Multilevel Symmetry-Constraint Generation for Retargeting Large Analog Layouts

Sambuddha Bhattacharya, Nuttorn Jangkrajarng, and
C.-J. Richard Shi, *Fellow, IEEE*

***Abstract*—The strong impact of layout intricacies on analog-circuit performance poses great challenges to analog layout automation. Recently, template-based methods have been shown to be effective in reuse-centric layout automation for CMOS analog blocks such as operational amplifiers. The layout-retargeting method first creates a template by extracting a set of constraints from an existing layout representation. From this template, new layouts are then generated corresponding to new technology processes and new device specifications. For large analog layouts, however, this method results in an unmanageable template due to a tremendous increase in the number of constraints, especially those emerging from layout symmetries. In this paper, we present a new method of multilevel symmetry-constraint generation by utilizing the inherent circuit structure and hierarchy information from the extracted netlist. The method has been implemented in a layout-retargeting system called Intellectual Property Reuse-based Analog IC Layout (IPRAIL) and demonstrated 18 times reduction in the number of symmetry constraints required for retargeting an analog-to-digital converter layout. This enables our retargeting engine to successfully handle the complexities associated with large analog layouts. While manual relayout is known to take weeks, our layout-retargeting tool generates the target layout in hours and achieves comparable electrical performance.**

***Index Terms*—Analog integrated circuit (IC), device matching, IC layout, layout automation.**

I. Introduction

AGGRESSIVE design cycles and rapid migrations towards newer technologies necessitate a reuse-based design philosophy in the semiconductor industry. Decades of innovations in the computer-aided-design (CAD) tools for digital circuits have resulted in standard flows and methodologies for the optimum reuse of existing digital designs. Unfortunately, the analog domain still awaits major innovations to facilitate effective design reuse. Indeed, with the recent focus on systems-on-chips that combine analog and digital functionalities on the same integrated circuit, the absence of CAD tools in the analog domain presents a serious bottleneck to the fast realization of mixed-signal designs.

Manuscript received December 6, 2004; revised March 10, 2005. This work was supported in part by the National Science Foundation (NSF) Information Technology Research (ITR) Program under Grant 9985507, in part by U.S. Defense Advanced Projects Agency NeoCad Program under Grant 66001-01-1-8920, and in part by a Grant from Conexant Systems. This paper was recommended by Associate Editor S. Sapatnekar.

S. Bhattacharya and N. Jangkrajarng were with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA (e-mail: sbb@u.washington.edu; njangkra@u.washington.edu).

C.-J. R. Shi is with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA (e-mail: cjshi@ee.washington.edu).

Digital Object Identifier 10.1109/TCAD.2005.855982

In analog design, tradeoffs between the major design goals like gain, bandwidth, stability, noise reduction, linearity, and power minimization demand considerable effort and time from the designers. Recently, significant progress has been made in the area of optimization tools [1], [2] that automatically synthesize analog circuits to meet desired performance specifications. However, the electrical behavior of high-performance analog designs is affected not only by the device sizes and biasing but also by the layout styles and intricacies.

Process and temperature variations introduce severe mismatches in transistors that are designed to behave identically [3]. Such mismatches drastically affect the performance of analog circuits [4], [5], leading to dc offsets and lower common-mode rejection. In differential structures, mismatches can also lead to finite even-order distortion. These effects can be alleviated by the symmetric layout of matched transistors. Thus, due to their strong impact on design performance, matching and symmetry, along with floorplanning, placement, and parasitic-driven wiring consideration, are of immense importance in analog layouts. Often, layout designers leverage their years of accumulated expertise to "squeeze-in" the desired analog-circuit performance by careful manual crafting of layouts.

Naturally, complex requirements on analog layouts pose a huge challenge to their design automation [4], [5]. Over the years, macro-cell-based constraint-driven automated placement and routing methodologies have been proposed for analog circuits [6]–[8]. Despite the generality of these layout-automation schemes, their output layouts are often inferior to the layouts manually crafted by expert designers in terms of electrical performance and quality; therefore, these layout-automation schemes are yet to attain acceptance in the industry.

Several attempts have been made to include designer's knowledge in the analog layout-automation process. These methods rely on templates constructed by designers through procedural languages [9]–[12], and require significant effort for template setup. Recently, layout retargeting by reusing designer expertise embedded in existing layouts has been proposed. The Intellectual Property Reuse-based Analog IC Layout (IPRAIL) tool suite, presented in [13] and [14], automatically creates a symbolic structural template from an existing layout by incorporating floorplan, symmetry, and device/wiring-alignment information. This structural template is then used to generate new layouts for new performance specifications and technology processes.

While these template-based layout-automation schemes successfully incorporate designer's expertise, they suffer from restricted topologies and can retarget designs only to compatible

processes. These limitations in topology can be largely alleviated by combining templates with device-layout generators [15]. More importantly, automatic tools such as IPRAIL that reuse the templates allow for a rapid evaluation of whether a given layout topology results in the desired circuit performance. Furthermore, different layouts corresponding to different specifications can be easily generated after the template is constructed once [16]. Thus, in addition to the retention of designer's expertise, the template-based methods offer several advantages that far outweigh their limitations.

Unfortunately though, all of these procedural/template-based schemes suffer a few critical shortcomings that prevent automation of large analog layouts. Firstly, the layout symmetry for matched transistors is manually imposed on the template through a graphical user interface and can get increasingly prohibitive as the circuit size increases. Secondly, and more importantly, the feasibility and efficiency of layout retargeting are strongly affected by the number of symmetry/matching constraints. This poses serious challenges to the retargeting of large layouts that contain numerous symmetry constraints. Therefore, the techniques in [9]–[14] are seldom used to handle layouts larger or more complex than operational amplifiers.

In this paper, we present techniques for efficient constraint generation that enable automatic layout retargeting of large analog IP blocks. Our new contributions are as follows.

1) Large analog circuits require symmetric layouts not only for matched transistors, but also for entire subcircuits that need to be identical to each other. Subcircuits may be split into halves and laid apart in one- or two-dimensionally symmetric styles so as to ensure similar effects of process and temperature gradients for all subcircuits that are identical by design. Utilizing extensive mappings between the extracted netlist and layout representations of the design, a new multilevel constraint-generation method is introduced. This multilevel templating scheme achieves a smaller template size, thereby allowing retargeting of large layouts.
2) We present an automatic method of identifying only relevant matched transistors from the circuit netlist and imposing corresponding layout-level symmetry constraints.
3) Large analog IP blocks usually contain on-chip resistors and capacitors. Such passive devices significantly affect circuit performance and need to be laid out carefully to minimize the parasitic effects. Furthermore, passive devices identical by design are also laid out symmetrically. Our layout-retargeting tool has the ability to automatically impose constraints to maintain these symmetry and spacing restrictions in passive devices.

Manual intervention during template creation restricts the usability of retargeting tools to smaller layouts. The tool presented in this paper achieves effective automation of template creation and subsequent layout generation, thereby allowing the retargeting of large analog layouts. Some preliminary results of this work were presented in [17] and [18].

The rest of the paper is organized as follows. Section II briefly describes the layout-reuse methodology with emphasis on IPRAIL and discusses various challenges in retargeting large analog layouts. Section III provides an overview of the proposed constraint-generation scheme. Sections IV–VI elaborate the various steps to establish mappings between the netlist and layout representations of the existing design. Section VII describes the actual constraint-generation process. Section VIII presents experimental results. Section IX concludes the paper.

II. Template-Based Layout-Reuse Methodology

Template-based layout automation attempts to extract the complex layout styles in existing high-quality analog layouts and generate a new layout targeted at a different set of functional specifications or a different technology. In this section, we provide an overview of IPRAIL [14], which incorporates template-based layout automation via layout reuse.

A. IPRAIL Tool Suite

A manually crafted analog layout along with source and target technology design rules is read into IPRAIL, which consists of a Layout Template Extractor and a Layout Generator. The Layout Template Extractor automatically creates a symbolic layout template that retains the input layout's topology, connectivity, and matching. The new device sizes for the target layout are obtained by manual circuit simulation or from automatic circuit-synthesis tools [1], [2]. By imposing these new device sizes pertaining to new specifications on the symbolic template, the Layout Generator constructs a target layout that maintains all the designer expertise embedded in the source layout.

1) Layout Template Extractor: The Layout Template Extractor identifies the active and passive devices, detects device matching and symmetry, and extracts device connectivity and net topology from the source (input) layout. Based on the extracted information and the technology-process design rules, it transforms the layout into a constraint-based resizable symbolic template representation. The symbolic layout template is an abstract representation of the extracted layout properties, namely device floorplan, connectivity, technology-process design rules, and analog layout intricacies.

The detailed flow of template extraction is shown in Fig. 1. First, the input layout is parsed and stored in the corner-stitching data structure [19]. The entire plane of each mask layer is represented explicitly in terms of solid and space rectangles called tiles. Each tile in a layer plane is connected to other tiles in the same plane by four stitches on its lower-left and upper-right corners.

Next, transistors and nets are extracted from the layout according to the algorithm proposed in [20]. The extractor detects the overlaps between polysilicon and diffusion tiles to identify all unit transistors, i.e., transistors with a single tile for the gate terminal. From the unit transistor's terminals, viz. gate, drain, and source, the nets are identified by a depth-first search [21] that traces the electrically connected tiles.

On-chip resistors are detected by searching through the tiles of the nets in the circuit. A single tile or a series of connected tiles of a net are classified as a resistor when the resistive value exceeds a user-defined threshold. Once a resistor is detected, its parent net is split into two. In IPRAIL, metal–insulator–metal

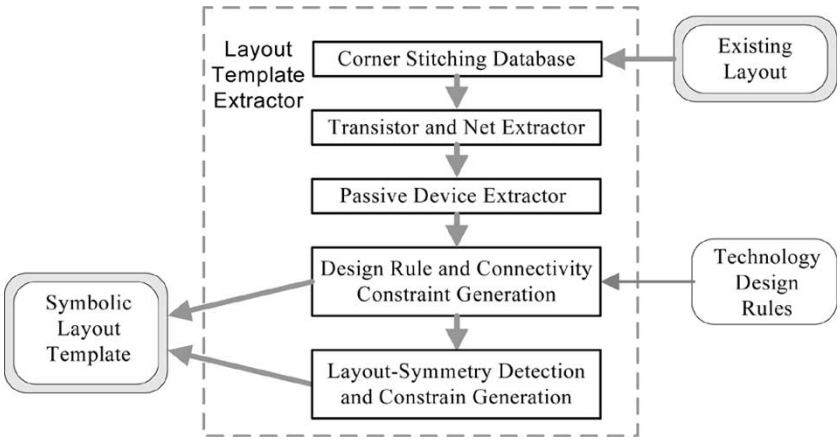

Fig. 1. Layout-template-extractor flow.

