View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by idUS. Depósito de Investigación Universidad de Sevilla

Geometrically-Constrained, Parasitic-Aware Synthesis of Analog ICs

Rafael Castro-López, Francisco V. Fernández, and Ángel Rodríguez Vázquez
Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica
Edificio CICA. Avda. Reina Mercedes s/n, E-41012- Sevilla, Spain
Tel.: +34 955 056 666. Fax: +34 955 056 686
E-mail: Rafael.Castro@imse.cnm.es

ABSTRACT

In order to speed up the design process of analog ICs, iterations between different design stages should be avoided as much as possible. More specifically, spins between electrical and physical synthesis should be reduced for this is a very time-consuming task: if circuit performance including layout-induced degradations proves unacceptable, a re-design cycle must be entered, and electrical, physical, or both synthesis processes, would have to be repeated. It is also worth noting that if geometric optimization (e.g., area minimization) is undertaken after electrical synthesis, it may add up as another source of unexpected degradation of the circuit performance due to the impact of the geometric variables (e.g., transistor folds) on the device and the routing parasitic values. This awkward scenario is caused by the complete separation of said electrical and physical synthesis, a design practice commonly followed so far. Parasitic-aware synthesis, consisting in including parasitic estimates to the circuit netlist directly during electrical synthesis, has been proposed as solution. While most of the reported contributions either tackle parasitic-aware synthesis without paying special attention to geometric optimization or approach both issues only partially, this paper addresses the problem in a unified way. In what has been called layout-aware electrical synthesis, a simulation-based optimization algorithm explores the design space with geometric variables constrained to meet certain user-defined goals, which provides reliable estimates of layout-induced parasitics at each iteration, and, thereby, accurate evaluation of the circuit ultimate performance. This technique, demonstrated here through several design examples, requires knowing layout details beforehand; to facilitate this, procedural layout generation is used as physical synthesis approach due to its rapidness and ability to capture analog layout know-how.

Keywords: Analog CAD, Layout Templates, Floorplan Sizing, Layout Parasitics, Layout-Aware Synthesis.

1. INTRODUCTION

The CMOS semiconductor industry has continuously evolved and prospered since the early 1970's. The ever shrinking minimum feature size has triggered a revolution in integrated designs, from application-specific (ASICs) to entire systems on a single chip (SoCs). Notwithstanding, a critical design productivity lag has been reported [1]: with a productivity growth rate of 21%, compared to a 58% complexity growth rate, design cost is increasing rapidly. Taking into account the ever demanding time-to-market pressures, this picture is clearly worrisome. For analog and/or mixed-signal (AMS) design the situation is even worse because of many different reasons, the most significant being the lack of commercial CAD tools and methodologies to efficiently support the analog design. In this scenario, productivity should be boosted to double every year in order to bridge the gap [2]. To this goal, three research directions have been suggested [2]: (1) increase the fraction of the design coming from reuse-based design practices, (2) improve synthesis methods so that subtle analog design knowledge is more efficiently managed, and (3) avoid iterations between design stages. This paper is focused on the third direction and, in particular, on removing the iterations between electrical synthesis (also known as sizing and here understood as the process of mapping performance specifications into device-level characteristics) and physical synthesis (layout generation).

In designing circuits, the required features for design quality and productivity are multiple. One of them is to ensure that the design achieves an effective use of silicon area because, in volume production, the area of the chip is paramount for determining the final production cost. This can be assured by maximizing the layout regularity, optimizing the component aspect ratios, or even by establishing careful floorplanning for the circuit. Another important concern is related to the degradation of performance due to layout-induced parasitics. Such degradation may require time-consuming, unsystematic iterations between layout generation and circuit sizing to improve a design that fails to meet the intended performance

VLSI Circuits and Systems II, edited by José Fco. López, Francisco V. Fernández,
José María López-Villegas, José M. de la Rosa, Proceedings of SPIE Vol. 5837
(SPIE, Bellingham, WA, 2005) 0277-786X/05/$15 · doi: 10.1117/12.607933

specifications, iterations that are a direct consequence of the complete separation of the bottom-up layout generation and the top-down electrical synthesis. When layout-induced errors degrade the circuit performance, circuit design changes are accomplished via circuit re-sizing. An important aspect of this problem is how to early evaluate layout-induced parasitics: in manual design, overestimation of parasitics results in wasted power and area, while underestimation may lead to fatal performance degradation. These iterations incur repeating the related synthesis processes and other interfacing costs, which altogether may eventually lead into product-to-market failure.

In this paper, a synthesis methodology that tries to solve the problem is described. The underlying idea is to bring layout generation into the very sizing process, so that circuit sizing is carried out with enough information about layout-induced degradation (parasitics) as well as geometric features (such as area occupation) of the eventually implemented layout. In this way, circuit sizing yields a solution that is robust against layout-induced degradation effects and that fulfils a number of user-defined geometric goals, among them area minimization being the most important.

This methodology, known as **layout-aware** synthesis, is approached here as composed of two sizing techniques, namely **parasitic-aware** and **geometrically-constrained sizing**. The latter technique concerns the inclusion of layout knowledge in the sizing process to obtain a solution for which the area and shape of the eventually implemented layout are optimized. Such a goal is accomplished be means of finding, during the sizing process, the values of geometric parameters (e.g., number of folds of MOS transistors) that yield optimal geometric features. This optimization can be defined either as a restriction on some geometric aspects of the layout (e.g., a pre-defined aspect ratio, or a maximum width or height of the whole circuit layout) or as a design objective (i.e., area minimization). Two aspects should be here carefully consider: first, adding new variables to the sizing process (i.e., geometric variables) must not simply consist in extending the design space because it would make the exploration of such space exponentially more complex; second, in order to being able to include layout details within the sizing process, the layout generation process must be such that retrieving this information can be done rapidly for the sizing process not to last too long. In parasitic-aware sizing, the values of layout parasitics are computed interactively during the sizing process from the device sizes and specific layout information (e.g., the possible implementation style of a group of MOS transistors). The finally obtained sizing is robust against these layout-induced parasitic effects. It is important to note that these two techniques are linked as different geometric variables may give rise to various, different layout-induced effects (e.g., the diffusion capacitance of the drain and source of a transistor change with the number of folds).

This paper is organized as follows. Section 2 describes the separate circuit sizing and layout generation techniques that have been used as foundation to lie the layout-aware synthesis of AMS circuits. Section 3 develops the first component of such synthesis technique, namely the geometrically-constrained sizing technique. Several design examples are provided. Section 4 completes the layout-aware sizing methodology with the parasitic-aware technique. Conclusions are drawn in Section 5.

