Simultaneous Transistor Folding and Placement in Standard Cell Layout Synthesis

Kyeonghyeon Baek* and Taewhan Kim†
Department of Electrical and Computer Engineering, Seoul National University, Seoul, Korea
{*khbaek, †tkim}@snucad.snu.ac.kr

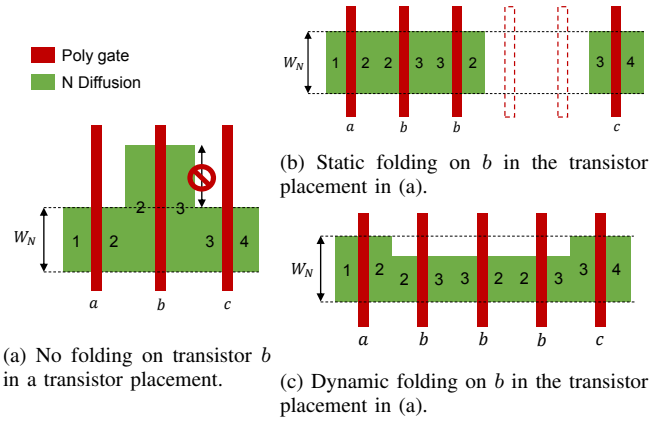
2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643537

***Abstract*—The three major tasks in standard cell layout synthesis are transistor folding, transistor placement, and in-cell routing, which are tightly inter-related, but generally performed one at a time to reduce the extremely high complexity of design space. In this paper, we propose an integrated approach to the two problems of transistor folding and placement. Precisely, we propose a globally optimal algorithm of search tree based design space exploration, devising a set of effective speeding up techniques as well as dynamic programming based fast cost computation. In addition, our algorithm incorporates the minimum OD (oxide diffusion) jog constraint, which closely relies on both of transistor folding and placement. To our knowledge, this is the first work that tries to simultaneously solve the two problems. Through experiments with the transistor netlists and design rules in the ASAP 7nm library, it is shown that our proposed method is able to synthesize fully routable cell layouts of minimal size within 1 second for each netlist, outperforming the cell layout quality in the ASAP 7nm library, which otherwise, may take several hours or days to manually complete layouts of the quality level comparable to ours.**

I. Introduction

Standard cell layout synthesis belongs to a constrained optimization problem whose most important objective is to find a solution that minimizes the cell area. Since cell layout optimization is NP-hard [1], it is generally performed in a sequence of three steps: transistor folding, transistor placement, and in-cell routing, in which *transistor placement* plays a key role in minimizing cell area. Through transistor placement, cell of minimum area, regardless of in-cell routability, can be obtained by ordering transistors in a way to maximize the possibility of diffusion sharing between two transistors adjacent to each other. A lot of methods of transistor placement have been proposed. The existing methods can be classified as: integer linear programming (ILP) based (e.g., [2], [3]), graph theory based (e.g., [4], [5]), Boolean satisfiability (SAT) based (e.g., [6], [7]), simulated annealing based (e.g., [8]), and search tree exploration based (e.g., [9]).

In practical cell layout synthesis, the size of each transistor is determined individually by taking into account the cell's performance constraints. For a pre-specified limit on the width, denoted by W_P and W_N, of P and N diffusions, transistors of large size cannot be implemented with single channels. *Transistor folding* is the process of splitting a large transistor into multiple small (sub)-transistors, called *fingers*, that are connected in parallel and placed contiguously with diffusion sharing [10]. *Static folding* refers to folding every transistor in the cell with its minimal number of fingers such that their channel widths[1] are exactly the same while *dynamic folding* refers to folding the transistors individually with some number of fingers of channel width that is less than or equals to the value of W_P or W_N. Fig. 1(a) illustrates no folding on transistor b in a transistor placement, resulting in the violation of W_N constraint in b. On the other hand, Fig. 1(b) shows a static folding on transistor b, resolving the violation, but requiring a diffusion break between transistors b and c. Fig. 1(c) shows a dynamic folding on b. It creates one more fingers ($2 \rightarrow 3$) over that in Fig. 1(b), but enables a diffusion sharing between b and c. Note that dynamic folding is a generalization of static folding. Given W_P and W_N, the cell size is determined by the following closely inter-related factors: (1) the number and orientation (i.e., flip or no-flip) of fingers for each transistor, (2) the placement of fingers, and (3) the amount of diffusion sharing.



(a) No folding on transistor b in a transistor placement.

(b) Static folding on b in the transistor placement in (a).

(c) Dynamic folding on b in the transistor placement in (a).

Fig. 1. An illustration of transistor folding on a transistor and the impact of folding on diffusion sharing. (W_N represents the maximum width constraint of N diffusion.)

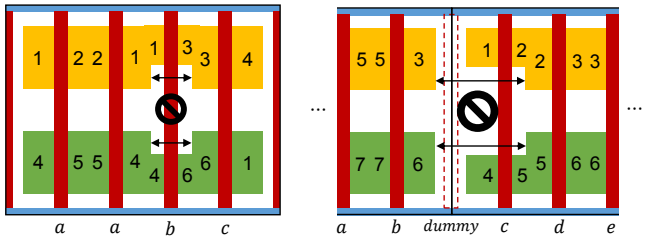
To our knowledge, all existing works have addressed transistor placement with no or partial consideration of transistor folding (e.g., [2]–[4], [9], [10]). For a set of pMOS and nMOS transistor sizes in a cell, Kim and Kang [10] proposed a polynomial-time algorithm of finding an optimal folding size, equivalently optimal values of W_P and W_N, which leads to minimize the resulting cell size. The algorithm has two disabilities. It did not consider the diffusion breaks which may be caused by the transistor placement and finger orientation. Further, it assumed to produce all fingers of identical size.

[1]The channel width means to the size of the finger.

Gupta and Hayes [2] formulated the combined problem of transistor folding and placement as a 0-1 integer linear programming (ILP) problem to produce optimal results. However, in a strict sense, they attempted to solve the problem of, so-called *transistor placement with static folding* rather than to solve the problem of *transistor placement with dynamic folding*, in that they fixed the number of fingers to $\lceil \frac{t_P}{W_P} \rceil$ or $\lceil \frac{t_N}{W_N} \rceil$ for each pMOS or nMOS transistor where t_P and t_N respectively represent the size of the pMOS or nMOS transistor, and then determined their position and orientation to minimize the cell size. Lu *et al.* [3] proposed an ILP based approach to the problem of transistor placement combined with sizing of individual fingers to minimize the resulting cell area. However, the approach assumed that the number of fingers for each transistor has been given as input and never considered as a parameter to be optimized. To sum up, the works in [2], [3], [10] did not fully exploit the various effects of dynamic folding on diffusion sharing, ultimately on reducing the cell size.

