Hierarchical Constraint Transformation using Directed Interval Search for Analog System Synthesis [*]

Nagu R. Dhanwada, Adrian Nunez-Aldana and Ranga Vemuri
{*nagu,anunez,ranga*}*@ececs.uc.edu*
Laboratory for Digital Design Environments
Department of ECECS, University of Cincinnati
Cincinnati, OH 45221–0030

Abstract

In this paper, we present a hierarchical approach for constraint transformation. The important features of this are: a genetic algorithm (GA) based search engine that computes design parameter ranges, a hierarchically organized characterization mechanism based on the concept of directed intervals that assists the search engine and an analog performance estimator. Experiments were conducted comparing the hierarchical approach with a flat bottom-up one. The results obtained demonstrate the effectiveness of the former approach. Experimental results highlighting the impact of using the characterization information within the constraint transformation process are also presented.

1. Introduction and Motivation

Crucial to a top–down mixed–signal design process [12] is a mechanism to propagate the specifications and constraints on the design elements used at one level to those at the next level. This task of transforming the system-level specifications onto component level constraints is called *Constraint Transformation* [3]. Efficient automation of this task is one of the the most important steps in automating the design of analog and mixed analog-digital systems.

VASE is a mixed–signal synthesis system being developed at the University of Cincinnati [17]. Figure 1 shows the analog synthesis flow in VASE. The VHDL-AMS compiler and architecture generator [18] transforms the VHDL-AMS specification into net–lists of components. The constraint transformation and component selection step translates this system level net–list and system constraints onto component design/performance parameters while selecting a suitable implementation for the components from a library.

To select these components the synthesis algorithms need information about the performance of components in the library. The task of characterizing an analog

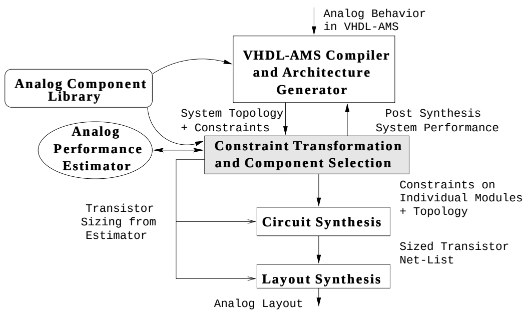

Figure 1. **Analog Synthesis in VASE**

component library is complex because of the number of interacting parameters involved. In the worst case, the characterization information for an analog component could be viewed as a very large table of values of design parameters versus the performance parameters. However, such a representation cannot be readily used in a meaningful way by an automatic synthesis tool. Thus from a synthesis point of view, we need a characterization mechanism that generates information that could be used effectively by the analog synthesis algorithms. Thus, a component characterization method should be an inherent part of a constraint transformation framework. Eckmuller et al [11] present a characterization technique based on numerical simulations for CMOS analog circuits. This characterization mechanism is geared more towards assisting the manual design of analog circuits, rather than automatic CAD tools. Feasibilty and performance evaluation of analog circuits using numerical macro–modeling techniques was proposed in [15].

There are two aspects to the constraint transformation problem: *Constraint Model Generation*, where a model relating the design/performance parameters is generated and *Constraint Allocation*, where actual values are assigned to these parameters. Although the latter problem has been addressed by many [7, 10], there have been very few methods that perform the entire process. Arsintescu et al [3] address the problem of AC constraint transformation for a class of linear

[*]This work is sponsored by USAF, Air Force Research Laboratory, WPAFB under contract number F33615-96-C-1911

continuous-time circuits using hierarchical parameter modeling and constrained optimization techniques. However in [3] the problem of topology selection during the process of constraint transformation is not considered as they start by assuming that the circuit is fully designed a priori. At the circuit synthesis level, Kruiskamp and Leenaerts [6] present an operational amplifier synthesis tool that does simultaneous topology selection and circuit sizing using genetic algorithms. Maulik et al present a method to handle simultaneous topology selection and circuit sizing using a mixed-integer non-linear programming method [7]. All these approaches attempt to find point values (not ranges) for the design parameters which is equivalent to computing a single point in the parameter space of the system that meets the constraints.

