Anaconda: Simulation-Based Synthesis of Analog Circuits Via Stochastic Pattern Search

Rodney Phelps, *Student Member, IEEE*, Michael Krasnicki, Rob A. Rutenbar, *Fellow, IEEE*, L. Richard Carley, and James R. Hellums, *Senior Member, IEEE*

***Abstract*—Analog synthesis tools have traditionally traded quality for speed, substituting simplified circuit evaluation methods for full simulation in order to accelerate the numerical search for solution candidates. As a result, these tools have failed to migrate into mainstream use primarily because of difficulties in reconciling the simplified models required for synthesis with the industrial-strength simulation environments required for validation. We argue that for synthesis to be practical, it is essential to synthesize a circuit using the same simulation environment created to validate the circuit. In this paper, we develop a new numerical search algorithm efficient enough to allow full circuit simulation of each circuit candidate, and robust enough to find good solutions for difficult circuits. The method combines the population-of-solutions ideas from evolutionary algorithms with a novel variant of pattern search, and supports transparent network parallelism. Comparison of several synthesized cell-level circuits against manual industrial designs demonstrates the utility of the approach.**

***Index Terms*—Algorithms, analog synthesis, mixed-signal design, pattern search.**

I. Introduction

MIXED-SIGNAL designs are increasing in number as a large fraction of new integrated circuits (IC's) require an interface to the external, continuous-valued world. The digital portion of these designs can be attacked with modern cell-based tools for synthesis, mapping, and physical design. The analog portion, however, is still routinely designed by hand. Although it is typically a small fraction of the overall design size (e.g., 10 000–20 000 analog transistors), the analog partition in these designs is often the bottleneck because of the lack of automation tools.

The situation appears to be worsening as we head into the era of System-on-Chip (SoC) designs. To manage complexity and time-to-market, SoC designs require a high level of reuse, and cell-based techniques lend themselves well to a variety of strategies for capturing and reusing digital intellectual property (IP). But these digital strategies are inapplicable to analog designs, which rely for basic functionality on tight control of low-level device and circuit properties that vary from technology to technology. The analog portions of these systems are still designed by hand today. They are even routinely ported by hand as a given IC migrates from one fabrication process to another.

A significant amount of research has been devoted to cell-level analog synthesis, which we define as the task of sizing and biasing a device-level circuit with 10–50 devices. However, as noted in [1], previous approaches have failed to make the transition from research to practice. This is due primarily to the prohibitive effort needed to reconcile the simplified circuit models needed for synthesis with the "industrial-strength" models needed for validation in a production environment. In digital design, the bit-level, gate-level and block-level abstractions used in synthesis are faithful to the corresponding models used for simulation-based validation. This is not the case for analog synthesis.

Fig. 1 illustrates the basic architecture of most analog synthesis tools. An *optimization engine* visits candidate circuit designs and adjusts their parameters in an attempt to satisfy designer-specified performance goals. An *evaluation engine* quantifies the quality of each circuit candidate for the optimizer. Most research here focuses on tradeoffs between the optimizer (which wants to visit many circuit candidates) and the evaluator (which must itself trade accuracy for speed to allow sufficiently vigorous search). Much of this work is really an attempt to evade a harsh truth—that analog circuits are difficult and time-consuming to evaluate properly. Even a small cell requires a mix of ac, dc, and transient analyzes to correctly validate. In modern design environments, there is enormous investment in simulators, device models, process characterization, and "cell sign-off" validation methodologies. Indeed, even the sequence of circuit analyzes, models, and simulation test-jigs is treated as valuable IP here. Given these facts, it is perhaps no surprise that analog synthesis strategies that rely on exotic, nonstandard, or fast-but-incomplete evaluation engines have fared poorly in real design environments. To trust a synthesis result, one must first trust the methods used to quantify the circuit's performance *during* synthesis. Most prior work fails here.

Given the complexity of, investment in, and reliance on simulator-centric validation approaches for analog cells, we argue that for a synthesis strategy to have practical impact, it *must* use a simulator-based evaluation engine that is *identical* to that used to validate ordinary manual designs. This, however, poses significant challenges. For example, commercial circuit simulators are not designed to be invoked 50 000 times in the inner loop of

Manuscript received October 1, 1999. This work was supported in part by the Semiconductor Research Corporation (SRC) under contract 068, by the National Science Foundation (NSF), and grants from Texas Instruments and Rockwell Semiconductor. This paper was recommended by Associate Editor E. Charbon.

R. Phelps, M. Krasnicki, and L. R. Carley are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 15213 USA.

R. A. Rutenbar is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 15213 USA (e-mail: rutenbar@ece.cmu.edu).

J. R. Hellums is with the Mixed Signal Products Division, Texas Instruments Inc., Dallas, TX 75243 USA.

Publisher Item Identifier S 0278-0070(00)05346-X.

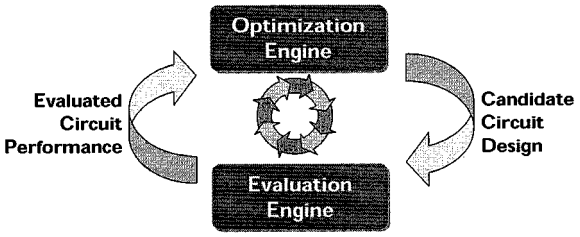


Fig. 1. Abstract model of analog synthesis tools.

a numerical optimizer. And, of course, the CPU time to visit and simulate this many solution candidates may be unacceptable.

In this paper, we develop a new strategy to support efficient simulator-in-the-loop analog synthesis. The approach relies on three key ideas. First, we *encapsulate* commercial simulators so that their implementation idiosyncrasies are hidden from our search engine. Second, we develop a novel global optimization algorithm called *stochastic pattern search*, combining ideas from evolutionary algorithms and pattern search, that is robust in finding workable circuits, and avoids the starting-point dependency problems of gradient and other down-hill search methods. Third, we exploit *network-level workstation parallelism* to render the overall computation times tractable. Our new optimization algorithm was designed to support transparent distribution of *both* the search tasks and the circuit evaluation tasks across a network.

We have implemented these ideas in a tool called ANACONDA. ANACONDA uses framework components from a companion synthesis tool, MAELSTROM [2]. ANACONDA has been successfully run on networks of 10–20 UNIX workstations, and currently runs Texas Instrument's proprietary TISpice circuit simulator as its evaluation engine [3]. In this paper, we extend the original treatment, describe in more detail the basic algorithms underlying ANACONDA, and present an expanded set of experimental synthesis results that demonstrate that simulator-in-the-loop synthesis can be made both practical and efficient for industrial designs. The remainder of the paper is organized as follows. In Section II, we briefly review prior work. In Section III, we formulate the overall synthesis problem and focuses on our global optimization algorithm, which we call *stochastic pattern search*. In Section IV, we offer experimental results on circuits. Finally, we offer some concluding remarks in Section V.

II. REVIEW OF PRIOR APPROACHES

Referring again to Fig. 1, we can broadly categorize previous work on analog synthesis by how it searches for solutions and how it evaluates each visited circuit candidate. See [4] for a more extensive survey.

Early work on synthesis used simple procedural techniques [5], rendering circuits as explicit scripts of equations whose direct evaluation completed a design. Although fast, these techniques proved to be difficult to update, and rather inaccurate. Numerical search has been used with equation-based evaluators [6]–[8], and even combinatorial search over different circuit topologies [9], [10], but equation-based approaches remain brittle in the face of technology changes. Recent work here has focused on coercing the required equations into a form more amenable to optimization; [11] shows results from rendering the equations as posynomials, thus, creating a convex optimization problem solvable via geometric programming. Hierarchical systems [12]–[15] introduced compositional techniques to assemble equation-based subcircuits, but still faced the same update/accuracy difficulties. Some of these systems can manipulate circuit equations automatically to suit different steps of the synthesis task [7]. Qualitative and fuzzy reasoning techniques [16], [17] have been tried to capture designer expertise, but with limited success. Equation-based synthesis offers fast circuit evaluation and is, thus, well suited to aggressive search over solution candidates. However, it is often prohibitively expensive to create these models—indeed, often more expensive than manually designing the circuit. Also, the simplifications required in these closed-form analytical circuit models necessarily limit their accuracy and completeness.

Symbolic analysis techniques, which have made significant strides of late [7], [8], [18]–[21] offer an automated path to obtaining some of these design equations. These techniques automatically derive reduced-order symbolic models of the linear transfer function of a circuit. The resulting symbolic forms can be obtained fairly quickly, offer good accuracy and can, thus, serve as evaluation engines. However, they are strictly limited to linear performance specifications, or at most some weakly nonlinear specifications [22]. Even a small analog cell may require a wide portfolio of dc, ac, and transient simulations to validate it. Symbolic analysis is a valuable but incomplete approach to circuit evaluation.

The synthesis systems most relevant to the ideas we develop in this paper are ASTRX/OBLX [1], [4] and the FRIDGE system from Seville [23]. In ASTRX/OBLX, we attacked the fundamental problem of tool usability with a compile-and-solve methodology. ASTRX starts from a SPICE deck describing an unsized circuit and desired performance specifications. ASTRX compiles this deck into a custom C program that implements a numerical cost function whose minimum corresponds to a good circuit solution for these constraints. OBLX uses simulated annealing [24] to solve this function for a minimum. This custom-generated cost code evaluates circuit performance via model-order reduction [25] for linear, small-signal analysis, and user-supplied equations for nonlinear specifications. ASTRX/OBLX was able to synthesize a wide variety of cells, but was still limited to essentially linear performance specifications. FRIDGE similarly uses annealing for search, but actually runs a SPICE-class simulator in its annealer. However, this tool appears to employ a simulator customized for synthesis, only evaluates a few thousand circuit candidates in a typical synthesis run (in contrast, OBLX evaluates 10^4–10^5 solutions), and has only been demonstrated solving problems with a small number of independent design variables.

