Area-Optimal Transistor Folding for 1-D Gridded Cell Design

Jordi Cortadella, *Member, IEEE*

***Abstract*—The 1-D design style with gridded design rules is gaining ground for addressing the printability issues in sub-wavelength photolithography. One of the synthesis problems in cell generation is transistor folding, which consists of breaking large transistors into smaller ones (legs) that can be placed in the active area of the cell. In the 1-D style, diffusion sharing between differently sized transistors is not allowed, thus implying a significant area overhead when active areas with different sizes are required. This paper presents a new formulation of the transistor folding problem in the context of 1-D design style and a mathematical model that delivers area-optimal solutions. The mathematical model can be customized for different variants of the problem, considering flexible transistor sizes and multiple-height cells. An innovative feature of the method is that area optimality can be guaranteed without calculating the actual location of the transistors. The model can also be enhanced to deliver solutions with good routability properties.**

***Index Terms*—Cell generation, design for manufacturability, linear programming, transistor folding, transistor sizing.**

I. Introduction

THE SCALING of transistor dimensions and the manufacturing challenges involved in the subwavelength optical lithography impose severe constraints on the layout patterns that can be reliably printed on the wafers.

According to several authors, the 1-D design style with gridded design rules (GDRs) is one of the principal trends toward addressing the manufacturing issues in future process technologies [8], [12], [18], [22], [23]. In 1-D GDRs, layout is composed of grating patterns with rectangular shapes located on a grid with fixed pitch.

An interesting study on different layout styles is presented in [6], where the impact on area, yield, and variability is studied. The 1-D style offers better yield and smaller variability than the 2-D style with nonrectangular shapes. The best style to minimize standard cell area seems to be the 1-D, although this requires a larger utilization of the M2 layer.

For standard cell design, 1-D style implies an underlying active area with equally spaced transistors and unidirectional metal layers routed on gridded layouts [17]. In this context, cell design is a problem that is moved from the continuous domain (any type of shape, any location) to the discrete domain (only rectangular shapes on a coarse grid with fixed pitch). Thus, cell synthesis becomes a combinatorial problem in which EDA algorithms can do a much better job than manual design.

Area is a critical resource that still needs to be minimized for cost-efficient manufacturability. The height of a cell depends on the number of tracks used for the active area, whereas the width is determined by the number of devices and the diffusion breaks inserted to isolate transistor chains.

Minimum-area cells are synthesized by finding good transistor orderings that allow to maximize diffusion sharing. The algorithms proposed to find these orderings are tightly related to the theory of finding Eulerian paths in undirected graphs. Several theoretical results and algorithms have been proposed to find optimal transistor orderings, either considering fixed transistor netlists [16], [20] or allowing transistor reordering of series–parallel graphs while preserving the functionality of the cells [13], [14]. Large transistors may exceed the maximum allowable size in a standard cell. This problem is solved by breaking large transistors into smaller ones (legs). For example, a transistor that needs seven tracks of active area may be implemented with three legs of 3+2+2 or 3+3+1 tracks. The strategy of creating multiple legs of the same transistor is called transistor folding.

A. Simple Example

Fig. 1 depicts the FEOL layers of three different implementations of an AND2 gate using multiple legs to implement large devices. Table I reports the characteristics of the devices. The second column specifies an interval of sizes (tracks) allowed for each device. For example, device `p1` can have a size between eight and 10 tracks.

In the example, the maximum size for p and n devices is four and three tracks, respectively. Double-height cells can also be designed as shown in Fig. 1(b). The two p strips can potentially be merged to extend the maximum size of the p devices, as shown in Fig. 1(c). Hybrid approaches having segments with two p strips and segments with one merged strip are also possible in double-height cells. The three layouts shown in Fig. 1 are area-optimal in each category (single height, double height, and double height with p-diffusion merging).

Diffusion strips may be interrupted for different reasons (see Fig. 2). When no Eulerian paths are found, diffusion breaks

Manuscript received November 19, 2012; revised April 13, 2013; accepted June 9, 2013. Date of current version October 16, 2013. This work was supported by a gift from the Intel Corporation, the project FORMALISM (CICYT TIN2007-66523), and the Generalitat de Catalunya (ALBCOM-SGR 2009-2013). This paper was recommended by Associate Editor G.-J. Nam.

The author is with the Department of Software, Universitat Politècnica de Catalunya, Barcelona 08034, Spain (e-mail: jordi.cortadella@upc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2269680

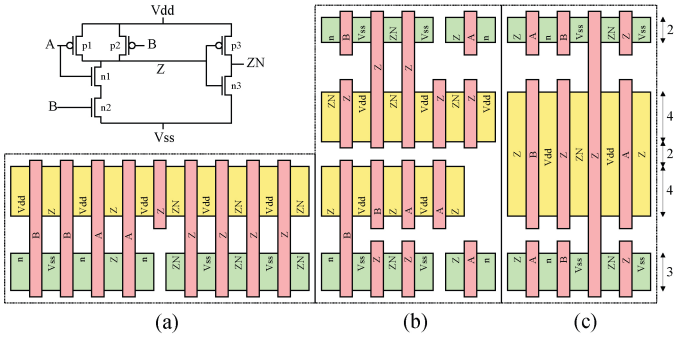


Fig. 1. Several symbolic layouts of the FEOL layers for an AND2 gate: (a) single height, (b) double height, (c) double height with merged p strips.

TABLE I
DEVICES (LEGS × SIZE) FOR THE LAYOUTS IN FIG. 1

Device	Size interval	Single height (a)	Double height (b)	Merged p-diffusion (c)
p1	$[8, 10]$	2×4	2×4	1×10
p2	$[8, 10]$	2×4	2×4	1×10
p3	$[18, 20]$	5×4	5×4	2×10
n1	$[5, 6]$	2×3	$1 \times 2 + 1 \times 3$	$1 \times 2 + 1 \times 3$
n2	$[5, 6]$	2×3	$1 \times 2 + 1 \times 3$	$1 \times 2 + 1 \times 3$
n3	$[10, 12]$	4×3	$2 \times 2 + 2 \times 3$	$2 \times 2 + 2 \times 3$

must be introduced to isolate devices. One strategy is to use isolation transistors (permanently off) by connecting the gate to Vdd or Vss, as shown in Fig. 2(a).

A row may also contain blocks of devices with different size. With 1-D GDRs, no diffusion sharing with differently sized transistors is allowed, since this would imply nonrectangular shapes for the active area, as shown in Fig. 2(b). Diffusion strips must have equally sized transistors and breaks must be introduced between devices with different size. The length of these breaks must be a multiple of the technological pitch, as shown in Fig. 2(c). Depending on the technological pitch, breaks between differently sized transistors may occupy one or two polysilicon slots.

Transistor folding is a combinatorial problem that cannot be simply reduced to minimizing the total number of devices of the cell. The existence of Eulerian paths in the transistor strips and the diffusion breaks required to isolate blocks with different size are crucial in defining the transistor folding strategy for each cell.

B. Previous Work and Contributions of This Paper

The transistor folding problem has been addressed by different authors in the past, either for single-height cells or multiple-height cells.[1]

In [11], an efficient algorithm for folding in single-height cells was proposed. A faster algorithm was later proposed in [4]. In both cases, the algorithms aim at minimizing the product *height* × *width* of the cell, and diffusion sharing is not taken into account. This approach is not realistic for standard cell design, since the height of the cell is defined a priori

[1]The terms 1-D and 2-D are traditionally used to refer to the synthesis of single- and multiple-height cells, respectively. In this paper, we change the nomenclature to avoid any confusion with 1-D and 2-D design rules.

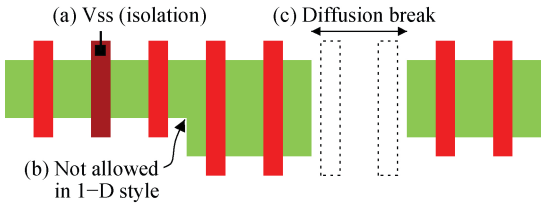


Fig. 2. Legal and illegal diffusion breaks for 1-D GDRs.

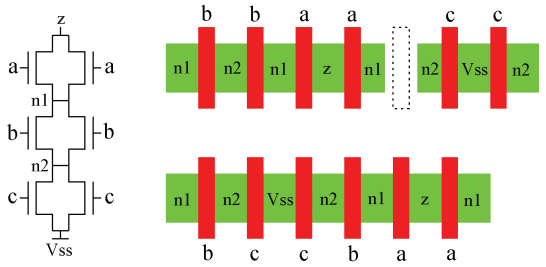


Fig. 3. Impact of adjacent legs in cell area.

for the complete library. Furthermore, diffusion sharing has a significant impact in area, as will be shown in this paper.

Gupta and Hayes [9] identify the interdependence between transistor folding and diffusion sharing. They propose an integer linear programming model for multiple-height cells. However, folding and transistor placement are solved independently, assuming that each transistor is folded with the minimum number of legs allowed by the cell height. This approach also assumes that the legs of each transistor are placed contiguously in the layout. For these reasons, this strategy does not guarantee a minimum-area layout.

An example is shown in Fig. 3, where the pull-down netlist of a NAND3 gate is depicted and each transistor has two legs. By enforcing the legs of the same transistor to be contiguous, a chain such as the one shown in the top can be obtained. This chain has a diffusion break since no Eulerian path can be found.[2] However, no diffusion break is necessary if some of the transistors are allowed to have separated legs, as shown in the chain at the bottom.

Berezowski [2] proposes the first approach in which folding and diffusion sharing are integrated for single-height cells. The approach is based on an extension of the dynamic programming algorithm presented in [1].

