Hierarchical Analog Layout Migration with Pcells

João Serras
INESC ID
IST - Tech. Univ. of Lisbon
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
email: joao.serras@gmail.com

L. Miguel Silveira
INESC ID / Cadence Research Laboratories
IST - Tech. Univ. of Lisbon
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
email: lms@inesc-id.pt

***Abstract*—Despite all the research efforts, some areas of IC design are still mainly manual, such as analog layouts. With the proliferation of systems on a chip (SoCs), the lack of efficient methods to generate analog layouts is preventing faster on-chip integration of digital and analog functionality. In this paper we propose a method that allows an efficient reuse of analog layouts between different fabrication processes. The main assumption of this work is that when designing a circuit in a new flavor or technology node, expert analog designers start by reusing a topology from an already silicon-proven circuit. If the fabrication processes are similar, then only minor adjustments in the device sizes may be needed to achieve the required performance specifications. However, different design rules might cause several small adjustments to be made. Moreover, if the source layout contains pcells, adjustments might also be needed due to small differences in the pcell sizes available in the target process. The proposed method automates these adjustments while preserving the hierarchy of the original database and re-establishes the connectivity to the target pcells, subject to the design rules of the target technology. All this is performed while maintaining the original layout topology so that the reliability of the source layout is kept in the target. An implementation of the method is discussed and results show that the connectivity is re-established in the target layouts when moving from a 0.13um node to a 90nm node, a 0.18um node and to a different 0.13um technology.**

I. Introduction

Advances in IC manufacturing technology in the last decades have enabled the realization of hundreds of millions of transistors in an area as small as 1cm^2. This steady progress created enormous opportunities and markets for consumer electronics. Cellphones, laptops, pocket-sized music players and game consoles are some of the devices that are enabled or continuously upgraded by this progress. While most of the processing in such equipments is digital, interfacing with the real world requires analog circuits for data acquisition, signal amplification or power management. For reasons of cost and performance, the trend has been to integrate these systems in a single silicon die, the SoC, Digital parts of SoCs are created in a systematic way with the help of mature tools for synthesis and physical design. In spite of the increasing complexity derived from the larger number of devices and from the non-idealities of deep sub-micron processes, the problem has been handled successfully by successive generations of these tools. In contrast, analog parts are still mainly manually crafted and require time consuming design cycles. Although progress has been made for analog circuit optimization tools, automatic generation of analog layouts is still a very difficult problem to solve and analog circuits are very sensitive to layout changes. Even though the analog portions of SoCs are generally smaller than the digital portions, the lack of a systematic and automated approach to analog design and the increasing complexity of the effects introduced by smaller feature sizes, make analog design a bottleneck in the development of SoCs.

This paper presents a method that enables a more efficient reuse of silicon-proven analog layout databases. This is accomplished by preserving the properties that make the database humanly editable, *i.e.* hierarchy and pcells, and by automatically adjusting the layout to cope with the target technology design rules and fixing connectivity errors introduced by the different sizes of the target pcells. The preservation of database hierarchy, along with the pcells allows manual adjustments in the layout, which although undesirable are in practice unavoidable when switching between technologies. Some adjustments on the device sizes might be needed which are easier to perform if the devices are implemented with pcells Also, if more deep transformations are needed, *e.g.* a new topology in a block, then if the layout is well partitioned (hierarchically), the change can be made in a single place. If performed manually, all these tasks are very time-consuming and error-prone but still a better option than starting a new layout from scratch. The main goal of this work is to automate much of these tasks in order to make this process more efficient. It is assumed here that the layout generated by the proposed method (the *migrated* layout) does not match the required specifications in the target technology, but produces a layout that is a very good first start for the analog layouter. This is accomplished by formulating the problem as a linear optimization problem where the design rules and connectivity are constraints to this problem, such as the topology of the source layout. Solving this problem yields the migrated layout.

II. Background

A. Analog IC Database Structure

Layouts for IC designs are stored in databases. The format of the databases is typically proprietary information of the electronic design automation (EDA) vendors, such as Design Framework II, of Cadence Design Systems, MilkyWay, from Synopsys, or OpenAccess, from the Si2 Consortium. Despite the multitude of EDA vendors and database formats, the most important features available in most are: cells and views, hierarchy and parametrized cells (pcells). At the organizational level, the unitary component of the database is the cell. A cell describes an individual building block of a system such as a NAND gate, a bandgap circuit or an A-D converter. Since different stages of the design process require different types of information, each cell may be further decomposed into views.

