Performance Space Modeling for Hierarchical Synthesis of Analog Integrated Circuits

Georges Gielen, Trent McConaghy, Tom Eeckelaert
Katholieke Universiteit Leuven, ESAT–MICAS
Kasteelpark Arenberg 10
B-3001 Leuven, Belgium
gielen@esat.kuleuven.ac.be

ABSTRACT

Automated analog sizing is becoming an unavoidable solution for increasing analog design productivity. The complexity of typical analog SoC subsystems however calls for efficient methods that can handle design hierarchy, in terms of both performance estimation and hierarchical design optimization method. This paper discusses and compares recent developments in this area, with special emphasis on automated modeling and on multi–objective bottom–up hierarchical design.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-aided design; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*

General Terms

Algorithms, Design, Performance

Keywords

Hierarchical Synthesis

1. INTRODUCTION

Analog design automation has progressed to the point where there are industrially useful and commercially available tools at the cell level, for both automated sizing and for automated layout. These tools are targeted for circuits having up to about 50–100 transistors (read : design variables). However, automation for larger systems with many more design variables is not prevalent in industry yet. This is largely because all the pieces of the system–level design puzzle are not fully in place. The challenge is to have a methodology that makes the complexity of the problem tractable.

There are many methodology alternatives, and they can be organized according to two key dimensions: how performance estimation is done, and how the design space is organized and traversed. Performance estimation can be done "flat", or hierarchy can be leveraged as well. There is additional freedom in the accuracy / speed tradeoff of a given performance estimator, going from transistor–level accuracies up to higher–level (behavioral or functional) modeling. We explore these issues further. The problem of performance estimation can be considered to be roughly orthogonal to how design space is organized. The design space can be flat, and traversed in a flat manner (i.e. all at once). Or, it can be organized hierarchically into subproblems, and traversed according to a hierarchical design methodology. Interestingly, there are different ways how a hierarchical methodology can be developed; this also has many implications, which we will explore later.

This paper is organized according to these two dimensions. Section 2 discusses performance estimation techniques, starting at the cell level and progressing to the system level. Section 3 discusses design space traversal options, including flat, bottom–up hierarchical, concurrent hierarchical, top–down constraint–driven (and the feasibility–modeling bottom–up extension), and multi–objective bottom–up methodologies. Section 4 finishes with some conclusions.

2. PERFORMANCE ESTIMATION

Figure 1 summarizes different performance estimation approaches.

2.1 Cell–Level Performance Estimation

We now explore the challenge of performance estimation for automated sizing further, first at the cell level, then to larger systems, with an emphasis on more recent techniques. The ideal is to have the most accurate performance estimation (i.e. SPICE) at least for the final designs, within the constraints of overall runtime of an automated sizer. One inevitably finds that there is a tradeoff between accuracy and speed. But, modeling can be used in various creative ways to improve runtime without hurting final accuracy.

At the cell level, the problem is the simplest, so let us start our discussion there. Until the early 1990's, SPICE–like simulators were not feasible within the loop of a circuit design optimization problem because they were too slow. So, many performance estimation techniques were devised that sacrificed accuracy for speed, such as ISAAC [16], AWE [33]; for a review see [17]. But, Moore's Law drove CPU speeds to progress to the point where SPICE–in–the–loop optimization was feasible, and academia soon exploited it

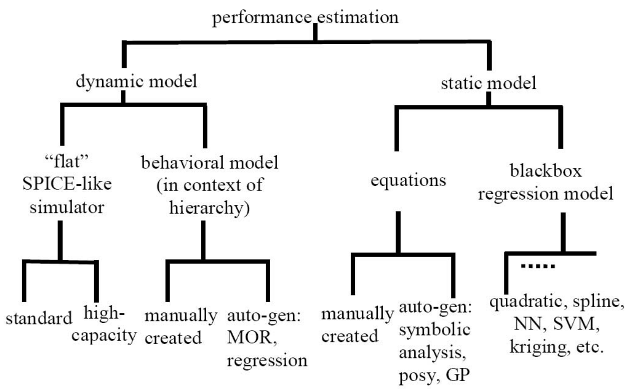
performance estimation
dynamic model
static model
"flat" SPICE-like simulator
behavioral model (in context of hierarchy)
equations
blackbox regression model
standard
high-capacity
manually created
auto-gen: MOR, regression
manually created
auto-gen: symbolic analysis, posy, GP
quadratic, spline, NN, SVM, kriging, etc.

