ASF: A Practical Simulation-Based Methodology for the Synthesis of Custom Analog Circuits

Michael J. Krasnicki[1], Rodney Phelps[2], James R. Hellums[1], Mark McClung[1],
Rob A. Rutenbar[2], and L. Richard Carley[2]
[1]Texas Instruments Incorporated, Dallas, TX 75243
[2]Carnegie Mellon University, Pittsburgh, PA 15213

Abstract: *This paper describes ASF, a novel cell-level analog synthesis framework that can size and bias a given circuit topology subject to a set of performance objectives and a manufacturing process. To manage complexity and time-to-market, SoC designs require a high level of automation and reuse. Digital methodologies are inapplicable to analog IP, which relies on tight control of low-level device and circuit properties that vary widely across manufacturing processes. This analog synthesis solution automates these tedious, technology specific aspects of analog design. Unlike previously proposed approaches, ASF extends the prevalent "schematic and SPICE" methodology used to design analog and mixed-signal circuits. ASF is topology and technology independent and can be easily integrated into a commercial schematic capture design environment. Furthermore, ASF employs a novel numerical optimization formulation that incorporates classical downhill techniques into stochastic search. ASF consistently produces results comparable to expert manual design with 10x fewer candidate solution evaluations than previously published approaches that rely on traditional stochastic optimization methods.*

I. Introduction

Cost pressures are forcing the creation of highly integrated System-on-Chip (SoC) solutions for mass-market products. To manage complexity and time-to-market, SoC designs require a high level of reuse. Cell-based techniques lend themselves well to a variety of strategies for capture and reuse of digital intellectual property (IP). But these digital IP strategies are inapplicable to analog IP, which relies on tight control of low-level device and circuit properties that vary widely across fabrication technologies. Thus, without an automated synthesis methodology tailored specifically to the analog problem, the analog portion has to be manually redesigned for each application and fabrication technology. So, a new design methodology is needed that replaces these tedious, technology-specific aspects of analog design with an automated synthesis solution. This will allow designers to focus on more interesting circuit and system level design problems. And, it will allow companies to create portfolios of reusable, retargetable, analog intellectual property than can be deployed in time-to-market critical products. This document describes an essential cornerstone of this next new mixed-signal design methodology.

Although many analog synthesis frameworks have been proposed in the past, none have proven practical and effective enough to be accepted as an integral part of the industrial design process. ASF extends the prevalent "schematic and SPICE" design methodology. It utilizes the simulation methods used by designers to validate manual circuit designs during the synthesis process. In addition, it incorporates modular, reusable, user-configurable test benches called evaluators. *Evaluators* are both topology and technology independent. They allow us to create a programming-free synthesis methodology that can be integrated into a commercial schematic capture environment. Our numerical optimization formulation is robust and consistent. It can consistently synthesize a 20 variable op-amp in less than 5,000 candidate solution evaluations, producing results comparable to expert manual design. This formulation is up to 10x faster than previously published approaches that rely on traditional stochastic optimization methods. This approach reduces the number of state evaluations while, at the same time, improving solution quality and optimizer consistency.

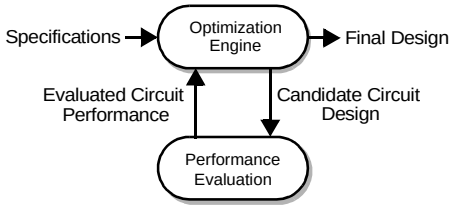


Fig. 1 Abstract model of analog cell synthesis

II. Review Of Prior Approaches

A significant amount of research has been devoted to cell-level analog synthesis. Fig. 1 shows the basic architecture of most analog synthesis tools. In this model, the optimization engine visits candidate circuit designs and adjusts their parameters in an attempt to satisfy the specified performance goals. An evaluation engine determines the quality of each circuit candidate. In this survey, we broadly categorize previous work based upon the evaluation strategy. Specifically, we subdivide previous approaches into four categories:

- *Equation-Based.* Early approaches utilized explicit scripts of equations to directly evaluate the quality of each proposed candidate circuit design. This formulation offers fast performance evaluation which is conducive to aggressive search over the entire candidate solution space. A number of optimization strategies have been attempted with equation-based techniques, including numerical search [10][2], combinatorial search [14], hierarchical systems that attempt to decompose the evaluation and optimization [5], qualitative and fuzzy reasoning techniques [25], and geometric programming [6]. However, creating a model that captures the behavior of a circuit topology as a set of compact, closed form, analytical equations for performance evaluation is prohibitively time consuming, indeed, often more time consuming than manually designing the circuit. Also, the simplifications required in these closed-form analytical models necessarily limit their accuracy and completeness.
- *Symbolic Analysis.* Since the creation of performance models is prohibitively time consuming, symbolic analysis techniques have been developed to mechanically derive these equations [4]. However, these techniques are still mostly limited to linear performance specifications.
- *Simulation-Lite.* While symbolic analysis has successfully automated select aspects of the equation generation process, it does not provide a complete solution that can rival the generality, flexibility, and accuracy of circuit simulation. The first crop of simulation based tools utilized custom lightweight simulators or hybrid methods based on simulation and equation-based modeling [19][15].
- *Full SPICE.* Full SPICE-based evaluation offers superior accuracy, generality, and easy of use. Simulation, however, is computationally expensive. There are two ways to cope with this drawback: less search or distributed/parallel evaluation. Gradient and sensitivity techniques can be used to rapidly find a local minimum in the neighborhood of a starting point [18]. However, finding an optimal

solution, without a known good starting point, is a much more difficult numerical problem which requires a computationally intensive global search strategy to avoid getting trapped in a poor local minimum. For this reason, [11][20] distribute this computational burden across a network of workstations.

III. GOALS

From the literature survey, we believe that the following attributes are necessary for a circuit synthesis solution to gain acceptance:

- *Accuracy*. Even though a significant amount of research has been devoted to cell-level analog synthesis, previous approaches have not seen widespread adoption as an integral part of industrial design process. This is primarily due to the prohibitive effort needed to reconcile the simplistic circuit models employed during synthesis with the "industrial-strength" models used for validation in a production environment. The synthesis formulation must leverage the existing investment in simulators, device models, process characterization, and "cell sign-off" validation methodologies.
- *Ease of Use.* Preparing a synthesis task should require no more work than specifying the topology, reasonable numerical search ranges for each of the independent variables, and the desired performance specifications. No methodology that attempts to turn a circuit designer into a programmer will gain acceptance.
- *Generality.* No methodology that limits the user to a set of topologies, no matter how arbitrarily large, will gain wide acceptance. A successful solution must allow the user to synthesize any arbitrary topology for any desired manufacturing process. It also must provide the user with a open framework for measuring arbitrary circuit characteristics with minimal effort.
- *Robustness.* The optimization/search heuristic should be able to produce good solutions consistently. A tool that requires an unpredictable number of synthesis runs or a tool that requires the user to tinker with numerous obscure "tuning" parameters will not gain acceptance.
- *Reasonable Run-Time.* Synthesis should be accomplished in a reasonable amount of time. While one could argue about the definition of "reasonable run-time", an analog designer's time is clearly more valuable than CPU compute time. Thus, any trade-off that saves designer time in favor of compute time is clearly worthwhile.