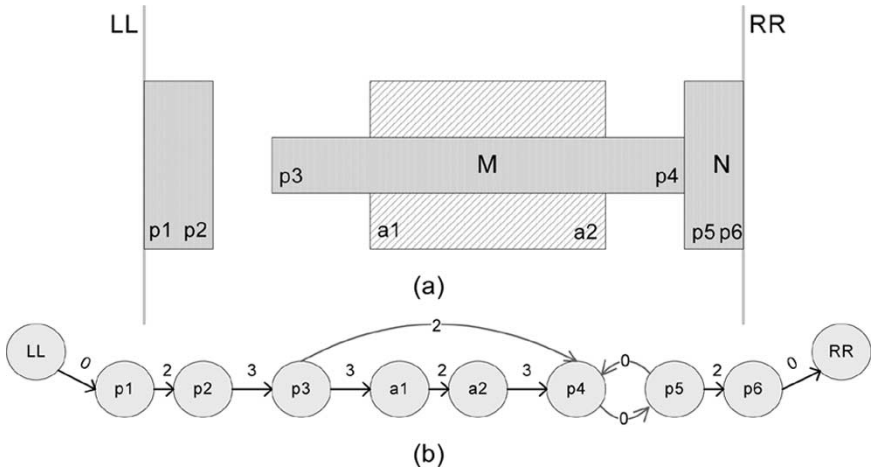

Fig. 2. Geometric constraints in graph form. (a) Layout example; (b) horizontal-constraint graph.

or polysilicon–polysilicon capacitors are defined as overlaps of two tiles in different layers that belong to different nets. Searching through the nets, the extractor detects capacitors when the capacitance due to the overlap exceeds a user-defined threshold.

Next, various layout properties such as connectivity and design rules between tiles are extracted and expressed as linear constraint equations [22] to sustain the layout integrity and correctness upon retargeting. The variables in the constraints correspond to the four edges of the tiles. Such constraints may be expressed in graph form, where each tile variable is represented by a node in the graph. A directed weighted arc connecting two such nodes represents a constraint where the weight of the arc represents the constant in the constraint inequality. Consider the simple layout of Fig. 2, the connectivity between rectangles M and N in the horizontal direction is retained by two constraint arcs of weight "0" between edges $p4$ and $p5$. The design-rule constraint is further decomposed into three types: minimum width—an arc from $p1$ to $p2$, minimum spacing—an arc from $p2$ to $p3$, and minimum extension—an arc from $a2$ to $p4$. Horizontal and vertical constraint graphs are constructed independently.

Matching between a pair of transistors is established by laying out the transistors symmetrically. Two transistor layouts are deemed symmetric if they are geometric mirror images of each other. As illustrated in the simplified example of Fig. 3, this implies equisized channel, drain, and source regions, identical orientation, and close proximity of the two transistors. The mirroring and location are then enforced by the following equations

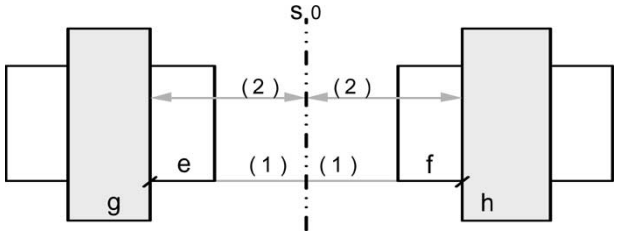

Fig. 3. Simplified layout of a symmetric pair of unit transistors. S_0 denotes the axis of symmetry.

$$(e_{\text{bottom}} - f_{\text{bottom}}) = 0 \tag{1}$$

$$(h_{\text{left}} - s_0) - (s_0 - g_{\text{right}}) = 0. \tag{2}$$

The layout-symmetry-detection algorithm, Direct Layout-Symmetry Detection (DLSD) [23], adopted initially in IPRAIL, relies on scanning the entire layout for symmetric transistors. All extracted unit transistors are stored in a queue sorted by their bottom edges. Devices with the same ordinate of bottom

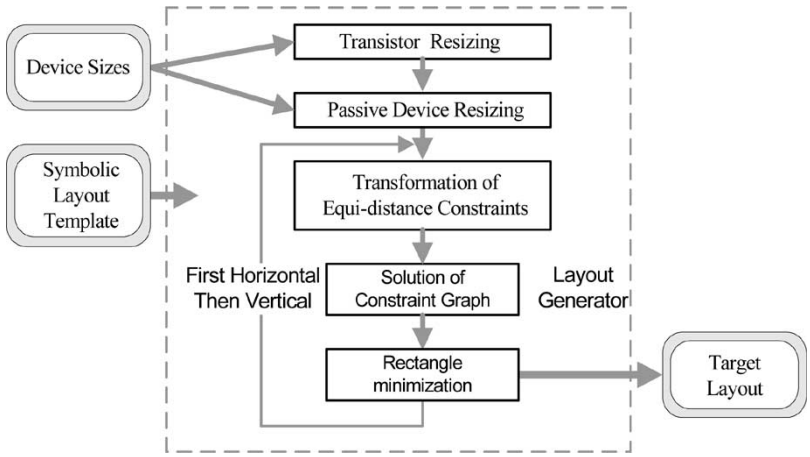

Fig. 4. Flow of the layout generator.

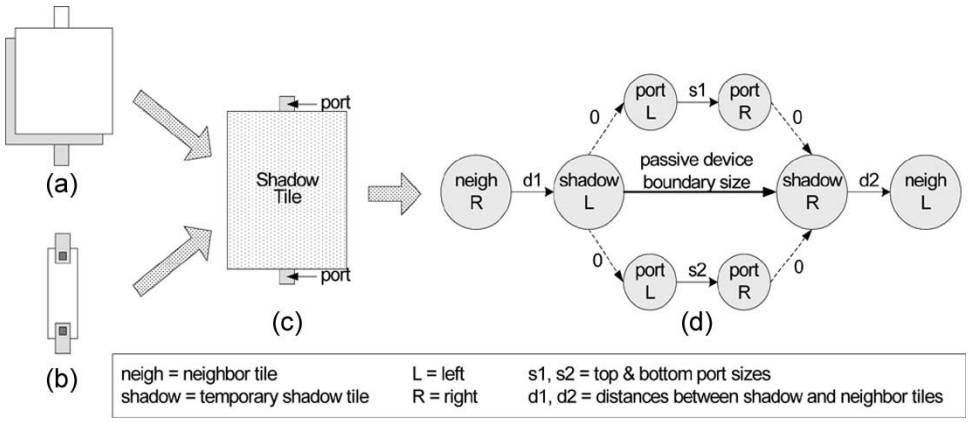

Fig. 5. Passive device retargeting with shadow tiles. (a) MIM or P–P capacitor; (b) on-chip unit resistor; (c) passive replaced by shadow tile; (d) simplified constraints for passive retargeting. Distances from neighbors are obtained according to (3).

edges are then pairwise compared for the existence of geometric mirror images. After the detection of all symmetric transistor pairs, all axes of symmetry with the same abscissa are merged into a single axis.

2) Layout Generator: Fig. 4 illustrates the various steps for the target-layout generation from the extracted symbolic template. First, the layout generator updates the template with the transistor and passive device sizes obtained from manual circuit-simulation or circuit-synthesis tools. As the updated template consists of the symmetry, connectivity, and design-rule constraints, the problem of target-layout generation from the symbolic template essentially is a modified symbolic compaction problem [22].

The exact transistor sizes for the target design are imposed on the template by two additional constraint arcs, for each transistor, with equal and opposite weights added in opposite directions. For passive devices, to prevent overlaps or close proximity to other devices upon retargeting, a shadow tile [16] is placed on top of the devices prior to the symbolic template generation, as shown in Fig. 5(c). A shadow tile is a temporary nonphysical-layer rectangle that is used to allocate a dedicated area for the passive device by applying spacing constraints to rectangles on every layer. The constraint for a shadow tile arises due to one of the following: coupling constraints, specialized design rules for passives as observed in certain technologies, or as input from the designer. Such constraints are of the general form

$$x_{\text{shadow}} - x_{\text{other}} \geq d. \tag{3}$$

Connectivity between a passive device and nets at its ends are maintained by constraints on port rectangles. Constraints are added between the shadow tile and the port rectangles in order to maintain connectivity upon retargeting. This is illustrated in Fig. 5(d). If the new device demands complete structural changes, it is obtained from a device library and placed inside the shadow tile. Otherwise, the new sizes are used to simply expand or compress the existing device.

The problem of target-layout generation is solved first horizontally and then vertically. In the horizontal direction, the problem can be expressed in the following form:

$$\min \quad (x_{\text{R}} - x_{\text{L}}) \tag{4.1}$$

$$\text{subject to} \quad x_i - x_j \geq \text{constant} \tag{4.2}$$

$$x_i - x_j = \text{constant} \tag{4.3}$$

$$x_i - x_k = x_k - x_j \tag{4.4}$$

where x_{L}, x_{R} represent the left and right boundaries of the target layout. The variables x_i, x_j, and x_k correspond to the

left or right edges of the tiles or symmetry axes of the layout. As has been illustrated in Fig. 2, the constraints in (4.2) and (4.3) can be expressed in graphical form where the right-hand-side constant represents the weight of the edges in the graph. However, as the right-hand-side constants are not known *a priori* for the equidistance constraints of (4.4), these cannot be expressed in graphical form.

The problem in (4) is a standard linear-programming (LP) [24] formulation. However, solving large problems with LP can be very expensive computationally. If the constraints in (4.4) are ignored, then the problem reduces to the standard layout-compaction problem and can be solved by the graph-based longest path algorithm [21]. In order to solve the modified compaction problem with graph-based methods, the constraints of (4.4) need to be transformed to a graph-imposable form. This is accomplished by a combination of LP and graph-based longest-path algorithm [25]. From the constraint graph corresponding to (4.2) and (4.3), a smaller set of constraints, called core constraints, are obtained such that the variables in the core constraints are the ones that appear in (4.4). This is accomplished by one shortest path run on the constraint graph for each variable in the equidistance constraint of (4.4). This small set of core constraints along with the constraints in (4.4) are then solved by LP. This yields the right-hand side constants for (4.4) in the form

$$x_i - x_k = x_k - x_j = \text{constant}. \tag{5}$$

These transformed constraints are then imposed back on the constraint graph corresponding to (4.2) and (4.3). The entire problem is then solved with the graph-based longest path algorithm. Finally, as the longest path algorithm results in some unwanted extension of rectangles, the rectangle-minimization algorithm [26] is applied to obtain the final target layout.