Leenaerts in [16] presents a circuit synthesis technique based on interval analysis. A numerical interval solution technique is used to partition the design constraints. This approach is limited by the solution method that can handle linear equations only. On the other hand our technique that is based on an exploration (search) and estimation oriented approach is not constrained by such issues.

In this paper we present an approach for performing constraint transformation and topology selection, starting from the system level. This is based on the formulation of constraint transformation as a search problem. The main features of this approach are a Genetic Algorithm (GA) based search engine, a component characterization method, and an analog performance estimator (APE). The APE estimates the performance of an analog circuit along with the anticipated sizes of the circuit elements, given the various design parameters [5]. Both component characterization and constraint transformation are based on a directed interval representation of the relationships between design and performance parameters. The search engine computes ranges for the design parameters that satisfy the system level constraints. Each set of design parameters constitute a sizing solution and the set of sizing solutions (corresponding to the design parameter ranges) generated by the APE aid in pruning the vast search space of an underlying circuit synthesis tool. This kind of a pruning would not be possible if the constraint transformation step computes point values instead of intervals for the design parameters.

Figure 2 shows the hierarchical and flat approaches to constraint transformation. In the latter approach the search method takes a flat view of the parameter space of all the components in the system. This causes the size of the search space to be huge which tends to affect the performance of the search engine. Therefore, in the hierarchical method we adopt a two-level approach where the top-level Interval Genetic Algorithm (IGA) acts as a search space partitioner. Each of these partitions generated by the search space partitioner could be searched independently by the lower level IGAs. Thus, such a hierarchical approach improves the overall search process by partitioning the otherwise huge search space into smaller segments. Each of these partitions correspond to the component level performance constraints that are generated by the top-level GA. Without any component characterization information, the search process would be oblivious to the interactions among the various design and performance parameters. Such knowledge about the relationships among the design and performance parameters would help guide the search engine. Besides, this information should be in a form that could be readily used by the search engine. All these necessitate the presence of a component characterization mechanism in a constraint transformation framework.

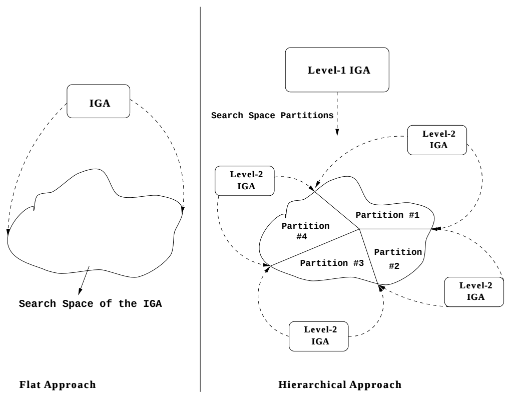

Figure 2. **Search Space Partitioning**

The characterization information is organized in two levels. At the lowest level are the component characterization tables that give the relationship between the component design and performance parameters. This table is generated once for each component. At the next higher level are the system level tables. The system level tables capture the relationships between the component performance parameters and the system performance. These tables have to be generated dynamically because the relationship between the component's performance parameters and system performance is dependent on it's interaction with the other components in the system net-list. This kind of a hierarchical organization of the characterization information helps in providing a link between the system performance and component design parameters.

The experimentation presented compares the hierarchical approach with the flat one. We also present experimental results showing the impact of using the characterization information in the search process. The rest

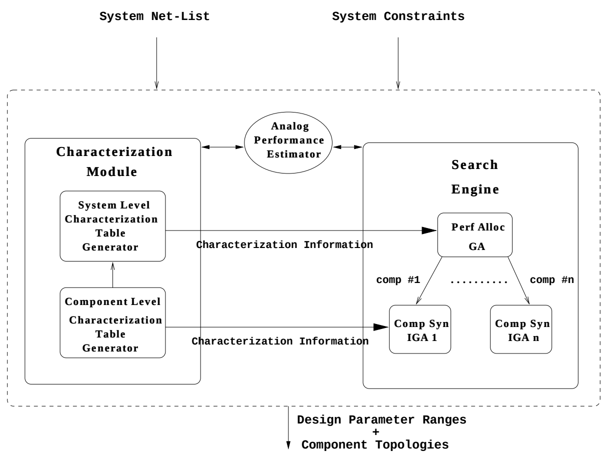
System Net-List
System Constraints
Characterization Module
Analog Performance Estimator
Search Engine
System Level Characterization Table Generator
Characterization Information
Perf Alloc GA
comp #1
comp #n
Component Level Characterization Table Generator
Characterization Information
Comp Syn IGA 1
Comp Syn IGA n
Design Parameter Ranges + Component Topologies

Figure 3. **The Overall Approach**

of the paper describes each of the constituents of our approach.