2. CIRCUIT SIZING AND LAYOUT GENERATION

This Section describes the two synthesis procedures of interest that take place in layout-aware synthesis, namely circuit sizing and layout generation.

2.1 Circuit sizing

For electrical sizing, a simulation-in-the-loop, optimization-based engine has been chosen [3]. This engine, called FRIDGE, uses a proper formulation of the cost function and features some innovative characteristics in the generation and acceptance of movements through the design parameter space that allow to drastically reduce the computational cost, such as preliminary exploration of the design space using a coarse multi-dimensional grid to determine the best regions for further exploration, adaptive control of the temperature in the simulated annealing statistical techniques, synchronization of movement amplitude in parameter space with the temperature, among others. Also an outstanding feature is the capability to incorporate design knowledge to the sizing procedures, which can be done by making use of powerful tools like embeddable C++ programs. As it is shown below, it is this capability which makes it possible to introduce layout-related aspects directly in the sizing process. From this point of view, FRIDGE is an optimization-based sizing approach incorporating the appealing features of knowledge-based ones.

2.2 Layout generation

For layout generation, a template-based approach has been followed. The main reasons for this choice are that (1) layout generation (which, in the context of layout template is usually referred as instancing) turnaround times are considerable smaller than those of optimization-based approaches;(2) layout templates permit searching for optimal block parameters while a priori revealing the knowledge needed for evaluation of layout parasitics, which turns out essential for minimizing the number of iterations between sizing and layout generation design steps. Other factors favoring this choice are that: (3) layout templates are very efficient at handling design expertise which cannot be easily considered by traditional placement and routing algorithms; (4) layout templates ease the placement phase, since the positions of the blocks in the template are stored according to pre-defined relationships embodying constraints from the layout expert; (5) layout templates can be straightforwardly ported, because technological independence can be achieved by writing the procedural template generators that are fully independent of the fabrication process parameters [4]-[6].

The slicing-style has been used to specify the layout floorplan [7]. A slicing floorplan is obtained when the layout components are arranged such that the layout area is recursively divided into horizontal and/or vertical slices, in contrast to non-slicing floorplan. Although non-slicing floorplans are a more general representation that can describe all kinds of tile packing, slicing floorplans have important advantages over non-slicing, namely:

1. Placement can be more easily specified by the relative positions of the layout tiles, since the hierarchy of slicing structures is better defined.
2. It yields more compact layout instances.
3. It also allows evaluating more easily other characteristics of the circuit layout such as routing.
4. As it will be explained in the next Section, it eases geometrically-constrained sizing.

This procedural approach has been carried out using the capabilities of PCELLS technology [8] and the SKILL™ programming [9], two resources of the Cadence's *DFWII* environment.

As said above, the optimization engine is able to incorporate design knowledge and use it during the sizing process. This turns out useful to include an optimization of the layout geometrical features (i.e., layout occupied area and its aspect ratio) directly during the circuit sizing. Thanks to the procedural nature of the adopted layout generation approach, it is possible to predict which is going to be the exact shape of the circuit layout without actually moving to the layout phase. In addition, built-in circuit devices can be actually implemented in different ways (e.g., resistors and MOS transistors can be folded, the transistors in a differential input pair can be arranged in various common-centroid arrangements, etc.), this knowledge being previously embedded in the circuit's layout template.

3. GEOMETRICALLY-CONSTRAINED SIZING

The problem of optimizing geometric aspects of the circuit layout is known as the **floorplan sizing problem**. For a specific electrical sizing of the circuit's building blocks and for a given fabrication process, this problem consists in finding the exact shape (i.e., the building block $width$ and $height$, and, correspondingly, the value of the geometric parameters) of each building block such that some objective function $\Psi(W, H)$ of the circuit layout width, W, and height, H, is minimized. An important requisite is that $\Psi(W, H)$ is nondecreasing in both arguments (i.e., if $W \leq W'$ and $H \leq H'$ then $\Psi(W, H) \leq \Psi(W', H')$. An example of such a function is the layout area, $\Psi(W, H) = W \cdot H$ [1].

As said above, slicing-style layout floorplan entails that the set of building blocks can be arranged in such a way that, when viewing the entire layout as an enclosing rectangle subdivided by horizontal and vertical lines, the two following conditions hold true:

1. There are no overlapping rectangles.
2. Either each basic rectangle is a building block or there is a line segment (a slicing cut) dividing it into two pieces such that each piece fulfils these two conditions.

A slice is, thus, a combination of two or more components, either building blocks or further slices. A useful way to describe the slicing floorplan is by representing the hierarchy structure of the slices in an oriented rooted binary[2] tree called a *slicing tree*. Fig.1 shows an example of a slicing tree. In any slicing tree, there are m non-leaf nodes labeled either h or v,

---

1. At the same time, it should be very appealing that other geometric aspects such as the layout aspect ratio or the layout width and/or height would be set to user-specified values (then, a typical problem should be to minimize the area and set the aspect ratio close to one). Another interesting application is the minimization of the area loss (i.e., area that, as a consequence of the fixed relative placement of layout templates, turns out unusable for latter compaction at higher levels of the hierarchy). Although area loss is not formulated as an objective of the sizing process (inasmuch as the main concern is area minimization), it can serve as a good indicative of the regularity and area usage of the layout template instance. These issues have also been addressed in this paper.
2. Requiring trees to be binary simplifies the description of the problem without loss of generality.

specifying whether the slice is horizontal or vertical (from S1 to S7 in Fig.1(b)), and $n = m + 1$ leaf nodes, each corresponding to one of the basic building blocks (labeled from 1 to 8 in Fig.1(b)).

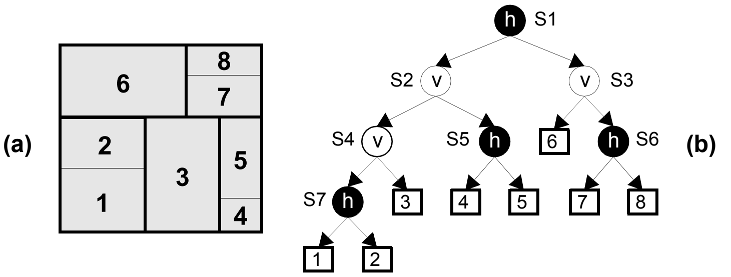

Figure 1: Example of a binary slicing tree representation of a layout template.