Here, it must be noted that the retargeting from a given technology to a completely different target technology may lead to infeasible constraints. Then, positive cycles are encountered during the execution of the longest path algorithm. Such positive cycles are detected and reported by IPRAIL.

B. Challenges in Retargeting Layouts of Large Designs

1) Symmetric Layout of Matched Transistors: The primary challenge in retargeting large analog layouts lies in the escalating number of constraints due to the increase in layout size and complexity. Industrial analog layouts typically consist of numerous multifinger transistors laid out symmetrically in one- or two-dimensional cross-coupled topologies, as shown in Figs. 6 and 7, respectively. Under the DLSD scheme [23], the layout of Fig. 6 has 21 axes of symmetry marked by the axes s_1 to s_{21} and 66 matched unit-transistor pairs. The layout in Fig. 7 has 12 axes of symmetry as indicated by the axes s_1 to s_{12} and 30 matched unit-transistor pairs.

As has been explained in Section II-A2, each symmetry axis introduces equidistance constraints that necessitate multiple longest-path-based transformations on the constraint graph [25]. Clearly, the presence of a large number of symmetry

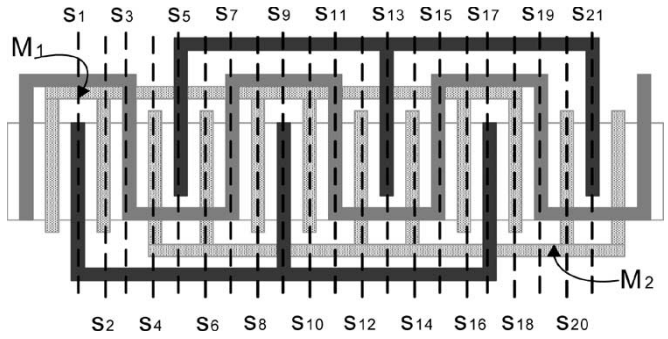

Fig. 6. One-dimensional cross-coupled symmetric multifinger-transistor pair (also called interdigitation or interleaving symmetry). The rectangles with dotted patterns represent the polysilicon layer.

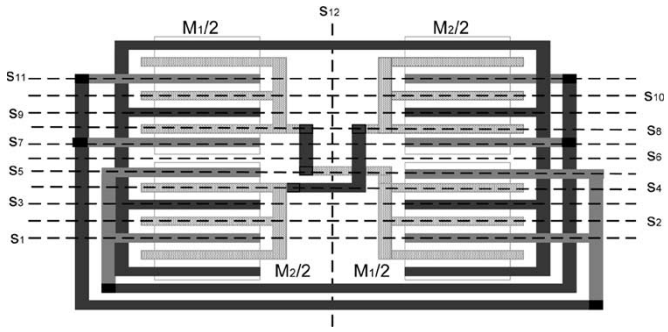

Fig. 7. Two-dimensional cross-coupled symmetric multifinger-transistor pair. The rectangles with dotted patterns represent the polysilicon layer.

constraints renders the process computationally expensive. Also, as we found during our retargeting experiments, too many redundant symmetry constraints may even render the problem unsolvable. This prohibitive increase in the size of the problem is experienced in retargeting large analog layouts.

Furthermore, layouts often have unrelated devices that may be symmetrically laid out by mere chance rather than by design. These unwanted symmetry constraints result in an increase in layout area. While the designer may explicitly identify the desired symmetry axes through a graphical user interface, it becomes unreasonable as the size and complexity of the layout increases.

2) Symmetric Layout of Matched Passive Devices: Large layouts often have multiple passive devices that are designed to be identical to each other. Consider the resistor-chain layout of Fig. 8. Three resistors are laid out in an interdigitated fashion with one-dimensional common-centroid symmetry [4]. This ensures identical resistances of the resistors A, B, and C, regardless of process and temperature gradients, or in other words, matching between the resistors.

3) Layout of Matched Blocks: In addition to matched devices, large analog circuits usually contain entire blocks that are identical by design. Process and thermal gradients across the entire layout introduce differences between blocks that are meant to behave identically. Consider a 2-bit comparator circuit that is composed of four unit comparators. In order to alleviate the differences in the performances of the unit comparators and the consequent nonlinearities in electrical behavior, they need to be laid out in a common-centroid fashion. This is illustrated

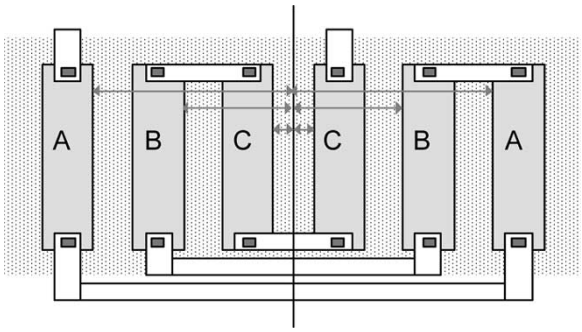

Fig. 8. One-dimensional symmetric layout of on-chip resistors.

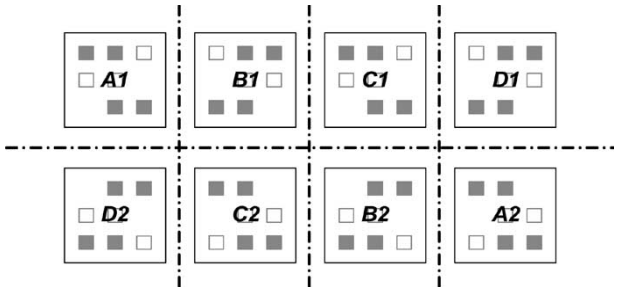

Fig. 9. Four comparator subcircuits (A, B, C, and D) laid out in a split-symmetric common-centroid layout.

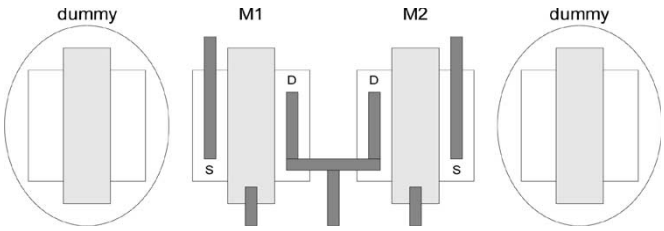

Fig. 10. Dummy transistors to improve matching.

in Fig. 9, where four unit comparators A, B, C, and D are laid out symmetrically in two dimensions. The unit comparator denoted by A is split and laid out across the ends in two parts A_1 and A_2. In general, identical blocks may be laid out with one- or two-dimensionally symmetric layouts and may also be translated with respect to each other. A naive direct-constraint generation for flipped or translated devices and nets leads to a tremendous increase in the template size for large circuits.

4) Dummy Devices: Analog circuits with strict matching requirements employ dummy transistors. Consider the layout in Fig. 10. Here, the two transistors on the sides are either not electrically connected to any other device or connected to nonsignal nets such as ground or power supply, thus are called dummy transistors. Dummy transistors are generally laid out symmetrically to the corresponding actual transistors. This ensures that the drain and the source regions of the actual transistors have an identical environment and therefore superior matching. Dummy transistors need to be identified and maintained upon retargeting.

5) Guard Rings: In lightly doped substrates, guard rings are commonly used in analog layouts to isolate sensitive circuits. Guard rings in the original layout need to be retained upon retargeting. As illustrated in Fig. 11, this introduces additional

Fig. 11. Guard ring around a sensitive analog circuit. Constraints on the guard ring are: 1) minimum distance from the boundary of the sensitive analog section to the edge of the guard ring: $x_2 - x_1 > d$; and 2) width of the guard ring: $x_3 - x_2 > w$.

constraints such as the width of the guard ring $(x_3 - x_2 > w)$ and distance of the guard ring from the boundary of the sensitive analog layout $(x_2 - x_1 > d)$. These constraints need to be extracted and imposed for retargeting.

III. Multilevel Mapping and Constraint Generation

In this section, we present a new method of multilevel mapping and constraint generation aimed at reducing the number of constraints necessary for retargeting large analog layouts. It is based on two key techniques. First, at the device level, a method for automatic detection of designer-intended symmetries in the source layout is developed. This avoids generation of redundant and/or unwanted constraints. Second, for identical blocks of devices (either flipped or translated), an extensive partitioning and mapping between the netlist and layout representations of the design are incorporated. Based on this, a reduced set of constraints that suffices for ensuring flips/translations of identical blocks upon retargeting is generated.

The layout–netlist mapping and constraint-generation flow is illustrated in Fig. 12. The process starts with the netlist extracted from the layout description. This consists of nets, passive devices, and unit transistors. By detecting the layout patterns and connectivity of the unit transistors, multifinger transistors are identified and a compact netlist is obtained.

An analog design environment usually consists of a library of schematics of analog cells that are sized during circuit design. These cells are used as subcircuits in large analog circuits. Examples of such commonly used subcircuits are differential stages, current mirrors, comparators, etc. In the netlist subcircuit-extraction step, two goals are accomplished. First, all instances of the library subcircuits in the compact netlist are identified and a list of designer-intended matched transistors in each subcircuit instance is recognized. This list of matched transistor pairs in the netlist is used to extract the transistors' layout-symmetry constraints. Second, based on identified subcircuits, a partitioned netlist is created.

From the partitioned netlist obtained at the end of subcircuit extraction, clustering of the physical layout tiles is executed next. This results in a partitioned layout representation. The partitions of the layout are then further analyzed to generate a list of identical layout clusters.