2. The Approach

Figure 3 shows the proposed constraint transformation approach. It consists of an analog performance estimator, a component characterization mechanism and a GA based search engine. The analog performance estimator is organized hierarchically and at the highest level is the user-application level that contains symbolic equations relating component performances to system performance for basic configurations like the cascaded, split and join. To estimate the performance of an arbitrary system level net-list, the components in the net-list are grouped into some combination of the basic configurations by the estimator.

The top-level *performance allocation IGA* in the search engine computes component performances that satisfy the system level constraints. The system level tables for each of the component in the given system net-list are used by this GA to guide it's search. For each component, the design parameter ranges satisfying the component performance constraints are computed by the lower level *component synthesis IGAs* executing in parallel. These IGAs use the component level characterization tables to guide their search. In the rest of the section we describe each of the constituents of the approach.

2.1 The Analog Performance Estimator

The analog performance estimator [4, 5] 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 symbolic performance equations of analog circuits at various levels of abstraction. APE uses 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. The estimation and sizing are performed bottom up starting from the transistor level through to the system level. The highest level is the user-application level, where 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. To estimate the performance of an arbitrary system level net–list, there is a wrapper above the user–application level that partitions the net–list into basic configurations and then uses the APE to estimate the performance. The wrapper reduces the given system–level net–list to one of the basic configurations that could be estimated using the user–application level in the APE. This is done by formulating it as a hierarchical network covering problem. The idea is to start from the system level net–list and apply recursively a network covering algorithm to arrive at a final basic configuration. The APE also has built into it some rules that detect the cases where transistors in the design go into saturation, or when some basic conditions governing the functionality of the circuit have been violated. For more details about the APE, we refer the reader to [5].

2.2 Characterization Method

The characterization method consists of component level and system level table generators. Both of the tables are based on the concept of *directed intervals* [4]. i.e each entry in the table has a direction as well as a pair of interval values associated with it. The interval values give the range within which the direction of change holds good. It is not enough if the direction alone is captured because the direction for each of the performance parameter need not be monotonic for the entire range of design parameter/component performance parameter values. This kind of characterization information is targeted towards assisting a move based search engine such as the one presented in the current approach.

2.2.1 Component Level Characterization

Component level characterization is done only once for each component to generate the component level tables, whose rows are indexed by the design parameters and whose columns are the performance parameters. The direction in which the performance parameter (column) changes with increase in the design parameter (row) value is stored in every cell. The direction may be $\rightarrow$ (increase), $\leftarrow$ (decrease), or a - (constant). Also stored is the interval value for the performance parameter and an interval value for the design parameter. Such a characterization process gives us the idea of how the design parameters should be varied to achieve a particular change in the performance. The details of component level characterization table generation can be found in [4].

```
procedure SysTableGen(Comp C, SysNetList sysnet)
 begin
    for all other comp's $\in$ sysnet
        set perf params to their MIN values
    end for
    Instantiate a topology for component C
    SysTab1 $\leftarrow$ GenSysTable(C,SysTab1)
    for all other comp's $\in$ sysnet
        set perf params to their MAX values
    end for
    SysTab2 $\leftarrow$ GenSysTable(C,SysTab2)
    merge(SysTab1, SysTab2)
end
```

Figure 4. **Overall System Table Generation**

2.2.2 System Level Characterization

The system level table for each component is generated dynamically. The rows of this table represent component performances and columns represent system performance. 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 characterization method can be seen as one of sampling the search space (component performance space) at different points to generate information about the space that could be used by the search engine. The number of points being sampled depends on the width of the component performance intervals. The characterization method takes only a small amount of time and does not involve moving through the entire search space.

