Calligrapher: A New Layout Migration Engine Based on Geometric Closeness

Fang Fang, Jianwen Zhu
Electrical and Computer Engineering
University of Toronto, Ontario M5S 3G4, Canada
{ffang, jzhu}@eecg.toronto.edu

Abstract—**As the foundries accelerate their update of advanced processes with increasingly complex design rules, the cost of hard intellectual property (IP) development becomes prohibitively high. Automated layout migration techniques used today, which are based on layout compaction developed a decade ago, corrupt advanced design considerations by honoring only design rules. In this paper, we present two new theoretical results: First, we propose a fast constraint generation algorithm proven to be linear, a step forward from the worst case quadratic complexity achieved in the literature. Second, we propose a new optimization metric, called geometric closessness, that can help retain advanced design intention. A layout migration engine based on these two results is implemented and integrated into a comprehensive hard IP development framework, under which the Berkeley low power libraries, originally developed for 1.2um MOSIS process, are successfully migrated into TSMC 0.25um and 0.18um technologies.**

I. Introduction

It is generally felt that the complexity involved in systems-on-chip (SOC) design can only be tamed by intellectual-property (IP) based design, where the reuse of existing components is advocated to boost design productivity. Two forms of IP are generally used: *soft IPs*, delivered in the form of synthesizable RTL, and *hard IPs*, delivered in the form of layout. Since soft IPs will be re-synthesized by the SOC integrator for their chosen technology, they are favored for their portability. On the other hand, soft IPs are only limited to digital logic and SOC is by no means only a sea of gates. In fact, 70% of the silicon area will be occupied by hard IPs such as SRAMs, FIFOs, CAMs, datapaths and analog front-ends, which are typically designed by abutting a *library* of hand crafted cells. With custom layout design, hard IPs can be delivered with higher performance and better predictability than the soft IPs, and therefore less effort in the SOC integration.

The major bottleneck that prevents the wide adoption of hard IPs is, as its name suggests, the dependency of layout on process. The cost of initial custom layout design is already very high. This applies even to IPs such as standard cells, which is used to be considered simple compared to other IP libraries. Today's standard cell library contains hundreds of cells with different functions and driving strengths. Most fabless companies choose to use libraries offered by hard IP companies precisely because of the high cost associated with library development. To make things worse, manufacturing processes are updated every 18 month, each with a different set of design rules. This makes the development cost of hard IPs too high even for hard IP vendors, since they have to offer different versions for different foundries as well. Automatic layout migration technology, which can amortize the high development cost associated with custom design across different foundries and processes, is therefore crucial for the sustained growth of hard IPs for the small, and the viability of IP-based design for the large.

Unfortunately, layout migration techniques reported in the past cannot cope with all the challenges involved. First, all migration tools are based on layout compaction, a technology developed a decade ago, when layout area is the primary concern. In modern design using aggressive circuit styles in deep submicron processes, space is often among the first class citizens of *advanced layout considerations*, for example, to combat signal integrity. Other specialized *advanced circuit considerations*, such as new transistor sizes, device matching, are rarely considered in an integrated fashion (with the exception of some commercial tools). Second, all techniques reported in the literature are designed to migrate a specific circuit that uses a library of cells, rather than the library itself. Without considering the *overall library architecture* such as power/ground net width, routing track number and port matching, the cell layouts migrated under this circuit-driven strategy work only for the specific circuit, and there is no guarantee they work under all occasions.

In this paper, we focus on addressing the first issue by introducing a new optimization objective, called *geometric closeness*, to reward geometric resemblance of migrated layout to the original layout. Under this metric, space is explicitly represented. Preservation of space and non-space polygons are given equal priority. This ensures that the circuit and layout level considerations of the original designer are not corrupted. Furthermore, We make an additional theoretical contribution by proposing a new algorithm for constraint generation, the core component of a migration engine. The best constraint generation algorithm among numerous attempts reported in the literature achieved quadratic worst case complexity in theory, even though $O(nlogn)$ in practice, where n is the number of nodes, corresponding to polygon edges in the layout. In contrast, we prove the linearity of our algorithm by observing the planar property of the layout.

