On the Suitability and Development of Layout Templates for Analog Layout Reuse and Layout-Aware Synthesis

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

Accelerating the synthesis of increasingly complex analog integrated circuits is key to bridge the widening gap between what we can integrate and what we can design while meeting ever-tightening time-to-market constraints. It is a well-known fact in the semiconductor industry that such goal can only be attained by means of adequate CAD methodologies, techniques, and accompanying tools. This is particularly important in analog physical synthesis (a.k.a. layout generation), where large sensitivities of the circuit performances to the many subtle details of layout implementation (device matching, loading and coupling effects, reliability, and area features are of utmost importance to analog designers), render complete automation a truly challenging task. To approach the problem, two directions have been traditionally considered, knowledge-based and optimization-based, both with their own pros and cons. Besides, recently reported solutions oriented to speed up the overall design flow by means of reuse-based practices or by cutting off time-consuming, error-prone spins between electrical and layout synthesis (a technique known as layout-aware synthesis), rely on a outstandingly rapid yet efficient layout generation method. This paper analyses the suitability of procedural layout generation based on templates (a knowledge-based approach) by examining the requirements that both layout reuse and layout-aware solutions impose, and how layout templates face them. The ability to capture the know-how of experienced layout designers and the turnaround times for layout instancing are considered main comparative aspects in relation to other layout generation approaches. A discussion on the benefit-cost trade-off of using layout templates is also included. In addition to this analysis, the paper delves deeper into systematic techniques to develop fully reusable layout templates for analog circuits, either for a change of the circuit sizing (i.e., layout retargeting) or a change of the fabrication process (i.e., layout migration). Several examples implemented with the Cadence's Virtuoso tool suite are provided as demonstration of the paper's contributions.

Keywords: Physical Synthesis, Design Reuse, Layout-Aware Synthesis, Procedural Layout Generation.

1. INTRODUCTION

Electronic Design Automation (EDA) is a key factor for fast and efficient development of complex electronic designs. Nowadays, when a reduction of the design productivity –dropping behind the available capacity to integrate due to increasingly tight product-to-market requirements and design complexity– is jeopardizing the phenomenal evolution of the semiconductor industry [1], EDA is, probably, more urgently required than ever. Unlike digital circuits, where, even though being far from the 'push-the-button-and-forget-it' era, there is a considerable stream of EDA resources at all stages of design, the analog domain is particularly impacted by a lack of EDA tools and methodologies that may help closing the gap between productivity and complexity. The very nature of analog circuits (much more heterogeneous, hierarchically loose, and extremely sensitive to different sources of 'noise', to name but a few differences with digital circuits) make design a nearly handicraft process and automation, therefore, becomes much more difficult.

Keeping the track of digital design automation, the paradigm of **reuse-based design** has been recently proposed as a complementary solution to speed up the analog design process [2]. Reuse, in this context, is the ability of using previous design knowledge, experiences, and databases to implement a different design, perhaps in a different fabrication process. Analog reuse, however, needs a differing set of solutions than those applied in the digital domain, just for the same reasons that digital and analog automation maturity levels differ. Particularly important is the creation of truly reusable circuit layouts, for this is one of the most intensive and time-consuming design tasks.

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: 10.1117/12.607932

Another complementary, novel approach to boost the analog design process focuses the issue of avoiding or completely removing any iterations between electrical and physical synthesis. Such iterations do traditionally take place when, after the performance of the circuit, including the unavoidable layout-induced parasitics, is verified, one or more unacceptable deviations from the initially expected performance have been found. This *modus operandi* has been so far considered as a standard in analog design. The novel approach, known as parasitic-aware synthesis [3] [4], consists in fully or partially embedding layout synthesis in electrical synthesis (also known as circuit sizing), so that detailed physical information can be considered at intermediate steps of said electrical synthesis. A much more complete approach considers not also the inclusion of parasitic details, but also the introduction of geometric details. By doing so, geometric aspects of the circuit's layout, such as the occupied area, can be reliably optimized during the electrical synthesis. This approach is referred here as geometrically-constrained electrical synthesis. On the other hand, as geometric features have influence upon the value of the layout-induced parasitics (e.g., varying the number of folds of a transistors changes the value of the diffusion areas and, therefore, alter the value of the diffusion parasitic capacitances), both techniques should simultaneously be applied; electrical synthesis is then known as **layout-aware synthesis** [5].

These two design methodologies, layout reuse and layout-aware synthesis, impose its own set of requirements on layout generation. This paper addresses the issue of finding out which is the most appropriate physical synthesis method that best deals with layout reuse and layout-aware synthesis. The paper is organized as follows. Section 2 and Section 3 analyze the requirements that the reuse-based design paradigm and the layout-aware synthesis methodology respectively impose on physical synthesis. Section 4 reviews existing approaches to layout synthesis and explores the suitability of layout templates in the light of such review. A methodology for layout template development is described in Section 5 and several examples are presented in Section 6. A summary discussion on the benefits and drawbacks of layout templates is given in Section 7. Last, conclusions are drawn in Section 8.

2. THE REQUIREMENTS OF ANALOG LAYOUT REUSE

From the layout point of view, design reuse implies two different scenarios:

(a) Reuse of the circuit layout database for changes in the circuit performance specifications. This concept of design reuse has one limitation: the specifications changes must be such that the new specifications can be addressed by using the same circuit architecture/topology. This does not mean, however, that the required changes translate into minor adjustments at the layout level at all [6]. Quite the opposite, specifications changes, though within the circuit's achievable behavior, may translate into drastic modifications of the circuit device sizes and biasing conditions, and, thereby, in the circuit layout. Whatever the layout generation approach is used, it has to solve the problem of how to accommodate these specification changes. In this scenario, layout reuse is called **layout retargeting** as the circuit target performance is modified and the previous working circuit architecture/topology is reused.

(b) Reuse of the circuit layout database for a change of the fabrication technology. In this case, layout reuse is known as **layout migration**, as the layout database is moved from the technology it was designed for, to a different goal technology, perhaps from a different foundry[1].

To reuse a circuit layout manually, either for a change in the device sizes or for a change of the fabrication process, could become a quite laborious and slow task. Actually, the great specificity of analog designs is the main factor that makes direct layout reuse utterly unfeasible. To actually understand how a reusable layout can be created, it is first essential to grasp the implications that retargeting and migration have on automated layout synthesis.

