Synthesis of High-Performance Analog Circuits in ASTRX/OBLX

Emil S. Ochotta, *Member, IEEE*, Rob A. Rutenbar, *Senior Member, IEEE*, and L. Richard Carley, *Senior Member, IEEE*

Abstract—**We present a new synthesis strategy that can automate fully the path from an analog circuit topology and performance specifications to a sized circuit schematic. This strategy relies on asymptotic waveform evaluation to predict circuit performance and simulated annealing to solve a novel unconstrained optimization formulation of the circuit synthesis problem. We have implemented this strategy in a pair of tools called ASTRX and OBLX. To show the generality of our new approach, we have used this system to resynthesize essentially all the analog synthesis benchmarks published in the past decade; ASTRX/OBLX has resynthesized circuits in an afternoon that, for some prior approaches, had required months. To show the viability of the approach on difficult circuits, we have resynthesized a recently published (and patented), high-performance operational amplifier; ASTRX/OBLX achieved performance comparable to the expert manual design. And finally, to test the limits of the approach on industrial-sized problems, we have synthesized the component cells of a pipelined A/D converter; ASTRX/OBLX successfully generated cells 2–3× more complex than those published previously.**

I. Introduction

A SURPRISING number of technologies that most people consider hallmarks of the *digital* revolution actually rely on a core of *analog* circuitry; cellular telephones, magnetic disk drives, and compact disc players are just a few such examples. Many of tomorrow's products—e.g., neural networks, speech recognition systems, and personal digital assistants—will also require analog circuitry. Unfortunately, the present state of analog CAD tools makes it difficult to quickly and cost effectively design the new analog circuitry that these new technologies will require. To conserve space and save money, it is now commonplace to implement entire mixed analog/digital systems on a single Application Specific Integrated Circuit (ASIC). But, to maximize profit, mixed analog/digital ASIC designers must also minimize design-time and thus time-to-market. Digital CAD tools facilitate this by providing a rapid path to silicon for the large digital component of these designs. Unfortunately, the analog component of these designs, although small in size, is still designed manually by experts using time-consuming techniques that have remained largely unchanged in the past 20 years [1], [2]. With the advent of logic synthesis tools [3] and semicustom layout techniques [4] to automate much of the digital design process, the analog section may consume 90% of the overall design time, while consuming only 10% of the ASIC's die area.

This paper describes a new approach to *analog circuit synthesis*, i.e., translating performance specifications into a circuit schematic with sized devices, thereby automating part of the analog design process. The scope of this paper is synthesis for *cell-level* (less than 100 devices) circuits. Starting from a transistor schematic, we seek both to design a dc bias point and size all devices to meet performance targets such as gain and bandwidth. Our approach combines the following ideas into an analog synthesis methodology.

- A novel *unconstrained optimization formulation* to which the circuit synthesis problem is mapped;
- *Simulated annealing* to solve the resulting optimization problem;
- *Asymptotic Waveform Evaluation* (AWE) to simulate circuit performance—the key component of the function to optimize;
- A compiled database of industrial quality, nonlinear device models, called *encapsulated device evaluators*, to provide the accuracy of detailed simulation while making the synthesis tool independent of low-level device modeling concerns;
- A *relaxed-dc formulation* of the nonlinear device simulation problem to avoid a CPU intensive complete dc operating point solution for each circuit simulation; and, finally,
- A separate *compilation phase* to translate the synthesis problem from a description convenient to the designer into an executable program that designs the circuit via optimization.

Although analog synthesis via simulated annealing is not new [5], the use of AWE and the added power of a separate compilation phase are completely novel. We believe the result is a *usable* synthesis system.

Throughout this paper, we will measure the effectiveness of our new analog circuit synthesis formulation and compare it to prior systems based on the five critical metrics for analog synthesis tools.

- **Accuracy**: the discrepancy between the synthesis tool's internal performance prediction mechanisms and those of a detailed circuit simulator that uses realistic device models;

Manuscript received February 15, 1995; revised July 21, 1995. This work was supported in part by the Semiconductor Research Corporation, The National Science Foundation, Harris Semiconductor, and Bell Northern Research. This paper was recommended by Associate Editor R. A. Saleh.

E. S. Ochotta was with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA. He is now with Xilinx, Inc., San Jose, CA 95124 USA.

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

Publisher Item Identifier S 0278-0070(96)03471-9.

- **Generality**: the breadth of the circuits and performance specifications that can be successfully handled by the synthesis tool;
- **Complexity**: the largest circuit synthesis task that can be successfully completed by the synthesis tool;
- **Synthesis time**: the CPU time required by the synthesis tool;
- **Preparatory effort**: the designer-time/effort required to render a new circuit design in a form suitable for the tool to complete.

An ideal system maximizes accuracy, generality, and complexity, while minimizing synthesis time and preparatory effort. Note that these metrics are not always easy to quantify. For example, the complexity of a synthesis task can be affected by many factors including the number of designable parameters (element values and device sizes), the number and difficulty of the performance specifications, the number of components in the circuit, and the inherent difficulty of evaluating the performance of the circuit. For these cases, where the definition of the term is qualitative, we select specific concrete metrics that provide a good indication of the underlying factor we wish to measure. For example, as the metric for complexity, we use the number of designable parameters the designer wishes the tool to determine plus the number of components in the circuit. This is easy to quantify and relates complexity to both the problem and circuit size. In addition to the five metrics above, we shall also use one additional term, *automation*, which we define as the ratio of the time it takes to design a new circuit for the first time manually to the time it takes with the synthesis tool. When comparing synthesis tools, manual design time will be the same for a given circuit, so maximizing automation is equivalent to minimizing the sum of preparatory time and synthesis time.

To provide a concrete set of synthesis tasks to compare our approach to that of other tools, we have generated synthesis results over a large suite of analog cells. This suite includes three classes of synthesis results.

- A suite of benchmark circuits that shows the generality of our approach by blanketing essentially all previous analog cell synthesis results;
- A redesign of a recently published manually designed analog cell that shows the ability of our approach to handle difficult circuits;
- A pipelined A/D converter that includes the most complex synthesized cells of which we are aware and shows the ability of our approach to handle large, realistic designs.

In comparison to prior approaches, our approach typically predicts circuit performance more accurately, yet requires 2–3 orders of magnitude less preparatory effort by the designer. In exchange for these substantial improvements, a small price is paid in synthesis time: our approach can require several hours of CPU time on a fast workstation, instead of seconds or minutes. This is an acceptable trade-off because automation is improved when designing a new circuit for the first time, i.e., spending these hours of a computer's time can save the months of designer's time required to complete the design manually or with other analog synthesis tools.

The remainder of this paper is structured as follows. Section II reviews prior approaches to analog circuit synthesis. Section III presents the basic ideas underlying our new formulation of the analog synthesis problem, while Section IV presents a circuit synthesis example to show how these ideas are applied to a real synthesis task. In Section V, we present the formulation in detail, and in Section VI, we revisit the few related approaches to synthesis and compare them to our approach. Section VII describes synthesis results and again compares to those from other approaches. Finally, Section VIII offers concluding remarks.

II. Review of Prior Approaches

Previous approaches to analog circuit synthesis [5]–[18] have failed to make the transition from research to practice. This is due primarily to the prohibitive one-time effort required to derive the complex equations that drive these synthesis tools. Because they rely on a core of equations, we refer to these previous approaches to synthesis as equation-based, and discuss their architecture in terms of the simplified search loop shown in Fig. 1. At each step in the synthesis process (each pass through this loop), a search mechanism perturbs the element values, transistor dimensions and other variable aspects of the circuit in an attempt to arrive at a design that more closely meets performance specifications and other objectives. Performance equations are used to evaluate the new circuit design, determine how well it meets its specifications, and provide feedback to the search mechanism to guide its progress. Because of their reliance on equations, these systems are still limited in the crucial areas of accuracy and automation. Let us examine these issues in greater detail.

- **Accuracy**: Equation-based approaches rely heavily on simplifications to circuit equations and device models. Consequently, the performance of the synthesized circuit often reflects the limitations of the simplified equations used to model it, rather than the inherent limitations of the circuit topology or underlying fabrication technology. The need for designs that push the limits of circuit topologies and use the latest technologies invalidates the use of many of these simplifications. For example, in a 3μm MOS process, $I_{DS} = K'W/2L(V_{GS} - V_T)^2$ is a workable model of the current-voltage relationship for a device, and equation-based approaches take advantage of the fact that it can be inverted to allow either voltage or current as the independent variable. This simply inverted equation allows an equation-based tool to quickly solve for a circuit's dc operating point, but it can yield grossly inaccurate performance predictions for a device with a submicron channel length. The need to support complex device models and high-performance circuits is fundamentally at odds with equation-based strategies that rely on these simple, easily inverted equations.
- **Automation**: Equation-based tools appear to design circuits quickly. But, the run-times of these tools are not an accurate measure of automation because they do not consider the preparatory time required to derive the circuit equations. Even for a relatively simple analog circuit,

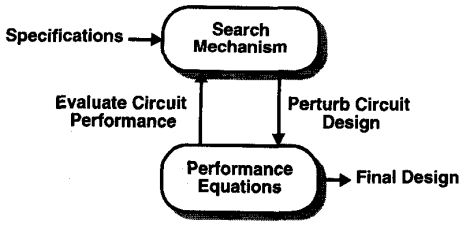


Fig. 1. Search process used in equation-based analog synthesis tools.

these equations are very complex, require considerable analog design expertise to derive, and must be entered as thousands of lines of program code. For a textbook design, this process can take weeks [6], while for an industrial design it can take several designer-years [7], and the process must be performed for each new circuit topology added to the synthesis tool's library. Moreover, adding these equations typically requires a user who is a programmer, an analog designer, and an expert intimate with the internal architecture of the tool. As a result, it is almost always easier for an industrial mixed-signal ASIC designer to design circuits manually rather than dedicate the effort required to teach these tools to do it.

However, researchers are aware of these accuracy and automation problems and several different techniques have evolved in an effort to address them. Early analog synthesis tools, such as IDAC [8] and OPASYN [6], used direct numerical means to optimize the analog circuit performance equations to meet specifications. Others, such as BLADES [9], attempted to achieve greater flexibility by using ruled-based strategies to organize the analog circuit equations. The automation problem inherent with equations was first addressed by OASYS [10], which eased the burden of equation derivation by introducing an aggressive hierarchical structure into circuit performance equations in an attempt to provide reusable circuit building blocks. This hierarchical structure became the core of later tools, such as CAMP [11] and An_Com [12]. The ability to reuse circuit equations led to the desire to be able to more easily edit and add to existing libraries of analog circuit design expertise. In early synthesis tools, analog circuit equations were hard-coded in the tool as part of the underlying solution strategy. OASYS VM [13] provided a first step toward an open system, by decoupling the expertise from the solution strategy. More recently, tools such as STAIC [14] and IDAC [15], allowed the user to specify analog circuit design equations directly using programming-like languages. Despite substantial early progress with these lines of research, the difficulty of deriving, coding, and testing performance equations still remains a daunting task, and researchers have made few strides with this style of synthesis in recent years.

The second major innovation that aimed at reducing preparatory effort was symbolic simulation [19], [20]. This technique was first introduced as a tool to aid manual design and education, and it was first integrated directly into a synthesis tool with ARIADNE [5]. Symbolic simulation generates analytical transfer functions for small linear or linearized circuits. For many important classes of circuits, such as operational amplifiers, most of the important performance specifications are linear in nature and are amenable to this kind of analysis. Moreover, fairly accurate equations for many of the remaining nonlinear specifications can be derived by inspection, whereas equations for the linear specifications are much more involved. Thus, in theory there is a great deal of leverage that can be gained from symbolic analysis. However, symbolic simulation has yet to overcome substantial technical obstacles before it can fully automate performance equation derivation for large, high-performance circuits. Applied blindly, symbolic simulation of a circuit linearized from a handful of devices can generate an expression with tens of thousands of terms, and the number of terms grows exponentially with circuit size. Because of memory and CPU time concerns, generating exact symbolic expressions for all but the smallest circuits is impractical. As a result, practical symbolic simulation algorithms generate *pruned* expressions, but this pruning leads to accuracy problems. If device models are pruned before symbolic analysis, the resulting expressions are more compact but lack accuracy for high-performance designs. If the final equations are pruned, symbolic simulation still suffers from memory and CPU time problems, and the result is faithful only to some performance concerns. For example, pruning terms whose magnitude is small typically distort phase information, on which the circuit's performance may critically depend. Very recently, strategies have been developed for effectively reducing the number of terms during simulation, and symbolic simulation is now efficient enough with computer resources to be applied to medium-sized opamps [21], [22]. However, even with these recent innovations, the pruned expressions are valid in only a very limited region of the achievable design space for the circuit, and would have to be frequently regenerated if the designable circuit parameters were varied significantly. The ability to do this in a manner efficient enough for synthesis has not been demonstrated to date, although in future these techniques may yet provide a completely automatic path to equation-generation.

Fewer technical innovations have been made to improve the *accuracy* of analog synthesis techniques. One recent area of improvement has been the incorporation of realistic device models. In [16], Maulik incorporated complete BSIM [23] device models from SPICE 3 [24] into a special purpose synthesis tool. The use of these models substantially complicates solving for dc operating points in an evolving circuit because the models cannot be inverted analytically. To address this problem, Maulik formulated the circuit synthesis problem as a constrained optimization problem and enforced Kirchhoff's current law by explicitly writing dc operating point constraints. This technique combines simultaneous circuit performance optimization with dc operating point solution. As discussed in Section III, our relaxed-dc formulation evolved from this paper.

Although many innovations have been made during the evolution of equation-based analog synthesis tools over the past decade, the combined problems of accuracy and automation (i.e., preparatory effort) have never been adequately addressed in a single cohesive approach. A new strategy is needed that addresses both these shortcomings.