The rest of the paper is organized as follows. We first give the problem formulation in Section II. We then introduce our design rule constraint generation algorithm and give complexity analysis result in Section III. In Section IV, we introduce and discuss the rationale of the geometric closeness metric. Other practical issues are discussed in Section V before we give experimental results in Section VI. Finally, we review the related work in Section VII.

II. Problem Formulation

In this section, we present a formal model of the layout, design rules and a simplified problem formulation.

In contrast to most reported migration tools, which operate on symbolic layout, our tool directly work on mask layout. Furthermore, since our goal is to preserve original layout design as much as possible, *automatic jog insersion*, a subject that has been explored extensively, is not pursued.

Layout consists of a set of polygons, each associated with a different layer, such as metal, poly, or diffusion. For simplification, people often constrain the polygons to be rectangular, called *Manhattan layout*, and organize related polygons of related layers in finite set of logical layers, called *planes*. Without loss of generality, we consider only Manhattan layout on a single plane.

Definition 1: A **layout topology** is a planar graph $G = \langle C, E = E_H \cup E_V, F, T\rangle$, where C is a set of geometric points, called **corners**, in a two dimensional plane, and $E_H \subset C \times C$ is a set of horizontal edges connecting two corners, and $E_V \subset C \times C$ is a set of vertical edges connecting two corners, and $F \subset 2^E$ is a set of faces, called **tiles**, each of which is an area covered by a cycle of edges, and $T : F \mapsto \mathcal{T} = \{space, poly, metal1, ...\}$ defines the layer type of each tile. Furthermore, the degree of each corner $c \in C$ should be no more than four.

Definition 2: A **layout** $\langle G, X\rangle$ is a layout topology G together with its geometric embedding $X : E \mapsto \mathcal{Z}$ such that each horizontal edge is defined with a coordinate in Y direction, and vertical edge a coordinate in X direction. Note that X fully determines the coordinate of each corner. For any edge $e \in E_V$, we denote y^b_e, y^t_e, x_e as the Y coordinates of its bottom corner, top corner and X coordinate respectively, and L_e, R_e its left and right tile respectively. For any edge $e \in E_H$, we denote x^l_e, x^r_e, y_e as the X coordinates of its left corner, right corner and Y coordinate respectively, and T_e, B_e its top and bottom tile respectively.

Example 1: A Manhattan layout is shown in Figure 1.

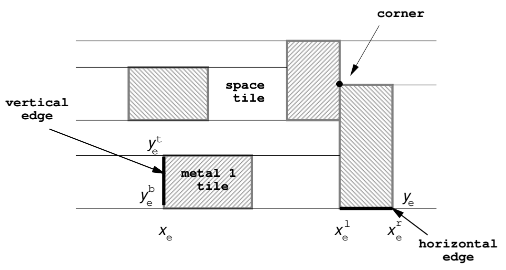


Fig. 1. (a) A layout example.

Design rules are abstractions defined by the foundry to ensure manufacturability. They are typically defined in terms of constraints for the minimum width of a tile, minimum spacing between tiles, minimum extension between tiles, etc, which we call *macro rules*. The majority of macro rules can be translated into a set of constraints on layout edges, called *edge based rule*, which constrain the spacing between overlapping edges. Without loss of generality, we only consider the application of edge-based design rules along the X direction from left to right, and it can be then formally defined as follows.

Definition 3: A **design rule** r is a tuple $\langle t_1, t_2, t_3, d, cd\rangle \in \mathcal{T} \times \mathcal{T} \times \mathcal{T} \times \mathcal{Z} \times \mathcal{Z}$ constraining the spacing between any vertical edges $e1, e2 \in E_V$, where $T(L_{e1}) = t_1, T(R_{e1}) = t_2$, $T(L_{e2}) = t3$, $[y^b_{e1} - cd, y^t_{e1} + cd] \cap [y^d_{e2}, y^u_{e2}] \neq \oslash$, such that $x_{e2} - x_{e1} \geq d$.