Figure 1: Performance Estimation Approaches

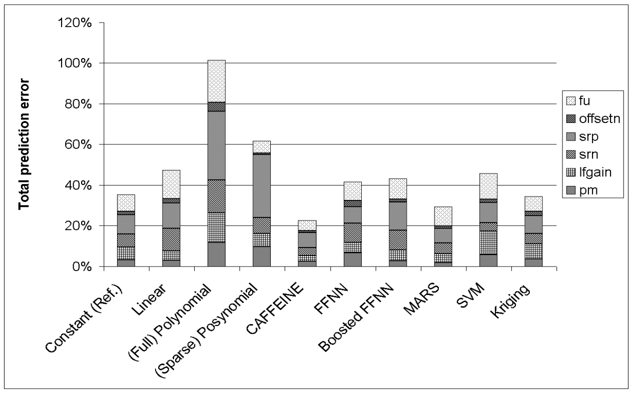

Figure 2: Comparison of Prediction Error on Six Modeling Problems (Within the Context of Optimization)

(e.g. [22, 3]), as well as industry [40, 4, 27, 29]. So, SPICE has become the (dynamic) performance estimator of choice for cell–level design.

There is still payoff to having faster optimization, and many sizing schemes that aim for speed, leverage performance–estimation techniques that map design variables to performances, but work in conjunction with SPICE. GPCAD [20] had manually–written, SPICE–tuned posynomials to model circuits, which were then subsequently optimized (and verified with SPICE after sizing). The advantage is that posynomials are especially easy to optimize on, but the disadvantages are that posynomials cannot model the multi–modality that many analog circuits have, and also that the models needed to be built manually. It is possible to automatically build models before sizing that do not have multi–modality constraints; [46] demonstrated this with neural networks, and recently showed on splines how the sampling can be more adaptive [45]. The problem with this approach is that the model tries to capture the whole mapping to a target error level, even for regions of the performance space that are useless, therefore needing far more samples than necessary and constraining the number of possible input dimensions.

An alternative is to adaptively build models during optimization, and use these models to make more intelligent choices of where to simulate next. The most well–known model of this type is sequential quadratic programming, which makes a local quadratic model mapping design variables to performances; the quadratic model is iteratively updated using an update scheme such as BFGS. However, one can be very flexible in the choice of modeling scheme; one could use automatically–created posynomials [8], neural networks [44], splines [25], genetic programming [25], kriging [25], boosted neural networks [25], or support vector machines [21]. Figure 2 compares the predictive abilities of these schemes (adapted from [25]). Each of these techniques has different strengths and weaknesses; what is most important is prediction ability, scalability to input dimensionality, and sufficiently small training time.

One can even make models after optimization with all the simulation data from the run, as demonstrated by [23] on boosted neural networks. Such a model might perhaps be used within a hierarchical methodology, or it could be used to improve understanding, as suggested in [24] in combination with genetic programming. Advanced regression modeling has also been used to mitigate potentially high runtimes when taking into account manufacturing variations [3, 37, 2, 10] and layout effects [1, 47].

2.2 System–Level Performance Estimation

Let us now discuss performance estimation for larger designs, beyond the cell level. Remember, design methodology is somewhat orthogonal to performance estimation. If we want, we can have a flat design methodology and hierarchical simulation. Or, we can have some sort of hierarchial design methodology and always flat SPICE–like simulation, though the problem is of course runtime. It may not be as bad as one might think, however, if the design spaces at the non–cell levels are sufficiently small. 'Flat' simulation may be better than one might expect too, if we use high–capacity simulators [28, 5, 41, 26]. From the perspective of the user, high–capacity simulators appear to simulate the circuit as if it were flat, even though there may be underlying hierarchical elements. The disadvantage of high–capacity simulators is that because they get speed by exploiting regularity in the circuit, they only result in effective speedups for certain types of circuits. Besides SPICE–like simulation, one could also use other performance estimators such as symbolic analysis or AWE, but these also have scaling issues.

To directly exploit hierarchy in performance estimation is promising for attaining sufficient accuracy in reasonable runtime. At each node of the hierarchy, all subcomponents would have 'compact' behavioral models rather than full SPICE models. Those compact models can be manually created, which is a long and tedious process, though the resulting models could be quite trustworthy. The most modern models can actually be tuned to a given manufacturing process.

