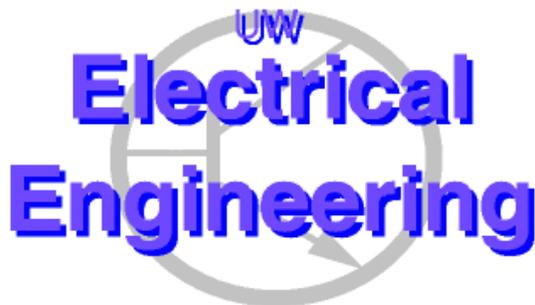

Multi-Level Symmetry Constraint Generation for Retargeting Large Analog Layouts

Sambuddha Bhattacharya, Nuttorn Jangkrajarn, Roy Hartono
and C.-J. Richard Shi
{sbb, njangkra, rhartono, cjshi}@ee.washington.edu

Dept of EE, University of Washington
Seattle WA, 98195-2500



UWEE Technical Report
Number UWEETR-2004-0004
March 30, 2004

Department of Electrical Engineering
University of Washington
Box 352500
Seattle, Washington 98195-2500
PHN: (206) 543-2150
FAX: (206) 543-3842
URL: <http://www.ee.washington.edu>

Multi-Level Symmetry Constraint Generation for Retargeting Large Analog Layouts^{*}

Sambuddha Bhattacharya, Nuttorn Jangkrajarn, Roy Hartono
and C.-J. Richard Shi

{ sbb, njangkra, rhartono, cjshi }@ee.washington.edu

Dept of EE, University of Washington at Seattle
Seattle WA, 98195-2500

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 multi-level 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 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 re-layout is known to take weeks, our layout-retargeting tool generates the target layout in hours and achieves comparable electrical performance.

Keywords: *Analog Integrated Circuit Design; Analog Physical Layout Automation; Layout Symmetry; Device Matching; Layout Retargeting.*

^{*} This research has been supported in part by U.S. Defense Advanced Projects Agency NeoCAD Program under Grant No. 66001-01-1-8920, in part by National Science Foundation (NSF) ITR Program under Grant No. 9985507, and in part by a grant from Conexant Systems. Some preliminary results of this work have been presented in the Asia and South Pacific Design Automation Conference 2004 and the Design Automation Conference 2004.

1. 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 circuits, the absence of CAD tools in the analog domain presents serious bottleneck to the fast realization of mixed-signal designs.

In analog design, trade-offs 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 form of optimization tools [1][2] that automatically synthesize analog circuits meeting 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, leading to DC offsets, finite even-order distortion and lower common-mode rejection [4][5]. These effects can be alleviated by 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][7][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][10][11][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][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 re-target 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][10][11][12][13][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 Intellectual Property (IP) blocks. Our new contributions are as follows:

- Large analog circuits not only require symmetric layouts 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 netlist and layout representations of the design, a new multi-level constraint generation method is introduced. This multi-level templating scheme achieves a smaller template size thereby allowing retargeting of large layouts.

- We present an automatic method of identifying only relevant matched transistors from the circuit netlist and imposing corresponding layout-level symmetry constraints.
- 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 complete automation of template creation and subsequent layout generation, thereby allowing retargeting of large analog layouts. Some preliminary results of this work were presented in [17][18].

The rest of the paper is organized as follows. Section 2 briefly describes the layout-reuse methodology with emphasis on IPRAIL and discusses various challenges in retargeting large analog layouts. Section 3 provides an overview of the proposed constraint generation scheme. Sections 4, 5 and 6 elaborate the various steps to establish mappings between the netlist and layout representations of the existing design. Section 7 describes the actual constraint generation process. Section 8 presents the experimental results. Section 9 concludes the paper.

2. 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.

2.1 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 structural 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.

2.1.1 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, connectivities, technology process design rules, and analog layout intricacies.

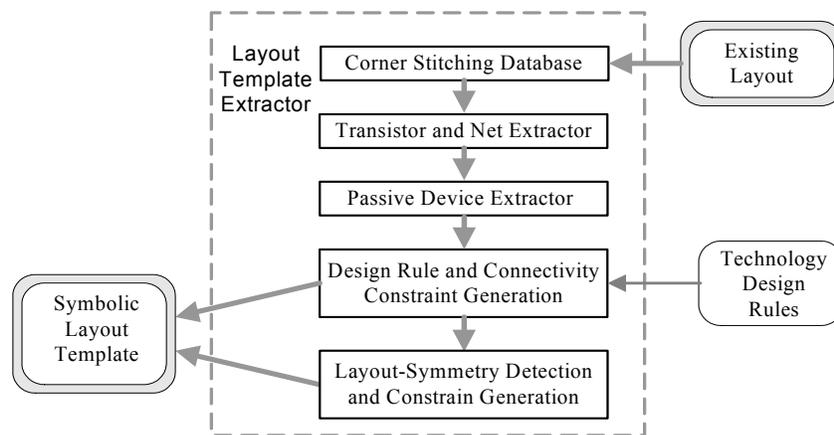


Figure 1: Layout Template Extractor Flow.

The detailed flow of template extraction is shown in Figure 1. First, the input layout is parsed in 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 [20] 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 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.

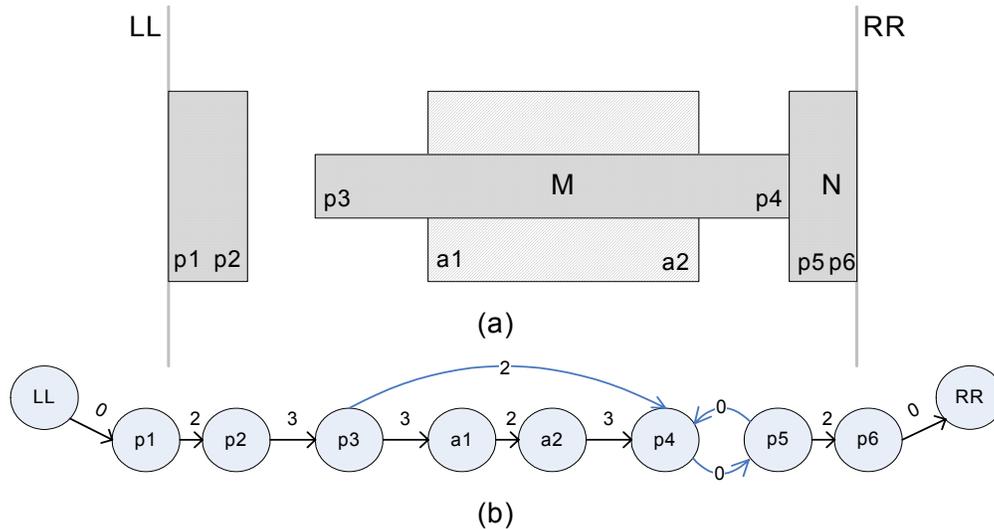


Figure 2: Geometric constraints in graph form (a) A layout example. (b) Horizontal constraint graph.

