Correct-by-Construction Layout-Centric Retargeting of Large Analog Designs*

Sambuddha Bhattacharya, Nuttorn Jangkrajarng, Roy Hartono and C-J. Richard Shi
Department of Electrical Engineering, University of Washington
Seattle, WA, 98195-2500
{sbb,njangkra,rhartono,cjshi}@ee.washington.edu

ABSTRACT

Aggressive design cycles in the semiconductor industry demand a design-reuse principle for analog circuits. The strong impact of layout intricacies on analog circuit performance necessitates design reuse with special focus on layout aspects. This paper presents a computer-aided design tool and the methodology for a layout-centric reuse of large analog intellectual-property blocks. From an existing layout representation, an analog circuit is retargeted to different processes and performances; the corresponding correct-by-construction layouts are generated automatically and have performances comparable to manually crafted layouts. The tool and the methodology are validated on large analog intellectual-property blocks. While manual re-design and re-layout is known to take weeks to months, our reuse tool-suite achieves comparable performance in hours.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids– *layout*, J.6 [**Computer Applications**] Computer-Aided Engineering – *computer-aided design*.

General Terms: Algorithms, Performance, Design.

Keywords: Analog Integrated Circuit Design, Analog Layout Automation, Layout Symmetry, Analog Synthesis and Optimization.

1. INTRODUCTION

The integration of digital and analog circuits on *system-on-chips* has revolutionized the semiconductor industry and, yet, has given rise to an escalating complexity. Furthermore, driven by the need for superior performance and lower power consumption, the semiconductor industry continues to innovate technologies towards shrinking *transistor feature sizes*. These, together with the added pressure of aggressive design cycles, necessitate the adoption of the design reuse philosophy.

Continued advances in the CAD tools and the cell-based design methodology have largely enabled reuse of digital designs. In analog design, significant trade-offs between the major design goals like gain, bandwidth, stability, noise, linearity and power minimization demand considerable amount of time and effort of the designer.

* This research has been supported in part by the U.S. DARPA's NeoCAD program and in part by the NSF's ITR program.

Fortunately, significant progress has been made recently in the form of optimization tools [1][2] that automatically synthesize analog circuits meeting desired performance specifications. However, the performance of analog designs is not just affected by the device sizes and biasing, but also by the layout styles and intricacies.

Process and temperature gradients introduce mismatches in transistors that are designed to behave identically [3]. Such mismatches drastically affect analog circuit performance leading to DC offsets, finite even-order distortion and lower common-mode rejection [4]. Symmetric layout of matched transistors alleviates the effect of mismatch in analog circuits. Symmetry along with floorplanning, placement and parasitics are of immense importance in analog layout due to their strong impact on design performance. Often, layout designers use their years of accumulated expertise to "squeeze-in" the desired analog circuit performance by careful manual crafting of layouts.

These complexities pose a huge challenge to the automation of analog layouts[4][5]. Over the years, macro-cell based automated placement and routing methodologies have been proposed for analog circuits [6][7]. Unfortunately, these schemes, despite their generality, fail to incorporate the expertise of the layout designer and are seldom accepted in the industry.

Clearly, a layout automation technique that reuses the designer's knowledge embedded in existing layouts promises to be a viable alternative, especially for retargeting layouts to different specifications and technologies. Recently, an analog layout retargeting methodology is proposed in [8] where an already fine-tuned layout is used to create a symbolic structural template incorporating floorplan, symmetry and device/wiring alignment information. This structural template is then used to automatically generate a new layout.

Unfortunately, the above scheme has several shortcomings. Firstly, the layout symmetry for matched transistors is manually imposed on the template and can get increasingly prohibitive as the circuit size increases. Secondly, the methodology does not provide a smooth integration between the circuit and layout design steps. More importantly, it does not have the ability to handle layouts more complex than operational amplifiers.