One simple solution is to replace the equations with a direct simulation technique. This is the basic approach that was first proposed for analog circuit *optimization* decades ago [25], rediscovered when faster computers and improved simulators made it practical for research [26]–[28], and is now making its way into industrial CAD systems. For example, DELIGHT.SPICE [26] follows the basic structure of Fig. 1, where the search is performed by the method of feasible directions, a gradient-based optimization technique, and the performance equations have been replaced by SPICE [29], [30]. Because SPICE is a detailed circuit simulator, no designer supplied equations are required (except to extract the performance specifications from simulation results), and the performance prediction is very accurate. Unfortunately, as the core of an optimization loop, SPICE-class simulators are slow. So slow, in fact, that DELIGHT.SPICE is an *optimization* tool, not a *synthesis* tool. The key hurdle that has not been overcome to make this transition from *optimization* to *synthesis* is that optimization requires a good initial starting point to find an excellent final answer, while synthesis requires no special starting point information. This critical distinction is more carefully explained as follows:

- **Efficiency/Starting Point Sensitivity**: Because SPICE-class simulators are slow, the search mechanism must invoke the simulator as *in*frequently as possible. As a result, simulation-based methods use local optimization techniques that require few iterations to converge. These techniques must be primed with a good initial circuit design, otherwise, an optimization may not converge or may converge to a local minima significantly worse than the circuit's best capabilities [31]. In circuit synthesis, a local optimizer is not practical because the search space contains many local but non-global minima [14], [26] and because—even with a good rough design—it is only luck if optimizing from the user's initial circuit design leads to the globally optimal final solution.

The accuracy and reduced preparatory effort that comes with simulation-based optimization are the two characteristics that have been substantially lacking from equation-based systems. One approach to incorporate simulation into an equation-based system, as taken in OAC [17], is to run a simulation-based optimizer as a post-processor. This improves accuracy, but there is no guarantee that the circuit generated by the equation-based synthesis will be the starting point needed by the simulation-based optimizer to find the globally optimal circuit. Furthermore, because of the extensive simulation run-times, only the performance specifications that can be validated with ac and dc analyses are optimized using these techniques [17]. And, perhaps most importantly, the months of preparatory effort required to derive analog circuit performance equations are still required by the equation-based part of the overall design process.

A solution to the problem of preparatory effort dictates that the user not derive, code, or prune analog circuit equations. A simulation-based approach meets this criteria, but requires innovations to avoid problems with efficiency due to circuit simulation and starting point dependency due to optimization. We are aware of two analog synthesis tools that meet these criteria: ASTRX/OBLX [32]–[35], which is the subject of this paper, and a more recent tool presented in [36]. Before comparing the differing approaches of these two tools, we first describe the architecture and underlying ideas behind ASTRX/OBLX in the following sections. We return to this comparison in Section VI.

III. Basic Synthesis Formulation

In this section, we present our basic analog circuit synthesis formulation. We begin with the specific design goals that guided the evolution of this formulation and the key ideas that form its foundation. We then outline its architectural aspects.

A. Design Goals

Our design goals for a new analog circuit synthesis architecture are to directly address the automation, accuracy, and efficiency problems we identified with previous approaches. First, to streamline the path from a circuit idea to a sized circuit schematic, our new architecture should require only hours rather than weeks/months of preparatory effort to design a new circuit. Second, the system should find high-quality circuit design solutions without regard to starting point rather than getting trapped in the nearest local minima. Third, our new system should yield accurate performance predictions for high-performance circuits rather than suffer from problems due to device model or performance equation simplifications. And, finally, the system must be able to design the complex, high-performance circuits required in modern products.

Realistically, we cannot hope to achieve progress in all these areas without making some trade-offs. The first concession we are willing to make is increased run-time. This is because our primary goal here is maximal automation, which is the sum of preparatory and run-times. Equation-based synthesis tools use only minutes of CPU time but require the designer to spend months deriving, coding, and testing equations. We believe the following scenario is more appealing: after an afternoon of effort, a circuit designer goes home while the synthesis tool completes the design overnight. Realizing this scenario is our primary goal. We are also willing to make two additional concessions for our initial implementation of our new formulation. The first of these is to exclude automatic topological design. We believe that sizing/biasing is the correct starting point for a new synthesis strategy, and a suitable mechanism for choosing among topological variants can be added later. Moreover, a tool that finds optimal sizes for a single user-supplied topology is still directly usable by analog designers. The third concession is to exclude operating range and manufacturablility concerns, and—like most previous synthesis tools—the work presented here performs only nominal circuit design. However, since the conclusion of our initial work, this formulation has been augmented with the ability to handle operating range and manufacturing concerns and preliminary results appear in [37].

B. Underlying Ideas

To achieve our goals, our circuit synthesis strategy relies on five key ideas: synthesis via optimization, AWE, simulated

annealing, encapsulated device evaluators, and the relaxed-dc numerical formulation. We describe these ideas below.

Synthesis via Optimization: We perform fully automatic circuit synthesis using a constrained optimization formulation, but solved in an unconstrained fashion. As in [6], [16], and [26], we map the circuit design problem to the constrained optimization problem of (1). Here $\underline{x}$ is the set of independent variables—geometries of semiconductor devices or values of passive circuit components—for which we wish to find appropriate values; $\underline{f}(\underline{x})$ is a set of objective functions that codify performance specifications that the designer wishes to optimize, e.g., power or bandwidth; and $\underline{g}(\underline{x})$ is a set of constraint functions that codify specifications must be beyond a specific goal, e.g., gain $\geq$ 60 dB. Scalar weights w_i balance competing objectives

$$\underset{\underline{x}}{\text{minimize}} \sum_{i=1}^{k} w_i \cdot f_i(\underline{x}) \quad \text{s.t.} \quad \underline{g}(\underline{x}) \leq 0. \tag{1}$$

To allow the use of simulated annealing, 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, $C(\underline{x})$, defined by

$$C(\underline{x}) = \sum_{i=1}^{k} w_i f_i(\underline{x}) + \sum_{j=1}^{l} w_j g_j(\underline{x}). \tag{2}$$

The key to this formulation is that the minimum of $C(\underline{x})$ corresponds to the circuit design that best matches the given specifications. Thus, the synthesis task becomes two more concrete tasks: 1) evaluating $C(\underline{x})$ and 2) searching for its minimum. However, performing these tasks is not easy. In equation-based synthesis tools, evaluating $C(\underline{x})$ is done using designer-supplied equations. To achieve our automation goals, we must avoid the large preparatory effort it takes to derive these equations. Moreover, in searching for the minimum, we must address the issues of starting point independence and global optimization, since $C(\underline{x})$ may have many local minima.

Asymptotic Waveform Evaluation: To evaluate circuit performance, i.e., $C(\underline{x})$, without designer supplied equations, we rely on an innovation in simulation called AWE [38], [39]. AWE is an efficient approach to analysis of arbitrary linear circuits that is several orders of magnitude faster than SPICE for ac analysis. By matching the initial boundary conditions and the first $2q-1$ moments of the actual circuit transient response to a reduced q-pole model, AWE can predict small-signal circuit performance using a reduced complexity model. AWE is a general simulation technique that can be applied to any linear or linearized circuit and yields accurate results without manual circuit analysis. Thus, for linear performance specifications, AWE replaces performance equations, but does so at a fraction of the run-time cost of SPICE-like simulation.

Simulated Annealing: We have selected simulated annealing [40] as the optimization engine that will drive our search for the best circuit design in the solution space defined by $C(\underline{x})$. This method provides the potential for global optimization in the face of many local minima. Simulated annealing has a theoretically proven ability to find a global optimum under certain restrictions [41]. Although these restrictions are not enforceable for most industrial applications, the proofs suggest an algorithmic robustness that has been validated in practice [42]. Because annealing incorporates controlled *hill-climbing*, it can escape local minima and is starting-point independent. Annealing has other appealing properties including its ability to optimize without derivatives. Furthermore, although annealing typically requires more function evaluations than local optimization techniques, it is now achieving competitive run-times on problems for which tuned heuristic methods exist [43]. Because annealing directly solves *un*constrained optimization problems, we require the scalar cost function of (2).

Encapsulated Device Evaluators: To model active devices, we rely on a compiled database of industrial models we call *encapsulated device evaluators*. These provide the accuracy of a general-purpose simulator while making the synthesis tool independent of low-level device modeling concerns. As with any analysis of a circuit, we use models to linearize nonlinear devices, generating a small signal circuit that can be passed to AWE. In a practical synthesis system, it is no longer a viable alternative to use one- or two-equation approximations instead of the hundreds of equations used in industrial device models. Unlike equation-based performance prediction, where assumptions about device model simplifications permeate the circuit evaluation process, with encapsulated device evaluators all aspects of the device's representation and performance are hidden and obtained only through requests to the evaluator. In this manner, the models are completely independent of the synthesis system and can be as complex as required. For our purposes, we rely entirely on device models adopted from detailed circuit simulators such as Berkeley's SPICE 3 [24].

Relaxed-DC Formulation: To avoid a CPU intensive dc operating point solution after each perturbation of the circuit design variables, we rely on a novel recasting of the unconstrained optimization formulation for circuit synthesis we call the *relaxed-dc formulation*. Supporting powerful device models is not easy within a synthesis environment because we cannot arbitrarily invert the terminal relationships of these models and choose which variables are independent and which are dependent. This critical simplification enables equation-based approaches to solve for the dc bias point of the circuit analytically and, as a result, very quickly. In contrast, when the models must be treated numerically, as in circuit simulation, an iterative algorithm such as Newton–Raphson is required. For synthesis, this approach consumes a substantial amount of CPU time that we would prefer not to waste on intermediate circuit designs that are later discarded. Instead, following Maulik, [16], we explicitly formulate Kirchhoff's laws, which are solved implicitly during dc biasing, and include them in $\underline{g}(\underline{x})$, the constraint functions in (2). Just as we must formulate optimization goals such as meeting gain or bandwidth constraints, we now formulate dc-correctness as yet another goal to meet. Of course, the idea of relaxing the dc constraints in this manner is not new, e.g., an analogous formulation for microwave circuits is discussed in [44]; however, it has been controversial [45].

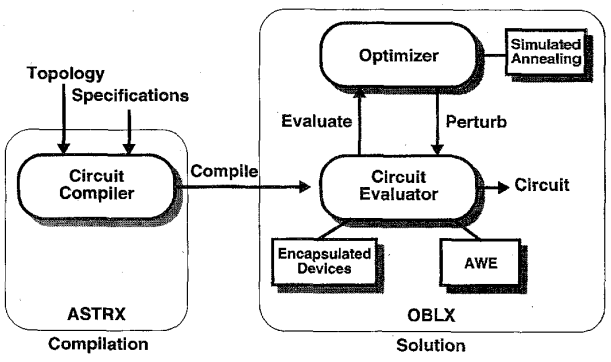


Fig. 2. New synthesis architecture.

C. System Architecture

We combine these five ideas to create an architecture that provides a fully automated path from an unsized circuit topology and a set of performance specifications to a completed, synthesized circuit (see Fig. 2). This path is comprised of two phases.

- **Compilation**: For each new circuit synthesis task, compilation generates code that implements the cost function, $C(\underline{x})$. To evaluate this cost function, the compiler will generate the appropriate links to the encapsulated device evaluators and AWE. Because of our relaxed-dc formulation, the compiler must also derive the dc-correctness constraints (KCL at each node) that will enforce Kirchhoff's laws and encode them in the cost function.
- **Solution**: This cost function code is then compiled and linked to our solution library, which uses simulated annealing to numerically find its minimum, thereby designing the circuit.

IV. Synthesis Example

In the previous section, we briefly introduced the concepts that underlie our new analog circuit synthesis formulation. In this section, we present a small but complete synthesis example to make concrete the entire path from problem to solution. Assume we wish to size and bias the simple differential amplifier topology shown in Fig. 3 to meet the specifications given in Table I. The topology of the circuit under design and the performance specifications the completed design must achieve—essentially the information in Fig. 3 and Table I—are the information the designer must supply to the compiler to generate the cost function and complete the synthesis process. In this section, we shall see exactly what information about the topology and specifications is required by the circuit compiler by showing how it uses this information to create $C(\underline{x})$, the cost function that is optimized (defined in (2)).

The first component of $C(\underline{x})$ is the set of independent variables, $\underline{x}$. These variables are readily apparent from the description of the circuit topology. Assume that for our example, the sizes for **M3** and **M4** are given as constants, the transistors **M1** and **M2** are matched to preserve circuit symmetry, and the rest of the component values are allowed

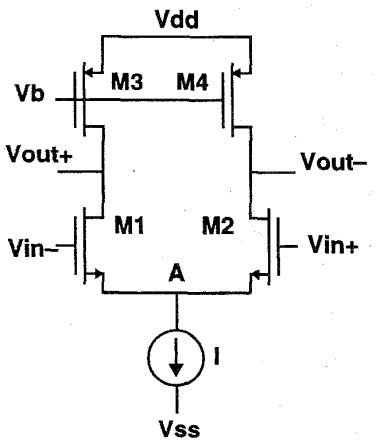


Fig. 3. Design example: Circuit under design.

TABLE I
Design Example: Specifications

Attribute	Specification
differential gain, A_{dm}	↑[a]
gain bandwidth, UGF	1 MHz
slew rate, SR	1 V/μs

a. ↑ means maximize.

to vary. Then, $\underline{x} = \{W, L, I, Vb\}$, where W and L represent dimensions for both **M1** and **M2**. However, as we shall see in a few paragraphs, because of the relaxed-dc formulation, $\underline{x}$ is not yet complete.