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 a graph form where each tile variable is represented by a node in the graph. An 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 Figure 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 simplified example of Figure 3, this implies equi-sized 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:

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

$$(s_0 - g_{right}) - (h_{left} - s_0) = 0 \quad (2)$$

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

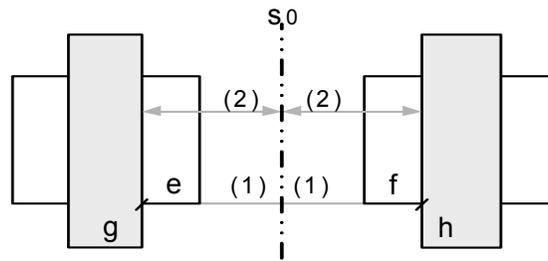


Figure 3: A simplified layout of a symmetric pair of unit transistors. S_0 denotes the axis of symmetry.

2.1.2 Layout Generator

Figure 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].

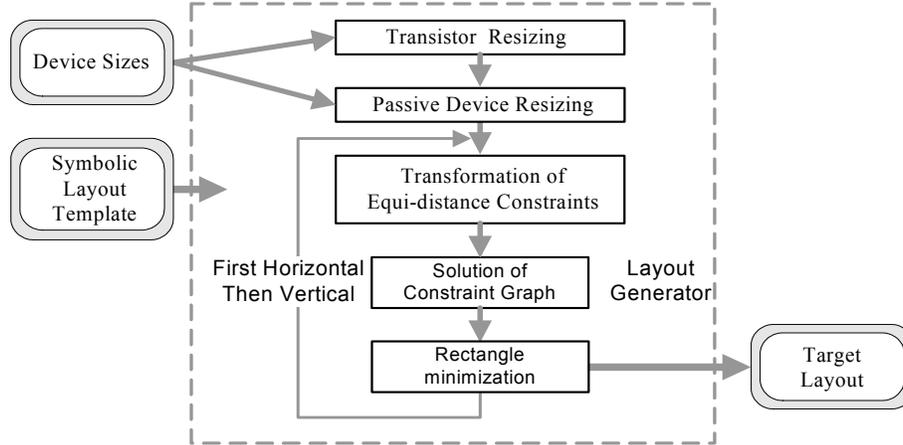


Figure 4: Flow of the Layout Generator.

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 Figure 5c. A shadow tile is a temporary non-physical-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:

$$x_{shadow} - x_{other} \geq d \quad (3)$$

Connectivity between a passive device and the 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 Figure 5d. 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.

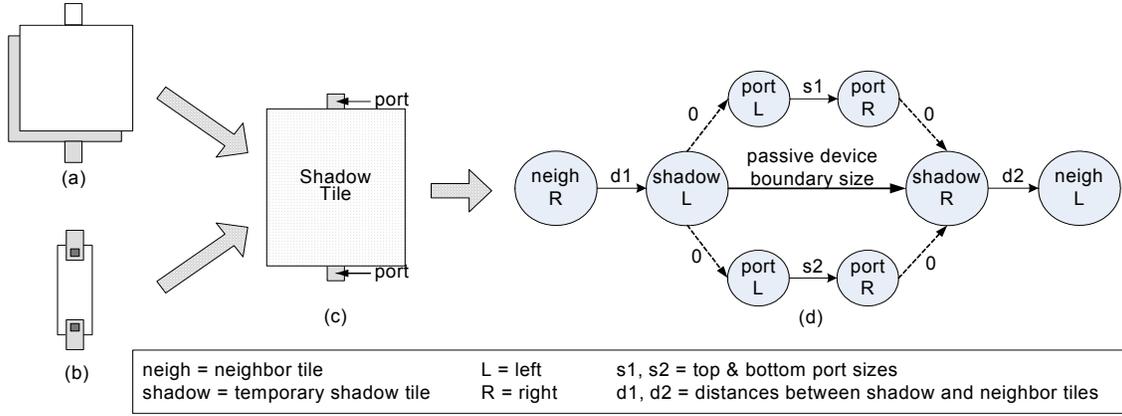


Figure 5: Passive device retargeting with shadow tiles. (a) MIM or P-P capacitor (b) on-chip unit resistor (c) Passive replaced by shadow tile (d) Simplified constraints for passive retargeting. The distances from neighbors are obtained according to Eq. (3).

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:

$$\text{Min} (x_R - x_L) \quad (4.1)$$

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

$$x_i - x_j = \text{constant} \quad (4.3)$$

$$x_i - x_k = x_k - x_j \quad (4.4)$$

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 and right edges of the tiles of the layout. As has been illustrated in Figure 2, the constraints in Eq. (4.2) and Eq. (4.3) can be expressed in a 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 equi-distance constraints of Eq. (4.4), these cannot be expressed in the graphical form.

The problem in Eq. (4) is a standard linear programming (LP) [24] formulation. However, solving large problems with LP can be very expensive computationally. If the constraints in Eq. (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 [22]. In order to solve the modified compaction problem with graph based methods, the constraints of Eq. (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 Eq. (4.2) and Eq. (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 Eq. (4.4). This is accomplished by one shortest-path run on the constraint-graph for each variable in the equidistance constraint of Eq. (4.4). This small set of core constraints along with the constraints in Eq. (4.4) are then solved by LP. This yields the right-hand side constants for Eq. (4.4) in the form

$$x_i - x_k = x_k - x_j = \text{constant} \quad (5)$$

These transformed constraints are then imposed back on the constraint-graph corresponding to Eqs. (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.