Hierarchy is the possibility to use the information of one cell in another. In hierarchical designs one cell may contain instances of other cells. Each cell references a master cell which may contain instances of other cells as well. This organization is needed to efficiently represent repeated structures and to manage large and complex IC components. Analog IC blocks that have some complexity may be aggregated in libraries.

For instance, the cells of a bandgap circuit may be stored in a library and the cells for a phase locked loop (PLL) circuit may be stored in another library. A system may aggregate several of these circuits by instantiating the top level cells of each.

Another important feature implemented in IC databases is the parametrized cell or pcell. A pcell is a cell whose size can change according to user paramaters. For instance, basic devices such as MOSFETs, resistors and capacitors are usually implemented as pcells because designers often need instances of many different sizes. Without pcells, each needed geometry of a device would had to be created and stored in a different cell or all the polygons must had to be drawn manually each time. With pcells, a template for the device exists which can be changed according to the designer needs for a pre-determined set of parameters. This feature is very important for the manual editing of the database.

B. Related Work

Analog layout synthesis is traditionally categorized in one of two approaches [3], [7]: template-driven procedural module generation and macro-cell style placement and routing.

Template-driven methods rely on the existence of a template layout that is updated with the required device sizes and technological constraints to generate a layout. The main advantage of these methods is that when the templates are from silicon-proven layouts, they capture the design expertize for both device placement and routing, which in analog layouts may be particularly difficult to done automatically due to several factors (e.g. parasitics). One example of these is the BALLISTIC language [5], which allows analog circuits to be described with tailor-made programming language constructs. The layout can be specified hierarchically using modules. Each module may use other modules which can be decomposed into finer-grained components such as transistors, resistors and differential pairs. The main drawback however is the encoding of the template in this language, which is a time consuming task and foreign to most designers. The OPASYN framework [4], includes a topology selection step from a set of pre-defined topologies. The optimal sizes of the devices are then selected after an optimization step and a layout generator module outputs the devices that best match the selected topology. After the placement, the routing proceeds iteratively. Another method is retargeting through layout compaction. Here, the idea is to generate the target layout by solving an optimization problem whose constraints are the new technology design rules and the cost function the area of the layout. This method can take into account parasitics, as implemented in IPRAIL [10], where upper bounds on parasitics are added to guide the generation of the retargeted layout. In this work, the template is automatically extracted from an existing layout and new layouts are generated by giving inputs on the new device sizes, technology constraints and parasitic bounds.

Macro-cell style approaches are more similar to the digital CAD methodologies, where the layout solution is determined by solving an optimization problem that minimizes a cost function that typically contains factors such as the total layout area and net length. These methods allow the automatic generation of layouts that can suffer large changes in device sizes and design rules. These layouts are not predefined by a template. Examples of such tools are ILAC [6], the different