Recall from (2), that $C(\underline{x})$ is composed of objective functions $\underline{f}(\underline{x})$ and constraint functions $\underline{g}(\underline{x})$. Since, for our example, the only objective is to maximize $A_{\rm dm}$, $\underline{f}(\underline{x})$ contains only $f_{A\rm dm}(\underline{x})$, which calculates $A_{\rm dm}$. We provide a *simulation-oriented* definition of $f_{A\rm dm}(\underline{x})$ since AWE will be used to simulate the circuit's performance. This consists of a *test jig*, a set of measurements to make on the jig, and any simple arithmetic that must be performed to calculate the values we are interested in from the simulation results. The test jig is important because it supplies the circuit environment (stimulus, load, supplies, *et cetera*) in which the circuit under design is to be tested. We use the test jig in Fig. 4 to measure $A_{\rm dm}$. Following [26], we also require a *good* and *bad* value for each objective to transform $f_{A\rm dm}(\underline{x})$, from the user-supplied function to a function more amenable to optimization (see Section V for further details). The resulting function is (3), where tf is the transfer function from input to output. The nodes at which to evaluate the transfer function must be specified by the user, but the transfer function itself is obtained for each new circuit by using AWE, and the function to calculate dc gain from a transfer function is predefined within the compiler

$$f_{A\rm dm}(\underline{x}) = \frac{\text{dc\_gain(tf)} - \text{good}}{\text{bad} - \text{good}}. \tag{3}$$

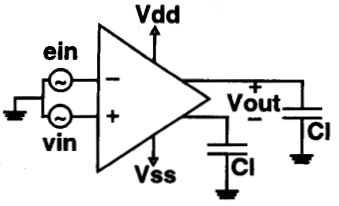


Fig. 4. Design example: Test jig for A_{dm}, UGF.

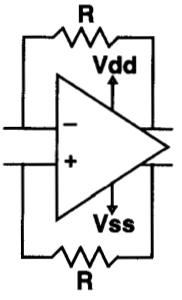


Fig. 5. Design example: Bias circuit.

The next step in creating $C(\underline{x})$ is to complete the definition of $\underline{x}$. To understand why we must add variables to $\underline{x}$, we trace the information required to calculate $f_{A\mathrm{dm}}(\underline{x})$. Here, AWE determines the needed transfer function, but AWE requires a linearized (i.e., small-signal equivalent) circuit. Like a detailed circuit simulator, we rely on device models to linearize our circuits—in our case, these models are encapsulated device evaluators. An evaluator converts the dimensions and port voltages of each device into a set of linear elements that models the device's behavior at that operating point. After replacing each transistor with its model, we can then use AWE to evaluate the circuit's performance. What we have not discussed so far is how we obtain the port voltages of each device. In a circuit simulator, these voltages must be explicitly solved for using a time-consuming iterative procedure such as Newton–Raphson. In contrast, in our relaxed-dc formulation, we simply include these voltages as additional variables in $\underline{x}$. The compiler includes these voltage variables automatically based on a circuit analysis. To provide greater flexibility, these dc bias concerns can be separated from the small-signal test jigs—a technique familiar to analog designers. Thus dc voltage variables are obtained from a bias circuit provided by the user. For our example, an analysis of the bias circuit of Fig. 5 yields $\underline{x} = \{W, L, I, Vb, V\mathrm{out}+, V\mathrm{out}-, V_A\}$. This completes $\underline{x}$, but it is only half of the relaxed-dc formulation.

We complete the relaxed-dc formulation by forcing the node voltages to take values such that Kirchhoff's laws are obeyed. This is accomplished by using members of the constraint functions, $\underline{g}(\underline{x})$, the other component of $C(\underline{x})$. To begin, we replace the transistors in the circuit with large-signal models returned by the device evaluators. Simplifying the device models for the sake of clarity gives the circuit of Fig. 6. Kirchhoff's current law can then be written at each node in the circuit, e.g., at node A: $I - Id1 - Id2 = 0$. To ensure this KCL equation is met when our optimization is complete, we

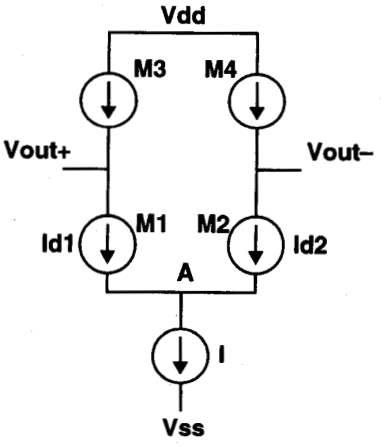


Fig. 6. Design example: Large-signal equivalent circuit.

include $g_A(\underline{x})$ from (4) as a member of $\underline{g}(\underline{x})$

$$g_A(\underline{x}) = \max(0, |I - Id1 - Id2| - \tau_{\mathrm{abs}}). \tag{4}$$

$g_A(\underline{x})$ contributes a penalty to the cost function whenever the KCL error at node A is larger than some numeric tolerance, τ_{abs}.[1] We formulate the other KCL equations in the same fashion, creating $g_{V\mathrm{out}+}(\underline{x})$ and $g_{V\mathrm{out}-}(\underline{x})$. Together, these three constraints enforce KCL, thereby completing the relaxed-dc formulation.

For our example, the other members of $\underline{g}(\underline{x})$ correspond to the other performance specifications for which we wish to design. Thus, we need constraint functions for UGF and SR (see Table I). These are formulated from the user provided expressions much like $f_{A\mathrm{dm}}(\underline{x})$ was formulated in (3); however, for specifications, the *good* value is a hard boundary and improvements beyond this value are not reflected in the cost function. The specification for UGF can be written in a simulation-based style, using the same test jig as was used for A_{dm}. This is not the case with the slew rate because measuring slew rate would require a transient simulation, which is not straightforward with AWE. However, unlike gain and unity gain frequency, slew rate is described with an easily derived expression. If we assume that we are interested only in the rate at which the output slews downwards, we can write this expression by inspection as $\mathrm{SR} = I/(2(Cl + Cd))$, where Cd is the capacitance at the output node due to the transistors. Thus, our new formulation supports a mix of simulation- and equation-based specifications. The decision of which method to use depends on the kind of specification. As experiments with symbolic simulation have shown, equations for linear specifications such as A_{dm} and UGF can be huge, e.g., 10 000 + terms for a circuit with ten devices, and the number of terms grows exponentially with circuit size. In contrast, simulation with AWE uses a numeric technique and can evaluate A_{dm} and UGF in a few tens of milliseconds for circuits of this size. Moreover, AWE's algorithmic complexity is roughly that of an LU factorization,

[1] The large-signal model and this formulation of the KCL constraint are somewhat simplified for clarity. For more detail, see Section V.

approximately $O(n^{1.4})$ where n is the number of nodes in the circuit. The speed of AWE and the ability to describe linear performance specifications without deriving circuit equations are two of the chief advantages of our new formulation over previous synthesis approaches.

Including these final terms, we obtain (5), the final form of the cost function the compiler generates and that must be minimized to complete the circuit design

$$C(\underline{x}) = w_{\mathrm{Adm}} f_{\mathrm{Adm}}(\underline{x}) + w_{\mathrm{UGF}} g_{\mathrm{UGF}}(\underline{x}) + w_{\mathrm{SR}} g_{\mathrm{SR}}(\underline{x}) + w_A g_A(\underline{x}) + w_{V\mathrm{out}+} g_{V\mathrm{out}+}(\underline{x}) + w_{V\mathrm{out}-} g_{V\mathrm{out}-}(\underline{x}). \quad (5)$$

In this section, we have used an example to sketch the process the compiler must follow to map a synthesis problem into a cost function. The compiler generates this cost function as C code that is complied and linked to the optimization library, creating an executable program that performs the synthesis task specified by the user.

V. Design Details

Now that we have explained the basic ideas behind our new analog circuit synthesis formulation, and used an example to show how these ideas can be applied to analog circuit synthesis, in this section we present a more complete look at the algorithms used. We then address two issues unique to this formulation: the implications the relaxed-dc formulation has on circuit synthesis and the practicality of the formulation in terms of automation and numerical robustness.

A. Algorithmic Aspects of the Compiler

As can be seen in the example, the circuit compiler performs a number of tasks when translating the user's problem description into a cost function. These can be summarized as: a) determine the set of independent variables ($\underline{x}$), b) generate large-signal equivalent circuits for biasing, c) write KCL constraints for the large-signal circuits, d) generate small-signal equivalent circuits for AWE, e) generate cost terms for each circuit performance metric specified by the user, and f) write all the code that describes the cost function for this circuit synthesis problem.

The majority of these tasks are algorithmically straightforward, or involve algorithms that are well understood by compiler writers [46]. However, one somewhat subtle aspect of compilation that deserves mention is the task of determining the set of independent variables, $\underline{x}$. The user specifies most of these, but the compiler must find a set of independent node voltages to include in $\underline{x}$ as part of the relaxed-dc formulation. To do so, the compiler performs a symbolic tree-link analysis [47] of the large-signal equivalent circuit, which is built from the input netlist with the help of device templates provided by the encapsulated device evaluators. A path is then traced from each node to ground. Whenever a node voltage cannot be trivially determined, its value becomes another variable in $\underline{x}$.

B. The Simulated Annealing Algorithm

As described in Section III, our optimization engine employs the simulated annealing algorithm [40], [41] to solve the analog synthesis problem via unconstrained optimization. This algorithm can be described with the following pseudocode

$\underline{x} = \underline{x}_0, T = T_{\mathrm{HOT}}$
while not *frozen*$(\underline{x}, T)$
  while not *done_at_temperature*$(\underline{x}, T)$
    $\Delta\underline{x} =$ *generate*$(\underline{x})$
    if *accept*$(\underline{x}, \underline{x} + \Delta\underline{x}, T)$ **then**
      $\underline{x} = \underline{x} + \Delta\underline{x}$
  $T =$ *update_temp*(T).

To describe the details of how our optimization engine works, we must describe each of the components of this algorithm. To begin, for all simulated annealing implementations, the *accept* function is called the Metropolis criterion [48], and is defined by (6), where *random()* returns a uniformly distributed random number on [0, 1]

$$random() < e^{\left[\frac{C(\underline{x}) - C(\underline{x}+\Delta\underline{x})}{T}\right]}. \quad (6)$$

Next, we describe annealing's four problem specific components.

- The problem *representation*, which determines how the present state of the circuit being designed is mapped to $\underline{x}$, the present state manipulated by the annealer;
- The *move-set*, which determines how the *generate* function perturbs the present circuit state $\underline{x}$ to create the new state $\underline{x} + \Delta\underline{x}$;
- The *cost function* $C(\underline{x})$ which determines how the cost of each visited circuit configuration is calculated;
- and the *cooling schedule*, which controls T, directing the overall cooling process. This defines the initial temperature, T_{HOT}, and the functions *frozen*, *done at temperature*, and *update temp*.

C. Problem Representation

The problem representation appears straightforward: the variables in $\underline{x}$ map to aspects of the evolving circuit design, such as device sizes and node voltages. However, there are two concerns here. The first is that the user defines a set of variables, then writes expressions to map these variables to circuit component values. As in the example of Section IV, these are typically identity relations, but may be arbitrarily complex to allow complex matchings and inter-relationships. For example, the expression '$\mathbf{2} * \mathbf{L}$,' might be used in a bias circuit where one device length must always be twice that of another device.

The second concern is that we do not represent all the variables as continuous values. Node voltage values must clearly be continuous to determine an accurate bias point. Device sizes, however, can be reasonably regarded as discrete quantities, since we are limited by how accurately we can etch a device. Moreover, there is considerable advantage to be had from properly discretizing device sizes: the coarser the discretization, the smaller the space of reachable sizes that must be explored. Because small changes in device sizes make proportionally less difference on larger devices, we typically

use a logarithmically spaced grid. The choice to discretize variables and the coarseness of the grid is made by the user and specified in the input description file. Thus, there may be three kinds of variables in $\underline{x}$: 1) user-specified discrete variables, e.g., a transistor width; 2) user-specified continuous variables, e.g., a current source value; and 3) automatically created continuous variables, e.g., a node voltage added to implement the relaxed-dc formulation.

D. Move-Set

Given the present state $\underline{x}$ the *move-set* $\Delta\underline{x}$ is the set of allowable perturbations on it, which are implemented by the *generate* function. The issue is the need for an efficient mechanism to generate each perturbation, and this is substantially complicated because we use a mix of discrete and continuous variables. For discrete variables, the problem is simpler because there is always a smallest allowable move, an *atomic* perturbation. The two issues here are what larger moves should be included in the move-set for efficiency and how to decide when to use these larger moves. We address these issues by defining a set of *move classes* for each variable. Each class is a tuple (x_i, r) where x_i is the variable to be perturbed by this move class and r is a range $[r^{\min}, r^{\max}]$, which is related to the range of allowable values for x_i. For example, for a transistor width variable that has been discretized such that it can take on 100 possible values, we might create three move classes with ranges $[-1, 1]$, $[-10, 10]$, and $[-50, 50]$. The idea is that during annealing, we will randomly select a move class. This determines not just x_i, the variable to perturb, but r the range with which to bound the perturbation. Once selected, we generate Δx_i as an integral uniform random number in $[r^{\min}, r^{\max}]$. Finally, Δx_i may be adjusted to ensure the new variable value, $x_i + \Delta x_i$, lies within the allowable values for x_i.

To improve annealing efficiency, we wish to optimize the usefulness of the move-set by favoring the selection of move classes that are most effective at this particular state of the annealing process. To do this, we use a method based on the work of Hustin [49]. We track how each move has contributed to the overall cost function, and compute a *quality factor* that quantifies the performance of that move class. For move class i, the quality factor Q_i is given by (7)

$$Q_i = (1/\|G_i\|) \sum_{j \in A_i} |\Delta C_j|. \tag{7}$$

Here, $\|G_i\|$ is the number of generated move attempts that used class i over a window of previous moves and A_i is the accepted subset of those moves (i.e., $A_i \subseteq G_i$). Furthermore, $|\Delta C_j|$ is the absolute value of the change in the cost function due to accepted move j. Q_i will be large when the moves of this class are accepted frequently ($\|A_i\|$ is large), and/or if they change the value of the overall cost function appreciably (some $|\Delta C_j|$ are large). We can then compute the probability that a particular move class should be selected. If Q is the sum over all the quality factors, $Q = \sum_i Q_i$

$$p_i = \frac{Q_i}{Q} \tag{8}$$

can then be regarded as the fraction of moves that should be dedicated to move class i. Initially, when almost any move is accepted, large moves will change the cost function the most, giving them the largest quality factors and likelihood of being selected. When the optimization is almost complete, most large moves will not be accepted, so more small moves will be accepted and their quality factors and probabilities will increase. Using this scheme, we automatically bias toward the moves that are most effective during a particular phase of the annealing run.