The netlist- and layout-partitioning process establishes mapping at different levels between the layout and the netlist. Actual constraint generation at the layout level is triggered from the list of matched transistors within each layout cluster and the

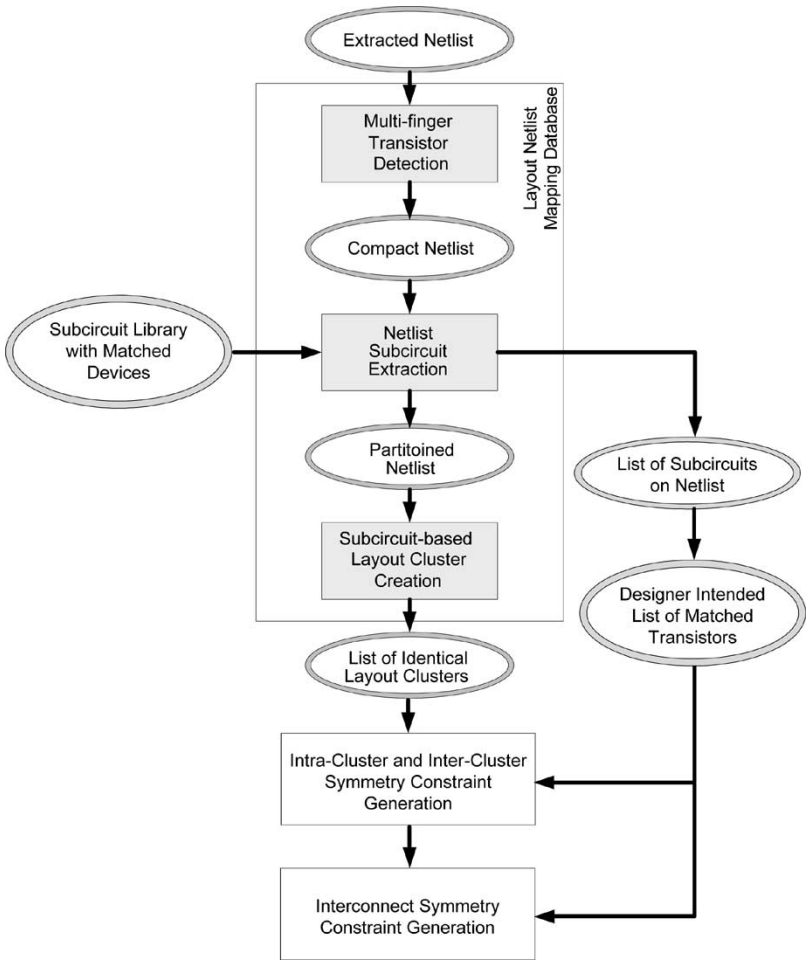

Fig. 12. Overview of the multilevel-mapping and constraint-generation flow.

lists of identical layout clusters. For large analog circuits, such mapping and partitioning is essential to reduce the size of the template to manageable levels.

Table I presents the multilevel constraint-generation algorithm. First, the procedure ExtractNetsTransistors extracts the netlist from the layout. Next, the DetectMultiFingerTransistors procedure groups the physically contiguous unit-transistor sets in the layout into multifinger transistors. The routine ExtractNetlistSubcircuits inside the loop identifies all instances of the library subcircuits. This identifies all the matched transistors that are meant to be symmetric in the layout. Next, the CreateLayoutClusters procedure accomplishes layout partitioning based on how the extracted subcircuit instances are laid out. The routine GenerateIntraClusterSymmetry-Constraints then generates the symmetry constraints within each layout cluster based on the list of matched transistors obtained previously. The constraints between different layout clusters are generated by GenerateInterClusterSymmetry-Constraints. Finally, the constraints for interconnect symmetry are generated by the procedure GenerateInterconnect-SymmetryConstraints.

IV. Multifinger-Transistor Detection

The netlist obtained after extraction comprises of a set of unit transistors (T^S) and a set of nets (N^S). The unit transistors in the layout are grouped into a set of contiguous unit transistors based on their physical adjacency and identical terminal connectivity. One or more sets of contiguous unit transistors with identical terminal connectivity are then grouped as a multifinger transistor.

In a multifinger-transistor-layout structure, a set of contiguous unit transistors (C) is defined as a collection of unit transistors (T) that posses the same width and length, share the same active rectangle, are physically adjacent, and connect to the same gate (G_T), source (S_T), and drain (D_T) nets. A multifinger transistor (M) is defined as a set of one or many set(s) of contiguous unit transistors that connect to the same gate, source, and drain nets. For example, the common-centroid layout of Fig. 13 has two multifinger transistors, each with two sets of three contiguous unit transistors. And the one-dimensional cross-coupled symmetric transistor pair of Fig. 6 contains two multifinger transistors, each with three sets of two contiguous unit transistors.

The DetectMultiFingerTransistors procedure in Table II presents the algorithm for the detection of multifinger transistors from the extracted netlist. Each multifinger transistor is stored in a hash table with the hash key formed by the drain, gate, and source nodes. For each unit transistor T connected to a net N, a new multifinger transistor M is created if it does not already exist in the hash table. This is accomplished by a

TABLE I
MAPPING AND CONSTRAINT-GENERATION ALGORITHM

```
GenerateMultiLevelConstraints
begin
  ExtractNetsTransistors
  DetectMultiFingerTransistors
  for each  s ∈ L            // s = subcircuit, L = Library
    ExtractNetlistSubcircuits
  end for
  CreateLayoutClusters
  GenerateIntraClusterSymmetryConstraints
  GenerateInterClusterSymmetryConstraints
  GenerateInterconnectSymmetryConstraints
end
```

TABLE II
ALGORITHM FOR MULTIFINGER-TRANSISTOR DETECTION

```
DetectMultiFingerTransistors
begin
  for each N ∈ N^S        // N^S is the set of nets
    // G_T, S_T, D_T are gate,source,drain nets of transistor T
    for each T ∈ T^S | N ∈ {G_T, S_T, D_T}
      M = CheckCreateMFT(G_T, S_T, D_T)
      X = CheckCreateContiguous(C^S, T) // M = C^S = {C}
      InsertSorted(T, X)  // doubly sorted w.r.t. x, y co-ordinates
    end for
  end for
end
```

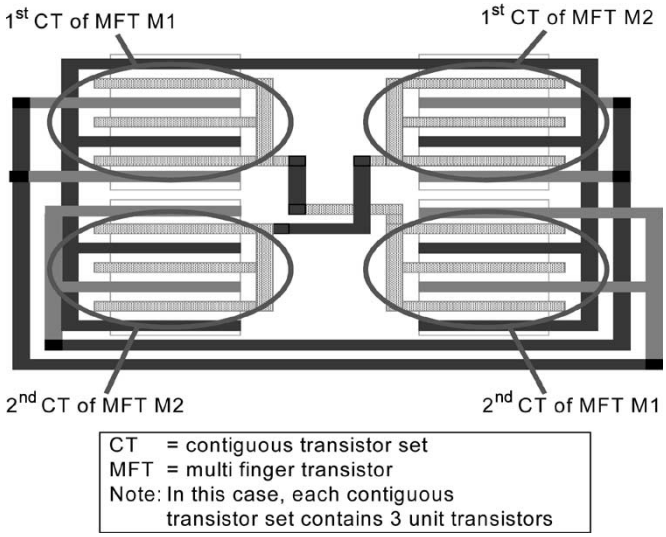

Fig. 13. Common-centroid layout that has two multifinger transistors, each with two sets of three contiguous unit-transistors.

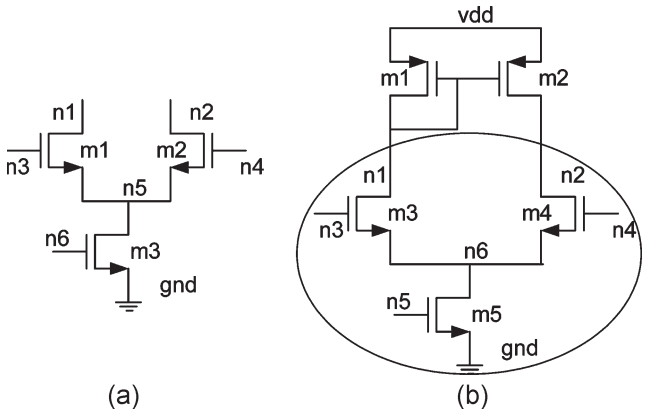

Fig. 14. (a) Differential-pair subcircuit (Ψ); (b) a differential-stage circuit (Γ) that comprises an instance of a differential pair as indicated by the encircled part of the schematic.

call to the routine CheckCreateMFT. The routine CheckCreateContiguous then checks if the unit transistor T is aligned with one of the sets of contiguous unit transistors in M. If T is not physically contiguous with any $C \in M$, a new set of contiguous unit transistors is created. In either case, the routine InsertSorted inserts T into a list of unit transistors of the corresponding set of contiguous unit transistors.

V. SUBCIRCUIT EXTRACTION

The multifinger-transistor detection results in a compact netlist. The subcircuit extraction process identifies instances of library subcircuits in this compact netlist. As mentioned in Section III, this partitioning of the netlist is essential for the following: 1) the automatic identification of matched-transistor pairs in the netlist and 2) the subsequent layout clustering.

The subcircuit extraction involves detection of the instances of a subcircuit Ψ in the circuit netlist Γ as illustrated in Fig. 14. This is accomplished by a subgraph isomorphism algorithm [27]. First, both the subcircuit Ψ and the circuit Γ are expressed as undirected bipartite graphs. Fig. 15 shows the bipartite graphs corresponding to the differential pair and the differential stage circuits of Fig. 14. A bipartite graph of a netlist comprises of two types of vertices—vertices corresponding to nets and vertices corresponding to devices. Each arc in a bipartite graph connects a net vertex with a device vertex. In the bipartite graphs of Fig. 15, the circular vertices correspond to the nets and the rectangular vertices correspond to the transistors.

The subgraph-isomorphism algorithm identifies instances of the subcircuit Ψ in a circuit Γ by iterative labeling-based partitioning. In this scheme, the vertices of the bipartite graphs corresponding to the subcircuit Ψ and the circuit Γ are iteratively labeled. Initially, each device vertex is labeled with the device type, e.g., N or P type for MOSFETs. Each net is initially labeled with the number of devices connected to it. Thus, the net n5 in Fig. 15(a) has an initial label 3. During iterative labeling, the device and net vertices are relabeled according to the following equations

$$m^{\text{label}} = m^{\text{label}} + \text{wt}(S) * S_m^{\text{label}} + \text{wt}(D) * D_m^{\text{label}} + \text{wt}(G) * G_m^{\text{label}} \quad (6)$$

$$N^{\text{label}} = N^{\text{label}} + \text{wt}(S) * \sum_i m_i^{\text{label}} + \text{wt}(D) * \sum_j m_j^{\text{label}} + \text{wt}(G) * \sum_k m_k^{label}. \quad (7)$$

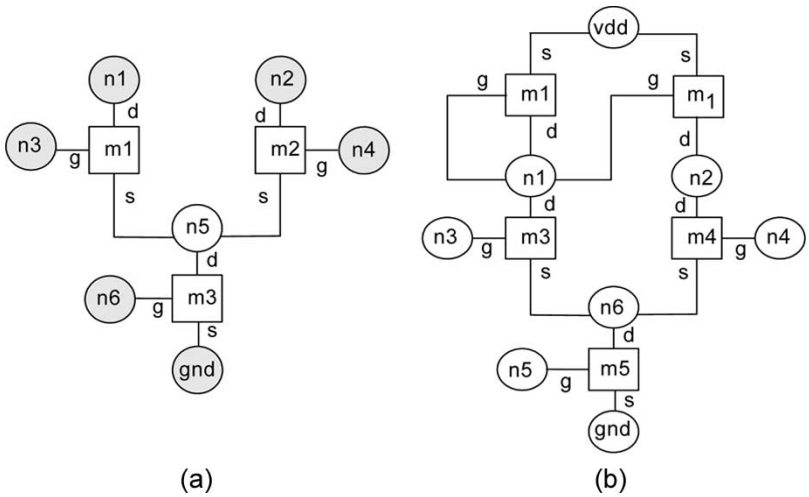

Fig. 15. Bipartite graphs of (a) a differential-pair subcircuit; (b) a differential-stage circuit. The circular vertices represent nets and the rectangular vertices represent transistors.