The next section describe a new synthesis strategy that addresses all of these concerns.

IV. SYNTHESIS FORMULATION OVERVIEW

Our circuit synthesis strategy relies on five key ideas, which have been implemented in ASF (Analog Synthesis Framework).

A. Simulator Encapsulation for Simulation-Based Evaluation

Simulator encapsulation creates a layer of abstraction that allows ASF to utilize a wide variety of different circuit simulators. Although different SPICE-class simulation engines share core mechanisms and offer similar input/output formats, they remain highly idiosyncratic in many features. Simulator encapsulation is a layer of insulating code that hides the idiosyncrasies and peculiarities of a simulator, rendering its behavior more generic. Encapsulation allows us to treat a given simulator as an abstract evaluation method. It turns the simulator into an object with formalized methods to invoke a simulation, to change circuit parameters, to retrieve simulation results as a simple vector of numbers, etc. The encapsulation hides varying data formats from the rest of the synthesis formulation.

After the test circuit has been built from information in an *evaluator* package (see Subsection IV.B. and Fig. 2), the encapsulation wrapper starts the simulator and opens an interactive data link. The simulator loads and simulates the instantiated evaluator to measure one or more performance characteristics of the candidate solution. After the interactive data link to the simulator is established, many candidate solutions can be evaluated in sequence. Specifically, the encapsulation library sends the candidate solution through the interactive

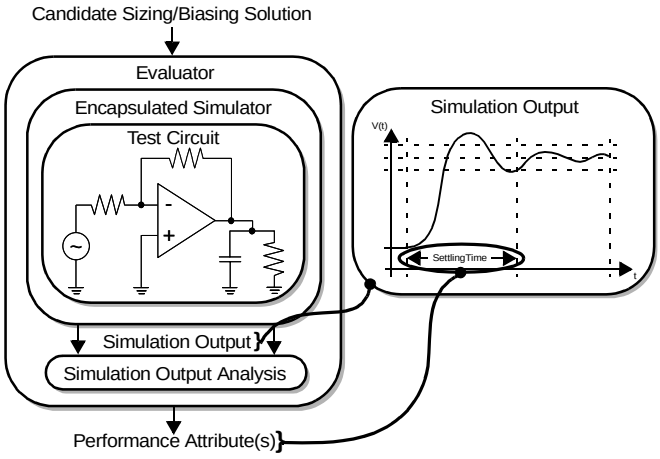


Fig. 2 Evaluator Overview

data link to the simulator. After all modifications associated with each candidate solution are made, the simulator executes the set of simulations associated with the evaluator to assess the performance of the candidate solution.

Full simulation provides a level of accuracy impossible to obtain from first order equations. This formulation allows us to leverage the existing investment in simulators, device models, process characterization, and "cell sign-off" methodologies that already exist as part of the current design environment. In addition, users do not have to provide any topology specific performance modelling equations, making this approach considerably easier to use than previously proposed approaches. Also, this approach affords us a level of generality that is impossible to obtain with equation based approaches. Essentially, one can synthesize any circuit that can be efficiently characterized with the use of simulation.

B. Evaluator: A Modular, Reusable, User-Configurable Test Circuit

The second key idea is the *evaluator*, which consists of a test circuit and an analysis script. An evaluator allows the user to measure circuit performance characteristics via simulation. The test circuit defines the bias and/or feedback network and stimulus for the simulation. As illustrated in Fig. 2, a candidate solution is simulated, producing simulation output that is analyzed with the use of the script. The purpose of the script is to reduce a potentially large amount of simulation data to a small set of numbers that capture one or more of the performance characteristics of the circuit. Creating evaluators is usually a small effort compared to writing an intricate set of performance equations.

Section V describes evaluators and their reuse strategy in greater detail. We demonstrate that it is possible to create a set of modular and reusable evaluators that can, with appropriate configuration, be utilized to characterize a whole family of analog blocks. This allows the maintainer of the synthesis framework to create a set of evaluator libraries that cover a broad range of analog blocks, thereby establishing an easy to use general purpose methodology for synthesizing any arbitrary topology. At the same time, it enables the measurement of relevant circuit properties with minimal effort. Assuming all evaluators are instantiated from a library, this methodology can be integrated with an existing commercial schematic capture environment. A dialog driven user interface can instantiate and configure each desired evaluator. This is a natural extension of the schematic annotations already created by designers on a daily basis and does not require any programming background.

C. Stochastic Search

As was shown in [11], circuit synthesis tasks have a highly non-linear cost surface with many local minima, jagged obstacles, and gently sloping plateaus. Stochastic style algorithms are a good choice for these types of landscapes because of their hill-climbing abilities. However, stochastic search algorithms, like simulated annealing [9],

have a reputation for slow execution due to the large number of solution candidates that must be visited to ensure consistently good results. In this particular application, this problem is greatly exacerbated by the use of simulation to fully characterize each candidate solution. We use three avenues of attack to mitigate the excessive run-time problem:

- *Parallel Stochastic Search.* This type of search evaluates multiple chains (sequences) of points in the cost landscape in parallel, providing some manner of synchronization that guarantees convergence to a final set of circuits of similar quality. Subsection IV.D. introduces parallel stochastic search as a component of our optimization methodology.
- *Less Search.* To further reduce the run-time, one has to look beyond traditional stochastic search. Subsection IV.E. will introduce Downhill Set MegaMoves; they embody a novel strategy that combines classical downhill techniques with stochastic search.
- *Parallel Circuit Evaluation.* Each candidate solution usually has to be characterized with multiple evaluators. These tasks can be distributed across a network of workstations. Subsection IV.F. provides an overview of our synthesis distribution strategy.

D. Parallel Stochastic Search

The fourth cornerstone of our strategy is parallel stochastic search. Specifically, our strategy is based on parallel recombinative simulated annealing (PRSA) [13]. PRSA is an algorithmic method that permits parallel annealing-type stochastic optimization. Conceptually, PRSA can be thought of as transforming a single annealing task into a population of parallel annealing tasks. In other words, n parallel computing nodes each anneal the same problem. The duration of each annealing run is divided by the degree of parallelism n. For example, suppose a serial annealer required 100,000 candidate solution evaluations, or moves in annealing nomenclature, to converge to an acceptable answer consistently. Further, suppose this serial annealer will be transformed into a population of 10 annealers. To maintain a constant move count, each annealer in the population can do at most 100,000/10 = 10,000 moves. Obviously, if each annealer in the population were to be run independently, none would converge to an acceptable result consistently. For this reason, PRSA introduces a synchronization mechanism that allows portions or entire solutions states to migrate between PRSA nodes.