Example 2: A layout that violates design rule $\langle t_1, t_2, t_3, d, cd\rangle$ between e_1 and e_2 is shown in Figure 2.

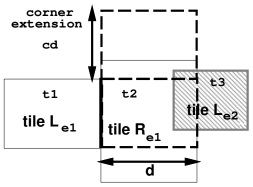


Fig. 2. An edge rule example.

Definition 4: Given a layout $\langle G, X^{old}\rangle$ and a set R of design rules, a layout migration problem finds a new layout $\langle G, X\rangle$, such that every design rule $r \in R$ is satisfied.

III. Design Rule Constraint Generation

The variable to be determined in layout migration is the X coordinate of each layout edge, or equivalently, the left coordinate of each tile. The primary goal is to obtain the values of these variables such that migrated layout is design rule clean. Deriving the design rule constraints is therefore the most important task. Unlike design rule checking against a fixed layout, where design rules need only to be checked for neighboring edges, design rule constraint generation has to assume that every edge can potentially move, and therefore has to generate constraints for every pair of edges in a naive solution.

To obtain more insight into the problem, we first build a graph that can capture the neighboring relation.

Definition 5: Given a layout topology $G = \langle C, E, F, T\rangle$, its neighborhood graph $N = \langle V, E^N\rangle$ is a directed graph whose vertices correspond to the the tiles of G, and an edge $\langle u, v\rangle \in E^N$ iff there is a common edge between the tiles corresponding to u and v.

It is important to note that our input mask layout is directly captured by Ousterhout's corner-stictch data structure [1], where layout edges are already total-ordered when stored on disk. The neighborhood graph is thus implicitly available and takes no time to build.

We first observe that since constraints should be generated only for layout edges overlapping in the Y direction, the neighborhood graph can help to prune the search space: only pairs of tiles that are connected by a path in N need to be considered. The candidate pairs can be identified by finding the transitive closure of the neighboring graph. This can be achieved by iterating on each tile, called the source tile, where a preorder depth-first traversal of its descendents is performed, adding the closure edges along the way.

We can then apply the shadowing principle to further prune the size of the transitive closure: during the depth-first traversal, we first examine if the left edge of the discovered tile overlaps with the source tile under consideration, and stop further exploration of the tile if it turns out false.

While the shadowing was effective in the past to limit search space in the Y direction, we propose a new strategy to further limit the search space in X direction. We observe that as we explore along the X direction, each tile has a minimum width requirement dictated by the design rule. This can be summed up along the path and be used as the lower bound estimate of the distance between the source tile and the current tile. If the lower bound exceeds the constraint distance d of the design rule under consideration, further exploration of the current tile can be stopped.

Example 3: Consider the layout example in Figure 3 (a). The shadow is indicated by the dashed line. A faction of the neighborhood graph is shown in Figure 3 (b), where each tile is labeled with its minimum width requirement of $d1, d2$ etc. Suppose the constraint distance is D, and $d1 + d2 + d3 \leq D$, $d1 + d2 + d3 + d4 \geq D$, then tile 10 and tile 11 will be pruned from the search space.

As such, it is guaranteed that there exists an upper bound of the depth we have to search. Algorithm 1 shows how the closure graph defined in Definition 6 can be built, upon which design rules can be directly constrained by examining the design rule set.

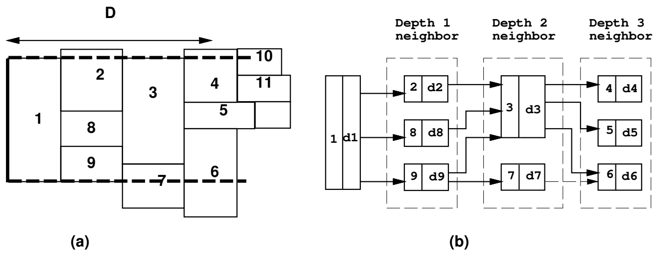


Fig. 3. (a) A layout consisting of tiles 1-11; (b) Neighborhood graph.