Finally, we also note that there are several circuit *optimization* techniques that rely on simulator-based methods (e.g., [26]–[28]). For circuit optimization we assume a reasonable initial circuit solution, and seek to improve it. This can be accomplished with gradient and sensitivity techniques requiring a modest number of circuit evaluations. In contrast, in circuit *synthesis* we can assume nothing about our starting circuit

(indeed, we usually have *no* initial solution). This scenario is much more difficult as a numerical problem, and requires a global search strategy to avoid being trapped in poor local minima that happen to lie near the starting point.

The problem with all these synthesis approaches is that they use circuit evaluation engines different from the simulators and simulation strategies that designers actually use to validate their circuits. These engines tradeoff accuracy and completeness of evaluation for speed. We argue that this is no longer an acceptable tradeoff.

III. Synthesis Formulation

In this section, we present the full synthesis formulation of ANACONDA. Our circuit synthesis strategy relies on three key ideas: simulator encapsulation, a novel combination of population methods and pattern search, and scalable network parallelism. We describe these ideas below, beginning with a review of our basic synthesis-via-optimization formulation.

A. Basic Optimization Formulation

We use the basic synthesis formulation from OBLX [1], which we review here. We begin with a fixed circuit topology that we seek to size and bias. We approach circuit synthesis using a constrained optimization formulation, but solve it in an unconstrained fashion. We map the circuit design problem to the constrained optimization problem of (1), where is the set of independent variables—geometries of semiconductor devices, device multiplicities, and values of passive circuit components—we wish to change to determine circuit performance; $f(x)$ is a set of objective functions that codify performance specifications the designer wishes to optimize, e.g., power or bandwidth; and $g(x)$ is a set of constraint functions that codify specifications that must be beyond a specific goal, e.g., (gain $>$60 dB). Scalar weights, w_{fi}, balance competing objectives

$$\min_x \sum_{i=1}^{k} w_{fi} \cdot f_i(x) \quad \text{s.t. } g(x) \leq 0. \tag{1}$$

Formulation of the individual objective $f(x)$ and constraint $g(x)$ functions adapts ideas from [26]. We require a *good* value, and a *bad* value for each specification. These are used both to set constraint boundaries and to normalize the specification's range. For example, a single objective $f_i(x)$ is internally normalized as

$$\hat{f}_i(x) = \frac{f_i(x) - \text{good}}{\text{bad}_i - \text{good}_i}. \tag{2}$$

This normalization process provides a natural way for the designer to set the relative importance of competing specifications, and it provides a straightforward way to normalize the range of values that must be balanced in the cost function.

To support the stochastic pattern search optimizer we introduce in Section IV, we perform the standard conversion of this constrained optimization problem to an unconstrained optimization problem with the use of additional scalar weights. As a result, the goal becomes minimization of a scalar cost function, $C(x)$, defined by

$$C(x) = \sum_{i=1}^{k} w_{fi} \hat{f}_i(x) + \sum_{j=1}^{l} w_{gj} \hat{g}_j(x). \tag{3}$$

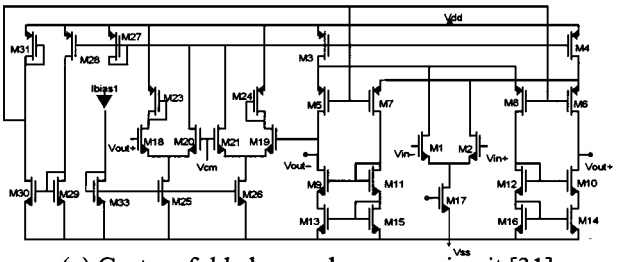
The key to this formulation is that the minimum of $C(x)$ corresponds to the circuit design that best matches the given specifications. Thus, the synthesis task becomes two more concrete tasks: evaluating $C(x)$ and searching for its minimum. Neither of these are simple. Our major contributions here are a new algorithm for global search that is efficient enough to allow use of commercial circuit simulators to evaluate $C(x)$, and a methodology for encapsulating simulators to hide unnecessary details from this search process. We treat the encapsulation methodology next.

B. Simulator Encapsulation for Simulation-Based Evaluation

Our overall goal is to be able to use the simulation methods trusted by designers—but *during* analog cell synthesis. This means invoking a sequence of detailed circuit simulations for each evaluation, $C(x)$, during numerical search. Although different SPICE-class simulation engines share core mechanisms and offer similar input/output formats, they remain highly idiosyncratic in many features. In our experience, the mechanics of embedding a simulator inside a numerical optimizer are remarkably untidy. This is a real problem since we seek a strict separation of the circuit optimization and circuit evaluation engines, and would like ultimately to be able to "plug in" different simulators. We handle this problem using a technique we refer to as *simulator encapsulation.*

Simulator encapsulation hides the details of a particular simulator behind an insulating layer of software. This software "wrapper" renders the simulator an object with a set of methods, similar to standard object-oriented programming ideas. Class members are invoked to perform a simulation, to change circuit parameters, to retrieve simulation results, and so forth. Clearly one major function of this encapsulation is to hide varying data formats from the optimizer; this engine need not concern itself with the details of how to invoke or interpret an ac, dc, or transient analysis.

A more subtle function of encapsulation is to insulate the optimization engine from "unfriendly" behavior in the simulator. Most simulators are designed either for batch-oriented operation, or for interactive schematic-update-then-simulate operation. In the latter, the time scales are optimized for humans—overheads of a few seconds per simulation invocation are negligible. But inside a numerical optimizer that seeks to run perhaps 100 000 simulations, these overheads are magnified. Our ideal is a simulator which can be invoked once, and, remaining live, can interpret quickly a stream of requests to modify circuit values and resimulate. Few simulators approach this ideal. For example, some insist on rechecking a licence manager key (possibly located remotely on a network) for every new simulation request; others flush all internal state or drop myriad temporary files in the local file system. Of course, the maximally difficult behavior exhibited by a simulator is a crash, an occurrence far from rare even in commercial



(a) Custom folded cascode opamp circuit [31]

(b) Simulation-based synthesis result for above circuit, on a 55MHz IBM Power2

Attribute	Manual Design	Auto-Synthesis: Spec	Result
CLoad (pF)	1.25	1.25	
Vdd (V)	5	5	
DC Gain (dB)	71.2	≥ 71:	91
UGF (MHz)	47.8	≥48:	55
Phase Margin (deg)	77.4	≥ 77:	83
PSRR - Vss (dB)	92.6	≥ 93:	119
PSRR - Vdd (dB)	72.3	≥ 72:	92
Output Swing (V)	± 1.4	± 1.4:	± 1.4
Settling Time (ns)	-	↓[a]:	47
Active Area ($10^3\mu^2$)	68.7	↓:	28
Circuits Evaluated		17,100	
CPU (hours)		11	

a. ↑ means maximize, while ↓ means minimize.

Fig. 2. Custom folded cascode opamp circuit and basic simulation-based annealing synthesis result.

offerings. This is especially problematic in synthesis, since the optimization engine may often visit circuit candidates with highly nonphysical parameter values, which occasionally cause simulator failure. Our encapsulation not only detects the crash but also restarts and reinitializes the simulator, all transparent to the optimizer. All these difficult behaviors can be hidden via appropriate encapsulation.

C. Global Optimization Issues

As in OBLX [1], we again favor global, stochastic search algorithms for the optimization engine because of their empirical robustness in the face of highly nonlinear, nonconvex cost functions. However, in OBLX we made an explicit tradeoff to use a customized, highly tuned, very fast circuit evaluator to permit search over a large number of solution candidates. When we replace this custom evaluator with commercial circuit simulation, we are faced with a 10× to 100× increase in CPU time. The central question we address in this section is how to retain the virtues of global, stochastic search, but deal with the runtime implications of simulator-in-the-loop optimization.

Before we describe our new optimizer, it is worth justifying our focus on global, rather than gradient-style local optimization. Given a good implementation of simulator encapsulation, we can replace the custom circuit evaluation used in OBLX with full, detailed simulation. We have rewritten the core annealing engine of OBLX in the form of a new, component-based optimization library called ANNEAL++ [29]. ANNEAL++ offers a range of annealing cooling schedules, move selection techniques, and dynamic updates on cost function weights. As an experiment, we encapsulated the Cadence Spectre circuit simulator [30] and used it with ANNEAL++ to resynthesize a standard OBLX benchmark circuit: the custom folded-cascode opamp from [31]. The circuit has 32 devices and 27 designable variables; the circuit schematic and associated synthesis results appear in Fig. 2.

This rather straightforward synthesis strategy yields an adequate result, albeit somewhat slowly. (In this simplified example, manufacturability concerns were ignored, which is the source of the extreme performance/area results; for annealing-style synthesis, these concerns can be addressed; we will return to these issues in Section IV.) Fig. 3 shows a set of sampled cross sections from the cost-surface for this annealing-style synthesis formulation. At an intermediate point in the synthesis, we stopped the optimizer, and then iteratively stepped each independent variable over its range, while freezing all other variables. At each step point we evaluated the synthesis cost function using Spectre. Fig. 3 shows a few of these resulting cross sections, suitably normalized for comparison. The mix of gently sloping plateaus and jagged obstacles is typical of these landscapes. We require global optimization algorithms because of their potential to avoid many of these inferior local minima.

However, annealing algorithms in particular have a reputation for slow execution because of the large number of solution candidates that must be visited. This is greatly exacerbated when we choose to fully simulate each solution candidate. There are three broad avenues of solution here.

1) **Less search:** attempt to sample the cost function at fewer points. This is essentially the approach taken by [23], which uses an unusual, truncated annealing schedule with some of the character of a random multistart approach. However, in our experience, wider search always yields better solutions and a more robust tool.
2) **Parallel circuit evaluation:** each visited circuit candidate requires more than one circuit simulation to evaluate it. We can easily distribute these over a network to parallel workstations. Indeed, our implementation supports this simple parallelism. For example, if we resynthesize the circuit of Fig. 2, but distribute the five simulations required to evaluate each circuit across three workstations, the 11-h sequential time drops to 192 min. This is a useful form of parallelism to exploit, but it is strictly limited.
3) **Parallel circuit search:** what we really seek is a technique to allow multiple, concurrent points of the cost landscape to be searched in parallel, but synchronized in some manner that guarantees convergence to a final circuit or set of circuits of similar quality.