Automatic creation of behavioral models is therefore of great interest. Model–order reduction (MOR) and regression are the two main approaches. MOR approaches find projections to map the system's states into a smaller set, yet roughly maintain the same behavior; [36] is a survey. Recent MOR research has explored modeling nonlinear circuits. Piecewise–linear [34] and piecewise–polynomial approaches [13] do this by tying together linear or polynomial models along likely trajectories in the state space. Alternatively, kernel methods [32, 31] nonlinearly map the state space into a higher–dimensional space that is then handled with linear regression. The results for kernel methods are particularly promising.

Regression approaches create models that learn from the (SPICE) simulation waveforms of the circuit's inputs and outputs. They have traditionally been synonymous with 'black box' behavioral models because the model is a black box and because the circuit's internal states are not used, as in [35]. However, [25] demonstrates how to generate interpretable equations to describe the state transitions. While [25] automatically generates its own internal states, it is also conceivable to generate interpretable equations that describe state transitions among a subset of the circuit's states, making it a MOR approach. Thus, perhaps a better way to distinguish among behavioral modeling methodologies is along the dimensions of: (1) use/reduce measured internal states y/n, and (2) model is interpretable y/n.

Model creation has traditionally been regarded as cleanly divided between: 'automatic' or 'manual'. The ability to generate interpretable models [24, 25] opens a new possibility: 'human–computer collaboration' in which the user interacts with the automated model, trying new variations on model assumptions, then quickly getting feedback on possible models and examining their validity.

It should be noted that modeling techniques are not mutually exclusive; for example one could use a combination of simulation and adaptive modeling at the cell level, and a mixture of high–capacity simulators and behavioral models at higher levels. One could even use more than one type of performance estimator to analyze a candidate design point. At greater–than–cell levels, dynamic modeling will likely end up being a combination of high–capacity simulators and behavioral modeling. Perhaps the fields will even converge, e.g. high capacity simulators working on a wider range of circuits, making all simulation look flat.

3. DESIGN SPACE ORGANISATION AND TRAVERSAL

Figure 3 summarizes different approaches to design space organization and traversal.

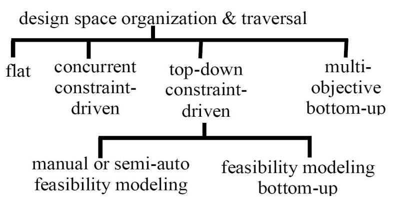
design space organization & traversal
flat
concurrent constraint-driven
top-down constraint-driven
multi-objective bottom-up
manual or semi-auto feasibility modeling
feasibility modeling bottom-up

Figure 3: Approaches to Design Space Organization and Traversal

3.1 Flat Methodology

A 'flat' design methodology is one in which the whole design is attacked at once. Think of the complexity of designing a whole A/D converter with all its transistors at once. With possibly thousands of transistors, the problem is just too complex to attack all at once. It takes also too long runtimes for simulator–in–the–loop automated sizing tools. However, it would actually be possible if one has a fast performance estimator and an algorithm that can handle many variables, such as posynomials with geometric programming for A/D converters for instance [19]. The problem, however, is that the models need to be manually created, and are constrained to convex surfaces (and analog circuit design problems have no guarantee of being convex).

3.2 Bottom–Up Methodology

To cope with complexity, larger designs are almost always broken up in a hierarchical fashion, then traversed according to a hierarchical design methodology. Each sub–problem has a size that is tractable for design by either a human or an automated tool. We now review possible methodologies.

The 'Bottom–Up' (BU) methodology is the way a typical analog designer would design an analog circuit. In it, the designer begins with a set of system specifications. Based on knowledge of previous designs, the system–level design is broken down into a set of sub–blocks, until all the blocks are at the transistor level. This is typically a fast phase, as no verification is done; it relies on designer experience and instinct to break up the problem. The blocks are designed in a bottom-up fashion. To compensate for non–idealities and uncertainties in feasibility, lower–level blocks tend to be overdesigned. Unfortunately, the complete system design has to be performed first before the performances can be checked and compared to the specifications; if the system doesn't satisfy specs, there is no clear method to resolve the problem; it may turn out that the complete bottom–up design may need to be done all over again, wasting precious time. The ad hoc nature of BU is probably its most distinguishing feature – a discomforting thought to design team managers who want to see predictability in their design flow.