In this paper, we present a layout-retargeting tool capable of automating layouts of large analog intellectual-property (IP) blocks. We introduce several techniques in the IPRAIL (Intellectual Property Reuse-based Analog IC Layout) framework [8] that allow it to handle layouts of large analog circuits. Firstly, 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 ways so as to ensure similar effects of process and temperature gradients on all subcircuits that are identical by design. IPRAIL addresses these issues with a novel multi-level templating scheme that reduces the template size. Secondly, large analog IP blocks almost always contain on-chip resistors and capacitors. Such passive devices are very sensitive and need to be laid out carefully to minimize the parasitic and coupling effects. Furthermore, passive devices identical by design are also laid out symmetrically. Layout retargeting has to be performed while maintaining such restrictions. Thirdly, user intervention at any step in the template creation or refinement restricts the usability of retargeting tools to smaller layouts. The tool presented in this paper achieves complete automation of the retargeting flow.

Given the strong dependence of circuit performance on layout, we present a scheme for design reuse with IPRAIL at its core. IPRAIL is combined with a commercial circuit optimization engine to formulate a viable correct-by-construction layout-centric design reuse methodology for analog circuits.

This paper is organized as follows. Section 2 discusses the layout-centric design reuse methodology. Section 3 describes the structure and flow of the IPRAIL tool-suite. Sections 4, 5, and 6 present the techniques for template size reduction. Section 7 discusses passive device retargeting. Section 8 presents the experimental results. Section 9 concludes the paper.

2. LAYOUT-CENTRIC DESIGN REUSE

The layout-centric design reuse methodology proposed in this paper is illustrated in Fig. 1. An existing high quality manually crafted analog layout is used as the starting point in the redesign of the circuit that targets a different specification and/or a different technology. This layout is read into IPRAIL that is at the core of this methodology.

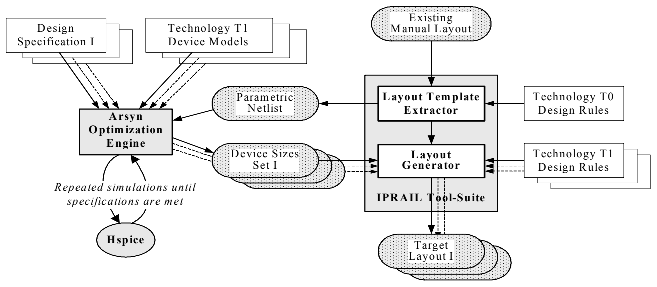
Design Specification I
Technology T1 Device Models
Existing Manual Layout
Parametric Netlist
Layout Template Extractor
Technology T0 Design Rules
Arsyn Optimization Engine
Device Sizes Set I
Layout Generator
Technology T1 Design Rules
Repeated simulations until specifications are met
IPRAIL Tool-Suite
Hspice
Target Layout I

Fig. 1: Reuse based design methodology for analog circuits.

The IPRAIL tool-suite comprises of a *layout template extractor* and a *layout generator*. The Layout template extractor creates a resizable symbolic template and extracts a parametric netlist. The parametric netlist, target design specifications and technology specific simulation models are fed into *Arsyn* [2], a commercially available simulation based circuit optimizer. Arsyn automatically synthesizes the analog circuit meeting the required design specifications. The device sizes generated by Arsyn are fed to the layout generator. The layout generator imposes the device sizes on the symbolic template and automatically constructs a new layout according to the target specifications and/or technology.

As the template extractor detects matched devices in the layout, this can be passed directly as constraints to Arsyn through the parametric netlist. The template also retains the mappings of the devices in the parametric netlist with the corresponding layout abstractions. This facilitates automated imposition of the device sizes obtained from Arsyn on the template prior to layout generation. Thus, the entire flow of data between the two tools requires minimal intervention from the designer.

As the intricacies in the existing design layout is automatically retained in the symbolic template, the layouts generated for different target specifications are automatically customized to the specific needs of the circuit under consideration. As shown in Fig. 1, multiple designs corresponding to different target specifications and technologies can be synthesized and laid out automatically. This reuse of the circuit and layout topologies by the automation tools significantly reduces the design cycle while involving very minimal designer intervention.

3. IPRAIL TOOL-SUITE

3.1 Symbolic Template Extractor

The template extractor creates the symbolic structural template based on the input layout, from which the layout topology, connectivity, and matching are acquired. These intellectual properties are maintained in the template and, along with the new devices sizes obtained from the optimizer, are later utilized as the basis for generating the retargeted layout. The preservation of the layout properties is achieved using sets of various constraint equations that arise from the physical form of the input layout.