2.2 Challenges in Retargeting Layouts of Large Designs

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 multi-finger transistors laid out symmetrically in one or two-dimensional cross-coupled topologies, as shown in Figure 6 and Figure 7 respectively. Under the DLSD scheme [23], the layout of Figure 6 has twenty-one axes of symmetry marked by the axes s_1 to s_{21} and sixty-six matched unit transistor pairs. The layout in Figure 7 has twelve axes of symmetry as indicated by the axes s_1 to s_{12} and thirty matched unit transistor pairs.

As has been explained in Section 2.1.2, each symmetry axes introduces equidistance constraints that necessitate multiple longest-path based transformations on the constraint graph [25]. Clearly, presence of large number of symmetry 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.

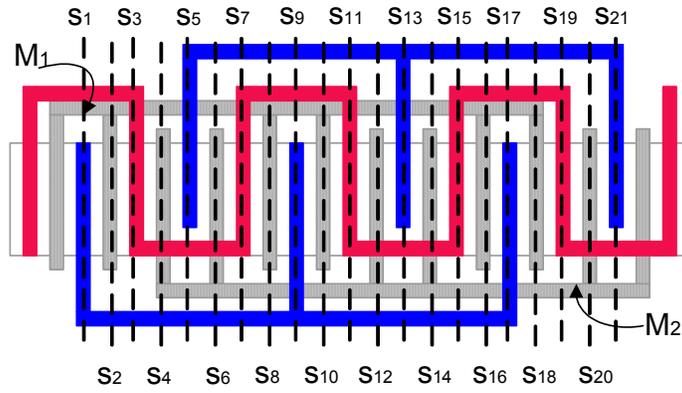


Figure 6: One-dimensional cross-coupled symmetric multi-finger transistor pair. The rectangles with dotted patterns represent the poly-silicon layer.

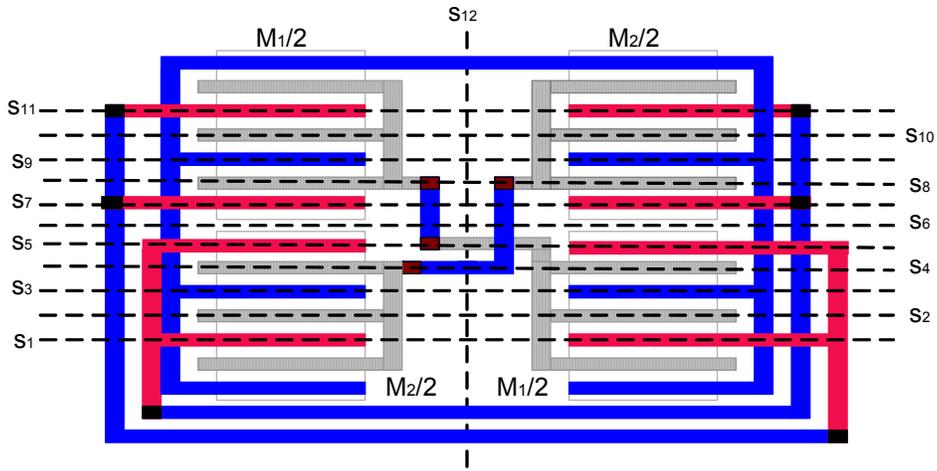


Figure 7: Two-dimensional symmetric transistor pair. The rectangles with dotted patterns represent the poly-silicon layer.

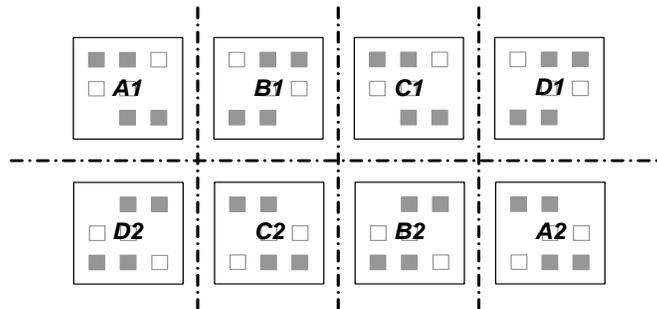


Figure 8: Four comparator subcircuits laid-out in a split-symmetric common-centroid layout.

In addition to matched transistors, 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 4 unit comparators. In order to alleviate the differences in the performances of the

unit comparators and the consequent non-linearities in electrical behavior, they need to be laid out in a common-centroid fashion. This is illustrated in Figure 8 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.

3. MULTI-LEVEL MAPPING & CONSTRAINT GENERATION

In this section, we present a new method of multi-level 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 Figure 9. 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 connectivities of the unit transistors, multi-finger transistors are identified and a compact netlist is obtained.

Large analog circuits consist of several commonly used subcircuit topologies, such as differential-pairs, current-mirrors, comparators etc. Normally, an analog design environment consists of a library of such subcircuits. In the *subcircuit-extraction* step, two goals are accomplished. First, all instances of the library subcircuits in the compact netlist are identified thereby creating partitions in the netlist. Second, the designer-intended matched transistors in the circuit netlist are recognized. This list of matched transistor pairs in the netlist is used to extract the transistors' layout-symmetry constraints.

Based on the partitioned netlist obtained at the end of subcircuit-extraction, clustering of the 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 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.

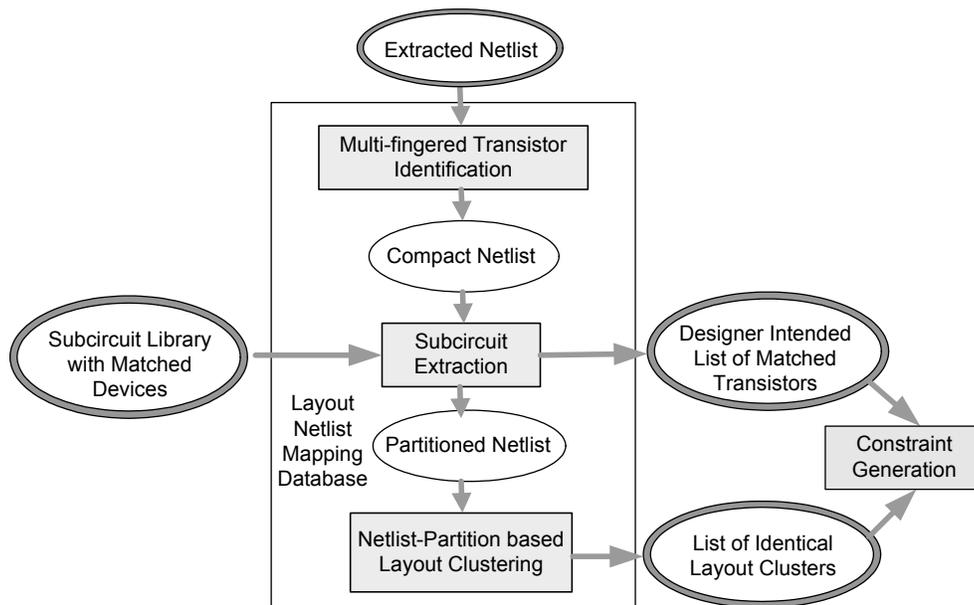


Figure 9: Overview of the multi-level mapping and constraint generation flow.

