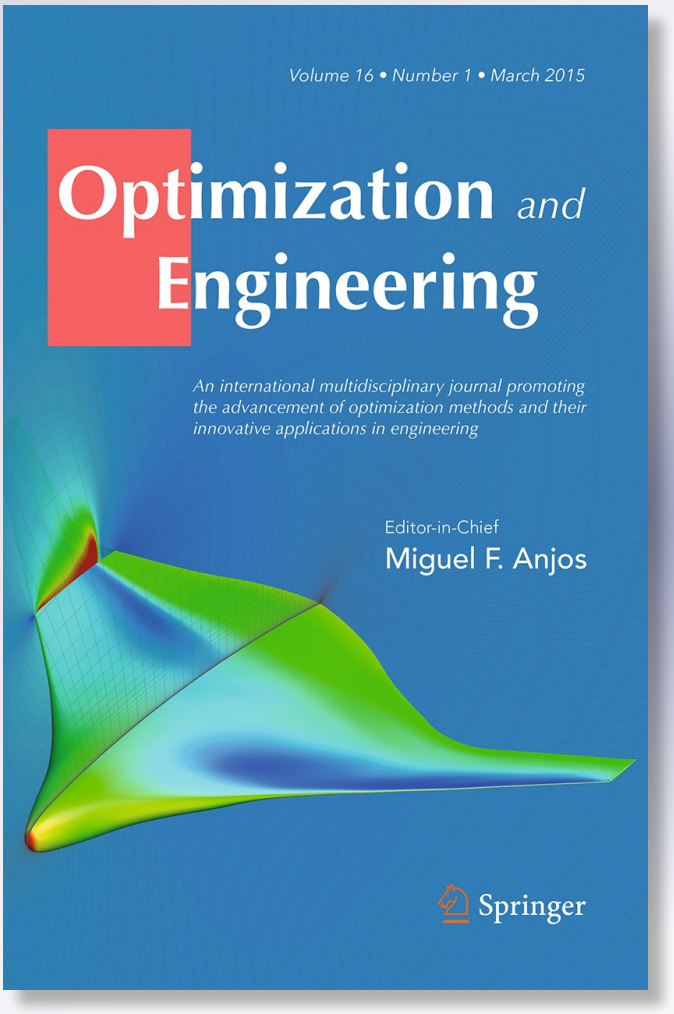
Efficient cell-based migration of VLSI layout

Eugene Shaphir, Ron Y. Pinter & Shmuel Wimer

Optimization and Engineering
International Multidisciplinary Journal to Promote Optimization Theory & Applications in Engineering Sciences

ISSN 1389-4420
Volume 16
Number 1

Optim Eng (2015) 16:203-223
DOI 10.1007/s11081-014-9257-7



Springer

Optim Eng (2015) 16:203–223
DOI 10.1007/s11081-014-9257-7

Efficient cell-based migration of VLSI layout

Eugene Shaphir · Ron Y. Pinter · Shmuel Wimer

Received: 27 September 2012 / Accepted: 4 February 2014 / Published online: 28 March 2014

Abstract In Intel's "Tick-Tock" roadmap a new processor is first manufactured in the most advanced stable process technology, followed in a 1-year delay by introducing chips comprising same microarchitecture but manufactured in a newer scaled process technology. Tick-Tock is enabled by the automation of chip's layout migration from an older into a newer process technology, known as hard-IP reuse. This is a very challenging computational task, involving billions of polygons. Migration algorithms have been thoroughly studied and implemented in the past but their computational capabilities fall short compared to today's demand. We describe a hierarchy-driven computationally efficient algorithm for cell-based layout conversion, used by Intel in its Tick-Tock roadmap. The algorithm transforms the full chip conversion problem into successive problems of significantly smaller size, having feasible solutions if and only if the full chip problem does. The proposed algorithm preserves the design intent, its uniformity, portability and maintainability, a key for the success of large-scale projects.

Keywords VLSI design migration · Layout compaction · Interconnects · Cell-based design

1 Introduction

Due to their high complexity, VLSI chips are built hierarchically as shown in Fig. 1. The entire chip corresponds to the top of the hierarchy, while at the bottom there are the individual transistors. Transistors are connected with each other in *standard-*

E. Shaphir · R. Y. Pinter
Computer Science Department, Technion, 32000 Haifa, Israel

S. Wimer (✉)
Engineering Faculty, Bar-Ilan University, 52900 Ramat-Gan, Israel
e-mail: wimers@biu.ac.il

cells, implementing basic logic and memory functions. Those are connected together in more complex functional blocks such as adders, multipliers, and memory arrays, among others. Functional blocks are connected by wires in higher level functions such as arithmetic–logic units, register-files, control units, etc. The top of this hierarchy are DSP, networking, communication and sensor modules, where their connection constitutes the so called system on chip, occupying the entire silicon die. Contemporary VLSI technologies may comprise a dozen of metal layers occupying a huge number of interconnecting wires.

The design of high-end full-custom microprocessors is a very complex engineering task, involving hundreds of man-years efforts. Hierarchical design methodology is a key to meet time-to-market requirements. Moore's Law (Perry 2008) enables the introduction of new processor's microarchitecture in a 2-year cycle, named by Intel "*Tick-Tock*" roadmap, as illustrated in Fig. 2. The Tock phase delivers every 2 years a new microarchitecture manufactured in the most advanced stable technology. It is then followed in 1-year delay by a Tick phase, delivering chips of the same microarchitecture but in a new scaled process technology, allowing higher production volumes, better performance and lower cost. An essential part of the Tick phase is the conversion of the underlying physical layout, comprising billions of polygons, into the new technology where they must satisfy complex geometric rules. Such conversion is known in VLSI jargon as *hard-IP reuse* (Nitzan and Wimer 2002), based on the automation of chip's layout conversion from older into newer technology. This is a very challenging computational task.

An alternative to hard-IP reuse is *soft-IP reuse*, where the hardware definition language of the Tock phase design is synthesized into the target technology by electronic design automation (EDA) logic and physical synthesis tools (Saleh et al. 2006; Chiang 2001). Compared to hard-IP reuse, soft-IP reuse has the advantages of flexibility and portability. It also eases the introduction of new features into the Tick phase. For high-end custom designs as those discussed in his paper it has a significant drawback though. It cannot guarantee timing and power convergence, since all the physical design effort invested in the Tock is being lost. Today's high-end VLSI designs require considerable engineering effort for timing and power convergence.

In hard-IP reuse the polygons are converted by *compaction* algorithms, a comprehensive description of which can be found in (Lengauer 1990; Reinhardt 2002). The compaction describes the positional relations of the *source* layout's polygons by a directed graph, called the *constraints graph*. Its vertices represent edges of polygons and arcs represent left-to-right (bottom-to-top) adjacency and visibility relations. The arcs are assigned weights corresponding to the sizes and spacing *design rules* of the new technology. The problem of sizing and positioning the polygons in the new *target* layout is to find the smallest possible area that can legally accommodate the layout.