The overall system level table generation procedure is shown in Figure 4. This routine is invoked for each component in the system net-list. In this procedure, first a random topology is instantiated for the component. This is based on the observation that the topology of a particular component does not affect the relationship between the component performance parameters and the system performance. The performance parameters of all the other components are set to their minimum values and the system table generation procedure *GenSysTable* is invoked to generate SysTab1. In a similar way SysTab2 is generated by setting all the other component performance values to their maximum. Finally these two tables are merged together to form the overall system level characterization table. Merging involves performing a union of cells in each of the tables. If the directions in both the cells are the same then a union of the corresponding system performance intervals is done. If the directions happen to be different the cell in the merged table would have a set of directed intervals. Each set corresponds to a unique direction value. In this case each cell in the merged characterization table captures the relationship between the component performances and system performance in different disjoint regions of the system performance space.

In the *GenSysTable* procedure, the performance parameters of the component are obtained first. Now, for each of these performance parameters, the set of design parameters that affect them are obtained by looking up the component level characterization table. These design parameters are moved within their interval value (obtained from the component level table) to increase the current component performance parameter value. The effect of increasing this component performance parameter on system performance is observed by calling the system performance estimator. The idea behind moving the design parameter values to increase the component performances in the system level table helps maintain the link between the system performance and the possible design parameter values of the component. The system performance values returned by the system performance estimator are checked for monotonicity. As long as the system performance parameter values remain monotonic, the component performance and system performance intervals keep widening. Once there is a discontinuity in the values of the performance parameters, the interval is closed and an entry is made in the table. The direction part is set based on whether the system performance value was increasing or decreasing with increasing component performance values.

2.3 Search Engine

The constraint transformation search engine computes component design parameter ranges that satisfy the system level constraints. It uses genetic algorithms, which are stochastic search techniques based on the mechanism of natural selection and genetics [1]. The reason for the success of the GAs is their ability to exploit the information about an initially unknown search space in order to bias subsequent searches into useful subspaces [2]. This adaptation feature of the GA is the key to it's success, particularly in large, complex and poorly understood search spaces, where classical search techniques are inappropriate. Thus, the *GA* is well suited to perform the task of constraint transformation that may be seen as the task of searching the component parameter space for a solution that falls within the constraint satisfying region of the system's performance space.

The search engine is organized in two levels. At the highest level is the performance allocation GA, that accepts the system net-list, system constraints, and the system characterization table and computes component performance values that satisfy the system constraints. The component synthesis GAs are executed in parallel for each of the components in the system net-list. The component synthesis GA computes component design parameter ranges that satisfy the performance constraints generated by the performance allocation GA.

2.3.1 Performance Allocation IGA

The first step in the GA is to suitably encode the solution representation. After initialization, the steps of selection, crossover, mutation and replacement are repeated 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 and mutation operators perturb the selected solutions to create new ones. Finally, the newly generated solutions are merged into the set of solutions, by replacing the two worst solutions with these new ones. This is done by the replacement method. Once the convergence condition has been reached the solution with the best cost function value represents a solution to constraint transformation problem. Typical convergence conditions would include number of iterations, or the cost function value for the best solution remaining the same over a number of iterations.

The solution in the performance allocation GA is represented as a set of interval values. Each interval represents the component performance parameter values. The performance estimator for the given net-list (which is generated by the APE) is used to estimate the system performance. The GA assigns values to these component performance values that satisfy the user imposed system level constraints.

Cost Function Evaluation: The cost function evaluation technique is used to evaluate the solution quality. The result of this is used by the GA in the selection process. Each solution represents a set of points, forming a region in the system performance space. If all the points in the solution are constraint satisfying, then the entire solution is a constraint satisfying one having a cost function value of zero. The evaluation of the solution quality is done using a two-level cost function.

The Cost Function : The IGA uses a two-level cost function which consists of a local part and a global one, which are shown below.