Table 1: Mapping and constraint generation algorithm.

```

GenerateMultiLevelConstraints
begin
  ExtractNetsTransistors
  DetectMultiFingerTransistors
  for each  $s \in L$  //  $s = \text{subcircuit}, L = \text{Library}$ 
    ExtractNetlistSubcircuits
  end for
  CreateLayoutClusters
  GenerateIntraClusterSymmetry
  GenerateInterClusterConstraints
end
  
```

Table 1 presents the multi-level constraint generation algorithm. First, the procedure *ExtractNetsTransistors* extracts the netlist from the layout. Next, the *DetectMultiFingerTransistors* procedure groups the physically contiguous unit transistors in the layout into multi-finger 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 procedure *CreateLayoutClusters* accomplishes layout partitioning based on how the extracted subcircuit instances are laid out. The *GenerateIntraClusterSymmetry* routine then generates the symmetry constraints within each layout cluster based on the list of matched-transistors obtained previously. Finally, the constraints between different layout clusters are generated by *GenerateInterClusterConstraints* procedure.

4. MULTI-FINGER 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 multi-finger transistors based on their physical adjacency and terminal connectivity. A multi-finger transistor M is defined as a set of physically contiguous elements C^S i.e., $C \in M$. Each contiguous element is defined as $C = \{T \mid T \in T^S, \forall T \{G_T, S_T, D_T\} = \{G_M, S_M, D_M\}, \text{ and } \forall T \in C \text{ are physically contiguous}\}$, where $\{G_M, S_M, D_M\}$ is the set of the gate, source and drain nets of the multi-finger transistor M . The detection of the multi-finger transistors establishes a compact netlist comprising of the set of nets N^S and the set of multi-finger transistors M^S defined as $\{M \mid \forall M, \exists \text{ a unique } \{G_M, S_M, D_M\} \subset N^S\}$.

Table 2: Algorithm for multi-finger transistor detection.

```

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

```

The *DetectMultiFingerTransistors* procedure in Table 2 presents the algorithm for the detection of multi-finger transistors from the extracted netlist. Each multi-finger 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 multi-finger transistor M is created if it does not already exist in the hashtable. This is accomplished by a call to the routine *CheckCreateMFT*. The routine *CheckCreateContiguous* then checks if the unit transistor T is aligned with one of the contiguous elements in M . If T is not physically contiguous with any $C \in M$, a new contiguous element is created. In either case, the routine *InsertSorted* inserts T into a list of unit transistors of the corresponding contiguous element.

For example, the one-dimensional cross-coupled symmetric transistor-pair of Figure 6 contains 2 multi-finger transistors, each with three contiguous sets of two unit transistors. And the common-centroid layout of Figure 7 has 2 multi-finger transistors, each with two contiguous sets of three unit transistors.

5. SUBCIRCUIT EXTRACTION

The multi-finger transistor detection results in a compact netlist. The subcircuit extraction process identifies instances of library subcircuits in this compact netlist. As mentioned in Section 3, this partitioning of the netlist is essential for the following: (a) the automatic identification of matched transistor pairs in the netlist and (b) the subsequent layout clustering.

The subcircuit extraction involves detection of the instances of a subcircuit Ψ in the circuit netlist Γ as illustrated in Figure 10. This is accomplished by a subgraph isomorphism algorithm [27]. First, both the subcircuit Ψ and the circuit Γ are expressed as undirected bi-partite graphs. Figure 11 shows the bipartite graphs corresponding to the differential pair and the differential stage circuits of Figure 10. 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 bi-partite graphs of Figure 11, the circular vertices correspond to the nets and the rectangular vertices correspond to the transistors.

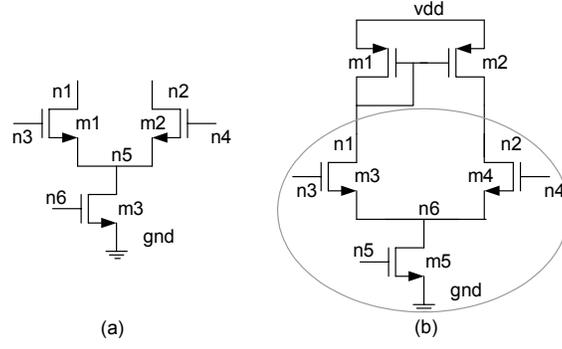


Figure 10: (a) A differential pair subcircuit (Ψ) (b) A differential stage circuit (Γ) that comprises an instance of a differential pair as indicated by the encircled part of the schematic.

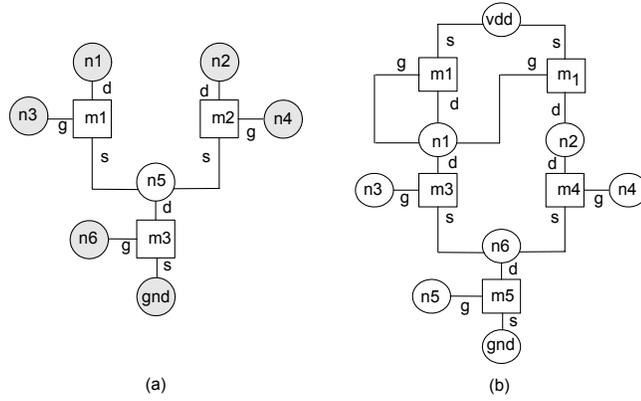


Figure 11: Bipartite graphs of (a) A differential pair subcircuit (b) A differential stage circuit. The circular vertices represent nets and the rectangular vertices represent transistors.

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 Figure 11(a) has an initial label 3. During iterative labeling, the device and net vertices are relabeled according to the following equations.