3.3 Concurrent Constraint–Driven Methodology

The Concurrent Constraint–Driven methodology[30] optimizes at the system level and the cell level at the same time, with a cost function that coerces the designs to agree. The design space includes the cell–level design space (e.g. opamp transistors' widths and lengths), and system–level design space (specifications of the system's subblocks). The optimizer goals are (a) to meet the system–level specifications, and (b) 'glue' constraints that coerce agreement between the system–level's design variables and what performances the opamp is actually achieving. The problem with this approach is that it cannot scale – the design space combining even two levels will be almost certainly too huge. Even in [30], the authors had to force symmetry among several amplifiers in order to make the search tractable, and the runtime was still far longer than their comparable work at just the cell level. Furthermore, this approach does not show a clear path to handle more than two levels of hierarchy.

3.4 Top–Down Constraint–Driven Methodology (TDCD)

Engineering schools throughout the world teach variants of a hierarchical methodology colloquially referred to as 'The' Engineering Design Methodology. It tends to have a structured means of traversing the design hierarchy, starting from a set of system–level specifications. In analog CAD, this methodology made its appearance as the Top–Down Constraint–Driven methodology [6], or TDCD, which formalized how the traversal can be done with rigor.

Starting from system–level specifications, an architecture is chosen, and designed (optimized) at the architecture level using an optimizer. The authors have equations to describe what combinations of performances are feasible; the optimizer's constraints are to meet the performance constraints, with the objective to 'maximize flexibility'. The design space for the optimizer is the target specifications of the next–lower–level design blocks. So, each sub–block is given a set of specifications to meet, also known as 'constraint transformation.' The lower–level building blocks are optimized in the same way, until all the blocks in the hierarchical tree are designed. At the leaf nodes of the tree are the transistor sizes. If, during these mapping stages, a subblock is not feasible or the specifications can not be met, then the hierarchy is climbed again to choose a different architecture or a different mapping function. At the tail end of the design process, a full bottom–up verification is done with accurate performance estimation (i.e. SPICE). Other authors have proposed similar methodologies [14, 42]. [12] emphasizes the use of VHDL–AMS for modeling. Many of these techniques have resulted in fully designed and fabricated analog systems, which underscores the success of the approach.

Remember that there are many performance estimation options available to users of TDCD, which we have already reviewed. Typically, manually–created behavioral models are used during optimization, and a SPICE–like simulation is done for verification.

In analog circuits, tradeoffs always exist. A designer might typically find him/herself striking a compromise between power, accuracy, speed, and area. Unfortunately, TDCD is fundamentally constrained to give, at best, a feasible solution at the system level; understanding system–level tradeoffs is out of the question.

TDCD heavily relies on a model of what performances are feasible. In early work at Berkeley and others, this was done with manually created equations, which of course is very time consuming, error–prone, and poor for adapting to new processes.

3.5 Feasibility Modeling Bottom–Up Plus TDCD

Researchers have recently been addressing the feasibility problem associated with TDCD. The idea is to construct feasibility models of a performance space in a bottom–up fashion (FMBU), which is then followed by the traditional TDCD scheme.

In [18], radial–basis–functions model the feasibility, and the performance of subblocks at all levels of the hierarchy. More recently, [9] uses Support Vector Machines and treats the problem as merely a classification problem, with performance values as inputs and feasible y/n as the output. That work has an adaptive sampling scheme (or optimization scheme, depending how you look at it) to iteratively refine the detail of the feasibility model. In a similar vein, [38] also models the region of feasibility, but with spec–wise linearized models.

Thus, FMBU+TDCD overcomes the feasibility modeling problem of plain TDCD. Another advantage is that once the feasibility region for a subblock is generated for a given technology, it does not need to be generated again. Over time, libraries of feasibility regions can be built. The FMBU half also provides the opportunity to see system–level tradeoffs: it's merely the subset of the feasibility region at the system level, specifically the nondominated performances.

A problem with FMBU+TDCD is that whenever a real design needs to be done, the TDCD portion needs to be carried out. This requires sizing at each node in the hierarchy, which can take considerable time (e.g. hours or more per node). Even if the set of designs found during the initial feasibility modeling are retained, they may not meet the exact specifications requested by the higher–level node, necessitating a sizing run. Another problem is that during FMBU, considerable simulation effort is expended to model the whole feasibility region; there are designs on the feasibility boundary that are worse in every way compared to other designs on the boundary.