2.1 Layout retargeting

Layout retargeting, performed when any of the circuit devices and/or any of the biasing conditions need to be modified to address the changes in the circuit performance specifications, entails the following two different aspects: first, several characteristics of analog layout quality may result spoiled, so they have to be carefully treated and maintained; second, the layout has to remain compliant with the process design rules. This latter aspect is covered in Section 2.2. The following analyzes the former aspect in more detail.

1. In a sense, layout retargeting can be seen as a component of layout migration, as changing the fabrication process, while trying to obtain the same circuit behavior, would likely require to adapt the layout to new device sizes as well (actually what layout retargeting aims at). Nevertheless, layout migration will be considered here, for the sake of simplicity, as a stand-alone aspect of analog layout reuse, meaning only the adaptation process of the layout database (i.e., database migration), and not of the circuit device sizes, to another fabrication process, with different design rules and mask layers.

The analog layout characteristics that need close attention during the course of layout retargeting are:

1. **Device mismatch**. All devices in an IC occupy the same piece of silicon, therefore suffering from the same manufacturing imperfections. There are devices which are specifically constructed to keep a known constant ratio between them and they are thus called matched devices. Six major layout geometric factors can affect the matching of identical devices [7]-[12]: size, shape, symmetry, separation, orientation, and boundary. To minimize the effect of device mismatch, the layout designer typically follow several guidelines [7] [10], such as the use of common-centroid structures, which should be preserved as layout retargeting is performed.
2. **Loading and coupling effects**. The physical nature of materials used in the fabrication process introduces capacitive and resistive *parasitic* elements[2]. However, the amount of effort needed to control these parasitic effects is indeed considerable since extremely low-level geometric details of the layout of individual devices can have a major impact on the circuit performance. Furthermore, parasitic elements cannot be fully predicted early in the design process, because the layout is not complete yet. Over-estimation of the parasitics results in wasted area and power, and under-estimation leads to specification non-fulfillment. It is then crucial that parasitic effects have to be taken into account during the design process[3] and that the selected layout synthesis method provides ways to minimize their impact. Layout can also introduce unexpected signal coupling between the circuit nodes, which may inject unwanted electrical noise and even destroy the circuit stability due to unintended feedback [7]. This capacitive coupling effect, known as *crosstalk*, may appear between two wires running in parallel over a long distance, or in two wires crossing at different levels. Capacitive and resistive coupling can also appear by means of substrate coupling [13].
3. **Reliability**. This characteristic refers to the total time that an IC can provide perfect operation and depends, at a high extent, on the quality of the IC layout. For instance, preventing a serious source of reliability loss like electromigration from occurring, can be attained by properly adjusting the wire width. Contact and via holes, making the current flow from geometries on different layers, should also be adjusted to minimize the resistance to such current flow, and so must done be during layout retargeting.
4. **Area occupation**. Minimizing the area occupation is usually a design concern in analog circuit design since it may lead to more integrated functionality and to eventually lower chip fabrication costs. Attaining a compact layout with minimal unused area can also improve the chip area usage. Therefore, when layout retargeting is required and changes in the circuit parameters result in changes in the circuit layout (small or large), both area and unused area should be kept as small as possible. Another important factor to make the assembly of several circuit layouts easier, is the aspect ratio (i.e., $\text{width}/\text{height}$) of the circuit layout.

All the characteristics described above are critical to analog layout design and, in this sense, a set of rules and guidelines should be followed to enhance the quality of the layout. The relevant conclusion is, actually, that whatever the method selected to create the layout-reusable analog block, it has to efficiently cope with all these noteworthy issues.

2.2 Layout migration

Circuit layouts are created by arranging a set of geometric shapes, each shape made of a particular mask layer (e.g., polysilicon or different metal levels), to form the devices (e.g., transistors, resistors, capacitors) present in the circuit device-level description. Each fabrication process stipulates its own set of mask layers and its own set of layout design rules, according to which all the circuit's devices and interconnections have to be laid out. Suppose a circuit layout made on one technology, T_1. The main problems arising when trying to port a layout from said technology to a different, goal technology, T_2, are [14]:

1. **Variation of the geometric process parameters**. Foundries provide sets of design rules and guidelines which encapsulate the fabrication geometric constraints (e.g., like the minimum feature size), and which the circuit layout must comply with. When the technology changes or even when the same technology evolves, these rules and guidelines may also change. A violation of any of these rules may lead to complete invalidity of the circuit layout.
2. **Variation of the electrical process parameters**. Electrical process parameters define the electrical characteristic of the process layer materials. Typical examples are the area and perimeter capacitance of poly-insulator-poly

2. At sufficiently high frequencies, inductive effects arise as well.
3. At certain phases of the design process it is possible to roughly estimate the parasitic elements (via area and perimeter measurements) but this estimation is not sufficient. That is why parasitic-aware synthesis has been proposed.

capacitors, the sheet resistance of the poly mask layer, the maximum current density of metal layers, etc. Calculations made in technology T_1 may be totally impractical in technology T_2.

3. **Variation of mask layers**. The set of available mask layers may vary from T_1 to T_2. Typical problems are:
   - *Number of routing layers*: it is quite common that T_1 and T_2 have a different number of metal routing layers (metal-one, metal-two, metal-three, and so on). There are not serious problems when T_1 has less routing layers than T_2, since every wire in T_1 has a counterpart in T_2. On the other hand, i.e., when T_2 has less routing layers, it becomes impossible to perform the layout migration, unless the original layout does not exhaust all the routing layers, and it uses as many (if not less) routing layers as those available in T_2.
   - *Device mask structure*: from one technology to another, the way or style atomic devices are laid out may also change. For instance, NMOS transistors in CMOS processes typically need a P+ diffusion mask layer. In many processes, it is not necessary to explicitly draw this layer, whereas, in other processes, it is required. The problem then arises when moving the NMOS layout from the former to the latter process.

As with the layout retargeting issues, the layout-reusable analog block must be created so that layout migration can be seamlessly and rapidly performed.