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

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

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 $wt(S)$, $wt(D)$ and $wt(G)$. As

illustrated in Eq. (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 Figure 11(a) after the 2nd iteration is given as $n5^{label} = 3 + wt(S) * 2N + wt(D) * N$.

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 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 Figure 11(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 Figure 11(a), the 2nd labeling iteration marks the vertices $m1$, $m2$ and $m3$ as corrupt vertices. The search for key vertex terminates when all vertices in the bipartite graph of the subcircuit Ψ are marked corrupt. The vertex with the smallest label is determined to be the key vertex in the bipartite graph of subcircuit Ψ . All vertices in the bipartite graph of the circuit Γ with the same label as the key vertex are marked as *candidate vertices*. For the bipartite graphs of Figure 11, 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 re-labels 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 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 Figure 10(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 Figure 10(b) are therefore marked as matched transistors. Thus, the subcircuit extraction step also identifies all designer-intended matched transistor pairs in the circuit.

6. LAYOUT PARTITIONING

6.1 Layout Clustering

The subcircuit extraction step creates several netlist partitions, each corresponding to a subcircuit instance. In the 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 3.

Table 3: 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  $m \in Q_{prox}$ 
    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
```

The algorithm starts from any seed device, m_{seed} , which belongs to a netlist partition Ψ . First, a layout cluster, L_c , is 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 Figure 8, 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 multi-finger transistor may be composed of two or more contiguous elements that are laid-out far apart to account for different process gradients. Thus, a layout cluster may consist of only some contiguous elements of a given multi-finger transistor.

6.2 Identical Layout Cluster Detection

The previous steps mark each each contiguous element of a multi-finger 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, connectivities and relative positions. The algorithm for detecting identical layout clusters is shown in Table 4.

Table 4: Algorithm for identifying identical layout clusters.

```

DetectIdenticalLayoutClusters ( $L_A, L_B$ )
begin
   $H_A = \text{CollectTiles}(L_A)$       // heap of tiles, sorted wrt x,y
   $H_B = \text{CollectTiles}(L_B)$ 
   $H_{BR} = \text{Reverse}(H_B)$ 
  if ( $\text{Translate}(H_A, H_B) == \text{TRUE}$ ) then
    return TranslateMatched
  else if ( $\text{Mirror}(H_A, H_{BR}) == \text{TRUE}$ ) then
    return FlippedMatched
  else
    return NotMatched
  endif
end

```

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 an increasing order with respect to the co-ordinates of the leftmost corner. The two heaps are pair-wise 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 a decreasing order of their rightmost corner. If the two heaps H_A and H_{BR} are identical upon pair-wise comparison, then they

represent *flip-matched* clusters. For example, in the simple layout of Figure 8, the clusters A_I and C_I are translate-matched while clusters B_I and C_I are flip-matched.

7. MULTI-LEVEL CONSTRAINT GENERATION

Apart from the connectivity and the design rule constraints described in Section 2, additional constraints are needed to maintain symmetry between multi-finger transistors, nets, different layout clusters and passive devices. This section describes a multi-level constraint generation method that reduces the number of such constraints necessary for retargeting large analog layouts.

7.1 Intra-Cluster 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 intra-cluster symmetry constraint generation scheme checks for 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 Eqs. (1) and (2) for retargeting.

The algorithm for intra-cluster layout symmetry detection is shown in Table 5. 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 elements. Based on the topology, the unit transistors are inserted into two or four sorted lists. Thus, for the common-centroid topology of Figure 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 *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 symmetric layout of Figure 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 co-ordinate based double sorting of the unit

transistors in each multi-finger 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.

Table 5: Intra-cluster symmetry detection algorithm.