Unfortunately, annealing *per se* does not easily support parallel search. An annealing-based optimizer generates a serial stream of proposed circuit perturbations, and relies on statistics from previous circuits to adjust its control parameters. To parallelize search itself, an obvious set of methods to consider here are the genetic [32], and evolutionary algorithms [33], [34], whose population-based evolution models distribute over parallel machines more naturally. We focus on a novel population-based strategy in Section II-D.

D. Global Optimization via Stochastic Pattern Search

We have developed two separate attacks on the global optimization problem. We initially focused on extending the annealing paradigm (which has empirically performed very well on the circuit synthesis task) to support an aggressively scalable form of parallelism. We employed an idea from Goldberg [35] called *parallel recombinative simulated annealing* (PRSA). PRSA, which has its roots in genetic algorithms, can be regarded as a strategy for synchronizing a population of annealers as they cooperatively search a cost surface. When generating a new circuit candidate, each annealer in the population makes one of two choices: perturb an element value in its previously generated solution, or recombine its previously generated solution with a solution obtained from another annealer in the population. Recombination is the *crossover* (mating) operator from genetic algorithms, which randomly combines features of two parent solutions into a single new offspring solution. The technique works extremely well to synchronize global search. In particular, good solutions found by one annealer quickly diffuse through the population, and drive annealers stuck in unpromising local minima toward better global solutions. Measured performance scaled essentially linearly out to 30 parallel UNIX CPU's, with Cadence Spectre [30] as the evaluation engine. Experimental results from our implementation of these ideas, called MAELSTROM, appear in [2].

Annealing is one member of a general class of optimization techniques for nondifferentiable cost functions. When cost surfaces are convex or nearly so, or we can assume we start optimization close to an acceptable final solution, gradient and sensitivity-based optimization techniques work well. However, these are generally unworkable in synthesis when we often have no feasible starting solution, many local minima, and gradients are either unreasonably expensive to compute (recall that we insist on full circuit-level simulation to evaluate solution candidate) or, more likely, numerically unreliable. In these cases, researchers have favored global optimization techniques based on sampling of the cost surface, with decisions for search based on the properties of previous samples. Annealing in particular has long been a favored approach here.

However, there are alternatives. An obvious class of methods are the so-called *direct-search* techniques, which sample the cost in a deterministic locus around a given solution point, and use this sample to construct a deterministic direction and distance to a conjectured better solution. As the optimization proceeds, the shape of the locus changes to reflect the success or failure of samples from previously visited regions of the cost surface. Ultimately, the locus converges to a single final locally optimal minimum. Coordinate search, Box search [36], Hook–Jeeves [37], and Nelder–Mead Simplex [38] are all variants of this idea. However, Torczon has recently suggested a unified formulation that renders many of these classical methods as particular cases of a more general method called *pattern search* [39], [40]. Surprisingly, Torczon was able to show pattern search to have provable convergence properties. The theoretical interest generated by Torczon's results motivates us to reconsider direct search ideas.

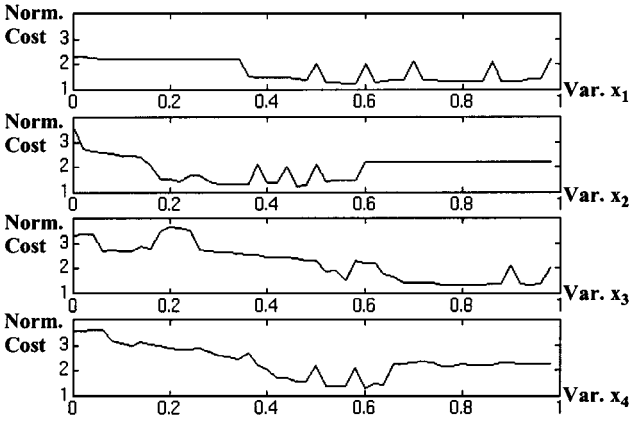


Fig. 3. Four one-dimensional normalized cross sections of the cost-surface for a typical simulation-based synthesis problem.

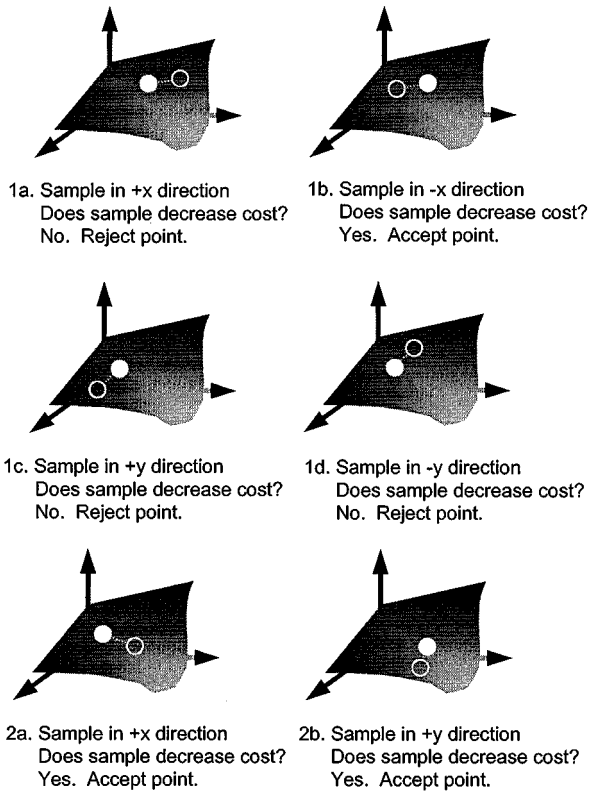


Fig. 4. Classical coordinate search algorithm.

Consider first a classical coordinate search algorithm, illustrated in Fig. 4, for a simple two-dimensional problem. Coordinate search visits in fixed order the individual coordinates x, y of the current solution, perturbs each in a deterministic pattern, and accepts only perturbations that decrease the cost. By making the starting perturbation suitably large, some local minima can be avoided: we "skip over" the hills, rather than accepting directly an uphill move, as in annealing. By shrinking the perturbation size appropriately over the course of the search, we can localize the search to converge on (we hope) a good local minimum.

Nelder–Mead Simplex has been tried in some circuit synthesis and optimization tools. Results to date have been negative: for very small problems with very simply cost surfaces

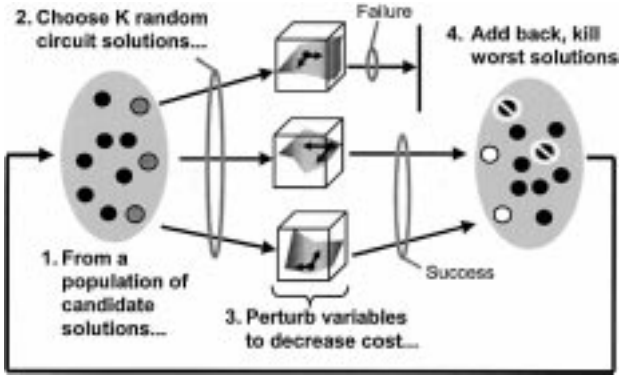


Fig. 5. A flow representation of the stochastic pattern search algorithm. Steps 1–4 are performed, and repeated until the population's cost can no longer be decreased. Step 3 corresponds to the subroutine decrease_cost() in the pseudocode of Fig. 7.

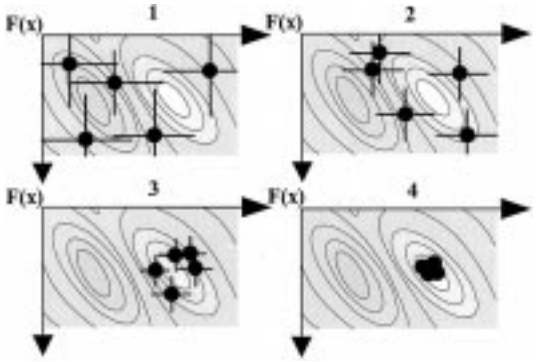


Fig. 6. Convergence to a cost-surface minimum during stochastic pattern search. The population model allows us to search multiple samples of the cost surface concurrently, all of which ultimately converge to solutions (circuits) of similar quality.

they perform adequately; large and difficult problems, they perform poorly. For example, in our preliminary comparisons to our own Nelder-Mead implementation, ANNEAL++ was always superior—although it obtained its solutions at the cost of considerably more CPU time.

To overcome these problems, we suggest coupling pattern search strategies with the population-of-circuits idea from MAELSTROM, and add a random component to the pattern search itself. The overall architecture is conceptually simple.

1) **Population of partial circuit solutions:** we maintain a large population of partial solutions. It is the population itself which combats the problem of cost surfaces with local minima. Each element is one sample of that cost surface. By maintaining a suitably diverse set of samples, and preferentially updating the population so that lower-cost samples survive and higher-cost samples are culled, we avoid the need to do explicit hill-climbing.
2) **Simple population update:** from a population of P partial solutions, we select k candidates, apply a short pattern search improvement process to each candidate, then replace these in the population. From this updated population of $P + k$ solutions, we remove the k solutions of highest (worst) cost.
3) **Evolution by random pattern search:** the update process for a candidate selected from the population is a truncated pattern search in the style of coordinate search, but with the key difference that the search pattern (direction and step size) are themselves randomized.

We evolve the population via short, randomized pattern searches, and allow only superior solutions to survive any update. The overall flow is illustrated in Fig. 5. The convergence of a population of cost samples to a set of solutions of similar numerical quality is illustrated in Fig. 6. Our intuition here is that pattern searches work well in the neighborhood of minima, but that the reliance on evolving only one solution, and doing so in a deterministic pattern, does not provide a sufficiently vigorous search. A population of solutions partially combats the problem of local minima. Evolving several elements of the population via short bursts of randomized downhill pattern search provides a suitably vigorous pressure to explore promising local minima. Favoring only improved solutions in the population coerces this search to converge to a set of solutions of similar quality, yet the combination of the population model and the randomized pattern searches helps the optimization swarm over a suitably diverse set of samples of the cost surface. This gives the technique its name: *stochastic pattern search*.

