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—Transistor folding, transistor sizing, cell generation, linear programming, design for manufacturability.

I. INTRODUCTION

The scaling of transistor dimensions and the manufacturing challenges involved in the sub-wavelength 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 towards 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 non-rectangular 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.

This work has been funded by a gift from Intel Corporation, the project FORMALISM (CICYT TIN2007-66523), and the Generalitat de Catalunya (ALCOM-SGR 2009-2013). J. Cortadella is with the Department of Software, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain.

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 7 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. A simple example

Figure 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 8 and 10 tracks.

In the example, the maximum size for p and n devices is 4 and 3 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 must be introduced to isolate devices. One strategy is to use isolation transistors (permanently off) by connecting the gates 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 non-rectangular 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 poly-silicon 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 work

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], and 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 $\text{height} \times \text{width}$ 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 for the complete library. Furthermore, diffusion sharing has a significant impact in area, as it 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 poly-silicon stick. The placement of transistor pairs also involves some area overhead (see, for example, Section II.B in [6]).

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.

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 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

\(^1\)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.

\(^2\)By simple enumeration, the reader can easily realize that no Eulerian path exists with the two $b$ gates being adjacent.
solutions that have better routability properties.

The 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 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.

II. A 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. 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 $[\text{SIZE}_{\text{min}}(t), \text{SIZE}_{\text{max}}(t)]$ 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 $S$ 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 (e.g., [1], [16]) and is out of the scope of this work.

The model assumes that transistors with different sizes cannot be chained, 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 $\text{DIFFSIZEGAP}$. Sets of transistors with the same size cannot always be chained due to the non-existence of Eulerian paths. In this case, the diffusion breaks may have a different gap, denoted by the constant $\text{SAMESIZEGAP}$. When using isolation gates, as in Fig. 2(a), $\text{SAMESIZEGAP}$ will be 1.

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

A. Example

Figure 4 shows an example of transistor folding. The graph at the left represents a netlist of transistors of 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 the every device can have 4 tracks at most ($S = 4$). 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 $n \sim n'$ 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 $\text{DIFFSIZEGAP} = \text{SAMESIZEGAP} = 1$.

Although solution (c) is the one that uses the largest transistor sizes for arcs $c \sim d$ and $e \sim f$, 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

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

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 $G$.

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

$$\text{SEMI EulerCost}(G) = \frac{|V_o(G)|}{2} + \text{ECC}(G) - 1.$$
TABLE II
VARIABLES OF THE MILP MODEL.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Number Domain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda(t,s) )</td>
<td>(</td>
<td>T</td>
</tr>
<tr>
<td>( i(n,s) )</td>
<td>(</td>
<td>N</td>
</tr>
<tr>
<td>( o(n,s) )</td>
<td>(</td>
<td>N</td>
</tr>
<tr>
<td>( Odd(s) )</td>
<td>(</td>
<td>S</td>
</tr>
<tr>
<td>( Breaks(s) )</td>
<td>(</td>
<td>S</td>
</tr>
<tr>
<td>( UseSize(s) )</td>
<td>(</td>
<td>S</td>
</tr>
</tbody>
</table>

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.

**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,

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

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:

\[
Odd(s) = \sum_{n \in N} o(n,s).
\]

Assuming that every graph \( G_s \) is connected\(^6\), Euler’s theory provides the number edges (semi-Eulerization cost) that need to be added to create an Eulerian path (Theorem 2):

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

\(^6\)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 Sect. IV.

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

\[
\forall s \in \{1,\ldots,|S|\} : \quad Breaks(s) \geq \frac{Odd(s)}{2} - 1.
\]

**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 \( s \).

\[
\forall s \in \{1,\ldots,|S|\} : \quad UseSize(s) \geq \frac{1}{|S|} \sum_{t \in T} \lambda(t,s).
\]

where \( \Gamma_s \) 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 \( \Gamma_s \) could be calculated by assuming that all legs of all transistors would have size \( s \), i.e.,

\[
\Gamma_s = \frac{1}{|S|} \sum_{t \in T} \text{SIZE}_{\text{max}}(t).
\]

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 \( s \).