3.6 Multi–Objective Bottom–Up Methodology

Recent research suggests a way to avoid the extra work that FMBU+TDCD has imposed upon itself. The core ideas are (a) to determine just tradeoffs among the performance objectives, not whole feasibility regions, and (b) to directly use designed circuits rather than models. We now discuss this in detail.

Recent cell–level analog CAD sizers in both academia [11, 39] and industry [40, 4] find the 'Pareto optimal sets' (discrete set of designs approximating a tradeoff) using multi–objective algorithms (e.g. [43, 48]). These cell–level Pareto optimal sets can be directly exploited for system–level design, as described in [15] in what amounts to a 'MUlti–objective Bottom–Up' (MUBU) methodology. The design space for the next level up is the 'selection' of a design for each of the sub–blocks. A 'selected' sub–block design is actually pointing to a specific design from the lower–level tradeoff Pareto–hypersurface of that sub–block. The hierarchy traversal proceeds in an upwards fashion, in the end providing an optimal system–level tradeoff. Figure 4 illustrates.

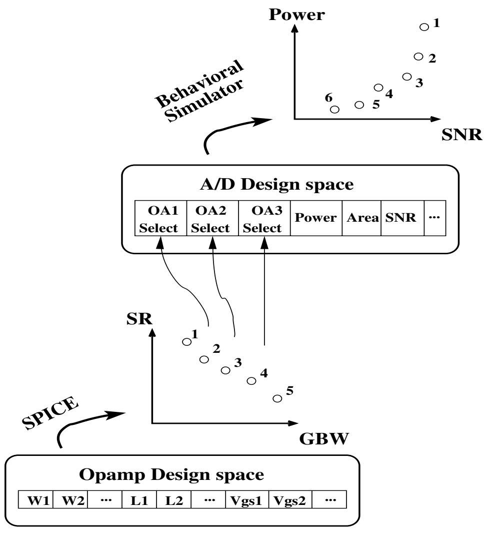

Figure 4: Multi–Objective Bottom–Up Methodology (MUBU) maps design points to performance points via performance estimation as usual. It finds and keeps just the tradeoff via multi–objective optimization. It then propagates tradeoffs upwards where they combine to make the next level's design space.

In MUBU, the shape of the design spaces at non–cell levels is perhaps surprising: it's a discrete set of points, but not on a Euclidean grid or any particular arrangement, except for the fact that if one interpolated among them, one would get a hypersurface. Thus, when doing system–level optimization, the algorithm needs different operators which jumps from discrete point to discrete point (with a bias towards points that are nearby). Such an approach is common in the layout literature (e.g. [7]).

In MUBU, and in contrast to TDCD, any design that is selected on any level is already fully sized. An analog designer just has to choose a solution at the system level according to the performance specs, and immediately all the design variables of the complete system are set. MUBU simultaneously provides both full sizings and flexibility in specs. In addition, once a given block has had its Pareto optimal set generated in a given techology, it can be reused. Figure 5 illustrates a system-level tradeoff for an A/D, generated by MUBU (details in [15]).

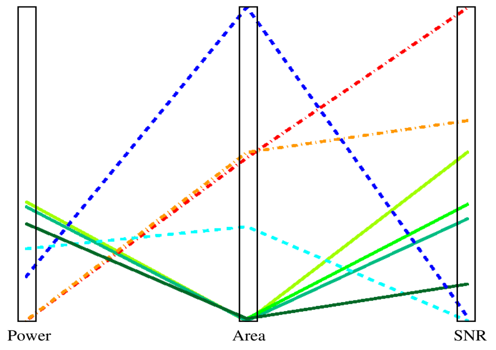

Figure 5: System-Level Tradeoffs Generated by MUBU. These are tradeoffs of an A/D, shown in parallel coordinates. Each 3-segment line represents another design. Each design is fully sized (no top-down traversal is needed).

In TDCD methodologies, a good feasibility–modeling approach is necessary for success (e.g. use FMBU). In contrast, MUBU completely avoids explicit modeling any performance surfaces. The closest thing it has to a 'model' is the Pareto–optimal set, which collectively approximates a performance surface. But it's no more and no less than a set of points. Of course, one could build a regression model, for example by extending [11] for use in hierarchical design. Doing this might give slight refinement in specs, but it then requires that a new design is done for each new problem, which is giving up a lot of the benefits of MUBU.