Each basic building block (each leaf node of the slicing tree) has several possible shapes, i.e., different values of the pair $(width, height)$. Given an electrical sizing of the built-in devices of the building block and a fabrication process, the collection of different shapes can be calculated by varying one o more geometric parameters (GP) defining the block shape (e.g., the number of transistor folds). This collection of $(width, height)$ pairs forms the **shape function** of the building block, by which it is possible to obtain the $height$ value corresponding to a particular $width$ value, and vice versa.

3.1 Review of previous approaches

For AMS circuits, there are two different approaches to solve the floorplan-sizing problem as formulated above. Both of them start from the premise that the floorplan has already been developed (i.e., the binary slicing tree is known), and, therefore, the relative block placement is fixed and known. The first approach is based on the Stockmeyer's algorithm [10], widely cited in the digital arena. The second approach is based on the formulation of the floorplan sizing as a linear programming problem and the application of the *simplex* method to solve it.

In Stockmeyer's approach, each individual shape function has just a pair of values or shape combinations, that is, if the cell height and width are h and w respectively, and $h \neq w$, the shape function is formed by the pair of values $\{(h, w), (w, h)\}$ (the non-rotated and rotated combinations). The proposed algorithm works in two phases. In the first phase, the tree is processed bottom-up, beginning by associating to each leaf node of the tree a list (according to the rules $h_i > h_{i+1}$ and $w_i < w_{i+1}$) that represents its respective pair of values. If the corresponding component is square-shaped, then there is only one possible orientation and, thus, only one pair. It then proceeds by associating to each non-leaf node v (a similar procedure is used for horizontal non-leaf nodes) a list of s pairs, $\{(h_1, w_1), \ldots, (h_s, w_s)\}$, satisfying the following properties:

1. $s \leq |L(v)| + 1$, where $L(v)$ stands for the set of leaf nodes of the sub-tree rooted at v.
2. $h_i > h_{i+1}$ and $w_i < w_{i+1}$.
3. For each of these pairs there is an orientation ρ of $L(v)$ in terms of the defined slicing cuts.
4. For each orientation ρ of $L(v)$ there is a pair (h_i, w_i) such that $h_i \leq h(\rho)$ and $w_i \leq w(\rho)$, which means that $(h(\rho), w(\rho))$ is kept in the list unless there is another orientation ρ' that is strictly better than ρ in the h or w dimension (or both) and is not worse than ρ in either dimension.

Therefore, if $\{(h_1, w_1), \ldots, (h_k, w_k)\}$ and $\{(h_1', w_1'), \ldots, (h_m', w_m')\}$ are the sorted lists (according with property 2) of the two children of the non-leaf node v, the list associated to this vertical cut is constructed by using the Stockmeyer's algorithm.

By recursively applying this algorithm, the first phase ends with the associated list or shape function of the overall slicing tree T. The second phase starts by choosing the minimum of the function $\Psi(W, H)$ (e.g., the area minimum) by simple inspection of the generated list T and finding out the combination that minimizes the Ψ function. Afterwards, the tree is traversed top-down for selecting the corresponding shape (width and height) of each leaf node, by using associated pointers. Stockmeyer's algorithm can be easily extended to allow an arbitrary number of realizations for each building block. Reported applications in the open literature of the Stockmeyer's algorithm on analog and/or mixed-signal design is limited to those in [11] and [12]. None of these approaches, however, consider the floorplan-sizing problem within a circuit sizing optimization loop, as it is considered here.

The second approach to solve the floorplan-sizing problem tries to formulate it as a linear programming problem and makes use of the above explained shape function of the building blocks too. This linear programming technique is used to solve the floorplan-sizing problem in [13] by means of the simplex method, and in [5], by using a similar solving algorithm. In the latter, the floorplan area is optimized at the slicing level, i.e., the algorithm is applied only to a vertically (or, correspondingly, horizontally) stacked set of horizontally (vertically) arranged devices (what above was called a building block), called groups. The shape function of each group is computed from the individual shape functions of each device, although no method to carry out this composition is detailed.

Unfortunately, neither of the approaches in [5] and [13] successfully attains the area optimum, since they actually do no tackle minimization of the function $\Psi(W, H)$ in a comprehensive way (that is, global minimization of $\Psi(W, H)$ is not carried out during the sizing phase). Besides, these approaches use initial heuristic estimates to solve the problem which makes the solution highly dependent of the starting guess. Furthermore, they may result in suboptimal solutions in regard with the area loss.

3.2 An integrated approach

With the approach presented here, the following three goals have been simultaneously pursued:

1. Minimizing the area of the circuit layout as well as the eventual area lost due to the fixed relative positioning of the template building blocks.
2. Addressing user's geometric constraints imposed on the aspect ratio, width, and height of the circuit layout.
3. Obtaining specific information on the circuit layout geometry (such as the number of fingers/strips, routing lines, and so on) in order to correctly evaluate the circuit parasitics and add them to the circuit sizing (see Section 4).

The first goal is reached effectively, in terms of storage and running time, by using floorplan-sizing procedures based on shape functions, binary slicing trees, and an area minimization algorithm based on Stockemeyer's approach [10]. Solutions to achieve the second goal have been reported in [11] and [5]. Regarding the third goal, none of the reviewed approaches includes the floorplan-sizing problem within the synthesis process, thus being impossible to include the layout-induced effects correctly. The only exception is the methodology reported in [5], but, as noted above, the algorithm used may fail to attain the global area minimum for a particular circuit sizing.

Fig.2 shows the flow diagram implementing the geometrically-constrained sizing technique. Its core, the optimization engine based on simulation explained above, features the means to add designers' expertise to such iterative optimization process. As, in general, the design space of any circuit is a multidimensional space defined by all the design parameters (e.g., physical parameters of transistors, resistors, and capacitors), this is necessary in order to bind the exploration of the circuit design space to only those regions yielding more suitable solutions, thereby improving the efficiency of the procedure. By making use of powerful tools like embeddable C++ programs, it is possible to incorporate valuable design expertise in the form of constraint satisfaction equations. This capability has just allowed carrying out the floorplan-sizing task at each iteration of the optimization process. The C++ program labeled **Geometric Constraints Module** (hereafter referred as the GC module) in Fig.2, has been created to perform the floorplan-sizing task following the technique explained above.