**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:

\[
N_{\text{legs}} = \sum_{s \in \{1,\ldots,|S|\}} \lambda(t,s).
\]

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

\[
N_{\text{breaks}} = \sum_{s \in \{1,\ldots,|S|\}} \text{Breaks}(s).
\]

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):

\[
N_{\text{bridges}} = -1 + \sum_{s \in \{1,\ldots,|S|\}} \text{UseSize}(s).
\]

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

\[
\text{min : AREA} = N_{\text{legs}} + \text{SAME}_{\text{SIZE}} \cdot N_{\text{breaks}} + \text{DIFF}_{\text{SIZE}} \cdot N_{\text{bridges}}
\]

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 \( \text{Breaks}(s) \) from constraint (4) corresponds to the semi-Eulerization cost of the resulting graph (Theorem 3).

Next section proposes an algorithmic approach to guarantee optimality even in the case that there is a discrepancy between \( \text{Breaks}(s) \) 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 expression (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 \( \text{MILP\_AREA}(S) \) the area estimated by the model and \( \text{GRAPH\_AREA}(S) \) 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 \( \text{MILP\_AREA}(S) = \text{GRAPH\_AREA}(S) \). 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 1 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 Sect. 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 (\( \text{Best}_S \)).

A. Generation of cuts to exclude a sub-optimal 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 non-zero \( \lambda(t, s) \) 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 non-zero value for which a new solution must be generated. Let \( x_i^0 \) 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, i.e.,

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

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. An alternative method to generate cuts

The method proposed by [19] requires the addition of \(|X| \) binary variables \( \alpha_i \), \(|X| \) 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 non-zero variables of the solution using a set of coefficients, i.e.,

\[
\text{HASH}(X) = \sum_{i=1}^{k} c_ix_i.
\]

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

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

The generation of cuts not only contributes to eliminate sub-optimal solutions, it 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 Sect. 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 Sect. III can solve the folding problem for single-height cells. The model can be solved independently for the \( p \) and \( n \) strips of transistors. The model determines the size and the row assignment of each transistor leg.

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, \( S_n \) and \( S_p \), 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, i.e.,

\[
S_n + S_p \leq S
\]

where \( S \) is a constant that now represents the maximum number of tracks for diffusion.

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:

\[
\forall s \in \{1, \ldots, S\} : s \cdot \text{UseSize}_n(s) \leq S_n, \quad s \cdot \text{UseSize}_p(s) \leq S_p,
\]

where \( \text{UseSize}_n(s) \) and \( \text{UseSize}_p(s) \) 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 \( R \times S \) 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:

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

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

- The rest of constraints (2-5) must be instantiated for each row of the layout.
- A new variable (\( \text{AREA} \)) and \( R \) constraints must be added to calculate the maximum area of all rows. For each row \( r \), the following constraints will be added:

\[
\text{AREA} \geq \sum_{s} \text{DiffSize}_{\text{Gap}} \cdot \text{Nbrakes}_{s} + \sum_{s} \text{SAMESize}_{\text{Gap}} \cdot \text{Nbreaks}_{r} + \sum_{s} \text{DIFFSize}_{\text{Gap}} \cdot \text{Nbridges}_{s}.
\]

- 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: \textit{nppnp}. Figures 1(b) and 1(c) depict double-height cells with the structure \textit{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.

Figure 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 \textit{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 \textit{nppn}. The reader will soon realize that the model can be easily extended for other templates.

If $S_p$ is the maximum size of a p transistor in a p row, then the legs occupying both rows can have a size up to $2S_p + \chi_p$, where $\chi_p$ is a constant that represents the extra size available between the two p rows. In the example of Fig. 1(c), we have $S_p = 4$ 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 Sect. V-B for multiple-height cells must be used. However, adjacent rows are not totally independent since they can allocate large transistors.

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 $\perp$ to denote the double-height row that can be used by merging the top and bottom rows.