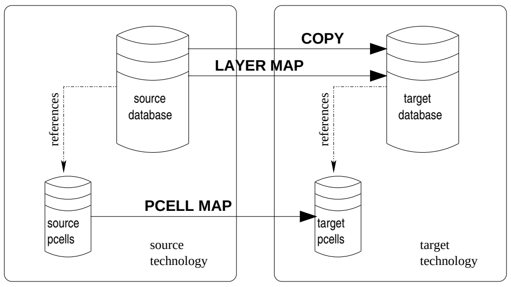


Fig. 1. Layout retarget process

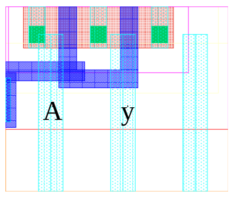
versions of KOAN/ANAGRAM [2] and ALADIN [9]. In these tools, several optimizations and algorithms are used for the placement and routing of the devices: [2] is able to minimize wire parasitics and [8] allows the specification of symmetry constraints for devices and also uses a layout generation tool capable of minimizing the performance degradation induced by unwanted layout effects. In [9], the placer uses a genetic algorithm and in [2] a simulated annealing algorithm. The main advantage of this approach is that it is generally applicable and not specific to a certain circuit or topology. However, there are some disadvantages: the large CPU run time, because these techniques are simulation intensive; the setup effort needed to build a cost function or an evaluation process to favor good layouts; and the non-deterministic nature of some of these algorithms makes the replication of one solution (even with the same constraints and objectives) very difficult. Another generic problem related to all of the above methods is that they typically fail to generate hierarchical and pcell based layouts. Fine-tunings that are relatively easy to do when the layout is hierarchical and pcell based may be impossible to do because the generated layout is flat. Therefore, a trivial task to perform may require another run on the tool after an additional configuration step. The proposed method on this paper avoids this because it specifically generates layouts that are meant to be further edited.

III. Layout Retarget

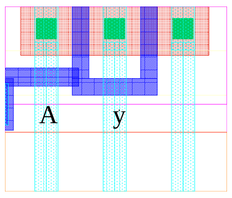
Layout retarget is the process by which a layout database in one technology is made usable in another technology.

A layout database is typically stored in a database system provided by an EDA vendor. As described in Section II-A, the database is typically hierarchical and uses pcells. To retarget a database, three basic steps are needed (Figure 1). The first step of the retarget process is the creation of a copy of the source database. This can be performed by a simple copy command in the underlying operating system (*e.g.* Unix's `cp`) or by database copy operations that the EDA vendors provide that allow the copy of multi-library designs (*e.g.* Cadence's DFII `ccpCopy`). The advantage of the latter method is that it may correct automatically the cross-references of the cell instances in the copied libraries. A second step, Layer map, is the step by which the mask and logic layers of the source technology are mapped to the layers of the target technology. This step is usually needed to map the shapes used for device formation and routing. This can be performed in two ways:

1) Export/import the layout database to an intermediate format (*e.g.* GDSII) and switch the layers in the process by manipulating the stream layer map table used in the export/import process.

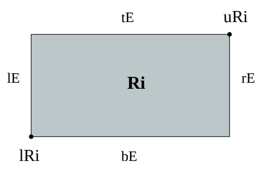


(a)

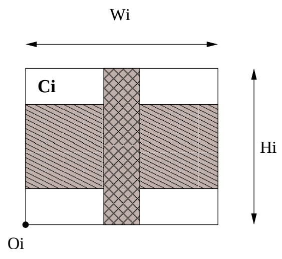


(b)

Fig. 2. (a) Source layout in UMC 0.13um, (b) Retarget layout in UMC 90nm. The connections that were aligned in (a) are out of place because the pcell on the top shrunk.



(a)



(b)

Fig. 3. (a) Rectangle object, (b) Pcell instance object (right)

2) Switch the layer references of the database in the target technology.

This step assumes there is a map between the layers of both technologies. Although this is difficult to achieve if the entire set of layers of both technologies is considered, it usually is possible to map directly the most used layers in a layout: N-well, diffusion, implants, contact, vias and metals. Also, this step is simplified if all the devices (*i.e.* MOSFETs, diodes, resistors, capacitors) and structures (*i.e.* guardring, N-taps, P-taps) that are used are implemented as pcells.

The final step is the translation of the pcells used in the source technology to the pcells available in the target technology. The challenge is to select the pcells of the target database that implement the geometries that are most similar to the ones used in the source technology. This process may be complex because, even if such similar geometries exist, the parameters used in the source pcells need to be mapped to the target pcells. If the pcell libraries are very different, this process may require several configuration steps. This is the most critical step in layout retargeting. If the source and target pcells have different geometries than this process is of little use. If, however, the geometries are similar, than if the pcell dimensions change it is possible to automatically adjust the target layout, as described in Section IV.

One of the most troublesome issues from retargeting is broken layout connectivity. This may occur when the source pcells are replaced with pcells of different geometries or sizes. This happens when the pcell terminals change positions or dimensions the routing to the pcell that was previously established and is implemented with polygons remains in the same place. This breaks the connectivity and originates layout-versus-schematic (LVS) errors (see Figure 2). Also pcells that did not overlap in the source layout may start to overlap and vice-versa. This only depends on the sizes of the target pcells. These situations are undesirable because they change the original layout topology. The unintended overlap may cause bad device formation or recognition or may even cause connectivity errors if two terminals of the pcells that were connected to different nets start overlapping. The unintended separation may create an open circuit.