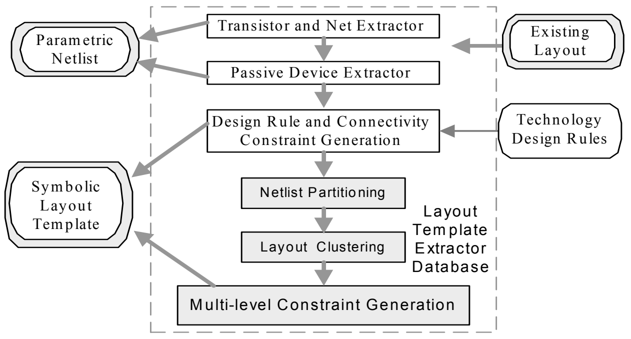
Parametric Netlist
Transistor and Net Extractor
Existing Layout
Passive Device Extractor
Design Rule and Connectivity Constraint Generation
Technology Design Rules
Symbolic Layout Template
Netlist Partitioning
Layout Clustering
Layout Template Extractor Database
Multi-level Constraint Generation

Fig. 2: Layout Symbolic Template Extractor flow.

The detailed tasks of the template extractor are depicted in Fig. 2. First, transistors, nets, and passive devices are extracted from the layout, and compiled to generate a parametric netlist for Arsyn. Next, the scan line [9] method is employed to establish constraints for the symbolic template based on connectivity, topology, and design rules.

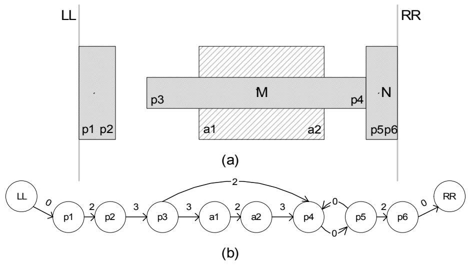

Fig. 3: A horizontal constraint graph as a symbolic template.

To enhance the computational speed of the ensuing layout generation process, the constraints equations, wherever possible, are converted into a constraint graph [9]. Each rectangle in the layout is transformed into four independent nodes, representing its left, right, top, and bottom edges. Constraints are placed between nodes in the graph to sustain layout integrity and correctness. Different constraint categories are connectivity, design rule, exact device size, and symmetry. Horizontal and vertical constraint graphs are constructed independently.

Consider the simple layout of Fig. 3, 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*.

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 Fig. 4, this implies equi-sized channel, drain and source regions, identical orientation and close proximity of the two transistors. Identical channel, drain and source regions are implicitly enforced by the device sizes obtained from Arsyn. Mirroring and location are enforced by the following equations.

$$(e_{bottom} - f_{bottom}) = 0 \quad (1)$$
$$(s_0 - g_{right}) - (h_{left} - s_0) = 0 \quad (2)$$

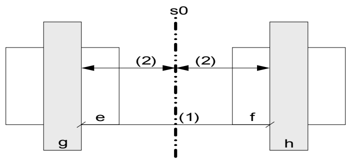

Fig. 4: Simplified layout of two transistors symmetric about axis 's_0'.

3.2 Layout Generator

The algorithm for generation of a new layout from the template is based on symbolic compaction [10]. This is accomplished in two steps: first, the new device sizes are imposed on the template. Then this updated template is solved by a combination of linear programming (LP) [11] and graph based longest-path algorithm [12]. The detailed steps for layout generation are shown in Fig. 5.

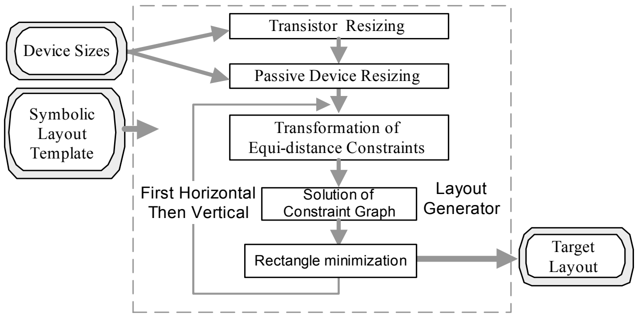

Fig. 5: The layout generator flow.