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

$$\text{SIZE}_{\min}(t) \leq \sum_{s=1}^{S_p} s \cdot (\lambda(t, \top, s) + \lambda(t, \bot, s)) + \sum_{s=s_p+1}^{2S_p+\chi_p} s \cdot \lambda(t, \perp, s) \leq \text{SIZE}_{\max}(t).$$

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:

$$\text{AREA} \geq N\text{legs}(\top) + N\text{legs}(\bot) + \text{DiffSizeGap} \cdot (N\text{sizes}(\top) + N\text{sizes}(\bot) - 1) + \text{SameSizeGap} \cdot (N\text{breaks}(\top) + N\text{breaks}(\bot)).$$

$$\text{AREA} \geq N\text{legs}(\bot) + N\text{legs}(\perp) + \text{DiffSizeGap} \cdot (N\text{sizes}(\bot) + N\text{sizes}(\perp) - 1) + \text{SameSizeGap} \cdot (N\text{breaks}(\bot) + N\text{breaks}(\perp)).$$

We can observe that the $\perp$ 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 it was discussed in Sect. V-A. For example, for a double-height cell, the constraint (7) could be extended as follows:

$$S_{\text{top}} + S_p + \chi_p + S_{\text{bottom}} + S_{\text{bottom}} \leq S$$

where S is the maximum total height of the cell. Additional constraints on the \textit{UseSize} variables should be included in a similar way as it was discussed in Sect. 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 Sect. 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 a 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 \( \text{InGroup}(n,r,s) \), 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 \( \text{InGroup}(n,r,s) = 1 \) when a transistor connected \( n \) is present in \( G_{r,s} \):

\[
K \cdot \text{InGroup}(n,r,s) \geq \sum_{t \in T(n)} \lambda(t,r,s)
\]

where \( K \) 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:

\[
\text{CostGlobalWires} = \sum_{\forall n,r,s} \text{InGroup}(n,r,s).
\]

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, i.e.,

\[
\text{CostLocalWires} = \sum_{\forall t,r,s} \lambda(t,r,s).
\]

Finally, the total cost of wires can be estimated as

\[
\text{CostWires} = \alpha \cdot \text{CostLocalWires} + \beta \cdot \text{CostGlobalWires}
\]

where \( \alpha \) and \( \beta \) 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 \( \gamma \), i.e.,

\[
\min : \text{Area} + \gamma \cdot \text{CostWires}.
\]

The experimental results in Sect. 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 \( [\text{SIZE}_{\text{min}}(t), \text{SIZE}_{\text{max}}(t)] \). Each leg can have a size in \( \{1, \ldots, S\} \). The two proposed heuristics always use the minimum number of legs for folding transistor \( t \):

\[
L = \left\lceil \frac{\text{SIZE}_{\text{min}}(t)}{S} \right\rceil
\]

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 towards generating legs with size \( S \). Every transistor is folded into \( L \) legs, with \( L-1 \) legs of maximum size \( S \) and one leg with the remaining size \( S' \) such that

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

B. Balanced heuristic

This is an “Euler-friendly” heuristic with two main goals:

- For large transistors, use only legs with size \( S \) whenever possible. If not possible, use only legs with size \( S \) and \( S-1 \).
- Whenever possible, use an odd number of legs with size \( S \).
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 equation (10), the rules to fold a transistor $t$ are as follows:

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

$$\text{SIZE}_{\text{min}}(t) \leq L' \cdot \mathcal{S} + L'' \cdot (S - 1) \leq \text{SIZE}_{\text{max}}(t)$$

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

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

To preserve the evenness of the nodes with size $\mathcal{S}$, the algorithm will select the smallest odd value for $L'$, unless $\text{SIZE}_{\text{min}}(t) = \text{SIZE}_{\text{max}}(t)$ 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 $\mathcal{S} = 4$.

<table>
<thead>
<tr>
<th>$[\text{SIZE}<em>{\text{min}}(t), \text{SIZE}</em>{\text{max}}(t)]$</th>
<th>greedy</th>
<th>balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>[13, 15]</td>
<td>4+4+4+1</td>
<td>4+3+3+3</td>
</tr>
<tr>
<td>[14, 17]</td>
<td>4+4+4+2</td>
<td>4+4+4+4</td>
</tr>
<tr>
<td>[17, 19]</td>
<td>4+4+4+4+1</td>
<td>4+4+4+3+3</td>
</tr>
<tr>
<td>[21, 23]</td>
<td>4+4+4+4+4+1</td>
<td>4+4+4+3+3+3</td>
</tr>
</tbody>
</table>

As an example, the case [17, 19] could have been implemented with three different balanced distributions: $4 + 4 + 3 + 3 + 3$, $4 + 4 + 4 + 3 + 3$ or $4 + 4 + 4 + 4 + 3$. 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 the 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 running at 2.67 GHz and 8 GBytes of memory. Gurobi [10] has been used as MILP solver. Gurobi can efficiently exploit the architecture of multi-core CPUs when solving complex MILP problems.