Definition 6: Given a neighborhood graph $N = \langle V, E^N \rangle$, its **depth-K shadowing closure** is a directed graph $\overline{N} = \langle V, \overline{E^N} \rangle$ such that $\langle u, v \rangle \in \overline{E^N}$ iff $\exists p \in u \rightsquigarrow v.|p| \leq K$ and $[y_u^b, y_u^t] \cap [y_v^b, y_v^t] \neq \oslash$.

Algorithm 1: Depth-K shadowing closure algorithm.

input: $N = \langle V, E^N \rangle$;
output: $\overline{N} = \langle V, \overline{E^N} \rangle$;
func explore($u, v, depth$) {
  if($[y_u^b, y_u^t] \cap [y_v^b, y_v^t] = \oslash \vee depth++ > K$)
    return;
  $\overline{E^N} = \overline{E^N} \cup \langle u, v \rangle$;
  forall($w \in \{w | \langle v, w \rangle \in E^N\}$)
    explore($u, w, depth$);
}
func depthKShadowingClosure() {
  forall($u \in V$) {
    forall($v \in \{v | \langle u, v \rangle \in E^N\}$)
      explore($u, v, 1$);
  }
}

We now consider the complexity of Algorithm 1. It is important to first observe that the number of edges in a planar graph is in the same order of vertices.

Lemma 1: The number of edges in the neighborhood graph $N = \langle V, E^N \rangle$ is of $O(n)$, where $n = |V|$.

Proof: N is the dual of the layout topology $G = \langle C, E, F, T \rangle$. N is therefore also planar. Let F^N be the set of faces of N, then according to Euler's formula, $|V| + |F^N| - |E^N| = 2$. Since $\forall f \in F^N.|f| \geq 3$, we have $2|E| = \Sigma_{f \in F^N} |f| \geq 3|F^N|$. By Euler, we have $|E| \leq 3|V| - 6$. □

Unfortunately, we have to search more than the immediate neighbors. It turns out the number of closure edges added by Algorithm 1 is still in the order of vertices. This is by no means obvious, and Lemma 2 demonstrates why.

Lemma 2: The number of closure edges of each depth obtained by Algorithm 1 is of $O(n)$, where $n = |V|$.

Proof: We classify vertices in the neighborhood graph into two types. Type 1 are those whose two end points are completely contained to its left neighbor (those who always have a incoming degree of 1), all others are Type 2 edges. It is easy to see that $\forall v \in V$, there are at most two Type 2 edges emitting from v. When building closure edges, we only need to consider decedents of Type 2 edges according to the shadowing principle. The total number of closure edges starting from a vertex v is therefore bounded by 2^{K-1}, a constant number. We can then conclude that the total number of closure edges are of $O(n)$. □

Theorem 1: The complexity of Algorithm 1 is $O(n)$, where n is the number of vertices, or tiles, in the neighborhood graph $N = \langle F, E^N \rangle$.

Proof: Since the complexity of the algorithm is propotional to the number of constraints generated, the conclusion follows easily from Lemma 2. □

IV. Objective Function

The *constraint-based migration* of a layout can be formulated as an integer linear programming (ILP) problem:

$$\begin{array}{ll} minimize & o^T x \\ subject\ to & Ax \geq b \\ & x \geq 0 \end{array}$$

Here x is a vector of X coordinates of edges in E_V. Vector o represent the coefficients of x in the objective function of optimization. A is the constraints, each row of which represents coefficients of x in an inequality. Typically, an inequality will be in the form $x_i - x_j \geq b_k$, where the constant b_k represents the minimum distance between two edges.

The migration engine generates constraints between these edges based on new design rules and dump these constraints and objective functions to the ILP solver. The new positions of each tile will be determined from the ILP solver.