3. THE REQUIREMENTS OF LAYOUT-AWARE SYNTHESIS

As explained earlier, the underlying idea behind layout-aware synthesis is to bring layout generation into the very sizing process, so that circuit automated sizing is carried out with enough information about layout-induced parasitics and 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.

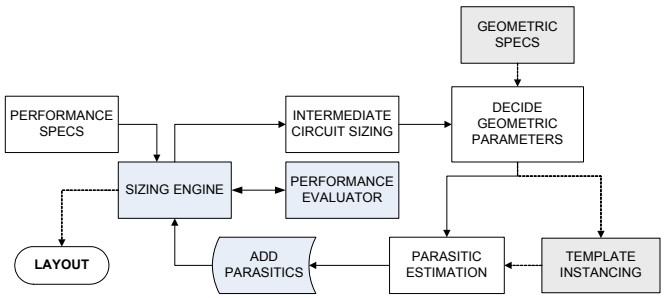

Figure 1: Layout-aware sizing flow.

The flow of layout-aware synthesis is depicted in Fig.1. The sizing process, carried out either by means of a knowledge-based or an optimization-based approach [6], begins with the circuit performance specifications (defined here as restrictions, involving inequalities, such as $<$ or $\geq$, and objectives, such as power consumption minimization). Then, either through a mapping of the performance specifications to device sizes (knowledge-based sizing) or through an iterative exploration of a pre-defined design space (optimization-based sizing), the sizing engine provides an intermediate circuit sizing. Afterwards, geometric parameters (e.g., the parameter controlling the number of fingers of a folded MOS transistors or one of the sides of a rectangular capacitor whose capacity has been given by the sizing engine) must be decided considering both area minimization and a set of user-defined geometric objectives such as the aspect ratio or the maximum layout width or height[4]. This decision-making process requires that the arrangement of the components of the layout (known as floorplaning) as well as the implementation style in which each component will be laid out, must be known beforehand. With such information, the task of finding adequate geometric parameter values to minimize a function of the circuit width and height, known as *floorplan sizing* problem can be tackled [15]. Different approaches exist to solve this problem, but all realizable choices are based on a slicing-style layout floorplan. Otherwise, the time required to solve the problem can be unaffordable, since it becomes a *NP*-complete problem [15].

Once geometric parameters have been decided, the inclusion of parasitics can be carried out either through parasitic modeling or through layout generation and subsequent layout extraction. Accurate estimation of parasitics requires knowing the circuit layout in full detail, which involves obtaining information on the implementation style of each device, the interconnect structure, as well on their relative positioning (placement). Furthermore, this layout knowledge may be required to be generated or retrieved at each iteration of an optimization-based sizing process. Therefore, whichever the method used to obtain this knowledge, it must be rapid enough to prevent circuit sizing from being prohibitively long. With

4. If the task of finding which values of the geometric parameters best optimize these geometric features is completely left to the sizing engine, the mapping or exploration may become over-constrained and will possibly take much longer.

parasitics accurately estimated and added to the circuit netlist, an evaluation of the circuit performance is carried out to quantify how the circuit deals with initial specifications. In case the resulting performance is not acceptable, the sizing process is re-entered, either guided with convergence criteria [3] or by simply moving to another point of the design space if using optimization-based sizing [14].
Setting aside the fact that the optimization engine has to be able to incorporate design knowledge and use it during the sizing process, two requirements can be drawn from the previous analysis. First, detailed information on the circuit layout implementation must be required early in the sizing process. Second, layout generation must be rather rapidly accomplished with respect to the sizing process itself.

4. ANALYSIS OF LAYOUT SYNTHESIS APPROACHES

The following review is by no means intended to be exhaustive (the interested reader is referred to excellent reviews on analog layout generation in [6] and [7]). Its sole objective is to find out which of the full-custom layout methods is best suited to deal with the requirements of analog layout reuse and layout-aware synthesis explained above.
A typical flow of the full-custom layout process for analog circuits is depicted in Fig.2. The input to the layout process is a circuit description, typically a completely sized netlist, with all device sizes and geometrical parameter values. Technological information is also used all throughout the layout generation.

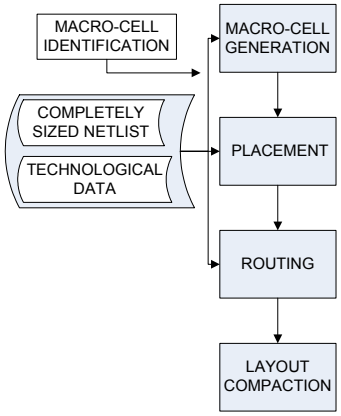

Figure 2: Typical analog layout design flow.

The first step consists in the generation of all the components of the circuit. At the cell level (e.g., an operational amplifier), these components are groups of one or more devices (e.g., mirror or cascode CMOS structures) known in the literature as *modules, structural entities*, or *macro-cells*. Each macro-cell can be generated in several ways, all electrically equivalent, called *geometric variants* (e.g., a transistor differential pair may be laid out in a 1-dimensional or 2-dimensional common-centroid style) [17]. At higher levels, the layout components can be functional blocks as well, which have also been generated by using device groupings at the cell level. The next step is the placement of every component, considering a wide set of analog constraints to obtain a better result. Then, in the routing phase, the placed components are interconnected, in accordance with the circuit connectivity provided earlier in the flow. After all components haven been routed, a compaction of the whole layout may take place, but it also can be regarded as an integral part of the placement and routing phases.

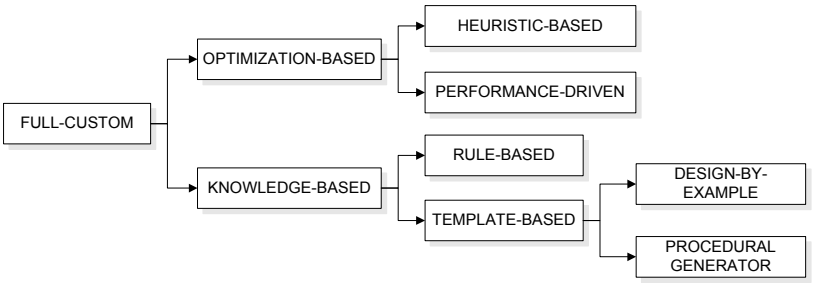

Figure 3: Taxonomy of CAD tools for layout generation of analog circuits.