All the previous approaches work with the assumption that differently sized transistors can share diffusions, i.e., a 2-D design style. Additionally, the methods are restricted to the placement of pairs of p and n transistors that must be aligned vertically to share the same polysilicon stick. The placement of transistor pairs also involves some area overhead (see, e.g., [6, Sec. II.B]).

This paper presents an exact algorithm that guarantees an area-minimal layout for the transistor folding problem considering different layout parameters: single- and multiple-height cells, parametrized diffusion breaks, flexible transistor sizes, and adaptable diffusion tracks.

[2]By simple enumeration, the reader can easily realize that no Eulerian path exists with the two b gates being adjacent.

An important feature of the approach is that the algorithm does not even deliver any specific transistor ordering. Instead, it generates a netlist for which an area-optimal transistor ordering is guaranteed to exist. In general, different solutions with the same area may exist, thus giving the opportunity for transistor placement tools to explore the best one in terms of routability.

Finally, the algorithm can also incorporate terms in the cost function that, still guaranteeing area optimality, can deliver solutions that have better routability properties.

This paper is organized as follows. Section II presents a graph model for the problem and reviews the relevant Euler's graph theory. Section III proposes the mixed-integer linear programming (MILP) formulation of the problem. A strategy to generate multiple solutions is discussed in Section IV. Several extensions of the model are presented in Section V. A strategy to deliver routability-aware solutions is proposed in Section VI. Section VII describes two heuristics that are compared with the MILP model. Finally, the impact of the proposed methods on area and routability is evaluated in Section VIII. Section IX concludes this paper.

II. Graph Model for the Transistor Folding Problem

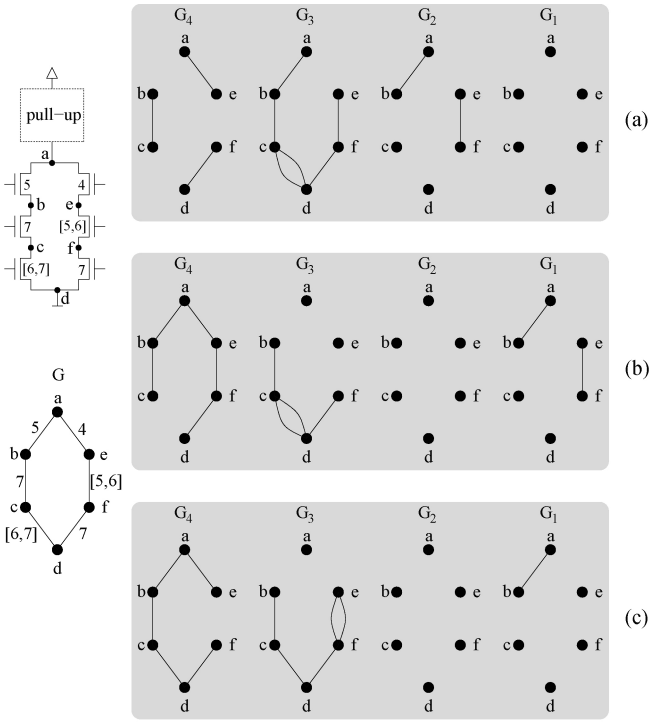
A transistor netlist is represented by an undirected graph $G(N, T)$ where N is the set of nodes, representing source/drain terminals of the transistors, and T is the set of transistors.[3] The gates of the transistors are irrelevant for the folding problem and are not represented in the model. Every transistor $t \in T$ has a target size, denoted by $\text{SIZE}(t)$. In general, the target size may be defined by an interval $[\text{SIZE}_{\min}(t), \text{SIZE}_{\max}(t)]$ of discrete values that represent the acceptable flexibility interval for the number of tracks of t.

We will consider the folding problem for arbitrary transistor netlists (not necessarily static CMOS) in which the rows for p and n devices can be optimized independently.

The folding problem consists of finding an equivalent implementation of the netlist with multiple transistor legs that cannot exceed a maximum size $\mathbb{S}$ and can be implemented with minimum area using an optimal transistor chaining.

The output of the folding algorithm is another netlist for which at least one area-minimal transistor chaining exists. Finding the transistor arrangement with the best routability characteristics is the goal of algorithms for transistor placement (see [1] and [16]) and is out of the scope of this paper.

The model assumes that transistors with different sizes cannot be chained,[4] as it was shown in Fig. 2. Diffusion breaks are required to separate transistors with different sizes and the separation gap is denoted by the constant DIFFSIZEGAP. Sets of transistors with the same size cannot always be chained due to the nonexistence of Eulerian paths. In this case, the diffusion breaks may have a different gap, denoted by the constant SAMESIZEGAP. When using isolation gates, as in Fig. 2(a), SAMESIZEGAP will be 1.



$$a \overset{4}{-} e \sim b \overset{4}{-} c \sim d \overset{4}{-} f \sim a \overset{3}{-} b \overset{3}{-} c \overset{3}{-} d \overset{3}{-} f \overset{3}{-} e \sim c \overset{3}{-} d \sim a \overset{2}{-} b \sim e \overset{2}{-} f$$
$$c \overset{4}{-} b \overset{4}{-} a \overset{4}{-} e \overset{4}{-} f \overset{4}{-} d \sim b \overset{3}{-} c \overset{3}{-} d \overset{3}{-} f \sim c \overset{3}{-} d \sim a \overset{1}{-} b \sim e \overset{1}{-} f$$
$$e \overset{4}{-} a \overset{4}{-} b \overset{4}{-} c \overset{4}{-} d \overset{4}{-} f \sim b \overset{3}{-} c \overset{3}{-} d \overset{3}{-} f \overset{3}{-} e \overset{3}{-} f \sim a \overset{1}{-} b$$

Fig. 4. Transistor netlist (pull-down), graph representation, and three different folding solutions with their associated transistor chains.

The folding problem can be reduced to the generation of a set of graphs $\{G_s\}$, where $s \in \{1, \ldots, \mathbb{S}\}$. Each graph $G_s(N, T_s)$ contains the legs with size s.

A. Example

Fig. 4 shows an example of transistor folding. The graph at the left represents a netlist of transistors with the same polarity in which each edge is a transistor that connects two nodes: source and drain. The gates of the transistors are omitted. This netlist could represent the pull-down network of an AOI33 gate. Each edge has a label that represents the size of the transistor. In some cases, the label represents an interval of discrete sizes.

The graphs at the right represent three different solutions of the folding problem under the assumption that every device can have four tracks at most ($\mathbb{S} = 4$). Each solution depicts the edges in the graphs $G_1, \ldots, G_4$. Apparently, all solutions have the same cost with 11 devices each one. However, the area cost is different when considering the optimal transistor chaining.

At the bottom of Fig. 4, optimal transistor arrangements for each one of the solutions are shown. Each edge $n \overset{s}{-} n'$ represents a transistor connecting n and n' with size s. The symbol $\sim$ represents a diffusion break. In this case, it has been assumed that DIFFSIZEGAP = SAMESIZEGAP = 1.

[3]To be more precise, G is a multigraph since there can be multiple transistors (edges) between the same pair of nodes.

[4]The reader can easily realize that allowing chaining with different diffusion sizes can be supported with a simplification of the model.

Although solution (c) is the one that uses the largest transistor sizes for arcs `c-d` and `e-f`, it turns out to be the most area efficient. This example clearly illustrates the impact of a good folding strategy in the cell area. This example also illustrates how different legs of the same transistor can be placed separately in the layout.

B. Basic Graph Theory on Eulerian Paths

This section reviews some fundamental concepts of graph theory and Eulerian paths [5] that will be used in the folding model.

Theorem 1 (Existence of Eulerian path): An undirected graph has an Eulerian path if and only if at most two nodes have odd degree, and if all of its nodes with nonzero degree belong to a single connected component. If there are two nodes with odd degree, these nodes must be the endpoints of any Eulerian path.

Henceforth, we will call odd nodes and even nodes those nodes with odd and even degree, respectively.

A non-Eulerian graph can become Eulerian by adding extra edges. This process is called Eulerization. Eulerizing a connected graph with a minimum number of edges is simple: it is sufficient to add edges between pairs of odd nodes until all nodes become even. In case of semi-Eulerization, one less edge is to be added so that two odd nodes may still remain. For graphs describing transistor netlists, semi-Eulerization represents the process of adding diffusion breaks in the transistor chains.

Definition 1 (Eulerization cost): The (semi-)Eulerization cost of a graph is the minimum number of edges that must be added to the graph to become (semi-)Eulerian.

Theorem 2: Let G be a connected graph and $V_o(G)$ the subset of odd nodes. The Eulerization cost of G is

$$\text{EULERCOST}(G) = \frac{|V_o(G)|}{2}.$$

The semi-Eulerization cost of G is[5]

$$\text{SEMIEULERCOST}(G) = \max(0, \frac{|V_o(G)|}{2} - 1).$$

Graphs with multiple connected components are not Eulerian. In general, each graph can have Eulerian connected components (ECCs) and non-Eulerian connected components (NCCs) depending on the property of individually being Eulerian. The Eulerization cost of a graph with multiple components must also account for the cost of connecting the graph.

The next theorem is essential to guarantee the optimality of the approach presented in this paper.

Theorem 3 (Eulerization cost [3]): Let G be a non-Eulerian graph and $V_o(G)$ the subset of odd nodes. The Eulerization cost of G is

$$\text{EULERCOST}(G) = \frac{|V_o(G)|}{2} + \text{ECC}(G)$$

where $\text{ECC}(G)$ is the number of ECCs of G.

[5]The max operator is required to prevent a negative cost in case the graph is Eulerian.

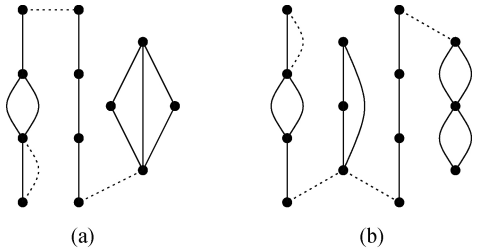


Fig. 5. Examples of semi-Eulerization cost for disconnected graphs.