The exact device sizes obtained from Arsyn are imposed on the template by an additional pair of constraint arcs with equal and opposite weight are added in opposite directions. The template mentioned so far consists of a constraint graph and three variable equi-distance constraints of Eq. (2) imposed due to layout symmetry. In order to solve this modified compaction problem in its graph form (due to superior speed compared to LP), the symmetry constraints need to be first transformed before they can be imposed on the constraint graph. This necessitates multiple longest-path based transformations of the constraint graph into a smaller graph and subsequent LP runs [10]. Finally, the equi-distance constraints of Eq. (2) are transformed to the form

$$(s_0 - g_{right}) = (h_{left} - s_0) = b \quad (3)$$

where *b* is a constant obtained from LP. This is then imposed on the constraint graph and the entire problem is solved with the longest-path algorithm first in the horizontal direction and then in the vertical direction. Finally, as the longest-path algorithm results in some unwanted extension of rectangles, the rectangle minimization algorithm [13] is applied to obtain the final target layout. The graph transformation process for symmetry constraints is computationally intensive.

Therefore, reduction of such constraints is a one of the motivations of our approach.

4. LAYOUT-NETLIST MAPPING FLOW

The primary challenge in retargeting large analog circuits lies in creating a template consisting of minimum number of constraints. In addition to matched transistors, large analog circuits contain entire sub-blocks that are identical by design and demand special attention during layout. Consider a 2-bit comparator circuit that is composed of 4 unit comparators. Gradients in the process parameters across the entire layout introduce differences between the unit comparators that result in non-linearities. This is alleviated by laying out the unit comparators in a common-centroid fashion as illustrated in Fig. 6, where the unit comparator denoted by *A* is split across the ends into two parts A_1 and A_2.

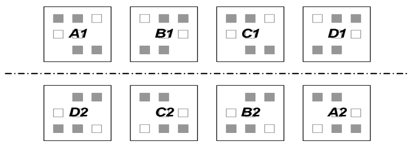

Fig. 6: Four analog subcircuits in two-dimensional symmetric layout.

In these cases, the layout comprises of several identical sub-blocks that are flipped or translated with respect to each other. A naive direct constraint generation for detecting all symmetric or translated devices and nets [14] leads to a tremendous increase in template size for large circuits. This paper addresses this difficulty in constraint generation by extensive partitioning of and mapping between the netlist and layout abstractions.

The layout-netlist mapping flow is shown in Fig. 7. The netlist extracted from the layout consists of *unit transistors*, i.e., transistors with only one rectangle each for its gate, drain and source nodes. These unit transistors are clustered into groups and a compact netlist is obtained by *proximity-based netlist clustering*.

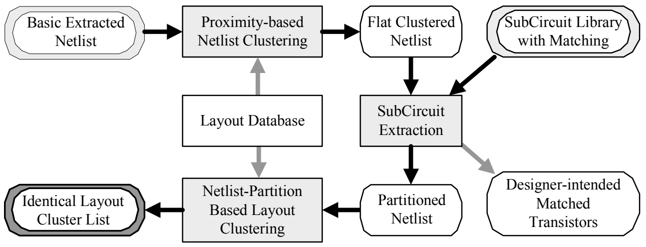
Basic Extracted Netlist
Proximity-based Netlist Clustering
Flat Clustered Netlist
SubCircuit Library with Matching
Layout Database
SubCircuit Extraction
Designer-intended Matched Transistors
Identical Layout Cluster List
Netlist-Partition Based Layout Clustering
Partitioned Netlist

Fig. 7: Multi-level Layout-Netlist mapping flow.

Often, analog design environments consist of a library of commonly used subcircuit topologies, such as differential pairs, current mirrors, comparators etc., that are extensively used in large designs. *Subcircuit extraction* identifies all instances of the subcircuits from the library in the clustered netlist.

The subcircuit mapping step creates a fully partitioned netlist. Based on this partitioned netlist, the corresponding clusters of rectangles in the layout are identified, thus resulting in a partitioned layout. Furthermore, since the library subcircuits contain information about designer-intended matched transistor pairs, subcircuit mapping also produces a full list of essential matched transistor pairs in the netlist [15].