Torczon-style pattern search admits parallelism in the obvious form of concurrent exploration of different search patterns [41]. In our case, the population model allows us to perform multiple pattern searches in parallel. A detailed description of the stochastic pattern search algorithm in the form of pseudocode appears in Fig. 7. We begin with a simple coordinate search algorithm, which is shown in pseudocode beginning on line 25 of Fig. 7. The main loop of the subroutine *decrease_cost()* perturbs each independent variable (coordinate) of the current solution vector; we refer to the process of perturbing all coordinates as one atomic *pattern-search*. However, the order in which each variable is perturbed is random, and the perturbation amount is also random, although bounded. Each variable may be perturbed in both the positive and negative direction: the goal is to find a perturbation that decreases the cost. Perturbation bounds decrease as the algorithm progresses, shrinking around the evolving solution. One nonrandom component is the acceptance criterion for each coordinate perturbation: we only accept solutions that decrease the cost. Unlike annealing, we do not accept uphill moves; however, because coordinate search can take very large steps, we can "skip over" hills.

The algorithm uses a master-slave organization for managing the population. The code for each parallel node begins on line 1 of Fig. 7. We start with a population of P circuit solutions, sorted on cost. We randomly select a subset of k of these solutions, with some bias toward those of lower cost. We perform one pattern-search on each of these k candidates, add them back to the population, re-sort, and then cull out the worst k solutions. This repeats until no more improvements can be found in any elements of the population, i.e., all circuits have essentially similar cost. We can distribute the pattern-search for each of the k candidates in parallel, and each individual coordinate search itself requires t multiple circuit simulations, allowing us to easily exploit kt parallel workstations. So, as noted earlier, we can distribute both search and evaluation.

```
1   if this node is the master
2       spawn m slaves
3       initialize the population, P, with n random points
4       while the population cost is still decreasing
5           foreach slave, s
6               choose a point, p, from the population, P
7               send p to s
8           end
9           wait for a single point from each slave, m total points
10          foreach point, p, received from the slaves
11              if p decreases the cost of P
12                  insert p into P
13                  remove the highest cost point from P
14              end
15          end
16      end
17  else this node is a slave
18      wait for a point, p, from the master
19      p_new = decrease_cost(p)
20      send p_new to the master
21      adjust step size bounds
22  end
23
24  subroutine decrease_cost(p)
25      p_proposed = p
26      min_cost = cost of p
27      while there is a coordinate of p_proposed that has not been visited
28          choose a random index, i, that has not been chosen yet
29          p_proposed[i] += random_petrurbation
30          if the cost of p_proposed is less than min_cost
31              min_cost = the cost of p_proposed
32          else
33              p_proposed[i] = p[i]
34          end
35      end
36      if min_cost < the cost of p
37          return proposed_p
38      else
39          return p
40      end
41  end subroutine
```

Fig. 7. Pseudocode for parallel stochastic pattern search.

E. Network Architecture

Our implementation distributes all computation over a pool of workstations. We use software components and organizational ideas from MAELSTROM, which at the lowest level manages concurrency and interprocessor communication using the publicly available PVM library [42]. The overall network architecture is illustrated in Fig. 8. The important point to make is that the master-slave structure implied by the stochastic pattern search pseudocode of Fig. 7 is only a *logical* organization for the required parallel computation, not a *physical* organization. In other words, we do not bind the master and individual slave nodes each to a separate physical CPU. In our terminology, the process of selecting and updating k candidates in the population is handled by k *slaves*. But each slave can use as many physical CPU's as necessary. Indeed, the actual mechanics of managing the population and updating the required state information consume a negligible fraction of the overall execution time, which is wholly dominated by the circuit simulations. The two physical characteristics of the method that we must deal with are as follows.

1) Support for an arbitrary number of available workstations. We should be able to use as many machines as we have available, or from which we can harvest spare cycles. But, we must be able to make progress on synthesis even if we have fewer machines than we would prefer.
2) Support for intelligent scheduling of the simulation tasks among available workstations. Different simulation tasks (dc, ac, transient, test-jig setup, early parameter estima-

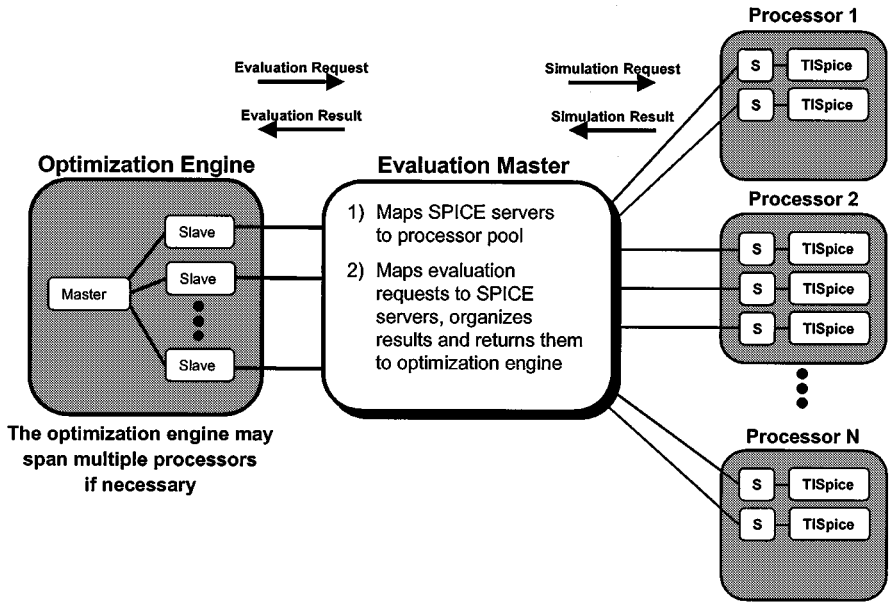


Fig. 8. ANACONDA network architecture.

tion, etc.) can each require vastly different amounts of time. Assuming that each simulation needs its own CPU is not only impractical, but inefficient. Our progress would always be limited by the slowest simulation.

The architecture in Fig. 8 addresses these concerns via a separate layer of software called the *evaluation master*. It brokers requests for simulations (from concurrent pattern searches on elements of the evolving population) among a set of available CPU's (slaves) that perform actual simulations. Any individual CPU may be performing several simulations in series. The evaluation master dynamically schedules simulation tasks to maximize the throughput on the available machines, using simple bin-packing heuristics. Thus, one CPU might be tasked to run several small simulations, each of which is expected to complete quickly, while another CPU might run only one simulation, which is expected to be long. The evaluation master tracks completion times for each simulation task on each node, and uses this information to periodically reschedule all tasks. Simulation times can vary over the course of a simulation both because of transient changes in machine loading, and because circuit candidates migrate around in the solution space, and not all solutions are equivalently easy to simulate. Dynamic scheduling makes for efficient use of the available computational resources.

At this point, we can offer a simple experiment that shows the merits of our approach. We revisited the simulator-in-serial-annealer approach used in Fig. 2, and compared it to stochastic pattern search when applied to the task of synthesizing the industrial circuit (to be described in Section IV) of Fig. 10(c). Twenty synthesis runs were performed using each algorithm, and the results are shown in Fig. 9. We show a scatter plot of cost [the OBLX-form cost-function being minimized, from (3)] versus "perturbations." For the annealer, this is the number of circuit candidates visited sequentially. For stochastic pattern search, each perturbation actually visits k new circuits, and we evaluate these in parallel across a pool of ten workstations. For this result, the parallel pattern search algorithm actually visited

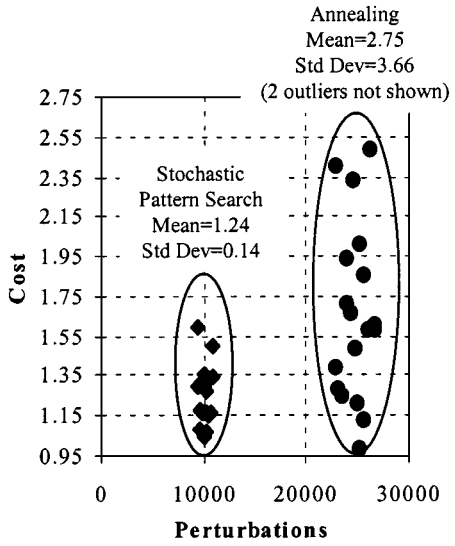


Fig. 9. Parallel stochastic pattern search compared to a well-tuned serial annealer for synthesis of the circuit in Fig. 10(c). The x axis is the number of serial perturbations. The total number of design points evaluated by the pattern search algorithm was $10\times$ the number of serial perturbations, or approximately 100 000. The total number of design points evaluated by the annealer is 25 000 because it is a serial annealer.

roughly 100 000 circuits, but with a CPU time proportional to only 10 000 circuit simulations.

Note that stochastic pattern search is producing both a better average answer (i.e., finding a better expected minimum, which corresponds to a better circuit solution), and a tighter spread of answers. We believe this is a result of our parallel population model, which not only allows us to visit more solutions in a reasonable amount of time, but also does a better job of pruning weak solution candidates before search becomes trapped in poor local minima. This is critical since any stochastic optimizer produces a spread of solutions on repeated runs; tighter spreads mean a higher likelihood of finding a good solution on each invocation of the tool.