A. Experimental setup

Transistor folding has been applied to the 45nm 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 has been as follows:

- 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.
- A horizontal pitch $P$ of 130nm 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:

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

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

where $\varepsilon$ 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 $\text{SIZE}_{\text{min}}(t) = 8$ and $\text{SIZE}_{\text{max}}(t) = 12$ tracks.

The experiments have been executed to synthesize standard cells with a maximum number of 5 and 3 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 of differently-sized transistors, respectively.

In these experiments, flexibility has been defined uniformly, i.e., the same value of $\varepsilon$ is applied to all transistors. In a real cell design, flexibility can be non-uniform, 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 ($\varepsilon$). The total area is calculated by adding the area 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 sub-optimal 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

$^7$As an example, INV_X1 has a $p$ and $n$ transistor of 630nm and 415nm, respectively.
### TABLE IV

**TOTAL LIBRARY AREA FOR DIFFERENT FOLDING METHODS AND FLEXIBILITIES (ε) IN TRANSISTOR SIZES.**

<table>
<thead>
<tr>
<th>ε</th>
<th>Optimal</th>
<th>Greedy Balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>1673 (+0.4%)</td>
<td>1682 (+0.5%)</td>
</tr>
<tr>
<td>0.05</td>
<td>1660 (+1.5%)</td>
<td>1667 (+0.4%)</td>
</tr>
<tr>
<td>0.10</td>
<td>1539 (+8.7%)</td>
<td>1546 (+0.5%)</td>
</tr>
<tr>
<td>0.15</td>
<td>1530 (+12.7%)</td>
<td>1538 (+0.5%)</td>
</tr>
<tr>
<td>0.20</td>
<td>1511 (+19.5%)</td>
<td>1527 (+1.1%)</td>
</tr>
<tr>
<td>0.25</td>
<td>1456 (+18.0%)</td>
<td>1505 (+3.4%)</td>
</tr>
<tr>
<td>0.30</td>
<td>1383 (+19.5%)</td>
<td>1447 (+4.6%)</td>
</tr>
</tbody>
</table>

### TABLE V

**AREA RESULTS FOR NANGATE LIBRARY (ε = 0.25)**

<table>
<thead>
<tr>
<th>Cell</th>
<th>Greedy</th>
<th>Balanced</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKBUF X1</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>CLKBUF X3</td>
<td>8</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>CLKGATETST X1</td>
<td>19</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>CLKGATETST X2</td>
<td>22</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>CLKGATETST X4</td>
<td>25</td>
<td>23</td>
<td>21</td>
</tr>
<tr>
<td>CLKGATETST X8</td>
<td>29</td>
<td>29</td>
<td>27</td>
</tr>
<tr>
<td>CLKGATETST X1</td>
<td>15</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>CLKGATE X1</td>
<td>26</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>DFFRS X1</td>
<td>28</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>DFFRS X2</td>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>DFFR X1</td>
<td>24</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>DFFR X2</td>
<td>24</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>DFFS X1</td>
<td>29</td>
<td>29</td>
<td>26</td>
</tr>
<tr>
<td>DFFS X2</td>
<td>29</td>
<td>29</td>
<td>26</td>
</tr>
<tr>
<td>DFF X1</td>
<td>22</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>DFF X2</td>
<td>26</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>DLH X2</td>
<td>15</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>DLL X2</td>
<td>15</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>SDFFRS X1</td>
<td>34</td>
<td>34</td>
<td>32</td>
</tr>
<tr>
<td>SDFFRS X2</td>
<td>37</td>
<td>36</td>
<td>34</td>
</tr>
<tr>
<td>SDFFR X1</td>
<td>30</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td>SDFFR X2</td>
<td>34</td>
<td>31</td>
<td>29</td>
</tr>
<tr>
<td>SDFFS X1</td>
<td>31</td>
<td>31</td>
<td>28</td>
</tr>
<tr>
<td>SDFFS X2</td>
<td>36</td>
<td>33</td>
<td>30</td>
</tr>
<tr>
<td>SDFF X1</td>
<td>28</td>
<td>28</td>
<td>26</td>
</tr>
<tr>
<td>SDFF X2</td>
<td>33</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td>TLAT X1</td>
<td>17</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>Total</td>
<td>671</td>
<td>644</td>
<td>595</td>
</tr>
</tbody>
</table>

deviation from the optimum when the flexibility increases, specially for ε ≥ 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 wirelength. 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. The algorithm aims at minimizing the horizontal wirelength required to connect all transistor terminals.