The netlist and layout partitioning process also establishes mapping at different levels between the layout and the netlist. Actual constraint generation is then triggered from the list of matched transistors and the lists of layout clusters. For large analog circuits, such mapping and partitioning is essential to reduce the size of template to manageable levels.

The steps in the multi-level layout-netlist mapping flow are elaborated in Section 5. The constraint generation process is described in Section 6.

5. NETLIST AND LAYOUT PARTITIONING

5.1 Proximity Based Netlist Clustering

In the layout, each multi-fingered transistor M contains multiple contiguous elements C, where each contiguous element consists of physically contiguous unit transistors T. Fig. 8 shows two multi-fingered transistors in a *common-centroid* layout. Here, each multi-fingered transistor has two contiguous sets of three unit transistors each. The clustering scheme partitions the netlist based on the manner in which the transistors are laid out [15].

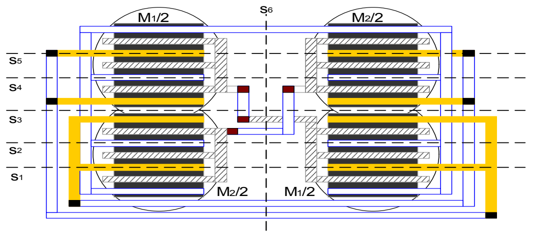

Fig. 8: A common-centroid layout of a symmetric transistor pair. Rectangles with crossed pattern represent the polysilicon layer.

The netlist, which at the end of extraction, comprised of the set of unit transistors T^S and the set of nets N^S, now consists of the same set of nets N^S and the set of multi-fingered transistors M^S defined as $\{M \mid \forall M, \exists$ a unique $\{G_M,S_M,D_M\} \subset N^S\}$ where $\{G_M,S_M,D_M\}$ is the set of the gate, source and drain nets of the multi-fingered transistor M. Each multi-fingered transistor M is a set of physically contiguous elements C^S i.e., $C \in M$. And each contiguous element is defined as $C = \{T \mid T \in T^S, \forall T\ \{G_T,S_T,D_T\} = \{G_M,S_M,D_M\}$, and $\forall T \in C$ are physically contiguous$\}$.

5.2 Subcircuit Extraction

A subgraph isomorphism algorithm [16] is adopted for subcircuit extraction from the clustered netlist. The extraction results in the netlist partitioning essential for layout clustering and also generates a list of designer-intended matched transistors [15].

First, an iterative labeling algorithm is used to partition both the subcircuit and the main circuit. This identifies a set of nodes in the main circuit and a single node, called a *key* node, in the subcircuit. The set of nodes in the main circuit obtained by this iterative labeling algorithm are potential start-points for checking a pattern match with the subcircuit. From each potential node in the main circuit and the key node in the subcircuit, another labeling algorithm detects isomorphism with the subcircuit graph.

5.3 Netlist-Partition Based Layout Clustering

After the subcircuit extraction creates several netlist partitions each corresponding to a subcircuit instance, regions in the layout belonging to the same netlist partition are clustered together. The algorithm for layout clustering is shown in Table 1.

The algorithm starts from a seed device, D_s, which belongs to a netlist partition N_s. First, a layout cluster, L_c, is created containing only D_s. All devices that are proximal to D_s in the layout are collected in a queue Q_s. This is accomplished by a scan line [9] based procedure *ProximalDevices*. From any device, four scan lines look for other devices in close proximity to its left, right, top and bottom edges. If a device D_p in Q_s, is in the netlist partition N_s, then D_p is added to the same layout cluster L_c that D_s belongs to. If D_p does not belong to N_s, the algorithm recursively calls itself to start a new layout cluster. The algorithm terminates when all devices in the layout have been grouped into layout clusters. In practice, a multi-fingered 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 not comprise all contiguous elements of a multi-fingered transistor.