Methods for generation of full-custom analog layout focus either on automating one or more of the different steps involved in the process, or on providing the layout as a whole single process. Whatever the focus, these methods can be broadly classified, like electrical synthesis, into two different approaches: **optimization-driven** approaches and **knowledge-driven** approaches [17]. Fig.3 shows a taxonomy of these two approaches.

4.1 Optimization-driven approaches

Optimization-driven approaches aim at automatically generating the layout while striving to minimize the layout-induced errors by means of an optimization algorithm. Following a digital-like approach, placement and routing stages of the layout generation are carried out by such an optimization program according to a certain cost function. This cost function typically considers minimization of some design aspects such as area and net length, while penalizing violation of some analog design constraints, such as device mismatch, loading capacitances, and crosstalk. The quality of these optimization-driven tools is mainly determined by the efficiency of the optimization algorithm and the set-up of the cost function.
Depending upon the way of deriving the cost function and dealing with constraints on analog performance, two categories are usually considered [17]. The first group is composed of **heuristic-based** approaches. Layout-induced errors are taken

into account by classifying nets according to their sensitivity and circuit function. Although optimizing the circuit performance, these approaches do not use a systematic way to generate net sensitivities and to handle performance constraints. Therefore, they may yield solutions that do not meet performance constraints after the layout is verified. Time-consuming layout-extraction-verification spins may thus be necessary. Besides, there is no way to identify which parasitics are mostly degrading the circuit behavior and, therefore, to know which changes are necessary. Examples of these tools are ILAC [18], and KOAN/ANAGRAMII [19]. Alternative approaches for placement of MOS transistors divide this step into device stacking and stack placement. This is done by using several heuristic algorithms rendering the circuit as diffusion graphs of connected drains and sources.

A notable improvement is accomplished by the other group of approaches, whose operation is based on **performance-driven** optimization of the circuit layout (also known as constraint-driven optimization). Unlike heuristic approaches, where no quantification of the performance degradation is done, performance-driven tools try to measure the layout-induced degradation on the circuit performance and keep it below desired margins. In this way, the impact of each layout parasitic is weighed out according to its effect on the circuit performance [20]. The first contributions reported were to perform channel routing [21]. In these works, the effect of layout parasitics are modeled by using sensitivities and, then, the performance constraints for the circuit are mapped to a set of constraints on the layout parasitics. Later approaches showed that this intermediate mapping could be skipped [16] [17].

The main advantage of optimization-driven tools is their *generality*: in principle, they can be applied to any analog or mixed-signal circuit. The drawbacks, however, are the complexity of the optimization problem (even for the simplest problems, these are *NP*-hard problems), the difficulty of the cost-function set-up, and the large turnaround time.

4.2 Knowledge-driven approaches

These approaches try to store and exploit the knowledge required to create the analog layout. This knowledge refers to the procedures that expert layout designers use to improve the quality of the layout, and spans a wide variety of techniques, from specific placement strategies used to improve device matching (e.g., complex common-centroid arrays) and minimize the layout area, to routing techniques to minimize the loading effects. Since this specific knowledge is to be stored and used whenever necessary, this approach is mainly intended to **reuse** previous experiences of expert layout designers.

Knowledge-driven approaches are specifically developed to generate the layout of fixed architectures/topologies. This means that the input information is not only a netlist of the sized circuit (see Fig.2), but also a description of the layout itself as well as valuable layout knowledge. Knowledge-driven layout generation is not as complex as the optimization-driven one, as placement and/or routing are specified in advance.

There are two types of knowledge-driven approaches, namely rule-based and template-based approaches. **Rule-based** approaches store the layout knowledge in a customizable rule set to be obeyed during layout placement and routing. A clear example of this approach is ALSYN [22]. Although every user can adapt the set of controlling rules to his/her own needs, the quality of the layout largely depends on the quality of this set of rules. Besides, the rules are difficult to formulate if they are intended to be general and context-independent.

Template-based tools are also developed to best use layout designers' expertise. The underlying idea is to capture this expertise in a pattern or **template** that specifies all necessary device-to-device and device-to-wire spatial relationships. Besides, it must capture analog specific constraints like symmetry, device matching, and parasitic minimization. To generate a circuit layout from this template, which is called *layout instancing*, it is required to provide the value of a set of electrical and geometric parameters (e.g., the transistor width and length, or the maximum current density allowed to flow on a certain mask layer). The template can be generated either in a procedural or a graphical way. The latter way consists in capturing the layout knowledge from a template previously laid out by an expert designer. A typical example is the **design-by-example** approach presented in [23]. The example provided by the expert captures his/her knowledge (regarding device placement, routing wire trajectories, material types and widths, and position of macro-cell terminals). To generate a new layout, it is necessary to provide the required electrical parameters for each device, the sets of matched devices, and the geometric constraints (e.g., a desired aspect ratio). Starting from a fixed device placement, the tool derives all possible layouts (emerging from all possible device layouts, e.g., from different values of the number of unitary components of a MOS transistor). Then, an exhaustive optimization is executed to find the one that satisfies the specified geometric constraints. Finally, routing and compaction phases are carried out. This approach can produce good compact layouts in a moderate amount of CPU time (around 37 minutes for a 24-device operational amplifier), but the layout templates have to be updated for each new fabrication process, which requires additional effort.

Capture of layout knowledge through templates can also be done by using **procedural generators** [24]. The mechanisms to describe these procedural generators can be specific layout languages such as BALLISTIC [25] and MSL [26], or common spreadsheet interfaces [27], but both approaches are intended to **code** the analog-specific layout knowledge into the software itself. Although the coding effort can be high, this effort needs no longer be wasted when the device sizes and/or the fabrication process are changed. The process simply consists in the compilation of the coded template (when necessary) and the update of the device sizes as well as the fabrication process parameters.

