Possible improvements can address both the analog knowledge and the exploration algorithms used by our tool. The analog performance estimator can be extended to consider more performance attributes, such as noise, or provide more accurate estimations. However, if the inaccuracy of our synthesis process is not within acceptable tolerance, our solution may be used as the initial starting solution for more precise circuit-level synthesis tools such ASTRX/OBLX [20]. Also, we are aware that because of its time-complexity, the architecture generation algorithm we propose might fail for larger designs. Ongoing work attempts to replace the *branch-and-bound* method by a more time-effective exploration heuristic.

References

[1] "IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS changes)", IEEE Std.1076.1.

[2] B.G. Arsintescu, E. Charbon, E. Malavasi, U. Choudhury, W.H. Kao, "General AC Constraint Transformation for Analog ICs", *Proc. of the 35th Design Automation Conference*, pp.38-43, 1998.

[3] P. Campisi, "A CMOS Analog Cell Library for Analog Synthesis Systems", Master of Science Thesis, University of Cincinnati, 1998.

[4] L.R. Carley, G. Gielen, R. Rutenbar, W. Sansen, "Synthesis Tools for Mixed-Signal ICs: Progress on Frontend and Backend Strategies", *Proc. of the 33rd Design Automation Conference*, pp.298-303, 1996.

[5] H. Chang et al, "A Top-Down Constraint Driven Methodology for Analog Integrated Circuits", Kluwer Academic, 1997.

[6] J. M. Cohn et al, "KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing", *IEEE JSSC*, Vol.26, Nr.3, March 1991.

[7] N.R. Dhanwada, A. Nunez, R. Vemuri, "Hierarchical Constraint Transformation using Directed Interval Search for Analog Synthesis" , *Proceedings of DATE'99*, pp.328-335, 1999.

[8] A. Doboli, R. Vemuri, "The Definition of a VHDL-AMS Subset for Behavioral Synthesis of Analog Systems", *IEEE/VIUF BMAS'98*, 1998.

[9] A. Doboli, A. Nunez-Aldana, N. Dhanwada, R. Vemuri, "VHIF - A Hierarchical Representation for Behavioral Synthesis of Analog Systems from VHDL-AMS", Technical Report, DDEL, University of Cincinnati, April 1998.

[10] A. Doboli, R. Vemuri, "A VHDL-AMS Compiler and Architecture Generator for Behavioral Synthesis of Analog Systems", *Proceedings of DATE'99*, pp.338-345, 1999.

[11] S. Donnay et al, "Using Top-Down CAD Tools for Mixed Analog/Digital ASICs: a Practical Design Case", *Analog Integrated Circuits and Signal Processing*, pp.101-117, 1996.

[12] S. Franco, "Design with Operational Amplifiers and Analog Integrated Circuits", McGraw Hill, 1988.

[13] M. Gen, R. Cheng, "Genetic Algorithms and Engineering Design", John Wiley & Sons, 1997.

[14] G. Gielen, H. Walscharts, W. Sansen, "Analog Circuit Design Optimization Based on Symbolic Simulation and Simulated Annealing", *IEEE Trans on Solid-State Circuits*, Vol.25, No.3, pp.707-713, June 1990.

[15] E. Horowitz, S. Sahni, "Fundamentals of Computer Algorithms", Computer Science Press, 1985.

[16] D. Leenaerts, "Application of Interval Analysis for Circuit Design", *IEEE Transactions of Circuits and Systems*, Vol.37, No.6, pp.803-807, June 1990.

[17] M.del Mar Hershenson,S.Boyd,T.Lee,"CMOS Operational Amplifier Design and Optimization via Geometric Programming",*Proc. 1st Int'l Workshop on Design of Mixed-Mode Integrated Circuits and Applications*, 1997.

[18] A. Nunez and R. Vemuri, "An Analog Performance Estimator for Improving the Effectiveness of CMOS Analog System Circuit Synthesis", *Proceedings of DATE'99*, pp.406-411, 1999.

[19] W. Nye, D. Riley, A. Sangiovanni-Vincentelli, A. Tits, "DELIGHT.SPICE: an optimization-based system for the design of integrated circuits", *IEEE Transaction on CAD*, vol.7, No.4, pp.501-519, April 1988.

[20] E. Ochotta, R. Rutenbar, R. Carley, "ASTRX/OBLX: Tools for Rapid Synthesis of High-Performance Analog Circuits", *Proc. of the 31st ACM/IEEE Design Automation Conference*, pp.24-30, 1994.