· *Local Cost function*

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

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

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

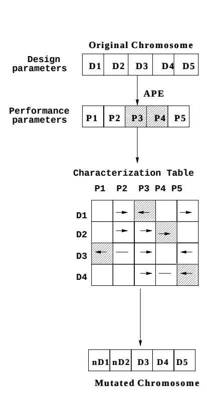

Figure 5. **Mutation**

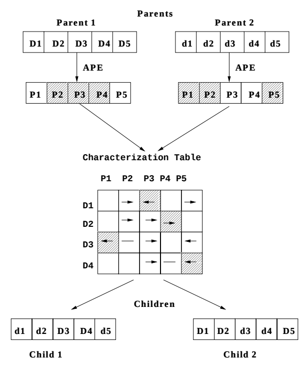

Figure 6. **Crossover**

P_{i_est} is the value for the system performance parameter in the current solution, and $P_{i\_constraint}$ is the user specified constraint. Such a cost function is typical of GAs that handle multiple constraints. This cost function gives the average factor by which each performance parameters deviates from the constraints. The GA works towards minimizing the cost function.

· *Global Cost function* which is defined as follows:

$$\frac{1}{N} * \sum_{i=1}^{N} local\_obj\_fn(i)$$

where N, denotes the number of points in the region. This global cost function returns a value of zero only if all points in the region represented by the solution are constraint satisfying.

Directed Interval based Operators:

During the early part of a search, we would like to have many individuals explore different parts of the search space. But once a promising region has been discovered, we would like to focus the remaining search in that area. The directed interval based operators defined here to some extent act as local optimization methods and help in focusing the search process. Therefore in our GA we start out with the traditional operators of non-uniform mutation and uniform crossover [1] and after some evolution we switch to the directed interval based operators.

Mutation: Figure 5 shows the directed interval based mutation operator. Initially, the solution to be mutated is evaluated using the driver program for the current system net-list. The resulting performance parameters are compared with the user defined constraints to identify those performance constraints that are being violated in the current solution. These are shown in the figure as the shaded parts in the performance parameter array. Here, the constraints P3 and P4 are being violated. Now the characterization table information is used to select which component performance

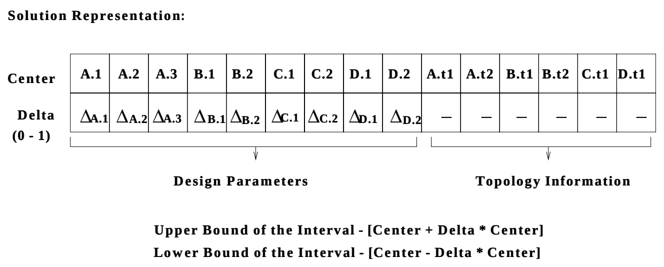

Figure 7. **Solution Representation for IGA**

parameters are to be changed and in what direction. The system–level characterization tables for all the components are seen globally to find all the component performance parameters that need to be changed to improve the system performance parameter that is being violated in the current solution. We see that parameters D1 and D2 are the ones to be changed in the new solution. The mutated solution has the values D1 and D2 replaced by nD1 and nD2. Thus such a mutation operator attempts to intelligently guide the evolution of the GA towards constraint satisfying areas in the search space through the use of characterization information.

Crossover: Figure 6 shows the directed interval based crossover operator. Similar to the mutation operator, the two parents are individually evaluated using the driver program, and the constraints that are violated are identified on comparison with the user defined constraints. In the figure, in parent 1 the only constraints that are being satisfied are P1 and P5. Similarly in parent 2, constraints P3 and P4 are the only constraints being satisfied. The characterization table is looked up and the component performance parameters that correspond to the satisfied performance constraints are combined together to form one child. The remaining parameters are combined together to form the second child. Therefore, child 1 is formed by combining the design parameters d1, d2 from parent 2 and D3 and D4 from parent 1. Child 2 is formed by combining the remaining design parameters.

2.3.2 Component Synthesis IGA

Figure 7 shows the solution representation for the Interval Genetic Algorithm. 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. Each component may have more than one entry in the topology part of the array if that component has sub-components having different topologies. The representation is a two dimensional array of real numbers. The first row represents the center of the interval, and the second row is a delta value, which lies between 0 and 1. The upper and lower bound of the interval are calculated as shown in Figure 7. These are calculated for every design parameter in the solution. A set of point values for the design parameters constitute a point in the region defined by the design parameter intervals. Thus, moving from one point to an other in this region of points requires us to define a step size, with which the design parameters would be incremented from their lower bound to the upper bound. These step sizes are calculated for each design parameter in the solution, based on the width of the interval. The smaller the step size, more would be the number of points in the region represented by the design parameter intervals. The step size in the IGA is set to about five percent of the width of the interval.