On the other hand, Cortadella [4] constructed a graph model from the transistor netlist of an input cell to formulate an instance of dynamic folding problem, and solved the folding problem by minimally adding new edges to the graph until the existence of Eulerian path in the graph is guaranteed. He applied a mixed ILP formulation to find the minimal number of extra edges called *Eulerization cost*, which was used to estimate the number of diffusion breaks in transistor placement. Since the method processed two graphs independently, one for netlist of nMOS transistors and one for pMOS, the Eulerization costs totally ignored the gate sharing of transistors, also known as *transistor pairing*, in placement. Recently, Cleef *et al.* [9] proposed a search tree based exploration approach to find the best placement and dynamic folding of transistors. As a search tree pruning technique, they estimated the number of diffusion breaks by using a graph model similar to that in [4]. They performed pMOS transistor placement and nMOS transistor placement separately and combined the two placement solutions for generating entire placement. The fundamental limitation of performing pMOS and nMOS placements separately is that it can incur many *gate (vertical) misalignments*. Gate misalignment incurs when a pair of pMOS and nMOS transistors sharing a net are not aligned vertically in a single column. Since those two transistors cannot be implemented with a vertical straight poly gate, a metal resource is required to connect them, which eventually hinders in-cell routability.

Clearly, it is highly desirable to solve the two problems of transistor folding and placement in an integrated fashion. However, all prior works have not proposed fully integrated solutions. One of the main reasons is an exponential time complexity on exploring the search space. Nevertheless, as the technology scaling continues, solving the two problems in an integrated framework is appealing since it introduces new DRs (design rules), which are closely affected by both of transistor folding and placement. One such noticeable DR is the minimum OD (oxide diffusion) jog rule [11]: Transistors



(a) OD jog violation occurs when synthesizing cell layout. (b) OD jog violation occurs when abutting cells in cell placement.

Fig. 2. Design rule for minimum OD (oxide-diffusion) jog.

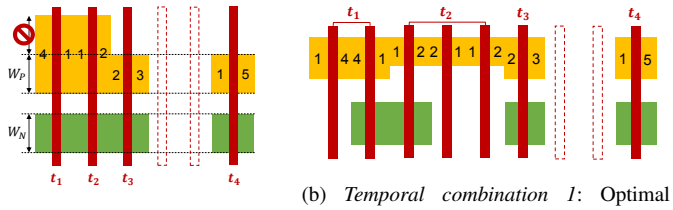
can have different OD region heights according to dynamic folding. When transistors in a cell with different OD heights abut or cells with different OD heights in the cell boundaries abut, OD jog violation can occur as shown in Figs. 2(a) and (b).

This work overcomes the critical limitation of existing works, that is, not fully integrating transistor folding into transistor placement. The distinct features of our work can be summarized as:

1. We *propose a search tree based algorithm that exhaustively explores the solution space of transistor folding and placement*. Unlike the conventional methods targeting to generate exactly one compact (optimal) layout for each input netlist of transistors in a cell, our algorithm is able to produce diverse optimal layouts, so that designers can choose the layout that is best suited to the implementation objectives and constraints.
2. To be computationally viable, we *devise a set of well-defined speeding up techniques as well as dynamic programming based fast cost computation* to effectively and efficiently prune the search space of transistor folding and placement without losing optimality.
3. We take into account the minimum OD jog constraint in our integrated algorithm of transistor folding and placement, so that *OD jog violation never occurs inside the synthesized cells.* We show in experiments that tightly linking the two tasks of transistor folding and placement to avoid OD jog violation can save more cell area, which otherwise, were necessary to resolve OD jog violations afterwards.

II. Motivations

If transistor folding is performed before or after transistor placement, optimal solutions may not be found. Fig 3 shows the impact of transistor folding on the cell width (i.e., cell size). Fig 3(a) shows an optimal transistor placement for four transistor pairs t_1, t_2, t_3, and t_4 with no transistor folding, which requires total of 6 poly gates including 2 dummy gates to make a diffusion break. Note that for transistor pairs t_1 and t_2, each has P diffusion region of width larger than W_P. Thus, it is required that the transistors of t_1 and t_2 in P diffusion region should be folded. There will be three possible temporal combinations of transistor folding and placement:



(a) Optimal placement with no consideration of folding.

(b) *Temporal combination 1*: Optimal placement and then optimal folding, allocating 9 poly gates.

(c) *Temporal combination 2*: Optimal folding and then optimal placement, allocating 8 poly gates.

(d) *Temporal combination 3*: Simultaneous optimal folding and placement, allocating 7 poly gates.

Fig. 3. An example illustrating three possible temporal combinations of transistor folding and placement.

- **Combination 1** (*placement* $\rightarrow$ *folding*): performing transistor placement first and then performing transistor folding. For example, Fig. 3(b) shows an optimal folding result for the optimal transistor placement in Fig. 3(a), in which to avoid diffusion breaks between t_1 and t_2 and between t_2 and t_3, t_1 and t_2 are folded with two and three fingers, respectively. This results in a total of 9 poly gates including 2 dummy gates between t_3 and t_4.
- **Combination 2** (*folding* $\rightarrow$ *placement*): performing transistor folding first and then performing transistor placement. For example, Fig. 3(c) shows an optimal placement result for the minimal number of fingers produced by applying folding individually to the transistor pairs in Fig. 3(a). This results in total of 8 poly gates including 2 dummy gates. Most of the existing works belong to this temporal combination (e.g., [2], [3], [5], [6]).
- **Combination 3** (*folding* + *placement*): performing transistor folding and placement simultaneously. For example, Fig. 3(d) shows an optimal result of folding and placement for the four transistor pairs in Fig. 3(a), saving two and one poly gates in comparison with the results produced by *Combinations* 1 and 2, respectively.

As illustrated in Fig. 3, it is highly desirable to perform two tasks of transistor folding and placement in an integrated framework, that is, *Combination 3* to produce cell layouts of minimal area.