For continuous variables, the situation is more complex. For an n-dimensional real-valued state, $\underline{x} \in R^n$, it is difficult to determine the correct *smallest* $\Delta\underline{x}$ because adjustments in real values may be infinitesimally small. We may need to explore across a voltage range of several volts and then converge to a dc bias point with an accuracy of a few microvolts. We are aware of several attempts to generalize simulated annealing to problems with real-valued variables [50]–[52], and each presents methods of controlling the move-set. These methods show promise, but require complex time-consuming matrix manipulations, large amounts of memory, or gradient information to determine the appropriate move sizes. Fortunately, for analog circuit synthesis, we can take advantage of problem-specific information to aid in selecting the correct largest and smallest moves. For user-specified variables, such as the size of a resistor or capacitor, the precision with which the value can be set—and the atomic move size—is a function of the underlying fabrication process and readily available to the designer. For node voltages, the smallest moves are dictated by the desired accuracy of performance prediction. Duplicating the method used in detailed circuit simulators, these tolerances can be specified with an absolute and relative value and the smallest allowable move derived from them. Thus, we use problem specific information to determine minimum move sizes and create move classes, allowing these continuous variables to be treated in the same fashion as the discrete variables.

To further assist in manipulating continuous node voltages, in addition to the purely random *undirected* move classes found in most annealers, we augment the annealer's move-set with *directed* move classes that follow the dc gradient. This technique is similar to the theoretically-convergent continuous annealing strategy use by Gelfand and Mitter [52]. They use gradient-based moves within an annealing optimization framework, and they prove that this technique converges to a global minimum given certain restrictions on the problem. Our addition of dc gradient moves has the same flavor. We incorporate gradient-based voltage steps as part of our set of move classes. Using KCL-violating currents calculated at each node, Δi, and the factored small-signal admittance matrix, Y^{-1}, we can calculate the voltage steps to apply at each node, Δv, as

$$\Delta\underline{v} = \alpha(Y^{-1}\Delta\underline{i}) \tag{9}$$

where α is a scaling factor that bounds the range of the move. Thus, this move effects all the node voltage variables simultaneously. Equation (9) also forms the core of a Newton–Raphson iterative dc solution algorithm—the technique

used in most circuit simulators to solve for the dc operating point—so we incorporate the complete algorithm as an additional move class in the move-set. Because of the complexity of performing Newton–Raphson within a simulated annealing algorithm, we have no theoretical proof of convergence. However, in practice, this technique allows the annealer to converge to a dc operating point at least as reliably as a detailed circuit simulator.

Finally, we also combine directed and undirected moves into a single move class to further augment the move-set. These combination moves are designed to alter the circuit component values, and thereby circuit performance, while maintaining a correct dc operating point. Without combination moves, when the optimization is nearly frozen, it is difficult for the annealer to adjust circuit component values—possibly improving circuit performance—without incurring a large penalty in the cost function as a result of dc operating point errors.

In summary, the complete move-set for the annealer comprises move classes of these types

- **Undirected moves**: where we modify a single variable (discrete or continuous) and generate Δx_i as a uniform random number in $[r^{\min}, r^{\max}]$;
- **Directed moves**: where the Newton–Raphson algorithm is used to perturb all the node voltage variables simultaneously;
- **Combination moves**: that perturb a user-specified variable in an undirected fashion and then immediately perform a Newton–Raphson solve.

E. Cost-Function

The heart of the formulation and the next problem specific component of the annealing algorithm is the circuit specific cost function $C(\underline{x})$ generated by the compiler, which maps each visited circuit configuration $\underline{x}$ to a scalar cost. The cost function was described by example in Section IV. In general, it has the form

$$C(\underline{x}) = \underbrace{C^{\text{obj}}(\underline{x})}_{\text{objective}} + \underbrace{C^{\text{perf}}(\underline{x}) + C^{\text{num}}(\underline{x}) + C^{\text{dev}}(\underline{x}) + C^{\text{dc}}(\underline{x})}_{\text{penaltyterms}} \quad (10)$$

where each term in (10) represents a group of related terms in a particular compiler-generated cost function. There are two distinct kinds of terms: the *objective* terms correspond to $\underline{f}(\underline{x})$ in (2) and must be minimized, while the *penalty* terms correspond to $\underline{g}(\underline{x})$ and must be driven to zero. Here, C^{obj} and C^{perf} are generated from the user-supplied performance objectives and specifications; C^{num} penalizes regions of the solution space that may lead to numerically ill-conditioned circuit performance calculations; C^{dev} forces active devices to be in particular user-specified regions of operation; and C^{dc} implements the relaxed-dc formulation. In the following paragraphs, we describe the most interesting of these cost-function components in greater detail.

Objective Terms: Following [26], circuit performance and figures of merit such as power and area can be specified as falling into one of two categories: an objective or a constraint. Regardless of the category, the user is also expected to provide 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. Performance targets that are given as objectives become part of C^{obj}, which we define as follows. Formally, let

$$\Omega^{\text{obj}} = \{f_i(x) \mid 1 \le i \le k\} \quad (11)$$

be the set of k performance objective functions provided by the user. Then Ω^{obj} is transformed into C^{obj} as follows:

$$C^{\text{obj}}(\underline{x}) = \frac{1}{\|\Omega^{\text{obj}}\|} \cdot \sum_{i \in \Omega^{\text{obj}}} \hat{f}_i(\underline{x}) \quad (12)$$

where

$$\hat{f}_i(\underline{x}) = \frac{f_i(\underline{x}) - \text{good}_i}{\text{bad}_i - \text{good}_i} \quad (13)$$

and $f_i(\underline{x})$ is the ith specification function provided by the user, and bad_i and $good_i$ are the *bad* and *good* normalization values specified for function i. The normalization process of (13) has these advantages. First, it provides a natural way for the designer to set the relative importance of competing specifications. Second, it provides a straightforward way to normalize the range of values that must be balanced in the cost function. Note that the range of interesting values between *good* and *bad* maps to the range [0, 1], and regardless of whether the goal is maximizing or minimizing $f_i(\underline{x})$, optimizing toward *good* will always correspond to minimizing the normalized function $\hat{f}_i(\underline{x})$. Finally, this normalization and the inclusion of the normalizing factor $(1/\|\Omega^{\text{obj}}\|)$ helps keep the cost function formulation robust over different problems, by averaging out the effect of having a large number of objectives for one circuit and a small number for another. The user has a wide range of predefined functions at his disposal to create $f_i(\underline{x})$. These include node voltages, device model parameters, and functions to derive linear performance characteristics such as gain and bandwidth from the transfer functions continually being derived by AWE.

Performance Specifications: The performance specifications provided by the user make up C^{perf}, and are quite similar to objectives with two exceptions. First, a specification is a hard constraint so circuit performance better than the *good* value does not contribute to C^{perf}. Second, an additional scalar weight biases the contribution of each performance specification to the overall cost function. These weights are determined algorithmically during the annealing process and should not need to be adjusted by the user. We defer detailed discussion of the weight control algorithm until Section V.H.

Operating Point Terms: The most unusual component of the cost function is the C^{dc} penalty term. This term implements the relaxed-dc formulation. Recall that we explicitly formulate Kirchhoff's current law constraints for each linearly independent node in all the bias circuits. At the end of the optimization, these C^{dc} terms will be zero and Kirchhoff's laws will be met. C^{dc} includes two views of Kirchhoff's current law (KCL), a *current view* C^{KCL}, and a *voltage* view C^{DV}. To show how these terms are calculated, (14)

defines the sum of the currents leaving node n via its incident branches B^n, and (15) defines the average magnitude of currents flowing through the branches at this node

$$\Delta i_n = \sum_{b \in B^n} I_b \tag{14}$$

$$\text{mag}_n = \frac{1}{\|B^n\|} \cdot \sum_{b \in B^n} |I_b|. \tag{15}$$

The KCL term for node n is C_n^{KCL} which can then be calculated as follows:

$$\begin{aligned} \text{err}_n &= |\Delta i_n| - (\tau_{\text{rel}} \cdot \text{mag}_n + \tau_{\text{abs}}) \\ C_n^{\text{KCL}} &= \max(0, \text{err}_n). \end{aligned} \tag{16}$$

As in detailed circuit simulation, the tolerances τ_{abs} and τ_{rel} ensure that numerical truncation when adding the magnitude of a small current to the magnitude of a large one does not adversely effect the measure of KCL correctness.

The C^{DV} terms present an alternative view of KCL. They measure the voltage *change* required at each node to make the circuit satisfy KCL. By this definition, calculating the DV terms in C^{dc} exactly would require actually solving for the correct circuit node voltage values, which is what we are trying to avoid with the relaxed-dc formulation. Instead, we estimate the DV error terms by multiplying the factored small-signal nodal admittance matrix, Y^{-1}, by the error currents

$$\Delta \underline{v} = Y^{-1} \Delta \underline{i}. \tag{17}$$

Since the circuit will typically contain nonlinear devices, the Y matrix will be a linearized estimate of the admittance taken at the present bias point and the DV errors first-order approximations to the actual voltage error. As the annealing state freezes, the DV terms will be quite accurate, but during the early stages of the anneal, the KCL terms will be the only reliable estimates of dc correctness because they are not approximations. The addition of the DV terms takes the impedance into account and provides more useful feedback to designers, since they tend to have better intuition regarding voltage errors than current errors.

Calculation of the Cost Function: The terms in $C(\underline{x})$ are calculated from the present problem state following the flow in Fig. 7. Each time a variable in $\underline{x}$ changes, circuit component values may change, the encapsulated device models be reevaluated, AWE used to recompute transfer functions, and cost function terms recomputed. Fortunately, this analysis is extremely fast, allowing a complete reevaluation for each circuit visited. The cell-level analog circuits we target have less than 100 devices, and produce small-signal models with less than 1000 elements. The bulk of the work is performed by AWE, which reduces such a small-signal model to an accurate low-order transfer function in 10 to 100 ms on a fast workstation. Previous synthesis strategies were unable to determine both a dc operating point and performance characteristics automatically.

F. Annealing Control Mechanisms

The final customizable component of the annealing algorithm is the cooling schedule. We have implemented the general purpose cooling schedule of Lam [43] as modified by Swartz [53]. This specifies T_{HOT}, *done at temperature* and *update temperature*. Our freezing criteria, the *frozen* function, which determines when the annealing has completed, was developed specifically for our analog synthesis application. The design is complete when both the discrete variables have stopped changing and the changes in the continuous variables are within their specified relative tolerances.

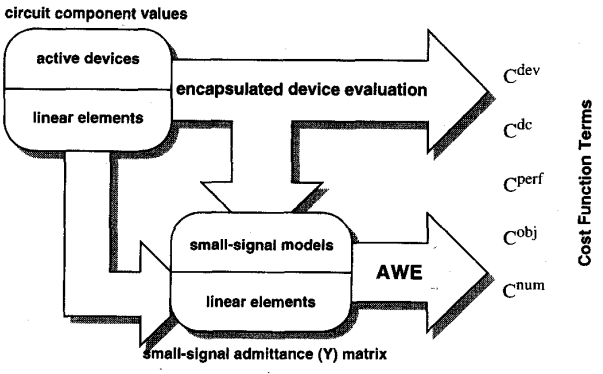


Fig. 7. Evaluation of the cost function.

G. Implications of the Relaxed-DC Formulation

Even after a complete description of our new analog circuit synthesis formulation, a few design decisions usually require further explanation. The first of these is the relaxed-dc formulation. As a result of the relaxed-dc formulation, early in the optimization process, the sum of the currents entering a given node has a significant nonzero value. An important issue to address is what it means to evaluate the performance of a circuit that is not dc-correct. One way to view this circuit is to imagine an additional current source at each node. This current source sinks the current required to ensure dc-correctness. Then, the goal of the Kirchhoff's law constraints C^{dc} is to reduce the value of these current sources to zero. When evaluating circuit performance, the fact that these current sources will not be in our final design means that our predicted performance will differ slightly from the final performance. This error factor allows us to visit many more possible circuit configurations within a given period of time, albeit evaluating each a little less accurately. As the optimization proceeds, the current sunk by these sources goes to zero and the performance prediction becomes completely accurate. This evolution is shown in Fig. 8. By the end of the annealing process, the circuit will be dc-correct within tolerances on the order of those used in circuit simulation.

A second analogy that can be used to understand how the relaxed-dc formulation behaves is to simply pretend a dc operating point solution is performed at each iteration. However, the tolerance for dc correctness is very relaxed early in the optimization process and evolves toward that of typical detailed simulations as the optimization proceeds. All simulators have numerical tolerances on dc correctness, and as a result, there is always numerical error in circuit simulation.

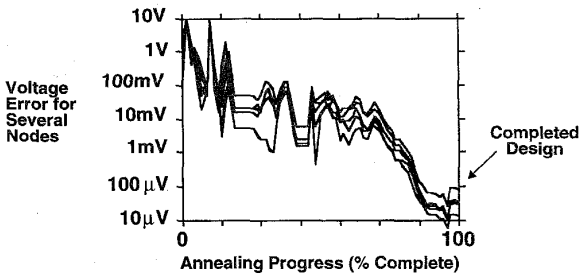


Fig. 8. Discrepancy from KCL correct voltages during optimization.

The relaxed-dc formulation simply trades increased error for increased speed of evaluation early in the optimization process.

These analogies point out a seeming inconsistency in our overall synthesis formulation. We clearly do not intend that each annealing move visits a dc-correct circuit, i.e., where the current sources ensuring dc-correctness are zero-valued. Nevertheless, we evaluate each circuit using detailed device models and highly accurate AWE techniques. Clearly this accuracy is not fully exploited early in the optimization when these Kirchhoff current law errors are substantial. Simpler models could be used in these early stages, but, practically, knowing when and how to switch from simple to accurate models would be difficult and it is unlikely significant changes in run-time would be achieved. Moreover, this accuracy is not entirely wasted because the annealer still learns much from these early circuits. For example, if we need to achieve more gain in our circuit, we probably need to increase the g_m on some critical device. We do not need a precise dc bias point to know that we must either increase that device's width, its current, or both. The optimizer can successfully make coarse decisions such as this even in the presence of the slight noise introduced by the relaxed-dc formulation.