Table 1: Layout clustering algorithm.

```
CreateLayoutClusters ($D_s$)
begin
  if ($D_s$ already assigned to a cluster)
    return
  endif
  $L_c$ = CreateCluster ($D_s$)
  $Q_s$ = ProximalDevices ($D_s$)          // Scan-line based routine
  foreach $D_p$ in $Q_s$
    If ($D_p$.partition == $D_s$.partition)
      AddCluster ($L_c$ , $D_p$)          // Add $D_p$ to $L_c$
    else
      CreateLayoutClusters ($D_p$)      // Recursively call with seed $D_p$
    endif
  end for
end
```

5.4 Detection of Identical Layout Clusters

The previous steps mark each device (or contiguous element of multi-fingered transistor) in the layout with its netlist level cluster, subcircuits and layout-cluster information. The layout clusters A and B are identical only when their devices are one-to-one matched in terms of device sizes and device locations. Consider two layout clusters, A and B, located on the same abscissa. First, all rectangles in each cluster are collected in heaps H_A and H_B. The rectangles are then sorted in an increasing order with respect to the coordinates of the leftmost corner. The two heaps are pair-wise compared [14]. If they are identical, A and B represent *translate-matched* clusters. In case they are not, another heap H_{B2} is created for cluster B where the rectangles are sorted in a decreasing order of their rightmost corner. If the two heaps H_A and H_{B2} are found to be identical upon pair-wise comparison, then they represent *flip-matched* clusters.

6 MULTI-LEVEL CONSTRAINT GENERATION

6.1 Intra-Cluster Symmetry Constraints

From the partitioned netlist generated through the subcircuit extraction process, a list of designer-intended matched devices is obtained [15]. The layout rectangles of such matched multi-fingered transistors within a layout-cluster are collected in sorted lists. For the common-centroid topology in Fig. 8, the six unit transistors on left are collected into a list L_L, and right into a list L_R. The unit transistors in L_L and L_R are then pair-wise compared to detect the vertical axis of symmetry, s_6, and generate the corresponding constraints. For the horizontal symmetry axis s_3, the bottom halves of both M_1 and M_2 are collected in a list L_B, and the top halves are collected in a list L_T and pair-wise compared.

6.2 Inter-Cluster Constraints

Identical and symmetric clusters are laid out either mirror-wise flipped or translated with respect to each other horizontally or vertically. Fig. 9 shows two translated and flipped identical clusters. To ensure matching at the cluster-level, in addition to the intra-cluster constraints, a minimal number of constraints are imposed between the two clusters. First, two devices that represent the same subcircuit elements on different layout-clusters are chosen as the reference devices of the respective clusters. By imposing constraints between two such reference devices, inter-cluster translation or flipped matching is ensured.

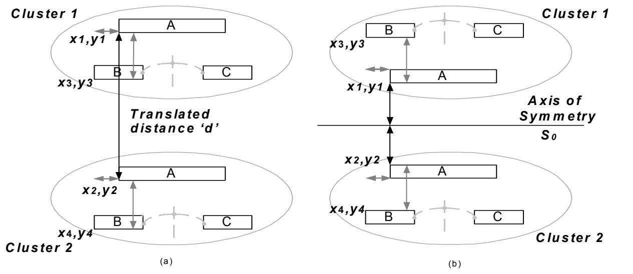

Fig. 9: Inter-cluster matching. (a) Translated matched clusters. (b) Flipped matched clusters. Dotted lines show intra-cluster symmetry.

For the translated clusters in Fig. 9a, the following two equations between two cluster's reference devices *1A* and *2A* are necessary for cluster matching.

$$y_1 - y_2 \geq d \tag{5}$$

$$x_1 = x_2 \tag{6}$$

where *d* is the distance between two reference devices and is determined by coupling or design rule constraints. In addition, an equation relating each symmetric group inside each cluster with the reference device of the cluster is included to ensure cluster matching. The following equations relate devices *1B* and *2B* with the reference devices *1A* and *2A*.

$$x_1 - x_3 = x_2 - x_4 \tag{7}$$

$$y_1 - y_3 = y_2 - y_4 \tag{8}$$

Similarly, for the flipped clusters in Fig. 9b, Eq. (5) is replaced with the following equation

$$y_1 - s_0 = s_0 - y_2 \tag{9}$$

where s_0 is the symmetry axis between two devices. The other constraints are similar to the translated case.

Secured by both 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 worst-case of *(nc)(nc-1)/2* to *(nc-1)*. 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 worst-case of *(nc)(nc-1)/2* to $n^{1/2}c(c-1)$, or by a factor of $n^{3/2}/2$, when *c >> 1*.