Each annealer randomly communicates its current candidate circuit solution to a subset of the other PRSA nodes. Given these migrant candidate solutions from other PRSA nodes, each annealing task may choose to do one of two kinds of moves, where $\underline{x} = [x_1, x_2, \ldots, x_m]$ is the vector of m independent variables:

- A classical perturbation that alters the last visited solution by randomly altering some of its variables, *i.e.* $x^{new} = x^{old} + \Delta x^{old}$.
- A crossover operation that combines the current solution $\underline{x}^{old}$ with components of a migrant solution from another node, i.e. $\underline{x}^{new} = combine(\underline{x}^{old}, \underline{x}^{migrant})$. This operation is done in the style of classical mating-style crossover from genetic algorithms [7].

[13] provides a proof of finite-time convergence for PRSA. In addition, [13] demonstrates the feasibility of this approach on some classical optimization problems.

E. DeviationTracker and The Downhill Set

To further reduce run-time, we look beyond traditional stochastic methods to an approach that incorporates classical gradient free downhill methods. The Downhill Set is a library of modular downhill optimizers. It consists of DownhillGreedy, a random walk component optimizer, DownhillCS, a coordinate search component optimizer, DownhillSimplex, a Nelder-Mead Simplex component optimizer [17], and DownhillPowell, a Powell's method component optimizer [22]. These optimizers are invoked explicitly as part of a novel annealing cooling schedule called DeviationTracker. Each such invocation, referred to as a MegaMove, searches out a local minimum in the neighborhood of the current annealer state. After the MegaMove completes, DeviationTracker ensures that the annealer remains in the neighborhood of the local minimum while ensuring that the annealer does not remain trapped in that local minimum. The size of the neighborhood is reduced over the course of the annealing run, eventually preventing the annealer from leaving its final solution. Section VI describes the Downhill Set and the DeviationTracker cooling schedule in greater detail. Section VII will show that this approach reduces the number of required state evaluations while, at the same time, improving solution quality and optimizer consistency.

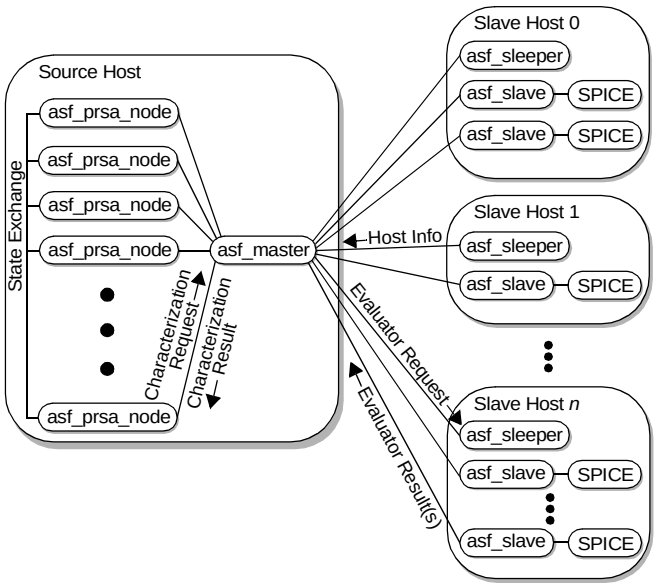


Fig. 3 Synthesis Distribution Strategy

F. Synthesis with "Idle" Cycles across a Network of Workstations

The final cornerstone of our methodology is synthesis with "idle" cycles across a network of workstations. Even if the number of iterations is greatly reduced through intelligent search, simulation based characterization is computationally expensive. To ensure that our formulation is viable and deployable across an existing hardware infrastructure, we have implemented a synthesis strategy that uses unused/idle CPU resources on a generic network of workstations. Large design organizations usually maintain a large pool of desktop workstations that are completely idle for 12 hours a day. This approach has the potential to yield an enormous amount of compute power at essentially no cost.

Fig. 3 provides an overview of our distribution strategy. The strategy consists of four separate applications that have been implemented as part of the ASF synthesis framework:

- *asf_prsa_node.* asf_prsa_node implements all the stochastic and deterministic components of our optimization strategy, as described above. Each PRSA node submits candidate solution characterization requests to the asf_master for evaluation. After a request is completed processed, the characterization results are passed back to the corresponding asf_prsa_node. The PRSA nodes also exchange state information in peer-to-peer transactions. A small queue of recent crossover states is maintained in each asf_prsa_node as source material for crossover splicing.
- *asf_sleeper.* An asf_sleeper process runs on all potential slave nodes. It monitors user active and system resource utilization. The asf_sleeper supports several usage policies. Under the most stringent usage policy (the default), a machine is only used if there is no user activity for at least 20 minutes and the machine is completely idle. It is also the responsibility of the asf_sleeper to enforce the usage policy. If a usage policy violation occurs (e.g., if the user returns and moves the mouse), the asf_sleeper kills all asf_slaves and associated SPICE jobs within 2 seconds, leaving the machine completely unloaded. The asf_sleeper also ensures that a host has suf-

ficient resources (such as physical memory) to host additional asf_slaves before they are started.

- *asf_master*. The asf_master starts asf_slaves on idle compute resources and administers the distribution of evaluation requests such that all available nodes are equally utilized. As noted previously, candidate solutions are usually characterized with several evaluators. The asf_master makes sure that each evaluator gets run for each characterization request in an environment where the resource set is perpetually changing. For example, when an asf_sleeper kills a set of asf_slaves as a result of a usage policy violation, the asf_master reschedules the evaluation requests associated with the dead asf_slaves with other asf_slaves. Further, as the available resource set changes, the asf_master changes the allocation of asf_slaves to ensure that the resource set is fully utilized.
- *asf_slave*. Each asf_slave instantiates one evaluator and its associated SPICE simulator. It processes evaluation requests from the asf_master in FIFO order. Execution continues until a kill signal is received from either the asf_master or asf_sleeper.

V. EVALUATORS

Given a sizing and biasing solution, an evaluator measures a set of one or more circuit performance attributes with the use of simulation. Fig. 2 illustrated this process. After it is fully instantiated, an evaluator consists of a test circuit SPICE deck and a simulation output analysis script. The candidate solution is embedded into the test circuit. The test circuit contains the bias and/or feedback network necessary to configure the candidate solution such that the desired circuit performance characteristics can be measured. The test deck also determines the stimulus that will be applied to the test circuit during the simulation and specifies the set of simulation results that will be captured for output analysis. After the test deck is loaded into the simulator, the simulation output is captured as a set of two-dimensional data vectors.

As shown in Fig. 2, the simulation output is typically a set of one or more waveforms, representing a potentially large amount of simulation data. This data has to be analyzed and reduced to a meaningful set of performance attributes. This task is performed by the output analysis script. This script examines the simulation data and computes the performance attributes. In the ASF framework, analysis scripts are written in Octave [3], a GNU open source clone of Matlab [16]. Octave was chosen for this task for several reasons. First, like Matlab, it is particularly well suited for this task because it provides an extensive and powerful set of functions for manipulating data in matrix form. Second, many analog designers are already familiar with Matlab. Matlab, with its rich set of constructs for signal processing, is frequently used as a system-level macro modeling language and environment. In addition, because Octave is a free open source project, we were able to modularize and incorporate it directly into the ASF framework. The tight integration eliminates the overhead associated with inter-process communication and improves efficiency and reliability.