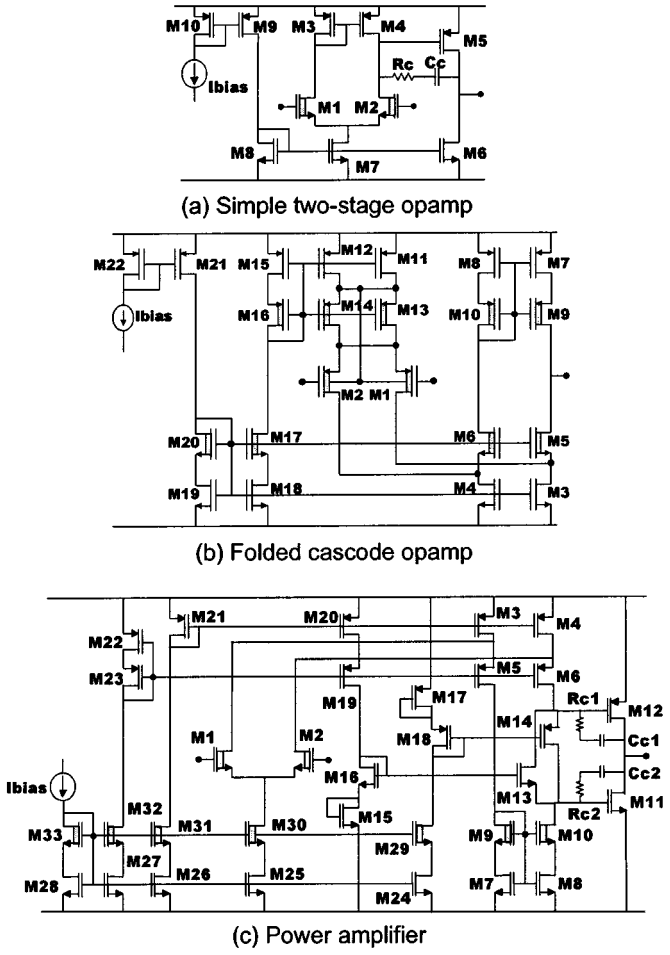


Fig. 10. Industrial test circuits for our synthesis experiments.

IV. Results

The ideas presented in this paper have been implemented in a tool called ANACONDA and tested on site at Texas Instruments in Dallas, TX. We benchmarked ANACONDA on the three opamps, shown in Fig. 10, that are all examples of production circuits for which we also have complete TI manual designs. The power amp was designed in a 0.8-μm CMOS process and the other two opamps were designed in a 0.6-μm CMOS process.

ANACONDA consists of a parallel stochastic pattern search algorithm coupled (via encapsulation) to an industrial circuit simulator that evaluates each candidate solution. The simulator used was TI's proprietary TISpice. In addition, all three circuits were synthesized using the same parameters for the pattern search algorithm: the population size was set to 200, and ten logical slaves (i.e., we select and replace ten elements of the population concurrently) were used to evaluate circuits in the population. Simulation tasks were dynamically rescheduled every 100 simulations. Also, the same weightings in the cost function from (3) were used for all three circuits. The constraints for each design had a weighting of one while the objectives had a weighting of 0.5.

Figs. 12 and 13 offer some insight into the population dynamics as these sets of benchmark circuits evolved. Fig. 12 plots the mean cost of each population of 200 circuits for each of the five synthesis runs, for each of the three benchmarks. An immediate and striking feature of the data is how closely the individual populations track each other, even across separate synthesis runs. We take this as indication that the population update/replacement strategies are performing well: they are maintaining an appropriately diverse sample of the cost surface, while encouraging convergence to solutions of similar cost by culling always the weakest solutions. Fig. 13 plots the maximum, mean, and minimum cost samples seen in each population for one sample run for each of the three circuits. As expected, there is significant disparity between the minimum and maximum values at the start of synthesis, but these rapidly converge as weaker solutions are culled and the solution candidates cluster around a single best value. Perhaps more interesting is the qualitative difference between the dynamics of the smaller two-stage and folded-cascode circuits, which show somewhat larger final spreads in contrast to the larger, more difficult power amp circuit. We interpret this as related to the size of the design—the number of degrees of freedom (DOF). The power amp is a large design; we believe there are simply more nearby solutions of equivalent quality than for the other smaller designs. For the smaller designs, there simply appear to be fewer good final solutions that are unique, hence the final population retains a larger fraction of weaker designs. A contributing factor here is also that, in industrial practice, device sizes are not continuous, but are restricted to some discrete (though closely spaced) values. This discretization affects the smaller designs somewhat more noticeably. Finally, it is worth noting that the cost scale is logarithmic: even though the spreads are visible in the plots, in reality the costs are extremely close. Table I gives the input performance constraints, simulation environment, and runtime statistics for each synthesized circuit. For our experiments, the performance constraints were set by running nominal simulations on the original hand designs. Each design was synthesized using a pool of 16 300-MHz Ultra 10's and four 300-MHz dual-processor Ultra 2's. Note that although we only update ten circuits in the population in parallel, each circuit required between five and seven simulations to evaluate. Thus, we can make use of the entire pool of 24 CPU's. Because runtime is highly dependent on the size of the circuit and the type of simulation being performed, there was significant variation among the circuits. For example, for the power amp, each total harmonic distortion (THD) analysis took a few CPU seconds and, consequently, that circuit had significantly longer synthesis times. Overall, each synthesis task required between a few hours and overnight on a pool of (typically) 20 available workstations.

Fig. 11 shows the results of five consecutive synthesis runs for each circuit. Given the large number of performance specifications for each circuit, we summarize these results as power-versus-area scatter plots. For each design, the objectives were to minimize area and static power dissipation. In all but one case, the synthesized circuits met *all* of the performance constraints specified in Table I. Several of these designs are in fact superior to their manual counterparts. And, it is worth noting again

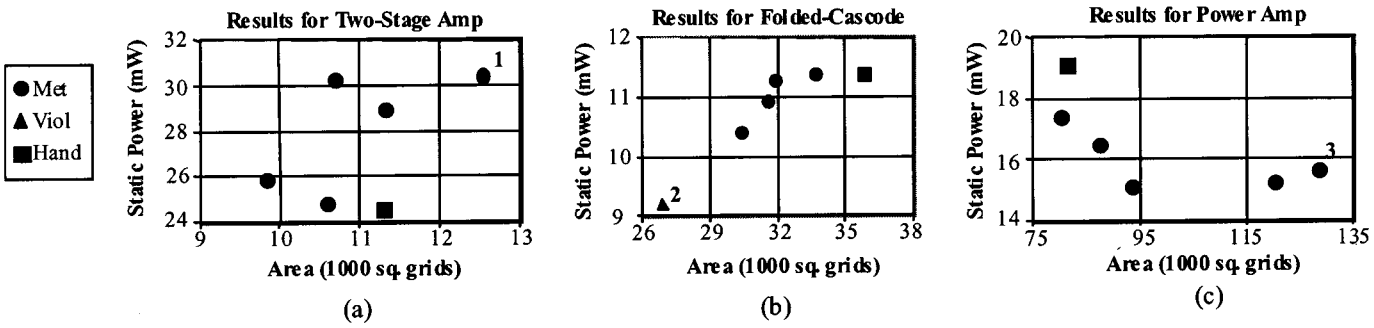


Fig. 11. Static power versus area scatter plots for five consecutive synthesis runs for each test circuit. Each scatter includes a manual circuit design represented by a square point. Designs such as "1" in result (a) that have significantly higher static power dissipation typically are over-designed for some specification; for result "1" the settling time was slightly smaller than necessary. Design "2" in (b) had a UGF of 159 MHz, and this was slightly less than the 162 MHz required. Designs such as "3" in (c) that have significantly larger area again represent over-design in a performance constraint; for result "3," the THD was 15% smaller than necessary.

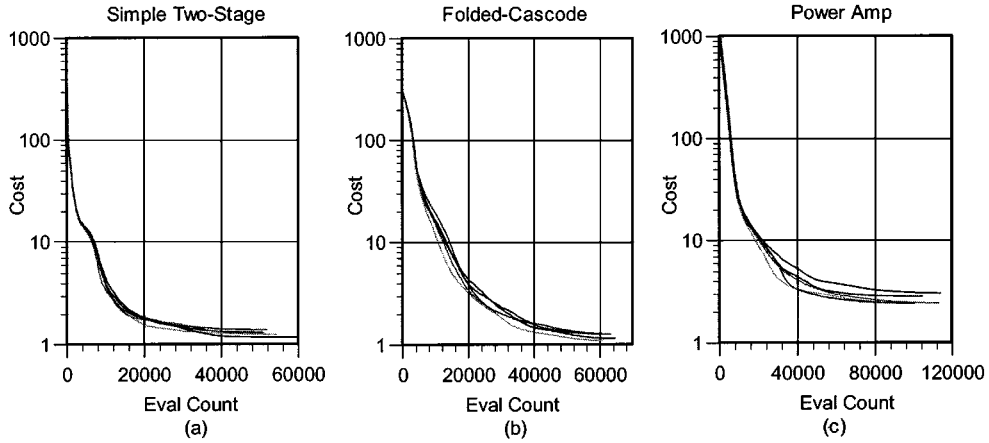


Fig. 12. Evolution of the population's mean cost over five consecutive synthesis runs, for the synthesis results shown in Fig. 11. The x axis is the total number of design points that have been evaluated.

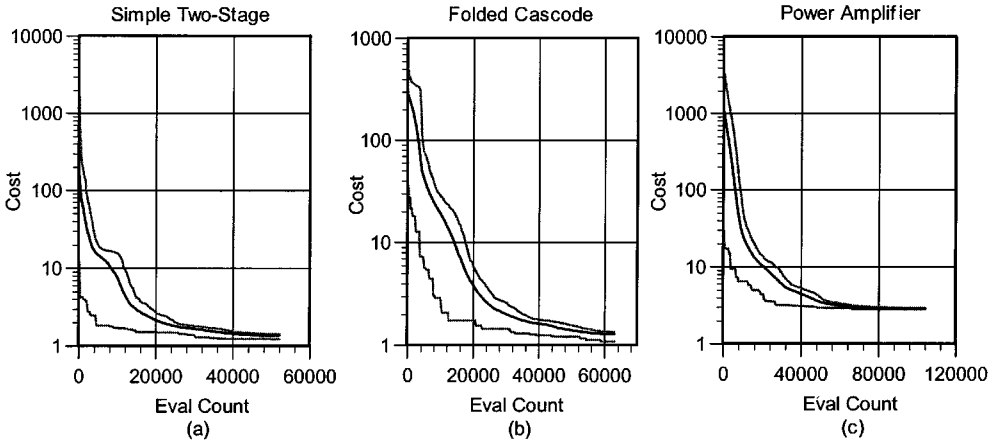


Fig. 13. A sample synthesis run for each test circuit from Fig. 10, illustrating how the minimum, mean and maximum cost of candidates in the population evolve over time. The x axis is the total number of evaluated design points.

that the quality metrics—meeting specifications under detailed circuit simulation—are *identical* to those used during manual design.