7. PASSIVE DEVICE RETARGETING

Layouts of large analog circuits consist of multiple on-chip passive devices that are usually isolated in space from other devices and nets to minimize the parasitic and coupling effects [4]. Passive devices like resistors and capacitors are automatically detected by a traversal through the nets in the layout when the calculated value of the passive exceeds a threshold. Connectivity between a passive device and the nets at its ends are maintained on *port* rectangles. To prevent overlaps or close proximity to other devices upon retargeting, a *shadow rectangle* is placed on top of the passive devices prior to the symbolic template generation, as shown in shaded area in Fig. 10. A shadow rectangle 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 constraints come from one of the following: coupling constraints, specialized design rules for passives as seen in certain technologies, or designer's input. Such constraints have the form:

$$X_{shadow} - x_{other} \geq d \tag{10}$$

Constraints are added between the shadow rectangle, the ports and the device to maintain connectivity upon retargeting.

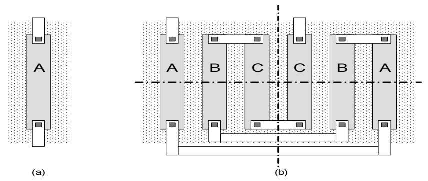

Fig. 10: (a) A unit resistor (b) One-dimensional interdigitated symmetric layout of resistors. In both cases, the shaded background represents the *shadow rectangle* used in retargeting.

Frequently, multiple passive devices are designed to be identical to each other. Fig. 10b shows three resistors laid out in an interdigitated fashion with one-dimensional common-centroid symmetry [5]. This layout ensures matched resistance between resistors *A*, *B*, and *C*, regardless of process gradients. The template extractor automatically detects such symmetry in passive devices and imposes appropriate symmetry constraints in a manner similar to intra-cluster symmetry for transistors. Prior to the layout generation step, new passive device sizes obtained from Arsyn are imposed as fixed width constraints. The shadow rectangle is appropriately extended and the entire set of constraints passed on to the layout generation engine.

8. RESULTS

We applied IPRAIL and our layout-centric design methodology to the design and layout of a *5-bit flash analog-to-digital converter* (ADC). The ADC comprised of a resistor chain, an analog section, and a digital block. The analog section consisted of 32 latched comparators. Each comparator, whose schematic is shown in Fig. 11, compares the input signal with different reference voltage levels obtained from resistor chain acting as a voltage divider. Based on the input voltage, the analog section produces a *thermometer code*. This is then converted to 5-bit binary output by the 32-to-5 decoder.

The analog section of the ADC was initially designed and laid out manually in 0.25um TSMC technology process, while the digital section was designed in a standard-cell based ASIC flow. To minimize 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 polysilicon layer and laid out in a fashion similar to Fig. 10b.

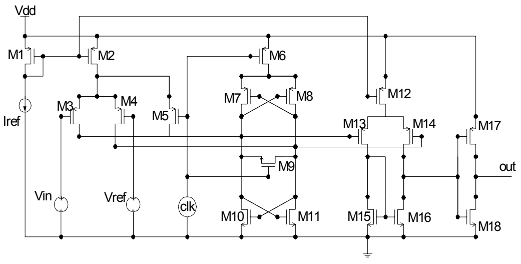

Fig. 11: Comparator schematic. M_3 and M_4 form the input diff-pair.

This ADC design in TMSC 0.25um technology process is then retargeted to TSMC 0.18um technology process with different specifications. The structural symbolic template for the layout was constructed by incorporating the scan line method and inter/intra cluster symmetry. 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 192 intra-cluster symmetric transistors, 31 inter-cluster symmetries, and 63 resistor symmetries. In comparison, a direct extraction [14] results in 43,935 symmetry related constraint arcs.

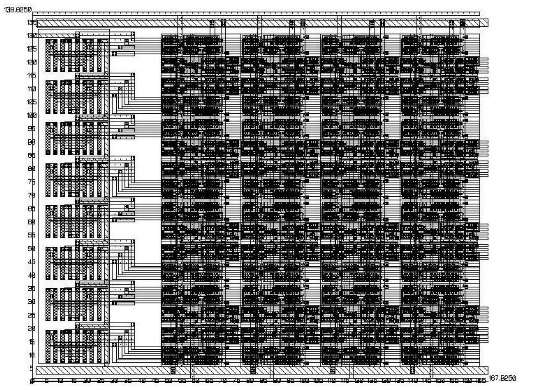
Fig. 12: Analog section of a 5-bit ADC in TSMC 0.25um technology.