Finally, Design rule check (DRC) errors also have to be taken into account. Minimum widths, separations, enclosures, extensions, areas and other topological layout design rules are constraints that the foundries impose in order to qualify the layout design for fabrication in a given process. The values for these rules are generally different, even in similar fabrication processes. Therefore, even if it is possible to maintain the same geometry and size of the source pcells, there is no guarantee that the target layout will be DRC clean in the target process.

IV. Layout Migration

Traditional template based methods rely on compaction to guide the retargeting process. In this work, the focus is to obtain a layout that is similar to the original but without the problems described in Section III. Below is the formulation used:

$$\begin{array}{rl} \text{minimize} & <\textit{source topology changes}> \\ \text{subject to} & <\textit{original connectivity}> \\ & <\textit{new topological design rules}> \\ & <\textit{original device relative positions}> \quad . \end{array} \tag{1}$$

The minimization term means that *some* changes are admissible in the source topology to generate a migrated layout with the fixed connectivity, new design rules and original device positions. Next we detail how this is performed for a single layout cell. Section V extends this to hierarchical layouts.

A. Layout objects

It is assumed that layouts are composed by two types of objects: rectangles and pcell instances. The position and size of a rectangle R_i is described by its lower left (xL_i, yB_i) and upper right (xR_i, yT_i) coordinates (Figure 3-a). A pcell instance P_i can be described by its origin (x_i, y_i) and the size of its bounding box with width W_i and height H_i (Figure 3-b) - the first capital letter means a fixed value. The size of a pcell (width and length) cannot be changed since the pcell parameters are set on the retarget phase, however its origin can be moved. Rectangles can shrink or enlarge. By only considering rectangles it is implicitly assumed that 45° shapes are not taken into account. This formulation assumes that all active and passive devices and regular structures (*e.g.* substrate and N-well contacts, vias, guardrings) are implemented with pcells and rectangular shapes are only used for limited purposes such as N-well definition, routing (polysilicon or metal), and device recognition. Additionally, it is assumed that the target pcell library implements pcells with the same geometry of the source pcells, but with different sizes. This means that for each pin in the source pcell there is a unique corresponding pin in the target pcell and vice-versa.

B. Constraints

This section describes the most relevant set of implemented constraints in the LP formulation. The retarget issues identified in Section III are avoided by the connectivity, DRC and pcell relative position described below. Additional topological constraints are also accounted for.

The connectivity constraint guarantees that two shapes that are electrically connected in the source layout remain connected in the target layout. This is accomplished by setting inequality constraints for the shape locations. Design rule

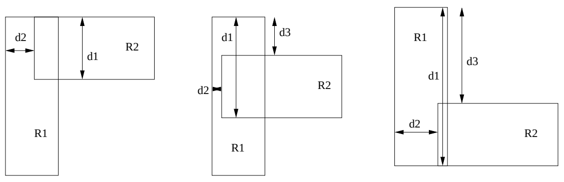


Fig. 4. Possible overlap topologies between two rectangles

constraints avoid the generation of layouts that cannot be fabricated in the target technology. Although the number and complexity of these constraints increases as the feature size diminishes, in the retarget of analog layouts the most common errors are caused by conceptually simple rules such as minimum widths and minimum separations. These are easy to describe for rectangles, for instance:

- minimum width
  $xR_i - xL_i >= \textit{WID}_i$ and $yT_i - yB_i => \textit{WID}_i$
- minimum separation
  $xR_j + \textit{SEP}_{jk} <= xL_k$ (R_j on the left of R_k) or $yT_j + \textit{SEP}_{jk} <= yB_k$ (R_j on the bottom of R_k)

where $\textit{WID}_i$ is the minimum width associated with layer of R_i and $\textit{SEP}_{jk}$ is the minimum separation between the layers of rectangles R_j and R_k.

To keep pcells from overlapping in the target technology, it is necessary to write the following constraints: if pcell P_i is on the left of pcell P_j then $x_i + W_i <= x_j$; if pcells P_i is below pcell P_j then $y_i + H_i <= y_j$. If pcells overlap in the source layout, then to keep this property, constraints similar to the connectivity constraints can be written.