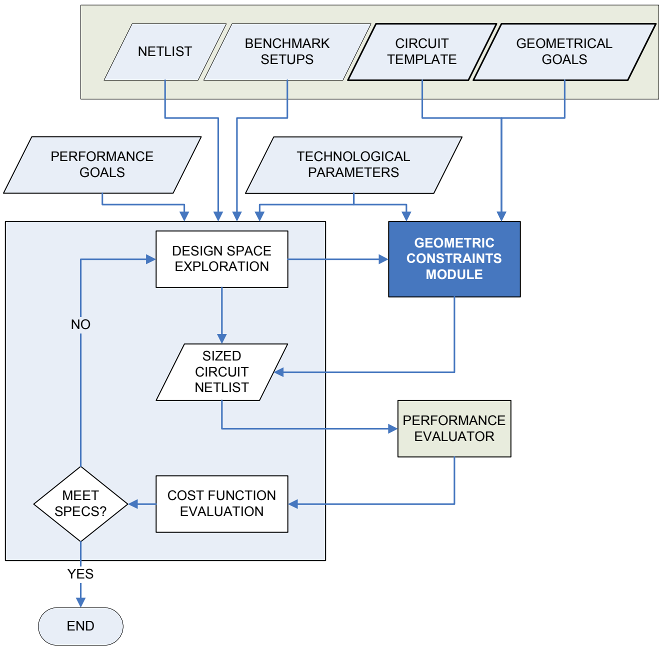

Figure 2: Circuit optimization including geometric layout constraints.

The interaction of the GC module and the optimization engine is as follows: prior to the circuit performance evaluation[3], and right after selecting a new point of the design space, the GC module is executed. The module returns parameter values such as the number of fingers or the physical dimension of passive devices, all defining geometric characteristics of the circuit devices. It also takes into account other geometric aspects, not belonging to any building block but to the circuit layout, as for instance the optimum width of the routing wires calculated for reliability purposes. Circuit optimization is subsequently accomplished with the confidence that for the specific visited point of the design space the resulting layout has minimal area. The GC module attains other geometric goals as well. Area can be also formulated as a design objective, so that the exploration of the design space is driven towards the minimization of this geometric figure. Note that this minimization refers to an optimization between the visited points of the design space rather than between the possible shape combinations at a specific point of the design space. That is, after the optimization engine has addressed all performance specifications, a fine-tuning of the design variables is carried out in order to minimize both area and power consumption.

---

[3] This point reveals an important difference with [5], as this approach solves the floorplan-sizing problem after performance evaluation by means of a convergence iterative procedure, with which it is not possible to accurately determine the value of induced parasitics and, therefore, accurately evaluate the circuit performance.

Fig.2 also specifies the collection of input data the optimization engine requires. Particular information needed by the GC module is:

1. **Circuit template description**: this is the binary slicing tree representation of the layout template the GC module works with.
2. **Geometric goals**: these are the user-specified objectives concerning the geometric characteristics of the eventually generated circuit layout. The following geometric goals are considered:
   - (a) **Aspect ratio**, $AR = W/H$, with an acceptable deviation Δ_{AR}, such that the final layout aspect ratio remains bounded ($AR - \Delta_{AR} \le W/H \le AR + \Delta_{AR}$).
   - (b) **Maximum and minimum width and/or height values**. There are some cases when the circuit layout has to fit into a known area (maximum width, W_M, and/or height, H_M). Accordingly, the user can impose minimum values of the circuit layout width (W_m) and/or height (H_m).
3. **Fabrication process data**: the generated shape functions depend on various characteristics of the implementation technology, mainly on the layout design rules and the process parameters.

Note that, due to the approach used to obtain both the shape function of the circuit layout and the suitable shape(s) of each individual building block, area is always minimized (i.e., for any circuit sizing, any shape combination of the building blocks has minimal area when compared with any other combination not considered in the shape function). The average execution time of the GC module heavily depends on the number of slices and building blocks. Thus, selecting a minimum-depth slicing tree improves the execution time of the GC module.

3.3 Experimental results

This subsection provides experimental assessment of the approach described above by means of two different experiments, each with a different set of geometric goals but sharing the same set of circuit performance specifications. Quality evaluation of the solution involves the comparison of the value of the final area, aspect ratio, and/or width and height geometric figures with their initially specified values. Area loss is additionally used to illustrate the fineness of the solutions.

Fig.3 shows the fully differential operational amplifier used as demonstration circuit. Fig.4(a) presents its layout template view for this circuit block. It consists of six slices, each of them having from one to five horizontally-distributed building blocks. Fig.4(b) shows the corresponding minimum-depth binary slicing tree.

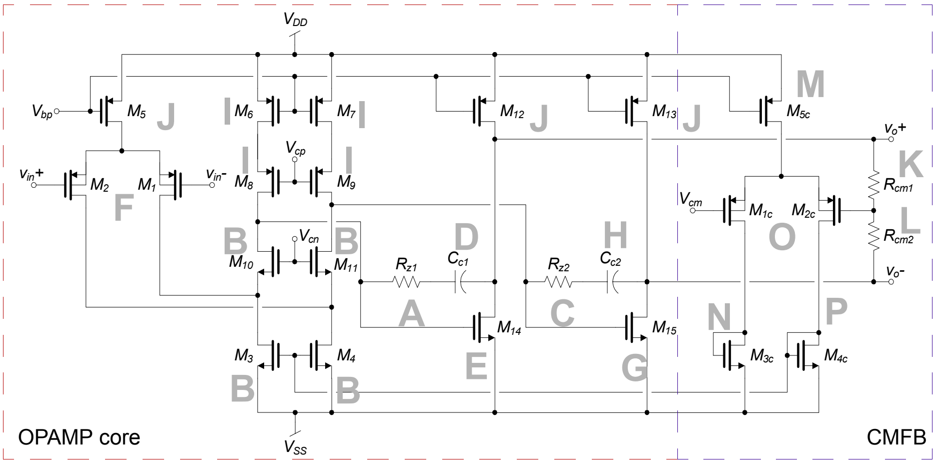

Figure 3: Schematic of the opamp example.