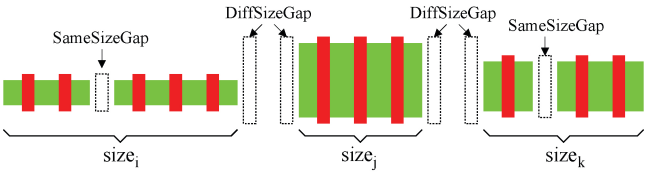


Fig. 6. Layout for min-area transistor chaining.

Corollary 1: Let G be a non-Eulerian graph. The semi-Eulerization cost of G is

$$\text{SEMIEULERCOST}(G) = \frac{|V_o(G)|}{2} + \text{ECC}(G) - 1.$$

Fig. 5 depicts two examples of disconnected graphs and their semi-Eulerian cost. The dotted edges represent extra edges for the graph to become semi-Eulerian. The graph in Fig. 5(a), ignoring the dotted edges, has three connected components and eight odd nodes. None of the connected components is Eulerian. For this reason, two Eulerization edges can be reused as bridges to connect the components.

The graph in Fig. 5(b) has four connected components and six odd nodes. However, two of the connected components are Eulerian (second and fourth from the left). In this case, extra edges must be used as bridges to connect these components.

III. MILP MODEL

An MILP formulation of the transistor folding problem is presented in this section. The formulation is based on the model illustrated in Fig. 4, where the graph G is decomposed into a set of graphs $\{G_s\}$, with $s \in \{1, \ldots, \mathbb{S}\}$.

Fig. 6 depicts a layout architecture for transistor netlists that guarantees minimal area. Since transistors with different sizes cannot be chained, groups of equally sized transistors are created. Each group corresponds to one of the G_i graphs for folding. This scheme minimizes the diffusion breaks between different sizes that have a cost of DIFFSIZEGAP slots.

Within each group, the number of allocated slots is equal to the number of transistors plus the semi-Eulerization cost (Corollary 1). The diffusion breaks for equally sized transistors have a cost of SAMESIZEGAP slots (typically one slot when using isolation gates).

We first explain the model for one row of transistors. In Section V, the model will be extended to support multiple rows and multiple-height cells.

TABLE II
VARIABLES OF THE MILP MODEL

Variable	Number	Domain	Description
$\lambda(t,s)$	$\lvert T\rvert \times \mathbb{S}$	$\mathbb{Z}$	Legs of transistor t with size s
$i(n,s)$	$\lvert N\rvert \times \mathbb{S}$	$\mathbb{Z}$	Degree of n in G_s (even part)
$o(n,s)$	$\lvert N\rvert \times \mathbb{S}$	$\mathbb{B}$	Degree of n in G_s (parity)
$Odd(s)$	$\mathbb{S}$	$\mathbb{R}$	Number of odd nodes in G_s
$Breaks(s)$	$\mathbb{S}$	$\mathbb{R}$	Semi-Eulerization cost in G_s
$UseSize(s)$	$\mathbb{S}$	$\mathbb{B}$	Presence of some edge in G_s

TABLE III
SUMMARY OF CONSTRAINTS OF THE MILP MODEL

Constraint	Number	Description
(1)	$2 \times \lvert T\rvert$	Total sum of leg sizes
(2)	$\lvert N\rvert \times \mathbb{S}$	Odd/even degree of each node
(3)	$\mathbb{S}$	Number of odd nodes in G_s
(4)	$\mathbb{S}$	Semi-Eulerization cost in G_s
(5)	$\mathbb{S}$	Number of different sizes

A. Variables and Constraints of the MILP Model

A summary of the variables and constraints is reported in Tables II and III.

1) *Size Constraints:* The main variables of the MILP model are $\lambda(t,s)$. They are integer variables representing the number of legs of size s to implement transistor t. The size constraint (two inequalities) is defined for each transistor and guarantees the sum of sizes of all legs to be in the interval $[\text{SIZE}_{\min}(t), \text{SIZE}_{\max}(t)]$, i.e.

$$\forall t \in T: \quad \text{SIZE}_{\min}(t) \leq \sum_{s=1}^{\mathbb{S}} s \cdot \lambda(t,s) \leq \text{SIZE}_{\max}(t). \tag{1}$$

The following constraints are defined to calculate the length of the transistor chains after folding based on the existence of Eulerian paths and the Eulerization cost.

2) *Eulerian Paths:* The degree of every node $n \in N$ at every graph G_s is the total number of legs with size s that are incident to n. We denote by $\delta(n,s)$ the degree of each node n in the graph G_s that can be expressed as follows:

$$\delta(n,s) = \sum_{t=(n,n')} \lambda(t,s)$$

where $t = (n,n')$ represents an edge incident to n.

For every size s, the existence of an Eulerian path in G_s and the cost of Eulerizing G_s can be calculated by knowing the number of odd nodes. For this, we introduce two sets of variables, $i(n,s)$ (integer) and $o(n,s)$ (binary), to calculate the parity of the degree of each node. Thus

$$\forall n \in N, \forall s \in \{1,\ldots,\mathbb{S}\}: \quad \delta(n,s) = 2 \cdot i(n,s) + o(n,s). \tag{2}$$

Given that $i(n,s)$ is integer and $o(n,s)$ is binary, the value of these two variables is unique for every value of $\delta(n,s)$. Therefore, $o(n,s) = 1$ indicates that the degree is odd. The total number of odd nodes in G_s can be calculated as follows:

$$Odd(s) = \sum_{n \in N} o(n,s). \tag{3}$$

Assuming that every graph G_s is connected,[6] Euler's theory provides the number of edges (semi-Eulerization cost) that need to be added to create an Eulerian path (see Theorem 2)

$$\forall s \in \{1,\ldots,\mathbb{S}\}: \quad Breaks(s) = \max(0, \frac{Odd(s)}{2} - 1).$$

Given that $Breaks(s)$ will be a variable minimized by the cost function, the previous constraint can be substituted by

$$\forall s \in \{1,\ldots,\mathbb{S}\}: \quad Breaks(s) \geq \frac{Odd(s)}{2} - 1. \tag{4}$$

3) *Diffusion Breaks Between Differently Sized Transistors:* In order to calculate the number of diffusion breaks between differently sized transistors (see Fig. 6), we need to know how many different sizes are used. A set of new binary variables, $UseSize(s)$, are defined to account for the usage of each size s

$$\forall s \in \{1,\ldots,\mathbb{S}\}: \quad UseSize(s) \geq \frac{1}{\Gamma_s} \sum_{t \in T} \lambda(t,s) \tag{5}$$

where Γ_s is a sufficiently large constant to guarantee that the right-hand side of the inequality is a value in the interval [0, 1]. A valid value for Γ_s could be calculated by assuming that all legs of all transistors would have size s, that is

$$\Gamma_s = \frac{1}{s} \cdot \sum_{t \in T} \text{SIZE}_{\max}(t).$$

If any transistor would have a leg of size s, then $UseSize(s)$ would be forced to take the value 1. Since $UseSize(s)$ will be a variable minimized by the cost function, its value will be 0 when no transistor uses any leg of size s.

4) *Cost Function:* The cost function aims at minimizing the area of the cell that includes the legs and diffusion breaks. The total number of legs is

$$Nlegs = \sum_{\substack{t \in T \\ s \in \{1,\ldots,\mathbb{S}\}}} \lambda(t,s).$$

The total number of diffusion breaks inside the blocks of equally sized transistors is

$$Nbreaks = \sum_{s \in \{1,\ldots,\mathbb{S}\}} Breaks(s).$$

Finally, extra edges must be added to bridge blocks of transistors with different sizes. The number of required bridges is equal to the number of different sizes minus one (see Fig. 6)

$$Nbridges = -1 + \sum_{s \in \{1,\ldots,\mathbb{S}\}} UseSize(s).$$

To obtain a min-area cell, we need to minimize the total number of slots in the row

$$\begin{aligned} \min: \text{AREA} \;=\; & Nlegs & + \\ & \text{SAMESIZEGAP} \cdot Nbreaks & + \\ & \text{DIFFSIZEGAP} \cdot Nbridges. \end{aligned} \tag{6}$$

[6]This assumption is an imperfection of the MILP model, since the graphs G_s may not be necessarily connected. A strategy to treat this imperfection will be discussed in Section IV.

The MILP model with the constraints (1)–(5) and the cost function (6) delivers an area-optimal folding solution under the assumption that the value of *Breaks*(s) from constraint (4) corresponds to the semi-Eulerization cost of the resulting graph (see Theorem 3).

The next section proposes an algorithmic approach to guarantee optimality even in the case that there is a discrepancy between *Breaks*(s) and the semi-Eulerization cost.

IV. Generation of Folding Solutions

The MILP model generates a set of graphs $\{G_s\}$, each one containing the edges of each size s. The area cost of implementing a transistor chain must be calculated by adding the semi-Eulerization cost (diffusion breaks).

As it was mentioned in the previous section, the MILP model may deliver a solution in which the area cost given by (6) does not coincide with the real cost of the solution when considering all diffusion breaks.

The discrepancy is originated by the difference of the semi-Eulerization cost for connected and disconnected graphs, formally modeled by the difference between Theorem 2 and Corollary 1. More precisely, the MILP model does not take into account the number of ECCs of each graph G_s.

Given a solution S of the MILP model, we will denote by MILP_AREA(S) the area estimated by the model and GRAPH_AREA(S) the exact area calculated from the graph associated to the solution. It holds that

$$\text{MILP\_AREA}(S) \leq \text{GRAPH\_AREA}(S)$$

and the difference arises when there is some G_s that is disconnected and $\text{ECC}(G_s) \neq 0$ (see Corollary 1).