Figure 9 depicts a symbolic layout produced by the placement tool from the netlists generated by the balanced (top) and optimal (bottom) methods. The 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 (2 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 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 wirelength (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 Equation (9). Column GW reports the value of CostGlobalWires in the same equation. Finally, HW reports the horizontal wirelength after placement.

---

The details of the transistor placement tool are out of the scope of this paper.
The results show a clear impact of the cost function on the final wirelength. 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 non-series/parallel structures. Most of the conventional static CMOS cells with series/parallel structures (NAND, NOR, AOI, OAI) with small transistor sizes (X1 or X2) do not show differences in the final netlists after transistor folding.

Figure 10 depicts the two layouts for one of the cells (CLKGATEST_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 wirelength when using the wire optimization term in the cost function.

Wire optimization has more impact when more flexibility (\( \varepsilon \)) 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 to route the nets in the cell. These lower bounds are the ones that are symbolically depicted in Figures 9 and 10. With 1-D GDRs, these resources will correspond to different metal layers (e.g., M1 and M2). In technologies from 20nm 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.

### Table VI

<table>
<thead>
<tr>
<th>Cell</th>
<th>TR</th>
<th>GW</th>
<th>HW</th>
<th>TR</th>
<th>GW</th>
<th>HW</th>
<th>ΔHW</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOI21_X4</td>
<td>22</td>
<td>4</td>
<td>78</td>
<td>22</td>
<td>4</td>
<td>78</td>
<td>0.0%</td>
</tr>
<tr>
<td>AOI21T_X4</td>
<td>22</td>
<td>8</td>
<td>52</td>
<td>22</td>
<td>8</td>
<td>52</td>
<td>-27.3%</td>
</tr>
<tr>
<td>AOI222_X4</td>
<td>24</td>
<td>9</td>
<td>72</td>
<td>23</td>
<td>9</td>
<td>72</td>
<td>-25.0%</td>
</tr>
<tr>
<td>BUF_X4</td>
<td>12</td>
<td>3</td>
<td>18</td>
<td>11</td>
<td>3</td>
<td>18</td>
<td>0.0%</td>
</tr>
<tr>
<td>BUF_X8</td>
<td>21</td>
<td>3</td>
<td>20</td>
<td>20</td>
<td>3</td>
<td>20</td>
<td>0.0%</td>
</tr>
<tr>
<td>BUF_X32</td>
<td>77</td>
<td>3</td>
<td>154</td>
<td>75</td>
<td>3</td>
<td>154</td>
<td>+1.3%</td>
</tr>
<tr>
<td>CLKGATETST_X4</td>
<td>32</td>
<td>21</td>
<td>106</td>
<td>31</td>
<td>20</td>
<td>106</td>
<td>-38.4%</td>
</tr>
<tr>
<td>CLKGATETST_X8</td>
<td>44</td>
<td>20</td>
<td>174</td>
<td>43</td>
<td>20</td>
<td>174</td>
<td>+7.4%</td>
</tr>
<tr>
<td>CLKGATE_X2</td>
<td>23</td>
<td>17</td>
<td>109</td>
<td>23</td>
<td>18</td>
<td>109</td>
<td>-11.0%</td>
</tr>
<tr>
<td>CLKGATE_X4</td>
<td>31</td>
<td>16</td>
<td>145</td>
<td>30</td>
<td>16</td>
<td>145</td>
<td>-27.6%</td>
</tr>
<tr>
<td>CLKGATE_X8</td>
<td>42</td>
<td>18</td>
<td>194</td>
<td>41</td>
<td>18</td>
<td>194</td>
<td>-21.1%</td>
</tr>
<tr>
<td>DFRRS_X1T</td>
<td>41</td>
<td>36</td>
<td>350</td>
<td>40</td>
<td>33</td>
<td>309</td>
<td>-11.7%</td>
</tr>
<tr>
<td>DFRRS_X2</td>
<td>46</td>
<td>34</td>
<td>390</td>
<td>44</td>
<td>33</td>
<td>320</td>
<td>-18.0%</td>
</tr>
<tr>
<td>DFRR_X1</td>
<td>34</td>
<td>31</td>
<td>247</td>
<td>32</td>
<td>29</td>
<td>197</td>
<td>-20.2%</td>
</tr>
<tr>
<td>DFRR_X2</td>
<td>37</td>