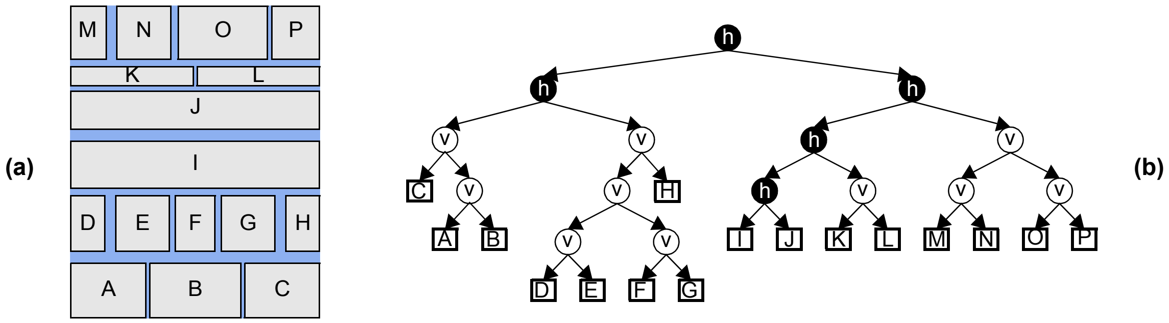

Figure 4: (a) Representation of the floorplan and (b) binary slicing tree representation of the opamp in Fig.3.

The relation between the leaf cells of the slicing tree and the devices (or group of devices) of the opamp is also shown in the schematic of Fig.3. The geometric parameters considered are the number of folds of transistors and resistors, and the side size of the unit rectangular capacitor implementing C_{c1} and C_{c2}.

The first column of Table 1 shows the specification list for each of the two experiments. Three groups of specifications have been defined. The first group, from *DC gain* to *Cload*, represents the electrical performance requirements and load conditions. The second group, from aspect ratio (*AR*) to maximum height value (H_M), is the set of geometric restrictions. The last group is composed of the design objectives to undertake once the electrical and geometric specifications are met, namely minimization of area and minimization of power consumption. In both experiments, the fabrication process selected is a three-metal, double-poly, 0.35–μm CMOS process. HSPICE was used as simulation tool in the optimization loop.

The experimental results are shown in columns third and fourth of Table 1. Notice that every optimization experiment has reached the required electrical performance specs and all geometric goals have been successfully addressed, whereas minimization of layout area and power consumption has been carried out. For illustration purposes, area loss is also listed. From all the possible sizes of the opamp layout at the final solution meeting the geometric goals, the described approach can ensure minimization of area and area loss. The physical implementations of the solution of each experiment are shown in Fig.5. The number of required iterations for experiment #1 was 1708, which took 409.92 seconds of CPU time. Experiment #2 took 1157.76 seconds to perform 4834 optimization iterations. As these figures demonstrate, taking into account geometric features of the circuit layout earlier in the optimization process, allows to obtain optimum results in terms of area and area loss and, also, with regard to certain user-defined goals, such as specific aspect ratio and width or height specified values. The average execution time of the *GC* module was is 0.2 seconds of CPU time[4]. As the *GC* module

Table 1: Achieved electrical performance.

Specification	**Goal**	**Exp.#1**	**Exp.#2**
DC gain ≥	85dB	110.37dB	100.82dB
Unity-gain frequency ≥	50MHz	65.79MHz	50.37MHz
Phase margin ≥	50°	57.26°	68.94°
DC gain (CMFB) ≥	85dB	114.07dB	111.04dB
Unity-gain frequency (CMFB) ≥	25MHz	27.64MHz	27.52MHz
Phase margin (CMFB) ≥	50°	50.13°	52.25°
Output swing ≥	5.5V	5.86V	5.76V
Slew rate ≥	55V/μs	59.15V/μs	59.96V/μs
Rload	50kΩ		
Cload	5pF		
AR	1.0 (Exp.#1)	1.0 (165.7/165.85)	--
Δ_{AR}	0.1 (Exp.#1)		--
W_M	300 (Exp.#2)	--	186.9
H_M	200 (Exp.#2)	--	138.1
Power	Minimize	3.18	2.98
Area	Minimize	27481.0	25811.0
Area Loss (% of total area)	Minimize	1.94	0.57

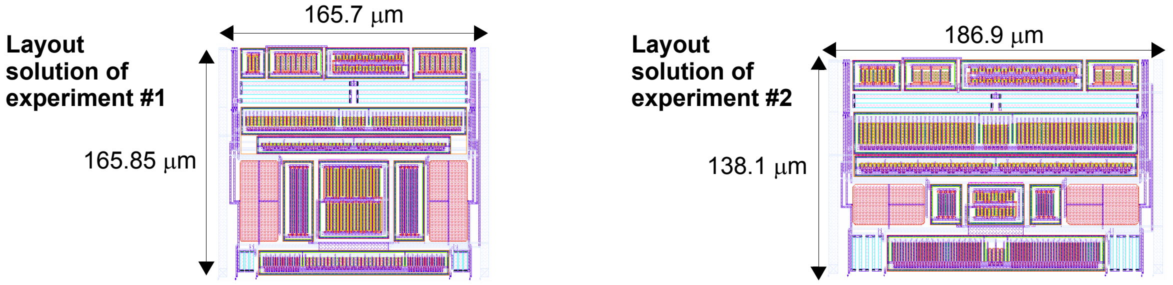
Layout solution of experiment #1
165.7 µm
165.85 µm
Layout solution of experiment #2
186.9 µm
138.1 µm

Figure 5: Resulting layouts of the geometrically-constrained sizing experiments.

operates during the optimization process, 0.2 seconds of CPU time are added to the simulation time at each iteration of the optimization

4. PARASITIC-AWARE SIZING

All integrated circuits contain passive electrical elements not required for their normal operation. These **parasitic** components, usually jeopardizing the correct performance of the integrated circuit, are consequence of the “imperfect” nature of the materials used in the physical implementation of the circuit. Layout-induced parasitic components can be of crucial importance for AMS circuit design. Actually, many of the differences between the measurements made on silicon circuits and the prediction of circuit simulation can be traced back to these parasitic components. During sizing of AMS circuits, the designer should consider the impact of the circuit parasitics in a balanced way: if this impact is underestimated, then the design process may result in sub-optimal solutions; on the other hand, if the value of the circuit parasitics is overestimated, then the circuit will perform as expected but with an unnecessary waste of power and area.

The parasitic extraction process aims at detecting and estimating all (or, at least, the most important) capacitive, resistive, and inductive parasitics. Different methods exist for this parasitic estimation process, being numerical, analytical, geometrical, and table look-up the most important ones. The trade-off between accuracy (i.e., how the simulation results from an extracted circuit layout compare to the on-chip measurements) and speed (i.e., CPU time it takes to obtain all parasitic values) is an important concern in parasitic estimation. Numerical methods, though very accurate in estimating parasitics, are slow due to the high computational resources demanded. Table lookup methods provide reasonably accurate estimates, but the memory storage requirements grow rapidly with the number and range of the parameters describing a given interconnect configuration. On the other hand, analytical-geometric methods are fast and reasonably accurate; actually, the simpler the parameterized formula, the faster the extraction is at the cost of lower accuracy.