Here, S_m^{label}, D_m^{label}, and G_m^{label} represent the source, drain, and gate terminals of the transistor m. The source, drain, and gate terminals are assigned a weight as indicated by $\text{wt}(S)$, $\text{wt}(D)$, and $\text{wt}(G)$. As illustrated in (7), the new label of a net is calculated based on the labels of the transistors connected to the net through its source, drain, and gate terminals. Thus, the label of net n5 in Fig. 15(a) after the second iteration is given as $\text{n5}^{\text{label}} = 3 + \text{wt}(S) * 2N + \text{wt}(D) * N$.

The algorithm proceeds in two phases. Phase I identifies a key vertex in the bipartite graph of the subcircuit Ψ and a set of candidate vertices with the same label in the bipartite graph of the circuit Γ. This is accomplished by the iterative labeling of the bipartite graphs of the subcircuit Ψ and the circuit Γ. Prior to labeling, the vertices corresponding to the external nets of the subcircuit Ψ are marked as corrupt vertices. In the bipartite graph of the differential pair subcircuit of Fig. 15(a), the corrupt vertices are indicated by shaded circles. During each subsequent labeling step, all neighboring vertices of a corrupt vertex are marked as corrupt. Thus, for the bipartite graph of the differential pair in Fig. 15(a), the second labeling iteration marks the vertices m1, m2, and m3 as corrupt vertices. The search for the key vertex terminates when either all net or all device vertices in the bipartite graph of the subcircuit Ψ are marked corrupt. Clearly, vertices with the same labels in the bipartite graph of the circuit Γ and the bipartite graph of subcircuit Ψ indicate possible matches. All vertices with the smallest label in the bipartite graph of the circuit Γ are considered candidate vertices if there exists a vertex with the same label in the bipartite graph of subcircuit Ψ. This vertex in the bipartite graph of subcircuit Ψ is called the key vertex. For the bipartite graphs of Fig. 15, the vertices corresponding to net n5 in subcircuit Ψ and the vertex corresponding to net n6 in circuit Γ are the key and the candidate vertices, respectively.

Phase II of the isomorphism algorithm starts from the key vertex in the subcircuit Ψ and one candidate vertex in the circuit Γ. The algorithm then iteratively relabels the neighboring vertices of the key and candidate vertices to identify a match. During each iteration, vertices in the bipartite graph of the circuit Γ that have the same label as the vertices in the bipartite graph of the subcircuit Ψ are marked as potential matches. When all vertices in Ψ have corresponding vertices with identical labels in Γ, an instance of the subcircuit is obtained.

The matched transistor pairs in the library subcircuits are assumed to be known *a priori*. The labeling of the vertices in the bipartite graphs during subcircuit extraction establishes a mapping between the individual devices of the subcircuit and its instance in the circuit netlist. This is utilized for identifying the matched transistor pairs in the circuit netlist. For the example in Fig. 14(a), the devices m1 and m2 in the differential pair are matched transistors. The corresponding devices m3 and m4 in the differential stage circuit of Fig. 14(b) are therefore marked as matched transistors. Thus, the subcircuit extraction step also identifies all designer-intended matched transistor pairs in the circuit.

At times, a given transistor in the circuit netlist may be a part of different subcircuits. An example of this is current mirror transistors that reflect current from a bias current source to various major subcircuits, e.g., a comparator. Here, the mirror transistor is a part of both the current mirror and comparator. In the subgraph-isomorphism algorithm, we do not replace the set of transistors that comprise a subcircuit from the circuit netlist upon the detection of an isomorphism. Rather, we tag a transistor with information about which subcircuit it belongs to. Allowing multiple subcircuit tags to a transistor ensures correct detection of matchings in such cases. In the case of a transistor with multiple subcircuit tags, the matching constraints arise due to one or more of the subcircuits it is affiliated to.

VI. Layout Partitioning

A. Layout Clustering

The subcircuit-extraction step creates several netlist partitions, each corresponding to a subcircuit instance. In a layout, devices and nets that belong to the same netlist partition are clustered together based on their spatial proximity. The algorithm for layout clustering is shown in Table III.

The algorithm starts from any seed device m_{seed} which belongs to a netlist partition Ψ. First, a layout cluster L_{c} is

TABLE III
LAYOUT-CLUSTERING ALGORITHM

```
CreateLayoutClusters(m_seed)
begin
  if (m_seed already assigned to a cluster) then
    return
  endif
  L_C = CreateCluster(m_seed)
  Q_prox = ProximalDevices(m_seed)          // scan-line based routine
  for each m ∈ Q_prox
    if (m.partition == m_seed.partition) then
      AddCluster(L_C , m)                   // add m to L_C
    else
      CreateLayoutClusters(m)               // recursive call with seed m
    endif
  end for
end
```

created containing only m_{seed}. All devices that are proximal to m_{seed} in the layout are collected in a queue Q_{prox}. This is accomplished by a scan-line [22]-based procedure named ProximalDevices. From a given device, four scan lines look for other devices in close proximity to its left, right, top, and bottom edges. If a device m in Q_{prox}, is in the netlist partition Ψ, then m is added to the same layout cluster L_{c} that m_{seed} belongs to. If m does not belong to Ψ, the algorithm recursively calls itself to start a new layout cluster. The algorithm terminates when all devices in the layout are grouped into layout clusters.

For the example in Fig. 9, the algorithm identifies two clusters A_1 and A_2 for the layout of the comparator circuit A. Each such layout cluster contains a subset of the devices in the corresponding netlist partition. In practice, even a single device may be split across different layout clusters. For example, a multifinger transistor may be composed of two or more contiguous transistor sets that are laid out far apart to account for different process gradients. Thus, a layout cluster may consist of only some contiguous transistor sets of a given multifinger transistor.

B. Identical-Layout-Cluster Detection

The previous steps mark each contiguous transistor set of a multifinger transistor in the layout with its netlist partition (i.e., which subcircuit instance it belongs to) and layout-cluster information. At this stage, we proceed to detect identical layout clusters. Two layout clusters are identical only when their devices and nets have one-to-one correspondence in terms of sizes, connectivity, and relative positions. The algorithm for detecting identical layout clusters is shown in Table IV.

Consider two layout clusters L_A and L_B located on the same abscissa. First, all tiles in the clusters are collected in heaps [21] H_A and H_B, respectively. The tiles are then sorted in increasing order with respect to the coordinates of the leftmost corner. The two heaps are pairwise compared. If they are identical, L_A and L_B are translate-matched clusters. In case they are not, another heap H_{BR} is created for cluster L_B, where the tiles are sorted in decreasing order of their rightmost corner. If the two heaps

TABLE IV
ALGORITHM FOR IDENTIFYING IDENTICAL LAYOUT CLUSTERS

```
DetectIdenticalLayoutClusters(L_A , L_B)
begin
  H_A = CollectTiles(L_A)          // heap of tiles, sorted wrt x,y
  H_B = CollectTiles(L_B)
  H_BR = Reverse(H_B)
  if (Translate(H_A , H_B) == TRUE) then
    return TranslateMatched
  else if ( Mirror(H_A , H_BR) == TRUE) then
    return FlippedMatched
  else
    return NotMatched
  endif
end
```

H_A and H_{BR} are identical upon pairwise comparison, then they represent flip-matched clusters. For example, in the simple layout of Fig. 9, the clusters A_1 and C_1 are translate matched, while clusters B_1 and C_1 are flip matched.

VII. MULTILEVEL-CONSTRAINT GENERATION

Apart from the connectivity and the design-rule constraints described in Section II, additional constraints are needed to maintain symmetry between multifinger transistors, nets, different layout clusters, and passive devices. This section describes a multilevel-constraint-generation method that reduces the number of such constraints necessary for retargeting large analog layouts.

A. Intracluster-Symmetry Constraints

Recall that the subcircuit-extraction step partitions the netlist according to subcircuit instances and also identifies the matched transistor pairs in each partition. Also recall that each netlist partition has one or more layout clusters associated with it. The intracluster-symmetry-constraint-generation scheme checks for the symmetric layout of matched transistor pairs within these layout clusters. By triggering layout-symmetry detection from circuit-level transistor-matching information, it avoids detecting unintended symmetries that may reside in the layout. Upon detection of layout symmetry in the input layout, it generates the symmetry-dictated layout constraints similar to (1) and (2) for retargeting.

The algorithm for intracluster-layout-symmetry detection is shown in Table V. For each transistor pair intended to be matched, the DetectTopology routine identifies the pair's layout topology by traversing through the list of their contiguous transistor sets. Based on two-dimensional common-centroid, one-dimensional cross-coupled (also known as interdigitation or interleaving), or simple symmetric-pair topologies, the unit transistors are inserted into two or four sorted lists. Thus, for the common-centroid topology of Fig. 7, the six unit transistors in the top and bottom halves of the transistors M_1 and M_2, respectively, are collected into a list L_{L}. The bottom and top halves of M_1 and M_2 are collected into another list L_{R}. The unit transistors in L_{L} and L_{R} are then pairwise compared in the

TABLE V
INTRACLUSTER-SYMMETRY-DETECTION ALGORITHM

```
DetectIntraClusterSymmetry
begin
    // ListSym = { (M_i , M_j) | M_i and M_j are intended matched pair }
    for each ( M_i , M_j ) ∈ ListSym
        topology = DetectTopology ( M_i , M_j )
        if ( topology == common_centroid ) then
            L_L = InsertToList ( M_i , M_j , left )
            L_R = InsertToList ( M_i , M_j , right )
            L_B = InsertToList ( M_i , M_j , bottom )
            L_T = InsertToList ( M_i , M_j , top )
            CheckSymmetry ( L_L , L_R )
            CheckSymmetry ( L_B , L_T )
        else if ( topology == horizontal_interleaving ) then
            L_L = InsertToList ( M_i , M_j , left )
            L_R = InsertToList ( M_i , M_j , right )
            CheckSymmetry ( L_L , L_R )
        else if ( topology == vertical_interleaving ) then
            L_B = InsertToList ( M_i , M_j , bottom )
            L_T = InsertToList ( M_i , M_j , top )
            CheckSymmetry ( L_B , L_T )
        else          // simple transistor layout
            CheckSymmetry ( M_i , M_j )
        end if
    end for
end
```

CheckSymmetry routine to detect the vertical axis of symmetry s_{12} and generate the corresponding constraints. For the horizontal symmetry axis s_6, the bottom halves of both M_1 and M_2 are collected into a list L_{B}, and the top halves are collected into a list L_{T} and pairwise compared. For the one-dimensional cross-coupled (interdigitation or interleaving) symmetric layout of Fig. 6, six unit transistors are inserted into each list L_{L} and L_{R} and a single axis of symmetry s_{11} is detected. Prior coordinate-based double sorting of the unit transistors in each multifinger transistor ensures that pairwise comparison can detect all axes of symmetry. If the matched transistors are split across different layout clusters, the symmetry constraints are generated inside the corresponding layout clusters.