Topological constraints are needed to maintain the overall consistency and aspect of the source layout in the target. Without these constraints, any solution that satisfied the constraints specified above would suffice, but those are not enough to obtain a layout that captures the intention of the source layouter. For instance, rectangle size constraints are used to keep the wells and routing wires with dimensions similar to the original. For instance, to maintain the horizontal width of a wire: $xL_i + \textit{HW}_i = xR_i$, where $\textit{HW}_i$ is the original horizontal width of the wire implemented with rectangle R_i. Rectangle inclusion and exclusion constraints are used to keep all the shapes and pcells inside a well or outside it. The constraints are similar to the connectivity constraints but are not restricted to electrical shape connectivity. When two rectangles overlap to form connectivity, they may overlap in different manners. Figure 4 illustrates this, two rectangles R_i and R_j overlap, but in the left case, the top edges are coincident, in the middle, both the bottom and top edges of R_j are contained inside R_i, in the right case the bottom edges are coincident. The connectivity constraints do not encapsulate this information, however for topology it is relevant to keep how shapes overlap.

C. Piece-wise linear transformation

The constraints described in the previous section may yield an infeasible domain. For instance, the minimum width DRC constraint and the rectangle size constraints form an infeasible set if $\textit{HW}_i < \textit{WID}_i$. Therefore, the constraints cannot be used to form the restrictions of a linear optimization problem. One solution for this is to *penalize* solutions that deviate from the constraint. If a solution for the migration problem is a rectangle with a size smaller or larger than the original value, then that solution can be penalized in favor of solutions that keep the original size. The greater the distance, the larger the penalization. However, if a minimum width DRC constraint imposes a size larger than the original, then the best solution will have some penalty associated, but that will be the minimum possible penalty. This can be implemented by a piece-wise linear function for each constraint. For the simple equality constraint $x_1 + d = x_2$ and function $f_e(\mathbf{x}) = x_2 - x_1 - d$, the constraint is satisfied when $f_e(x) = 0$. The function $f_{pe}(\mathbf{x}) = \max(f_e(\mathbf{x}), -f_e(\mathbf{x}))$ is a piece-wise linear function. Additionally, $f_{pe}(\mathbf{x}) >= 0$. f_{pe} is a penalty function because its minimum occurs when the constraint is satisfied $f_{pe}(\mathbf{x}) = 0 <=> f_e(\mathbf{x}) = 0$. For inequality constraints, considering $f_i(\mathbf{x}) <= 0$ whenever the constraint is satisfied, then a possible penalty function is $f_{pi}(\mathbf{x}) = \max(f_i(\mathbf{x}), 0)$.

D. LP formulation

In summary, the constraints specified in Section IV-B can be wrapped into penalty functions as follows:

$$\begin{aligned} f_{pe1}(\mathbf{x}) &= \max(f_{e1}(\mathbf{x}), -f_{e1}(\mathbf{x})) \\ &\dots \\ f_{pem}(\mathbf{x}) &= \max(f_{em}(\mathbf{x}), -f_{em}(\mathbf{x})) \\ f_{pi1}(\mathbf{x}) &= \max(f_{i1}(\mathbf{x}), 0) \\ &\dots \\ f_{pin}(\mathbf{x}) &= \max(f_{in}(\mathbf{x}), 0) \end{aligned}$$

where m is the number of equality constraints and n the number of inequality constraints. Finding a solution that satisfies the constraints is the same problem as minimizing all the penalties, hence:

$$\text{minimize} \quad f_{pe1}(\mathbf{x}) + \ldots + f_{pem}(\mathbf{x}) + f_{pi1}(\mathbf{x}) + \ldots + f_{pin}(\mathbf{x}).$$

Through the epigraph transformation [1], the minimization of a piece-wise linear function can be further factored. Finally, since some constraints are more important than others it is useful to scale some penalty factors. The greater the scaling is for some constraint, the more importance it will have in relation to the others. This yields the final linear optimization problem when the constraint functions are linear:

$$\begin{aligned} \text{minimize} \quad & \sum_i k_i t_i + \sum_j k_j t_j \\ \text{subject to} \quad & f_{eqi}(\mathbf{x}) \le t_i \\ & -f_{eqi}(\mathbf{x}) \le t_i \\ & f_{iqj}(\mathbf{x}) \le t_j \\ & t_j \ge 0 \\ & k_i, k_j > 0 \end{aligned} \tag{2}$$

where $1 \le i \le m$ and $1 \le j \le n$. k_i and k_j are the scaling factors. The LP specified in (1) can be implemented by the LP in (2), by choosing appropriate scaling factors for each penalty function.