4.3 Automated layout generation for layout reuse and layout-aware synthesis

Having all this in mind, the most suitable layout generation approach to both layout reuse and layout-aware synthesis turns out to be the knowledge-driven generation of procedural layout templates. The reasons supporting this conclusion are the following [28] [29]:

- *Layout templates are very efficient at handling design expertise*. Analog layout retargeting, as explained in Section 2.1, requires imposing several layout constraints based on accumulated design knowledge. These constraints cannot be easily considered by traditional placement and routing algorithms. On the contrary, layout templates can be defined, as shown in Section 5, as structures where user-defined constraints are easily stored.
- *Layout templates can be straightforwardly ported*. Full technology independence can be achieved by coding the procedural template generator using symbolic process parameters and mask layers. Therefore, neither scaling methods nor complex compaction techniques would be required to be applied. The main advantage of procedural layout with respect to scaling and compaction methods is its higher precision and speed, respectively.
- *Layout template instancing time is comparatively much smaller than layout generation with optimization-driven approaches.* For instance, the tools reported in [17] yield CPU times from 550 to 800 seconds for opamp-like circuits, while instancing their layout templates would take no more than 0.01 seconds of CPU time [14]. This allows shortening the overall design time while managing the inherent complexity of analog circuits.
- *Layout templates ease placement*. The layout generation procedure is simplified because the positions of the blocks in the template are stored according to pre-defined relationships embodying constraints from the layout expert that enhances the layout quality. Optimization-driven methods try to attain the same quality at the expense of time-consuming algorithmic techniques. Having the placement thus defined (especially if, as it will be explained below, it follows a slicing style approach), also eases the floorplan sizing problem, since binary slicing trees can be readily built and it is then possible to reckon every building block's shape to minimize certain geometric function, such as the area occupation or the aspect ratio.j
- *Layout templates permit searching for optimal block parameters while revealing the knowledge needed for estimation of layout parasitics in parasitic-aware sizing*. As said in Section 3, it turns out critical to reduce the CPU time of layout generation. Heuristic-based or performance-driven approaches are currently too slow for layout generation to be called within the circuit sizing process [17]. Consider, for instance an optimization-based where typically a few thousand iterations are required. 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. Using procedural layout template allows, on the other hand, fast generation of circuit layout since no time-consuming optimization algorithms are involved. Furthermore, it is possible to have a complete and detailed description of the circuit layout (placement and routing characteristics as well) without actually instancing it, for the template is a fully parameterized object, the parameters depending on design variables and technological constants. Therefore, modeling layout parasitics becomes also possible.

Despite these important benefits, procedural methods have two drawbacks, namely cost –the effort to generate every new template may largely exceed the effort to create, manually, the corresponding full-custom layout– and flexibility –large changes of the circuit performance may lead to a dramatic degradation of the layout regularity, aspect ratio, and area usage. Both issues will be discussed in Section 7.

5. A METHODOLOGY FOR LAYOUT TEMPLATE GENERATION

A layout template is a data structure that completely defines the physical implementation of a certain circuit architecture/topology without having detailed intelligence on actual device sizing. The most important factor common to all types of layout templates is that **experience and knowledge from expert layout designers can be stored in an orderly systematic way**. Therefore, *designer*'s *expertise* on analog layout can be reused when needed. This architecture/topology describes only which are the circuit components and how they are connected. The layout template does not contain information about

a specific circuit sizing or the fabrication process: quite the opposite, the layout template must be as generic as possible. The consequence is that the layout template must be a fully *parameterized* entity. The properties of a parameterized layout template are the following:

1. **Parameterized components**. The layout template can adapt different performance specifications –or, in other words, different device sizing– because each one of the circuit design parameters (e.g., transistor width and length) is a parameter of the template itself.
2. **Relative placement**. The location of every single block in the layout template must be a function of the location and dimension of the rest of its neighboring blocks.
3. **Relative routing**. As with relative placement, the physical implementation of the connections between all the circuit tiles must be stored in a relative way. Note that routing must be defined as a function of the block placement, of block dimensions, of block pin positions, and, to avoid wire crossing over, of the location of other routing wires.
4. **Technology independence**. Any reference to a particular fabrication process in the layout template has to be completely avoided, and turned completely generic. That is, all mask layers, relative placement, and routing have to be stored in a process-independent way. When the circuit layout is implemented in a particular technology, it must be able to adapt to both the technological design rules and the set of layout mask layers.
5. **Hierarchy**. The layout template is the physical implementation of a circuit at any hierarchical level. Therefore, the layout template may contain lower hierarchical levels within. Suitable procedures are therefore required for transmitting down the parameters of the *parent* block to its immediate hierarchically lower building blocks, correspondingly called *child* or *leaf* components.

To implement all these properties, the methodology described in this paper relies on two resources: the constraint graphs technique and a set of geometric-database procedures. Layout template generation is then organized in two stages, first, constraint graph generation to set relative placement and routing and, second, template coding using the set of procedures, by means of which parameterization, technology independence, and hierarchy are attained.

An optimal way of describing the structure of a layout template placement is by means of *corner-stitching* data structures and *constraint graphs* [30]. Fig.3 illustrates this type of description. The entire plane of the block layout is represented explicitly with rectangles called *tiles*. These tiles represent physical layers (e.g., metal or polysilicon), primitive devices (transistors, resistors, capacitors, inductors) and connectors (contacts and vias), an arrangement of devices or any other hierarchically higher circuit layout. The set of tiles define vertical and horizontal line segments or cuts. Each tile is linked to the rest of tiles by a set of pointers, called *corner stitches,* at two of their four corners, and related *geometric constraints*. As illustrated in Fig.4, these stitches are at the bottom-left corner and at the top-right corner.

Each stitch represents two coordinates, horizontal and vertical, so each tile is defined by four coordinates, left (l) and down (d), for the bottom-left stitch, and right (r) and up (u), for the top-right stitch. In this way, the arquitecture's/topology's floorplan can be represented by two planar graphs $G_H(\boldsymbol{V},\boldsymbol{E})$ and $G_V(\boldsymbol{V},\boldsymbol{E})$, called horizontal and vertical graphs respectively. A vertex V in $G_H(\boldsymbol{V},\boldsymbol{E})$ (or $G_V(\boldsymbol{V},\boldsymbol{E})$) represents a vertical (horizontal) cut of the floorplan. The vertices are ordered according to the distance of the corresponding cuts from LL (DD), the left-most (bottom-most) side of the floorplan rectangle, until the right-most side RR (the top-most side UU) are reached. Two vertices v_i and v_j in G_H (G_V) are connected by an arc e_{ij} directed from the former to the latter if there is a sub-rectangle in the floorplan whose left (bottom) and right (top) edges lie on the corresponding vertical line segments, respectively.