H. Reliability Without User-Supplied Constants

The final aspect of our formulation that requires further explanation is the issue of numeric constants. Numeric algorithms in general require constants that tune the algorithm such that it reliably produces high quality solutions, and our annealer is no exception. If a numeric algorithm and the design automation tool of which it is a part are poorly designed, the constants will need to be adjusted for each new problem solved. As a result, the tool will not do a very good job of automation, since a user of the tool will spend much of his time adjusting constants, rather than designing circuits. Coupled with this automation problem is a robustness problem inherent in the choice of simulated annealing as the core of our optimization engine. Simulated annealing is a stochastic process, and each time the annealer is run it may find a slightly different trade-off between its constraints. As a result, it is not possible to guarantee that a single run will provide the best answer. However, it is important that the tool is still robust, i.e., we wish to be confident that running the annealer several (5–10) times will provide several high-quality solutions from which to choose.

Substantial effort has been spent designing our formulation and its implementation such that it is truly a robust automation tool, i.e., the user is not required to provide problem-specific constants and the tool produces a high percentage of top quality solutions. One key aspect of this algorithmic design is the use of adaptive algorithms to replace the majority of the numeric constants that would otherwise be needed within the annealer. For example the penalty terms in $C(\underline{x})$ require scalar weights to balance their contributions to the cost function. Similar to the strategies used in [54], these weights have been replaced with an adaptive algorithm. The adaptive algorithm in the annealer is based on the observation that all the penalty terms, $g(\underline{x})$, must be zero at the end of a successful annealing run. Thus, we can use a simple feedback mechanism to force each penalty term to follow a trajectory that leads from its initial value to zero; i.e., if a penalty term presently exceeds its expected value as given by its trajectory, the weight is increased and the natural annealing action focuses on reducing that penalty term in subsequent annealing steps. With this adaptive algorithm, the problem is then one of determining what the best trajectories are and setting their values. The advantage of this technique is that, as we have shown over the circuit synthesis problems we have solved with our implementation, a single set of trajectories is much more robust than a single set of individual weights. As a result, an analog circuit designer can use our analog circuit synthesis system without understanding the internal architectural details needed to adjust components of its cost function.

To set the trajectories, we treated them as independent variables in a large optimization problem solved using Powell's algorithm [55]. We refer to the large problem as "optimizing the optimizer" because the annealer was executed as the inner loop of the Powell optimization process. Each time a trajectory was perturbed, the annealer was run and the quality of the resulting circuit solution fed back to the Powell optimizer. However, because of the stochastic nature of annealing, it is insufficient to characterize a set of trajectories on a single annealing run. Instead, using a large network of workstations, the annealer was run 200 times on the same synthesis problem and a statistical analysis performed to determine the quality of a particular set of trajectories. Because of the large amount of CPU time involved, we optimized the trajectories for a small circuit problem and validated them over our benchmark suite. Verifying that this single set of trajectories provided good results across all our circuits—and will likely do so for new circuits—was essential because optimizing the optimizer consumed approximately four years of CPU time. Comparing our initial "best guess" trajectories to those that resulted from optimizing the optimizer, the number of top quality solutions for a typical synthesis problem increased from about 30% to about 80%. As a result, the careful design and tuning of dynamic algorithms within the annealer have freed the user from providing algorithmic constants and greatly improved the overall robustness of the tool. See [33] for more details on the design and optimization of these trajectories.

VI. Comparison to Related Work

In this section, we compare and contrast our new analog circuit synthesis formulation to recent related work. Our for-

mulation is unique among analog synthesis systems in several key ways. Architecturally, it is unique because of its two-step synthesis process. It includes a complete compiler that translates the problem from a compact user-oriented description into a cost function suitable for subsequent optimization. Compilation provides significant flexibility. It provides the opportunity for the user to interact with the tool in a language that is familiar to designers, yet, because the compiler produces an executable designed specifically to solve the user's problem, it also provides optimal run-time efficiency. Our formulation is also unique in its use of AWE for analog circuit synthesis. Although recently other researchers have used AWE for the synthesis of power distribution networks [56], to the best of our knowledge ours was the first tool to employ AWE for performance evaluation within an optimization loop.

Of course, our paper uses and builds upon the successes and failures of previous approaches to analog circuit synthesis. As discussed in Section II, previous systems are equation-based and thus suffer from problems with accuracy and automation. Because it is a simulation-based approach, our paper is a substantial departure from these previous approaches. However, the encapsulated device models and relaxed-dc formulation are based on similar ideas used by Maulik [16]. The key distinction is that Maulik still relies on hand-crafted equations to compute circuit performance from the parameters returned by the device models, and these equations must be hand-coded into the system for each new circuit topology.

Our formulation also borrows ideas from—but differs from—simulation-based optimization systems. As discussed in Section II, these tools are not true synthesis tools because they are starting point dependent. Our formulation avoids this problem by using simulated annealing, which is *not* starting point dependent, as the optimization algorithm and avoids the resulting efficiency problem by using AWE.

There is one other simulation-based synthesis tool of which we are aware. The tool described in [36] appeared shortly after the initial publication of our paper [32], [34]. It uses several of the same strategies: simulation is used to evaluate circuit performance and a form of simulated annealing is used for optimization. However, it relies on SPICE for the simulation algorithm which is typically 2–3 orders of magnitude slower than AWE for ac analysis [39]. To achieve reasonable run-times, it does not use transient analysis within SPICE (which is even slower than ac analysis) and relies on several heuristics for small-signal specifications. The first heuristic is to substantially reduce the number of iterations that would normally be required for optimization with simulated annealing. Results generated with this heuristic adaptation seem reasonable, but comparisons to difficult manual designs have not been presented so the efficacy of this heuristic is unclear. The second heuristic is to save the dc operating point from the previous SPICE run and use it as the starting point for the dc analysis of the next SPICE run. This is the first step toward a relaxed-dc formulation, where the dc operating point is maintained as part of the overall design state. They do not, however, take the final step toward relaxed-dc by loosening the dc requirements early in the design process. As a result, they cannot afford the CPU time to explore as large a region of the design space as is possible with our system.

VII. Results

In this section, we first describe our implementation of our new analog circuit synthesis formulation. We then present circuit synthesis results and compare them with those of previous approaches.

A. Implementation

To show the viability of our new formulation for analog circuit synthesis, we have implemented the ideas described in this paper as a pair of synthesis tools called ASTRX and OBLX. ASTRX is the circuit compiler that translates the user's circuit design problem into C code that implements $C(\underline{x})$. ASTRX then invokes the C compiler and the linker to link this cost function code with the OBLX solution library, which contains the annealer, AWE, and the encapsulated device library. Invoking the resulting executable completes the actual design process. Translation of the user's problem into C code requires only a few seconds, so run-times are completely dominated by optimization time. ASTRX and OBLX are themselves implemented as approximately 125 000 lines of C code.

The syntax used to describe the synthesis problem is the SPICE format familiar to analog designers, with the addition of a few cards to describe ASTRX/OBLX specific information, such as specifications. For our example of a simple differential amplifier design (Section IV), the complete input file (excluding process model parameters) is shown in Fig. 9. A complete description of the format can be found in [57].

A. Circuit Benchmark Suite

The primary goal of ASTRX/OBLX is to reduce the time it takes to size and bias a new circuit topology to meet performance specifications. Fig. 10 summarizes a representative selection of previous analog circuit synthesis results.[2] Here, each symbol represents synthesis results for a single circuit topology. The length of the symbol's "tail" represents the complexity of the circuit, which, as described in Section I, we quantify as the sum of the number of devices in the circuit and the number of variables for which the user asks the synthesis tool to determine values. The other axes represent metrics for accuracy and automation. The prediction error axis measures accuracy by plotting the worst case discrepancy between the synthesis tool's circuit performance predication and the predictions of a circuit simulator. The time axis measures automation by plotting the sum of the preparatory time spent by the designer and CPU time spent by the tool to synthesize a circuit for the first time. Fig. 10 reveals three distinct classes of synthesis results. The first class is on the right and contains the majority of previously published papers. Here, the synthesis tool predicts performance with

[2] Not all prior approches could be included in Fig. 10 because of the unavailability of the necessary data. In [14] where preparatory time was not published, we equated 1000 lines of circuit-specitic custom code to a month of effort.

```
Design Example                                                                    1
                                                                                  2
.param cl = 1pF                                                                   3
.param vddval=2.5                                                                 4
.param vssval=-2.5                                                                5
                                                                                  6
.subckt oa      inpos   inneg   outpos outneg nvdd    nvss                       7
M2      outpos inneg    A       A       Ne      w='W'   l='L'                      8
+       constraint1 = 'vgs - von > 0.2' constraint2 = 'vds - vdsat > 0.05'        9
M1      outneg inpos    A       A       Ne      w='W'   l='L'                      10
+       constraint1 = 'vgs - von > 0.2' constraint2 = 'vds - vdsat > 0.05'        11
M3      outpos nvb      nvdd    nvdd    Pe      w=2u    l=1.2u                     12
M4      outneg nvb      nvdd    nvdd    Pe      w=2u    l=1.2u                     13
vvb     nvb     nvss    'Vb'                                                      14
ib      A       nvss    'I'                                                       15
.ends                                                                             16
                                                                                  17
.VAR W RANGE=(1.8u,500u) GRID=1                                                   18
.VAR L RANGE=(1.2u,100u) GRID=1                                                   19
.VAR I RANGE=(1uA,1mA)   RES=0.001                                                20
.VAR Vb RANGE=(0,5)      RES=0.001                                                21
                                                                                  22
.SYNTH example                                                                    23
                                                                                  24
.OblxCkt self_bias bias                                                           25
xamp    inpos   inneg   outpos outneg nvdd    nvss    oa                         26
vdd     nvdd    0       vddval                                                    27
vss     nvss    0       vssval                                                    28
rf1     inneg   outpos  1e6                                                       29
rf2     inpos   outneg  1e6                                                       30
.spec SR value 'I/(2*(cl+xamp.m1.cdd+xamp.m3.cdd))' good = 1Meg bad = 10k         31
.endOblxCkt                                                                       32
                                                                                  33
.OblxCkt openloop awe                                                             34
.bias self_bias                                                                   35
xamp    inpos   inneg   outpos outneg nvdd    nvss    oa                         36
vdd     nvdd    0       vddval                                                    37
vss     nvss    0       vssval                                                    38
vin     inpos   0       0       ac      1                                         39
ein     inneg   0       0       inpos   1                                         40
cl1     outpos  0       Cl                                                        41
cl2     outneg  0       Cl                                                        42
.pz tf V(outpos) vin                                                              43
.spec   ugf     value 'unity_gain_freq(tf)'  good = 1Meg    bad = 10k             44
.obj    Adm     value 'dc_gain(tf)'          good = 1000    bad = 10              45
.endOblxCkt                                                                       46
                                                                                  47
.END                                                                              48
```

Fig. 9. Design example: ASTRX input file. (Process parameters are not included.)

reasonable accuracy, but only because a designer has spent months to years of preparatory time deriving the circuit performance equations. The second group of results, those on the left, trade reduced preparatory effort for substantially reduced circuit performance prediction accuracy. Finally, the center group of results is for ASTRX/OBLX. In contrast to the other two groups, generating each new design with ASTRX/OBLX typically involved an afternoon of preparation followed by 5–10 annealing runs performed overnight, yet produced designs that matched simulation with at least as much accuracy as the best prior approaches.

For all eight circuit topologies discussed in this paper, Table II lists information about the input file used to describe the problem to ASTRX and the resulting cost function and C code generated. Note that five of these topologies (Simple OTA, OTA, Two-Stage, Folded-Cascode, and Comparator) form our benchmark suite, and cover essentially all[3] synthesis results published at the time of this writing. The limited range and performance of these prior results is perhaps the best indicator that the first-time effort to design a circuit has always been a substantial barrier to obtaining a broader range of results. The first two rows of the table give the number of lines required for each synthesis problem description. The number of lines is reported as two separate values: 1) the lines required for the netlists of the circuit under design and the test jigs (about the number of lines that would be required to simulate the circuit with SPICE), and 2) the lines for the independent variables and performance specifications (lines specific to ASTRX/OBLX). Note that this is a modest amount of input, most of which is the netlist information required for any simulation deck. The synthesis-specific information is predominantly a list of the variables the user wishes ASTRX/OBLX to determine and the performance specifications. For small circuits, creating the input to ASTRX usually takes a few hours (compared to the weeks/months required to add a new circuit to other synthesis tools). For

[3] We make the reasonable assumption that a circuit topology can represent topologies that vary in only minor detail.

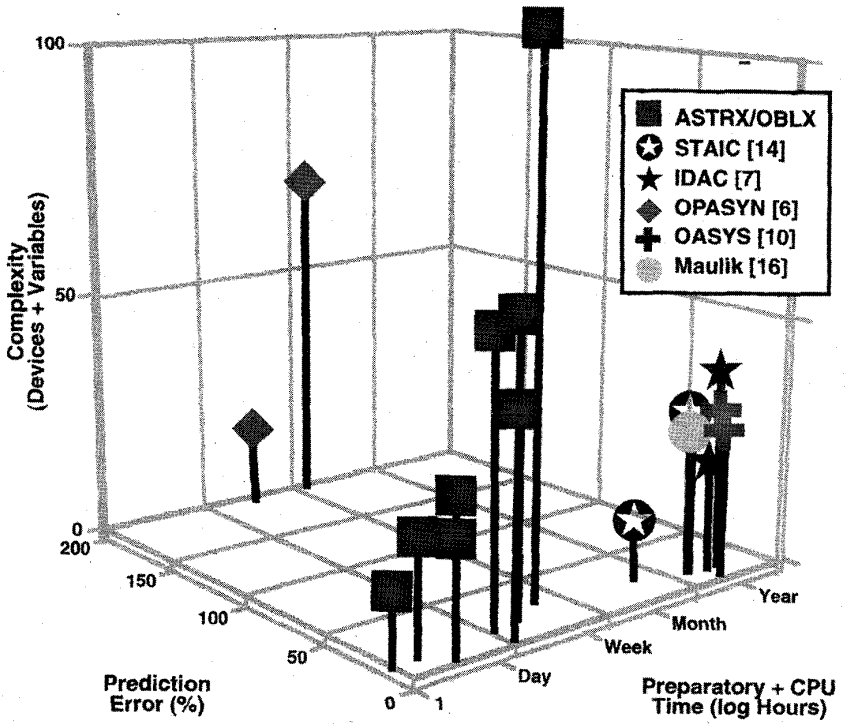


Fig. 10. Complexity, error, and first time design effort. ASTRX/OBLX compared with prior approaches.