Symmetric layouts for dummy transistors cannot be handled by simply identifying matched transistors in the circuit netlist. This is because dummy transistors are not electrically connected to any other device through signal nets or are connected to nonsignal nets such as ground or power supply. Usually, a dummy transistor is laid out in close proximity to an actual transistor. Thus, there exist direct constraint arcs due to design rules between vertices corresponding to the tile edges of the dummy and actual transistors in the layout constraint graph. As a result, a dummy transistor is identified by a local search around the vertices corresponding to the tile edges of the actual transistors in the constraint graph. Once the dummy transistor and the associated actual transistor are identified from the layout, symmetry is detected by pairwise comparison of the corresponding layout tiles.

B. Intercluster-Symmetry Constraints

Recall from Section II-B that identical layout clusters may be laid out either mirrorwise flipped or translated with respect to each other. Fig. 9 illustrates an example of two such translated and flipped identical clusters. This section describes the generation of constraints between layout clusters for maintaining translations or flips of identical clusters upon retargeting. While this can be accomplished by generating constraints between every pair of devices belonging to two identical layout clusters, it leads to an exceedingly large number of constraints. The method described here generates a reduced number of constraints that would suffice for retargeting.

Note that the intracluster-constraint-generation process groups devices inside a layout cluster into symmetric layout pairs. It then extracts the constraints between the devices in each such symmetric group within the cluster. The intercluster-constraint-generation process establishes two key sets of constraints. First, it imposes constraints between identical groups of devices located in two different layout clusters. Second, it enforces additional constraints between groups of devices inside a cluster.

Consider the examples in Fig. 16 that show the translation and flip of two identical layout clusters named cluster1 and cluster2. In the ith cluster, the intracluster-constraint-generation process identifies that devices B_i and C_i represent a symmetric-layout pair. Thus, each cluster has two groups, one group comprises the device A_i and the other group comprises the symmetric pair B_i and C_i.

The first step in an intercluster-constraint generation involves identifying a representative device in each group within each cluster, referred to as group-representative devices. For cluster1, let devices A_1 and B_1 be the group-representative devices. One group-representative device in each cluster is considered as a cluster-representative device. In the example in Fig. 16, the devices A_1 and A_2 are the cluster-representative devices in cluster1 and cluster2.

For the translated clusters along a y-axis in Fig. 16(a), the following set of equations relates the group-representative devices B_1 and B_2 with the cluster-representative devices A_1 and A_2

$$x_1 - x_3 = x_2 - x_4 \tag{8}$$

$$y_1 - y_3 = y_2 - y_4 \tag{9}$$

where (x_1, y_1) and (x_2, y_2) are the lower left coordinates of the devices A_1 and A_2, and (x_3, y_3) and (x_4, y_4) are the lower left coordinates of the devices B_1 and B_2, respectively. The two cluster-reference devices A_1 and A_2 are related by the following equations

$$y_1 - y_2 \geq d \tag{10}$$

$$x_1 = x_2. \tag{11}$$

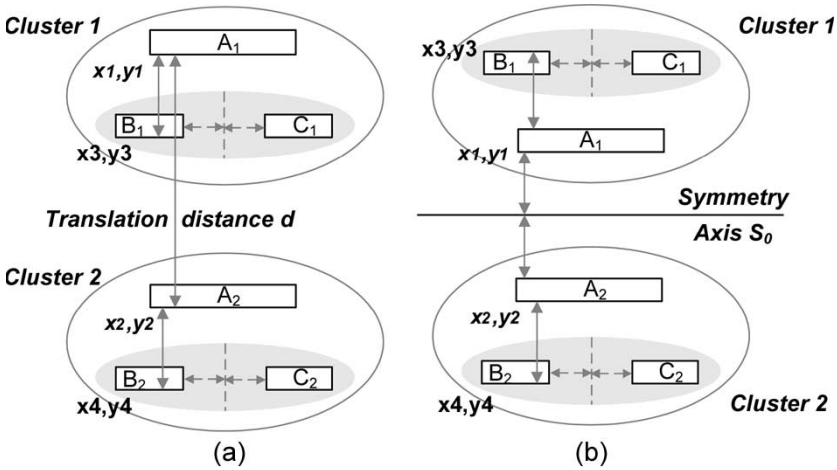

Fig. 16. (a) Translated matched clusters. (b) Flipped matched clusters. Symmetric devices B_i and C_i within each cluster are grouped together as indicated by the shaded background. Intracluster symmetry is indicated with dotted lines.

Here, d is the distance between two cluster-reference devices and is determined by coupling or design-rule constraints.

Similarly, for the flipped clusters along a y axis in Fig. 16(b), the group-reference devices in the two clusters are related with equations similar to (8) and (9). The two cluster-reference devices are related by the following equation

$$y_1 - s_0 = s_0 - y_2 \tag{12}$$

where s_0 is the symmetry axis between cluster1 and cluster2. Equations (10)–(12) are adjusted accordingly in case of translation or flip along the x axis.

Based on the intracluster and intercluster symmetries, if a layout consists of c clusters with n devices in each cluster, the number of symmetry axis is reduced from the worst case of $nc(nc-1)/2$ to $(nc-1)$, or by a factor of $nc/2$. And if each cluster consists of g groups of intracluster symmetric devices, where $n > g$, the total number of symmetry constraints will reduce from the worst case of $nc(nc-1)/2$ to $n^{1/2}c(c-1)$, or by a factor of $n^{3/2}/2$, where $c \gg 1$.

TABLE VI
INTERCONNECT-SYMMETRY-DETECTION ALGORITHM

```
DetectInterconnectSymmetry
begin
  // ListIntraSym = {  ( Mi , Mj )|Mi and Mj are intended matched pair }
  // ListInterSym = { ( Ci , Cj )|Ci and Cj are intended matched cluster }
  foreach ( Mi , Mj ) ∈ ListIntraSym
      ( Gi , Si , Di ) =TileTree( Mi )  // G, S, D=tile-tree for gate, source, drain of M
      ( Gj , Sj , Dj ) =TileTree( Mj )
      match_type =CheckSymmetry ( Mi , Mj )  // type: Gi-Gj , Si-Sj , Di-Dj
      if (match_type == TRUE  for any type of symmetry) then
         RecursiveCreateConstraints ( Ni , Nj )  // N =G or S or D
      end if
  end for
  foreach( Ci , Cj ) ∈ ListInterSym
      (N1i ,N2i ,....,Nni ) =TileTree ( Ci )  // NX =tile-tree from port x of cluster i
      (K1j , K2j ,....,Knj) =TileTree ( Cj )  // KX =tile-tree from port x of cluster j
      for (L=1 to n)
          match=CheckNetSymmetry ( NLi , KLj )
          if (match == TRUE) then
             RecursiveCreateConstraints ( NLi , KLj )
          end if
       endfor
  end for
end
```

C. Interconnect-Symmetry Constraints

Besides the intracluster and intercluster symmetries between devices, symmetries between interconnects or wires of symmetric devices are also important. In order to avoid unnecessary constraints due to wire symmetry, only wire sections that belong to the symmetric devices or clusters and are symmetric on the original layout are preserved.

Table VI presents an algorithm for the detection and generation of constraints for interconnect symmetry. The algorithm uses the intracluster and intercluster symmetry lists found earlier. First, for intracluster symmetric transistor pairs, gate, source, and drain port tiles are located. For the intercluster pairs, all port tiles of the outgoing nets of the clusters are detected. Second, with a depth-first search [21] along each net, a tree of interconnect rectangle tiles from each port is constructed. Next, correct pairing of symmetric nets is established. For a matched transistor pair, since drains and sources are not unique in the CMOS structure, there are five possible symmetric types, i.e., gate–gate, source–source, drain–drain, source–drain, and drain–source. For intercluster symmetries, the match pairs between nets can be recognized from ports of the matched devices in each cluster.

Once the interconnect-trees are correctly paired up, a recursive procedure RecursiveCreateConstraints generates symmetry constraints between the tiles of the nets. Two interconnect tiles are deemed symmetric only when they have the same size, are on the same layer, and have same distance to the corresponding transistor or cluster-symmetry axis. If two interconnect tiles are symmetric, constraints of the form of (5) are generated and the function is recursively called between next tiles in each tree. The procedure terminates if two tiles are not symmetric.

TABLE VII
COMPARISON OF INTRACLUSTER-SYMMETRY DETECTION WITH DLSD

Circuits	# Multi-Fingered Transistors	# Unit Transistors	Design Rule Constraints	Direct Symmetry Detection (DLSD)			Intra-Cluster Constraint Generation		
				Symm. Axes	Tran. Pairs	Symm. Constraints	Symm. Axes	Tran. Pairs	Symm. Constraints
Differential Amplifier	5	5	1,602	1	2	18	1	2	18
Latched Comparator	15	20	8,639	10	19	132	2	6	42
2-stage Opamp	9	48	5,902	69	262	1,578	2	12	78
Folded Cascode Opamp	14	43	8,352	29	173	1,206	6	20	168
4:1 Comparator	20	32	26,182	26	166	1,026	3	13	78
VCO	16	198	645,986	680	5,525	33,156	4	362	2,178
5-bit Flash ADC	435	590	320,937	261	6,218	41,943	12	192	1,302

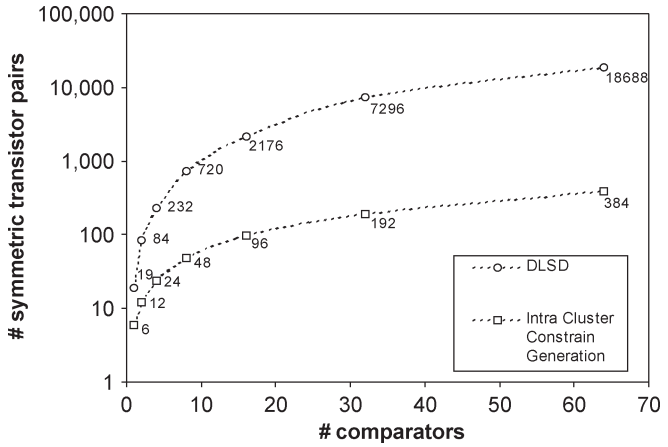

Fig. 17. Scaling of symmetry axes detected by DLSD and intracluster constraint generation.