We propose to solve this imperfection algorithmically by generating different solutions until we found one in which MILP_AREA(S) = GRAPH_AREA(S). Although the strategy may theoretically require a large number of iterations, the experiments show that most of the initial solutions are already optimal and only few extra iterations are required for a small set of cells.

Algorithm IV generates a solution for transistor folding by iteratively solving different MILP models. When a solution is found in which the estimated cost and the real cost coincide (line 8), the solution is guaranteed to be optimal.

In case the estimated area and the real area do not coincide, the cause of the discrepancy is investigated. This is always produced by a set disconnected G_s's with $\text{ECC}(G_s) \neq 0$ (line 11). The MILP model is modified by introducing cuts that prevent the same solution to be generated by the G_s's causing the area underestimation (line 12). The technique to introduce these cuts will be discussed in Section IV-A.

Finally, a new constraint is added to cut all solutions that have an estimated area greater than or equal to the real area of the last solution (line 13).

To avoid an excessive computational cost, a maximum number of iterations is allowed (line 10). If this number is exceeded, the returned solution cannot be guaranteed to be optimal.

It may occur that the progressive introduction of cuts makes the MILP model unsatisfiable (line 7). In this case, the optimal solution contains some disconnected G_s with $\text{ECC}(G_s) > 0$ and is one of the generated in previous iterations (*Best_S*).

A. Generation of Cuts to Exclude Suboptimal Solution

The introduction of cuts to exclude a particular solution is based on the technique proposed in [19]. The technique can be slightly simplified for the folding problem by observing that the solution is only characterized by the variables $\lambda(t, s)$ and that a new solution with optimal cost will always imply that one of the nonzero $\lambda(t, s)$ variables of disconnected G_s's with $\text{ECC}(G_s) \neq 0$ will be modified.

Let us assume that $X = \{x_1, \ldots, x_k\}$ is the set of integer variables with nonzero value for which a new solution must be generated. Let x_i^0 be the value of x_i in the current solution. Then, a new set of constraints is added to the MILP model to enforce that at least one of the variables will change its value, that is

$$\sum_{i=1}^{k} |x_i - x_i^0| \geq 1.$$

To linearize the previous expression, new variables $\alpha_i \in \{0, 1\}$ and $W_i \geq 0$ are defined for each $x_i \in X$, with the following constraints:

$$0 \leq W_i - x_i + x_i^0 \leq M(1 - \alpha_i)$$
$$0 \leq W_i - x_i^0 + x_i \leq M\alpha_i$$

where M is a large constant. Finally, a new constraint is added to enforce some variable to change its value

$$\sum_{i=1}^{k} W_i \geq 1.$$

B. Alternative Method to Generate Cuts

The method proposed in [19] requires the addition of $|X|$ binary variables (α_i), $|X|$ real variables (W_i), and $2|X| + 1$ constraints.

We propose a new method to eliminate a solution that only requires one new binary variable and two constraints. However, the method may also eliminate other optimal solutions, but hopefully with very low probability.

The method consists in calculating a hash value of the solution and eliminating all solutions that have the same hash value. The hash function is calculated as a linear combination of the nonzero variables of the solution using a set of coefficients, that is

$$\text{HASH}(X) = \sum_{i=1}^{k} c_i x_i.$$

In our case, we selected small prime numbers for the coefficients c_i. Let Φ be a constant that is the hash value of the solution provided by the MILP model. Let α be a new binary variable and M a large constant. The next constraint enforces $\text{HASH}(X) \neq \Phi$

$$(\Phi + 1)(1 - \alpha) \leq \text{HASH}(X) \leq (\Phi - 1)\alpha + M(1 - \alpha).$$

Algorithm 1 Transistor folding

```
1: M = MILP model for transistor folding;
2: Best_S = ∅;                                              ▷ No initial solution
3: n_iter = 0;                                              ▷ Iteration counter
4: loop
5:     n_iter = n_iter + 1;
6:     S = ILP_SOLVER(M);                                   ▷ S is the solution delivered by the MILP solver
7:     if M is unsatisfiable then return Best_S;            ▷ Can never occur on the first iteration. The returned solution is optimal
8:     if MILP_AREA(S) = GRAPH_AREA(S) then return S;       ▷ Optimal solution: the area estimated by the model is correct
9:     if GRAPH_AREA(S) < GRAPH_AREA(Best_S) then Best_S = S;   ▷ A better solution has been found (may not be optimal yet)
10:    if n_iter ≥ MAXITERATIONS then return Best_S;        ▷ Too many iterations: return the best one found so far (may not be optimal)
11:    {G_Ecc} = Set of disconnected G_s with Ecc(G_s) ≠ 0;   ▷ Set of graphs that generate area underestimation
12:    M = M ∪ {constraints to exclude {G_Ecc}};            ▷ Add constraints to avoid the generation of the same solution for {G_Ecc}
13:    M = M ∪ {AREA ≤ GRAPH_AREA(S) − 1};                  ▷ Add constraint to improve the previous cost
14: end loop
```

The generation of cuts not only contributes to eliminate suboptimal solutions, but can also be used to generate multiple solutions with the same or similar cost.

The generation of a cell layout also depends on the subsequent steps in the synthesis flow: transistor placement and routing. Some cells may require a highly congested layout (e.g., flip-flops, full adders, or simple cells with multiple inputs) and the routability of the cell may depend on subtle variations of the folding and placement solutions. The availability of different solutions may contribute to increase the probability of finding routable solutions with optimum area.

To avoid the unlikely situation in which a cut also eliminates some optimal solution, a hybrid approach can be used combining the cuts presented in Section IV-A (to find an optimal solution) with the cuts presented in this section (to generate a diversity of similar solutions).

V. Extensions of the Model

The MILP proposed in Section III can solve the folding problem for single-height cells. The model can be solved independently for the p and n devices and the width of the cell will be determined by the maximum width of the two rows. It is also easy to formulate an integrated model to solve both problems simultaneously.

This section presents extensions of the model to support different variants of the problem.

A. Adaptable Diffusion Tracks

For the synthesis of standard cells, it is often the case that the total number of tracks for active area is defined a priori, whereas the height of the n and p strips can be adaptable as long as the sum of both heights does not exceed the available tracks for diffusion.

We propose an extension of the model that supports this flexibility. The extension is proposed for the case of synthesizing single-height cells. The extension to multiple-height cells is briefly discussed in Section V-C.

Let us introduce two new variables, $\mathbb{S}_n$ and $\mathbb{S}_p$, that represent the maximum size of n and p transistors, respectively. Since we have a fixed number of tracks for active area, we add a constraint on the total number of tracks used by the diffusions, that is

$$\mathbb{S}_n + \mathbb{S}_p \leq \mathbb{S} \tag{7}$$

where $\mathbb{S}$ is a constant that now represents the maximum number of tracks for diffusions.

We now have to make sure that no transistor exceeds the maximum allowable size. For that, we can add new constraints on the variables that represent the usage of each size

$$\forall s \in \{1, \ldots, \mathbb{S}\}: \; s \cdot UseSize_n(s) \leq \mathbb{S}_n, \quad s \cdot UseSize_p(s) \leq \mathbb{S}_p.$$

where $UseSize_n(s)$ and $UseSize_p(s)$ are the binary variables that represent the presence of legs of size s in the p and n rows, respectively.

B. Multiple-Height Cells: Folding and Row Assignment

The previous model can be extended for multiple-height cells. A common case is the synthesis of double-height cells, as shown in the example of Fig. 1(b). As for the synthesis of single-height cells, the problem can be solved independently for p and n devices.

Let us assume that we can have R different rows of transistors. The MILP model will now generate $R \times \mathbb{S}$ graphs. The edges of graph $G_{r,s}$ will represent the legs in row r and size s. Therefore, the model determines the size and the row of each transistor leg.

The modifications to the basic MILP model are the following.

1) All the variables of the model (see Table II) must have a different instance for each row r: $\lambda(t, r, s)$, $i(n, r, s)$, $o(n, r, s)$, $Odd(r, s)$, $Breaks(r, s)$, and $UseSize(r, s)$, where t represents a transistor, n a node of the netlist, r a row, and s a size.
2) The size constraint (1) needs to be slightly modified

$$\forall t \in T: \quad \text{SIZE}_{\min}(t) \leq \sum_{\substack{r \in \{1,\ldots,R\} \\ s \in \{1,\ldots,\mathbb{S}\}}} s \cdot \lambda(t, r, s) \leq \text{SIZE}_{\max}(t).$$

3) The rest of constraints (2)–(5) must be instantiated for each row of the layout.

4) A new variable (AREA) and R constraints must be added to calculate the maximum area of all rows. For each row r, the following constraints will be added:

$$\begin{aligned} \text{AREA} \;\geq\; & \mathit{Nlegs}_r & + \\ & \text{SAMESIZEGAP} \cdot \mathit{Nbreaks}_r & + \\ & \text{DIFFSIZEGAP} \cdot \mathit{Nbridges}_r. \end{aligned} \tag{8}$$

5) The cost function must minimize the area of the cell

$$\min : \quad \text{AREA}.$$

It is interesting to observe that the support for multiple-height cells not only calculates the folding for transistors but also partitions and assigns the legs to the rows of the layout. Therefore, the model also guarantees the existence of a partition with the target AREA. However, this partition may not be unique and the subsequent synthesis tools for transistor chaining may find a different one with the same folding configuration, but possibly assigning the legs to different rows.

Finally, the reader may realize that the previous model can be easily generalized to accept rows with a different number of tracks of active area.

C. Double-Height Diffusions

Multiple-height cells are typically organized by interleaving n and p rows in such a way that the Vdd and Gnd rails can be shared internally. For example, a triple-height cell can be laid out with adjacent n and p rows organized as follows: $nppnnp$. Fig. 1(b) and (c) depicts double-height cells with the structure $nppn$.