A. Evaluator Components

Evaluators consist of three components that fulfill the functionality of the evaluator and enable the built-in reuse strategy:

- *Test Circuit Template.* As shown in Fig. 4, the test circuit template is the basis for the test deck that will be loaded and simulated. The final test deck is constructed from the test circuit template when the evaluator is instantiated with the user's topology and configuration parameters.
- *Simulation Analysis Script Template.* The analysis script, which is used to extract the performance attributes from the simulation data, is generated from its template based upon the user's configuration.
- *Evaluator Configuration File.* This file specifies three aspects of the evaluator configuration. First, the configuration file specifies the simulator and its configuration options. The second component of the configuration file specifies the evaluator parameters. A default value and change string is specified for each parameter. The default value is utilized in the instantiation of the evaluator, unless the user provides an overriding value in the synthesis task description. The second item of information, the change string, is utilized after the evaluator has been instantiated. It is frequently desirable to specify adjustable evaluator parameters. Consider a parameter that controls the temperature in the circuit simulation. It needs be set as a function of a variable that captures the temperature range over which the circuit has to function according to its specifications. The change string is required to tell the synthesis framework what command to use to alter such an adjustable configuration parameter. The final component of the configuration file specifies the set of performance attributes that are be computed by the evaluator. They are used to construct the optimization objectives of the synthesis task.

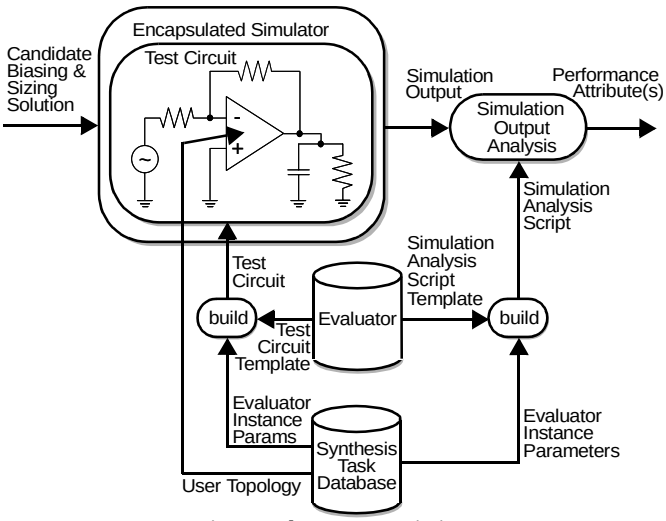


Fig. 4 Evaluator Instantiation

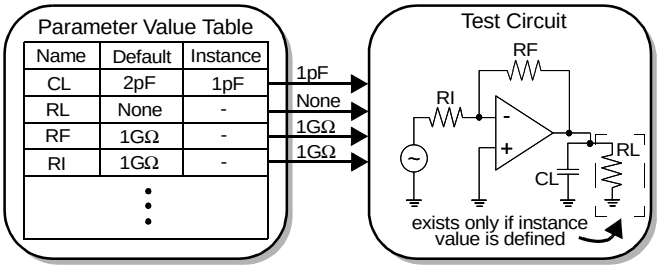


Fig. 5 Built-in Reuse Strategy

B. Build-In Reuse Strategy

Reconfigurability allows the evaluator to be reused for various synthesis tasks. This is achieved through parameterization of relevant values in the test circuit and the analysis script. Consider the simple test circuit for single-ended op-amps shown in Fig. 5. The load capacitor (CL) and feedback network components (RF and RI) are parameterized. For each parameter, the default value is specified in the evaluator configuration file. Unless the user provides an overriding value in the synthesis task description, the default value is utilized in the instantiation. Thus, in the example shown in Fig. 5, CL gets the value from the synthesis task description (1pF), while RF and RI retain their default values (both 1GΩ). The methodology also allows evaluator parameters to determine the test circuit topology. As an example, RL is an optional element. If the user specifies RL, it is incorporated into the test circuit; otherwise it is completely omitted.

In the ASF framework, the final test deck and analysis script are generated with the use of cpp [23], the C preprocessor. cpp is a macro processing and text substitution engine. It offers the correct balance of functionality and simplicity for the parameter substitution task. Also, because it is a standardized component of ANSI C, it should be familiar to all individuals who have taken a C programming class.

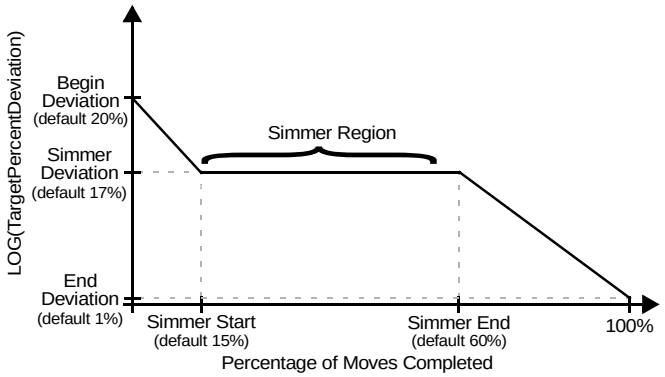


Fig. 6 Target Deviation Percent for the DeviationTracker Schedule

VI. DeviationTracker and The Downhill Set

DeviationTracker is a novel annealing cooling schedule that has been designed to integrate a set of classical downhill optimization methods into a stochastic optimization formulation. Each downhill optimizer invocation, referred to as a MegaMove, returns a local minimum in the neighborhood of the current annealer state. After the MegaMove completes, DeviationTracker encourages the annealer to remain in the neighborhood of that local minimum, while ensuring that the annealer does not remain trapped in that local minimum. The size of the neighborhood is reduced over the course of the annealing run, eventually preventing the annealer from escaping its final solution.

The DeviationTracker cooling schedule is derived from the Modified Lam cooling schedule [24]. Like Modified Lam, it is an empirical fixed length cooling schedule that adjusts the temperature at a prespecified fixed interval. DeviationTracker also builds a trajectory array and uses a feedback mechanism to adjust the temperature. Unlike Modified Lam, *the temperature is adjusted such that the cost deviation follows a specified trajectory.* The cost deviation is defined as the standard sample deviation of all accepted cost values sampled during the previous temperature,

$$C_{deviation} = \sqrt{\frac{\sum_{i=1}^{n} (C_i - \bar{C})^2}{n-1}} \qquad (1)$$

where C_i is the cost of the ith accepted candidate solution evaluated at the previous temperature and $\bar{C}$ is the sample mean. The target deviation is computed with the use of a target trajectory array referred to as the *TargetDevPercentAry* . The array entries specify the target deviation as a function of the current best cost. Specifically, the target deviation is given by

$$TD = BestCost \cdot TargetDevPercentAry[TemperatureCount] \qquad (2)$$

where the *TemperatureCount* variable keeps track of the annealer's progress through the fixed length schedule. Fig. 6 shows the trajectory stored in the *TargetDevPercentAry* as a function of temperature count. Under the control of the DeviationTracker cooling schedule, the optimization process can be divided into three phases that are repeated until the optimizer terminates. Each phase is described below.