The performance of migrated layout is largely influenced by the choice of objective function. The traditional minimum area objective function takes layout area as the only criterion and shrinks area to a maximum extent without considering layout design issues such as keeping two important signals apart to reduce coupling between them. In order to take full advantage of the original design and make minimum changes to the migrated layout, *Minimum perturbation* objective function is proposed in [2], which is defined as:

$$minimize \quad \sum |x - x^{old}| \tag{1}$$

where x is the vector of variables that determine X coordinates of all vertical edges and x^{old} is the vector of constants that are the old X coordinates of all vertical edges in the layout. The minimum perturbation function minimizes the position changes of all edges and snaps the edges to their original positions as much as possible. However, the disadvantage of this function is that it minimizes the absolute coordinates of edges and will penalize more on the movement of edges on the right side of the layout.

Example 4: Consider a simple layout given in Figure 4 with two tiles of metal2 type. Because of technology change, the minimum width requirement of tile 1 is changed from 4λ to 5λ and distance requirement between tile 1 and tile 2 is changed to 5λ too. Based on minimum perturbation objective function, the layout will be migrated to the one on the lower part of Figure 4. The right edge of tile 1 will be stretched rightward by 1λ and left edge of tile 2 will be moved rightward by 2λ. However, the right edge of tile 2 stays at the old position because without change of its position, its width is 5λ which satisfies the new design rule. Otherwise, the objective function value will be greater than the one with $x_2^{'}$ unchanged.

It can been seen from this example that the movement of edges on the left side will add penalties on the edges on the right side if we want to preserve the original shape of tiles on the right.

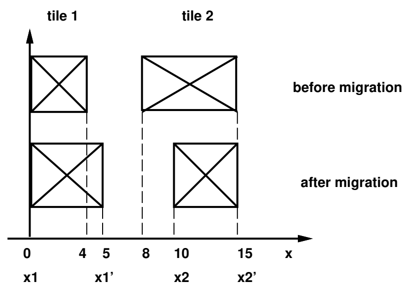


Fig. 4. The old layout and migrated layout with minimum perturbation objective function.

To remove this penalty, a new objective function, *geometric closeness* objective function is used in our migration tool which is defined as :

$$\sum \left|(x_{ri} - x_{li}) - (x_{ri}^{old} - x_{li}^{old})\right| \tag{2}$$

Here, x_{ri} and x_{li} are the X coordinates of the right and left edges of each tile in the layout. The constants x_{ri}^{old} and x_{li}^{old} are the X coordinates of the right and left edges of the corresponding tile in the original layout. Instead of minimizing the absolute coordinate changes of edges, this function minimizes the shape changes of all tiles so that each tile change will not add penalty to other tiles.

Example 5: With geometric closeness objective function, the example given in Figure 4 will be migrated into the layout in Figure 5. It can be seen that the width of tile 2 is set to the original value and the topology of the old layout is preserved to a maximum extent.

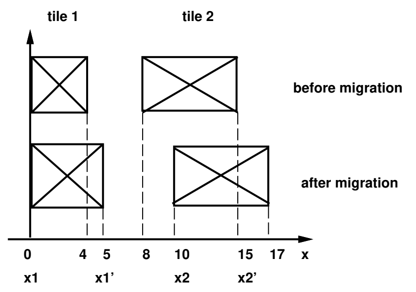


Fig. 5. The old layout and migrated layout with geometric clossness objective function.

V. Other Considerations

Having discussed the baseline algorithm, we now discuss other practical considerations.

- **Topology preprocessing**: As new technology was not available when old layout was created, it is highly likely that there are *forbidden edges* which are not allowed in new technology. The design rule constraint generation discussed before cannot solve this problem because it assumes all edges in the layout are legal and generate constraints according to the type of edges. So, before constraint generation is processed, topology preprocessing runs first to search forbidden edges and separate abutted two tiles that cause forbidden edges.
- **Corner checking**: The constraint generation in corner extension region is similar to the algorithm discussed in Section III except that the checking area is moved to corners.
- **Interpass constraint generation**: As we employ the traditional 1-D compaction strategy, X direction migration and Y direction migration are independent of each other, which will give rise to missing of constraints for both X migration and Y migration.