```

DetectIntraClusterSymmetry
begin
  // ListSym = { (Mi, Mj) | Mi and Mj are intended matched pair }
  for each (Mi, Mj) ∈ ListSym
    topology = DetectTopology (Mi, Mj)
    if (topology == common_centroid) then
      LL = InsertToList (Mi, Mj, left)
      LR = InsertToList (Mi, Mj, right)
      LB = InsertToList (Mi, Mj, bottom)
      LT = InsertToList (Mi, Mj, top)
      CheckSymmetry (LL, LR)
      CheckSymmetry (LB, LT)
    else if (topology == horizontal_interleaving) then
      LL = InsertToList (Mi, Mj, left)
      LR = InsertToList (Mi, Mj, right)
      CheckSymmetry (LL, LR)
    else if (topology == vertical_interleaving) then
      LB = InsertToList (Mi, Mj, bottom)
      LT = InsertToList (Mi, Mj, top)
      CheckSymmetry (LB, LT)
    else // simple transistor layout
      CheckSymmetry (Mi, Mj)
    end if
  end for
end

```

7.2 Inter-Cluster Constraints

Recall from Section 2.2 that identical layout-clusters may be laid-out either mirror-wise flipped or translated with respect to each other. Figure 8 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 intra-cluster 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 inter-cluster 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.

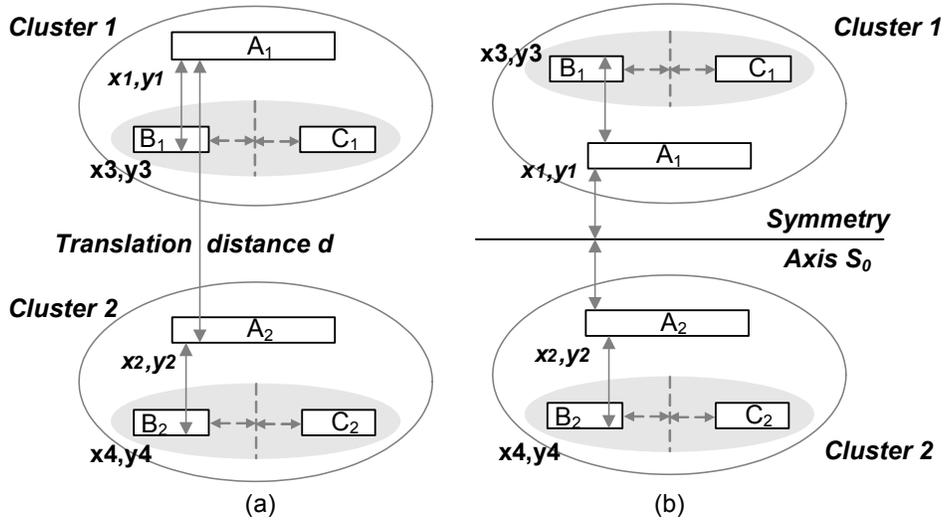


Figure 12: (a) Translated matched clusters. (b) Flipped matched clusters. Symmetric devices B_i and C_i within each cluster are grouped together as indicated by the shaded background. Intra-cluster symmetry is indicated with dotted lines.

Consider the examples in Figure 12 that show translation and flip of two identical layout clusters named *Cluster1* and *Cluster2*. In i^{th} cluster, the intra-cluster 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 inter-cluster 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 Figure 12 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 Figure 12(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 .

$$x_1 - x_3 = x_2 - x_4 \quad (8)$$

$$y_1 - y_3 = y_2 - y_4 \quad (9)$$

where, (x_1, y_1) and (x_2, y_2) are the lower left co-ordinates of the devices A_1 and A_2 , (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.

$$y_1 - y_2 \geq d \quad (10)$$

$$x_1 = x_2 \quad (11)$$

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 Figure 12(b), the group-reference devices in the two clusters are related with equations similar to Eq. (8) and (9). The two cluster-reference devices are related by the following equation

$$y_1 - s_0 = s_0 - y_2 \quad (12)$$

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

Based on the intra-cluster and inter-cluster 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 intra-cluster symmetric devices, where $n > g$, the total number of symmetry constraints will reduce from the worst-case of $nc(nc-1)/2$ to $n^{\frac{1}{2}}c(c-1)$ or by a factor of $n^{\frac{3}{2}}/2$, when $c \gg 1$.

7.3 Symmetry Constraints for Passive Devices

Large layouts often have multiple passive devices that are designed to be identical to each other. Consider the resistor-chain layout of Figure 13. 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.

The template extractor clusters the resistors into two groups, one on the left and one on the right each with half of the resistors A , B and C . It then imposes symmetry constraints between the

two groups as has been shown in Figure 13. The sizes of the resistors under new specifications are imposed as fixed width constraints in the manner explained in Section 2.1.

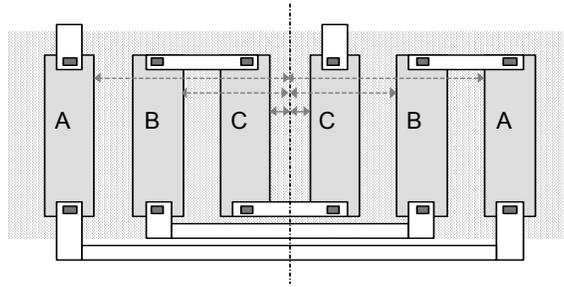


Figure 13: One-dimensional symmetric layout of on-chip resistors.

8. EXPERIMENTAL RESULTS

The multi-level 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 intra-cluster symmetry constraints suffices for the circuits other than the ADC. The experimental results comparing the intra-cluster 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 inter cluster constraint generation are described.

The intra-cluster 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. Table 6 compares intra-cluster symmetry constraint generation with 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 multi-finger structures. The voltage-controlled oscillator was laid out with extensive multi-finger symmetric transistors. The two-stage and the folded-cascode operational amplifiers utilized multi-finger interleaved and common-centroid symmetry topologies respectively. And the 5-bit flash analog-to-digital converter consisted of 32 instances of a latched-comparator.