Phase I: MegaMove. DeviationTracker selects, at random, one of the downhill optimizers. Each of the available optimizers has an equal probability of being selected. The Downhill Set consists of:

- *DownhillGreedy.* DownhillGreedy is a random walk component optimizer. It starts with the annealer's current candidate solution state. Like the annealer, it stochastically selects design variables, perturbing them one at a time according to a step range. The algorithm only accepts perturbations that decrease the cost. The step range is gradually reduced if the algorithm fails to make progress. The algorithm terminates if insufficient progress is made.
- *DownhillCS.* DownhillCS is a coordinate search component optimizer, derived from [20]. Like DownhillGreedy, DownhillCS starts with the annealer's current candidate solution and perturbs design variables one at a time. The optimization process consists of perturbation sets, where each design variable is perturbed once. The order in which variables are perturbed is selected stochastically and changes for each perturbation set. The algorithm only accepts perturbations that decrease the cost. For each variable, the algorithm attempts a perturbation in one randomly selected direction and, if the perturbation fails to yield a lower cost state, it attempts a perturbation in the opposite direction. The size of each perturbation is bounded by a step range. DownhillCS starts with a large step range, which is gradually reduced if the algorithm fails to make progress. The algorithm terminates if insufficient progress is made.
- *DownhillSimplex.* DownhillSimplex is a Nelder-Mead Simplex component optimizer [17]. [8] provides an overview of the algorithm with a useful flow diagram. [22] provides further discussion of the algorithm and an implementation in C. A detailed discussion of Nelder-Mead Simplex is beyond the scope of this document. [12] provides a detailed description of our implementation.
- *DownhillPowell.* DownhillPowell is a Powell's method component optimizer [21]. [8] provides a useful overview of the algorithm. [1] provides a detailed discussion of this algorithm, several insightful extensions, and a complete implementation in ALGOL W. [22] also provides an overview of the algorithm and an implementation in C. [12] provides a detailed description of our implementation.

Most of the component optimizers in the Downhill Set take a set of configuration parameters that determine how hard and with what tolerance the downhill optimizer will look for a local optimum. These configuration parameters are set as a function of the percentage of the cooling schedule that has been completed. At the beginning of the cooling schedule, the optimizer favors less accuracy and shorter search. Conversely, at the end of the cooling schedule, the optimizer favors more accuracy and longer search. At the beginning of the run, the annealer is primarily interested in finding a neighborhood with good solutions. Finding the exact bottom of the local minimum is a waste of compute time since the annealer will most likely skip out of it anyway. However, towards the end of the run, the annealer is interested in finding the best final solution. At that time, seeking out the exact minimum is worthwhile.

Phase II: Anneal Set. The selected Downhill Set optimizer returns its solution to the annealer. The annealer, in turn, adopts this solution as its current state. At this point, DeviationTracker has to select a temperature that will encourage the annealer to remain in the neighborhood of the MegaMove solution while, at the same time, ensuring that the optimizer does not remain permanently trapped in that minimum. Specifically, we want to set the temperature such that we obtain the cost deviation specified in *TargetDevPercentAry* . Setting the temperature to match the target deviation after a MegaMove is a two step process. For the first temperature after the MegaMove, the temperature value is approximated with the use of a heuristic.

(A) $DevCostPercentTarget = TargetDevPercentAry[TemperatureCount]$

(B) $DevCostTarget = DevCostPercentTarget \cdot BestCost$

(C) $Temperature^{new} = DevCostTarget \cdot DevToTempConstant$

In our formulation, we have found that a *DevToTempConstant* equal to 1.2 works reasonably well. This value typically produces a deviation slightly less than the desired amount. We correct for the pessimistic estimate in the next step.

Phase III: Anneal Adjust. The temperature value computed with the heuristic above is used until the next temperature check point. At that juncture, a feedback mechanism, conceptually derived from Modified Lam, is used to adjust the temperature. So, if the actual deviation is greater than the target deviation, the temperature is decreased. A decrease in temperature should decrease the deviation of accepted move costs. Conversely, if the actual deviation is less than the target deviation, an increase in temperature should increase the deviation of accepted move costs. Pseudo-code for adjusting the temperature is shown below:

(A) Retrieve the cost deviation of accepted moves during the last temperature from the annealer's cost statistics and assign it to $ActualDevCost$

(B) Set $DevCostPercent = \frac{ActualDevCost}{BestCost}$

(C) Set $DevCostTarget = TargetDevPercentAry[TemperatureCount]$

(D) Set $Adjust = \frac{DevCostPercent - DevCostTarget}{DampingConstant \cdot DevCostTarget}$

(E) Place limits on $Adjust$: if $(Adjust > 0.5)$

(1) Set $Adjust = 0.5$

(F) Else if $(Adjust < -0.5)$

(1) Set $Adjust = -0.5$

(G) Compute new temperature:

(1) Set $Temperature^{new} = Temperature^{old} \cdot (1 - Adjust)$

In the default DeviationTracker configuration, the temperature is adjusted three times according to this feedback mechanism before attempting another MegaMove. So, the DeviationTracker schedule can be summarized by the following sequence of six steps:

(A) MegaMove: Pick a downhill optimizer and execute it.
(B) Anneal Set: Set the annealing temperature according to desired deviation. Perform 50 (default) moves.
(C) Anneal Adjust #1: Use feedback mechanism to adjust the temperature to the desired deviation and perform 50 moves.
(D) Anneal Adjust #2.
(E) Anneal Adjust #3.
(F) Go to step (A).

Before continuing to the next topic, some concluding remarks regarding this cooling schedule need to be made. First, this cooling schedule discards many traditional ideas found in an annealing style optimization formulation. It attempts to minimize run-time by narrowing the search to a subset of interesting local minima. To phrase this differently, this approach could be characterized as random multi-start in the neighborhood of a good answer. The annealer, using this cooling schedule, serves as a stochastic mechanism for generating starting points. Initially, all starting points have an equal probability of selection. However, as the cooling schedule progresses, the mechanism increasingly favors starting points in the neighborhood of previously obtained good answers.