Recall that FMBU needs to generate a full feasibility model for each node. In contrast, MUBU captures all the good circuits and nothing more (i.e. tradeoffs). Figure 6 illustrates.

For these various design space traversal methodologies, we have not discussed the choice of a particular optimization algorithm. That choice can be orthogonal to the methodology; the main thing is that the algorithm should efficiently accomplish its job. Popular choices in analog CAD include variants and combinations of simulated annealing, evolutionary algorithms, and pattern search. Some methodologies may have special requirements, most notably MUBU, which needs an algorithm flexible enough to work on arbitary landscapes, and needs to be multi–objective.

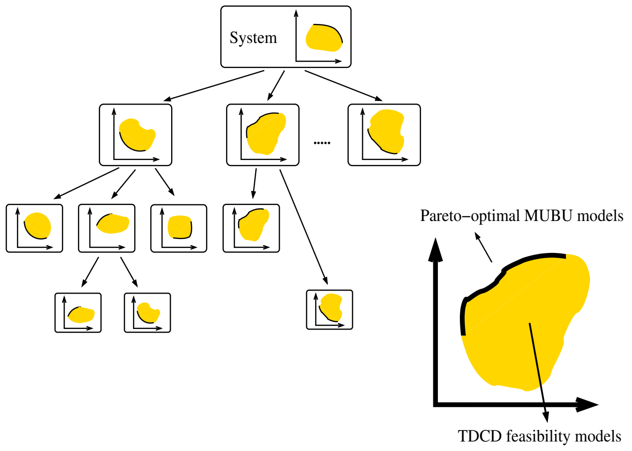

Figure 6: FMBU+TDCD vs. MUBU. The shaded–out area in each block represents its performance feasibility region, which FMBU generates bottom–up. The line on one edge of one quadrant of the region represents the Pareto–optimal set that MUBU generates bottom–up. FMBU has to work a lot harder than MUBU, to merely learn about designs not even on the tradeoff surface.

4. CONCLUSIONS

In this paper, we reviewed the state of the art for automation in analog system–level sizing, which we divided into two issues: how to do performance estimation at each level, and how to tackle the sizing such that the design space for any given subproblem has a tractable size. Good techniques for automated behavioral modeling, in combination with high–capacity simulators, create the possibility of having simulations for systems that are not excessively long in runtime, yet sufficiently detailed. We paid particular attention to the top–down constraint–driven methodology and its recent extension for bottom–up feasibility modeling, as well as the multi–objective bottom–up methodology (MUBU). MUBU points to a possible means of achieving analog IP libraries that are both sufficiently 'hard' (i.e. pre–designed) yet with sufficiently flexible specifications.

5. REFERENCES

[1] A. Agarwal, H. Sampath, V. Yelamanchili, and R. Vemuri. Fast and accurate parasitic capacitance models for layout-aware. In *Proceedings Design Automation Conference*, pages 145–150, 2004.

[2] G. Alpaydin, S. Balkir, and G. Dundar. An Evolutionary Approach to Automatic Synthesis of High-Performance Analog Integrated Circuits. *IEEE Trans. on Evol. Computation*, 7(3):240–252, 2003.

[3] K. Antreich, J. Eckmueller, H. Graeb, M. Pronath, F. Schenkel, R. Schwencker, and S. Zizala. WiCkeD: Analog Circuit Synthesis Incorporating Mismatch. In *Proceedings Custom Integrated Circuits Conference*, 2000.

[4] Cadence. NeoCircuit Product. *Inc http://www.cadence.com*, 2005.

[5] Cadence. UltraSim Product. *http://www.cadence.com*, 2005.

[6] H. Chang, E. Charbon, U. Choudhury, A. Demir, E. Felt, E. Liu, A. Malvasi, A. L. Sangiovanni-Vincentelli, and I. Vassiliou. *A Top–Down, Constraint–Driven Design Methodology for Analog Integrated Circuits.* Kluwer Academic Publishers, 1997.

[7] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley, editors. *Analog Device-Level Layout Automation.* Kluwer Academic Publishers, 1994.