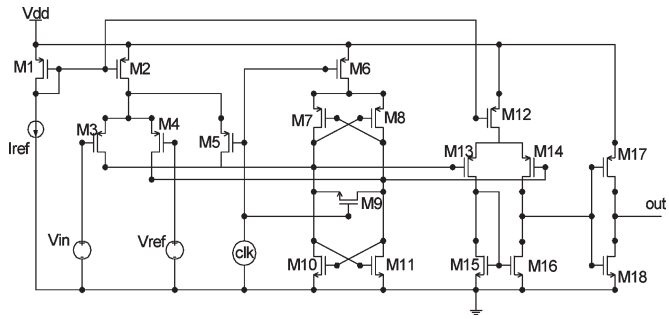

Fig. 18. Schematic of the unit comparator. M_3 and M_4 form the input differential pair.

TABLE VIII
COMPARISON OF LAYOUT RETARGETING OF ADC VIA DLSD AND MULTILEVEL CONSTRAINT GENERATION

Constraint Generation Method	DLSD	Multi-level Method
Multi-finger Transistors	435	435
Unit Transistors	590	590
Design Rule Constraints	320,937	320,937
Resistor Symmetries	63	63
Intra-cluster Symm. Transistor Pairs	-	186
Inter-cluster Symm. Transistor Pairs	-	31
Total Symmetric Transistor Pairs	6,218	217
Total Symmetry Constraints	43,935	2,574
Runtime for Template Creation	16 hrs 10 min.	8 hrs and 52 min.
Layout Generation Runtime	Unfinished after 3 days	1 hr and 51 min.

VIII. EXPERIMENTAL RESULTS

The multilevel-symmetry-constraint-generation scheme has been implemented in IPRAIL. The tool was applied on several analog layouts ranging from an operational amplifier to a 5-bit flash analog-to-digital converter (ADC). In these experiments, generation of the intracluster-symmetry constraints suffices for the circuits other than the ADC. The experimental results comparing the intracluster-symmetry-constraint generation with the direct-layout-symmetry generation (DLSD) [23] are presented first. Then, the results of retargeting the ADC layout with both intra- and intercluster-constraint generation are described.

Intracluster-symmetry-constraint generation starts with a list of matched transistors detected through subcircuit extraction. This method ensures avoidance of unintended layout symmetry that may reside in the layout. The subcircuits are described in schematic form, including transistors, interconnects, and transistor matchings, and are usually available in a standard-cell library. In case the subcircuits or standard-cell libraries are not included with the layout, a built-in set of commonly known subcircuits, such as a current mirror, a differential pair, etc., are used.

Table VII compares intracluster-symmetry-constraint generation with the DLSD method that was initially adopted in IPRAIL. Various symmetry topologies were employed in these layouts. The differential amplifier, the latched comparator, and the 4:1 comparator comprised of symmetric transistors with some multifinger structures. The voltage-controlled oscillator (VCO) was laid out with extensive multifinger symmetric transistors. The two-stage and the folded-cascode operational amplifiers utilized multifinger-interleaved and common-centroid symmetry topologies, respectively. And the 5-bit flash ADC consisted of 32 instances of a latched comparator.

For each method, the number of symmetry axes, symmetric transistor pairs, and symmetry constraints are reported. As DLSD detects symmetries between every pair of unit transistors in each multifinger transistor, a great number of redundant symmetry axes were extracted and a large number of axes were observed for the two-stage operational amplifier and the VCO circuits. In the array structure of the comparator blocks in the 5-bit ADC, transistors of different clusters are aligned in rows and columns. Therefore, the number of symmetries detected from DLSD becomes excessively large.

The difference between the two symmetry-constraint-generation methods is further compared for arrays of comparators. Fig. 17 shows the number of symmetric transistor pairs, in a logarithmic scale, detected by DLSD and the intracluster-symmetry-detection method as the number of comparators is

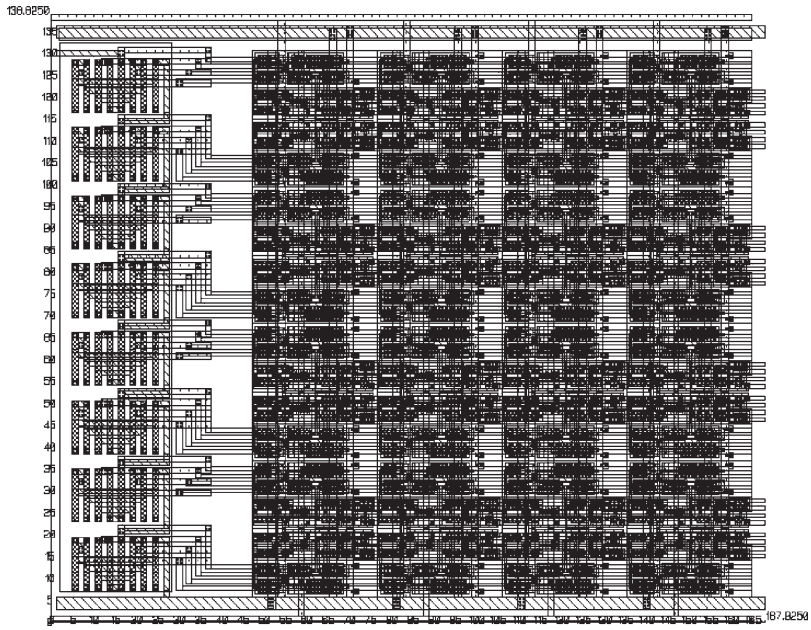

Fig. 19. Analog section of the 5-bit ADC in TSMC 0.25-μm technology. Area is 29300 μm^2.

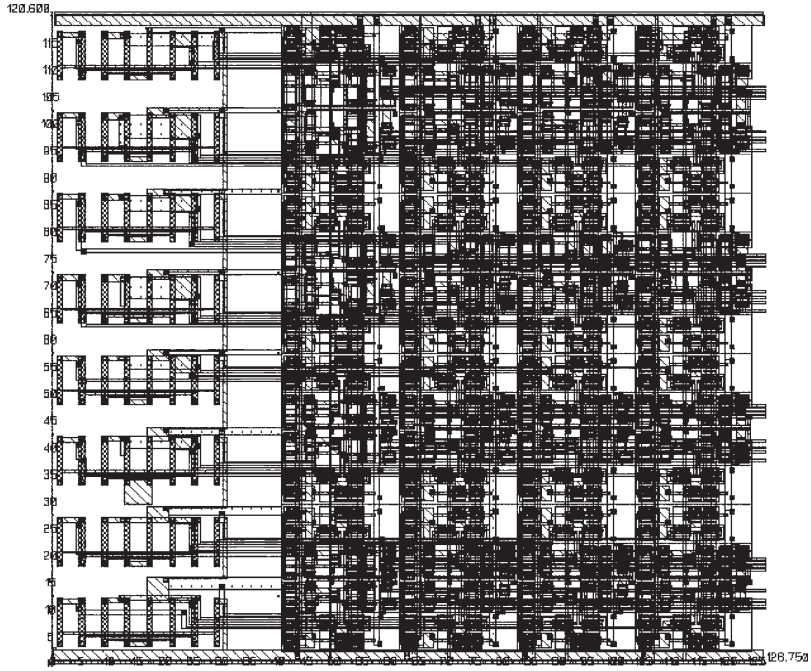

Fig. 20. Retargeted layout of the analog section of a 5-bit ADC in TSMC 0.18-μm technology. Area is 15300 μm^2.

scaled. The graph shows that DLSD detects a huge number of redundant symmetry axes. For instance, DLSD detects transistor pairs about 3.2 times more in one comparator cell and about 48.7 times more in 8×8 comparator cells than the intracluster-symmetry detection.

Next, the results of retargeting the 5-bit flash ADC layout with both intra- and intercluster-constraint generation are presented. The 5-bit flash ADC comprises of a resistor chain, an analog comparator array, and a digital decoder block. The analog comparator array consists of 32 latched comparators. Each comparator, whose schematic is shown in Fig. 18, compares an input signal with different reference-voltage levels obtained from a resistor chain acting as a voltage divider. Based on the input voltage, the analog-comparator section produces a thermometer code. This is then converted to a 5-bit binary output by a 32-to-5 decoder.

The analog section of the ADC was initially designed and laid out manually in the 0.25-μm CMOS Taiwan Semiconductor Manufacturing Company (TSMC) technology process, while the digital-decoder section was designed in a standard-cell-based application-specified integrated circuit (ASIC) flow. To minimize the mismatch between comparators due to process gradients, they are laid out in a one-dimensional common-centroid fashion. The resistor chain is constructed in the

TABLE IX
SPECIFICATIONS OF THE 5-BIT ADC IN 0.25-μm TSMC AND THE TARGET DESIGN IN TSMC 0.18 μm

Performance Parameters	Original Layout 0.25um	Target Layout 0.18um
Supply Voltage	*2.50 V*	*1.80 V*
Reference Voltage	*1.28 V*	*1.28 V*
½ LSB Resolution	*20 mV*	*20 mV*
Sampling Rate	*500 MHz*	*750 MHz*
Differential Non-linearity (max)	*0.12 LSB*	*0.11 LSB*
Integral Nonlinearity (max)	*0.66 LSB*	*0.39 LSB*
Power Consumption	*42 mW*	*18 mW*
Total Area	*79,800* μm^2	*36,650* μm^2

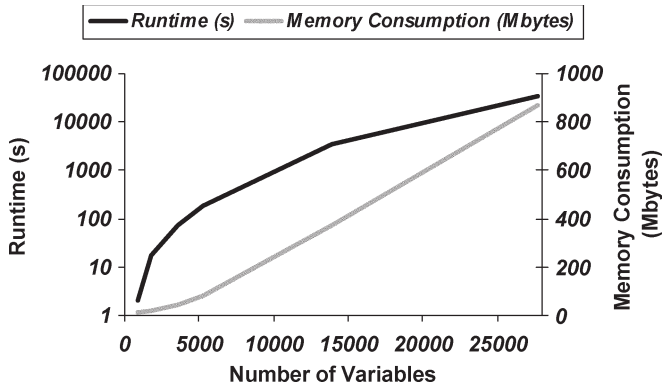

Fig. 21. Runtime (in logarithm scale) and memory consumption (in linear scale) when layout size increases. The template for a layout comprising 32 comparators consisted of 27 668 variables, 320 937 design-rule constraints, and 1932 symmetry constraints.

polysilicon layer and laid out with one-dimensional common-centroid symmetry. In order to alleviate the effect of process and temperature gradients, each resistor is split into two rectangles and laid out similar to the example shown in Fig. 8.