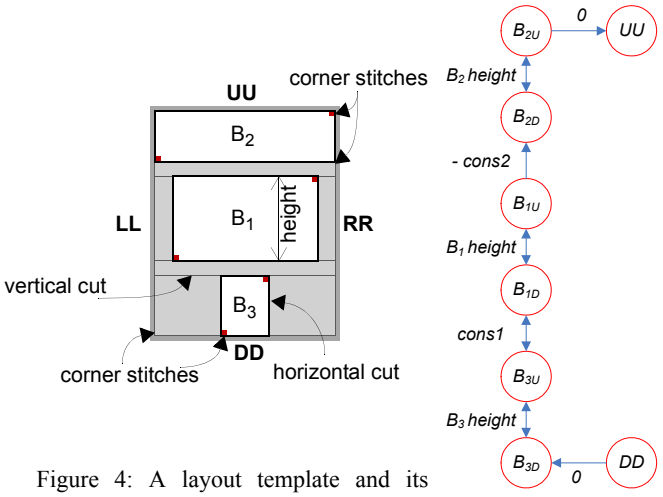

Figure 4: A layout template and its vertical constraint graph.

Through this representation, geometric constraints between the tiles can be easily established by assigning each arc e_{ij} a weight w_{ij}. These constraints arise as consequence of (1) process design rules, (2) connectivity (to ensure that two tiles remain electrically connected after layout retargeting/migration), and (3) analog specific issues (see Section 2.1).

Fig.4 also illustrates the vertical constraint graph of the template floorplan. cons1 and cons2 represent two hypothetical restrictions between position of tiles B_3, B_1, and B_1, B_2, respectively. Two-sided arrows mean "equal", while one-side

arrows mean "higher than" for positive constraints (e.g., $cons1$) or "lower than", for negative constraints (e.g., $-cons1$). Carefully and smartly devising a constraint graph is essential to, (1), avoid overlapping of the layout tiles (which may result in failing to comply with the process layout design rules), (2), reduce the complexity of parameterization process, most specially of routing, and, (3), improve the flexibility of the layout template to adapt a new circuit sizing while rendering compact layout solutions.

Once the vertical and horizontal constraint graphs have been worked out, the procedural generator of the layout template is built. Several resources have been reported to create procedural generators, most of them based on a versatile programming language. On the one hand, there are procedural layout approaches based on common-purpose languages such as C or C++. Examples are the high-level languages CAIRO [3] and MSL [26]. On the other hand, specific-purpose languages are limited to a particular circuit design environment such as BALLISTIC [25], written in the Mentor Graphics' LX language for the *GDT* environment, or SKILL™ [31], the programming language of the Cadence's *Design Framework II*™ (*DFWII*) environment.

Whatever the language used, the following set of basic geometric-database procedures can be used to implement the template properties:

(a) *Resizing* and *moving*: geometric procedures used to shrink/stretch a tile and to move groups of objects while stretching others, both essential in the placement phase.

(b) *Repetition*: a geometric procedure used to create arrays of objects in the horizontal, vertical or both directions.

(c) *Conditional inclusion*: a geometric procedure by which an object can be included or excluded from an instance of the layout template, upon fulfilment of certain pre-defined conditions.

(d) *Layer aliasing*: a database procedure used to make selectable the manufacturing material each layout polygon is to be made of. This procedure is vital to attain the technology independence property of layout templates.

(e) *Inheritance*: a database procedure that lets a leaf component inherit or use one or more parameter values from the parent block in which it is placed. This allows hierarchically generating nested parameterized layouts as well as maintaining complete control over all the template's parameters.

A final but very important question remains. Layout templates, at any hierarchical level, must be developed having in mind all issues explained in Section 2, especially when focusing analog layout reuse. As said, layout templates feature the ability to capture the required layout expertise to do so, but it is only through adequate constraint graph and floorplan devising that truly reusable analog layout templates can be carried out successfully.

6. IMPLEMENTATION

In the implementation presented here, SKILL™ language and the PCELL technology [32] from the Cadence's *DFWII* environment have been chosen for their built-in capabilities and their widespread acceptance within the design community. Layout templates can be created either by using a dedicated user interface where the geometric and database procedures are graphically applied to a collection of mask layers and other PCELLS, or by directly writing out the SKILL™ code of the template. The graphic method, however, may result rather involved for complex layout parameterization especially if technology migration is also a goal (a MOS transistor primitive parameterization requires more than 20 graphic operations). Writing SKILL™ code to bring about the same parameterization provides higher flexibility for creation of complex designs and an easier way to maintain and upgrade the layout template code.

The layout template SKILL™ file uses generic design rules as well as generic mask layer names. A numerical value of each design rule and a mask layer name should be supplied every time the template is instanced. Provided that the design rule *pairs* and the mask layer *pairs* (i.e., the generic design rule or mask layer name and the corresponding actual value at the targeted process), are both available, these can be automatically read off and each migration parameter correctly adapted every time the layout template is ported. In addition to the built-in, available SKILL™ functions, several new functionalities (e.g., to automatically locate and retrieve the size and position of the layout tiles in order to implement the parameterized placement and routing) have been created to accelerate the generation process of the template code [5].

Fig.5 show several layout instances of the layout template of an analog comparator whose schematic is depicted in Fig.5(a). The instances correspond to different technologies as well as to different values of the layout parameters (i.e., device sizes). The layout template in Fig.6(a) implements the fully differential opamp core in Fig.6(b). The shaded devices in the opamp schematic are the leaf components of the opamp layout template. This opamp layout template has been instanced for different technologies (instances Fig.6(c) and (d) in a 0.5-μm CMOS process, instances Fig.6(e) and (f) in a 0.35-μm CMOS process) and different values of the performance specifications. The implemented device sizes were obtained from

different layout-aware synthesis experiments, all of them also pursuing area minimization. For instance, an aspect ratio of 0.5 was aimed for the layout instance in Fig.6(e).