As shown in Fig. 1(c), adjacent rows with the same polarity can be extended and merged to allocate larger legs. In this example, the active area for p transistors occupies four tracks. However, by merging two adjacent p rows, transistors up to ten tracks can be implemented.

Fig. 7 shows a possible structure for the active area of standard cells with double-height diffusions. The layout is organized by putting all double-height blocks of transistors at the left of the cell and the single-height blocks at the right. As in the basic layout model, differently sized blocks will be separated by DIFFSIZEGAP tracks. This structure guarantees area optimality but does not prevent the placement tools to find another ordering with better routability.

For simplicity, we will formulate a model for the double-height p diffusion in a cell organized with rows $nppn$. The reader will soon realize that the model can be easily extended for other templates.

If $\mathbb{S}_p$ is the maximum size of a p transistor in a p row, then the legs occupying both rows can have a size up to $2\mathbb{S}_p + \chi_p$, where χ_p is a constant that represents the extra size available between the two p rows. In the example of Fig. 1(c), we have $\mathbb{S}_p = 4$ and $\chi_p = 2$.

To incorporate the double-height diffusions, the MILP model must be slightly modified. A similar approach as the one presented in Section V-B for multiple-height cells must be used. However, adjacent rows are not totally independent since they can allocate large transistors.

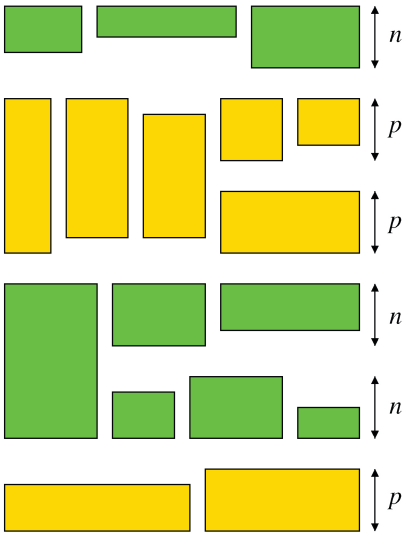


Fig. 7. Layout organization for double-height diffusions.

In the model, we will assume we have two single-height rows that we will represent as $\top$ (top) and $\bot$ (bottom). We will also use the symbol $\mathbb{I}$ to denote the double-height row that can be used by merging the top and bottom rows.

For every p transistor t, (8) must be rewritten as follows:

$$\begin{aligned} \text{SIZE}_{\min}(t) \leq & \sum_{s=1}^{\mathbb{S}_p} s \cdot (\lambda(t, \top, s) + \lambda(t, \bot, s)) + \\ & \sum_{s=\mathbb{S}_p+1}^{2\mathbb{S}_p+\chi_p} s \cdot \lambda(t, \mathbb{I}, s) \leq \text{SIZE}_{\max}(t). \end{aligned}$$

The rest of constraints must also be adapted to accommodate these new variables.

Finally, the cost function must take into account that the area of the double-height diffusions must be accounted in the two p rows simultaneously. The following constraints guarantee that AREA takes the maximum area of both rows:

$$\begin{aligned} \text{AREA} \;\geq\; & \mathit{Nlegs}(\top) + \mathit{Nlegs}(\mathbb{I}) + \\ & \text{DIFFSIZEGAP} \cdot (\mathit{Nsizes}(\top) + \mathit{Nsizes}(\mathbb{I}) - 1) + \\ & \text{SAMESIZEGAP} \cdot (\mathit{Nbreaks}(\top) + \mathit{Nbreaks}(\mathbb{I})). \end{aligned}$$

$$\begin{aligned} \text{AREA} \;\geq\; & \mathit{Nlegs}(\bot) + \mathit{Nlegs}(\mathbb{I}) + \\ & \text{DIFFSIZEGAP} \cdot (\mathit{Nsizes}(\bot) + \mathit{Nsizes}(\mathbb{I}) - 1) + \\ & \text{SAMESIZEGAP} \cdot (\mathit{Nbreaks}(\bot) + \mathit{Nbreaks}(\mathbb{I})). \end{aligned}$$

We can observe that the $\mathbb{I}$ variables representing the legs and breaks of the double-height diffusions equally contribute to the area of the top and bottom rows.

The synthesis of multiple-height cells with double-height diffusions can be extended to incorporate an adaptable number of tracks, as was discussed in Section V-A. For example, for a double-height cell, the constraint (7) could be extended as follows:

$$\mathbb{S}_n^{\text{top}} + \mathbb{S}_p^{\text{top}} + \chi_p + \mathbb{S}_p^{\text{bottom}} + \mathbb{S}_n^{\text{bottom}} \leq \mathbb{S}$$

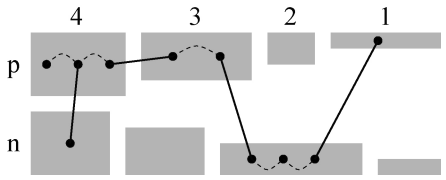


Fig. 8. Local (dashed lines) and global (solid lines) wires between transistor groups.

where $\mathbb{S}$ is the maximum total height of the cell. Additional constraints on the *UseSize* variables should be included in a similar way, as was discussed in Section V-A. This scheme can be easily extended for any arbitrary number of rows.

VI. Wire Optimization

The proposed MILP model aims at minimizing the area of the cell and delivers a solution that guarantees (by Euler's theory) the existence of a transistor alignment with the area calculated by the model. Still, there may be multiple folding solutions with the same optimal area cost. An interesting question is: among all the area-optimal solutions, can the MILP model provide one with good routability properties? In this section, we propose an optimization criterion that has a direct impact on wire congestion.

The MILP model generates transistor legs with different sizes. In the case of multiple-height cells, it also assigns the legs to one of the rows of the cell. At an abstract level and using the nomenclature from Section V-B, every solution assigns legs to the $G_{r,s}$ graphs of the cell. Every graph $G_{r,s}$ represents a group of transistors with the same size.

Given the gaps required to separate transistors with different sizes, area-optimal layouts have a tendency to group (and chain) transistors with the same size in the same active area. Let us call local wires the ones used to connect terminals of the same signal within the same group of transistors represented by graph $G_{r,s}$.

If a signal belongs to various $G_{r,s}$, there will be wires across different transistors groups. Let us call them global wires.

This is illustrated in Fig. 8, where the shadowed boxes represent active areas allocating transistors with the same size, i.e., active area corresponds to a graph $G_{r,s}$. In the picture, the p and n transistors are allocated in the p and n rows, respectively. The numbers on top of the picture represent transistor sizes (e.g., number of tracks). The dots represents the terminals of one signal and the dashed and solid lines represent local and global wires for that signal.

If a signal is present in k transistor groups, there will be at least $k-1$ global wires across these groups, i.e., the number of edges of the spanning tree connecting the groups.

We propose to enhance the cost function of the MILP model with a term that aims at minimizing the local and global wiring cost in the layout. The experimental results will show that this minimization has a positive impact on routing the cell.

We define a set of new binary variables, called $InGroup(n, r, s)$, that denote the presence of a signal n in graph $G_{r,s}$. If we call $T(n)$ the set of transistors connected to node n (gate, source or drain), the following constraint enforces $InGroup(n, r, s) = 1$ when a transistor connected n is present in $G_{r,s}$

$$\mathcal{K} \cdot InGroup(n, r, s) \geq \sum_{t \in T(n)} \lambda(t, r, s)$$

where $\mathcal{K}$ is a big constant. The total cost of global wires across groups is directly related to the number of groups in which each signal is present and can be estimated as follows:

$$CostGlobalWires = \sum_{\forall n,r,s} InGroup(n, r, s).$$

The number of local wires is directly related to the number of transistor pins in the netlist. The exact number of pins requiring wires in the netlist will finally depend on how transistors are placed and diffusions are shared. The total cost of local wires can be estimated by calculating the total number of transistors in the netlist, that is

$$CostLocalWires = \sum_{\forall t,r,s} \lambda(t, r, s).$$

Finally, the total cost of wires can be estimated as

$$CostWires = \alpha \cdot CostLocalWires + \beta \cdot CostGlobalWires \quad (9)$$

where α and β are constants that define the relative weight of each term. Empirically, it has been observed that $\alpha = 1.5$ and $\beta = 1$ deliver good results.

The minimization of global wires can be incorporated in the cost function as a new term weighted with a small constant (γ), that is

$$\min : \text{AREA} + \gamma \cdot CostWires.$$

The experimental results in Section VIII-C will confirm the positive impact of the wire minimization.

VII. Heuristics for Transistor Folding

This section presents two heuristics for transistor folding as an alternative to the MILP model. The comparison of the results delivered by the MILP model and the two heuristics will contribute to emphasize the importance of a good transistor folding.

Before folding, each transistor t has an interval of legal discrete sizes $[\text{SIZE}_{\min}(t), \text{SIZE}_{\max}(t)]$. Each leg can have a size in $\{1, \ldots, \mathbb{S}\}$. The two proposed heuristics always use the minimum number of legs for folding transistor t

$$L = \left\lceil \frac{\text{SIZE}_{\min}(t)}{\mathbb{S}} \right\rceil. \quad (10)$$

Both heuristics are myopic, in the sense that the distribution of legs for each transistor does not depend on the other transistors in the same netlist. The difference between the heuristics comes from the way sizes are distributed among the legs.

A. Greedy Heuristic

This is a naive heuristic biased toward generating legs with size $\mathbb{S}$. Every transistor is folded into L legs, with $L-1$ legs of maximum size ($\mathbb{S}$) and one leg with the remaining size ($\mathbb{S}'$) such that

$$(L-1)\cdot\mathbb{S}+\mathbb{S}' = \text{SIZE}_{\min}(t).$$

B. Balanced Heuristic

This is a Euler-friendly heuristic with two main goals.