All in all, the ultimate goal of parasitic extraction is to obtain a device-level description or netlist of the circuit layout that includes the circuit parasitics in order to perform a simulation of the circuit behavior and its deviation from nominal performances. Estimation of most layout-induced parasitics requires detailed knowledge on the circuit layout, either routing, placement or device realization, a knowledge that is not typically available when the designer undertakes the electrical sizing problem.

4.1 Extraction of parasitics in the design process: review of previous approaches

Fig.6 illustrates the traditional design flow, where layout generation follows the sizing process. The layout is optimized without real quantification of the circuit performance degradation due to parasitic effects. Afterwards, the design flow continues with formal verification (design rules check, electrical rules check, and layout vs. schematic check, not shown in Fig.6) and parasitic extraction, followed by a performance verification of the extracted layout. If the difference between the intended circuit performance and the extracted layout performance is not permissible, one or more re-design loops are needed. This trial-and-error approach reveals several drawbacks. First, no systematic method exists that guides the correction process of the circuit sizing, its layout, or both: the problem may not be necessarily due to a single parasitic, but a combined effect of several of them. Second, the list of extracted parasitics may be so large that such correction process can be extremely laborious. Third, the number of iterations in the re-design loop may be rather large for commonly designed analog circuits. This all means that efficiency of the re-design process relies, largely, on the designer’s expertise and the heuristics used, and, consequently, re-design iterations can be very time-consuming.

---

4. This time has been measured on a PentiumIV@1.3GHz Pc with a 512MB RAM.

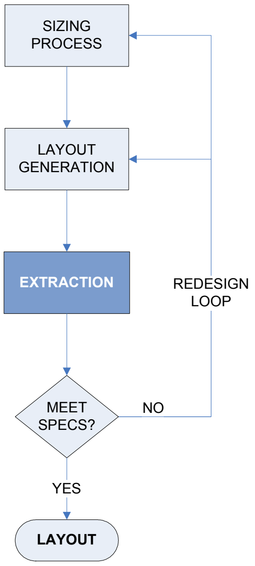

Figure 6: Traditional flow.

With the first analog placement and routing tools, the extraction process has been, however, gradually included in the layout generation phase. The ILAC [14], for instance, routes nets following a given priority order (power nets, sensitive nets, and, last, noisy nets) with distance and crosstalk penalties. The KOAN/ANAGRAM II [15] performs placement and routing based on weighted parasitics minimization. Both tools use a very simple analytical-geometric extraction method to estimate parasitics within their layout optimization loops.

Later approaches try to reduce the number of re-design loops of Fig.6 by including parasitic extraction directly in the layout generation phase and constraining both placement and routing with quantitative knowledge on parasitics and the relation between these and the induced performance degradation. Two alternatives exist here: indirect and direct methodologies, both usually referred as constraint-driven approaches [16]. Note also that all these methods use an already sized design. Even if the designer uses direct methods, which should return correct-by-constructions layouts, circuit sizing needs, however, to leave sufficient performance margin to accommodate the eventual layout-induced degradation on performance, which could lead to unnecessary waste in power and area. This suggests that layout parasitics should be somehow included in the very sizing process.

4.2Complete layout-aware sizing methodology

Parasitic-aware circuit sizing aims at improving the automatic design of AMS circuits by (1) avoiding unsystematic and time-consuming iterations between layout extraction and circuit re-design, and (2) arriving at better optima of the circuit design space, which is possible because parasitics are included early in the sizing process rather than merely considering them in subsequent re-design loops.

As explained above, accurate extraction of parasitics requires knowing the circuit layout in full detail, which involves obtaining information on the implementation style of each device, the interconnect structure, and their relative positioning. Furthermore, this layout knowledge must be generated or retrieved at each iteration of the sizing process. Therefore, whichever the method used to obtain this knowledge, it must be rapid enough to prevent circuit sizing from being prohibitively long. A way to obtain such knowledge is to actually generate the layout at each iteration of the sizing process, thereby being critical to reduce the CPU time that such layout generation requires. For this reason, constraint-driven approaches are currently too slow to be called in the inner loop of an automated parasitic-aware sizing process. For instance, the tools reported in [16] yield CPU times from 550 to 800 seconds for opamp-like circuits. Considering that a typical circuit sizing involves a few thousand iterations, and neglecting the CPU time for the rest of processes (simulation, extraction, and so on), it would take several days to complete the parasitic-aware circuit sizing, which can be comparable (if not worst) to manual design. On the contrary, procedural layout based on templates is, as said earlier, a more suitable solution for parasitic-aware circuit sizing since it is possible to have a complete and detailed description of the circuit layout (even without actually instancing it, for a template is a fully parameterized object, the parameters depending on design variables and technological constants) and the CPU time for layout instancing is typically negligible when compared to said constraint-driven approaches.

Fig.7 outlines the general design flow of parasitic-aware circuit sizing. The sizing engine returns first relevant electrical information on the visited point of the design space being explored: MOS transistor sizes, passive device values, and biasing currents and voltages. This information is required by the procedural layout generation to enable parasitic estimation. The *GC* module (see Section 3.2) determines the value of geometric parameters (e.g., as the number of MOS transistor folds, the exact shape of passive devices, or the optimum width of the routing wires) prior to parasitic extraction. These geometric parameters may be constrained by some kind of user's defined restrictions, like the aspect ratio of the circuit layout. Such information is then processed to estimate the layout-induced parasitics. Afterwards, the electrical description of the circuit is completed with the parasitic estimates and the overall performance is checked against intended performance specifications, which requires

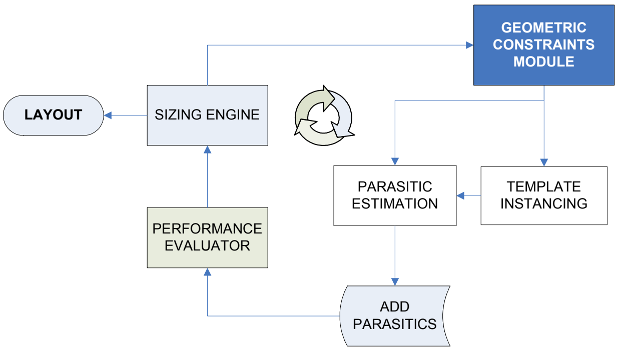

Figure 7: Parasitic-aware sizing flow.

evaluation of the circuit performance. A correct layout (that is, one whose degradation on the circuit performance is acceptable) can be eventually generated.