V. Hierarchical Layout Migration

As referred in Section II an hierarchical design contains at least one layout cell that instantiates one or more different layout cells. To ensure a successfull migration, the layout cells cannot be migrated independently from each other. For instance, if a layout cell **B** instantiates a layout cell **A**, then

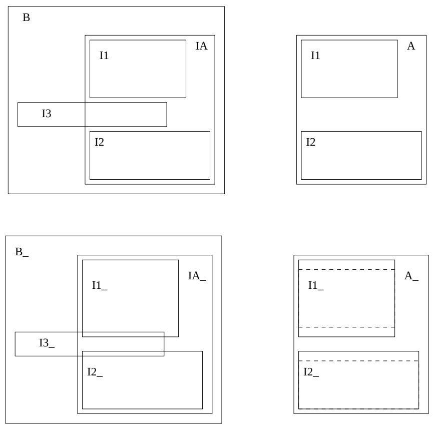


Fig. 5. Cellviews $\mathbf{A}$ and $\mathbf{B}$ in the source (top) and migrated (bottom) technologies. After migration, $I3^{'}$ doesn't have space to overlap $I^{'}_{A}$ between $I1^{'}$ and $I2^{'}$, as in the source technology. The original layout topology is infeasible in the target technology.

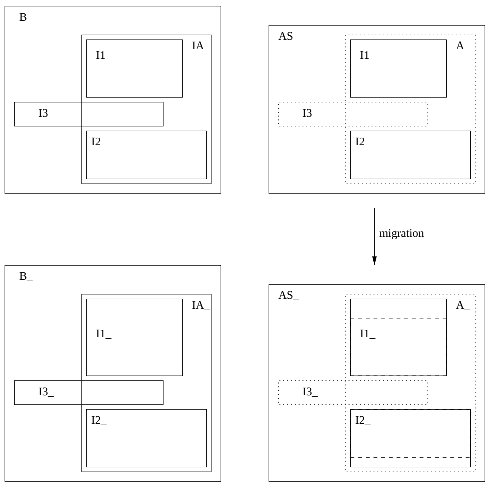


Fig. 6. Hierarchical migration with stamping procedure

it is natural that $\mathbf{A}$ should be migrated before $\mathbf{B}$, because $\mathbf{A}$ is contained in $\mathbf{B}$ but $\mathbf{B}$ is not in $\mathbf{A}$. When the cells overlap, migrating by the instantiation order might not be enough. As shown in Figure 5 if the cells in the lower levels of hierarchy are migrated without taking into account where and how they are instanced (*e.g.* how they are overlapped), then the migration of higher hierarchy cells may become too constrained or infeasible. For instance, in the example of Figure 5, instances $I1^{'}$, $I2^{'}$ and $I3^{'}$ must overlap, however they should not - this might create a violation on the original devices relative position, for instance. One solution for this is to incorporate the objects from other overlapping cells in the migration of lower level cells, as shown in Figure 6. A stamp cellview ($\mathbf{A}_S$) is created to contain all the objects from original cellview ($\mathbf{A}$) plus the overlapping objects ($I3$ from $\mathbf{B}$). This cellview is then migrated ($\mathbf{A}^{'}_S$) and it satisfies the constraints imposed by all the overlapping objects ($I1^{'}$, $I2^{'}$ and $I3^{'}$). A stripped version of the migrated stamp cellview is created ($\mathbf{A}^{'}$), which contains only the new positions of the original objects ($I1^{'}$ and $I2^{'}$). Instances of the cellview ($I^{'}_{A}$) are then used in the migration of other higher hierarchy cells ($\mathbf{B}^{'}$). In this manner, the formulation of Section IV can be extended for hierarchical designs by migrating the cells in the lower levels of the hierarchy first, but by taking into account its context of instantiation in the higher hierarchy cells.