The most general form of compaction is two-dimensional (2D). It moves the polygons of the layout in x- and y- coordinates simultaneously, a problem shown to be NP-complete (Sastry and Parker 1982; Lengauer 1984). Compactors therefore decompose the 2D problem into a sequence of alternating independent one-dimensional (1D) compaction steps, each change only one set of coordinates. 1D

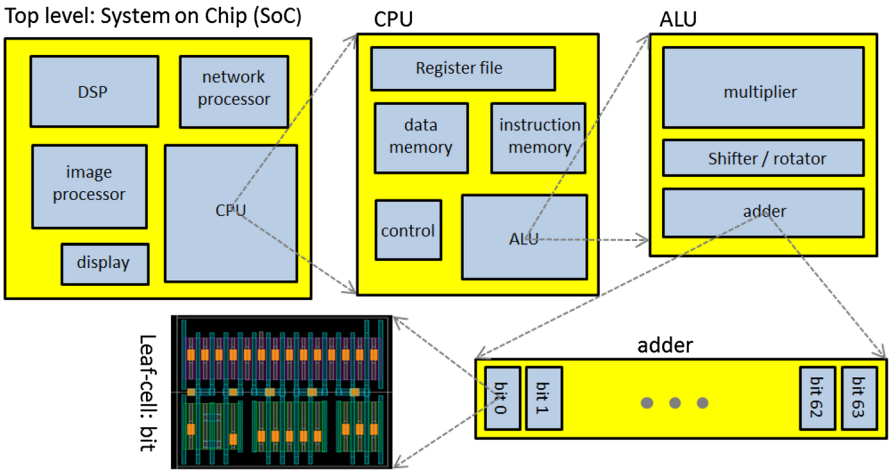


Fig. 1 The hierarchy of a chip

Architectural Change		Process Technology	Microarchitecture	Release date
Tick	Die shrink	65nm	P6, NetBurst	5 / 2006
Tock	New microarchitecture		Core	7 / 2007
Tick	Die shrink	45nm		11 / 2007
Tock	New microarchitecture		Nehalem	11 / 2008
Tick	Die shrink	32nm		1 / 2010
Tock	New microarchitecture		Sandy Bridge	1 / 2011
Tick	Die shrink	22nm		4 / 2012
Tock	New microarchitecture		Haswell	6 / 2013

Fig. 2 Intel's "Tick-Tock" roadmap

compaction can be solved efficiently with longest path algorithms (Lengauer 1990; Reinhardt 2002). Polygons not on the critical paths are positioned such that some cost reflecting a design goal (e.g., performance, sensitivity for manufacturing defects, among a few others) is minimized, for which Linear Programming (Wang and Lai 2001) and Integer Linear Programming (ILP) (Fang and Zhu 2004) are used.

A heuristic solution of the 2D problem was proposed in (Shin et al. 1986) where it was first solved by alternating 1D compactions. The layout was then relaxed by introducing extra jogs into the wires to enable further compression. In today's technologies which due to lithographic limitations require high regularity and uniformity of the interconnecting wires, jog insertions are prohibited. The work in (Schlag et al. 1983) showed the NP completeness of the 2D problem and proposed a branch-and-bound algorithm, suitable for very small layouts, but impractical for the large-scale problems. The 2D problem was studied also in the context of *graph-*

drawing (Klau and Mutzel 1999; Patrignani 2001), leading to similar conclusions of its difficulty. The EDA compactor is also 1D-based, allowing the user to control the compaction iterations by specifying an appropriate parameter to be either *x-y-x* or *y-x-y*. The EDA tool (Reinhardt 2002) that was used for the layout migration of the DEC's Alpha microprocessor is also 1D.

Traditional compaction algorithms are working on flattened layouts, suited to cope with up to few tens of thousands of polygons. With the evolution of VLSI technology in the 90's to integration of few million transistors on a chip, design methodology moved towards more standardization, modularity and re-use, dictating hierarchical chip structure. In parallel, design rules became more complex, which altogether made layout migration a computation challenge (we subsequently use the terms compaction, migration and conversion interchangeably).

Compaction algorithms and EDA tools supporting hierarchy were presented in (Burns and Newton 1987; Burns and Feldman 1998; Yao et al. 1993; Wang and Lai 2001). Compaction creates unique blocks (also called *modules* or *cells*), which cannot be shared and re-used among different layouts. Therefore, although those compactors maintained layout hierarchy, the duplication and mutations of same logic blocks is a major drawback that today's designs cannot afford. A new layout migration technology called *cell-based compaction* is thus in order. It uses an ordinary *standard-cell library*, optimized regardless of its instances in the entire layout. Cell-based migration has the problem of creating a huge constraints graph incorporating all the instances of all blocks, which is then translated into a huge optimization problem whose solution may take enormous computation time. This paper reduces the size of the compaction problem in one to two orders of magnitude.

Hierarchical compaction was first addressed in (Burns and Newton 1987), ensuring that the modularity of the target layout will stay similar to its source. It efficiently handled layouts comprising a few dozen transistors, but did not take advantage of the repetitive instantiation of the same cells to reduce computation complexity, which our work does. This is an enabler of layout migrating at chip-scale. The work in (Burns and Feldman 1998) handled larger blocks comprising thousands transistors, proven on real IBM design. It was tailored to a regularly structured control-logic, comprising two-levels of hierarchies: leaf-cells and the entire block. This prohibits its usage for general, multi hierarchy, custom layouts. Furthermore, same leaf-cells in different blocks were in-place compacted, resulting in mutations of the same logic cell. It prohibits cell-level electrical characterization, a key for efficient timing analysis. Timing analysis must therefore take place at transistor-level, a big engineering effort overhead. This paper in contrast supports any hierarchy depth, making it useful for custom data-path and register-file design styles. Moreover, our migration flow is cell-based, enabling the usage of standard-cell library with all the benefits of modular design, portability, and efficient timing analysis.

Cell-based layout compaction was claimed in (Yao et al. 1993). It emphasized the pitch-matching of cells and heavily relied on the slicing structure of the layout. It is applicable to two-level place and route layout style, but inadequate for other layout styles, full-custom in particular. As all the other hierarchical compactors, the cells are in-place compacted, prohibiting the advantages of real cell-based design.

Layout position constraints are naturally described by as LP or ILP. The work in (Wang and Lai 2001) took advantage of the special LP matrix form derived by layout constraints. It supports hierarchy, but as other works, the leaf-cells are compacted in-place, a drawback mentioned before. It was also proven to work on problems comprising only few thousands of variables and constraints, which is impractical for chip-scale problems. In today's nanometers scale process technologies design rules are restricted to discrete values and layout objects must align to predefined grids. This requires using ILP instead. The work in (Fang and Zhu 2004) presented ILP-based data-path macros layout migration where the wires in the layout were enforced to predefined grids. It used geometric closeness optimization objective to reward geometric resemblance of target layout to the source layout.

The above works evolved into an EDA large-scale hierarchical compaction tools. It was successfully used by Intel for several process generations (Nitzan and Wimer 2002; Wimer 2014), from 130 nm, through 90 and 65, to 45 and 32 nm, shown in Fig. 2. It still in-place compacted leaf-cells, which limited its usage for cell-based design. Moreover, while all past works worked on the entire flattened layout, thus addressing huge problems, our algorithm is successively solving a series of far smaller problems, but still exploring the entire solution space.

Figure 3 shows that the 1D interconnect migration proposed by this paper is useful for 2D problems. Due to the uniform longitudinal and latitudinal nature of wires, there is not much optimality loss by successive 1D treatment. The target widths of the wires are decided prior to compaction by timing considerations. The algorithm x-shifts wires in vertically routed layers and y-shifts wires in horizontal layers. Shifted wires are hooked to perpendicular wires residing in an adjacent layer below and above by vias at their ends. Stretching the perpendicular wires to the new coordinate of their ends maintains the connectivity.

The layout design rules in modern VLSI process technologies are complex and their amount may reach a few hundreds. The majority of those however relate to the lower layers involving transistors and their interconnections used within standard, leaf-cells, whose layouts are manually migrated. We discuss the problem of migrating the wires in the higher metal layers used to interconnect leaf-cells into higher level functions. The primary design rules for wires are minimal width, minimal spacing, and the metal coverage of vias. While the target width of wires is set by performance considerations prior to migration, their spacing is solved by compaction. Vias are formed at the incidence of orthogonal wires as shown in Fig. 3. It may happen that the resulting layout still has some design rule violations. Those are manually fixed at a later stage of the design.