Our results so far suggest that ANACONDA is effective for nominal cell-level analog synthesis. But we must also make some efforts to address manufacturing process variations and environmental operating range constraints. We know from experience that we cannot ignore these issues, since a well-performing synthesis algorithm can push a final design very close to the edge of the feasible region where all constraints are met. Even modest changes in process or operating conditions can then render the circuit nonfunctional [43]. We have two broad classes of solutions: add first-order constraints to the synthesis task, mimicking the "conservative" design practices of skilled designers, or fold a numerical manufacturability optimization into the synthesis process itself [44]. We choose the former

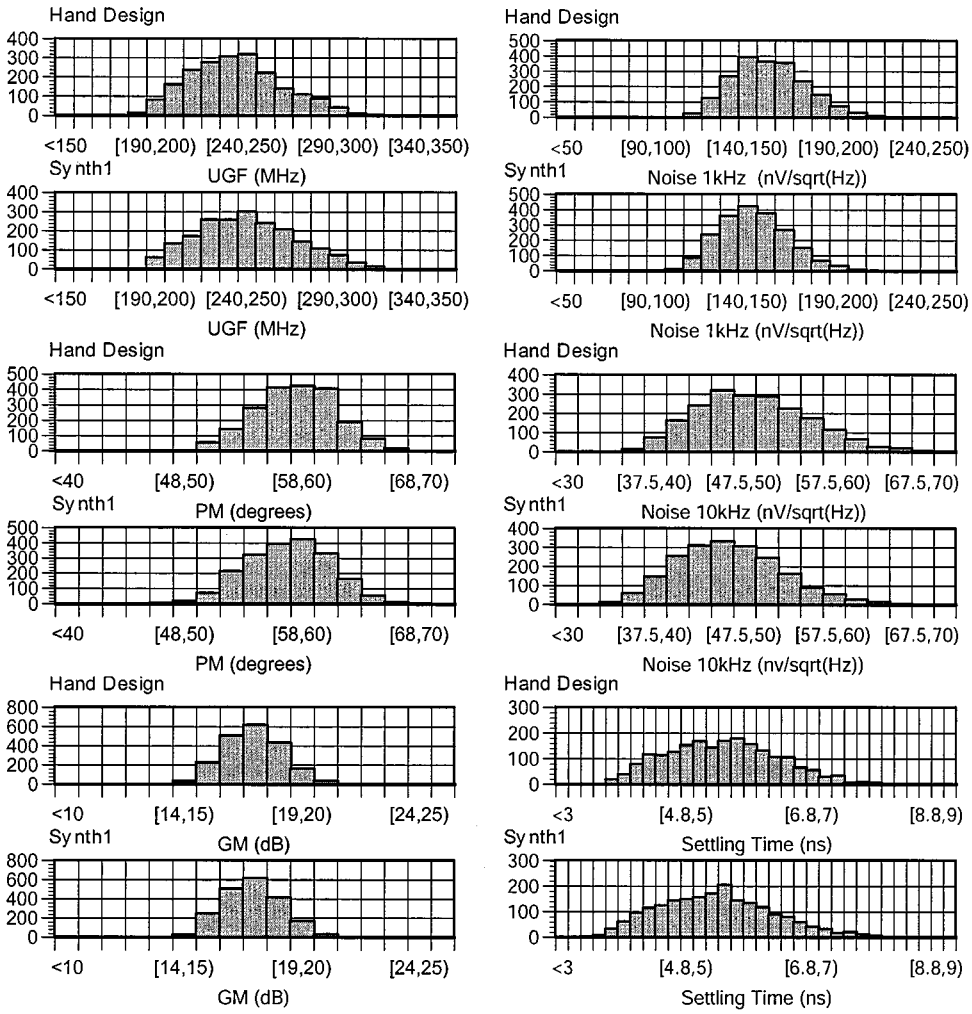


Fig. 14. A sample of the monte-carlo results for the simple two-stage opamp. Histograms for the performance specs are shown first for the hand design and second for the synthesized design.

option here, and focus on how to properly constrain a circuit so that synthesis yields a result with design centering comparable to an expert manual design. In ANACONDA, the two types of constraints supported for this purpose are *operating region constraints* and *component constraints*. These adapt ideas from ASTRX/OBLX.

Operating region constraints simply specify that individual devices should be "far enough" into the desired region that process or environmental variations cannot force the device out of this region. If operating region constraints are not used, the optimizer may choose to bias a transistor on the edge of the active region for nominal performance gains. To avoid such a situation, we can designate how large we would like the *effective voltage*, i.e., how far above V_T we require V_{GS} to be for a MOSFET, for individual devices in the design. For example, a typical number when designing CMOS opamps in micron-scale processes is simply to fix this effective voltage to 250 mV; we do not expect individual devices each to have precisely tailored, unique voltage constraints. When designing by hand the value for the effective voltage is set using first order equations. Unfortunately, the simulated value may be significantly different than the hand calculated value. One advantage of using simulator-in-the-loop synthesis is that when the constraint is met one can be assured that the effective voltage will be the value specified.

Designers can eliminate unnecessary or possibly dangerous DOF by specifying *component constraints*. As an example, consider the simple two-stage opamp in Fig. 10(a). Clearly, a reasonable design would have the input differential pair matched. This could be specified using the simple constraints $W1 = W2$ and $L1 = L2$. Such constraints are trivial to accommodate. But in real designs, many component values are determined parametrically, as functions of other designables in the circuit. For example, it is common practice to set the compensation resistor R_C in the circuit of Fig. 10(a) to the value

$$R_C = \frac{1+\delta}{gm_5} \tag{4}$$

where δ is a designer-input constant reflecting the degree of overdesign desired, and gm_5, the transconductance of another device in the circuit, requires a *separate* SPICE simulation to compute. Indeed, it is not uncommon to require a series of these "setup" simulations to first create the proper component relationships *before* real evaluation of the circuit can begin. Sim-

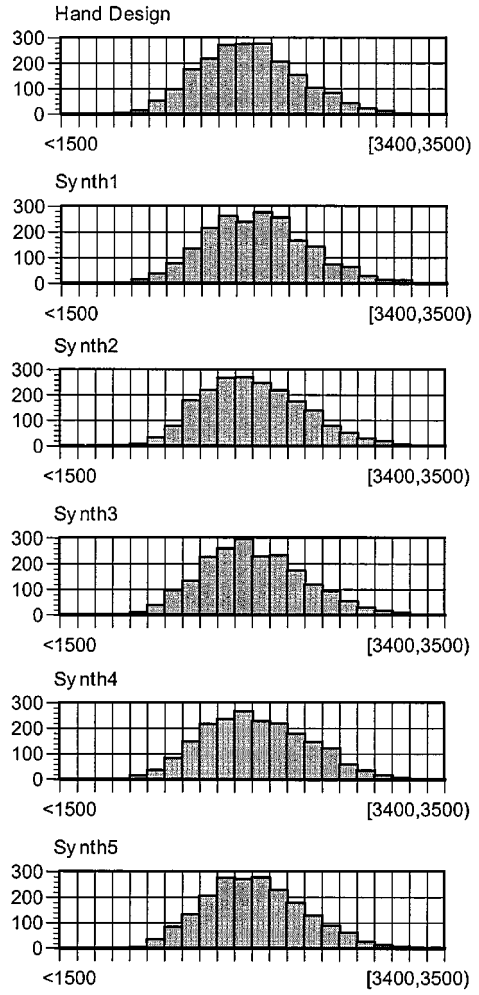


Fig. 15. Histograms illustrating how the two-stage opamp's low-frequency gain (x axis) varies across process variations. Results from the hand design are shown first, followed by results for the five consecutive synthesis runs. These histograms illustrate the consistent quality across consecutive synthesis runs.

TABLE I
DETAILED PERFORMANCE SPECIFICATIONS, SIMULATION ENVIRONMENT, AND SYNTHESIS RUNTIME RESULTS FOR THE THREE TEST CIRCUITS OF FIG. 10

Environment Spec		*Two-Stage*	*Folded-Cascode*	*Power Amp*
T=25°C				
Vdd	V	2.7	2.7	5
C_L	pF	1.5	1.75	100
R_L	Ω			25
Performance Constraint				
Gain	dB	≥ 68	≥ 71	≥ 92
UGF	MHz	≥ 260	≥ 162	≥ 0.6
PM	deg.	≥ 56	≥ 52	≥ 84
GM	dB	≥ 17		
Noise 1kHz [1]	nV $Hz^{-0.5}$	≤ 145	≤ 70	≤ 52
Noise 10kHz	nV $Hz^{-0.5}$			≤ 40
Noise 10MHz	nV $Hz^{-0.5}$	≤ 3	≤ 4	
CMRR	dB	≥ 76	≥ 108	> 138
PSRR (Vss)	dB	≥ 85	≥ 89	≥ 90
PSRR (Vdd)	dB	≥ 70	≥ 72	≥ 94
Settling Time	ns	≤ 4.6 [4]	≤ 16 [5]	
THD [2]	%			≤ 0.06
THD [3]	%			≤ 0.1
Runtime Info				
Independent Variable Count		15	13	20
Approx. Evaluation Count [6]		57,000	65,000	102,000
Ave. Runtime [7] (hrs)		2.8	1.8	10

1 – All noise specs are input referred and as measured in an ac analysis
2 – 4.0V p-p 1kHz input
3 – 2.6V p-p 1kHz input, R_L=5Ω
4 - 0.5V input step, 10-Bit accurate final voltage
5 - 0.1V input step, 10-Bit accurate final voltage
6 – Number of circuit evaluated in series is approx the evaluation count/10
7 – Using a pool of 16 300MHz SUN Ultra10 and 4 dual 300MHz Ultra2 workstations running Solaris 2.5.1.

ulation-based synthesis algorithms like ANACONDA (and also MAELSTROM) extend gracefully to handle these sorts of practical usage scenarios.