III. Algorithm for Simultaneous Transistor Folding and Placement

A. Problem Definition

Let t_i = (t_i^p and t_i^n) be a pair of pMOS and nMOS transistors sharing a poly gate in an input transistor netlist of CMOS standard cell. We assume a transistor pair set $\mathcal{T}$ is composed of such K pairs of transistors. Each pMOS or nMOS transistor t_i^x, $x = p$ or n is characterized by a tuple $(n_s, n_g, n_d, size, type)$ where n_s, n_g, and n_d represent source, poly gate, and drain nets of t_i^x, respectively, $size(t_i^x)$ denotes the transistor size[2], and $type \in \{pMOS, nMOS\}$ indicates the transistor type. Design rules to be satisfied in the process of transistor folding and placement are the followings. (Constant parameters are shown in parentheses.)

- **DR 1** (*Diffusion break constraint* (N_{dummy}^{gate})): If two transistors adjoining each other have different diffusion nets in between them, at least N_{dummy}^{gate} number of dummy poly gates should be inserted between the transistors. Typically, the value of N_{dummy}^{gate} is 1 or 2 depending on the process technology used.
- **DR 2** (*Diffusion width constraint* (W_P, W_N, W_{MIN})): Diffusion widths, in terms of the number of fins, of all transistors should be larger than or equal to the value of W_{MIN} but should not exceed the value of W_P for pMOS or the value of W_N for nMOS.
- **DR 3** (*Oxide diffusion jog constraint* (L_{OD})): The oxide diffusion region height should be uniformly maintained. The uniformed height should be lengthened to be more than the value of L_{OD}.

Let $\Lambda(t_i, \cdot)$ be an instance of folding shape that can be produced by applying dynamic folding to both t_i^p and t_i^n to satisfy DR 2 i.e. the diffusion width constraint and DR 3 i.e. OD jog constraint. In addition, let S_i (= $\{\Lambda(t_i, 1), \Lambda(t_i, 2) \cdots\}$) be the set of all valid folding shapes of t_i and $len(\Lambda(t_i, \cdot))$ denote the horizontal length of folding shape $\Lambda(t_i, \cdot)$, expressed in terms of the number of poly gates. (Details on folding shapes and their generation will be described in Sec. III-C.) Then, we define a function, called *folding function*, $\mathcal{F}(t_i)$, $i = 1, 2, \cdots, K$:

$$\mathcal{F}(t_i) = \Lambda(t_i, \cdot) \in S_i, \tag{1}$$

which maps transistor pair t_i to a legal folding shape in set S_i.

Problem 1 (*Transistor folding and placement*): *For an input transistor netlist of CMOS standard cell with K pairs of transistors, find (i) a mapping function $\mathcal{F}(\cdot)$ and (ii) a linear order of the folding shapes $\mathcal{F}(t_1), \mathcal{F}(t_2), \cdots, \mathcal{F}(t_K)$ which minimizes the quantity of Cost:*

$$Cost = \sum_{i=1,\cdots,K} len(\mathcal{F}(t_i)) + N_{dummy}^{gate} \cdot N_{break} - \alpha_{tot} \tag{2}$$

while satisfying DR 1, 2, and 3, where N_{break} is the number of diffusion breaks inserted between the folding shapes to meet DR 1 i.e. diffusion break constraint, and α_{tot} is the total number of vertical lines of poly gates that can be saved by abutting folding shapes[3].

B. Overall Flow

Fig. 4 shows the flow of our proposed cell layout generator, called CSyn-fp (Cell Synthesis with simultaneous Folding and Placement), which performs the following four steps: For an input transistor netlist $\mathcal{N}$ of cell with K transistor pairs

[2]We set $size(t_i^x)$ to the number of fins required in implementation.

[3]For example, see Fig. 7.

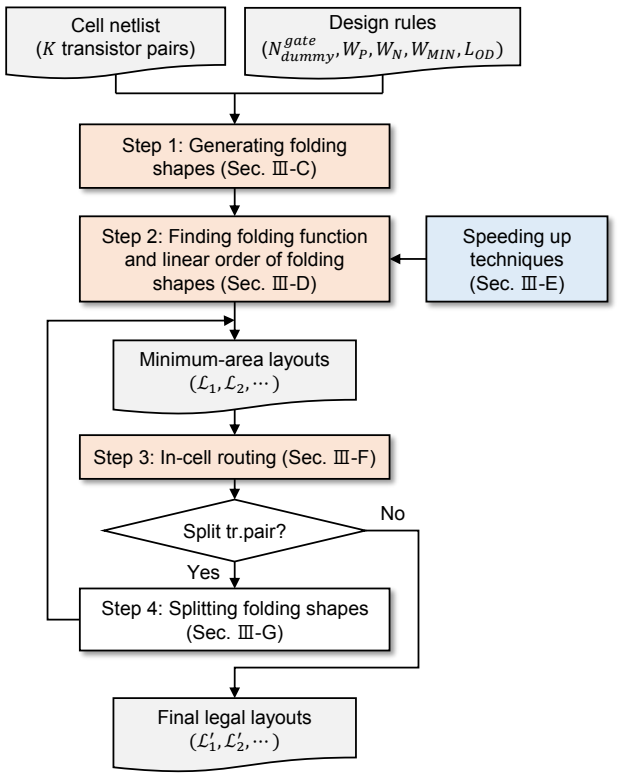


Fig. 4. The flow of our proposed cell layout generator CSyn-fp.

and design rule parameters (N^{gate}_{dummy}, W_P, W_N, W_{MIN} and L_{OD}), (Step 1) *enumerating the set of all feasible folding shapes* S_i with no DR 2 and DR 3 violations for every transistor pair $t_i, i = 1, \cdots, K$ in $\mathcal{N}$; (Step 2) *building-up a search tree based design space exploration* to find mapping functions $\mathcal{F}(\cdot)$ and linear orders of the folding shapes $\mathcal{F}(t_1), \mathcal{F}(t_2), \cdots, \mathcal{F}(t_K)$ which minimize the quantity of $Cost$ in *Eq.2* while considering the impact of DR 1 and DR 3; (Step 3) *applying a conventional in-cell router* to the folding and placement results $\mathcal{L}_1, \cdots$ in Step 2 to produce legal and complete layouts $\mathcal{L}'_1, \cdots$ corresponding to $\mathcal{N}$. (Step 4) *splitting folding shapes* in solutions of Step 3 to further reduce cell area. (It is an optional step since it may induce additional in-cell routing resource.);

C. Step 1: Generation of Folding Shapes

We illustrate our method of folding shape generation for a transistor pair $t_i = (t^p_i, t^n_i) \in \mathcal{T}$ using an example shown in Fig. 5(a) where $size(t^p_i) = 6$, $size(t^n_i) = 4$, $W_P = 4$, $W_N = 3$, and $W_{MIN} = 2$.