Figure 4a illustrates a typical VLSI layout comprising several blocks placed within each other, thus constituting the hierarchy. All the wires residing on even layers are vertical, while those in odd layers are horizontal. Mixing both directions in the same layers is forbidden (see Fig. 3). Each block has *IO ports* for interconnections to other blocks (the terms wires and interconnections are used interchangeably). The wires connecting child blocks placed within a parent to each other, and to the IO ports of their parent, belong to the parent. The different colors of blocks' borders represent levels of the hierarchy represented in Fig. 4b. Notice that a block may be placed multiple times within different parents. The unique definition

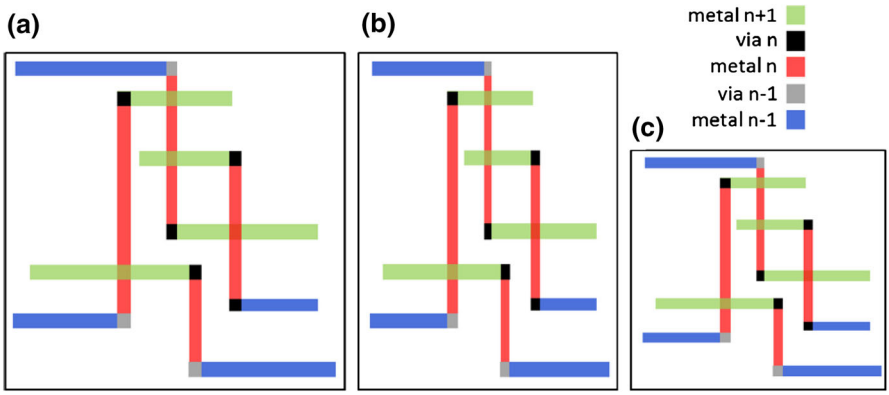


Fig. 3 A 2D compaction by successive 1D compactions. **a** Original layout, **b** after *x* compaction, **c** after *y* compaction

of a block is called a *master*, and its specific occurrence in the design is called an *instance* (the terms master and block are used interchangeably). The origin of a master is associated to its lower-left corner. Instances are placed within their parent at some *x* and *y offsets*. Instances place in a common parent do not overlap.

The wires in Fig. 4a reside on two adjacent metal layers. Wires are connected at their incidence point by a VIA. The solid wires interconnect black-border children within their green-border parent, whereas the dotted ones connect orange-border children within their purple-border parent. Internal wires of orange-border blocks are not shown. Solid and dotted wires of the same color belong to the same physical layer and are therefore not allowed to touch each other as otherwise an electrical short occurs. Wires of same layer must satisfy minimum width and spacing rules, as otherwise a malfunction at manufacturing may occur. Extra widths and spacing may be specified per wire to satisfy design goals as performance, noise immunity, small *IR* drop and reliability.

The progression from old to new technology is featuring a 0.7 average scale of all lateral dimensions, thus enabling to double the numbers of transistors per silicon area. This is the well-known Moore's Law, governing the VLSI evolution for already five decades (Perry 2008). Until late-90's, the 0.7 scaling equally applied to all lateral dimensions of the polygons over all layers. VLSI layouts could therefore be migrated by a purely linear transformation called "*optical shrink*". The linear scaling has been broken in the last 15 years due to many technology difficulties, turning layout conversion into a strongly non-linear problem. Performance requirements of specific signals (called also nets) may only worsen the nonlinearity by introducing extra geometric constraints.

The main contribution of this paper is in presenting an algorithmic design-flow for cell-based compaction of complete VLSI chips. Although compaction algorithms have been thoroughly studied and implemented in the past, their computational capabilities fall short compared to today's demand. The novelty of this work is in exploiting the hierarchical structure of VLSI chips to achieve one to two orders of

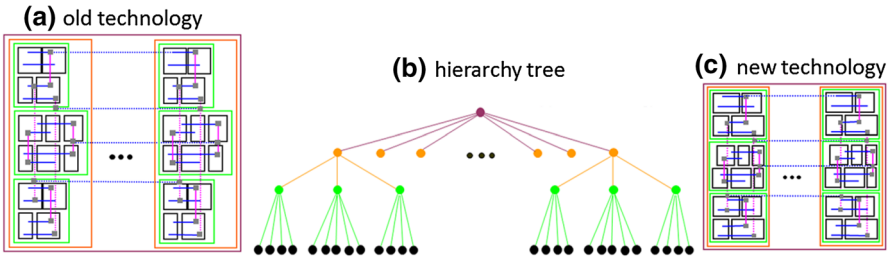


Fig. 4 A four-level hierarchical layout in old technology is shown in (**a**), its corresponding hierarchy tree in (**b**) and the target layout in new technology in (**c**)

magnitude improvement in computation time. We present time and space efficient bottom-up and top-down migration algorithms. The rest of the paper is organized as follows. Section 2 describes the cell placement migration and its relation to the entire problem. Section 3 presents the interconnect migration problem and outlines its solution by using a suitable graph-theoretic model. Section 4 shows experimental results obtained for Intel's VLSI blocks migrated to 45 nm technology. Section 5 concludes the discussion.

2 Migration overview

Due to its complexity, the generation of the physical layout of VLSI systems takes place in two steps: placement and routing (Saleh et al. 2006). This work follows a similar pace; the placement of the instances of all the masters is migrated first, and then migration of all their underlying interconnections is performed.

2.1 Placement migration

The placement migration is subsequently described. The target of the placement in Fig. 4a is shown in Fig. 4c. Placement migration is aimed at three goals:

1. Producing a similar scaled layout, preserving all the left-to-right and bottom-to-top adjacency relations between its block instances.
2. Obtaining a small, compact layout, reflecting the 0.7 average Moore's Law scaling in both x and y dimensions.
3. Target placement must accommodate the target width and spacing of the underlying wires whose migration will follow in a later step.

Placement migration takes place by reversely topologically ordering the masters according to the hierarchy tree. Masters whose instances reside at tree's leaves are migrated first, while the whole layout, corresponding to the top master, is migrated last. This is accomplished by the depth first search (DFS) shown in Fig. 5. It follows by the DFS and the hierarchical structure of the chip that once the placement migration of a master is addressed, all its descendants in the hierarchy tree have already been migrated, and their width and height are known (elaborated below).

Setting the target dimensions of the currently migrated master by the function work called within the DFS is done by placing its child instances in the same relative positions in the source layout, as illustrated in Fig. 4c. Note that the instances' masters are already migrated, thus having target dimensions defined.

We further elaborate the function migrate_placement of Fig. 5, which is the heart of placement migration. Let a master m have n son instanced placed over its area. We wish to set m target size as small as possible, and legally place its entire son instances in their target size, so it will accommodate all its interconnections with their widths and spacing specifications. As noted before, sons' masters have already been migrated and hence their target size is already determined.

The placement takes place separately for x and y. A weighted digraph G_x (V, E), shown in Fig. 6 is defined as follows. Its source vertex v_0 and target vertex v_{n+1} correspond respectively to the left and right borders of m, respectively. The other vertices correspond to the son instances. Two vertices v_i and v_j are connected with an arc $e_{ij} \in E$ pointing from v_i to v_j if the corresponding sons are visible to each other, no other cell is placed in between, and v_i is placed left to v_j. We consider the left and right border of the parent similarly as son instances.

The determination of an arc's weight $w(e_{ij})$ is delicate. It should reflect the minimum distance required between the origins of v_i and v_j (lower-left corners) in the target layout. Figure 6 illustrates how the weights are determined. The weight considers the target widths of $m(v_i)$ and the target space between v_i and v_j. The latter must account for the existing space between v_i and v_j in the source layout, the wires passing between the blocks, their specified target widths and their estimated target spacing. Spaces overlaid by high wire density in layers of poor technology scaling (larger than 0.7) use factor 0.8 or higher, while for good technology scaling (smaller than 0.7) factor 0.6 is used. The resulting scaling factors of masters' dimensions may vary from 0.5 to 1.0, depending on their contents. This can be seen by comparing Fig. 4a, c. While some of the masters scaled better than 0.7, a few others got worse than 0.7. The x -position of an instance within its parent is determined by the longest paths from v_0. y -position is determined similarly by defining an appropriate G_y (V, E) graph. Figure 7 shows the outcome of the placement migration of a register-file block, where the illustration of the target layout (b) was scaled up to emphasize its similarity to the source layout in (a). The cell borders of all hierarchies are shown to illustrate the similarity of the target to the source placement.