1) For large transistors, use only legs with size $\mathbb{S}$ whenever possible. If not possible, use only legs with size $\mathbb{S}$ and $\mathbb{S}-1$.
2) Whenever possible, use an odd number of legs with size $\mathbb{S}$.

The first goal reduces the diversity of sizes used for transistor folding, thus reducing the area penalty associated with the gaps between differently sized transistors.

The second goal aims at preserving the evenness of the nodes in the graphs. By folding one transistor into an odd number of legs with the same size, the evenness of the degree of the source/drain nodes is not modified. Bearing in mind that the original transistor netlists have a tendency to have good Eulerian properties, this approach contributes to maintain them.

Assuming that L is the minimum number of legs required to fold the transistor, according to (10), the rules to fold a transistor t are the following.

1) If $\text{SIZE}_{\max}(t) \le \mathbb{S}$, use only one leg with size $\text{SIZE}_{\max}(t)$.
2) If $\text{SIZE}_{\min}(t) \le \mathbb{S} < \text{SIZE}_{\max}(t)$, use only one leg with size $\mathbb{S}$.
3) If $L\cdot\mathbb{S} \le \text{SIZE}_{\max}(t)$, use L legs with size $\text{SIZE}_{\max}(t)$.
4) Otherwise, use L' legs with size $\mathbb{S}$ and L'' legs with size $\mathbb{S}-1$ such that $L'+L''=L$ and

$$\text{SIZE}_{\min}(t) \le L'\cdot\mathbb{S}+L''\cdot(\mathbb{S}-1) \le \text{SIZE}_{\max}(t).$$

In the latter case, the valid values for L' are in the range

$$\text{SIZE}_{\min}(t) - L\cdot(\mathbb{S}-1) \le L' \le \text{SIZE}_{\max}(t) - L\cdot(\mathbb{S}-1).$$

To preserve the evenness of the nodes with size $\mathbb{S}$, the algorithm will select the smallest odd value for L', unless $\text{SIZE}_{\min}(t) = \text{SIZE}_{\max}(t)$ in which case L' is uniquely determined.

The following table shows some examples on how some transistors would be folded using both heuristics and assuming $\mathbb{S}=4$.

$[\text{SIZE}_{\min}(t), \text{SIZE}_{\max}(t)]$	Greedy	Balanced
[13, 15]	4+4+4+1	4+3+3+3
[14, 17]	4+4+4+2	4+4+4+4
[17, 19]	4+4+4+4+1	4+4+4+3+3
[21, 23]	4+4+4+4+4+1	4+4+4+3+3+3

As an example, the case [17, 19] could have been implemented with three different balanced distributions: 1) 4 + 4 + 3+3+3; 2) 4+4+4+3+3; or 3) 4+4+4+4+3. The distribution 4 + 4 + 4 + 3 + 3 is preferred to preserve the evenness of legs with maximum size.

VIII. Experimental Results

This section describes various experiments performed to evaluate the MILP model and heuristics presented in this paper. The experimental setup is first described and the results are later reported. Finally, the impact in area, routability, and computational complexity are discussed.

All the experiments have been performed in a quad-core CPU runnning at 2.67 GHz and 8 GB of memory. Gurobi [10] has been used as the MILP solver. Gurobi can efficiently exploit the architecture of multicore CPUs when solving complex MILP problems.

A. Experimental Setup

Transistor folding has been applied to the 45-nm Nangate standard cell library [15], which contains 127 cells. The original cells already have large transistors that have been folded to fit in the active area. The procedure applied to obtain the netlists for transistor folding is as follows.

1) The SPICE netlists have been parsed and functionally equivalent transistors have been merged (unfolded) into one larger transistor with a size equivalent to the sum of sizes of the original transistors.
2) A horizontal pitch P of 130 nm has been defined for each track of active area. With this pitch, most transistors in the small cells end up by taking 5 p tracks and 3 n tracks.[7] The minimum and maximum number of tracks for each transistor has been calculated as follows:

$$\text{SIZE}_{\min}(t) = \left\lceil \frac{\text{SIZE}(t)}{P}\cdot(1-\varepsilon) \right\rceil$$

$$\text{SIZE}_{\max}(t) = \left\lfloor \frac{\text{SIZE}(t)}{P}\cdot(1+\varepsilon) \right\rfloor$$

where ε determines the flexibility in size by defining the maximum deviation of the size of the folded transistor with regard to the original size. As an example, for $\varepsilon = 0.25$, a transistor with width 1260 will be allowed to take between $\text{SIZE}_{\min}(t) = 8$ and $\text{SIZE}_{\max}(t) = 12$ tracks.

The experiments have been executed to synthesize standard cells with a maximum number of five and three tracks for the p and n transistors, respectively. The gaps for diffusion breaks have been defined to be 1 and 2 depending on whether the gaps were located between equally sized or differently sized transistors, respectively.

In these experiments, flexibility has been defined uniformly, i.e., the same value of ε is applied to all transistors. In a real cell design, flexibility can be nonuniform, e.g., giving more flexibility to internal transistors and less flexibility to those transistors that need to drive the output capacitive loads.

B. Area Minimization for Single-Height Cells

Table IV reports the total area of the complete library using the MILP model (optimal) and the two heuristics (greedy and balanced) presented in Section VII for different degrees of flexibility (ε). The total area is calculated by adding the area

[7] As an example, `INV_X1` has a p and n transistor of 630 and 415 nm, respectively.

TABLE IV
TOTAL LIBRARY AREA FOR DIFFERENT FOLDING METHODS AND FLEXIBILITIES (ε) IN TRANSISTOR SIZES

ε	Optimal	Greedy	Balanced
0.00	1673	1681 (+ 0.4%)	1682 (+0.5%)
0.05	1660	1685 (+ 1.5%)	1667 (+0.4%)
0.10	1539	1734 (+ 8.7%)	1546 (+0.5%)
0.15	1530	1724 (+12.7%)	1538 (+0.5%)
0.20	1511	1746 (+15.6%)	1527 (+1.1%)
0.25	1456	1718 (+18.0%)	1505 (+3.4%)
0.30	1383	1652 (+19.5%)	1447 (+4.6%)

of one instance of each cell in the library (127 cells). The area of a cell is calculated as the number of columns (polysilicon slots) occupied by the cell. No separation columns between cells are accounted for the area calculation.

Several conclusions can be drawn from the table. The greedy method delivers highly suboptimal solutions as the flexibility in transistor sizes increases. The balanced method is still competitive for small flexibilities and shows a monotonic behavior, i.e., area is reduced as the flexibility increases. However, the lack of global optimization produces a growing deviation from the optimum when the flexibility increases, especially for $\varepsilon \geq 0.25$.

Table V reports individual area results for those cells in which the optimal solution is better than the one provided by the balanced heuristic.

A transistor placement tool for single-height cells has been implemented to find an area-optimal layout with minimum wire length. A dynamic programming algorithm based on the approach proposed in [1] has been designed and adapted to the specific aspects of litho-friendly regular fabrics, considering the constraints about gaps between diffusion breaks.[8] The algorithm aims at minimizing the horizontal wire length required to connect all transistor terminals.

Fig. 9 depicts a symbolic layout produced by the placement tool from the netlists generated by the balanced and optimal methods for one of the cells (SDFFR_X1). The routing channel depicted between the active areas represents a symbolic view of the wiring resources required to connect the pins. The actual layout after detailed routing would use wires over the active areas. The picture also shows the difference in diffusion gaps when bridging active areas between equally sized (one slot) or differently sized (two slots) transistors.

Both cells have the same number of p and n devices. However, the heuristic approach does not consider the global combination of diffusion sizes to reduce the gaps between diffusion breaks and to create more internal Eulerian paths, thus resulting in a larger cell.

The first conclusion is that the strategy used for transistor folding can have a significant impact in area. The balanced and greedy heuristics are local strategies that lack a global view of the graph in terms of diffusion chains between different transistors.

The second conclusion is that the balanced heuristic is superior to the greedy heuristic. The main reason is because

[8]The details of the transistor placement tool are out of the scope of this paper.

TABLE V
AREA RESULTS FOR NANGATE LIBRARY (ε = 0.25)

Cell	Greedy	Balanced	Optimal
CLKBUF_X1	4	4	3
CLKBUF_X3	8	5	4
CLKGATETST_X1	19	19	17
CLKGATETST_X2	22	20	18
CLKGATETST_X4	25	23	21
CLKGATETST_X8	29	29	27
CLKGATE_X1	15	15	14
CLKGATE_X8	26	26	25
DFFRS_X1	28	28	27
DFFRS_X2	31	30	29
DFFR_X1	24	24	23
DFFR_X2	29	26	24
DFFS_X1	24	24	23
DFFS_X2	29	26	24
DFF_X1	22	22	20
DFF_X2	26	23	22
DLH_X2	15	15	13
DLL_X2	15	15	13
SDFFRS_X1	34	34	32
SDFFRS_X2	37	36	34
SDFFR_X1	30	30	27
SDFFR_X2	34	31	29
SDFFS_X1	31	31	28
SDFFS_X2	36	33	30
SDFF_X1	28	28	26
SDFF_X2	33	30	27
TLAT_X1	17	17	15
Total	671	644	595

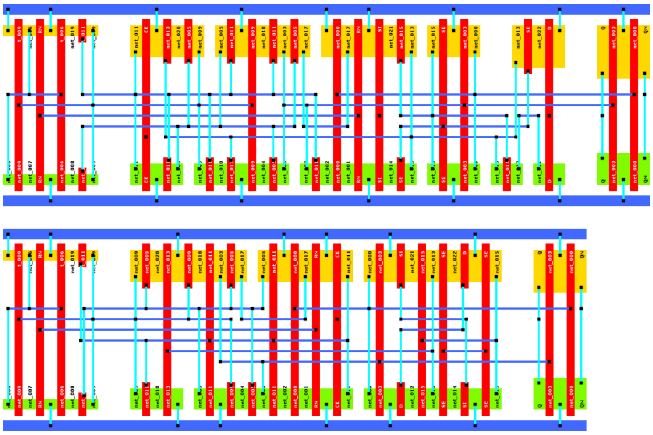


Fig. 9. Symbolic layouts for cell SDFFR_X1 obtained from the netlists generated by the balanced (top) and optimal (bottom) methods.

the balanced heuristic tries to minimize the number of different sizes used for each transistor. This tends to reduce the costly diffusion gaps between differently sized transistors. Another reason is that it also tries to generate an odd number of instances of each transistor, thus preserving the evenness of the degree of the nodes in the transistor graph.