Cost Function: The cost function is similar to the one used in the performance allocation IGA, but for component performance parameters being used instead of system performance parameters.

Genetic Operators: A mix of the traditional non-uniform mutation and uniform crossover operators were used in conjunction with directed interval based operators. The directed interval based operators for the component synthesis GA are similar to the ones defined for the performance allocation GA. The only difference being the usage of the component characterization tables instead of the system characterization tables.

3. Experimentation

The first example (cascaded amplifiers) is a system of four cascaded amplifiers. The second example(Acquisition System) is an acquisition system front-end that consists of a series of seven amplifiers connected to an analog-digital converter. The next example (Neural Control-1) comprises of two banks of amplifiers connected in a join configuration with a comparator. The fourth example (Neural Control-2) is a system of six blocks, which consists of an amplifier driving two banks of cascaded amplifier pairs which are connected in a join configuration with a comparator. These examples are typical of control circuits used for the activation of neurons in artificial neural networks [14]. The system in the final example (Neural Trainer) consists of ten blocks. Inputs from two sensors are amplified by cascaded amplifiers and are compared. The output of the comparator drives two cascaded pairs of amplifiers whose outputs are summed using a summing amplifier. This kind of a system is used to implement weighing functions that are used in the training of artificial neural networks.

The constraint transformation algorithm was implemented using a publicly available GA package, GAlib [13]. All the experiments used mutation and crossover rates of 0.1 and 0.9 with a population size of 1000. We

ckt name	Constraint set 1	Constraint set 2

cascaded amplifiers	area=1200$sq\mu$, power=40mW, gain=2e4
	bw=30KHz, sr=100000
		area=800$sq\mu$, power=40mW, gain=1e4
		bw=100KHz, sr=70000

Acquisition System	area=1800$sq\mu$, power=80mW, gain=10e3
	bw=100KHz, sr=70000
		area=1800$sq\mu$, power=90mW, gain=1e3
		bw=800KHz, sr=80000

Neural Control-1	area=900$sq\mu$, power=90mW, gain=1e3
	bw=100KHz, sr=80000
		area=1200$sq\mu$, power=90mW, gain=2e3
		bw=90KHz, sr=80000

Neural Control-2	area=1500$sq\mu$,power=80mW, gain=2e3
	bw=300KHz, sr=70000
		area=1200$sq\mu$, power=80mW, gain=2e3
		bw=90KHz, sr=80000

Neural Trainer	area=9000$sq\mu$, power=90mW, gain=1000
	bw=20KHz, sr=70000
		area=1800$sq\mu$, power=12mW, gain=500
		bw=200KHz, sr=70000

Table 1. **Example Circuits and Constraint Sets**

Circuit	Flat		Hierarchical		Flat		Hierarchical		savings time(ser)	savings time(par)	avg imp obj func
	set 1	set 2	set 1	set 2	set 3	set 4	set 3	set 4			
cascaded amplifiers	0 5083s	0 6219s	0.1 2180s [6013s]	0 2386s [6114s]	0.2 5567s	0.1 6434s	0 2273s [5464s]	0 2178s [5633s]	5.5%	61.5%	0.2
Acquisition System	0.34 21414s	0.24 19033s	0 2344s [14620s]	0 2831s [13990s]	0.2 20414s	0.41 16370s	0 2740s [11332s]	0.4 2463s [12311s]	31.9%	86.5%	0.26
Neural Control-1	0 26283s	0.08 24443	0 2678s [13844s]	0 2525s [13963s]	0.3 23528s	0.2 19847s	0.2 2684s [13587s]	0.17 2933s [14088s]	40.4%	88.3%	0.06
Neural Control-2	1.6 16869s	1.3 14738s	0 2854s [10632s]	0 3427s [10344s]	0.52 18180s	0.51 14849s	0 2781ss [10063s]	0.2 2665s [9690s]	36.5%	81.7%	1.24
Neural Trainer	0.65 20860s	0.59 19400s	0 2980s [14160s]	0 3112s [13980s]	0.4 24695s	0.37 16025s	0 2750 [14268s]	0.09 2960 [14033s]	28.7%	85.08%	0.41