2.2 The interaction between placement and routing migration

As shown in Fig. 4a, wires of the same metal layer are distributed across different levels of the hierarchy, causing "blindness" of physically adjacent wires belonging to different hierarchy levels. Since the determination of masters' target sizes takes place bottom-up in reversed topological order, when a child master is migrated it is not aware of other overlapping wires belonging to its ancestors. More severely, instances of the same master may interact differently with wires of ancestors. Such blindness is resolved in Sect. 3, provided that the dimensions of the master have been properly set to accommodate all the overlaying wires incurred across all its

```
function DFS_migration_traversal (m : source_master)
  If m is migrated return ;
  For each son_instance v_i ∈ m
    Get source_master m(v_i) of v_i ;
    Do DFS_migration_traversal (m(v_i)) ;
  Do migrate_placement (m) ;
```

Fig. 5 Pseudo code of placement migration

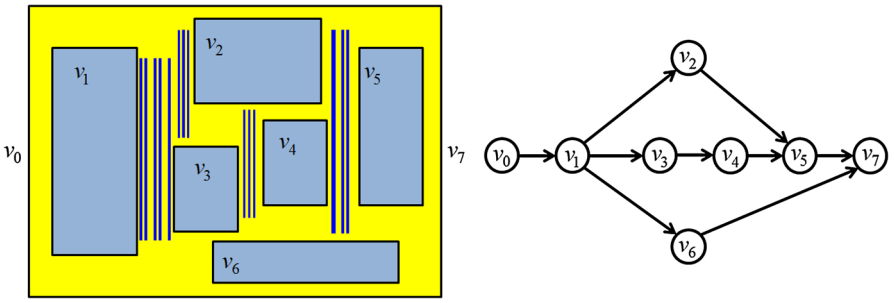


Fig. 6 Adjacency graph for placement migration

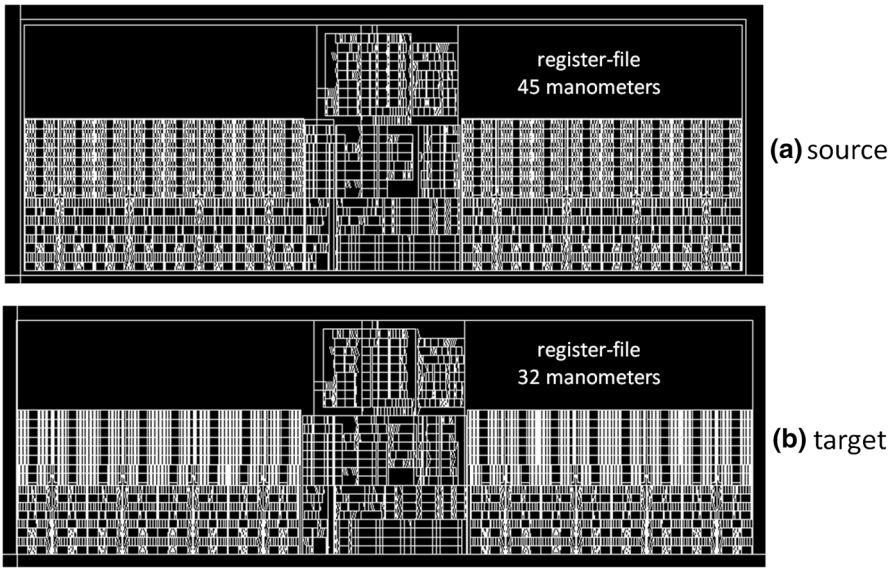


Fig. 7 Placement migration of a register-file block

instances in the layout. The latter is the role of the placement migration described before.

Figure 8 shows the complete migration flow and the placement–routing interaction. It is supplemented with manual artwork by layout design experts when automation fails to legally complete the migration. Placements migration takes place first, followed by routing migration. There, the wire contents of the physical metal layers are migrated layer by layer, taking into consideration delay, power and other performance constraints. For instance, the wires involved in critical delay paths are enforced to a certain width and spacing to reduce their resistance and capacitance. For noise sensitive nets, extra spacing to adjacent wires are enforced to reduce cross-coupling interference. Given a layer, the wires of each master are migrated in masters' reversed topological order. If it happens that a target master is found to be too small to accommodate the constraints imposed by its internal wires, non-feasibility is alerted. A few options exist to remedy. The size of the problematic master and the corresponding ancestors are relaxed, the width and spacing constraints of the problematic wires are modified, or the violations (e.g. wire shorts) are manually fixed.

3 Routing migration

Interconnect migration is the most difficult and time consuming part of the flow. The formal proofs of the subsequent arguments were elaborated in (Shaphir et al. 2013). Interconnect migration takes place layer by layer as shown in Fig. 8. Two vertical wires are said to be visible if a horizontal line can be used to connect them without being interfered by other wires. Visibility implies a planar digraph $G = (V, E)$, as shown in Fig. 9. Once wires' widths and spacing are specified, their abscissa can be computed by the longest path in G (Lengauer 1990). A longest path is used by the compaction algorithm to detect positive cycles resulting from non-feasibility.

The input of the compaction algorithm consists of:

1. The layout hierarchy and relative block positions as defined by the source (old) layout.
2. The visibility graph G and target wires' widths and spacing specifications.
3. The blocks' target sizes as determined by the placement.

The compaction aims at setting the abscissae of the wires within their blocks in the target layout to satisfy the following constraints.

1. The layout of a master must be identical across all its instances in the target layout, hence uniquely defined.
2. The left-to-right order of the instances is preserved across the entire layout hierarchy.
3. The widths of the blocks in the target layout must accommodate their descendant blocks and wires.
4. The visibility graphs are preserved.
5. Satisfy the wires' spacing requirements in the target layout.

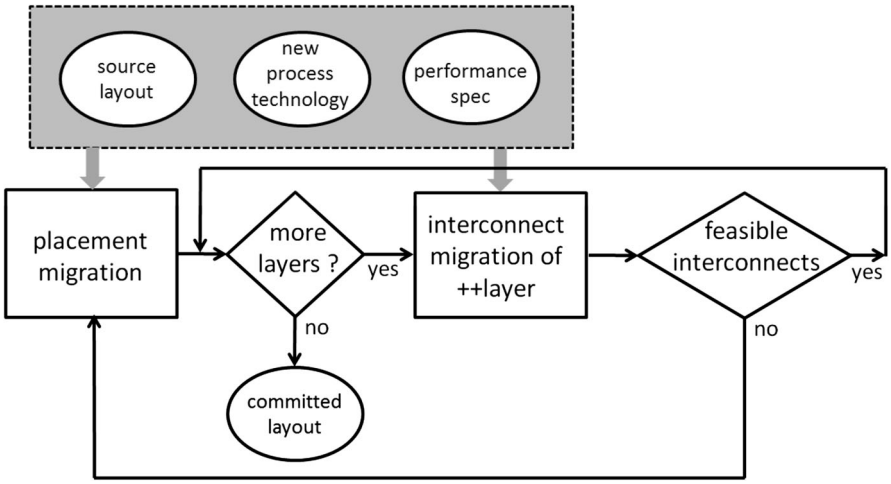


Fig. 8 Complete migration flow

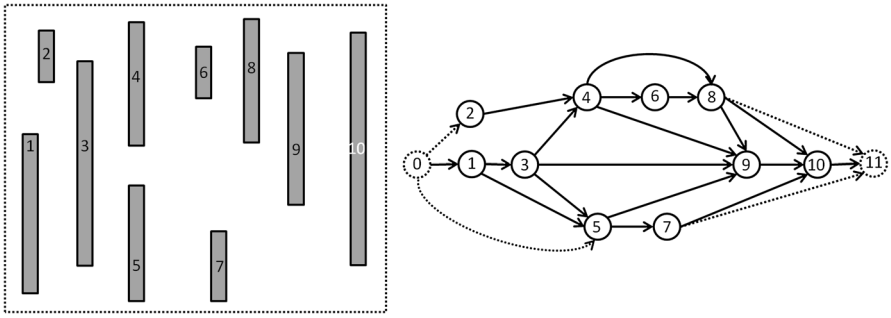


Fig. 9 Wires positioned within a block and their visibility graph

Figure 10a shows the steps of the wires' compaction algorithm.