[8] W. Daems, G. Gielen, and W. Sansen. Simulation-Based Generation of Posynomial Performance Models for the Sizing of Analog Integrated Circuits. *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, 22(5):517–534, 2003.

[9] F. De Bernardinis, M. I. Jordan, and A. L. Sangiovanni-Vincentelli. Support Vector Machines for Analog Circuit Performance Representation. In *Proceedings Design Automation Conference*, pages 964–969, 2003.

[10] B. De Smedt and G. G. E. Gielen. Holmes: Capturing the Yield-Optimized Design Space Boundaries of Analog and RF Integrated Circuits. In *Proceedings Design Automation and Test in Europe Conference*, page 10256, 2003.

[11] B. De Smedt and G. G. E. Gielen. WATSON: Design Space Boundary Exploration and Model Generation for Analog and RF IC Design. *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, 22(2):213–224, Feb. 2003.

[12] A. Doboli, N. Dhanwada, A. Nunez-Aldana, and R. Vemuri. A Two–Layer Library–Based Approach to Synthesis of Analog Systems from VHDL–AMS Specifications. *ACM Transactions on Design Automation of Electronic System*, 9(2):238–271, Apr. 2004.

[13] N. Dong and J. Roychowdhury. Piecewise Polynomial Nonlinear Model Order Reduction. In *Proceedings Design Automation Conference*, pages 484–489, 2003.

[14] S. Donnay, G. G. E. Gielen, W. Sansen, W. Kruiskamp, D. Leenaerts, S. Buytaert, K. Marent, M. Buckens, and C. Das. Using Top–Down CAD Tools for Mixed Analog/Digital ASIC's: a Practical Design Case. *Journal of Analog Integrated Circuits and Signal Processing*, 10:101–117, 1996.

[15] T. Eeckelaert, T. McConaghy, and G. G. E. Gielen. Efficient Multiobjective Synthesis of Analog Circuits using Hierarchical Pareto–optimal Performance Hypersurfaces. In *Proceedings Design Automation and Test in Europe Conference*, pages 1070–1075, 2005.

[16] G. Gielen and W. Sansen. *Symbolic Anaysis for Automated Design of Analog Integrated Circuits*. Kluwer, 1991.

[17] G. E. Gielen. Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits. In R. Rutenbar, G. Gielen, , and B. Antao, editors, *Computer Aided Design of Analog Integrated Circuits and Systems*, chapter 1, pages 3–30. IEEE Press, Piscataway, NJ, 2002.

[18] R. Harjani and J. Shao. Feasibility and Performance Region Modeling of Analog an Digital Circuits. *Journal of Analog Integrated Circuits and Signal Processing*, 10(1):23–43, June 1996.

[19] M. Hershenson. Design of Pipeline Analog-To-Digital Converters Via Geometric Programming. In *Proceedings IEEE/ACM International Conference on Computer Aided Design*, pages 317–324, 2002.

[20] M. Hershenson, S. Boyd, and T. Lee. GPCAD: A Tool for CMOS Op-Amp Synthesis. In *Proceedings IEEE/ACM International Conference on Computer Aided Design*, pages 296–303, 1998.

[21] T. Kiely and G. G. E. Gielen. Performance Modeling of Analog Integrated Circuits Using Least-Squares Support Vector Machines. In *Proceedings Design Automation and Test in Europe Conference*, pages 448–453, 2004.

[22] M. Krasnicki, R. Phelps, R. A. Rutenbar, and L. R. Carley. MAELSTROM: Efficient Simulation-Based Synthesis for Custom Analog Cells. In *Proceedings Design Automation Conference*, pages 945–950, 1990.

[23] H. Liu, A. Singhee, R. Rutenbar, and L. Carley. Remembrance of Circuits Past: Macromodeling by Data Mining in Large Analog Design Spaces. In *Proceedings Design Automation Conference*, pages 437–442, 2002.

[24] T. McConaghy, T. Eeckelaert, and G. G. E. Gielen. CAFFEINE: Template-Free Symbolic Model Generation of Analog Circuits via Canonical Form Functions and Genetic Programming. In *Proceedings Design Automation and Test in Europe Conference*, Mar 2005.

[25] T. McConaghy and G. Gielen. Analysis of Simulation-Driven Numerical Performance Modeling Techniques for Application to Analog Circuit Optimization. In *ISCAS*, May 2005.