The ADC in TSMC 0.25-μm technology is then retargeted to the TSMC 0.18-μm technology process with different performance specifications. First, incorporating the scan-line method, the intracluster- and intercluster-symmetry-constraint generation, a structural symbolic template was constructed from the layout. The template, in a graph-constraint form, consisted of 29 918 nodes, 320 937 arcs, and 2574 symmetry-related arcs. The symmetry-related arcs were extracted from 186 intracluster symmetric transistors, 31 intercluster symmetries, and 63 resistor symmetries. The statistics on constraints generated by DLSD and the multilevel method are presented in Table VIII.

New transistor and resistor sizes for achieving the desired specifications in TSMC 0.18-μm technology are obtained by Spectre [28] simulations. These sizes are then enforced on the template. The target layout was obtained by solving the template using a combination of LP and a graph-based longest path algorithm. The layout of the comparator-and-resistor section is shown in Fig. 19 for the source layout and in Fig. 20 for the target layout. The specifications achieved in the postlayout simulation using Spectre [28] for both the designs (0.18- and 0.25-μm) are listed in Table IX.

Our experiments on IPRAIL were conducted on a 900-MHz Sun UltraSparc10 workstation. For the 5-bit ADC, the layout-template-extraction phase took 8 hour and 52 minutes, and the generation of the target layout was completed in 1 hour and 51 minutes.

In addition, an automatic-retargeting program was tested on comparator arrays of various sizes to determine the growth in runtime and memory consumption of the algorithm. Fig. 21 shows the increment of runtime (in logarithm scale) and memory (in linear scale) when layout size increases.

IX. CONCLUSION

A new and efficient multilevel-symmetry-constraint-generation method for analog layout retargeting is presented. Several key techniques, such as multifinger transistor identification, subgraph isomorphism-based subcircuit extraction, and layout clustering, have been developed. These techniques establish extensive mappings between the netlist and the layout representation of an analog design and accomplish automatic identification of matched transistor pairs. This facilitates efficient multilevel-constraint generation that enables retargeting of large analog layouts. The efficacy of the multilevel-symmetry-constraint-generation scheme is demonstrated for several complex analog layouts.

IPRAIL, a symbolic-template-based layout-retargeting tool, has been enhanced with this multilevel-symmetry-constraint-generation method. This enables IPRAIL to successfully retarget large analog layouts as has been illustrated by the retargeting of a 5-bit ADC layout. Large analog circuits that are known to take several weeks for manual layout generation are automatically retargeted to different processes and specifications within hours with comparable electrical performances.

Future research includes techniques for migrating physical layouts to considerably different technologies. This entails automatic device removal and generation, as well as constraint relaxation to address potential template infeasibility. Additionally, explicit wire and device parasitic control is essential for high-performance analog and radio-frequency integrated-circuit layouts.

ACKNOWLEDGMENT

The authors would like to thank R. Hartono for help with the development of IPRAIL. C. Baker helped with the design of the digital section of the ADC. The authors are very thankful to S. Aniruddhan and Dr. W. Law for valuable discussions on the design and layout of the analog portion of the ADC.

REFERENCES

[1] M. D. M. Hershenson, S. P. Boyd, and T. H. Lee, "Optimal design of a CMOS opamp via geometric programming," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 1, pp. 1–21, Jan. 2001.

[2] M. J. Krasnicki, R. Phelps, J. R. Hellums, M. McClung, R. A. Rutenbar, and L. R. Carley, "ASF: A practical simulation-based methodology for the synthesis of custom analog circuits," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 2001, pp. 350–357.

[3] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers, "Matching properties of MOS transistors," *IEEE J. Solid-State Circuits,* vol. 24, no. 5, pp. 1433-1440, Oct. 1989.

[4] A. Hastings, *The Art of Analog Layout*. Upper Saddle River, NJ: Prentice-Hall, 2001.

[5] B. Razavi, *Design of Analog CMOS Integrated Circuits*. New York: McGraw-Hill, 2001.

[6] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley, "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 330–342, Mar. 1991.
[7] K. Lampaert, G. Gielen, and W. M. Sansen, "A performance-driven placement tool for analog integrated circuits," *IEEE J. Solid-State Circuits*, vol. 30, no. 7, pp. 773–780, Jul. 1995.
[8] E. Malavasi, E. Charbon, E. Felt, and A. Sangiovanni-Vincentelli, "Automation of IC layout with analog constraints," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 8, pp. 923–942, Aug. 1996.
[9] H. Koh, C. Sequin, and P. Gray, "OPASYN: A compiler for CMOS operational amplifiers," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 9, no. 2, pp. 113–125, Feb. 1990.
[10] 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, Apr. 1990.
[11] J. D. Conway and G. G. Schrooten, "An automatic layout generator for analog circuits," in *Proc. Eur. Design Automation Conf.*, Brussels, Belgium, Mar. 1992, pp. 513–519.
[12] R. Castro-Lopez, F. V. Fernandez, F. Medeiro, and A. Rodriguez-Vazquez, "Generation of technology-independent retargetable analog blocks," *Int. J. Analog Integr. Circuits Signal Process.*, vol. 33, no. 2, pp. 157–170, Dec. 2002.
[13] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C.-J. R. Shi, "Automatic analog layout retargeting for new processes and device sizes," in *Proc. IEEE Int. Symp. Circuits and Systems*, Bangkok, Thailand, May 2003, pp. 704–707.
[14] ——, "IPRAIL—Intellectual property reuse-based analog IC layout automation," *Integr.—VLSI J.*, vol. 36, no. 4, pp. 237–262, Nov. 2003.
[15] E. Malavasi and D. Pandini, "Optimum CMOS stack generation with analog constraints," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 1, pp. 107–122, Jan. 1995.
[16] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C.-J. R. Shi, "Multiple specifications radio-frequency integrated circuit design with automatic template driven layout retargeting," in *Proc. Asia and South Pacific Design Automation Conf.*, Yokohama, Japan, Jan. 2004, pp. 394–399.
[17] S. Bhattacharya, N. Jangkrajarng, R. Hartono, and C.-J. R. Shi, "Hierarchical extraction and verification of symmetry constraints for analog layout automation," in *Proc. Asia and South Pacific Design Automation Conf.*, Yokohama, Japan, Jan. 2004, pp. 400–405.
[18] ——, "Correct-by-construction layout-centric retargeting of large analog designs," in *Proc. IEEE/ACM Design Automation Conf.*, San Diego, CA, Jun. 2004, pp. 139–144.
[19] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. CAD-3, no. 1, pp. 87–100, Jan. 1984.
[20] W. S. Scott and J. K. Ousterhout, "Magic's circuit extractor," in *Proc. IEEE/ACM Design Automation Conf.*, Las Vegas, NV, Jun. 1985, pp. 286–292.
[21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.
[22] S. L. Lin and J. Allen, "Minplex—A compactor that minimizes the bounding rectangle and individual rectangles in a layout," in *Proc. IEEE/ACM Design Automation Conf.*, Las Vegas, NV, Jun. 1986, pp. 123–130.
[23] Y. Bourai and C.-J. R. Shi, "Symmetry detection for automatic analog layout recycling," in *Proc. Asia and South Pacific Design Automation Conf.*, Hong Kong, Jan. 1999, pp. 5–8.
[24] D. Luenberger, *Linear and Nonlinear Programming*, 2nd ed. Reading, MA: Addison-Wesley, 1984.
[25] R. Okuda, T. Sato, H. Onodera, and K. Tamaru, "An efficient algorithm for layout compaction problem with symmetry constraints," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 1989, pp. 148–151.
[26] G. Lakhani and R. Varadarajan, "A wire-length minimization algorithm for circuit layout compaction," in *Proc. IEEE Int. Symp. Circuits and Systems*, Philadelphia, PA, May 1987, pp. 276–279.
[27] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather, "Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm," in *Proc. IEEE/ACM Design Automation Conf.*, Dallas, TX, Jun. 1993, pp. 31–37.
[28] *Spectre User's Manual*, Cadence Design Systems, San Jose, CA, 2000. Version 4.46.

Sambuddha Bhattacharya received the B.Eng. degree in electrical engineering from the Birla Institute of Technology and Science, Pilani, India, in 1997, and the M.S. and Ph.D. degrees in electrical engineering from the University of Washington, Seattle, in 2002 and 2005, respectively.

His research interests are in analog/RF design automation and timing and noise issues in digital circuits.

Nuttorn Jangkrajarng received the B.Eng. degree from Chulalongkorn University, Bangkok, Thailand, in 1997, and the M.S. and Ph.D. degrees from the University of Washington, Seattle, in 1999, and 2006, respectively, all in electrical engineering.

While at the University of Washington, he worked at the Mixed-Signal Computer-Aided Design (MSCAD) Research Laboratory. His research interests are in the fields of mixed-signal very-large-scale integration (VLSI), digital and analog IC design automation, and circuit simulation.

C.-J. Richard Shi (S'92–M'93–SM'99–F'06) received the Ph.D. degree in computer science from the University of Waterloo, Waterloo, ON, Canada, in 1994.

From 1994 to 1998, he worked at Analogy, Rockwell Semiconductor Systems, and the University of Iowa. In 1997, he founded the IEEE International Workshop on Behavioral Modeling and Simulation. In 1998, he joined the University of Washington, Seattle, where he is currently a Professor in electrical engineering. He is a key contributor to the IEEE std 1076.1-1999 (VHDL-AMS) standard for the description and simulation of mixed-signal circuits and systems. His research interests include several aspects of the computer-aided design and test of integrated circuits and systems, with particular emphasis on analog/mixed-signal and deep-submicrometer circuit modeling, simulation, and design automation.

Dr. Shi was an Associate Editor, as well as a Guest Editor, of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II, Analog and Digital Signal Processing. Since 1999, he has been an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He has received several awards for his research including a Doctoral Prize from the Natural Science and Engineering Research Council of Canada (1995), a Best Paper Award from the 1998 IEEE VLSI Test Symposium, a Best Paper Award from the 1999 IEEE/ACM Design Automation Conference, a National Science Foundation CAREER Award (2000), and a Semiconductor Research Corporation (SRC)-TECHCON Best Paper Award (2003).