Step 1.1 (*Generating all folding configurations for pMOS* t^p_i): This step generates all feasible folding configurations for t^p_i while satisfying $W_P = 4$ and $W_{MIN} = 2$. For example, configuration Γ^p_1, shown in Fig. 5(b), is composed of two fingers, one with 4 fins and the other with 2 fins, thus total of 6 fins (= $size(t^p_i)$). Γ^p_2 is produced by cyclic orientation of the diffusion nets by one-step move to the left or right while Γ^p_3 and Γ^p_4 are produced by applying left-to-right flipping to Γ^p_1 and Γ^p_2, respectively.

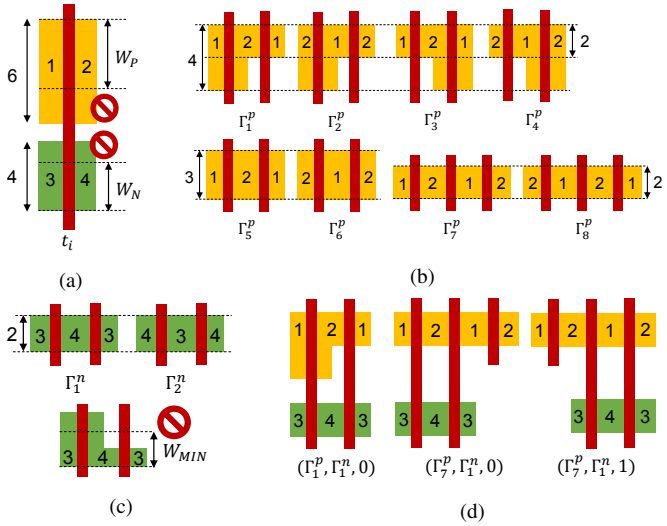


Fig. 5. Illustration of folding shape generation. (a) Specification of a transistor pair $t_i = (t^p_i, t^n_i) \in \mathcal{T}$ and diffusion width constraints: $size(t^p_i) = 6$, $size(t^n_i) = 4$, $W_P = 4$, $W_N = 3$, and $W_{MIN} = 2$. (b) Generating all folding configurations for t^P_i. (c) Generating all folding configurations for t^N_i. (d) Generating all folding shapes $(\Gamma^p_{j1}, \Gamma^n_{j2}, \delta)$.

Step 1.2 (*Generating all folding configurations for nMOS* t^n_i): This step is identical to step 1.1 except that the transistor to be folded is now t^n_i. For example, since $W_{MIN} = 2$, the bottom configuration shown in Fig. 5(c), which has a finger of size 1, is not allowed while Γ^n_1 and Γ^n_2 (cyclic orientation of Γ^n_1) are both feasible nMOS folding configurations.

Step 1.3 (*Generating all folding shapes* $\Lambda(t_i, \cdot) = (\Gamma^p_{j1}, \Gamma^n_{j2}, \delta)$ *for the pair* t_i): A distinct folding shape can be produced by a pair of pMOS and nMOS configurations and offset value (δ) with respect to the left alignment of the configurations. For example, for Γ^p_1 and Γ^n_1, the minimum length folding shape is only that labeled as (Γ^p_1, Γ^n_1, 0) in Fig. 5(d) whereas for Γ^p_7 and Γ^n_1, the minimum length folding shapes are the those labeled as (Γ^p_7, Γ^n_1, 0) (Γ^p_7, Γ^n_1, 1) in Fig. 5(d).

It should be noted that since some transistor pairs have the same values of pMOS size (i.e., the number of fins) and the same values of nMOS size, to avoid redundant computation, CSyn-fp stores only the distinct folding shapes in a lookup table with pMOS and nMOS sizes as key.

D. Step 2: Search-tree Based Design Space Exploration

Fig. 6 shows a conceptual view of our search tree based design space exploration for finding linear orders of minimal length for the transistor pairs in $\mathcal{T}$ of input transistor netlist. Whenever a new node (e.g. (t_1, t_3, t_2) in Fig. 6) is expanded in the tree traversal, CSyn-fp extracts a minimal amount of information on the partial order of the least cost from the information retained in its parent node (e.g., (t_1, t_3)) and will retain the information to be used for its children. (The details will be described in the following.) In addition, CSyn-

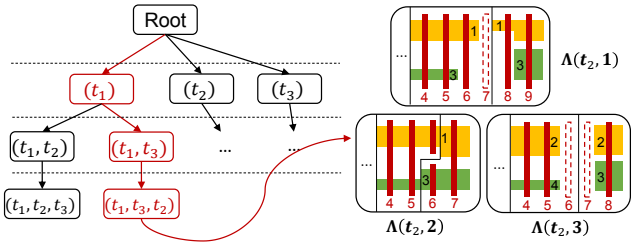


Fig. 6. A conceptual view of a search tree based design space exploration by CSyn-fp for finding linear orders of minimal length for K transistor pairs. CSyn-fp employs a fast cost computation based on dynamic programming as well as a set of effective speeding up techniques.

fp employs a number of simple but effective speeding up techniques, which will be described in Sec. III-E.

Definition 1 (Cost formulation for partial linear placement of folding shapes)**.** *$C[t_{i_1}, t_{i_2}, \cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j)]$ is defined to be the minimal number of poly gates including dummy gates that are required for making the linear order listed as $t_{i_1}, t_{i_2}, \cdots, t_{i_{k-1}}, t_{i_k}$ under the condition that folding shape of t_{i_k} should be $\Lambda(t_{i_k}, j)$.*