VII. Experimental Results

A. Three Benchmark Circuits

To demonstrate the viability of our methodology, we have tested our synthesis approach on several circuits of varying degrees of difficulty. Fig. 7, Fig. 8, and Fig. 9 show three production op-amps from Texas Instruments. The folded cascode op-amp in Fig. 7 and the simple two-stage op-amp in Fig. 8 were synthesized in a 0.6µm, 3.3V technology. The power amplifier in Fig. 9 was synthesized in a 1.0µm, 5.0V technology. The synthesis tasks had 13, 15, and 20 independent variables, respectively. Each of the variables had a broad (yet reasonable) range. A randomly generated starting was used.

This synthesis methodology supports two types of objectives: constraints and goals. Each constraint has a target value that has to be satisfied to make that circuit useful for its application. Each goal has a target range. The synthesis tool seeks a solution where all constraints are satisfied and each of the goals falls within its target range. Once a satisfactory solution is found, the optimizer tries to further push each goal, while ensuring that the constraints continue to be satisfied.

Table 1 shows that the synthesis result for the folded cascode op-amp. The result satisfies all constraints and both goals (MOS area and static power) were optimized beyond the original hand design. 3,573 candidate solution states were evaluated during the synthesis run. Since each candidate solution state characterization utilized 6 evaluators and each evaluator ran one SPICE simulation, a total of 21,438 SPICE simulations were performed. The slave pool consisted of 10 desktop Ultra Sparc Solaris machines. This synthesis run employed

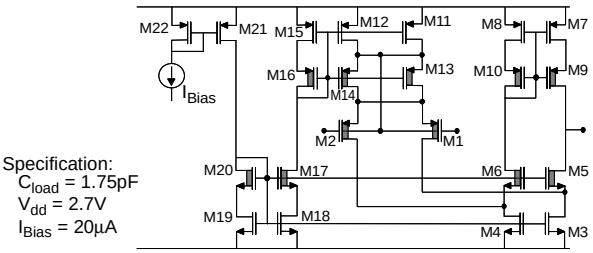


Fig. 7 Folded Cascode Op-Amp

Table 1. ASF Result for Folded Cascode Opamp Circuit in Fig. 7

	Manual Design	Objective Type	Synthesis Target	Synthesis Result
Constraints/Goals:				
DC gain (dB)	72	Constraint	≥ 71	73
Gain Bandwidth (MHz)	167	Constraint	≥ 162	172
Phase Margin (deg)	53	Constraint	≥ 52	52
Noise 1kHz (nV $Hz^{-0.5}$)	70	Constraint	≤ 70	63
Noise 10MHz (nV $Hz^{-0.5}$)	3.9	Constraint	≤ 4.0	3.6
CMRR (dB)	108	Constraint	≥ 108	108
PSRR Vss (dB)	89	Constraint	≥ 89	92
PSRR Vdd (dB)	72	Constraint	≥ 72	74
Settling Time[1] (ns)	17.3	Constraint	≤ 17.3	16.7
DC Offset (mV)	0.10	Constraint	≤ 0.20	0.088
DC Bias (mV)	0	Constraint	≤ 30	22
MOS Area ($10^3\mu^2$)	28	Min Goal	26 to 35	26
Static Power (mW)	11.4	Min Goal	9 to 12	11.1
Run-time Info:				
Independent Variables				13
States Evaluated				3573
Eff. Run-time (min)				26

(1) 0.1V input step, 10-bit accurate final voltage

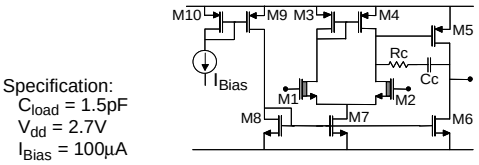
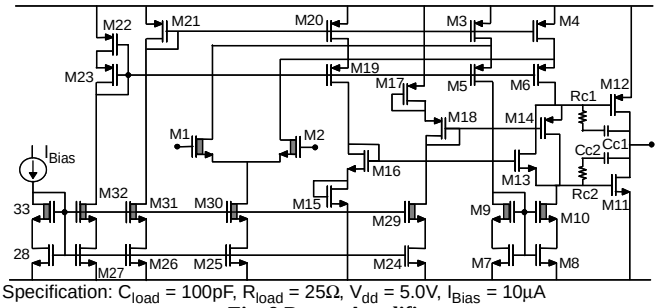


Fig. 8 Simple Two-Stage Op-Amp

Table 2. ASF Result for Simple Two-Stage Op-Amp in Fig. 8

	Manual Design	Objective Type	Synthesis Target	Synthesis Result
Constraints/Goals:				
DC gain (dB)	68	Constraint	≥ 68	69
Gain Bandwidth (MHz)	255	Constraint	≥ 255	258
Phase Margin (deg)	60	Constraint	≥ 56	58
Noise 1kHz (nV $Hz^{-0.5}$)	145	Constraint	≤ 145	145
Noise 10kHz (nV $Hz^{-0.5}$)	46	Constraint	≤ 46	46
CMRR (dB)	76	Constraint	≥ 76	77
PSRR Vss (dB)	85	Constraint	≥ 85	86
PSRR Vdd (dB)	70	Constraint	≥ 70	71
Settling Time[1] (ns)	5.0	Constraint	≤ 5.0	4.9
DC Offset (mV)	0.123	Constraint	≤ 0.125	0.124
DC Bias (mV)	0	Constraint	≤ 30	27
Gain Margin (dB)	18	Constraint	≥ 17	17
Area ($10^3\mu^2$)	9.4	Min Goal	9 to 15	9.0
Static Power (mW)	24.6	Min Goal	20 to 24.5	24.2
Run-time Info:				
Independent Variables				15
States Evaluated				17952
Eff. Run-time (min)				74

(1) 0.5V input step, 10-bit accurate final voltage



Specification: C_{load} = 100pF, R_{load} = 25Ω, V_{dd} = 5.0V, I_{Bias} = 10μA

Fig. 9 Power Amplifier

Table 3. ASF Result for Power Amplifier in Fig. 9

	Manual Design	Objective Type	Synthesis Target	Synthesis Result
Constraints/Goals:				
DC gain (dB)	92	Constraint	≥ 92	93
Gain Bandwidth (MHz)	0.67	Constraint	≥ 0.60	1.0
Phase Margin (deg)	86	Constraint	≥ 84	84
Noise 1kHz (nV $Hz^{-0.5}$)	52.2	Constraint	≤ 52.5	48.9
Noise 10kHz (nV $Hz^{-0.5}$)	22	Constraint	≤ 40	20
CMRR (dB)	140	Constraint	≥ 138	140
PSRR Vss (dB)	91	Constraint	≥ 90	99
PSRR Vdd (dB)	94	Constraint	≥ 94	102
THD[1]	0.057	Constraint	≤ 0.060	0.046
THD[2]	0.10	Constraint	≤ 0.11	0.086
DC Offset (mV)	0.066	Constraint	≤ 0.20	0.029
DC Bias (mV)	0	Constraint	≤ 50	46
Area ($10^3\mu^2$)	81.6	Min Goal	75 to 100	90.5
Static Power (mW)	19	Min Goal	14 to 19	15
Run-time Info:				
Independent Variables				20
States Evaluated				3085
Eff. Run-time (min)				38

(1) 4.0V p-p 1kHz input (2) 2.6V p-p 1kHz input, R_L = 5Ω

the most restrictive slave host usage policy; machines were only used if there was no user activity for at least 20 minutes and the machines were completely idle. Wall clock run-time was 46 minutes, but ASF was completely idle for the first 20 minutes to ensure compliance with the usage policy. Thus, effective run-time was 26 minutes.