1. Construction of the visibility graph of the entire layout. Multiple instances of the same master within a parent block are registered.
2. *Contracting* the visibility graph by *merging* the multiple instances of a wire into a single object. *If the merged graph M is free of positive cycles then a feasible compaction solution exists.*
3. Defining a series of concise (called *reduced*) graphs $\{R_i\}_1^N$ which capture all the essential information of the fully expanded graph. The reduced graphs are then solved successively. A solution is a setting of nonnegative weights to the graph's arcs such that the above constraints are satisfied.
4. The solutions obtained in step 3 satisfy the constraints within each block, thus representing a family of feasible solutions. Those are floated to their parents and are used to find the family of feasible solutions for positioning the wires of the parent master.

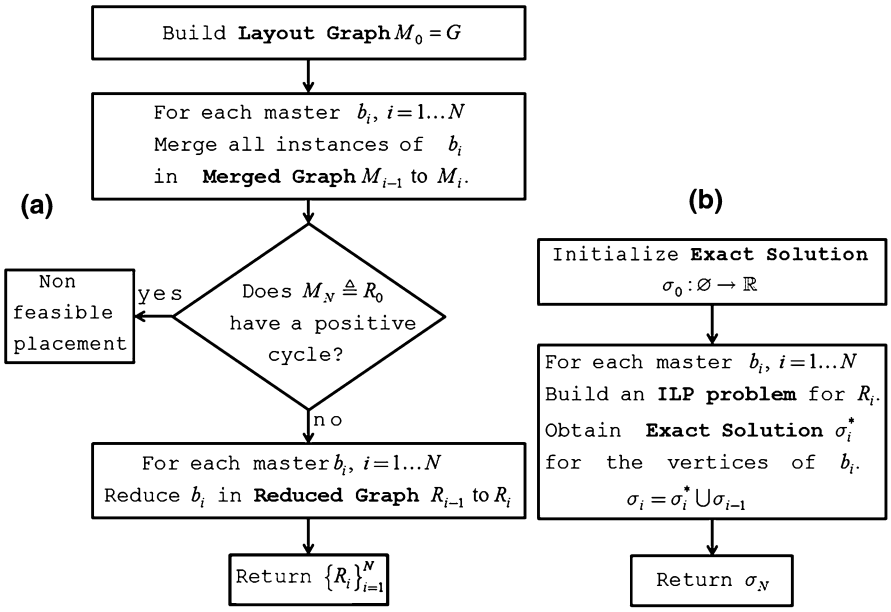


Fig. 10 Finding of the feasibility ranges of wires' abscissae (**a**) and committing for wires' abscissae (**b**)

A feasible solution of wires' abscissae within a master is a set of closed intervals obtained by the flow shown in Fig. 10a. Each wire can be positioned anywhere in its corresponding interval and the resulting layout is guaranteed to be legal. The final abscissae of the wires are determined by solving an ILP problem. Its constraints are the above intervals. Applying this process progressively to each master from top to bottom (in reversed topological order) as shown in Fig. 10b, yields the entire layout.

The ILP's objective function is defined as a weighted sum of wire spacing in an attempt to maintain the delays of the critical nets sufficiently small. It is well known that the coupling capacitance between adjacent wires is a predominant delay, power and noise factor, inversely proportional to the space between wires (Moiseev et al. 2009). Nets are assigned with positive coefficients, reflecting their relative delay criticality. Power consumption can also be minimized by weighting reflecting nets' switching activity. The weighted sum of spaces is then proportional to the switching power of interconnects. Nets sensitive to noise can similarly be treated.

3.1 Construction of the flat layout visibility graph

Let V be the wires and borders of a layout L. For $v \in V$, $I(v)$ denotes the instance to which v belongs, $m(v)$ denotes the corresponding object (wire or border) of the master, $T(I)$ is the instance's master, and $D(T(I))$ is its width in target technology. The parent master of an instance I is $P(I)$. A *Flat Layout Visibility Graph* (FLVG) $G = (V, E)$ is defined, satisfying the properties below.

1. The vertices V are all the wire and border objects in L. We subsequently use the term vertex and object interchangeably.
2. A pair of wires $u,v \in V$ visible to each other implies an arc $(u,v) \in E$ with a weight $W(u,v)$ defined by the sum of three terms: half target width of u, the required wire-to-wire spacing of the new technology, and half target width of v.
3. A pair of vertices $u,v \in V$, where u and v are a left border and a wire of I, respectively, implies an arc $(u,v) \in E$ with a weight $W(u,v)$ defined by the sum of half wire's target width and half of the minimum wire-to-wire spacing of the new technology. It ensures that minimum spacing design rule is satisfied for adjacent wires belonging to two abutting instances.
4. Similar to 3 but u is a wire and v is a right border.
5. Right and left borders u and v, respectively, visible to each other, satisfying $P(I(u)) = P(I(v))$and $I(u) \neq I(v)$, imply an arc $(u,v) \in E$ with a weight $W(u,v) = 0$. Its role is to preserve the left-to-right instances order and avoid their overlap.

The next property enforces the size of an instance to the size of its master.

6. Left border u and right border v, satisfying $I(u) = I(v)$, imply two oppositely directed arcs $(u,v) \in E$ and $(v,u) \in E$ with weights $W(u,v) = D(T(I(u)))$ and $W(v,u) = -D(T(I(u)))$.

The next two properties enforce an instance to entirely reside within the area of its parent.

7. Left borders u and v satisfying $T(I(u)) = P(I(u))$ imply an arc $(u,v) \in E$ with a weight $W(u, v) = 0$.
8. Right borders u and v satisfying $P(I(u)) = T(I(v))$ imply an arc $(u,v) \in E$ with a weight $W(u, v) = 0$.

Figure 11a illustrates a parent master A comprising two child instances of the same master B. A's wires are colored in green and those of B in blue. From A's perspective the left-to-right order of all the wires (both blue and green) must be preserved. Also, the green wires cannot extend beyond A's border, whose target size has already been determined at placement migration. Similar requirements hold for the blue wires in B. Furthermore, relative position of the blue wires in B must be identical in both instances, since B is unique. Figure 11b is the corresponding FLVG. The vertical borders are represented by gray vertices, designated by a master name indexed L or R for the left and right borders, respectively. The oppositely directed parallel red arcs enforce the master size in the target layout.

3.2 Merging block instances and the corresponding graph

The second step of the compaction algorithm transforms the FLVG G into a simpler graph M obtained by merging vertices corresponding to the same wire across the entire hierarchy. This step also adds arcs imposing similarity constraints to ensure uniqueness of the master in the target layout. Let the layout comprise N masters.