Then, we can derive a recurrence relation for $k \geq 2$:

$$C[t_{i_1}, \cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j_1)]$$
$$= \min_{j_2 \in U} \{C[t_{i_1}, \cdots, t_{i_{k-1}} | t_{i_{k-1}} = \Lambda(t_{i_{k-1}}, j_2)] + len(\Lambda(t_{i_k}, j_1))$$
$$+ \max(\mu_{DB} \cdot N^{gate}_{dummy},\ \mu_{OD} \cdot L_{OD}) - \alpha\}. \quad (3)$$

where $U = \{1, \cdots, |S_{i_{k-1}}|\}$[4], and μ_{DB} = 0 if the left side of folding shape $\Lambda(t_{i_k}, j_1)$ can abut on the right side of $\Lambda(t_{i_{k-1}}, j_2)$ without a diffusion break and μ_{DB} = 1, otherwise. If the minimum OD jog violation occurs when $\Lambda(t_{i_{k-1}}, j_2)$ and $\Lambda(t_{i_k}, j_1)$ abut to each other, μ_{OD} = 1. Otherwise, μ_{OD} = 0. α is the number of vertical line of poly gates saved by abutting the folding shapes.
In addition, for $k = 1$:

$$C[t_i | t_i = \Lambda(t_i, j)] = len(\Lambda(t_i, j)), \forall t_i \in \mathcal{T}. \quad (4)$$

Definition 2 (Cost formulation for full linear placement of folding shapes)**.** *$C[t_{i_1}, t_{i_2}, \cdots, t_{i_K}]$ is defined to be $\min_{j \in U}\{C[t_{i_1}, t_{i_2}, \cdots, t_{i_K} \mid t_{i_K} = \Lambda(t_{i_K}, j)]\}$ where $U = \{1, \cdots, |S_{i_K}|\}$.*

Then, the minimum among the values of $C[t_{i_1}, t_{i_2}, \cdots, t_{i_K}]$ for every linear order of the K transistor pairs in $\mathcal{T}$ is exactly the quantity of the minimal $Cost$ in *Eq.2* in Problem 1.

Fig. 7 shows an illustration of calculating the values of $C[\cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j_1)]$, j_1 = 1, 2, and 3 in *Eq.3* and the chosen (parent) folding shapes by utilizing the parent information of $C[\cdots, t_{i_{k-1}} | t_{i_{k-1}} = \Lambda(t_{i_{k-1}}, j_2)]$, j_2 = 1, 2, and 3. For example, CSyn-fp obtains the value of $C[\cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, 2)]$, shown on the second row and third column in Fig. 7 by computing the sum of the minimum of $C[\cdots, t_{i_{k-1}} | t_{i_{k-1}} = \Lambda(t_{i_{k-1}}, j)] + \max(\mu_{DB_j} \cdot N^{gate}_{dummy}, \mu_{OD_j} \cdot L_{OD})$ - α_j for j = 1, 2, and 3, and $len(\Lambda(t_{i_k}, 2))$, which is $\min\{6+0-1, 5+2-0, 6+1-1\}$ + 2 = 7 where N^{gate}_{dummy} and L_{OD} are set to 2 and 1, respectively, and $len(\Lambda(t_{i_k}, 2)) = 2$ poly gates, as shown in Fig. 7. Note that $\alpha_1 = 1$ since abutting $\Lambda(t_{i_k}, 2)$ on $\Lambda(t_{i_{k-1}}, 1)$ saves one vertical space of poly gate. The red arrow in Fig. 7 indicates the folding shaping combination with the least cost.

[4] $S_{i_{k-1}}$ is the set of folding shapes for transistor pair $t_{i_{k-1}}$.

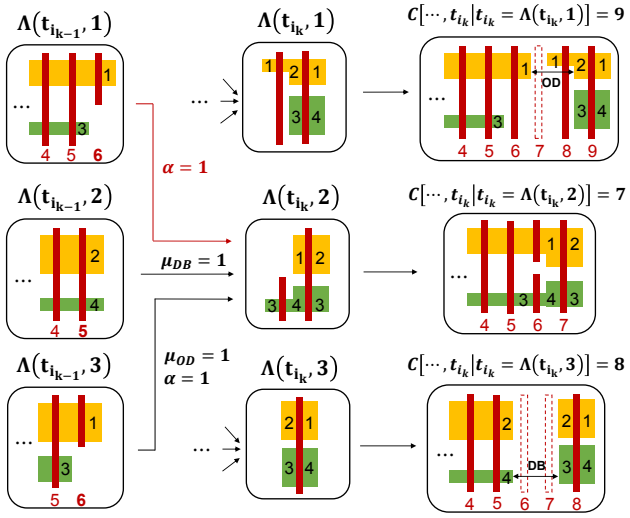


Fig. 7. Example illustrating the dynamic programming based calculation of the values of $C[\cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j_1)]$, j_1 = 1, 2, and 3 in *Eq.3* together with the corresponding (parent) folding shapes by utilizing the parent information of $C[\cdots, t_{i_{k-1}} | t_{i_{k-1}} = \Lambda(t_{i_{k-1}}, j_2)]$, j_2 = 1, 2, and 3.

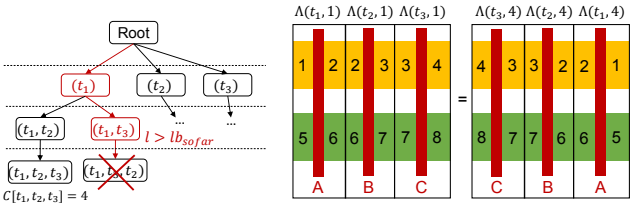
E. Speeding up Techniques

• **Pruning partial linear placement of folding shapes**: Let us suppose that the cost computation of the current node in the search tree is completed, producing a set of optimal (conditional) partial linear placement of folding shapes corresponding to $C[\cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j)]$, $j = 1, 2, \cdots |S_{i_k}|$.

Definition 3 (Dominance relation)**.** *For two partial linear placements of folding shapes, $\mathcal{L}_1$ and $\mathcal{L}_2$, corresponding to $C[\cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j_1)]$ and $C[\cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j_2)]$, it is said that $\mathcal{L}_1$ dominates $\mathcal{L}_2$ if and only if the following four conditions are satisfied.*

1. $len_p(\mathcal{L}_1) \leq len_p(\mathcal{L}_2)$ where $len_p(\mathcal{L}_i), i = 1, 2$ is the horizontal length, in terms of the number of vertical spaces for poly gates, up to the rightmost pMOS poly gate for $\mathcal{L}_i$.
2. $len_n(\mathcal{L}_1) \leq len_n(\mathcal{L}_2)$ where $len_n(\mathcal{L}_i), i = 1, 2$ is the horizontal length, in terms of the number of vertical spaces for poly gates, up to the rightmost nMOS poly gate for $\mathcal{L}_i$.
3. The two diffusion nets on the right of the rightmost pMOS in $\Lambda(t_{i_k}, j_1)$ and in $\Lambda(t_{i_k}, j_2)$ are the same.
4. The two diffusion nets on the right of the rightmost nMOS in $\Lambda(t_{i_k}, j_1)$ and in $\Lambda(t_{i_k}, j_2)$ are the same.