<td>29</td>
<td>236</td>
<td>36</td>
<td>29</td>
<td>236</td>
<td>0.0%</td>
</tr>
<tr>
<td>DFSS_X1</td>
<td>35</td>
<td>29</td>
<td>233</td>
<td>32</td>
<td>29</td>
<td>195</td>
<td>-16.3%</td>
</tr>
<tr>
<td>DFSS_X2</td>
<td>38</td>
<td>28</td>
<td>229</td>
<td>36</td>
<td>27</td>
<td>194</td>
<td>-15.3%</td>
</tr>
<tr>
<td>DFFS_X1</td>
<td>32</td>
<td>26</td>
<td>160</td>
<td>32</td>
<td>24</td>
<td>127</td>
<td>-20.6%</td>
</tr>
<tr>
<td>DFFS_X2</td>
<td>39</td>
<td>26</td>
<td>236</td>
<td>38</td>
<td>23</td>
<td>195</td>
<td>-16.3%</td>
</tr>
<tr>
<td>DFIH_X1</td>
<td>19</td>
<td>14</td>
<td>93</td>
<td>16</td>
<td>13</td>
<td>49</td>
<td>-47.3%</td>
</tr>
<tr>
<td>DFIH_X2</td>
<td>20</td>
<td>13</td>
<td>77</td>
<td>16</td>
<td>13</td>
<td>49</td>
<td>-36.4%</td>
</tr>
<tr>
<td>OA211_X4</td>
<td>22</td>
<td>6</td>
<td>72</td>
<td>20</td>
<td>8</td>
<td>70</td>
<td>-2.8%</td>
</tr>
<tr>
<td>OA222_X4</td>
<td>24</td>
<td>9</td>
<td>96</td>
<td>23</td>
<td>9</td>
<td>95</td>
<td>-1.0%</td>
</tr>
<tr>
<td>SDFRRS_X2</td>
<td>54</td>
<td>39</td>
<td>443</td>
<td>54</td>
<td>37</td>
<td>365</td>
<td>-17.6%</td>
</tr>
<tr>
<td>SDFRR_X2</td>
<td>47</td>
<td>26</td>
<td>222</td>
<td>46</td>
<td>25</td>
<td>221</td>
<td>-0.5%</td>
</tr>
<tr>
<td>SDFSS_X1</td>
<td>39</td>
<td>30</td>
<td>222</td>
<td>38</td>
<td>30</td>
<td>214</td>
<td>-17.1%</td>
</tr>
<tr>
<td>TLAT_X1</td>
<td>23</td>
<td>16</td>
<td>137</td>
<td>20</td>
<td>17</td>
<td>76</td>
<td>-44.5%</td>
</tr>
</tbody>
</table>

Total 861 490 4451 818 480 3716 -16.5%

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

### 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 3-5-3 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.

The number of cells affected by the optimization is much larger that for single-height cells (108 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 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.

9 The estimation of horizontal wirelength 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.
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 seconds. However, the worst case was observed for cell SDFFRS_X1 (164.91 secs).

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 1 with regard to the number of iterations of the main loop. A new iteration is executed when MILP \text{AREA}(S) \neq \text{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.

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. However, since the diffusion sizes are determined after the Eulerization, this may lead to an overestimation of the Eulerization cost. This is because the MILP model uses this flexibility to reduce the diversity of transistor sizes. However, this will limit the space of solutions for gate sizing.

As a by-product, 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 transistors: 1 and 2 slots. The following facts can be observed:

- 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.
- As expected, the area overhead approximately doubles when the cost of the diffusion breaks also doubles (Gap=2).
- 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 of 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 trade-off is necessary.
something that should be further investigated in the future.

IX. Conclusions

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 semi-custom 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 sub-problems 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