Experiments for double-height cells led to similar conclusions in terms of area. For this reason, no results are reported.

C. Wire Optimization for Single-Height Cells

The MILP model for transistor folding has been also executed with the cost function for wire optimization (see Section VI). The transistor placement tool has been used to find an area-optimal layout with minimum horizontal wire

TABLE VI
WIRE OPTIMIZATION FOR SINGLE-HEIGHT CELLS (ε = 0.25)

	No wire optimization			Wire optimization			
Cell	TR	GW	HW	TR	GW	HW	ΔHW
AOI21_X4	22	4	84	21	4	78	-7.1%
AOI221_X4	22	8	72	21	8	52	-27.3%
AOI222_X4	24	9	96	23	9	72	-25.0%
BUF_X4	12	3	18	11	3	18	0.0%
BUF_X8	21	3	40	20	3	40	0.0%
BUF_X32	77	3	152	75	3	154	+1.3%
CLKGATETST_X4	32	21	172	31	20	106	-38.4%
CLKGATETST_X8	44	20	162	43	20	174	+7.4%
CLKGATE_X2	23	17	109	21	18	97	-11.0%
CLKGATE_X4	31	16	145	27	18	105	-27.6%
CLKGATE_X8	42	18	194	39	18	153	-21.1%
DFFRS_X1	41	36	350	40	33	309	-11.7%
DFFRS_X2	46	34	390	44	33	320	-18.0%
DFFR_X1	34	31	247	32	29	197	-20.2%
DFFR_X2	37	29	236	36	29	236	0.0%
DFFS_X1	35	29	233	32	29	195	-16.3%
DFFS_X2	38	28	229	36	27	194	-15.3%
DFF_X2	32	26	160	32	24	127	-20.6%
DLH_X1	19	14	93	16	13	49	-47.3%
DLL_X1	20	13	77	16	13	49	-36.4%
OAI221_X4	22	8	72	21	8	70	-2.8%
OAI222_X4	24	9	96	23	9	95	-1.0%
SDFFRS_X2	54	39	443	54	37	365	-17.6%
SDFFR_X2	47	26	222	46	25	221	-0.5%
SDFF_X1	39	30	222	38	30	184	-17.1%
TLAT_X1	23	16	137	20	17	76	-44.5%
Total	**861**	**490**	**4451**	**818**	**480**	**3716**	**-16.5%**

length (HW), measured as the total length of horizontal wires to connect the transistor pins.

Table VI reports the set of cells in which the MILP model provides a different solution when wire optimization is incorporated in the cost function. Column TR reports the number of transistors in the cell that corresponds to the term *CostLocalWires* in (9). Column GW reports the value of *CostGlobalWires* in the same equation. Finally, HW reports the horizontal wire length after placement.

The results show a clear impact of the cost function on the final wire length. Out of 127 cells, the MILP model delivered different solutions for 26 cells. In most of them, there was a clear improvement of HW, which contributes to a better routability and efficiency of the cell. Interestingly, many of the optimized cells were sequential. This is understandable given the fact that wire optimization has more impact on gates with complex nonseries/parallel structures. Most of the conventional static CMOS cells with series/parallel structures (NAND, NOR, AOI, OAI) and with small transistor sizes (X1 or X2) do not show differences in the final netlists after transistor folding.

Fig. 10 depicts the two layouts for one of the cells (CLKGATETST_X4) after transistor placement. In this case, the optimized layout has one less n device and one less global wire. This contributed to a better reorganization of active areas to reduce the wiring cost. The figure clearly demonstrates the reduction in wire length when using the wire optimization term in the cost function.

Wire optimization has more impact when more flexibility (ε) is provided, given that the solution space is vaster and more configurations can be explored.

The transistor placement tool provides a lower bound on the number of routing resources (horizontal and vertical) required

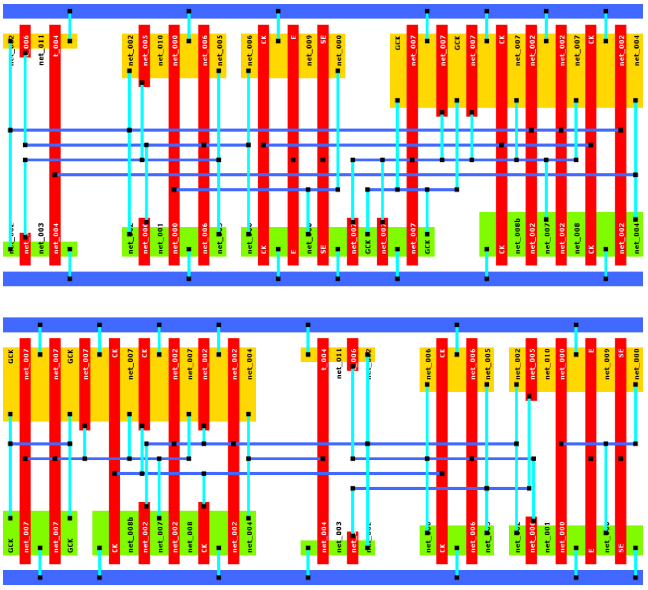


Fig. 10. Symbolic layouts for cell CLKGATETST_X4 after transistor placement without (top) and with (bottom) wire optimization.

to route the nets in the cell. These lower bounds are the ones that are symbolically depicted in Figs. 9 and 10. With 1-D GDRs, these resources will correspond to different metal layers (e.g., M1 and M2). In technologies from 20 nm and below, new layers of local interconnects are usually provided for the contacts with poly and active areas [21]. The results reported in Table VI for HW correspond to the lower bound on horizontal routing.

D. Wire Optimization for Double-Height Cells

Multiple-height cells are usually laid out to improve the routability of complex cells. By having a more balanced aspect ratio, congested channels of signals that go across long cells are avoided.

As discussed in Section V-B, the MILP model can be adapted to handle multiple-height cells. In this section, we estimate the impact of wire optimization in double-height cells.

Table VII reports results on wire optimization for cells using active areas organized as n-p-p-n strips with a maximum number of 3-5-5-3 tracks, respectively. The table reports the number of transistors (TR) and the estimation of global wires (GW) in the solution delivered by the MILP model with and without wire optimization.[9]

The number of cells affected by the optimization is much larger than for single-height cells.[10] (88 out of 127). The reason is because the amount of solutions with the same area is larger for double-height cells, since devices are allocated in a larger set of active areas with different sizes distributed in the top and bottom rows of the cell. The most relevant information in the table is that the number of global wires is

[9]The estimation of horizontal wire length is not provided since the placement tool is not supporting multiple-height cells and no experiments could be run.

[10]The table only reports the details for the largest cells, even though the totals are referred to the 88 cells.

TABLE VII
WIRE OPTIMIZATION FOR DOUBLE-HEIGHT CELLS (ε = 0.25)

	No wire opt.		Wire opt.		CPU
Cell	TR	GW	TR	GW	(sec)
BUF_X16	40	8	38	7	0.18
BUF_X32	79	8	75	7	0.14
CLKGATETST_X4	32	25	31	22	2.44
CLKGATETST_X8	44	21	43	21	0.90
CLKGATE_X8	41	22	39	20	1.23
DFFRS_X1	41	37	40	34	1.33
DFFRS_X2	44	42	45	33	5.61
DFFR_X1	33	31	32	29	0.69
DFFR_X2	39	29	37	30	28.32
DFFS_X1	32	32	32	29	0.74
DFFS_X2	36	33	36	28	10.63
DFF_X2	32	28	32	26	2.15
SDFFRS_X1	50	46	50	41	164.91
SDFFRS_X2	54	53	54	39	12.74
SDFFR_X1	42	35	42	29	32.22
SDFFR_X2	49	30	46	27	4.63
SDFFS_X1	42	44	42	35	3.57
SDFFS_X2	46	37	46	31	2.05
SDFF_X1	39	35	42	26	5.23
SDFF_X2	44	27	44	26	3.17
TBUF_X16	39	14	39	9	0.64
TBUF_X8	30	14	29	10	2.07
⋮	⋮	⋮	⋮	⋮	⋮
Total (88 cells)	2106	1493	2064	1198	342.91

reduced from 1493 down to 1198 (almost 20%), which may have a positive impact in the routability of the cell.

E. Computational Complexity

An important aspect to evaluate is the computational complexity of this problem. MILP is NP-hard, but the instances of the problem evaluated in this paper can be solved in affordable CPU times.

The last column of Table VII reports the CPU time required to deliver the optimal solution when including wire optimization, which is the most complex model instance of the problem. On average, each instance took about 4 s. However, the worst case was observed for cell SDFFRS_X1 (164.91 s).

An important observation is that Gurobi [10] contains very sophisticated heuristics to solve MILP efficiently. Similar experiments were done using Glpk [7] with a CPU time between one and two orders of magnitude longer.

Another interesting aspect is that MILP solvers usually offer a timeout option that allows to deliver the best solution obtained when the timeout expires. With this option, an order of magnitude can often be reduced while still obtaining a probably optimal solution.