Table 2. **Flat Vs Hierarchical approaches**

Circuit	GA without charac		GA with charac		GA without charac		GA with charac		savings in time(ser)	savings in time(par)
	set 1	set 2	set 1	set 2	set 3	set 4	set 3	set 4		
cascaded amplifiers	0.3 4268s [10806s]	0.2 4441s [9154s]	0.1 2180s [6013s]	0 2218s [5991s]	0.2 3218s [9430s]	0.8 3597s [9654s]	0 2273s [5464s]	0 2178s [5633s]	40.6%	41.9%
Acquisition System	0.1 4594s [20978s]	0 5188s [21635s]	0 2344s [14620s]	0 2831s [13990s]	0.7 6554s [19637s]	0.8 5277s [18841s]	0 2740s [11332s]	0.4 2463s [12311s]	35.7%	51.5%
Neural Control-1	0 3561s [19483s]	0 3463s [19981s]	0 2548s [10844s]	0 2525s [10963s]	0.27 4107s [19793s]	0.1 4111s [18113s]	0.2 2684s [13587s]	0.17 2933s [14088s]	37.4%	29.68%
Neural Control-2	0 4300s [14081s]	0.1 2883s [14552s]	0 2854s [10632s]	0 3427s [10344s]	0.1 3731s [16311s]	0.2 3648s [15449s]	0 2781s [10063s]	0.2 2665s [9690s]	32.3%	24.4%
Neural Trainer	0 3641s [20741s]	0 3244s [23695s]	0 2980s [14160s]	0 3112s [13980s]	0.15 3189s [24729s]	0.2 3263s [19808s]	0 2750s [14268s]	0.09 2690s [14033s]	36%	13.4%

Table 3. **GA without characterization Vs with characterization info**

Cascaded Amps	Component 1	Component 2

perf alloc GA constraints	area=200$sq\mu$, power=8mW, gain=88
	bw=190KHz, sr=100000
		area=300$sq\mu$, power=12mW, gain=122
		bw=100KHz, sr=70000

Design Parameter Ranges	Ibias=[2.6,2.9]e-07, Adm=[2157,3457]
	gainw_l=[45,149]
	zout=[101,107], DCgain=[80,93]
		Ibias=[1.3,1.7]e-06, Adm=[10157,10415]
		gainw_l=[120,166]
		zout=[3,67], DCgain=[119,126]

perf after comp syn	area=[166,190]$sq\mu$, power=[3,7]mW, gain=[90,133]
	bw=[640,1180]KHz, sr=[86033,107993]
		area=[158,186]$sq\mu$, power=[5,8]mW, gain=[78,156]
		bw=[90,312]KHz, sr=[22245,287000]

Table 4. **Constraints and design parameter ranges for cascaded amplifier example**

compare the flat approach to constraint transformation with the hierarchical one presented here. Table 1 shows two of the constraint sets that were used. Table 2 compares the hierarchical approach with the flat one for four constraint sets. Each entry for the flat approach has the objective score and the time taken. Each entry for the hierarchical approach has the objective score, time taken when the component synthesis IGAs are run in parallel (total time = perf alloc GA time + max(comp syn IGA times)), and the last row shows the time taken when the IGAs are run serially. The objective score value shown for the hierarchical approach is the sum of the objective scores of the component synthesis IGAs, and the performance allocation GA. The final columns show the average savings in time and the average improvement in the objective function value obtained using the hierarchical approach. The results in the table clearly show that the hierarchical approach better than the flat one in terms of both the search quality and the time taken. The time taken for system level table generation is very small (less than 1.5 - 2 mins) and is included in the hierarchical method's time.