TABLE II
RESULT OF ASTRX'S ANALYSES

		Circuit							
		Simple OTA	OTA	Two-Stage	Folded Cascode	Comparator	BiCMOS Two-Stage	Novel Folded Cascode	2x Gain Stage
Input (lines)	Netlist/Models	30	34	43	65	131	39	68	189
	Synth. Specific	28	33	40	56	68	33	51	65
x	User-Supplied	7	11	19	28	19	12	27	39
	Node Voltages	14	24	26	70	57	26	84	215
C (x)	Terms	56	85	88	212	169	86	246	483
	Lines of C	1443	1809	1894	3408	3088	1723	3960	7173
Circuits	Circuit Type[a]: nodes, elements	B: 20, 31 A: 20, 67	B: 28, 49 A: 29, 114	B: 34, 54 A: 33, 118	B: 75, 138 A: 75, 324	B: 65, 126 A: 63, 265 A: 64, 266 A: 29, 115	B: 33, 54 A: 32, 105	B: 90, 167 A: 90, 395	B: 219, 450 A: 152, 693 A: 152, 693 A: 90, 395

a. Type 'A' is a linearized, small-signal AWE circuit. Type 'B' is a bias circuit.

the 2× gain stage, creating and checking the topology, and specifications took two to three days. The third and fourth rows of Table II show the number of independent variables that must be manipulated by OBLX during optimization. Recall that since the relaxed-dc formulation requires Kirchhoff's laws to be explicitly constrained, OBLX must also manipulate most of the circuit node voltages. As shown in the fourth line of Table II, as a result of internal nodes in the device models we employ, these added variables typically outnumber the user-specified variables. This dimensional explosion is substantially alleviated by the inclusion of Newton–Raphson moves as one of the circuit perturbations OBLX can use during simulated annealing (see Section V.D.). The fifth and sixth rows of the table show the other results of ASTRX's analysis of the input description: the number of terms in the cost function OBLX will optimize and the number of lines of C code automatically generated by ASTRX for this synthesis problem. Recall that ASTRX compiles the problem description, generating the cost function as C code then compiling and linking it with OBLX. Finally, Table II shows the size of the linearized small-signal test jig circuit(s) generated by ASTRX to be evaluated by AWE for each new circuit configuration and the size of the bias circuits generated by ASTRX using the large-signal models for the nonlinear devices.

TABLE III
BASIC SYNTHESIS RESULTS, BSIM AND GUMMEL–POON MODELS, 1.2 μm PROCESS

	Specification: OBLX / Simulation				
Attribute	**Simple OTA**	**OTA**	**Two-Stage**	**Folded Cascode**	**BiCMOS Two-Stage**
Cload (pF)	1	1	1	1.25	1
Vdd	5	5	5	5	5
dc gain (dB)	↑[a]: 36.6 / 36.6	↑: 40.4 / 40.2	≥60: 66.4 / 66.4	≥70: 70.1 / 70.1	↑: 99.1 / 99.1
gain bandwidth (MHz)	≥50: 50.1 / 50.6	≥25: 25.0 / 25.4	≥10: 10.6 / 10.6	↑: 72.4 / 72.1	≥50: 73.7 / 75.1
phase margin (°)	≥60: 71.4 / 74.8	≥45: 57.9 / 57.8	≥45: 87.3 / 86.5	≥60: 80.0 / 80.0	≥45: 45.2 / 49.6
PSRR (Vss)	≥20: 21.9 / 21.9	≥40: 42.1 / 42.0	≥20: 31.0 / 30.9	≥105: 107 / 107	≥60: 78.9 / 79.0
PSRR (Vdd)	≥20: 36.8 / 36.8	≥40: 52.8 / 52.8	≥40: 45.8 / 45.8	≥105: 125 / 125	≥40: 52.2 / 52.2
output swing (V)	≥2.3: 3.7 / 3.6	≥2.5: 4.0 / 4.0	≥2: 2.7 / 2.8	≥1.0: ±1.5 / ±1.5	≥2: 3.3 / 4.0
slew rate (V/μs)	≥10: 130 / 131	≥10: 51.6 / 48.2	≥2: 3.8 / 4.0	≥50: 67 / 57	≥10: 10 / 9.5
active area ($10^3\mu^2$)	↓: 2.8	↓: 0.9	↓: 2.1	↓: 46	↓: 11.9
static power (mW)	≤1: 0.72 / 0.72	≤1: 0.33 / 0.34	≤1: 0.16 / 0.16	≤15: 10 / 10	≤20: 1.3 / 1.5
time/ckt. eval (ms)	36	37	38	116	38
CPU time (min./run)	6	9	16	120	12

a. ↑ means maximize, while ↓ means minimize.

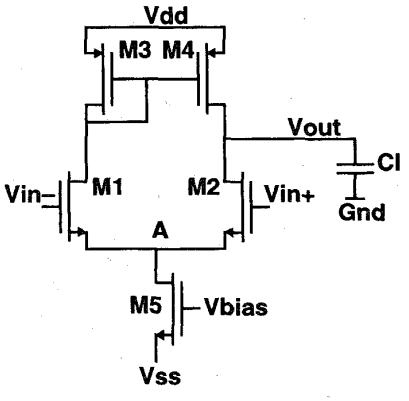


Fig. 11. Simple OTA schematic.

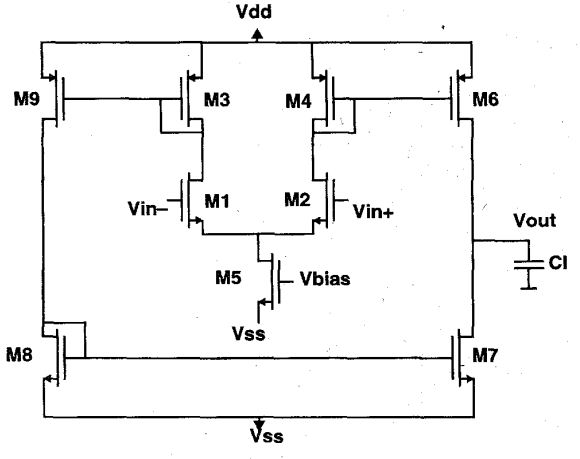


Fig. 12. OTA schematic.

The schematics for all our circuits are shown in Figs. 11–18 and the basic synthesis results for the benchmark suite are in Table III. CPU times given are on an IBM RS/6000-550 (about 60 MIPS). It is impossible to compare directly circuit performance of these ASTRX/OBLX synthesized circuits with that of circuits synthesized with other tools because of the unavailability of device model parameters to describe the processes for which they were designed. As a result, we compare the *accuracy* of ASTRX/OBLX to that of previous approaches. Because of the simplifications made during circuit analysis, results from equation-based synthesis tools differ from simulation by as much as 200% (see Fig. 10). In contrast, for the small-signal specifications where AWE predicts performance, ASTRX/OBLX results match simulation almost exactly. By simulating linearized versions of these circuits, we have determined that these minor differences are due to differences between the models used during simulation with HSPICE [58] and our models adopted from SPICE 3 [24]. For nonlinear performance specifications, such as slew rate, we used first-order equations written by inspection. This was done to validate our assertion that most large-signal specifications can be readily, yet realistically, handled despite the lack of transient simulation within the present implementation of ASTRX/OBLX. The accuracy of these is better than similar equations in completely equation-based techniques because circuit parameters used to write the equations, such as branch currents and device parasitics, are updated automatically by OBLX as the circuit evolves.

Presently ASTRX/OBLX employs three different encapsulated device models: the level 3 MOS model from SPICE [24], the BSIM MOS model [23] and the Gummel–Poon model for BJT devices [59]. Because the encapsulated model interface in OBLX was designed to be compatible with SPICE, adopting these models from SPICE source code required little more than the addition of topological information. To demonstrate the importance of supporting different models, we synthesized the same circuit (Simple OTA) with three different model/process combinations: BSIM/2μ, BSIM/1.2μ, and MOS3/1.2μ. AS-

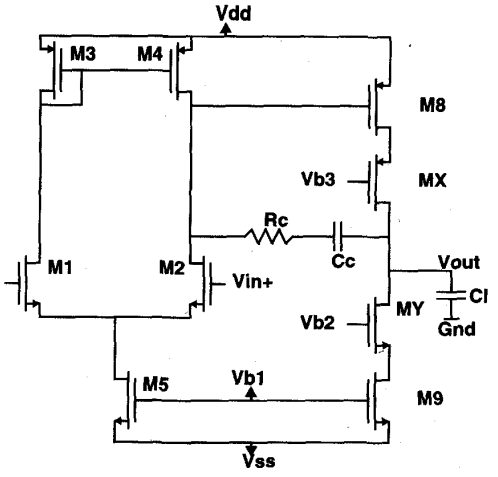


Fig. 13. Two-stage schematic.

TABLE IV
COMPARISON WITH MANUAL DESIGN FOR CIRCUIT NOVEL FOLDED CASCODE

Attribute	Manual Design	Automatic Re-Synthesis Spec:OBLX / Sim
Cload (pF)	1	1
Vdd (V)	5	5
dc gain (dB)	71.2	≥71.2: 82 / 82
gain bandwidth (MHz)	47.8	↑[a]: 89 / 89
phase margin (°)	77.4	≥60: 91 / 91
PSRR (Vss)	92.6	≥93: 112 / 112
PSRR (Vdd)	72.3	≥73: 77 / 77
output swing (V)	± 1.4	≥±1.4: 1.4 / 1.3
slew rate (V/μs)	76.8	≥76: 92 / 87
active area ($10^3\mu^2$)	68.7	↓: 56
static power (mW)	9.0	≤25.0: 12 / 12
time/ckt. eval (ms)		83
CPU (min. / run)		116

a. ↑ means maximize, while ↓ means minimize.

TRX/OBLX was given (and achieved) the same specifications for each, but was told to minimize active area. The resulting areas are shown in Fig. 20. As expected, the BSIM/2μ design required the largest area ($580\mu^2$). But, surprisingly, the two designs for the *same* 1.2μ process also differed substantially in area: $300\mu^2$ for BSIM and $140\mu^2$ for MOS3. These models differ in their performance predictions, even though they are both intended to model the same underlying fabrication process. Clearly the choice of device model greatly effects circuit performance prediction accuracy. As a final experiment to show the utility of encapsulated device models, we designed a BiCMOS two-stage amplifier, which shows the ability of ASTRX/OBLX to handle a mix of MOS and bipolar devices. Synthesis and simulation results appear as the last column of Table III. Here again, performance predictions match detailed circuit simulations, confirming the importance of encapsulated devices for both accuracy and generality.

TABLE V
COMPARATOR SYNTHESIS RESULTS

Key Attribute	Specification:	OBLX/Simulation
Vdd		5V
stage 1 settling, 0.1%, (ns)	≤5:	4.5 / 4.7
stage 2 settling, 0.1%, (ns)	≤5:	4.6 / 4.8
stage 1 slew rate (V/μs)	≥800:	800 / 750
stage 2 slew rate (V/μs)	≥800:	800 / 620
latch 1, positive pole (MHz)	≥200:	430 / 450
latch 2, positive pole (MHz)	≥200:	360 / 330
error rate, (1 in years)	≥10:	8E+24
active area ($10^3\mu^2$)	↓[a]:	1.6
static power (mW)	↓:	2.7 / 2.7
time/ckt eval (ms)		136
CPU time (min. / run)		97

a. ↓ means minimize.

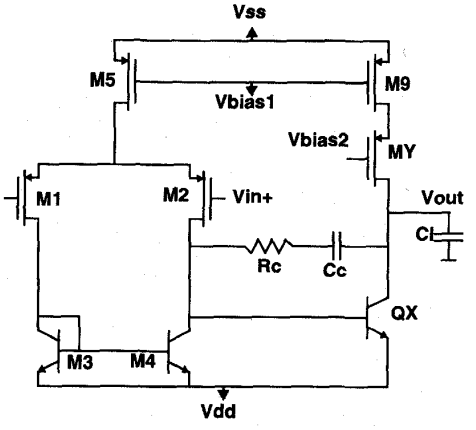


Fig. 14. BiCMOS two-stage schematic.

B. Comparison to Manual

Our next example shows the ability of ASTRX/OBLX to design difficult circuits and achieve results similar to those obtained by manual design. This circuit, a novel folded cascode fully differential opamp, shown in Fig. 18, is a new high-performance design recently published in [60] and as such is a significant test for any synthesis tool because the performance equations cannot be looked up in a textbook. Moreover, the performance of the circuit is difficult to express analytically, and as many as six poles and zeros may nontrivially effect the frequency response near the unity gain point. Table IV is a comparison of a redesign of this circuit using ASTRX/OBLX with the highly optimized manual design for the *same* 2μ process. Surprisingly, ASTRX/OBLX finds a design with higher nominal bandwidth at the cost of less area. Although we are pleased with the ability of OBLX to find this corner of the design space, this does not mean that ASTRX/OBLX outperformed the manual designer. In fact, the manual designer was willing to trade *nominal* performance for better estimated yield and performance over varying operating conditions.