For example, the partial linear placement corresponding to $C[\cdots, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, 3)]$ on the bottom row and right



(a) Pruning branches based on lower bound. (b) Avoiding redundant trials of linear placements by cell flipping in placement and folding shape flipping in CSyn-fp.

Fig. 8. Illustration of search tree pruning based on lower bound and placement symmetry.

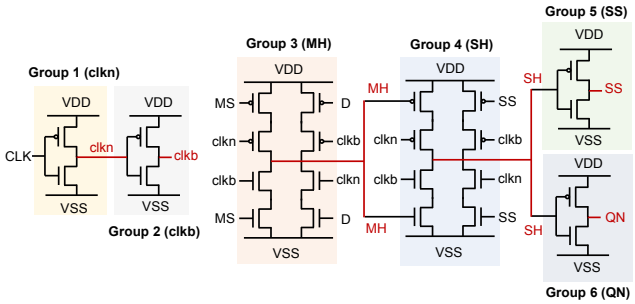


Fig. 9. Example of *articulation* point based netlist graph partitioning for cell DFFHQNx1 in ASAP7 7nm library.

column in Fig. 7 dominates the order corresponding to $C[\cdots, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, 1)]$ on the first row and right column in Fig. 7. Whenever a new node is created in the search tree, CSyn-fp keeps only the most dominating partial linear placements.

• **Pruning based on lower bound of minimal length of linear placement**: From a current node in search tree, CSyn-fp does not expand the search tree further if the following two conditions are satisfied. (An illustrated example is shown in Fig. 8(a).)

1. Let $l^p_{current}$ be the smallest horizontal length of pMOS diffusion region among all the partial linear placements corresponding to the current node and $l^p_{unvisit}$ be the total sum of the smallest numbers of poly gates among the pMOS folding configurations of the transistor pairs that are unexplored yet. Then, it should be that $l^p_{current} + l^p_{unvisit} \leq lb_{sofar}$ where lb_{sofar} is the minimal length among the full linear placements of all transistor pairs explored so far.
2. By the same token, it should be that $l^n_{current} + l^n_{unvisit} \leq lb_{sofar}$.

• **Eliminating linear placement redundancy by cell flipping**: Since a cell can be flipped in the cell placement and CSyn-fp considers all feasible folding shapes including their flipped ones, a linear placement of all transistor pairs from left to right by our CSyn-fp also implicitly includes the consideration of the linear placement from right to left. (An illustrated example is shown in Fig. 8(b).) Consequently, it suffices for CSyn-fp to constrain one particular transistor pair in $\mathcal{T}$ to be placed only to the first half of full linear placement.

• **Partitioning netlist for large cells**: For a cell with large number of transistor pairs like flip-flop cells, CSyn-fp partitions the netlist into a number of sub-nets of reasonable size. We observe that some transistor pair in a certain group of transistor pairs in netlist should not necessarily be placed in adjacent to some transistor pair in another group of transistor pairs. For example, the schematic of cell DFFHQNx1 in ASAP 7nm library shown in Fig. 9 has 6 such groups. An *articulation* point in netlist graph can be a candidate to split the netlist into multiple parts, each of which can be then treated a single unit for group-level transistor folding and placement. In other words, CSyn-fp performs hierarchical two-level transistor folding and placement. At the first level, CSyn-fp finds an optimal solution for each group of transistor pairs. At the second level, CSyn-fp finds an optimal solution for the cell by exhaustively exploring the group placements.

F. Step 3: In-cell Routing

We can apply any of conventional in-cell routers to the solutions produced by Steps 1 and 2 in CSyn-fp. In our experiment, we adopt the router in [5].

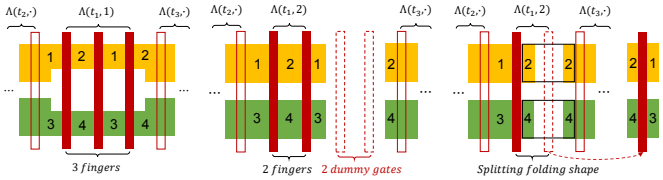
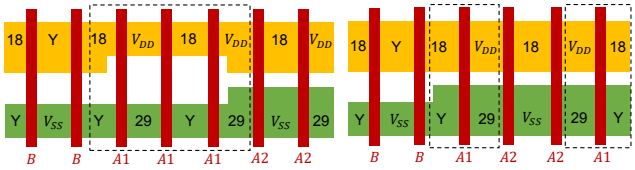


Fig. 10. Impact of splitting folding shape on cell size.

G. Step 4 (optional): Splitting Folding Shapes

CSyn-fp has used the folding shapes as basic units since treating the fingers in the folding shapes individually requires additional metal resource to connect the fingers apart. However, splitting the fingers in folding shapes may reduce the cell size, as illustrated in Fig. 10. Fig. 10(a) shows a section of transistor folding and placement, in which folding shape $\Lambda(t_1, 1)$ consists of three fingers. This is because if transistor pair t_1 were replaced with $\Lambda(t_1, 2)$ of two fingers, as shown in Fig. 10(b), two dummy poly gates ($N^{gate}_{dummy} = 2$) should be needed to break diffusion. However, if one finger in $\Lambda(t_1, 2)$ (generally half of the fingers in the folding shape of even number of fingers) is displaced as shown in Fig. 10(c), the dummy gates are not needed, thereby providing a potential saving of cell area.

CSyn-fp implements this idea as an optional step as follows. For each folding shape with an odd number of fingers in a layout solution, $\mathcal{L}_i$, obtained in Step 3 of CSyn-fp, we update the folding shape to have one fewer number of fingers and then perform an iterative process of swapping fingers in a short distance while controlling the number of iterations. We repeat this process for all folding shapes of odd number of fingers in $\mathcal{L}_i$. For example, Figs. 11(a) and (b) show the layouts of



(a) Before folding shape splitting (b) After folding shape splitting

Fig. 11. Layouts of AOI21x1 in ASAP 7nm library [12] produced before and after the application of our refinement step in CSyn-fp.

cell AOI21x1 in ASAP 7nm library [12] produced before and after the application of our refinement step (i.e., folding shape splitting) in CSyn-fp, respectively. It is shown that by splitting the folding shape with dashed box and moving one split finger labeled $A1$ to the right of fingers labeled $A2$, CSyn-fp reduces the total number of poly gates from 7 to 6.