VI. Results

This section presents the obtained results on an Intel dual-core 1.83GHz processor with 1Gb RAM. The databases were made with Cadence version 5.10.41_USR5.90.69 (32-bit). The LP solver used was `lp_solve` running on MATLAB version R2007a. The DRC/LVS verifications were performed with Calibre version v2007.1_24.22.

Table I shows the used benchmarks. It lists the number of instances (#Insts), the number of shapes (rectangles, paths, etc) in the layout cells (#Shapes), and the size of the linear program to be solved: number of inequality constraints (#Ineqs), number of equalities (#Eqs), number of lower and upper bound restrictions (#Lowers and #Uppers, respectively) and total number of variables (#Vars). The cells that have a leading $\uparrow$ belong to the hierarchy of nearest cell on the top. The source layout circuits are all from UMC 0.13μm.

In the implementation of the constraints, the priority was given to the restoration of the connectivity, since these cause LVS errors which are typically more difficult to correct than DRC errors. The DRC implemented DRC constraints were the minimum width and minimum separation for all interconnect layers.

To measure the obtained results three types of indicators were used: the number of DRC errors, the number of LVS errors and the total simulation time. The number of DRC and LVS errors are used to measure the improvement of the migration method in relation to a layout retarget. The simulation time gives a notion about the needed time to obtain the results. The results can be seen in Table II (UMC 90nm) and Table III (TSMC 0.18μm). The migration process generates layouts that are LVS clean and reduces the number of DRC errors in relation to the retarget process. The simulation time is divided into the time needed to generate the constraints (Cadence CPU run time) and the time needed to solve the LP (MATLAB CPU run time). In most of the benchmarks, the majority of the time was spent in the generation of the constraints. Some DRC errors persist in the migrated layouts because the target pcell sizes or design rules might not allow the original topology.

VII. Conclusions

This work presented a method that allows the generation of layouts that are better than the layouts generated through a retarget. The results clearly show that the connectivity of the source layout can be restored, even when migrating to different feature sizes and technologies. Moreover, the migrated layouts follow the topology of the source layouts, contain pcells, are hierarchical and thus can be edited and fine-tuned by layout designers as easily as the original layouts allowed it. The results showed that most of the time was spent on the generation of constraints. For future work, the generation of these can be optimized. Also, the generated LPs were fairly regular and the LP solver used was generic. Therefore, the solution of the LPs can be optimized by developing a custom LP solver.

TABLE I
BENCHMARKS DESCRIPTION AND LP DATA

Name	Description	#Insts	#Shapes	#Ineqs	#Eqs	#Lowers	#Uppers	#Vars
INV_8_3S	Inverter gate	17	53	1983	50	592	30	1202
NAND2_8	2-in NAND gate	39	31	2404	24	700	74	1503
XOR2_2	2-in XOR gate	25	51	2471	49	735	40	1494
refiref	Current reference	428	161	42785	110	6612	872	23408
LV2HV_60	Level shifter	42	29	2235	32	774	79	1470
$\uparrow$ INV_4	Inverter gate	16	33	1692	14	462	28	1019
SMTG_1	Schmitt trigger	20	34	1725	33	520	30	1061
$\uparrow$ NAND2_4	2-in NAND gate	45	83	4109	53	1377	74	2738
$\uparrow$ INV_1	Inverter gate	19	48	1945	21	692	26	1336
adctopmirror	Current mirror	104	108	10431	107	2810	174	6109
$\uparrow$ libinv	Inverter gate	14	17	854	17	262	24	535
lvs_refvref	Voltage reference	1	5	0	10	12	0	12
$\uparrow$ refgen	Reference generator	391	118	45571	110	8167	733	26831
$\uparrow$ ESDunit	ESD protection unit	32	8	2893	8	1162	77	2062