Table 2 and Table 3 show the results for the simple two-stage op-amp and the power amplifier, respectively. These results were obtained with the same machine configuration as was used for the folded cascode result in Table 1. The two-stage op-amp synthesis task utilized 6 evaluator instances. The power amplifier utilized 8 evaluator instances. The same set of eight single-ended op-amp evaluators were reused in all three synthesis tasks. These benchmark circuits have different topologies and are targeted to a set of two different manufacturing technologies that utilize different power supply voltages. So, while this evidence is limited in scope, it does demonstrate that it is possible to create a set of modular and reusable evaluators that can, with appropriate configuration, be utilized to characterize a whole family of analog blocks. This allows the maintainer of the synthesis framework to create a set of evaluator libraries that cover a broad range of analog blocks. Reusable evaluators are a critical cornerstone of this formulation. They are essential to satisfying our goal of creating a easy to use general purpose methodology. Assuming all evaluators are instantiated from a library, this methodology can easily be integrated with an existing commercial schematic capture environment.

B. Serial Modified Lam vs. Parallel DeviationTracker

Fig. 10 compares a serial Modified Lam configuration to a parallel DeviationTracker configuration. The graph shows the final cost as a function of the number of characterization requests for 10 serial Modified Lam synthesis runs and 10 parallel DeviationTracker synthesis runs. The Modified Lam configuration utilized one annealer. The cooling schedule trajectory was configured for 3000 temperature

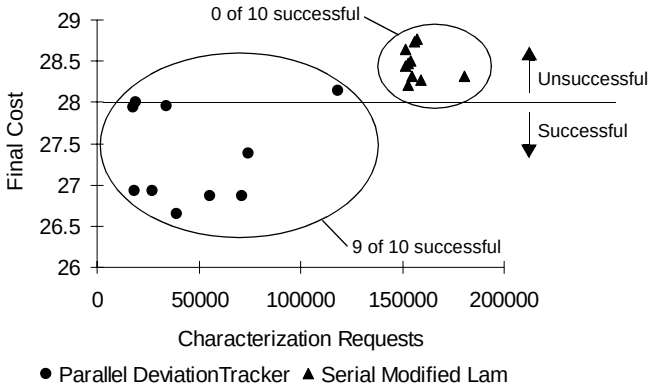


Fig. 10 Robustness and Run-time Comparison for Two-Stage Op-Amp

adjustments. 50 moves were performed at each temperature. Since the configuration only utilized one annealer, all crossover related mechanisms were disabled. The DeviationTracker configuration explored the solution space in parallel with a population of 10 optimizers. Only standard uniform crossover [12] was used as a synchronization mechanism. The cooling schedule trajectory was configured for 80 temperature adjustments. 50 moves were performed at each temperature. A MegaMove was performed every 4 temperatures (200 moves). It should be noted that ASF terminates early if it finds the bottom of a feasible local minimum.

Both experiments were performed on the two-stage op-amp synthesis task shown in Fig. 8. For this synthesis task, a cost less than or equal to 28 signifies that all goals and constraints have been satisfied. If the final cost is greater than 28, one or more of the constraints and goals are not satisfied. As shown in Fig. 10, the DeviationTracker configuration was successful in 9 out of 10 runs. The Modified Lam configuration failed in all 10 runs. On average 47,275 characterization requests were performed for each synthesis run in the DeviationTracker experiment. On average 157,052 characterization request were performed for each synthesis run in the Modified Lam experiment. In a similar experiment, DeviationTracker successfully synthesized the power amplifier (Fig. 9) in 10 out of 10 runs. On average 3,684 characterization requests were performed in each run. These results show that the DeviationTracker approach produces consistently higher quality results while, at the same time, reducing the number of required state evaluations.

It is also important to note that the parallel DeviationTracker configuration has a larger maximum degree of parallelism than the serial Modified Lam configuration. The DeviationTracker configuration uses 10 optimizers, which synchronize asynchronously through crossover, to exploring the candidate solution space simultaneously. Thus, at any point in time, the framework is evaluating 10 candidate solutions in parallel. In contrast, all candidate solutions are evaluated sequentially in the Modified Lam configuration. Thus, even though both experiments were conducted on roughly equivalent pool of 10 machines, the DeviationTracker configuration performed, on average, 20.8 SPICE evaluations per second while the Modified Lam configuration performed 8.7 SPICE evaluations per second.

C. Comparison of Downhill Set Optimizers

Fig. 11 compares the effectiveness of all the component optimizer variants implemented in the Downhill Set. These component optimizers are invoked explicitly as part of the DeviationTracker cooling schedule to search out a local minimum in the neighborhood of the current annealer state. Four experiments, each consisting of 10 synthesis runs, were performed. In each experiment, only one of the Downhill Set component optimizers was utilized. Except for the MegaMove configuration, the same optimizer configuration was utilized for all experiments. The defaults were used for all DeviationTracker trajectory parameters as shown in Fig. 6, except the end deviation was set to 2%. The cooling schedule trajectory was configured for 80 temperature adjustments, with 50 moves at each temperature. A MegaMove was performed every 4 temperatures (200

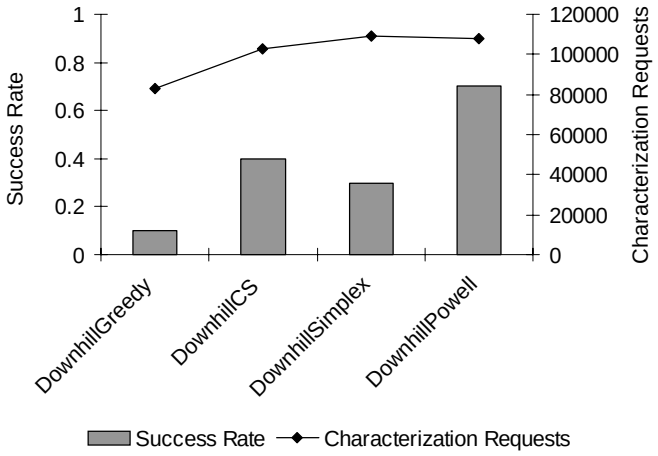


Fig. 11 Comparison of Downhill Set Optimizers for Two-Stage Op-Amp

moves). A population of 10 optimizers was used with all crossover mechanisms enabled [12]. The two-stage op-amp synthesis task was used. For this synthesis task, the DownhillPowell component optimizer proved to be the most reliable. It is also worth observing that the coordinate search variant is actually more robust than the greedy random walk. Both of these optimizers perform an undirected stochastic exploration of the cost surface. The only difference is that the search is structured more rigidly in the coordinate search formulation. As these results suggest, the structure of the search can have a non-trivial impact on robustness.