IV. Experimental Results

We implement CSyn-fp using C++ on a linux machine with Intel i7-8700K 4.7GHz CPU and 64GB memory. CSyn-fp of transistor folding and placement is applied to the ASAP 7nm standard cell library publicly available in [12] in which the design rule parameters have been set as: $N^{gate}_{dummy} = 2$, $W_{MIN} = 1$, $W_P = 3$, $W_N = 3$ and $L_{OD} = 1$. CSyn-fp synthesizes 165 cells in the ASAP 7nm library, followed by applying the in-cell router in [5].

A. Cell Quality by Optimal Cell Size Exploration

• **Assessing cell layout quality for the netlists of ALL 165 cells in the ASAP 7nm library**: Table I summarizes the comparison of the average size of all cells of each cell type produced by CSyn-fp with in-cell routing completion and in ASAP 7nm library. For example, for cell types XOR and XNOR which have the netlists for six cells XORx1, XORx2, XORxp5, XNORx1, XNORx2, and XNORxp5 where XORx2 and XNORx2 each consists of 12 FETs while for the rest, each consists of 10 FETs, CSyn-fp produces the cell layouts of 10.3 CPPs (contacted poly pitches) on average corresponding to the six netlists. On the other hand, the ASAP 7nm library has the cell layouts of 10.7 CPPs on average for the same six netlists, which is 3.12% more CPPs in comparison with that produced by CSyn-fp.

• **Assessing cell layout quality for the netlists of TOP 10 best-synthesized cells in the ASAP 7nm library**: Table II summarizes the comparison of the average number of CPPs (i.e. cell size), total M1 (metal 1) wire length and the total number of V0 vias (i.e., vias in between M1 and MOL (middle-of-line) layers) used in each cell layout for top 10 best-synthesized cells produced by CSyn-fp with in-cell routing completion and in the ASAP 7nm cell library [12]. CSyn-fp respectively shows 10.4% and 9.9% of improvements on average in the number of CPPs and the total M1 wire length. A slight increase in the number of vias is mainly due to the increase in the number of fingers in order to avoid diffusion breaks at the cost of the increased number of diffusion regions to be connected, requiring V0 vias.

TABLE I
Comparison of 165 all cell sizes (i.e., cell widths in terms of the number of CPPs (contacted poly pitches)) produced by CSyn-fp with in-cell routing completion and in the ASAP 7nm cell library [12].

Cells		#FETs	#CPPs (i.e. size)		Impr.
Type	#Cell.	Smallest/Largest	ASAP [12]	CSyn-fp	
AO/AOI	42	8 / 20	9.67	9.38	**2.96**%
OA/OAI	34	8 / 20	9.24	9.12	**1.27**%
AND/NAND	22	4 / 12	9.41	9.27	**1.45**%
OR/NOR	22	4 / 12	8.43	8.43	-
XOR/XNOR	6	10 / 12	10.7	10.3	**3.12**%
BUF/INV	23	2 / 4	10.1	10.1	-
HA/FA	2	10 / 24	11.5	11.5	-
Latch	6	16 / 16	16.0	16.0	-
DFF	8	24 / 26	22.0	22.0	-
Total	165	-	-	-	-

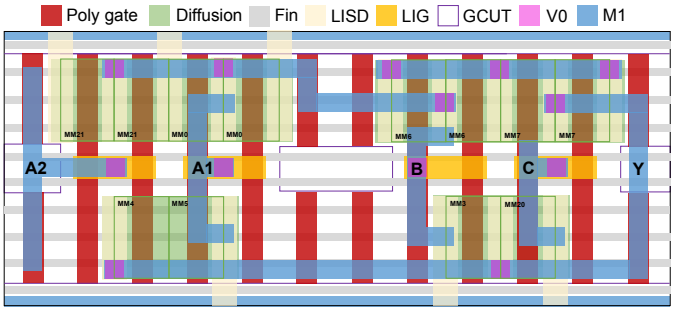
TABLE II
Comparison of top 10 best-synthesized cell sizes (i.e., cell widths in terms of the number of CPPs (contacted poly pitches) produced by CSyn-fp with in-cell routing completion and in ASAP 7nm cell library [12]. WL : total M1 metal length, #VIAs : total number of V0 vias.

Cells	ASAP [12]			CSyn-fp		
	#CPPs	WL	#VIAs	#CPPs	WL	#VIAs
AND5x2	20	3359	24	18	3405	23
AO211x2	16	2967	18	14	2496	17
AO22x1	9	1845	12	8	1581	13
AOI211x1	12	2301	13	11	1956	14
AOI221x1	14	2567	17	13	2598	21
AOI222xp33	10	1847	15	9	1707	16
AOI31xp67	13	2950	19	11	2202	17
OA31x2	15	2726	20	14	2690	19
OAI31xp67	13	2677	19	11	2302	17
XOR2x1	12	2429	16	11	2198	18
Avg.	13.4	2566.8	17.3	12.0	2313.5	17.5
Impr.				**10.4%**↓	**9.9%**↓	**1.2%**↑

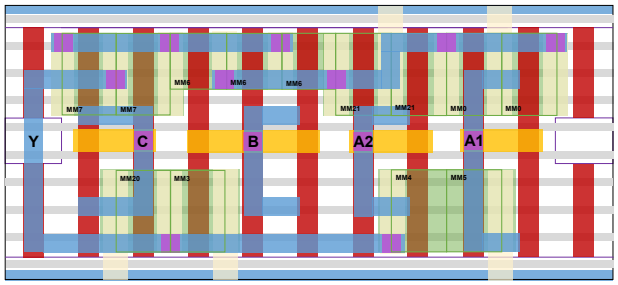
For example, Figs. 12(a) and 12(b) show the layouts of cell AOI211x1 in the ASAP 7nm library and produced by CSyn-fp, respectively. It shows that the layout in ASAP 7nm library has two fingers for every transistor of size 6, causing a diffusion break in the middle of the layout, resulting in a cell size of 12 CPPs. On the other hand, CSyn-fp controls the number of fingers for transistors individually to prevent the generation of diffusion break, resulting in a cell size of 11 CPPs.