TABLE II
RESULTS FOR UMC 90NM

	Retarget		Migration		MATLAB CPU	Cadence CPU	Total elapsed
Name	#DRC	#LVS	#DRC	#LVS	run time (s)	run time (s)	time (s)
INV_8_3S	21	61	2	0	2.28	42.95	50.0
NAND2_8	25	72	3	0	3.0	40.82	48.0
XOR2_2	22	51	1	0	2.69	17.23	24.0
refiref	49	113	7	0	848.76	3158.22	4223.0
LV2HV_60	18	83	0	0	2.65	25.98	32.0
$\uparrow$ INV_4	-	-	-	-	1.76	3.97	10.0
SMTG_1	64	88	11	0	1.78	28.25	34.0
$\uparrow$ NAND2_4	-	-	-	-	7.44	24.83	37.0
$\uparrow$ INV_1	-	-	-	-	2.48	7.62	14.0
adctopmirror	82	238	1	0	38.55	206.22	252.0
$\uparrow$ libinv	-	-	-	-	0.75	2.20	6.0
lvs_refvref	0	115	0	0	0.03	32.12	41.0
$\uparrow$ refgen	-	-	-	-	1338.4	286.48	1752.0
$\uparrow$ ESDunit	-	-	-	-	3.55	9.90	17.0

TABLE III
RESULTS FOR TSMC 0.18μM

	Retarget		Migration		MATLAB CPU	Cadence CPU	Total elapsed
Name	#DRC	#LVS	#DRC	#LVS	run time (s)	run time (s)	time (s)
INV_8_3S	91	23	8	0	2.19	45.43	52.0
NAND2_8	41	26	31	0	3.05	45.18	53.0
XOR2_2	84	32	12	0	3.01	17.25	24.0
refiref	68	30	42	0	812.19	3206.93	4271.0
LV2HV_60	92	20	29	0	2.46	48.42	54.0
$\uparrow$ INV_4	-	-	-	-	1.62	10.55	27.0
SMTG_1	106	23	70	0	1.71	26.23	31.0
$\uparrow$ NAND2_4	-	-	-	-	7.42	23.97	36.0
$\uparrow$ INV_1	-	-	-	-	2.14	7.10	13.0
adctopmirror	33	24	15	0	40.82	212.93	262.0
$\uparrow$ libinv	-	-	-	-	0.69	1.97	6.0
lvs_refvref	91	0	37	0	0.03	29.30	41.0
$\uparrow$ refgen	-	-	-	-	1273.5	317.77	1631.0
$\uparrow$ ESDunit	-	-	-	-	3.51	11.30	19.0

REFERENCES

[1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 1st edition, 2004.

[2] J. Cohn, D. Garrod, R. Rutenbar, and L. Carley. KOAN/ANAGRAM II: new tools for device-level analog placement and routing. *IEEE Journal of Solid State Circuits*, 26:330–342, Mar. 1991.

[3] G. Gielen and R. Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. In *Proceedings of the IEEE*, volume 88, pages 1825–1854, 2000.

[4] H. Koh, C. Sequin, and P. Gray. OPASYN: a compiler for CMOS operational amplifiers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(2):113–125, Feb. 1990.

[5] B. Owen, R. Duncan, S. Jantzi, C. Ouslis, S. Rezania, and K. Martin. BALLISTIC: an analog layout language. *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 41–44, May 1995.

[6] J. Rijmenants, J. Litsios, T. Schwarz, and M. Degrauwe. ILAC: an automated layout tool for analog CMOS circuits. *IEEE Journal of Solid State Circuits*, 24(2):417–425, Apr. 1989.

[7] R. Rutenbar and J. Cohn. Layout tools for analog ICs and mixed-signal SoCs: a survey. In *ISPD '00: Proceedings of the 2000 International Symposium on Physical Design*, pages 76–83, New York, NY, USA, 2000. ACM.

[8] G. Van der Plas, G. Debyser, F. Leyn, K. Lampaert, J. Vandenbussche, G. Gielen, W. Sansen, P. Veselinovic, and D. Leenarts. AMGIE-A synthesis environment for CMOS analog integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1037–1058, Sept. 2001.

[9] L. Zang and U. Kleine. A novel analog layout synthesis tool. In *ISCAS '04: Proceedings of the International Symposium on Circuits and Systems, 2004*, volume 5, pages 101–104, May 2004.

[10] L. Zhang, N. Jangkrajarng, S. Bhattacharya, and R. Shi. Parasitic-Aware Optimization and Retargeting of Analog Layouts: A Symbolic-Template Approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(5):791–802, May 2008.