Example 6: Consider the layout in Figure 6, tile 3 is located beyond the shadow area A of tile 1. So, during X direction migration, tile 3 is not included in the neighborhood tree of tile 1 and no constraint is generated between left edge of tile 1 and left edge of tile 3. After X direction migration and Y direction migration, it is possible that tile 3 moves into area A. A design rule violation may occur if the distance between left edge of tile 3 and left edge of tile 1 is less than the distance requirement.

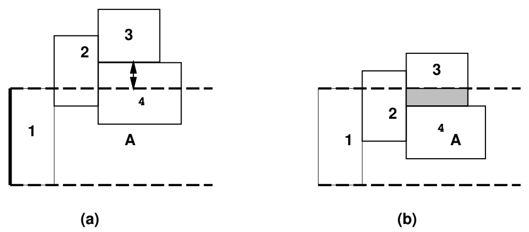


Fig. 6. (a)In the old layout, tile 3 doesn't overlap with tile 1 in Y direction. (b)After X direction migration and Y direction migration, tile 3 moves into shadow area of tile 1.

To solve this problem, additional constraints, *interpass constraints*, need to be generated to prevent the arbitrary movement tiles during X and Y direction migration. Given a source tile $f_1 \in F$ (eg, tile 1 in Figure 6), for each tile f_2 in its neighborhood graph N and $y_{e1}^t \in [y_{e2}^t, y_{e2}^b]$, we generate constraints: $y_{e1}^t \leq y_{e2}^t$ and $y_{e1}^t \geq y_{e2}^b$. With interpass constraints, those tiles that donnot overlap with source tile in the original layout will always be kept away from checking area, thus no design rule violations will occur after migration.

- **High level architectural constraint generation**: In order to migrate the whole leaf cells in libraries, more high-level architectural constraints such as power line net width, port matching need to be considered. Otherwise, leaf cells cannot be used repeatedly by synthesis tools and place and route tools to build entire design [3]. Our migration framework employs a two-pass strategy which breaks the dependency between different leaf cells for generating high level architectural constraints. The tool accepts a library of leaf cells and a specification of the library architecture that is abstracted from characteristics of the library. During the first pass, migration engine generates a temporal design rule clean layout with integer linear programming solver (ILP). In the second pass, the temporal layout solutions in the first pass are analyzed and all the architecture specifications are translated into linear constraints according to the temporal solutions. As this is not the main focus of this paper, interested readers can refer to [4] for more details.

VI. Experimental Results

We implemented the migration tool on top of an IP-centric CAD infrastructure, called *ipsuite* which uses Magic's corner-stitching data structure [5]. An open-source ILP solver is used as well. The migration tool itself is implemented by 10K lines of C code.

To test the effectiveness of our tool, we apply our tool on the low-power standard library and datapath library developed by Burd [6] at University of California, Berkeley. This library was based on SCMOS 1.2μm technology. Our targeted process is TSMC 0.25 μm and TSMC 0.18μm technologies.

Figure 7 (a) shows a two-input multiplexer leafcell layout in standard cell library. Figure 7 (b) shows the layout that has been migrated towards TSMC 0.25μm technology based on minimum perturbation objective function and Figure 7 (b) shows the layout that has been migrated toward TSMC 0.18μm technology based on geometric closeness objective function.

Figure 8 shows the migration result of an 8-bit adder which consists of abutting of eight identical full adder cells as well as a control cell. Note that many of the ports, for example *carryin* signal and *carryout* signal need to be matched between cells. And power line widths are user defined.