7. DISCUSSION

The generation cost of a layout template refers to the time required to go from floorplan design to layout coding and debugging, which depends on two factors: the size of the circuit and the parameterization scope. The latter factor accounts for the number of generic mask layers and design rules involved, as well as the number of leaf component and template variants considered, which is in direct relation to the number of retargeting parameters. The broader the scope, the harder placement and routing parameterization becomes. In the course of the analysis of parameterized layouts, it has been realized that achieving technological parameterization is hardest at the lower levels of the layout hierarchy, i.e., for parameterized device primitives. Thus, parameterizing a hierarchically higher layout template to deal with migration is much easier since most of the technological considerations have already been taken into account for the template components. Carefully routing mask layers and design rules, as well as parameterization of component spacing, are the only concerns at higher

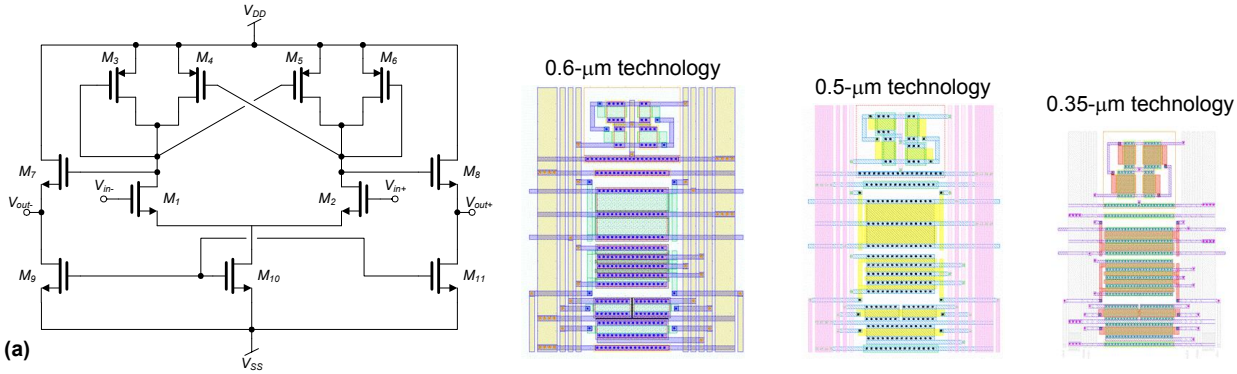

Figure 5: Layout template example: CMOS comparator.

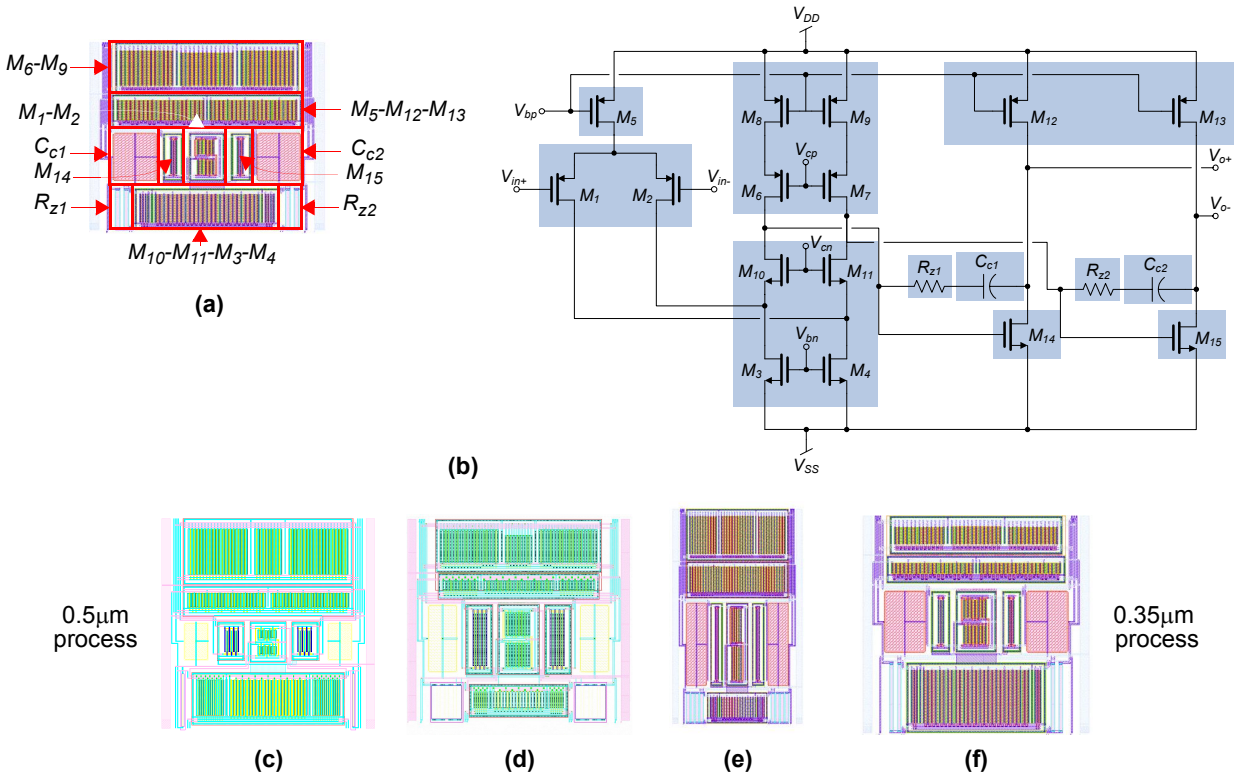

Figure 6: Layout template example: CMOS fully differential opamp core.