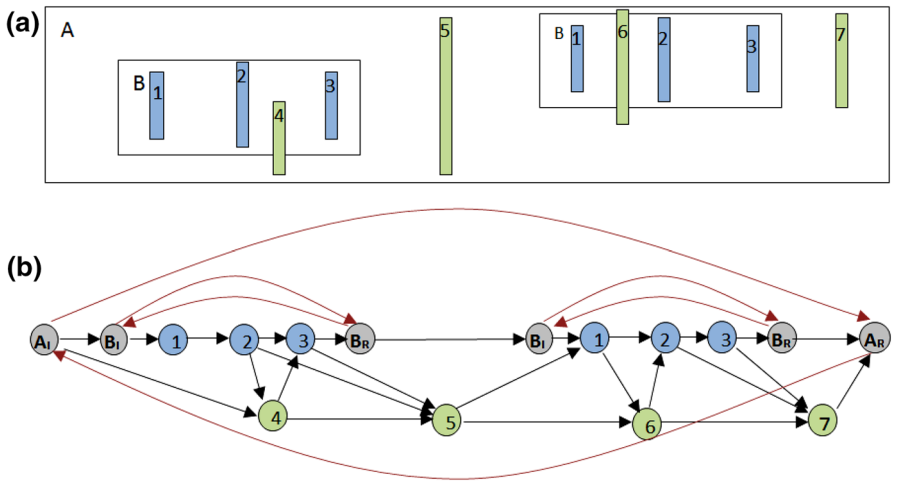


Fig. 11 Two-level hierarchical layout (**a**) and its corresponding FLVG in (**b**)

M is successively constructed in N steps. We say that two masters A and B satisfy a *partial hierarchical order* $B \prec A$, if B is a descendant of A. The hierarchical layout can then be represented by tree as illustrated in Fig. 12.

The construction of the graph M takes place bottom-up. The masters b_i, $1 \leq i \leq N$, are indexed by their partial order $\prec$. A corresponding sequence of merged graphs M_i $1 \leq i \leq N$, is implied, where $M_0 \triangleq G$. M_i is derived from M_{i-1} by merging all the vertices corresponding to the instances of the same wire or border of b_i in a single vertex, to ensures uniqueness of the target master. The weights of the incoming and outgoing arcs of that vertex are updated according to the offset of the instance to which the vertex belongs. The contraction significantly reduces the number of vertices while capturing the complete spacing and size constraints. Figure 13 is the merging pseudo code.

The procedure is working iteratively in a reversed topological order (bottom-up) of the masters (see Fig. 10a). Let $G_t = (V_t, E_t)$ denote the flattened graph of a master t. All the vertices $v \in V$ induced by t's instances across the entire layout are replaced by a vertex $v_t \in V_t$. Each incoming arc (u,v) (outgoing arc (v,u)) is replaced by an arc (u,v_t) (v_t,u) and its weight is modified by subtracting (adding) the offset of the instance to which v belongs. We denote by M the graph resulting by the iterative merging transformations. G is free of positive cycles, namely, the compaction has a feasible solution, if and only if M does. The number of vertices in M is significantly reduced compared to G by a factor related to the average number of master's instances. Figure 14 illustrates the result of merging the graph in Fig. 11. The time complexity of the merging is $O(|V| + |E|)$, as each vertex and arc in G is treated exactly once.

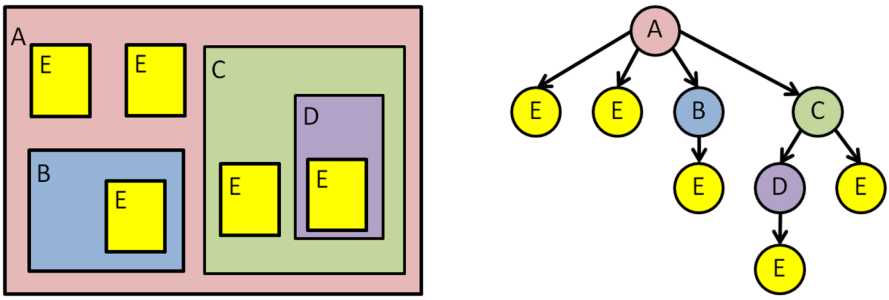


Fig. 12 Hierarchical layout and its corresponding tree

proc **mergeMasterInstances** (graph: G , master: t)
Let $G=(V,E)$;
Let $G_t=(V_t,E_t)$ be the flat graph of master t ;
$V \leftarrow V \cup V_t$;
For each instance i of t in the layout
Let x be the offset of i in its parent ;
For each $v_t \in V_t$
Let $v \in V$ represent v_t in instance i ;
For each $u \in V$ so that $(u,v) \in E$
$E \leftarrow E \cup \{(u,v_t)\}$; $W(u,v_t) \leftarrow W(u,v) - x$; $E \leftarrow E \setminus \{(u,v)\}$;
For each $u \in V$ so that $(v,u) \in E$
$E \leftarrow E \cup \{(v_t,u)\}$; $W(v_t,u) \leftarrow W(v,u) + x$; $E \leftarrow E \setminus \{(v,u)\}$;
$V \leftarrow V \setminus \{v\}$;

Fig. 13 Pseudo code for vertex merging

3.3 Graph reduction

While number of vertices has been considerably reduced, we subsequently reduce the number of arcs too. Reduction is accomplished successively by transforming M in reversed topological order (bottom-up) of the masters. Iteration eliminates the vertices corresponding to the current master. The topological ordered $\prec$ defines the sequence of reduced graphs $R_i, 1 \leq i \leq N$, where $R_0 \triangleq M$. R_i is obtained from R_{i-1}. Let v be a vertex corresponding to a wire or the vertical border of b_i. An incoming arc (u,v) and an outgoing arc (v,w) are replaced by an arc (u,w) satisfying $W(u,w) = W(u,v) + W(v,w)$, thus eliminating the vertex v. If an arc (u,w) already

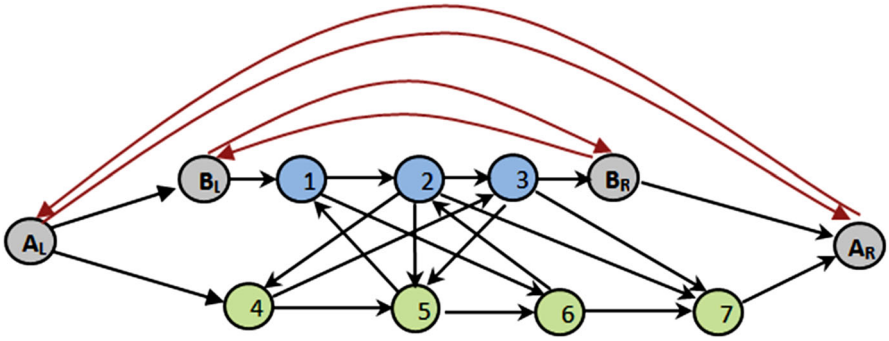


Fig. 14 The contracted graph resulting after merging the graph in Fig. 11

exists in R_i the weight of the new arc (u,w) is set to $W(u,w) \leftarrow \max\{W(u,w), W(u,v)+W(v,w)\}$. Figure 15 illustrates the reduced graph obtained from the merged graph in Fig. 14. It follows from the planarity of the FVLG that the worst-case time complexity of the reduction phase is $O(|V|)$. A pseudo code of the graph reduction is shown in Fig. 16.

3.4 Deriving an exact solution to commit wires' positions

It was claimed that the FLVG G has no positive cycle if and only if the corresponding merged graph M does. Assuming that M is such, it is subsequently described how wires' exact locations are determined by successive solutions of small ILP for each of the masters, for which a commercial solver was used. Unlike the merging and reduction phases which took place in reversed topological order (bottom-up) of the masters, the commitment of exact locations takes place in direct topological order (top-down). This is the primary advantage of the algorithm. While past hierarchical compactors worked on the entire flattened layout, thus solving a huge LP or ILP, this algorithm solves a series of far smaller ILPs, while exploring exactly the same solution space as the "flattened" ILP does.