The circuits from Fig. 10 and results from Fig. 11 were obtained using these common-practice, first-order constraints. To assess the robustness of the approach, each of the synthesized designs was verified by performing Monte Carlo simulations with 3σ process, 10% voltage supply, and 0 °C to 100 °C temperature variations. Selected results are illustrated in Figs. 14–16. Fig. 14 shows histograms for most of the simple two-stage opamp's performance specifications. A single synthesis result, synthesis run 1, is compared to the hand design. From the histograms we see that the synthesized design performs as well as the hand design across process and environment variations for each of the performance specifications. Figs. 15–17 shows sample histograms for the low-frequency gain of the three test designs. Notice that all five consecutive synthesis runs are shown. This illustrates how the robustness of the circuits varies from one synthesis run to the next. For the simple two-stage and folded cascode opamps there is very little variation. However, for the power amplifier, Fig. 17, we see that four of the five synthesized designs actually have tighter spreads than the hand design. The main reason for these overall tight spreads is the rigorous enforcement of the operating region and component constraints. These sorts of constraints are a standard part of good manual design practice—but not always enforced with the discipline we can achieve in a numerical synthesis tool.

These results are noteworthy in several respects. First, these are production-quality industrial analog cells with difficult performance specifications. Second, our synthesis approach is using as its evaluation engine the *identical* simulation environment used by TI's designers to validate their manual designs. As a result, we can deal accurately with difficult design specification such as noise, settling time, and THD, which require detailed simulations to evaluate complex nonlinear effects. Finally, nearly all of these synthesized designs compare favorably with their manually designed counterparts, both in performance and in robustness across manufacturing and environmental corners. We believe this is a significant advance in demonstrating how an analog synthesis tool can attack realistic industrial circuit designs.

V. CONCLUSION

We have presented a novel synthesis strategy for custom analog cells. Our central contribution is stochastic pattern search, a parallelizable global optimization algorithm that

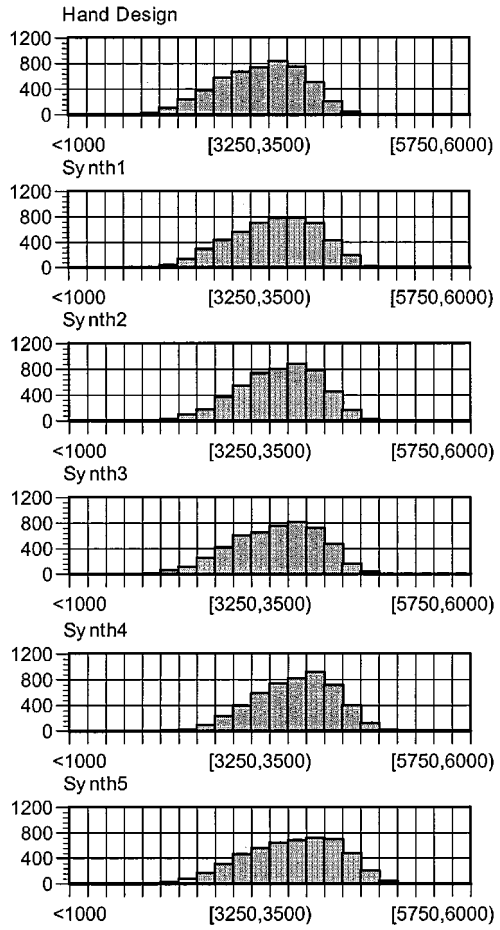


Fig. 16. Histograms illustrating how the folded cascode opamp's low-frequency gain (x axis) varies across process variations. Results from the hand design are shown first, followed by results for the five consecutive synthesis runs. These histograms illustrate the consistent quality across consecutive synthesis runs.

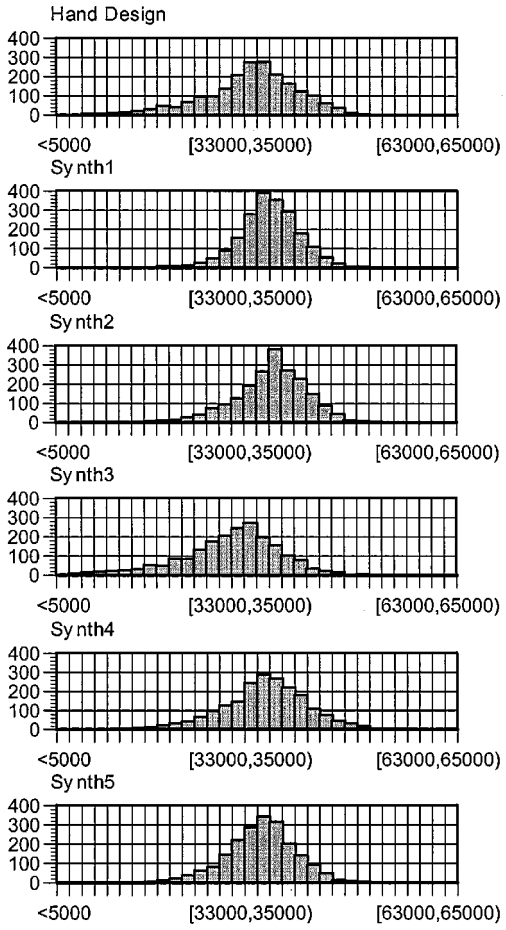


Fig. 17. Histograms illustrating how the power amp's low-frequency gain (x axis) varies across process variations. Results from the hand design are shown first, followed by results for the five consecutive synthesis runs. These histograms illustrate the consistent quality across consecutive synthesis runs.

combines ideas from evolutionary algorithms and numerical pattern search. By encapsulating commercial circuit simulators and distributing the search across a pool of workstations, we can visit 50 000–100 000 circuit candidates, and fully simulate each, in a few hours. ANACONDA, an implementation of these ideas, has successfully synthesized several difficult industrial cells. We believe analog synthesis is a necessary component of any strategy for reusing and retargeting analog circuits. We are currently working to apply ANACONDA to much larger designs, in particular, system-level designs. Other current work focusses on comparing the relative merits of the genetic-annealing approach used in MAELSTROM with the population-pattern approach used in ANACONDA. In addition, the practical ability to synthesize analog circuits raises a host of questions about analog intellectual property—how it should be archived, what information model is necessary for practical reuse, how much effort is required to "package" a cell for future use, etc. We are currently evolving the MAELSTROM framework to address these issues.

ACKNOWLEDGMENT

The authors would like to thank E. Ochotta (Xilinx) for discussions about OBLX, B. Bearden (TI) for help with TISpice and G. Richey and F. James (TI) for valuable discussions about this work.

REFERENCES

[1] E. Ochotta, R. A. Rutenbar, and L. R. Carley, "Synthesis of high-performance analog circuits and ASTRX/OBLX," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 237–294, Mar. 1996.

[2] M. Krasnicki, R. Phelps, R. A. Rutenbar, and L. R. Carley, "MAELSTROM: Efficient simulation-based synthesis for analog cells," in *Proc. ACM/IEEE Design Automation Conf.*, June 1999, pp. 945–950.

[3] R. Phelps, M. Krasnicki, R. A. Rutenbar, and L. R. Carley, "ANACONDA: Robust synthesis of analog circuits via stochastic pattern search," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1999, pp. 567–570.

[4] E. Ochotta, T. Mukherjee, R. A. Rutenbar, and L. R. Carley, *Practical Synthesis of High-Performance Analog Circuits*. Norwell, MA: Kluwer Academic, 1998.

[5] M. Degrauwe *et al.*, "Toward an analog system design environment," *IEEE J. Solid-State Circuits*, no. 3, p. 24, June 1989.

[6] H. Y. Koh, C. H. Sequin, and P. R. Gray, "OPASYN: A compiler for MOS operational amplifiers," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 113–125, Feb. 1990.

[7] G. Gielen *et al.*, "Analog circuit design optimization based on symbolic simulation and simulated annealing," *IEEE J. Solid-State Circuits*, vol. 25, pp. 707–713, June 1990.

[8] F. Leyn, W. Daems, G. Gielen, and W. Sansen, "A behavioral signal path modeling methodology for qualitative insight in and efficient sizing of CMOS opamps," in *Proc. ACM/IEEE ICCAD*, 1997, pp. 374–381.