[26] MentorGraphics. Eldo Mach Product. *http://www.mentor.com*, 2005.

[27] Muneda. WiCkeD Product. *http://www.muneda.com*, 2005.

[28] Nassda. Hsim Product. *http://www.nassda.com*, 2005.

[29] Orora. Arsyn Product. *http://www.orora.com*, 2005.

[30] R. Phelps, M. Krasnicki, R. A. Rutenbar, L. R. Carley, and J. R. Hellums. A Case Study of Synthesis for Industrial–Scale Analog IP: Redesign of the Equalizer/Filter Frontend for an ADSL CODEC. In *Proceedings Design Automation Conference*, pages 1–6, 2000.

[31] J. Phillips. A Statistical Perspective on Nonlinear Model Order Reduction. In *Behavioral Modeling and Simulation Workshop*, 2003.

[32] J. Phillips:2003a, J. Afonso, A. Oliveira, and L. Silveira. Analog Macromodeling Using Kernel Methods. In *Proceedings IEEE/ACM International Conference on Computer Aided Design*, pages 446–443, 2003.

[33] L. Pillage and R. Rohrer. Asymptotic Waveform Evaluation. *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, pages 352–366, April 1990.

[34] M. Rewienski and J. White. A Trajectory Piecewise-Linear Approach to Model-Order Reduction and Fast Simulation of Nonlinear Circuits and Micromachined Devices. In *Proceedings IEEE/ACM International Conference on Computer Aided Design*, pages 252–257, 2001.

[35] D. Root, J. Wood, and N. Tufillaro. New Techniques for Non-linear Behavioral Modeling of Microwave/RF ICs from Simulation and Nonlinear Microwave Methods. In *Proc. the ACM International Symposium on Physical Design*, pages 76–83, 2000.

[36] J. Roychowdhury. Automated Macromodel Generation for Electronic Systems. In *Behavioral Modeling and Simulation Workshop*, 2003.

[37] F. Schenkel, M. Pronath, S. Zizala, R. Schwencker, H. Graeb, and K. Antreich. Mismatch Analysis and Direct Yield Optimization by Spec-Wise Linearization and Feasibility-Guided Search. In *Proceedings Design Automation Conference*, 2001.

[38] G. Stehr, H. Graeb, and K. Antreich. Feasibility Regions and their Significance to the Hierarchical Optimization of Analog and Mixed–Signal Systems. *International Series of Numerical Mathematics*, 146:167–184, 2003.

[39] G. Stehr, H. Graeb, and K. Antreich. Performance Trade–off Analysis of Analog Circuit By Normal–Boundary Intersection. In *Proceedings Design Automation Conference*, pages 958–963, 2003.

[40] Synopsys. Circuit Explorer Product. *http://www.synopsys.com*, 2005.

[41] Synopsys. Nanosim Product. *http://www.synopsys.com*, 2005.

[42] G. Van der Plas, G. Debyser, F. Leyn, K. Lampaert, J. Vandenbussche, G. G. E. Gielen, W. Sansen, P. Veselinovic, and D. Leenaerts. AMGIE — A Synthesis Environment for CMOS Analog Integrated Circuits. *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, 20(9):1037–1058, Sept. 2001.

[43] D. A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: classifications, analysis, and new innovations.* PhD thesis, Air Force Institute of Technology, Wright–Patterson AFB, USA, June 1999.

[44] P. Vancorenland, G. V. der Plas, M. Steyaert, G. Gielen, and W. Sansen. A Layout-aware Synthesis Methodology for RF Circuits. In *Proceedings IEEE/ACM International Conference on Computer Aided Design*, pages 358–, 2001.

[45] G. Wolfe, M. Ding, and R. Vemuri. Adaptive Sampling and Modeling of Analog Circuit Performance Parameters. In *Proc. VLSI-SOC*, pages 142–, 2003.

[46] G. Wolfe and R. Vemuri. Extraction and Use of Neural Network Models in Automated Synthesis of Operational Amplifiers. *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, 22(2), 2003.

[47] G. Zhang, E. A. Dengi, R. A. Rohrer, R. A. Rutenbar, and L. R. Carley. A Synthesis Flow Toward Fast Parasitic Closure for Radio-Frequency Integrated Circuits. In *Proceedings Design Automation Conference*, pages 155–158, 2004.

[48] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, 2000.