Table I gives more comprehensive results carried out on a SUNBlade 100 system running at 500 MHZ. Here, related cells that need port matching are migrated in groups. The fifth column and sixth column demonstrate the run time for constraint generation and run time for ILP sover respectively (measured in seconds). Define value of the *layout change* as the value of geometric objective function. The last two columns, *MP layout change* and *GC layout change* present layout change values for layout based on minimum perturbation objective function and geometric closesness perturbation objective function respectively. The number of tiles (including space) in the layout, and the total numbers of design rule constraints generated are shown in the third and forth column. As we can observe, the ILP solution dominates the computation time. Since our two-pass framework allows us to run ILP one cell at a time, the total migration time is limited in minutes and is therefore tolerable. It is interesting to note that while both the standard cell and datapath library is designed very compactly and use minimal size as much as possible to reduce power consumption and leave very little flexibility in the solution space, GC and MP produces as much as 20-45% difference for some cases.

VII. Related Work

Automatic layout migration was among the oldest CAD problems investigated and a large body of research was carried out under the layout compaction problem. A comprehensive survey of the literature is out of the scope of this paper and the readers are referred to [7] and [8]. The compaction methodologies include: shear-line approach, virtual grid approach and constraint graph approach. For constraint graph approach, shadow-propagation method [9] is widely used for design rule constraint generation. The time complexity of this method is proven to be $O(n^2)$ and n is the number of edges in symbolic layout. If no elements swapping allowed in symbolic layout, the time complexity of $O(nlog(n))$ can be achieved for constraint generation [10].

The *minimum perturbation objective function* proposed in [2] was the first work that departed from the traditional optimization goal and argued the importance of rewarding geometric similarity between the migrated layout and the original layout. Our geometric closeness objective is inspired by this work. However, Section VI shows that these two functions may deviate in some situations and as discussed, the proposed objective can avoid biasing along a particular direction.

The port matching problem, or sometimes referred to as pitch matching problem, was solved in the past by hierarchy compaction [11] [12]. This was achieved by the so-called port abstraction of each cell which might seem to be similar to our two-pass strategy, it is not designed to migrate a library where a top-level circuit does not exists, and it can only perform minimum area compaction, which is less important than other considerations.

VIII. Conclusion

In conclusion, we have argued that migration technology is essential for the success of hard IPs. We have demonstrated a layout migration engine featuring a linear time constraint generation algorithm and a new optimization metric based on geometric closeness, which when applied, can help retain advanced design considerations. Future work includes a more ambitious transistor-sizing capability where layout topology can be changed to insert transistor fingers, as well as the integration with automatic transistor sizing tools.

References

[1] John K.Ousterhout, "Corner stitching: a data-structuring technique for vlsi layout tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 87–100, 1984.

[2] Fook-Luen Heng, Zhan Chen, and Gustavo E.Tellez, "A VLSI artwork legalization technique based on a new criterion of minimum layout perturbation," in *1997 International Symposium on Physical Design*, 1997, pp. 116–121.

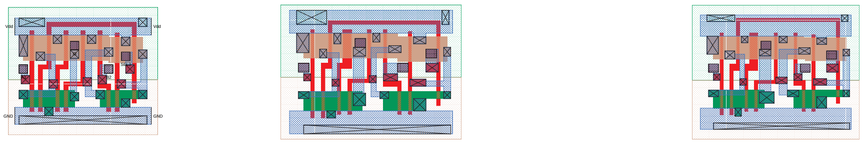
[3] *CMOS IC layout: concepts, methodologies, and tools*, Butterworth-Heinemann, 1999.

[4] Fang Fang and Jianwen Zhu, "Automatic migration of datapath hard IP libraries," in *Asia South Pacific Design Automation Conference*, 2004.

[5] John K.Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 87–100, 1984.

[6] Tom Burd, "Very low power cell library," Tech. Rep., University of California, Berkeley, 1995.

[7] David G.Boyer, "Symbolic layout compaction review," in *Proceeding of the 25th Design Automation Conference*, 1988, pp. 383–389.



(a) Original layout (b) Migrated in TSMC 0.25μm with MP (c) Migrated layout in TSMC 0.25μm with GC

Fig. 7. A two-input Multiplexer.

(a) Original layout (b) Migrated layout in TSMC 0.25 μm (c) Migrated layout in TSMC 0.18μm

Fig. 8. Migration of 8-bit adder with GC.