The hierarchical approach uses both the system level and the component level characterization tables. The ratio between the number of generations the traditional operators was used and the number of generations for which the directed interval based operators were used was 40-60. The improvement obtained using the hierarchical approach is clear from the table. Next, we observed the impact of using the characterization tables in the constraint transformation process. For this we compared the hierarchical approach using only the traditional non-uniform mutation and uniform crossover operators [2], to one that uses the traditional operators in conjunction with the directed interval based operators. The results shown in Table 3 clearly indicate the importance of the characterization tables in the constraint transformation process. UltraSparc 2s and 30s running solaris were used to run the examples, and the time was measured using the system time command. Table 4 shows the design parameter ranges generated by the constraint transformation process for two of the components in the cascaded amplifier example. The first row shows the constraints generated by the performance allocation GA, the second row shows the design parameter ranges generated by the component synthesis GAs, and the final row is the performance after component synthesis. The sizing solutions corresponding to the generated design parameter ranges could be used to prune the search space of an underlying circuit synthesis tool [19].

4. Conclusion

A hierarchical approach to constraint transformation was presented. The main constituents of this include a hierarchically organized search engine, a component characterization method and an analog performance estimator. Some of the salient features of the approach include the dynamic generation of system level characterization tables, computing intervals for the component design parameters and using the directed interval based characterization method to improve the search process. Experimental results comparing the hierarchical approach with a flat one clearly establish the superiority of the former approach. Also, the impact of using the characterization information within the constraint transformation process was observed.

References

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

[2] F. Herrera, M.Lozano, J.L. Verdegay, "Tackling Real-Coded Genetic Algorithms : Operators and Tools for Behavioural Analysis", Tech Report DECSAI-95107, Feb'95.

[3] B.G. Arsintescu et al, "General AC Constraint Transformation for Analog ICs", Proc of the 35th DAC,1998.

[4] N.R. Dhanwada, A. Nunez, R. Vemuri, "Component Characterization and Constraint Transformation based on Directed Intervals for Analog Synthesis" ,Proc Intl Conf on VLSI Design'99.

[5] A. Nunez and R. Vemuri, "An Analog Performance Estimator for Improving the Effectiveness of CMOS Analog Circuit Synthesis", Proceedings of DATE 1999.

[6] W. Kruiskamp and D. Leenaerts," DARWIN : Analogue Circuit Synthesis based on Genetic Algorithms", International Journal of Circuit Theory and Applications, Vol 23, 1995.

[7] P.C. Maulik, L.R. Carley, and R.A. Rutenbar, "A Mixed-Integer Non-linear programming approach to analog circuit synthesis", Proc 29th DAC, 1992.

[8] R.Harjani, R.A. Rutenbar, and L.R.Carley, "OASYS : A Framework for Analog Circuit Synthesis",IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 8, no. 12, Dec 1989.

[9] E.S.Ochotta, "Synthesis of High Performance Analog Cells in ASTRX/OBLX", Ph.D Dissertation, Feb 1994, CMU.

[10] N.R. Dhanwada and R. Vemuri, " Constraint Allocation in Analog System Synthesis", Intl Conf on VLSI Design'98.

[11] J.Eckmuller, M.Gropl, and H. Grab, "Hierarchical Characterization of Analog Integrated CMOS Circuits", Proc of DATE'98.

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

[13] M. Wall, "GAlib: A C++ Library of Genetic Algorithm Components".

[14] C. Mead, "*Analog VLSI and Neural Systems*", Addison Wesley, 1989.

[15] R.Harjani and J. Shao, "Feasibility and Performance Region Modeling of Analog and Digital Circuits",Analog Integrated Circuits and Signal Processing, Kluwer Academic Publishers, 1996.

[16] D.M.W. Leenaerts, "Application of Interval Analysis to Circuit Design", IEEE Trans. on Circuits and Systems, June 1990.

[17] R. Vemuri, N. Dhanwada, A. Nunez-Aldana and P. Campisi,"VASE - VHDL-AMS Synthesis Environment: Tools for Synthesis of Mixed-Signal Systems" *Proceedings EETimes Conference on Analog and Mixed-Signal Applications,* July 1997.

[18] A. Doboli and R. Vemuri, "A VHDL-AMS Compiler and Architecture Generator for Behavioral Synthesis of Analog Systems", Proc. DATE 1999.

[19] N.R. Dhanwada, A. Nunez-Aldana and R. Vemuri,"Automatic Constraint Transformation with Integrated Parameter Space Exploration in Analog System Synthesis" Proc. ASP-DAC 1999.