The ILP problems are solved for the masters $b_i, 1 \leq i \leq N$, from b_N (root) down to b_1 (leaf). After the ILP problem for b_{N-j} has been solved, the exact locations of the wires (values of ILP variables) in each of the masters $b_N, b_{N-1}, \ldots, b_{N-j}$ are determined. Recall that b_{N-j-1} implied a corresponding reduced graph b_{N-j-1}. Therefore, only those variables (wires locations) related to b_{N-j-1} are left to be determined, where a feasible solution is guaranteed. Appropriate pseudo code is shown in Fig. 17. For the special case of deriving a solution for the root master b_N we define a reduced graph $R_{N+1} \triangleq (V_{N+1}, E_{N+1})$, where V_{N+1} is the set of vertices that represent the borders of b_N, and $E_{N+1} = \emptyset$. Note that $\sigma(v)$ is determined for V_{N+1} since the borders of b_N are known.

An arc of R_{i-1} implies a constraint of the ILP problem. Since the exact locations of V_i's vertices have already been determined, only the arcs of $E_{i-1} \backslash E_i$ are of interest. An arc (u,v) implies a constraint of the ILP if the exact location of both vertices have not yet been determined, i.e., both vertices belong to $V_{i-1} \backslash V_i$. Each

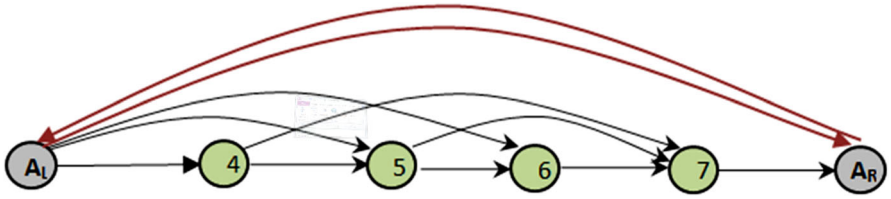


Fig. 15 The graph obtained by reducing the graph in Fig. 13

proc **reduceMasterInstances** (graph: G , master: t)
Let $G=(V,E)$;
Let $G_t=(V_t,E_t)$ be the flat graph of master t ;
$V' \leftarrow V$; $E' \leftarrow E$;
For each $v_t \in V_t$
Let $v \in V$ represent v_t in G ;
For each $u \in V$ so that $(u,v) \in E$
For each $w \in V$ so that $(v,w) \in E$
$E' \leftarrow E' \cup \{(u,w)\}$; $W(u,w) \leftarrow \max\{W(u,v)+W(v,w), W(u,w)\}$;
For each $u \in V$ so that $(u,v) \in E$ $E' \leftarrow E' \setminus \{(u,v)\}$;
For each $w \in V$ so that $(v,w) \in E$ $E' \leftarrow E' \setminus \{(v,w)\}$;
$V' \leftarrow V' \setminus \{v\}$;
return (V',E') ;

Fig. 16 A pseudo code of graph reduction

vertex is then represented by an ILP variable. If the exact location of one vertex has already been determined, there is no corresponding ILP variable and the exact location σ is used.

Figure 18 illustrates the relation between a reduced graph and its corresponding ILP. The green vertices represent the wires of master A, whose locations have already been determined upon the solution of the ILP corresponding to R_A. Their committed locations are specified next to the vertices. The positions of the gray vertices, representing vertical borders, have also been determined, since their vertices and the parallel arcs enforcing A's size (as determined by placement), also exist in R_A. The reduced graph R_B is illustrated in (a), for which the ILP variables are those corresponding to the wires of master B. The constraints imposed on arc lengths are translated into the inequalities in (b).

Let $G=(V,E)$; $R_i=(V_i,E_i)$; $R_{i-1}=(V_{i-1},E_{i-1})$;
Let $\sigma: V \to \mathbb{R}$;
For each $(u,v) \in E_{i-1} \setminus E_i$
  Let x_v and x_u be the ILP variables corresponding to u and v, respectively ;
  If $u,v \in V_{i-1} \setminus V_i$ add an ILP constraint $x_v - x_u \geq W(u,v)$;
  Else if $u \in V_{i-1} \setminus V_i$ add an ILP constraint $\sigma(v) - x_u \geq W(u,v)$;
  Else add an ILP constraint $x_v - \sigma(u) \geq W(u,v)$;
Invoke ILP solver ;
For each $v \in V_{i-1} \setminus V_i$ $\sigma(v) \leftarrow$ ILPsolverSolution(v) ;

Fig. 17 An ILP solution for positioning the wires of a master

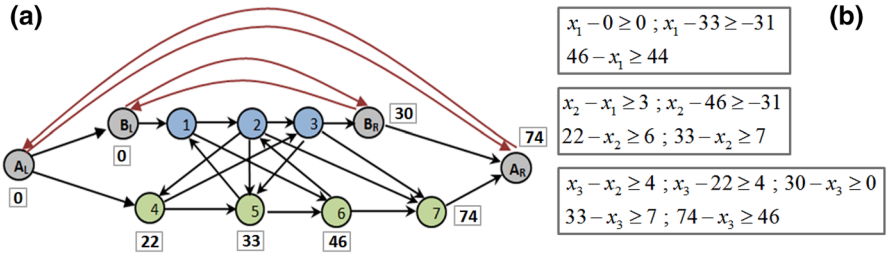


Fig. 18 A reduced graph is in (**a**) and the corresponding ILP in (**b**)

4 Experimental results

The following results have been obtained for Intel's 65 nm process technology (Tock) microprocessors, branded as Core 2 Duo (see Tock in Fig. 1), where the interconnecting wires have been migrated into 45 nm process technology (Tick). The examples below incorporate the results for several blocks, each comprising thousands of nets. The quality of such migration is determined by the performance of the underlying circuits in the target layout. The migration targeted delay reduction of 0.7, which is X1.4 clock frequency speedup.

Wire widths and line-to-line spacing specifications have been derived from the Elmore delay model, based on the electrical parameters of the 45 nm technology. The positive cycles discovered by the algorithm have been resolved by reducing wire widths and spaces of vertices and arcs involved in the positive cycles. Although such relaxations resolves the problems and legalized the positions of wires, delay violations due to resistance and coupling (line-to-line) capacitance increase may occur. Those are later resolved by VLSI design techniques such as using stronger circuit drive or re-timing (Weste and Harris 2005).

Figure 19 depicts how the target-to-source delay ratio, measured by Elmore delay model, is scattered around 0.7. As shown, some of the nets did not meet that goal, having higher ratio. Fortunately, the delays of the majority of those are much smaller than the target clock cycle, so they do not impose any problem. Only those encircled required further treatment in the design to avoid timing violations. Here lays the major advantage of automatic layout migration; it delivers satisfactory performance for the majority of the interconnects, leaving a relatively small percentage (less than 10 %) for further fix-up by engineering effort.

The experimental results for a set of nine blocks and are summarized in Table 1. For each block the number of vertices and arcs in its corresponding FVLG are specified. Next are shown the number of vertices and arcs in the merged graph. The computational efficiency is demonstrated by the column specifying the largest ILP problem incurred by the series of the graph reductions. The ILP column shows reductions of the problem sizes by one to two orders of magnitude compared to the flattened layout. The computational efficiency premise is also shown in the last column. To grasp the runtime that could be obtained by using the algorithms in (Burns and Newton 1987; Burns and Feldman 1998; Yao et al. 1993; Wang and Lai 2001; Fang and Zhu 2004), the step of contracting of all the instances of a master into a single node was skipped. This effectively turns the algorithm to behave similarly to those mentioned in the introduction. A runtime speedup of one to two orders of magnitude is shown.

Figure 20 illustrates the area scaling distribution of all 601 master cells comprising the block of experiment 9. The area scaling of the top-level block is specified for each experiment. By Moore's Law 0.5 area scaling could be expected, but as mentioned earlier, scaling factors vary in a wide range.