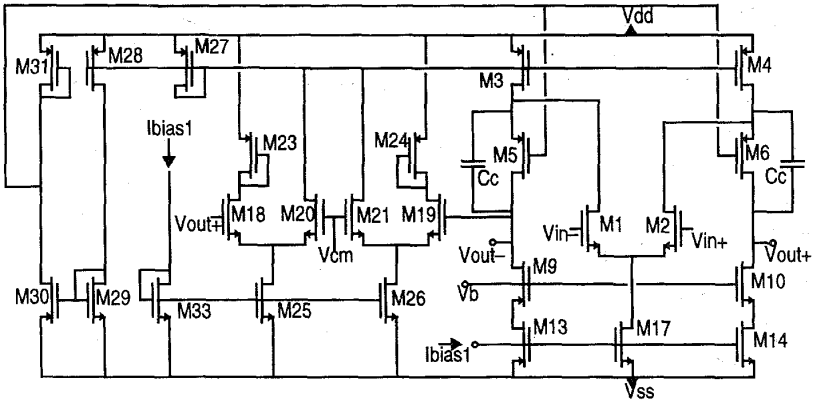


Fig. 15. Folded cascode schematic.

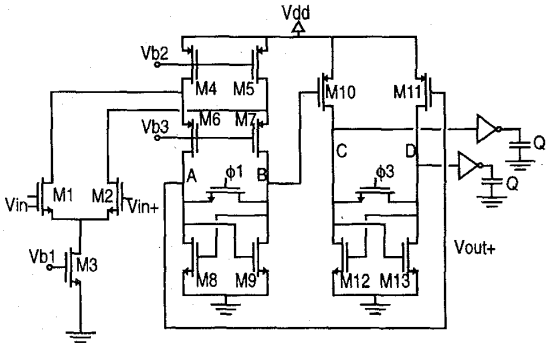


Fig. 16. Comparator schematic.

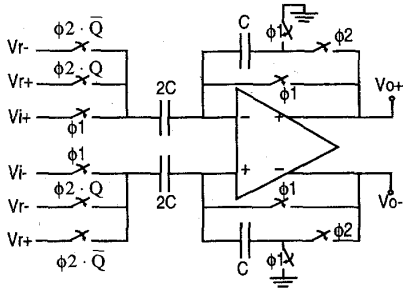


Fig. 17. 2× gain stage schematic. The amplifier is the Novel Folded Cascode (Fig. 18), and the switches are MOS pairs.

Adding this ability to ASTRX/OBLX is the subject of ongoing research and preliminary results are reported in [37].

C. Pushing the Limits of Complexity and Generality

Our final result shows the utility of ASTRX/OBLX when confronted with the much more complex task of designing a pipelined A/D converter. This is an important test because it addresses the issues of generality and complexity. The pipelined A/D topology converter we selected, Fig. 19, employs two cells: a comparator (Fig. 16) and a 2× gain stage (Fig. 17). This is a test of generality because these cells display important nonlinear behavioral characteristics whose modeling

TABLE VI
SWIT-CAP 2× GAIN STAGE SYNTHESIS RESULTS

Key Attribute	Specification:	OBLX/Simulation
Vdd		5V
settling at output (ns)	≤100:	93 / 98
settling at amp inputs (ns)	≤100:	42 / 61
input range (V)	≥±0.5:	1.6 / 1.6
output range (V)	≥±0.5:	0.8 / 0.7
common mode gain (dB)	≤-10:	-33 / -33
active area ($10^3\mu^2$)	↓[a]:	58
static power (mW)	↓:	21 / 21
time/ckt eval (ms)		158
CPU time (hrs. / run)		11.8

a. ↓ means minimize.

in ASTRX/OBLX is not as straightforward as the performance characteristics of amplifiers and other linear cells. This is a test of complexity because, using the metric of devices plus designable parameters, these cells are 2–3× more complex than those published previously. To aid noise rejection in a mixed-signal environment, the A/D converter uses a fully differential structure. The 2× gain stage is implemented as a switched-capacitor circuit, and the input switches provide a convenient method to perform the multiplexing and subtraction needed to complete the design. The 2× gain stage also employs an operational amplifier (Fig. 18) [60]. It is important to note that ASTRX/OBLX optimizes the *entire* gain cell as a *single* circuit. This allows the optimizer to explore crucial tradeoffs between the sizes of the switches, capacitors and amplifier devices. This also yields a very large optimization problem, which contributes to the lack of published synthesis results of this complexity. However, the ability to handle problems of this magnitude is fundamental for industrial design situations where the designer naturally works with cells of this complexity. Other characteristics of the two circuit design problems are shown in Table II.

Sample ASTRX/OBLX synthesis results for a comparator appear in Table V and results for a sample 2× gain stage appear in Table VI. These results are for a 1.2μ MOS process

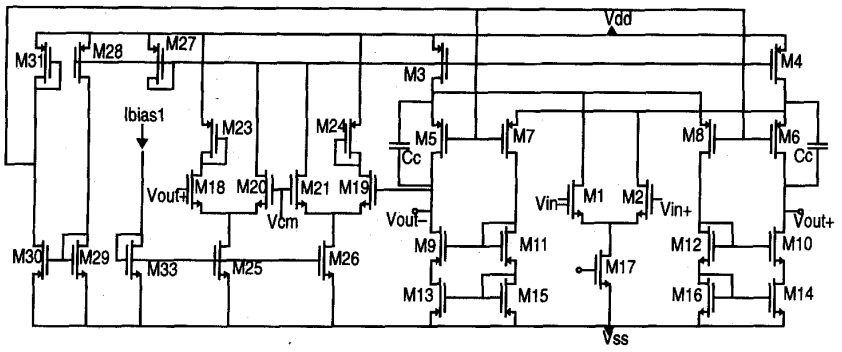


Fig. 18. Novel folded cascode schematic [60].

and use a BSIM model for the devices. Reported run-times are again on an IBM RS/6000-550 (about 60 MIPS), and simulation results were obtained with HSPICE. As before, there is a close correspondence between OBLX prediction and simulation.

One key benefit of circuit synthesis is the ability to explore the design space for a given circuit and process, quantifying the interactions between competing circuit performance constraints. Fig. 21 shows the results of several ASTRX/OBLX runs for the comparator of Fig. 16. Each point on the graph is a different complete circuit design, obtained by increasing the clock frequency specification and asking ASTRX/OBLX to minimize static power. The graph shows the expected increase in static power consumption as a function of the circuit's maximum clock frequency.

Finally, Fig. 22 shows the simulated input/output response at 3 MHz for a single stage of our completed A/D converter formed by connecting the two synthesized cells. The input–output response follows the saw-tooth pattern expected of an A/D converter stage. The 3 MHz performance is a few years behind the state of the art, but we consider this perhaps modest performance to be quite acceptable for the first fully automatic design of cells of this complexity.

VIII. Conclusion and Future Work

Our experience with ASTRX/OBLX as it has evolved over the past few years has given us considerable insight into the practical aspects of the system's use and the strengths and weaknesses of both its underlying ideas and their implementation. In practice, since the input format was designed to be very familiar to users of SPICE, analog designers need little assistance before they can begin experimenting with ASTRX/OBLX, and several colleagues at Carnegie Mellon have successfully used the tool. However, obtaining usable circuits generally requires some experience and patience. The typical failure mode is that the synthesis problem is underconstrained, and the optimizer finds a circuit that meets all the specifications given and yet is not usable. For example, if no output swing specification is given, OBLX will exploit this omission and design an amplifier that meets all the given specifications, but whose output devices will be pushed out of saturation by almost any change in the input. Correctly

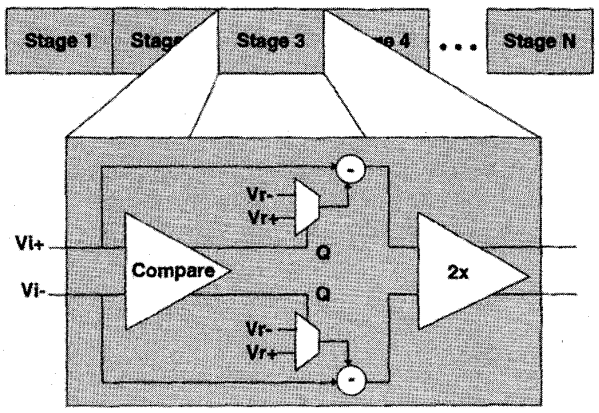


Fig. 19. Pipelined A/D converter topology.

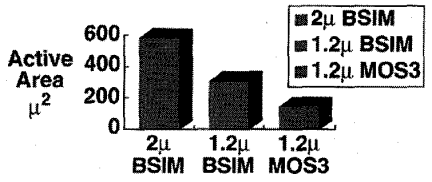


Fig. 20. Active area for the circuit simple OTA synthesized for three different process/model combinations.

specifying the circuit synthesis task is usually overcome with a few iterations through the synthesize/verify cycle. This process of tuning specifications could likely be enhanced by some form of debugger that could point toward the cause of input specification problems, but as with conventional programming languages, creation of sophisticated development tools will follow only when the language is well established.

There are several ways that ASTRX/OBLX itself could be extended, and some of these are the focus of ongoing research. Three general areas of improvement are the speed and scope of circuit performance evaluation, automatic topology selection, and hierarchical design. Faster and more powerful circuit simulation is itself an open research area. Fast evaluation with AWE makes ASTRX/OBLX practical, yet it limits the information a

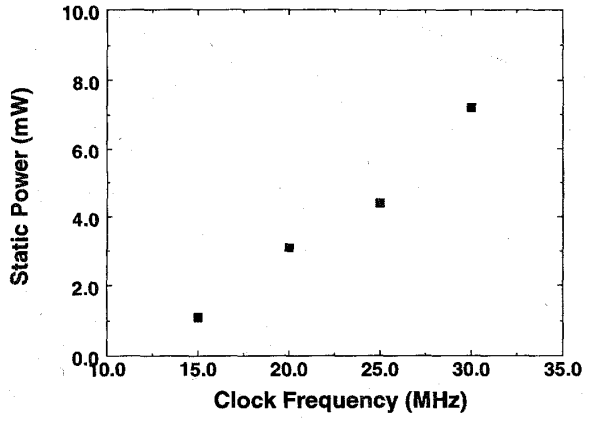


Fig. 21. Clock frequency versus static power for the comparator of Fig. 16.

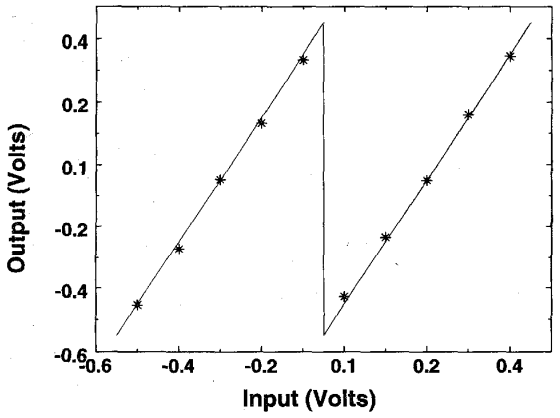


Fig. 22. Input/output response of pipeline stage.

designer can obtain about the circuit under design. The AWE techniques we employ do not work for transient specifications, and conventional transient simulation is impractical because it is many orders of magnitude too slow. This forces the designer to write some equations. Although first-order equations here are not difficult to derive, if fast transient simulation were possible, it would be a better solution. In the near term, an area in which the scope of ASTRX/OBLX circuit evaluation can be improved is manufacturablility. Since manufacturablility concerns are a critical aspect of any industrial circuit design, adding manufacturablility evaluation to ASTRX/OBLX is a critical aspect of creating a synthesis tool usable in an industrial setting. Ongoing work in this area is described in [37]. Unfortunately, adding more power and flexibility to circuit evaluation within ATRX/OBLX exacts a penalty in run-time, making synthesis of large circuits less practical. In part, the continuing trend of ever-increasing CPU speeds will alleviate this problem and make more sophisticated circuit evaluation techniques more practical in a desktop environment in the next few years.

Determining the topology—the interconnection of transistors and other circuit devices—is the other component of analog circuit design, and automating this task is another area of ongoing work with ASTRX/ OBLX. The key challenge is to extend ASTRX/OBLX while continuing to ensure that it does not contain any hard-coded circuit-specific knowledge. Our strategy is to perform simultaneous topology selection and sizing via optimization, as first introduced in [16].

Finally, a completely different way in which ASTRX/OBLX could be expanded would be to support larger circuits via hierarchy and macromodeling. Basically, this would involve using a macromodeling technique to convert a complete circuit such as an opamp into the equivalent of an encapsulated device model [61]. These macromodeled circuits would then become the atomic building blocks of a larger circuit structure such as a filter or A/D converter.

We have presented ASTRX/OBLX, tools that accurately size high-performance analog circuits to meet user-supplied specifications, but do not require prohibitive preparatory effort for each new circuit topology. For a suite of benchmark analog circuits that covers nearly all previously published synthesis results, we have validated our formulation by showing that ASTRX/OBLX requires several orders of magnitude less preparatory effort, yet can predict results more accurately. By comparing to a novel manual design, we have also shown that ASTRX/OBLX can handle difficult-to-design circuits and produce circuits comparable to those designed manually. Finally, by designing the cells of a pipelined A/D converter, we have shown that ASTRX/OBLX can successfully generate designs for problems of industrial complexity.

ACKNOWLEDGMENT

The authors wish to thank their colleagues whose critical analyses and insightful discussions have helped shape this paper. In particular, the authors wish to thank S. Kirkpatrick of IBM, and their coresearchers at Carnegie Mellon: R. Rohrer and his AWE group, R. Harjani, P. C. Maulik, B. Stanisic, and particularly, T. Mukherjee.