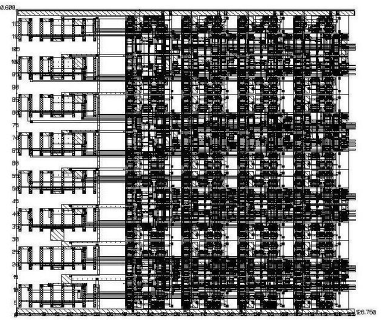
Fig. 13: Analog section of a retargeted ADC in TSMC 0.18um.

Table 2: ADC post layout specifications in 0.25um and 0.18um TSMC.

Performance Parameters	Original (0.25um)	Target (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*
Diff. Nonlinearity (max)	*0.12 LSB*	*0.11 LSB*
Int. Nonlinearity (max)	*0.66 LSB*	*0.39 LSB*
Power Consumption	*42 mW*	*18 mW*
Total Area	*79,800 um^2*	*36,650 um^2*

The extracted parametric netlist was passed to Arsyn and the circuit synthesized in TSMC 0.18um to achieve the desired specifications. The new device sizes obtained from Arsyn were enforced on the template. The target layout was obtained by solving the template by a combination of LP and graph based longest-path algorithm. The analog section of the original layout is shown in Fig. 12 and the target layout is in Fig. 13.

The specifications achieved in the post layout simulation for both designs (0.18um and 0.25um) are listed in Table 2. IPRAIL was run on a 900MHz SUN UltraSparc3 workstation. The template extraction phase took 8 hours 52 minutes, and the generation of the target layout was completed in 1 hour 51 minutes.

9. CONCLUSIONS

A symbolic structural template based layout-retargeting tool capable of handling the complexities associated with large analog designs is presented. The fully automated IPRAIL tool combines several key techniques like netlist partitioning, layout clustering, automatic symmetry detection, and passive device matching in a novel way as a part of the template extraction. Multiple high quality layouts can be automatically generated from the symbolic template. As the target layouts are generated from a high quality manual layout, they are correct by construction. More importantly, the target layouts retain all complex layout techniques employed to alleviate the process gradients and other factors that affect circuit performance. This layout-retargeting tool is successfully combined with a commercial circuit synthesis tool in a layout-centric design reuse methodology. The methodology is shown to be effective in retargeting existing large analog designs to multiple specifications across different technologies. Large analog circuits that are known to take several weeks to months to synthesize and lay out are automatically created within hours with comparable performance.

REFERENCES

[1] 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. Int. Conference Computer-Aided-Design*, Nov. 2001, pp. 350-357.

[2] Arsyn User’s Manual, Orora Design Technologies Inc., 2003.

[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, pp. 1433-1440, Oct. 1989.

[4] B. Razavi, *Design of Analog CMOS Integrated Circuits*, McGraw Hill, 2001.

[5] A. Hastings, *The Art of Analog Layout,* Prentice Hall, 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, 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, pp. 773-780, Jul. 1995.

[8] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and R. Shi, “IPRAIL – Intellectual property reuse-based analog IC layout automation”, *Integration – The VLSI Journal*, vol. 36, pp. 237-262, Nov. 2003.

[9] 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.

[10] R. Okuda, T. Sato, H. Onodera and K. Tamaru, “An efficient algorithm for layout compaction problem with symmetry constraints”, *Proc. Int. Conference Computer-Aided-Design*, Nov. 1989, pp. 148-151.

[11] D. Luenberger, *Linear and Nonlinear Programming* 2nd Edition, Addison-Wesley, 1984.

[12] T. H. Cormen, C. E., Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 2001.

[13] G. Lakhani and R. Varadarajan, “A wire-length minimization algorithm for circuit layout compaction”, *Proc. Int. Symposium on Circuits and Systems*, May 1987, pp. 276-279.

[14] 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.

[15] S. Bhattacharya, N. Jangkrajarng, 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.

[16] 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.