The salient characteristics of reported approaches on parasitic-aware sizing are listed in Table 2. The table collects, for each reference, the sizing method, the performance evaluator, and the extraction technique used. It specifies whether layout generation is needed or not, as well as the process employed (if so) to decide on the geometric parameter values.

In this paper, two approaches are described, both using a simulation-based optimization process with a numerical simulator in the loop (to perform circuit sizing) and the geometrically-constrained sizing technique presented in Section 3. The first solution relies, on the one hand, on accurate calculation of the MOS transistor diffusion areas and perimeters by using the appropriate analytic equations; on the other hand, estimates of the routing parasitics are obtained by sampling the layout template [18] [19]. Prior to circuit sizing, a number of different instances of the circuit layout template are extracted with a commercial off-the-shelf extraction tool. To do so, each one of the n design variables is uniformly sampled with m data points. The instances are simulated (including parasitics), the critical parasitics are identified, and their values stored in a look-up table. Then, the value of the parasitic is retrieved from such table depending upon the value of the design variables. The problem with this technique is that, since complex analog circuits typically have a "large" number of design parameters, both generation time and storage requirements for the look-up table may be quite large. For that reason, only $N < m^n$ instances from all the m^n possible combinations are generated and extracted, and only critical parasitics (e.g., those in the signal path) are considered. Identifying which are the most critical parasitics and the most significant layout instances require a big deal of design expertise to be called upon.

To further improve the accuracy of the parasitic-aware sizing technique illustrated above, a dedicated, high-accuracy extraction process has been included in the optimization loop. After a selection of a new point of the design space, the geometric parameters are decided by using the module explained in Section 3. This module returns a file with parameter values for the instancing of the layout template. Afterwards, extraction is done by using an accurate off-the-shelf extractor, and, last, the design is simulated including the extracted layout-induced parasitics. From layout instancing to performance evaluation, all different tasks are carried out by using the resources available in the Cadence's *DFWII* environment, while selection of the geometric parameters and design optimization is done externally. Note that a solution obtained with this parasitic-aware sizing technique will feature the highest accuracy provided the extractor used is the most accurate. That is, the final sized circuit will meet specifications even considering the performance degradation caused by layout parasitics. In contrast, better accuracy implies longer CPU times employed for extraction and, possibly, longer simulation times due to the presence of a larger number of parasitics.

Table 2: Parasitic-aware sizing approaches.

Reference	**Sizing method**	**Performance evaluation**	**Estimation method**	**Layout generation**	**Geometric parameters**
[4]	Differential evolution genetic algorithms	Equations/numerical simulation	A-G [a] models	Yes	Equations
[5]	Plan-based	Equations	A-G[b] models	No	Linear programming
[6]	Simulation-based opt.	Numerical simulation	A-G[c] models	Yes	No
[20]	Knowledge & simulation-based opt.[d]	Numerical simulation (SPICE)	Analytical-geometric (AG) models	No	No
[21]	Simulation-based opt.	Numerical simulation (NG-SPICE)	Layout sampling[e]	No	No
[22]	Simulation-based opt.	Symbolic analysis	Accurate off-the-shelf extractor	Yes	No
This paper	Simulation-based opt.	Numerical simulation (HSPICE)	Layout sampling & fixed estimates+accurate diffusion models	No	Slicing tree algorithm
			Accurate off-the-shelf extractor	Yes	

a. Only the overlap and fringing routing parasitics of critical nets are extracted.

b. Neither interconnect coupling capacitive nor resistive parasitics are considered.

c. Only interconnect and substrate parasitics are considered.

d. Parasitic extraction is only carried out during simulation-based optimization.

e. Only capacitive parasitics are considered.

To illustrate parasitic-aware sizing, the three different sizing experiments have been carried on the same fully-differential two-stage operational amplifier shown above in Fig.3 (CMFB not included this time), in the same CMOS process, now with $Rload = 100\,\text{k}\Omega$ and $Cload = 8\,\text{pF}$ and performance specifications in Table 3. These experiments are:

- **Experiment I**. In this experiment, neither layout geometric constraints nor parasitic estimates are considered during the size process. Area minimization is pursued by assuming a direct relationship with device sizes and not using any information from the layout template.
- **Experiment II**. This experiment considers layout geometric constraints and parasitic estimates using the layout sampling technique. The aspect ratio required is $AR \approx 1$.
- **Experiment III**. In this last experiment, a dedicated, high-accuracy extraction tool, in combination with the geometric constraint module, is used to estimate layout parasitics (also pursuing $AR \approx 1$).

Third, fourth and fifth columns of Table 3 show the results of each experiment. For experiments I and II, the numbers between brackets correspond to the values of the performance features obtained after layout template instancing, extraction, and simulation. For experiment III, the numbers already correspond to extracted layout performance.

Table 3: Experimental results for the parasitic-aware sizing experiments.

Feature	Specified	Experiment I	Experiment II	Experiment III	Units
DC gain	>110	110.0 (110.0)	113.0 (113.0)	111.7	dB
Unity-gain frequency	>90	91.8 (89.8)	105.7 (105.4)	106.6.	MHz
Phase margin	>65	67.6 (63.4)	65.4 (65.0)	66.2	º
Output swing	>5.25	5.3 (5.3)	5.3 (5.3)	5.4	V
Slew rate	>40	46.9 (46.8)	57.2 (57.0)	56.7	V/µs
Power dissipation	minimize	7.3 (7.3)	8.0 (8.0)	8.3	mW
Total area	minimize	195.8 x 358.8	173.8 x 191.25	189.6 x 193.05	µm x µm
Aspect ratio	≈ 1	0.55	0.91	0.98	

Template instantiation of the resulting devices sizes are shown in Fig.8 (notice that all layouts have been captured with the same resolution and, therefore, the relative dimensions are real). It can be seen in Table 3 that the required nominal performance is accomplished in the three experiments. The exclusion of geometric layout constraints and/or parasitic estimates, however, results in additional re-design iterations (when, due to layout parasitics, the final design does not meet the required specifications –see the non-fulfilled phase margin of experiment I) or in hardly compact layouts with large empty areas when the sizing process is geometrically unconstrained.