5 Conclusions

A cell-based layout migration algorithm for hard-IP reuse of high-end VLSI processors was presented. It took advantage of the natural separation of the physical

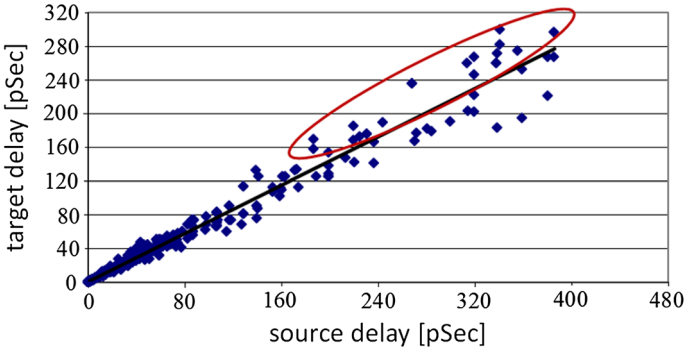


Fig. 19 Delay results of net in target layout versus source layout

Table 1 Tests statistics—size of graphs, area scaling and runtime comparisons

Test No.	FLVG vertices	FLVG arcs	Merged graph vertices	Merged graph arcs	Variables in largest ILP problem	Reduced to FLVG ratio	Area scaling	Hierarchical to flattened runtime ratio
1	1,511	5,755	499	2,249	302	0.2	0.44	0.102
2	2,064	8,456	1,687	7,130	598	0.29	0.49	0.125
3	4,457	16,890	1,253	6,101	434	0.098	0.46	0.125
4	12,928	55,892	2,535	19,524	386	0.03	0.50	0.096
5	23,020	106,554	4,683	40,334	1,561	0.0,678	0.55	0.075
6	29,714	109,212	4,205	16,255	1,173	0.0395	0.47	0.048
7	45,964	177,946	5,963	24,928	1,308	0.0284	0.50	0.034
8	62,944	248,915	7,850	44,963	1,277	0.0202	0.58	0.049
9	73,005	294,479	9,455	63,737	1,420	0.0194	0.54	0.033

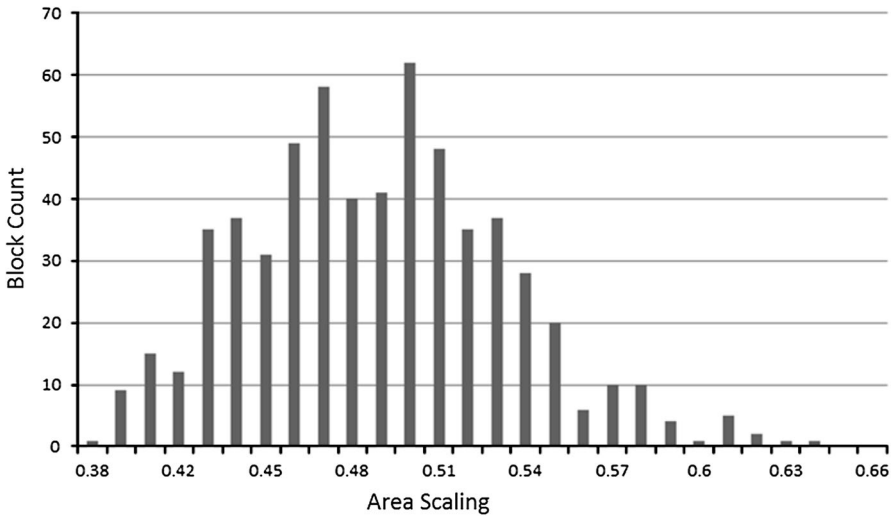


Fig. 20 Area scaling distribution of master cells

layout generation into placement and routing phases. Hierarchical-based algorithms have been developed and combined in a design flow that enabled the migration of full chips across several process technology generations. The algorithm takes full advantage of the inherent hierarchy built into VLSI designs. By applying a series of transformations, the underlying computational problems have been considerably reduced by one to two orders of magnitude, making the solution of large problems feasible.

Although compaction algorithms were developed at the early EDA days for simple design rules and linear technology scaling, compaction approach is still useful for today's nanometer-scale design rules. The enforcement of discrete sizes requires the usage of discrete optimization methods such as ILP. Another possible

solution is to apply a dynamic programming, as done in (Moiseev et al. 2010) for flat layouts. This still requires further development to support hierarchy. There may also be wires that will have no solution, so those must be left for later manual fixes.

Acknowledgments The authors are thankful to Intel Corporation for supporting this work. They are also thankful for the useful reviewers' comments, which helped improving the manuscript.

References

Burns JL, Feldman JA (1998) C5M—a control-logic layout synthesis system for high-performance microprocessors. IEEE Trans CAD Integr Circuits Syst 17(1):14–23

Burns JL, Newton AR (1987) Efficient constraint generation for hierarchical compaction. In: IEEE international conference on computer design, pp. 197–200

Chiang S-Y (2001) Foundries and the dawn of an open IP era. Computer 34(4):43–46

Fang F, Zhu J (2004) Automatic process migration of data-path hard IP libraries. In: Proceedings of the 2004 Asia and South Pacific design automation conference. IEEE Press, pp. 887–892

Intel's Tick-Tock Model, http://www.intel.com/technology/tick-tock/index.htm

Klau G, Mutzel P (1999) Optimal compaction of orthogonal grid drawings. Lecture notes in computer science: integer programming and combinatorial optimization, vol 1610. Springer, Berlin, pp 304–319

Lengauer T (1984) On the solution of inequality systems relevant to IC-layout. J Algorithms 5(3):408–421

Lengauer T (1990) Compaction. In: Combinatorial algorithms for integrated circuit layout. Wiley, Chichester, pp 579–643

Moiseev K, Kolodny A, Wimer S (2009) Power-delay optimization in VLSI microprocessors by wire spacing. ACM Trans Des Autom Electron Syst 14(4):1–28 Article No. 55

Moiseev K, Kolodny A, Wimer S (2010) Interconnect bundle sizing under discrete design rules. IEEE Trans CAD Integr Circuits Syst 29(10):1650–1654

Nitzan R, Wimer S (2002) AMPS and SiClone integration for implementing 0.18 um to 0.13 um design migration. In: Proceedings of the Synoposys Users Group (SNUG) Conference, San Jose CA, March 2002

Patrignani M (2001) On the complexity of orthogonal compaction. Comput Geom 19(1):47–67

Perry TS (2008) Gordon Moore's next act. IEEE Spectr 45(5):38–47

Reinhardt M (2002) Automatic layout modification: including design reuse of the alpha cpu in 0.13 micron soi technology. Kluwer, Norwell, MA, USA

Saleh R et al (2006) System-on-chip: reuse and integration. Proc IEEE 94(6):1050–1069

Sastry S, Parker A (1982) The complexity of two dimensional compaction of VLSI layouts. In: Proceedings of international conference on circuits and computers, pp 402–406

Schlag M, Liao YZ, Wong CK (1983) An algorithm for optimal two-dimensional compaction of VLSI layouts. Integration 1(2–3):179–209

Shaphir E, Pinter RY, Wimer S (2013) Cell-based interconnect migration by hierarchical optimization. Integration. doi:10.1016/j.vlsi.2013.10.003

Shin H, Sangiovanni-Vincentelli AL, Siquin CH (1986) Two-dimensional compaction by zone refining. In: Proceeding of the 23rd ACM/IEEE design automation conference, pp 115–122

Simultaneous, n-level hierarchical layout compaction, process migration and physical optimization. http://www.sagantec.com/SiClone.pdf

Wang L-Y, Lai Y-T (2001) Graph-theory-based simplex algorithm for VLSI layout spacing problems with multiple variables constraints. IEEE Trans CAD Integr Circuits Syst 20(8):967–979

Weste N, Harris D (2005) CMOS VLSI design: a circuits and systems perspective. Pearson, Addison-Wesley

Wimer S (2014) Planar CMOS to multi-gate layout conversion for maximal fin utilization. Integration 47:115–122

Yao S-Z, Cheng C-K, Dutt D, Nahar S, Lo C-Y (1993) Cell-based hierarchical pitchmatching compaction using minimal LP. In: Proceedings of the 30th international design automation conference, ACM, pp 395–400