Additionally, it should be noted that the best choice of component optimizer depends on the cost surface and difficulty of the synthesis task. The power amplifier, for example, is considerably easier to synthesize. All of the component optimizers can synthesize that circuit with a perfect success rate. Given that all variants in the Downhill Set can do the job, DownhillGreedy has the lowest average run-time. Also, DownhillSimplex provides the best overall solution quality for that synthesis task. These results show that no one component optimizer is ideally suited for all synthesis task cost surfaces. Thus, the optimizer formulation should incorporate a variety of different component optimizers to ensure the best possible results across a broad range of synthesis tasks.

VIII. Conclusions

This paper described a novel cell-level analog synthesis solution that can size and bias a given circuit topology subject to a set of performance objectives and a manufacturing process. It incorporates modular, reusable, user-configurable test benches called evaluators. Evaluators are both topology and technology independent. They allow us to create a programming-free synthesis methodology that can be integrated into a commercial schematic capture environment. Furthermore, ASF employs a novel numerical optimization formulation that incorporates classical downhill techniques into stochastic search. It can, for example, consistently synthesized a 20 variable op-amp in 10x fewer candidate solution evaluations than previously published approaches that rely on traditional stochastic optimization methods. The approach reduces the number of state evaluations while, at the same time, improving solution quality and optimizer consistency.

Acknowledgments

Felicia James (TI), more than anyone else, has ensured the continued funding of this project. In addition, her feedback and encouragement has shaped the direction and development of this project. David Yeh (TI) and David Guillou (CMU) provided help and guidance throughout this project and contributed to its development through many valuable discussions. Brent Bearden (TI), Charles LeMaire (TI), Laura Herriott (TI), Chuck Tomiello (TI), Tamal Mukherjee (CMU), and Gary Richey (TI) provided assistance at various stages of this project. Lawrence Arledge (TI) is an obnoxious and cruel proof reader; this paper has greatly benefited from his feedback. Finally, in addition to Texas Instruments, this research has been generously supported by the Semiconductor Research Corporation under SRC contract DC-068.065.

References

[1] R.P. Brent, *Algorithms for Minimization without Derivatives*, Prentice Hall, Englewood Cliffs, New Jersey, 1972.

[2] M.G. DeGrauwe *et al.*, "Towards an Analog System Design Environment," *IEEE J. of Solid-State Circuits*, SC-24(3), pp. 659-672, June 1989.

[3] J.W. Eaton, *Octave Home Page*, 1998, <http://www.octave.org/>, (27 December 2000).

[4] G. Gielen, P. Wambacq, W. Sansen, "Symbolic Analysis Methods and Applications for Analog Circuits: A Tutorial Overview," *Proc. IEEE*, vol. 82, no. 2, Feb. 1994.

[5] R. Harjani, R.A. Rutenbar, and L.R. Carley, "OASYS: A Framework for Analog Circuit Synthesis," *IEEE Trans. on Computer-Aided Design*, CAD-8(12), pp. 1247-1266, December 1989.

[6] M. del Mar Hershenson, S.P. Boyd, T.H. Lee, "GPCAD: A tool for CMOS op-amp synthesis," *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, pp. 296-303, November 1998.

[7] J.H. Holland. *Adaptation in Nature and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[8] S.L.S. Jacoby, J.S. Kowalik, and J.T. Pizzo, *Iterative Methods for Nonlinear Optimization Problems.* Prentice Hall, Englewood Cliffs, N.J., 1972.

[9] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, 220(4598), pp. 671-680, November 1983.

[10] H.Y. Koh, C.H. Sequin and P.R. Gray, "OPASYN: A Compiler for MOS Operational Amplifiers," *IEEE Trans. on Computer-Aided Design*, CAD-9(2), pp. 113-125, February 1990.

[11] M. Krasnicki, R. Phelps, R. Rutenbar, and R. Carley, "MAELSTROM: Efficient simulation-based synthesis for custom analog cells," *Proc. of the 1999 ACM/IEEE Design Automation Conference*, pp. 945-950, 1999.

[12] M.J. Krasnicki, *A Methodology for Distributed Simulation-Based Synthesis of Custom Analog Circuits*, Ph.D. Thesis, Carnegie Mellon University, December 2000.

[13] S.W. Mahfoud and D.E. Goldberg, "Parallel Recombinative Simulated Annealing: A Genetic Algorithm," *Parallel Computing*, 21(1), pp. 1-28, 1995.

[14] P. C. Maulik, L. R. Carley, and R. A. Rutenbar, "Integer Programming Based Topology Selection of Cell Level Analog Circuits," *IEEE Trans. CAD*, vol. 14, no. 4, April 1995.

[15] F. Medeiro, F.V. Fernandez, R. Dominguez-Castro, and A. Rodriguez-Vazquez, "A Statistical Optimization-Based Approach for Automated Sizing of Analog Cells," *Proc. of IEEE International Conference Computer-Aided Design*, pp, 594-597, November 1994.

[16] C. Moler, "The MathWorks - MATLAB Introduction," *The MathWorks: Developers of MATLAB and Simulink for Technical Computing*, 2000, <http://www.mathworks.com/products/matlab/>, (27 December 2000).

[17] J.A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *Computer Journal*, vol. 7, pp. 308-313, 1965.

[18] W. Nye, D.C. Riley, A. Sangiovanni-Vincentelli and A. L. Tits, "DELIGHT.SPICE: An Optimization-Based System for Design of Integrated Circuits," *IEEE Trans. on Computer-Aided Design*, CAD-7(4), pp. 501-518, April 1988.

[19] E. Ochotta, R.A. Rutenbar, L.R. Carley, "Synthesis of High-Performance Analog Circuits in ASTRX/OBLX," *IEEE Trans. CAD*, vol. 15, no. 3, March 1996.

[20] R. Phelps, M. Krasnicki, R.A. Rutenbar, and L.R. Carley, and J.R. Hellums, "ANACONDA: Simulation-based Synthesis of Analog Circuits via Stochastic Pattern Search," *IEEE Trans. Computer-Aided Design*, CAD-19(6), pp. 703-717, June 2000.

[21] M.J.D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *Computer Journal*, vol. 7, pp. 303-307, 1964.

[22] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press*, Cambridge, Second Edition, 1992.

[23] R.M. Stallman, *The C Preprocessor*, 2000, <http://gcc.gnu.org/onlinedocs/cpp_toc.html>, (27 December 2000).

[24] W. Swartz and C. Sechen, "New Algorithms for the Placement and Routing of Macrocells," *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, pp. 336-339, November 1990.

[25] A. Torralba, J. Chávez and L. Franquelo, "FASY: A Fuzzy-Logic Based Tool for Analog Synthesis," *IEEE Trans. on Computer-Aided Design*, CAD-15(7), pp. 705-715, July 1996.