• **Assessing synthesis efficiency**: Table III shows the running times of CSyn-fp when cell layouts are synthesized with the option of setting static folding or dynamic folding. For both options, CSyn-fp generates layouts for every netlist of combinational cells within 1 second. Furthermore, since the design space for dynamic folding properly covers the design space of static folding, there is no reason not to use the option of choosing dynamic folding for combinational cells to produce optimal layouts.

Table IV shows the running time of CSyn-fp with the option of no-partitioning or partitioning based on articulation points for netlists of six sequential cells in ASAP 7nm library. The comparison in run times indicates that CSyn-fp employing articulation point based netlist partitioning can generate layouts for large sequential cells within 1 second. Furthermore, the



(a) Layout of 12 CPPs in ASAP 7nm cell library.



(b) Layout of 11 CPPs produced by CSyn-fp.

Fig. 12. Layout comparison for cell AOI211x1 in [12].

TABLE III
RUNNING TIME OF CSyn-fp WITH THE OPTION OF STATIC OR DYNAMIC FOLDING FOR NETLISTS OF COMBINATIONAL CELLS IN ASAP 7NM LIBRARY.

Cells	#FETs	CSyn-fp w/ sta. fold.		CSyn-fp w/ dyn. fold.	
		#CPPs	Time (s)	#CPPs	Time (s)
AND4x2	10	16	0.239	15	0.413
AO22x1	10	9	0.017	8	0.025
AO322x2	16	13	0.751	12	0.902
AOI211x1	8	12	0.033	11	0.050
AOI221x1	10	14	0.178	13	0.252
AOI222xp33	12	10	0.064	9	0.067
O2A1O1Ixp5	8	9	0.011	8	0.046
OAI221xp5	10	9	0.015	8	0.029
XNOR2x1	10	12	0.091	11	0.870
XOR2x1	10	12	0.052	11	0.885

layout optimality in terms of #CPPs still holds for all netlists in Table IV.

V. CONCLUSION

In this paper, we proposed an integrated approach to the two problems of transistor folding and placement. Precisely, we proposed a *globally optimal* algorithm of search tree based design space exploration, devising a set of four effective speeding up techniques as well as dynamic programming based fast cost computation. In addition, our algorithm incorporated the minimum oxide diffusion jog constraint. Through experiment with all transistor netlists and design rules in the ASAP 7nm cell library, it was shown that our proposed method was able to synthesize fully routable cell layouts of minimal size *within 1 second for each netlist* in the ASAP 7nm library, outperforming the cell layout quality in the ASAP 7nm library, which otherwise, might take several hours or days to manually complete layouts of the quality level comparable to ours.

TABLE IV
RUNNING TIME OF CSyn-fp WITH THE OPTION OF NO-PARTITIONING OR PARTITIONING BY ARTICULATION POINTS FOR NETLISTS OF SEQUENTIAL CELLS IN ASAP 7NM LIBRARY.

Cells	#FETs	CSyn-fp w/ no-part.		CSyn-fp w/ part.	
		#CPPs	Time (s)	#CPPs	Time (s)
DHLx1	16	15	93.9	15	0.051
DHLx2	16	16	123.8	16	0.109
DHLx3	16	17	17.1	17	0.047
DFFHQNx1	24	20	7864.0	20	0.519
DFFHQNx2	24	21	10998.1	21	0.579
DFFHQNx3	24	22	7890.1	22	0.597

ACKNOWLEDGMENT

This work was supported in part by Samsung Electronics Company, Ltd. under Projects IO201216-08205-01 and FOUNDRY-202010DD003F, in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MEST) under Grant 2020R1A4A4079177 and Grant 2021R1A2C2008864, in part by the Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by Korea government (MSIT) under Grant 2021-0-00754, Software Systems for AI Semiconductor Design), and in part by the BK21 Four Program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2021. The EDA tool was supported by the IC Design Education.

REFERENCES

[1] S. Chakravarty, X. He, and S. Ravi, "Minimum area layout of series-parallel transistor networks is np-hard," *IEEE TCAD*, vol. 10, no. 7, pp. 943–949, 1991.

[2] A. Gupta and J. P. Hayes, "Optimal 2-d cell layout with integrated transistor folding," in *Proceedings of ICCAD*, 1998, pp. 128–135.

[3] A. Lu, H.-J. Lu, E.-J. Jang, Y.-P. Lin, C.-H. Hung, C.-C. Chuang, and R.-B. Lin, "Simultaneous transistor pairing and placement for cmos standard cells," in *Proceedings of DATE*. IEEE, 2015, pp. 1647–1652.

[4] J. Cortadella, "Area-optimal transistor folding for 1-d gridded cell design," *IEEE TCAD*, vol. 32, no. 11, pp. 1708–1721, 2013.

[5] K. Jo, S. Ahn, J. Do, T. Song, T. Kim, and K. Choi, "Design rule evaluation framework using automatic cell layout generator for design technology co-optimization," *IEEE TVLSI*, vol. 27, no. 8, pp. 1933–1946, 2019.

[6] T. Iizuka, M. Ikeda, and K. Asada, "Exact minimum-width multi-row transistor placement for dual and non-dual cmos cells," in *Proceedings of ISCAS*, 2006.

[7] D. Lee, D. Park, C.-T. Ho, I. Kang, H. Kim, S. Gao, B. Lin, and C.-K. Cheng, "Sp&r: Smt-based simultaneous place-&-route for standard cell synthesis of advanced nodes," *IEEE TCAD*, 2020.

[8] M. Guruswamy, R. L. Maziasz, D. Dulitz, S. Raman, V. Chiluvuri, A. Fernandez, and L. G. Jones, "Cellerity: A fully automatic layout synthesis system for standard cell libraries," in *Proceedings of DAC*, 1997, pp. 327–332.

[9] P. Van Cleeff, S. Hougardy, J. Silvanus, and T. Werner, "Bonncell: Automatic cell layout in the 7-nm era," *IEEE TCAD*, vol. 39, no. 10, pp. 2872–2885, 2019.

[10] J. Kim and S.-M. Kang, "An efficient transistor folding algorithm for row-based cmos layout design," in *Proceedings of DAC*, 1997, pp. 456–459.

[11] K. Han, A. B. Kahng, and H. Lee, "Scalable detailed placement legalization for complex sub-14nm constraints," in *Proceedings of ICCAD*. IEEE, 2015, pp. 867–873.

[12] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.