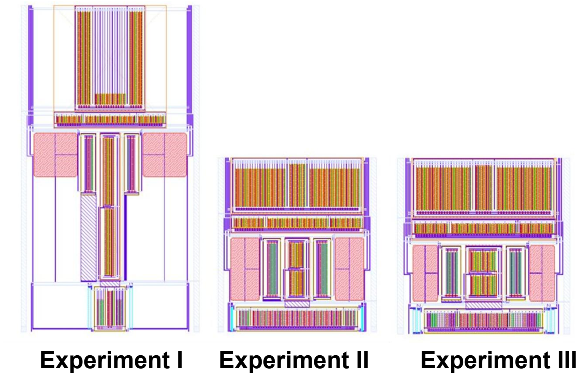

Figure 8: Layout instances of the experiments in Table 3.

The number of iterations executed in each experiment were 2116 (Experiment I), 2437 (Experiment II) and 3044 (Experiment III), with CPU times of 507.8 seconds (Experiment I), 612.31 seconds (Experiment II), and 880.3 seconds (Experiment III). Thus, using an off-the-shelf extractor requires around 15% more CPU time than when using the layout sampling technique (layout instancing and extraction takes 17% of the total sizing time).

5. CONCLUSIONS

This paper describes layout-aware synthesis methodology that minimizes the iterations between electrical and physical synthesis. The two techniques composing this methodology, namely geometrically-constrained sizing and parasitic aware sizing, allows to include both geometric concerns, such as area occupation or aspect ratio, and layout-induced parasitics, right in the sizing loop. Thanks to these modified sizing processes, the exploration of the design space is guided toward solutions that, once implemented at the layout level, yield user-defined geometric specifications and prove robust enough to layout-induced degradation caused by unwanted parasitics, rendering any iteration between sizing and layout generation unnecessary. Layout templates turn out, unlike other physical generation approaches, fundamental to layout-aware synthesis. Design examples here provided demonstrate the validity of both geometrically-constrained and parasitic-aware sizing techniques.

ACKNOWLEDGEMENTS

This work has been partially supported by project TEC2004-01752, funded by the Spanish Ministry of Education and Science.

REFERENCES

1. *Int. Technology Roadmap for Semiconductors. Edition 1999*. Available: http://public.itrs.net/files/1999_SIA_Roadmap/Design.pdf, 1999.
2. *Int. Technology Roadmap for Semiconductors. Edition 2001*. Available: http://public.itrs.net/Files/2001ITRS/Home.htm, 2001.
3. F. Medeiro, B. Pérez-Verdú, and A. Rodríguez-Vázquez, *Top-down design of high-performance sigma-delta modulators*. Boston: Kluwer Academic Publishers, 1999.
4. P. Vancorenland, G. Van der Plas, M. Steyaert, G. Gielen, and W. Sansen, “A layout-aware synthesis methodology for RF circuits,” in *Proc. of ACM/IEEE Int. Conf. on Computer-Aided Design*, 2001, pp. 358-362.
5. M. Dessouky, *Design for reuse of analog circuits. Case study: very low-voltage delta-sigma modulator*. Ph.D. Thesis, University of Paris VI, 2001.
6. H. Tang and A. Doboli, “Employing layout-templates for synthesis of analog systems,” in *Proc. of IEEE Midwest Symp. on Circuits and Systems*, 2002, pp. 505-508.
7. R. H. J. M. Otten, “Automatic floorplan design,” in *Proc. of ACM/IEEE Design Automation Conf.*, 1982, pp. 261-267.
8. *Virtuoso® parameterized cell reference. Product version 4.4.6*, Cadence Design Systems Inc., 2000.
9. *SKILL language reference. Product version 06.00*, Cadence Design Systems Inc., 2001.
10. L. Stockmeyer, “Optimal orientation of cells in slicing floorplan designs,” *Information and Control*, vol. 57, no. 2-3, pp. 91-101, Academic Press, May-June 1983.
11. H. Y. Koh, C. H. Sequin, and P. R. Gray, “OPASYN: A compiler for CMOS operational amplifiers”. *IEEE Tran. Computer-Aided Design*, vol. 9, no. 2, pp. 113-125, February 1990.
12. J. D. Conway and G. G. Schrooten, “An automatic layout generator for analog circuits,” in *Proc. of European Design Automation Conf.*, 1992, pp. 513-519.
13. H. Onodera and K. Tamaru, “Analog circuit placement – Branch-and-bound placement with shape optimization,” in *Proc. of IEEE Custom Integrated Circuits Conf.*, 1992, pp. 11.5.1-11.5.6.
14. J. Rijmenants, J. B. Litsios, T. R. Schwarz, and M. G. R. Degrauwe, “ILAC: an automated layout tool for analog CMOS circuits,” *IEEE J. Solid-State Circuits*, vol. 24, no. 2, pp. 417-425, April 1989.
15. J. M. Cohn, 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, no. 3, pp. 330-342, March 1991.
16. K. Lampaert, G. Gielen, and W. Sansen, *Analog layout generation for performance and manufacturability*. Boston: Kluwer Academic Publishers, 1999.
17. *Virtuoso® layout editor user guide. Product version 4.4.6*, Cadence Design Systems Inc., 2000.
18. R. Castro-López, F. V. Fernández, F. Medeiro, and A. Rodríguez-Vázquez, “Generation of technology-independent retargetable analog blocks,” *Analog Integrated Circuits and Signal Processing*, vol. 33, no. 2, pp. 157-70, Kluwer Academic Publishers, November 2002.
19. F. V. Fernández, F. Medeiro, R. del Río, R. Castro-López, B. Pérez-Verdú, and A. Rodríguez-Vázquez, “Design methodologies for sigma-delta converters,” in *CMOS telecom data converters*, A. Rodríguez-Vázquez, F. Medeiro, E. Janssens, Eds. Boston: Kluwer Academic Publishers, 2003, pp. 15-1, 15-38.
20. H. Onodera, H. Kanbara, and K. Tamaru, “Operational-amplifier compilation with performance optimization,” *IEEE J. Solid-State Circuits*, vol. 25, no. 2, pp. 466-473, April 1990.
21. A. Agarwal, H. Sampath, V. Yelamanchili, and R. Vemuri, “Accurate estimation of parasitic capacitances in analog circuits,” in *Proc. of Design, Automation and Test in Europe Conference and Exhibition*, 2004, pp. 1364-1365.
22. M. Ranjan, W. Verhaegen, A. Agarwal, H. Sampath, R. Vemuri, and G. Gielen, “Fast, layout-inclusive analog circuit synthesis using pre-compiled parasitic-aware symbolic performance models,” in *Proc. of Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 2004, pp. 604-609.