levels. Dealing with retargeting parameterization is, on the other hand, much complex at these levels, since placement and routing may often require transforming available designer's expertise into code, which is not always straightforward.
A reduction of the template generation time can be, however, successfully achieved by means of a large, well-provided library with parameterized device-level blocks (i.e., macro-cells) and primitives. Moreover, high-level SKILL™ functions as those mentioned in the preceding section help reducing the time spent on code generation. Other recent approaches have tackled the problem of automatic recognition of a template from a manually-created circuit layout [33]. Although manual layout generation may be faster than layout template generation, the real benefit of layout templates becomes manifest when considering reuse, geometrically-constrained, and parasitic-aware synthesis altogether. Modification of the manually generated layout to meet new reuse requirements implies, first, that possibly new device sizes have to be implemented, and, second, that placement and routing has to be either adapted or created from scratch. Depending on the number of modifications required, the total updating time may be similar to the time it took to create the initial layout manually. Layout templates provide the way to perform an extremely fast, completely automatic updating of the circuit layout for different device sizes or a different fabrication process. Furthermore, the layout template generation times are yet better than those coming from other layout synthesis approaches, such as optimization-based ones. Therefore, the more a circuit block is reused, the higher the benefits obtained from the layout template approach. On the other hand, the template's low flexibility (i.e., the capability of adapting new device sizes while preserving analog features such as device matching, minimization of layout-induced parasitics, and area optimization) is usually brought up as a serious drawback when compared to manual or optimization-based approaches. Yet, the low turnaround times and the parameterized nature of the layout templates allow us to palliate the flexibility drawback to a certain extent: layout template information (obtained from actual layout template instantiation or from its parameterized structure) can be incorporated into the circuit sizing process, such that device sizes, layout-induced parasitics, and layout instance area can all be simultaneously optimized.

8. CONCLUSIONS

In this paper, the benefits of using procedural layout generation based on templates on analog layout reuse and layout-aware synthesis has been analyzed. The ability of layout templates to embed analog layout expertise while revealing the knowledge required to perform both geometrically-constrained and parasitic-aware synthesis in comparatively smaller CPU times, clearly favor this approach when compared to other layout synthesis approaches. This paper also describes a methodology for suitable generation of analog layout templates and provides several examples.

ACKNOWLEDGMENTS

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. H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, *Surviving the SOC Revolution*. Boston: Kluwer Academic Publishers, 2003.
3. M. Dessouky, *Design for reuse of analog circuits. Case study: very low-voltage delta-sigma modulator*. Ph.D. Thesis, University of Paris VI, 2001.
4. F. 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.
5. R. Castro-López, *Reuse-based Methodologies and Tools in the Design of Analog and Mixed-Signal Integrated Circuits*. PhD Thesis, University of Seville, 2005.
6. G. Gielen and R. A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proc. IEEE*, vol. 88, no. 12, pp. 1825-1854, December 2000.
7. R. A. Rutenbar and J. M. Cohn, "Layout tools for analog ICs and mixed-signal SoCs: a survey," in *Computer-Aided Design of Analog Integrated Circuits and Systems*, R. A. Rutenbar, G. Gielen, and B. A. Antao, Eds. Piscataway: John Wiley & Sons Inc., 2002, pp. 365-372.

8. A. Hastings, *The art of analog layout*. New Jersey: Prentice Hall, 2001.
9. J. Bastos, *Matching characterization for precision analog design*. Ph.D. Thesis, Katholieke Universiteit Leuven, 1996.
10. Y. Tsividis, *Mixed analog-digital VLSI devices and technology*. New York: McGraw-Hill, 1996.
11. W. A. Lane and G. T. Wrixon, "The design of thin-film polysilicon resistors for analog IC applications," *IEEE Trans. Electron Devices*, vol. 36, no. 4, pp. 738-744, April 1989.
12. M. J. McNutt, S. LeMarquis, and J. L. Dunkley, "Systematic capacitance matching errors and corrective layout procedures," *IEEE J. Solid-State Circuits*, vol. 29, no. 5, pp. 611-616, May 1994.
13. B. R. Stanisic, N. K. Verghese, R. A. Rutenbar, L. R. Carley, and D. J. Allstot, "Addressing substrate coupling in mixed-mode ICs: simulation and power distribution synthesis," *IEEE J. Solid-State Circuits*, vol. 29, no. 3, pp. 226-238, March 1994.
14. 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.
15. 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.
16. J. A. Prieto, *GELSA: un colocador flexible para circuitos integrados analógicos*. Ph.D. Thesis, University of Seville, 2001.
17. K. Lampaert, G. Gielen, and W. Sansen, *Analog layout generation for performance and manufacturability*. Boston: Kluwer Academic Publishers, 1999.
18. 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.
19. 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.
20. E. Malavasi, E. Charbon, E. Felt, and A. Sangiovanni-Vincentelli, "Automation of IC layout with analog constraints," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 8, pp. 923-942, August 1996.
21. U. Choudhury and A. Sangiovanni-Vincentelli, "Constraint-based channel routing for analog and mixed analog/ digital circuits," in *Proc. of ACM/IEEE Int. Conf. on Computer-Aided Design*, 1990, pp. 198-201.
22. V. M. zu Bexten, C. Moraga, R. Klinke, W. Brockherde, and K.-G. Hess, "ALSYN: flexible rule-based layout synthesis for analog IC's," *IEEE J. Solid-State Circuits*, vol. 28, no. 3, pp. 261-268, March 1993.
23. 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.
24. J. Kuhn, "Analog module generators for silicon compilation," *VLSI Systems Design*, vol. 8, no. 5, pp. 74-80, CMP Publications, May 1987.
25. B. R. Owen, R. Duncan, S. Jantzi, C. Ouslis, S. Rezania, and K. Martin, "BALLISTIC: an analog layout language," in *Proc. of IEEE Custom Integrated Circuits Conf.*, 1995, pp. 41-44.
26. H. Sampath, A *Module Generation Environment for Mixed-Signal Circuits*. PhD Thesis, University of Cincinnati, 2003.
27. R. K. Henderson, L. Astier, A. El Khalifa, and M. G. R. Degrauwe, "A spreadsheet interface for analog design knowledge capture and re-use," in *Proc. of IEEE Custom Integrated Circuits Conf.*, 1993, pp. 13.3.1-13.3.4.
28. 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.
29. 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.
30. J. K. Ousterhout, "Corner stitching: a data-structure technique for VLSI layout tools," *IEEE Trans. on Computer-Aided Design*, vol. 3, pp. 87-100, December 1984.
31. *SKILL language reference. Product version 06.00*, Cadence Design Systems Inc., 2001.
32. *Virtuoso® parameterized cell reference. Product version 4.4.6*, Cadence Design Systems Inc., 2000.
33. N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C. J. R. Shi, "IPRAIL - Intellectual Property Reuse-based Analog IC Layout Automation," *Integration - The VLSI Journal*, vol. 36, pp. 237-262, November 2003.