REFERENCES

[1] R. Harjani, "OASYS: A framework for analog circuit synthesis," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, 1989.
[2] P. G. Gray, "Analog IC's in the submicron era: Trends and perspectives," *IEEE IEDM Dig. Tech. Papers*, pp. 5–9, 1987.
[3] R. K. Brayton, A. L. Sangiovanni-Vincentelli, and G. D. Hachtel, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, pp. 264–300, Feb. 1990.
[4] E. S. Kuh and T. Ohtsuki, "Recent advances in VLSI layout," *Proc. IEEE*, vol. 78, pp. 237–263, Feb. 1990.
[5] G. Gielen, H. C. Walscharts, and W. C. Sansen, "Analog circuit design optimization based on symbolic simulation and simulated annealing," *IEEE J. Solid-State Circuits*, vol. 25, pp. 707–713, June 1990.
[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, 113–125, Feb. 1990.
[7] M. G. R. DeGrauwe, B. L. A. G. Goffart, C. Meixenberger, M. L. A. Pierre, J. B. Litsios, J. Rijmenants, O. J. A. P. Nys, E. Dijkstra, B. Joss, M. K. C. M. Meyvaert, T. R. Schwarz, and M. D. Pardoen, "Toward an analog system design environment," *IEEE J. Solid-State Circuits*, vol. 24, pp. 659–672, June 1989.
[8] M. G. R. DeGrauwe, O. Nys, E. Dijkstra, J. Rijmenants, S. Bitz, B. L. A. G. Goffart, E. A. Vittoz, S. Cserveny, C. Meixenberger, G. van der Stappen, and H. J. Oguey, "IDAC: An interactive design tool for analog CMOS circuits," *IEEE J. Solid-State Circuits*, vol. SC-22, pp. 1106–1116, Dec. 1987.

[9] F. El-Turky and E. E. Perry, "BLADES: An artificial intelligence approach to analog circuit design," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 680–691, June 1989.

[10] 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.

[11] B. J. Sheu, A. H. Fung, and Y. Lai, "A knowledge-based approach to analog IC design," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 256–258, 1988.

[12] E. Berkcan, M. d'Abreu, and W. Laughton, "Analog compilation based on successive decompositions," in *Proc. 25th ACM/IEEE Design Automation Conf.*, 1988, pp. 369–375.

[13] E. S. Ochotta, "The OASYS virtual machine: Formalizing the OASYS analog synthesis framework," M.S. thesis, Rep. #89-25, Carnegie Mellon University, Pittsburgh, PA, Mar. 1989.

[14] J. P. Harvey, M. I. Elmasry, and B. Leung, "STAIC: An interactive framework for synthesizing CMOS and BiCMOS analog circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1402–1418, Nov. 1992.

[15] J. Jongsma, C. Meixenberger, B. Goffart, J. Litsios, M. Pierre, S. Seda, G. Di Domenico, P. Deck, L. Menevaut, and M. Degrauwe, "An open design tool for analog circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*, June 1991, pp. 2000–2003.

[16] P. C. Maulik, L. R. Carley, and R. A. Rutenbar, "A mixed-integer nonlinear programming approach to analog circuit synthesis," in *Proc. Design Automation Conf.*, June 1992, pp. 693–703.

[17] H. Onodera, H. Kanbara, and K. Tamaru, "Operational-amplifier compilation with performance optimization," *IEEE J. Solid-State Circuits*, vol. 25, pp 466–473, Apr. 1990.

[18] C. Toumazou, C. A. Makris, and C. M. Berrah, "ISAID-A methodology for automated analog IC design," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 531–533.

[19] S. Seda, M. DeGrauwe, and W. Fichtner, "Lazy-expansion of symbolic expression approximation in SYNAP," in *Proc. IEEE/ACM Int. Conf. CAD*, Nov. 1992, pp. 310–317.

[20] G. Gielen, H. C. Walscharts, and W. C. Sansen, "ISAAC: A symbolic simulation for analog integrated circuits," *IEEE J. Solid-State Circuits*, vol. 24, pp. 1587–1597, Dec. 1989.

[21] Q. Yu and C. Sechen, "Approximate symbolic analysis of large analog integrated circuits," in *Proc. IEEE Int. Conf. CAD*, Nov. 1994, pp. 664–672.

[22] P. Wambacq, F. V. Fernandez, G. Gielen, and W. Sansen, "Efficient symbolic computation of approximated small-signal characteristics," in *Proc. Custom Integrated Circuits Conf.*, vol. 21, no. 5, May 1994, pp. 1–4.

[23] B. Sheu, J. H. Shieh, and M. Patil, "BSIM: Berkeley short-channel IGFET model for MOS transistors," *IEEE J. Solid-State Circuits*, vol. SC-22, pp. 558–566, Aug. 1987.

[24] B. Johnson, T. Quarles, *et al.*, "SPICE version 3e user's manual," Univ. California, Berkeley, Tech. Rep., Apr. 1991.

[25] R. A. Rohrer, "Fully automatic network design by digital computer, preliminary considerations," *Proc. IEEE*, vol. 55, pp. 1929–1939, Nov. 1967.

[26] W. Nye, D. C. Riley, A. Sangiovanni-Vincentelli, and A. L. Tits, "DELIGHT.SPICE: An optimization-based system for the design of integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 501–518, Apr. 1988.

[27] J.-M. Shyu and A. Sangiovanni-Vincentelli, "ECSTASY: A new environment for IC design optimization," in *Proc. IEEE Int. Conf. CAD*, Nov. 1988, pp. 484–487.

[28] D. E. Hocevar, R. Arora, U. Dasgupta, S. Dasgupta, N. Subramanyam, and S. Kashyap, "A usable circuit optimizer for designers," in *Proc. IEEE Int. Conf. CAD*, Nov. 1990, pp. 290–293.

[29] L. Nagle and R. Rohrer, "Computer analysis of nonlinear circuits, excluding radiation (CANCER)," *IEEE J. Solid-State Circuits*, vol. SC-6, pp. 166–182, Aug. 1971.

[30] L. Nagle, "SPICE2: A computer program to simulate semiconductor circuits," Univ. California, Berkeley, Memo. UCB/ERL-M520, May 1975.

[31] W. Nye *et al.*, "DELIGHT.SPICE user's guide," Dept. EECS, Univ. California, Berkeley, May 1984.

[32] E. S. Ochotta, R. A. Rutenbar, and L. R. Carley, "Equation-free synthesis of high-performance linear analog circuits," in *Proc. Brown/MIT Adv. Res. VLSI and Parallel Syst.* Providence, RI: MIT, Mar. 1992, pp. 129–143.

[33] E. S. Ochotta, "Synthesis of high-performance analog cells in ASTRX/OBLX," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, Feb. 1994.

[34] E. S. Ochotta, L. R. Carley, and R. A. Rutenbar, "Analog circuit synthesis for large, realistic cells: Designing a pipelined A/D converter with ASTRX/OBLX," in *Proc. Custom Integrated Circuits Conf.*, vol. 15, no. 4, May 1994, pp. 1–4.

[35] E. S. Ochotta, R. A. Rutenbar, and L. R. Carley, "ASTRX/OBLX: Tools for rapid synthesis of high-performance analog circuits," in *Proc. IEEE/ACM Design Automation Conf.*, June 1994, pp. 24–30.

[36] F. Medeiro, F. V. Fernandez, R. Dominguez-Castro, and A. Rodriguez-Vazquez., "A statistical optimization-based approach for automated sizing of analog circuits," in *Proc. IEEE Int. Conf. CAD*, Nov. 1994, pp. 594–597.

[37] T. Mukherjee, L. R. Carley, and R. A. Rutenbar, "Synthesis of manufacturable analog circuits," in *Proc. IEEE Int. Conf. CAD*, Nov. 1994, pp. 586–593.

[38] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 352–366, Apr. 1990.

[39] V. Raghavan, R. A. Rohrer, M. M. Alabeyi, J. E. Bracken, J. Y. Lee, and L. T. Pillage, "AWE inspired," in *Proc. IEEE Custom Integrated Circuits Conf.*, vol. 18, May 1993, pp. 1–8.

[40] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 13, 1983.

[41] F. Romeo and A. Sangiovanni-Vincintelli, "A theoretical framework for simulated annealing," *Algorithmica*, vol. 6, pp. 302–345, 1991.

[42] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley, "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE J. Solid-State Circuits*, vol. 26, pp. 330–342, Mar. 1991.

[43] J. Lam and J. M. Delosme, "Performance of a new annealing schedule," in *Proc. 25th ACM/IEEE Design Automation Conf.*, 1988, pp. 306–311.

[44] G. W. Rhyne and M. B. Steer, "Comments on 'Simulation of nonlinear circuits in the frequency domain," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 927–928, Aug. 1989.

[45] K. S. Kundert and A. Sangiovanni-Vincentelli, "Reply to: Comments on 'Simulation of nonlinear circuits in the frequency domain,'" *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 928–929, Aug. 1989.

[46] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley, 1987.

[47] J. Vlach and K. Singal, *Computer Methods for Circuit Analysis and Design*. Princeton, NJ: van Nostrand Reinhold, 1983.

[48] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, and A. H. Teller, "Equation of state calculations by fast computer machines," *J. Chem. Phys.*, vol. 21, p. 1087, 1953.

[49] S. Hustin and A. Sangiovanni-Vincentelli, "TIM, a new standard cell placement program based on the simulated annealing algorithm," presented at the IEEE Physical Design Workshop on Placement and Floorplanning, Hilton Head, SC, Apr. 1987.

[50] D. Vanderbilt and G. Louie, "A Monte Carlo simulated annealing approach to optimization over continuous variables," *J. Comput. Phys.*, vol. 56, pp. 259–271, 1984.

[51] G. L. Bilbro and W. E. Snyder, "Optimization of functions with many minima," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 840–849, July/Aug. 1991.

[52] S. B. Gelfand and S. K. Mitter, "Simulated annealing type algorithms for multivariate optimization," *Algorithmica*, vol. 6, pp. 419–436, 1991.

[53] W. Swartz and C. Sechen, "New algorithms for the placement and routing of macrocells," in *Proc. IEEE Int. Conf. CAD*, Nov. 1990, pp. 336–339.

[54] C. Sechen and K. Lee, "An improved simulated annealing algorithm for row-based placement," in *Proc. IEEE/ACM Int. Conf. CAD*, 1987, pp. 478–481.

[55] M. J. D. Powell, "A view of minimization algorithms that do not require derivatives," *ACM Trans. Math Software*, pp. 197–107, 1975.

[56] B. R. Stanisic, R. A. Rutenbar, and L. R. Carley, "Mixed-signal noise-decoupling via simultaneous power distribution design and cell customization in rail," in *Proc. Custom Integrated Circuits Conf.*, vol. 24, no. 2, May 1994, pp. 1–4.

[57] E. S. Ochotta, "User's Guide to ASTRX/OBLX," ECE Dept., Carnegie Mellon University, Pittsburgh, PA, Rep. CAD 94-36, July 1994.

[58] HSPICE manual, Metasoft Corp., 1990.

[59] H. K. Gummel and H. C. Poon, "An integral charge control model of bipolar transistors," *Bell Syst. Tech. J.*, pp. 827–851, May 1970.

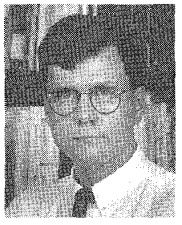
[60] K. Nakamura and L. R. Carley, "A current-based positive-feedback technique for efficient cascode bootstrapping," in *Proc. VLSI Circuits Symp.*, June 1991, pp. 107–108.

[61] J. Shao and R. Harjani, "Macromodeling of analog circuits for hierarchical circuit design," in *Proc. 1994 IEEE Int. Conf. CAD*, Nov. 1994, pp. 656–663.

Emil S. Ochotta (S'88–M'92) received the B.Sc. degree in computer engineering from the University of Alberta, Edmonton, Canada, in 1987. He received the M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1989 and 1994, respectively.

He is presently a Senior Systems Engineer at Xilinx, Inc., where he is developing new ideas for hardware and software in the field programmable gate array industry. His research interests include programming and hardware description languages, biomedical engineering, and CAD tools to support analog design and FPGA's.

Dr. Ochotta received a Best Paper award at the Semiconductor Research Corporation TECHCON conference in 1993. He was awarded the APEGGA (Association of Professional Engineers, Geologists, and Geophysicists of Alberta) Gold Medal for graduating first in his class and is a member of ACM and Sigma Xi.



Rob A. Rutenbar (S'78–M'84–SM'90) received the Ph.D degree in computer engineering (CICE) from the University of Michigan, Ann Arbor, in 1984.

He is currently a Professor of electrical and computer engineering and of computer science, and Director of the Semiconductor Research Corporation—CMU Center of Excellence in CAD and IC's with Carnegie Mellon University, Pittsburgh, PA. His research interests focus on circuit and layout synthesis for mixed-signal ASIC's, high-performance digital IC's, and FPGA's.

Dr. Rutenbar received a Presidential Young Investigator Award from the National Science Foundation in 1987. At the 1987 EIII-ACM Design Automation Conference, he received a Best Paper Award for work on analog circuit synthesis. He is currently on the Executive Committee of the IEEE International Conference on CAD, and Program Committees for the ACM/IEEE Design Automation Conference and European Design & Test Conference. He is on the Editorial Board of *IEEE Spectrum*, and chairs the Analog Technical Advisory Board for Cadence Design Systems. He is a member of ACM, Eta Kappa Nu, Sigma Xi, and AAAS.

L. Richard Carley (S'77–M'81–SM'90) received the S.B., M.S., and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge, in 1976, 1978, and 1984, respectively.

He is a Professor of electrical and computer engineering with Carnegie Mellon University, Pittsburgh, PA. He was with MIT's Lincoln Laboratories and has acted as a consultant in the area of analog circuit design and design automation for Analog Devices and Hughes Aircraft, among others. In 1984, he joined Carnegie Mellon, and in 1992 he was promoted to Full Professor. His current research interests include the development of CAD tools to support analog circuit design, design of high-performance analog signal processing IC's, and the design of low-power high-speed magnetic recording channels.

Dr. Carley received a National Science Foundation Presidential Young Investigator Award in 1985, a Best Technical Paper Award at the 1987 Design Automation Conference, and a Distinguished Paper Mention at the 1991 International Conference on Computer-Aided Design. He was also awarded the Guillemin Prize for the best EE Undergraduate Thesis.