For each method, the number of symmetry axes, the number of symmetric transistor pairs, and the number of symmetry-constraints are reported. As DLSD detected symmetries between every pair of unit transistors in each multi-finger 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 voltage-controlled oscillator 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 detected symmetries from DLSD becomes excessively large.

Table 6: Comparison of Intra-Cluster Symmetry Detection with Direct Symmetry Detection (DLSD).

Circuits	# Multi-Fingered Transistors	# Unit Transistors	Design Rule Constraints	Direct Symmetry Detection			Intra-Cluster Constraint Generation		
				Symmetry Axes	Transistor Pairs	Symmetry Constraints	Symmetry Axes	Transistor Pairs	Symmetry 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

The difference between the two symmetry constraint generation methods is further compared for arrays of comparators. Figure 14 shows the number of symmetric transistor pairs, in a logarithmic scale, detected by DLSD and the intra-cluster symmetry detection method as the number of comparators is 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 1 comparator cell and about 48.7 times more in 8x8 comparator cells than the intra-cluster symmetry detection.

Next, the results of retargeting the 5-bit flash ADC layout with both intra and inter cluster constraint generation is 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 Figure 15, compares an input signal with different reference voltage levels obtained from 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.

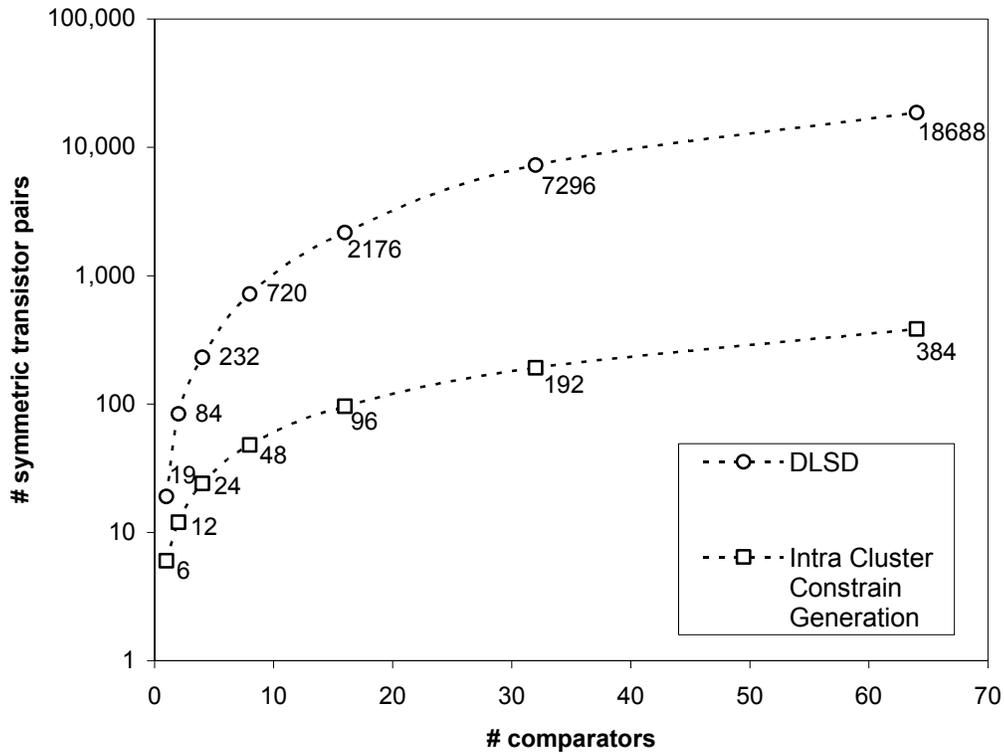


Figure 14: Scaling of symmetry axes detected by DLSD and intra-cluster constraint generation.

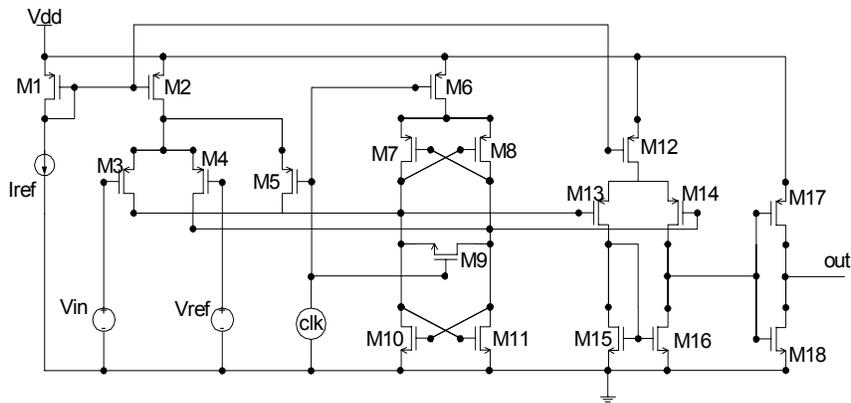


Figure 15: Schematic of the unit comparator. M_3 and M_4 form the input differential pair.

The analog section of the ADC was initially designed and laid-out manually in the $0.25\mu\text{m}$ CMOS TSMC technology process, while the digital decoder section was designed in a standard-cell based 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 poly-silicon 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 Figure 13.

The ADC in TSMC 0.25 μm technology is then retargeted to TSMC 0.18 μm technology process with different performance specifications. First, incorporating the scan-line method, the intra-cluster and inter-cluster symmetry constraint generation, a structural symbolic template was constructed from the layout. The template, in a graph constraint form, consisted of 29,918 nodes, 1,113,599 arcs, and 2,574 symmetry-related arcs. The symmetry-related arcs were extracted from 186 intra-cluster symmetric transistors, 31 inter-cluster symmetries, and 63 resistor symmetries. The statistics on constraints generated by DLSD and by our multi-level method are presented in Table 7.

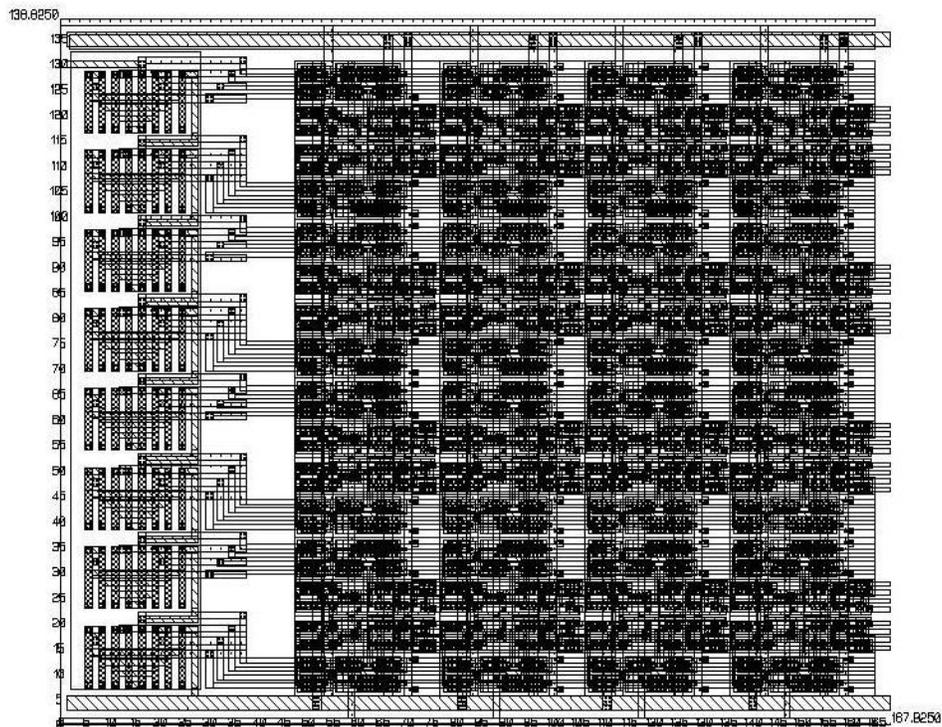


Figure 16: Analog section of the 5-bit ADC in TSMC 0.25 μm technology. Area is 29,300 μm^2 .

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 by combination of linear programming and graph-based longest-path algorithm. The layout of the comparator and resistor section is shown in Figure 16 for the source layout and in Figure 17 for the target layout.

The automatic retargeting program was conducted on a 900MHz SUN UltraSparc3 workstation. The layout template extractor phase took 8 hours 52 minutes, and the generation of target layout was completed in 1 hour 51 minutes. The specifications achieved in the post-layout

simulation in Spectre [28] for both the designs (0.18 μm and 0.25 μm) are listed in Table 8.

Table 7: Comparison of layout retargeting of ADC via DLSD and multi-level constraint generation.

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

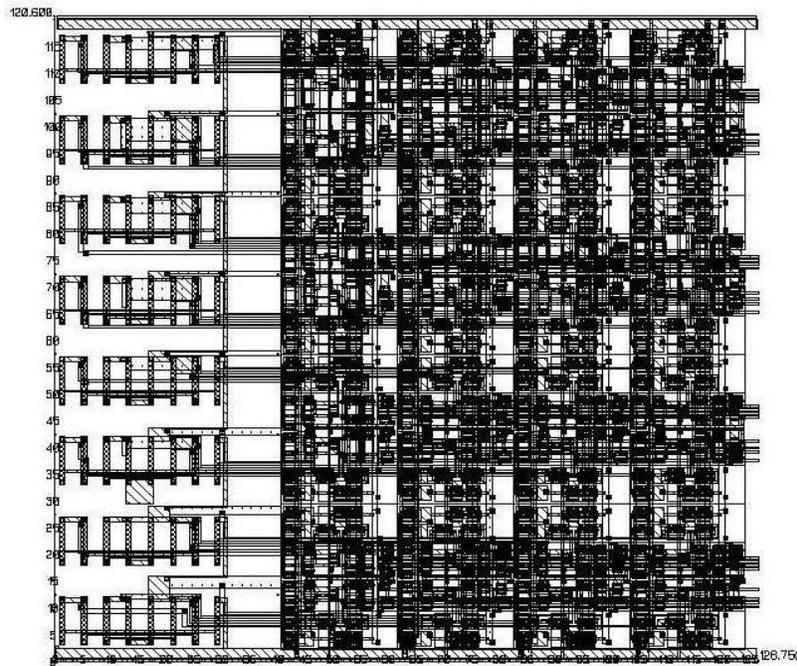


Figure 17: Retargeted layout of analog section of 5-bit ADC in TSMC 0.18 μm . Area is 15,300 μm^2

Table 8: Specifications of the 5-bit ADC in 0.25 μm TSMC and the target design in TSMC 0.18 μm .

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

9. CONCLUSIONS

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

IPRAIL, a symbolic template based layout-retargeting tool, has been enhanced with this multi-level symmetry constraint generation method. This enables the IPRAIL tool suite 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 performance.