TABLE I

EXPERIMENT RESULTS.

Library Name	Cell Name		# of tiles	# of DRC constraints 0.18μm / 0.25μm	ILP run time (s) 0.18μm / 0.25μm	DRC CG run time(s) 0.18μm / 0.25μm	GC layout change	MP layout change	
standard cell library	andf301		243	6208/6224	18.6/20.6	0.45/0.61	791	879	11%
	aof3201		463	14842/14878	107.4/110.5	1.16/1.17	1851	1918	4%
	blf00001		195	4181/4184	12.1/10.1	0.25/0.32	434	466	7%
	buff102		169	3550/2719	8.1/6.9	0.32/0.27	534	551	3%
	dfnf401		353	10643/10643	58.4/56	0.73/0.92	1145	1208	5%
	drif101		275	7636/7656	27.6/27.9	0.64/0.55	579	590	2%
	invf101		100	1583/1587	1.2/1.2	0.13/0.15	243	309	27%
	labf211		237	5906/5922	16.2/17	0.53/0.49	793	830	5%
	lrbf202		76	997/997	0.5/0.5	0.05/0.08	352	358	2%
	muxf201		337	10006/10034	40.5/49.5	0.81/0.74	1059	1137	7%
	nanf201		143	2707/2715	3.8/4.2	0.22/0.18	409	452	10%
	norf211		202	4699/4711	11.7/11.6	0.28/0.45	663	703	6%
	oaif2201		243	5951/5967	18.4/18.6	0.40/0.37	778	794	2%
	orf401		338	9947/9975	49.7/49.8	0.74/0.78	1798	1903	6%
	swcf020		154	3177/3185	5.0/5.2	0.24/0.33	418	483	16%
	xnof201		307	8892/8888	45.3/41.5	0.67/0.77	793	803	1%
	xorf201		303	8684/8660	41.5/37.3	0.70/0.66	732	742	1%
datapath libarary	adder	add	1079	36866/56127	1140/1110	149.5/134.9	2763	3044	10%
		add_cs_sel	167	3070/4288			269	309	13%
		add_cs_0	29	223/225			22	22	0%
	mux2	mux2	357	7745/11797	174/192	10.5/10.9	752	878	17%
		mux2_cs1	179	2203/4248			124	137	10%
		mux2_cs2	181	14574/14573			133	149	12%
		mux2_cs3	266	17512/17514			287	334	14%
		mux2_cs4	198	8518/11096			352	408	16%
	register	rf0	204	2748/2748	36/36	3.9/3.7	229	287	25%
		rf1	202	2756/2756			185	220	20%
		rf01_cs1	228	3517/3516			372	397	7%
		rf01_cs2	230	3584/3584			381	401	5%
		rf01_cs3	223	3464/3464			332	481	45%
	random logic	and2blp	300	5940/5940	198/198	6.6/6.4	552	587	6%
		nand2lp	240	4015/4015			294	304	3%
		nandand3lp	319	6706/6706			786	870	11%
		or2blp	273	5319/5319			384	411	7%
		xnor2lp	343	5319/5319			438	460	5%
		xor2lp	345	8003/8003			436	467	7%

[8] Y.Eric Cho, "A subjective review of compaction," in *Proceeding of the 22nd Design Automation Conference*, 1985, pp. 396–404.

[9] M.Y.Hsueh and D.O.Pederson, *Computer-aided layout of LSI circuit building-blocks*, Ph.D. thesis, Univeristy of California at Berkeley, 1979.

[10] Jurgen Doenhardt and Thomas Lengauer, "Algorithm aspects of one-dimensional layout compaction," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1987, pp. 863–878.

[11] David Marple, "A hierarchy preserving hierarchical compactor," in *27th ACM/IEEE Design Automation Conference*, 1990, pp. 375–381.

[12] Christopher Kingsley, "A hiererachical, error-tolerant compactor," in *21st Design Automation Conference*, 1984, pp. 126–132.