Finally, it is important to discuss the behavior of Algorithm IV with regard to the number of iterations of the main loop. A new iteration is executed when MILP_AREA(S) $\neq$ GRAPH_AREA(S) in line 8 of the algorithm, unless the maximum number of iterations has been exceeded. In the experiments reported in Table VII, all solutions were guaranteed to be area-optimal. For 119 cells (out of 127), optimality was achieved with only one iteration. For the rest of cases, two iterations were required for two cells, three iterations for four cells, and four iterations for two cells.

TABLE VIII
LIBRARY AREA FOR 1-D AND 2-D DESIGN STYLES

ε	2D	1D (Gap=1)	1D (Gap=2)
0.00	1407	1540 (+9.5%)	1673 (+18.9%)
0.05	1403	1531 (+9.1%)	1660 (+18.3%)
0.10	1341	1441 (+7.5%)	1539 (+14.8%)
0.15	1333	1433 (+7.5%)	1530 (+14.8%)
0.20	1323	1419 (+7.3%)	1511 (+14.2%)
0.25	1307	1383 (+5.8%)	1456 (+11.4%)
0.30	1249	1323 (+5.9%)	1383 (+10.7%)

The main reason for obtaining the optimal solution at the first iteration is that equally sized transistors are usually grouped in only one connected component (Vdd/Vss are common nodes of the component). In this way, there is no error in the Eulerization cost estimated by the MILP model.

F. 1-D Versus 2-D Design Rules

The adoption of the 1-D design style implies a new tradeoff between area and performance. On one hand, the diversity of transistor sizes has a negative impact in area due to the overhead introduced by the diffusion breaks. On the other hand, the diversity of sizes is convenient to have a larger space of solutions for gate sizing.

This section analyzes the area cost for the adoption of a gridded 1-D style with regard to the usage of a gridded 2-D style. The only difference between both styles is the area cost of the diffusion breaks between differently sized transistors.

For this purpose, the MILP model has been modified in such a way that the diversity of transistor sizes is ignored when calculating the Eulerization cost of each solution. The details of this modification are not explained, but the reader can easily devise them by assuming that no breaks are used for differently sized transistors.

As a byproduct, the modification of the MILP model subsumes previous approaches and provides an optimal solution for the folding problem in the 2-D design style also.

Table VIII summarizes the results for the area of the complete library using different degrees of flexibility (ε). For 1-D, two different gaps have been considered for differently sized diffusions: one and two slots. The following facts can be observed.

1) The area overhead introduced by 1-D GDRs is 5–10% (for gap = 1) depending on the flexibility. This overhead is produced by the diffusion breaks enforced by different diffusion sizes.
2) As expected, the area overhead approximately doubles when the cost of the diffusion breaks also doubles (gap = 2).
3) The overhead is smaller if more flexibility is tolerated (large value of ε). This is also expected, since the MILP model uses this flexibility to reduce the diversity of transistor sizes.

For area minimization, it may be more convenient to decrease the diversity of transistor sizes and reduce the number diffusion gaps. This can be achieved by increasing the flexibility of transistor sizes. However, this will limit the space of solutions for gate sizing, thus having a negative impact in

performance. On the other hand, by allowing a higher diversity of transistor sizes, performance can be better adjusted at the cost of increasing the area produced by the diffusion breaks.

Indeed, any impact in area and/or performance has a corresponding impact in power. The exploration of this tradeoff is something that should be further investigated in the future.

IX. Conclusion

The 1-D design style is becoming a major trend in current nanometric technologies and will be unavoidable in the future. Layouts with regular patterns are becoming a viable alternative to handcrafted layouts for semicustom design. When severe manufacturability constraints are imposed, the design of a standard cell is progressively evolving from an art to a combinatorial problem. In this context, design automation is playing a predominant role.

Transistor folding is one of the subproblems in the design flow of standard cells. 1-D GDRs enforce active areas to be rectangular, thus reducing the chances to find area-efficient transistor chains for netlists with multiple transistor sizes. This constraint originates a new formulation of the folding problem that can be efficiently solved algorithmically.

The method presented in this paper has an important feature: it can guarantee area optimality without calculating the exact location of the devices. With this approach, folding and placement can be decoupled without sacrificing area, which is essential to provide automation with affordable computational cost.

References

[1] R. Bar-Yehuda, J. A. Feldman, R. Y. Pinter, and S. Wimer, "Depth-first-search and dynamic programming algorithms for efficient CMOS cell generation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 7, pp. 737–743, Jul. 1989.

[2] K. S. Berezowski, "Transistor chaining with integrated dynamic folding for 1-D leaf cell synthesis," in *Proc. Euromicro Symp. Dig. Syst. Design*, 2001, pp. 422–429.

[3] F. T. Boesch, C. Suffel, and R. Tindell, "The spanning subgraphs of eulerian graphs," *J. Graph Theory*, vol. 1, no. 1, pp. 79–84, 1977.

[4] E. Y. C. Cheng and S. Sahni, "A fast algorithm for transistor folding," *VLSI Design*, vol. 12, no. 1, pp. 53–60, 2001.

[5] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Commentarii Academiae Scientiarum Petropolitanae*, vol. 8, pp. 128–140, 1741.

[6] R. S. Ghaida and P. Gupta, "DRE: A framework for early co-evaluation of design rules, technology choices, and layout methodologies," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 9, pp. 1379–1392, Sep. 2012.

[7] GNU linear programming kit [Online]. Available: http://www.gnu.org/software/glpk/glpk.html

[8] R. T. Greenway, R. Hendel, K. Jeong, A. B. Kahng, J. S. Petersen, Z. Rao, and M. C. Smayling, "Interference assisted lithography for patterning of 1D gridded design," *Proc. SPIE, Alternative Lithographic Technol.*, vol. 7271, pp. 72712U-1–72712U-11, Mar. 2009.

[9] A. Gupta and J. P. Hayes, "Optimal 2-D cell layout with integrated transistor folding," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 1998, pp. 128–135.

[10] Gurobi Optimization, Inc. (2012). Gurobi optimizer reference manual [Online]. Available: http://www.gurobi.com

[11] J. Kim and S. M. Kang, "An efficient transistor folding algorithm for row-based CMOS layout design," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 1997, pp. 456–459.

[12] L. Liebmann, L. Pileggi, J. Hibbeler, V. Rovner, T. Jhaveri, and G. Northrop, "Simplify to survive, prescriptive layouts ensure profitable scaling to 32 nm and beyond," *Proc. SPIE, Design Manuf. Design-Process Integr. III*, Mar. 2009, pp. 72750A-1–72750A-9.

[13] R. L. Maziasz and J. P. Hayes, *Layout Minimization of CMOS Cells*. Norwell, MA, USA: Kluwer, 1992.

[14] C. T. McMullen and R. H. J. M. Otten, "Minimum length linear transistor arrays in MOS," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 1988, pp. 1783–1786.

[15] Nangate 45nm open cell library [Online]. Available: http://nangate.com

[16] M. A. Riepe and K. A. Sakallah, "Transistor placement for noncomplementary digital VLSI cell synthesis," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, no. 1, pp. 81–107, Jan. 2003.

[17] N. Ryzhenko and S. Burns, "Physical synthesis onto a layout fabric with regular diffusion and polysilicon geometries," in *Proc. 48th ACM/EDAC/IEEE Des. Autom. Conf.*, Jun. 2011, pp. 83–88.

[18] M. Smayling, "Gridded design rules: 1-D approach enables scaling of CMOS logic," *Nanochip Technol. J.*, vol. 6, no. 2, pp. 33–37, 2008.

[19] J.-F. Tsai, M.-H. Lin, and Y.-C. Hu, "Finding multiple solutions to general integer linear programs," *Eur. J. Oper. Res.*, vol. 184, no. 2, pp. 802–809, 2008.

[20] T. Uehara and W. M. Vancleemput, "Optimal layout of CMOS functional arrays," *IEEE Trans. Comput.*, vol. C-30, no. 5, pp. 305–312, May 1981.

[21] K. Vaidyanathan, S. H. Ng, D. Morris, N. Lafferty, L. Liebmann, M. Bender, W. Huang, K. Lai, L. Pileggi, and A. J. Strojwas, "Design and manufacturability tradeoffs in unidirectional & bidirectional standard cell layouts in 14 nm node," *Proc. SPIE*, vol. 8327, p. 83270K, Feb. 2012.

[22] P.-H. Wu, M. P. Lin, T.-C. Chen, T.-Y. Ho, Y.-C. Chen, S.-R. Siao, and S.-H. Lin, "1-D cell generation with printability enhancement," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 32, no. 3, pp. 419–432, Mar. 2013.

[23] H. Zhang, M. D. F. Wong, and K.-Y. Chao, "On process-aware 1-D standard cell design," in *Proc. 15th Asia South Pacific Design Autom. Conf.*, Jan. 2010, pp. 838–842.

Jordi Cortadella (M'88) received the M.S. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively.

He is currently a Professor with the Department of Software, Universitat Politècnica de Catalunya. In 1988, he was a Visiting Scholar at the University of California, Berkeley. He has coauthored numerous research papers and has been invited to present tutorials at various conferences. His current research interests include formal methods and computer-aided design of very large scale integration systems with special emphasis on asynchronous circuits, concurrent systems, and logic synthesis.

Dr. Cortadella has served on the technical committees of several international conferences in the field of design automation and concurrent systems. He received the Best Paper Awards at the International Symposium on Advanced Research in Asynchronous Circuits and Systems in 2004, the Design Automation Conference in 2004, and the International Conference on Application of Concurrency to System Design in 2009. In 2003, he received a distinction for the promotion of the university research by the Generalitat de Catalunya.