[9] P. C. Maulik, L. R. Carley, and R. A. Rutenbar, "Integer programming based topology selection of cell level analog circuits," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 401–412, Apr. 1995.
[10] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS Opamp synthesis by means of a genetic algorithm," in *Proc. 32nd ACM/IEEE DAC*, 1995, pp. 433–438.
[11] M. Hershenson, S. Boyd, and T. Lee, "GPCAD: A tool for CMOS Op-Amp synthesis," in *Proc. ACM/IEEE ICCAD*, Nov. 1998, pp. 296–303.
[12] R. Harjani, R. A. Rutenbar, and L. R. Carley, "OASYS: A framework for analog circuit synthesis," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1247–1266, Dec. 1989.
[13] B. J. Sheu *et al.*, "A knowledge-based approach to analog IC design," *IEEE Trans. Circuits and Systems*, vol. 35, no. 2, pp. 256–258, 1988.
[14] E. Berkcan *et al.*, "Analog compilation based on successive decompositions," in *Proc. 25th IEEE DAC*, 1988, pp. 369–375.
[15] J. P. Harvey *et al.*, "STAIC: An interactive framework for synthesizing CMOS and BiCMOS analog circuits," *IEEE Trans. Computer-Aided Design*, pp. 1402–1417, Nov. 1992.
[16] C. Makris and C. Toumazou, "Analog IC design automation part II—Automated circuit correction by qualitative reasoning," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 239–254, Feb. 1995.
[17] A. Torralba, J. Chavez, and L. Franquelo, "FASY: A fuzzy-logic based tool for analog synthesis," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 705–715, July 1996.
[18] G. Gielen, P. Wambacq, and W. Sansen, "Symbolic analysis methods and applications for analog circuits: A tutorial overview," *Proc. IEEE*, vol. 82, pp. 287–304, Feb. 1990.
[19] C. J. Shi and X. Tan, "Symbolic analysis of large analog circuits with determinant decision diagrams," in *Proc. ACM/IEEE ICCAD*, 1997, pp. 366–373.
[20] Q. Yu and C. Sechen, "A unified approach to the approximate symbolic analysis of large analog integrated circuits," *IEEE Trans. Circuits and Sys.*, vol. 43, pp. 656–669, Aug. 1996.
[21] W. Daems, G. Gielen, and W. Sansen, "Circuit complexity reduction for symbolic analysis of analog integrated circuits," in *Proc. ACM/IEEE Design Automation Conf.*, June 1999, pp. 958–963.
[22] P. Wambacq, J. Vanthienen, G. Gielen, and W. Sansen, "A design tool for weakly nonlinear analog integrated circuits with multiple inputs (mixers, multipliers)," in *Proc. IEEE CICC*, San Diego, CA, May 1991, pp. 5.1.1–5.1.4.
[23] F. Medeiro, F. V. Fernandez, R. Dominguez-Castro, and A. Rodriguez-Vasquez, "A statistical optimization based approach for automated sizing of analog cells," in *Proc. ACM/IEEE ICCAD*, 1994, pp. 594–597.
[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 45–54, May 1983 .
[25] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 352–356, Apr. 1990.
[26] W. Nye *et al.*, "DELIGHT.SPICE: An optimization-based system for the design of integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. 7, Apr. 1988.
[27] K. Saab, D. Marche, N. N. Hamida, and B. Kaminska, "LIMSoft: Automated tool for sensitivity analysis and tesst vector generation," *Inst. Elect. Eng Proc. Circuits Devices Systems*, vol. 143, no. 6, pp. 386–392, Dec. 1996.
[28] A. R. Conn, R. A. Haud, C. Viswesvariah, and C. W. Wu, "Circuit optimization via adjoint lagrangians," in *Proc. ACM/IEEE ICCAD*, Nov. 1997, pp. 281–288.
[29] M. Krasnicki, "Generalized analog circuit synthesis," masters thesis, Dept. of ECE, Carnegie Mellon, Pittsburgh, PA, Dec. 1997.
[30] K. S. Kundert, *The Designer's Guide to SPICE & SPECTRE*. Norwell, MA: Kluwer Academic, 1995.
[31] K. Nakamura and L. R. Carley, "A current-based positive-feedback technique for efficient cascode bootstrapping," in *Proc. VLSI Circuits Symp.*, June 1991, pp. 107–108.
[32] J. H. Holland, *Adaptation in Nature and Artificial Systems*. Ann Arbor: Univ. Michigan Press, 1975.
[33] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Trans. Neural Networks*, vol. 5, pp. 3–14, Jan. 1994.
[34] ——, *Evolutionary Computation: The Fossil Record*, D. B. Fogel, Ed. Piscataway, NJ: IEEE Press, 1998.
[35] S. W. Mahfoud and D. E. Goldberg, "Parallel recombinative simulated annealing: A genetic algorithm," *Parallel Computing*, vol. 21, pp. 1–28, 1995.
[36] G. E. P. Box, "Evolutionary operation: A method for increasing industrial productivity," *Appl. Statist.*, vol. 6, pp. 81–101, 1957.
[37] R. Hooke and T. A. Jeeves, "'Direct search' solution of numerical and statistical problems," *ACM JACM*, vol. 8, pp. 219–229, 1961.
[38] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Comput. J.*, vol. 7, pp. 308–313, 1965.
[39] V. Torczon, "On the convergence of the multidirectional search algorithm," *SIAM J. Optimization*, vol. 1, no. 1, Feb. 1991.
[40] ——, "On the convergence of pattern search algorithms," *SIAM J. Optimization*, vol. 7, no. 1, Feb. 1997.
[41] ——, "PDS: Direct search methods for unconstrained optimization on either sequential or parallel machines," Dept. of Mathematical Sciences, Rice Univ., Houston, TX, Tech. Report 92-9, 1992.
[42] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Network Parallel Computing*. Cambridge, MA: MIT Press, 1994.
[43] T. Mukherjee, L. R. Carley, and R. A. Rutenbar, "Synthesis of manufacturable analog circuits," in *Proc. ACM/IEEE ICCAD*, 1994, pp. 586–593.
[44] G. Debyser and G. Gielen, "Efficient analog circuit synthesis with simultaneous yield and robustness optimization," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 1998, pp. 308–311.

Rodney Phelps (S'96) received the B.S.E.E degree with university honors from Carnegie Mellon University, Pittsburgh, PA, in 1994. In 1996, he returned to Carnegie Mellon University where he is currently completing the Ph.D. degree.

From 1994–1996, he worked at Cirrus Logic, Fremont, CA, in the corporate EDA group. He worked at Texas Instruments, Dallas, TX, from June 1998 to August 1999, in the Mixed-Signal EDA group where he focused on synthesis algorithms for analog circuits and systems. In January 2000, he joined Neo Linear, Pittsburgh, PA.

Mr. Phelps is a member of Eta Kappa Nu and Tau Beta Pi.

Michael Krasnicki received the B. S. degree with highest distinction from the University of Virginia, Charlottesville, in 1995. He received the M. S. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1995. He is currently working toward the Ph. D. degree at Carnegie Mellon University.

He is a Semiconductor Research Corporation (SRC) Fellow. He currently works at Texas Instruments, Dallas, TX. His research focus is performance-driven synthesis of analog cells with the use of industrial quality simulation.

As an undergraduate, Mr. Krasnicki received the William L. Everett Student Award for Excellence from the Department of Electrical Engineering.

Rob A. Rutenbar (S'77–M'84–SM'90–F'98) received the Ph.D. degree from the University of Michigan, Ann Arbor, in 1984.

He joined the faculty of Carnegie Mellon University (CMU), Pittsburgh, PA, where he is currently Professor of Electrical and Computer Engineering, and (by courtesy) of Computer Science. From 1993 to 1998, he was Director of the CMU Center for Electronic Design Automation. He is a cofounder of NeoLinear, Inc., and served as its Chief Technologist on a 1998 leave from CMU. His research interests focus on circuit and layout synthesis algorithms for mixed-signal ASIC's, for high-speed digital systems, and for FPGA's.

In 1987, Dr. Rutenbar received a Presidential Young Investigator Award from the National Science Foundation. He has won Best/Distinguished paper awards from the Design Automation Conference (1987) and the International Conference on CAD (1991). He has been on the program committees for the IEEE International Conference on CAD, the ACM/IEEE Design Automation Conference, the ACM International Symposium on FPGA's, and the ACM International Symposium on Physical Design. He also served on the Editorial Board of IEEE SPECTRUM. He was General Chair of the 1996 ICCAD. He chaired the Analog Technical Advisory Board for Cadence Design Systems from 1992 through 1996. He is a member of the ACM and Eta Kappa Nu.

L. Richard Carley received the S.B. degree in 1976, the M.S. degree in 1978, and the Ph.D. degree in 1984, all from the Massachusetts Institute of Technology (MIT), Cambridge.

He joined Carnegie Mellon University, Pittsburgh, PA, in 1984. In 1992, he was promoted to Full Professor of Electrical and Computer Engineering. He was the Associate Director for Electronic Subsystems for the Data Storage Systems Center [a National Science Foundation (NSF) Engineering Research Center at CMU] from 1990–1999. He has worked for MIT's Lincoln Laboratories and has acted as a Consultant in the areas of analog and mixed analog/digital circuit design, analog circuit design automation, and signal processing for data storage for numerous companies; e.g., Hughes, Analog Devices, Texas Instruments, Northrop Grumman, Cadence, Sony, Fairchild, Teradyne, Ampex, Quantum, Seagate, and Maxtor. He was the principal Circuit Design Methodology Architect of the CMU ACA-CIA analog CAD tool suite, one of the first top-to-bottom tool flows aimed specifically at design of analog and mixed-signal IC's. He was a co-founder of NeoLinear, a Pittsburgh, PA-based analog design automation tool provider; and, he is currently their Chief Analog Designer. He holds ten patents. He has authored or co-authored over 120 technical papers, and authored or co-authored over 20 books and/or book chapters.

Dr. Carley was awarded the Guillemin Prize for best Undergraduate Thesis from the Electrical Engineering Department, MIT. He has won several awards, the most noteworthy of which is the Presidential Young Investigator Award from the NSF in 1985 He won a Best Technical Paper Award at the 1987 Design Automation Conference (DAC). This DAC paper on automating analog circuit design was also selected for inclusion in 25 years of Electronic Design Automation: A Compendium of Papers from the Design Automation Conference, a special volume, published in June of 1988, including the 77 papers (out of over 1600) deemed most influential over the first 25 years of the Design Automation Conference.

James R. Hellums (S'75–M'77–SM'96) received the B.S.E.E. (highest honors) and M.S.E.E. degrees from the University of Texas at Arlington in 1976 and 1983, respectively. He is working toward the Ph.D. degree at University of Texas at Dallas with a research topic on quantum transport theory.

In January 1978, he joined MOSTEK as an Integrated Circuit Design Engineer where he worked on five MOS analog IC's for telecommunications. He left in August 1981 to co-found Nova Monolithics where he was involved in consulting and custom IC designs which included mixed-signal chips with analog filtering and A/D conversion. He joined Texas Instruments, Dallas, TX, in May 1984 as a Senior IC Design Engineer. Since joining TI he has worked on the design of 24 analog and mixed-signal IC's of which 17 required analog-to-digital conversion. Jim has authored or coauthored 21 papers, given five conference talks, holds ten US patents, seven foreign patents with 12 patents pending.

Mr. Hellums was elected an MGTS in 1987, an SMTS in 1989, a DMTS in 1996 and a TI Fellow in 1997. He is a member of the American Physical Society, Eta Kappa Nu, Tau Beta Pi, and Alpha Chi.