REFERENCES

- [1] M. D. M. Hershenson, S. P. Boyd and T. H. Lee, "Optimal design of a CMOS opamp via geometric programming", *IEEE Trans. Computer-Aided-Design*, vol. 20, 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", *Proc. IEEE/ACM International Conference on Computer-Aided-Design*, 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 Jour. Solid State Circuits*, vol. 24, pp. 1433-1440, Oct. 1989.
- [4] A. Hastings, *The Art of Analog Layout*, Prentice Hall, 2001.
- [5] B. Razavi, *Design of Analog CMOS Integrated Circuits*, 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 Jour. Solid State Circuits*, vol. 26, pp. 330-342, Mar. 1991.
- [7] K. Lampaert, G. Gielen and W. M. Sansen, "A performance-driven placement tool for analog integrated circuits", *IEEE Jour. Solid State Circuits*, vol. 30, 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. Computer-Aided-Design*, vol. 15, pp. 923-942, Aug. 1996.
- [9] H. Koh, C. Sequin and P. Gray, "OPASYN: A compiler for CMOS operational amplifiers", *IEEE Trans. Computer-Aided-Design*, vol. 9, pp. 113-125, Feb. 1990.
- [10] H. Onodera, H. Kanbara and K. Tamaru, "Operational-amplifier compilation with performance optimization", *IEEE Jour. Solid State Circuits*, vol. 25, pp. 466-473, Apr. 1990.
- [11] J. D. Conway and G. G. Schrooten, "An automatic layout generator for analog circuits", *Proc. European Design Automation Conference*, 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", *International Jour. Analog Integrated Circuits and Signal Processing*, vol. 33, pp. 157-170, Dec. 2002.
- [13] N. Jangkrajarn, S. Bhattacharya, R. Hartono, and C.-J. R. Shi, "Automatic analog layout retargeting for new processes and device sizes", *Proc. IEEE International Symposium on*

Circuits and Systems, May 2003, pp. 704-707.

- [14] N. Jangkrajarn, 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, Nov. 2003.
- [15] E. Malavasi and D. Pandini, "Optimum CMOS stack generation with analog constraints", *IEEE Trans Computer-Aided-Design*, vol. 14, pp. 107-122, Jan. 1995.
- [16] N. Jangkrajarn, S. Bhattacharya, R. Hartono and C.-J. R. Shi, "Multiple specifications radio-frequency integrated circuit design with automatic template driven layout retargeting", *Proc. Asia and South Pacific Design Automation Conference*, Jan. 2004, pp. 394-399.
- [17] S. Bhattacharya, N. Jangkrajarn, R. Hartono and C.-J. R. Shi, "Hierarchical extraction and verification of symmetry constraints for analog layout automation", *Proc. Asia and South Pacific Design Automation Conference*, Jan. 2004, pp. 400-405.
- [18] S. Bhattacharya, N. Jangkrajarn, R. Hartono and C.-J. R. Shi, "Correct-by-construction layout-centric retargeting of large analog designs", *Proc. IEEE/ACM Design Automation Conference*, Jun. 2004, pp. 139-144.
- [19] J. K. Ousterhout, "Corner Stitching: A data-structuring technique for VLSI layout tools", *IEEE Trans Computer-Aided-Design*, vol. 3, pp. 87-100, Jan. 1984.
- [20] W. S. Scott and J. K. Ousterhout, "Magic's circuit extractor", *Proc. IEEE/ACM Design Automation Conference*, Jun. 1985, pp. 286-292.
- [21] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.
- [22] S. L. Lin and J. Allen, "Minplex – A compactor that minimizes the bounding rectangle and individual rectangles in a layout", *Proc. IEEE/ACM Design Automation Conference*, Jun. 1986, pp. 123-130.
- [23] Y. Bourai and C.-J. R. Shi, "Symmetry detection for automatic analog layout recycling", *Proc. Asia and South Pacific Design Automation Conference*, Jan. 1999, pp. 5-8.
- [24] D. Luenberger, *Linear and Nonlinear Programming 2nd Edition*, Addison-Wesley, 1984.
- [25] R. Okuda, T. Sato, H. Onodera and K. Tamaru, "An efficient algorithm for layout compaction problem with symmetry constraints", *Proc. IEEE/ACM International Conference on Computer-Aided-Design*, Nov. 1989, pp. 148-151.
- [26] G. Lakhani and R. Varadarajan, "A wire-length minimization algorithm for circuit layout

compaction”, *Proc. IEEE International Symposium on Circuits and Systems*, May 1987, pp. 276-279.

- [27] M. Ohlrich, C. Ebeling, E. Ginting and L. Sather, “Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm”, *Proc. IEEE/ACM Design Automation Conference*, Jun. 1993, pp. 31-37.
- [28] *Spectre User’s Manual*, Version 4.46, Cadence Design Systems, 2000.