Row Planning and Placement for Hybrid-Row-Height Designs

Ching-Yao Huang
National Tsing Hua University
Hsinchu, Taiwan
rabit66119929@gapp.nthu.edu.tw

Wai-Kei Mak
National Tsing Hua University
Hsinchu, Taiwan
wkmak@cs.nthu.edu.tw

2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3503-9354-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASP-DAC58780.2024.10473801

***Abstract*—Traditionally, a standard cell library is composed of pre-designed cells all of which have identical height so that the cells can be placed in rows of uniform height on a chip. The desire to integrate more logic gates onto a single chip has led to a continuous reduction of row height with reduced number of routing tracks over the years. It has reached a point that not all cells can be designed with the minimum row height due to internal routability issue. Hybrid-row-height IC design with placement rows of different heights has emerged which offers a better sweet spot for performance and area optimization. [7] proposed the first row planning algorithm for hybrid-row-height design based on k-means clustering to determine the row configuration so that the cells in an initial placement can be moved to rows with matching height with as little cell displacement as possible. The biggest limitation of the k-means clustering method is that it only works for designs without any macros. Here we propose an effective and highly flexible dynamic programming approach to determine an optimized row configuration for designs with or without macros. The experimental results show that for designs without any macros, our approach resulted in 30.7% reduction in total cell displacement and 7.4% reduction in the final routed wirelength on average compared to the k-means clustering approach while satisfying the timing constraints. Additional experimental results show that our approach can comfortably handle designs with macros while satisfying the timing constraints.**

I. Introduction

To simultaneously optimize power, performance, and area, a design strategy is to mix the usage of standard cells of differing heights from different cell libraries where the height of a taller cell (e.g., a 12-track cell) is not necessarily an integer multiple of the height of a shorter cell (e.g., a 8-track cell) [1], [3], [12]. It assumes a layout is divided into regions such that each region can accommodate only cells of the same height but each region can use a row height different from its neighboring regions. In [1], [3], [12], horizontal or vertical spacing between neighboring regions is required to satisfy well-to-well spacing rule or to avoid P/G rail encroachment due to the usage of different cell libraries developed independently.

Lately a brand new hybrid-row-height design paradigm [6], [4], [2], [5] has emerged. For a hybrid-row-height design, the layout consists of placement rows of two or more distinct heights which do not have to be multiples of a single height. Moreover, by carefully crafting a cell library consisting of cells of distinct heights to avoid P/G rail encroachment, no extra spacing is required between placement rows of differing heights. TSMC's 3mn FinFlex design is an early example of hybrid row-height design [10]. The initial offering of hybrid-row-height design consists of alternate rows of two distinct row heights. But going beyond the alternate row configuration will result in an even larger solution space which will provide more flexibility for optimizing power, timing, and area.

This work is supported in part by the National Science and Technology Council under grant 111-2221-E-007-119-MY3.

The placement problem of hybrid-row-height chip designs is more challenging than that of uniform-row-height chip designs. There are only very limited previous works [2], [7], [5] on it. [2] and [5] introduced a new global placement algorithm and a new legalization algorithm, respectively, targeting hybrid-row-height design when the desired row configuration is provided. But [2] and [5] did not address how to determine what row configuration should be used for a design. On the other hand, [7] leveraged a conventional commercial placement tool for uniform row height design to address the row planning and placement problem of hybrid-row-height design with two distinct row heights. However, it relies on an implicit assumption that there are no macros in the design and the experiments in [7] were performed on designs without macros. In addition, [7] only considers the displacement of minority height cells but fails to take the displacement of majority height cells into account which greatly affects the solution quality.

Here we address the row planning and placement problem as in [7]. Given an initial placement without row type restrictions by a conventional placement tool, we want to determine an optimized row configuration and move cells to rows with matching height to minimize the total cell displacement. Our contributions are as follows. We propose an effective and highly flexible dynamic programming approach to determine the most desirable row configuration. Unlike [7], our method takes both minority height cells and majority height cells into consideration and can determine an optimized row configuration for any design with or without macros. Experimental results show that for designs without any macros, compared to [7] our approach resulted in 30.7% reduction in total cell displacement and 7.4% reduction in the final routed wirelength on average while satisfying the timing constraints. Additional experiments show that our approach can comfortably handle designs with macros while satisfying the timing constraints.

The rest of the paper is organized as follows. Section II gives the problem formulation and the details of the placement method. The experimental results are reported in section III. We conclude this article in section IV.

II. Hybrid-Row-Height Design Placement

First, we define the problem addressed in this paper and provide an overview of our approach.

Definition (*Row Planning and Placement for Hybrid-Row-Height Design*). *Given a gate-level netlist of a design with cells of two distinct heights and a set of macros, timing constraints, Liberty and technology models, and the desired layout size. We want to determine a placement solution such that every cell is placed on a site of a row that has the same height as the cell to minimize the wirelength and satisfy the timing constraints.*

Suppose there are two distinct cell heights h_m and h_M where the total width of cells with height h_m is less than the total width of cells with height h_M. We refer to the cells with height h_m and height h_M as the *minority cells* and the *majority cells*, respectively. Similarly, the rows which will be assigned height h_m and height h_M are referred to as the *minority rows* and the *majority rows*, respectively.

Given an input design, a cell library with cell heights h_m and h_M, and timing constraints, logic synthesis is performed to obtain a hybrid-row-height design netlist. Algorithm 1 gives the overall flow of our placement approach. In the initial placement stage (line 1), we modify the standard cell library exchange format (LEF) file to make all standard cells and rows have the same height. We feed this modified LEF file and netlist to a commercial placement tool to obtain an initial placement of the design under timing constraints. In the row configuration determination stage (line 2), some rows are selected to be the minority rows. Then, the heights of minority rows and majority rows are modified to be the height of minority cells and the height of majority cells, respectively (line 3). Finally, we legalize the placement of the minority cells and majority cells to rows with matching height using a variant of Abacus[11] as in [7] (line 4). We elaborate on the initial placement stage in subsection II-A and the row configuration stage in subsection II-B, and how we perform legalization in subsection II-C.

Algorithm 1: Hybrid-Row-Height Design Placement

```
1 Initial placement without row type restrictions
2 Determine row configuration
3 Configure row heights for minority rows and majority rows
4 Legalize cells with row type restrictions minimizing cell displacement
```

A. *Initial Placement*

In this stage, we first estimate the number of rows in the layout region and the average row height. The LEF files are modified such that all rows and cells share the same height to be defined below. Then, we leverage a commercial placement tool to place cells with timing awareness and obtain an initial placement.

Given a netlist, the layout width W and desired aspect ratio R, we get the approximate layout height $H = RW$. The expected number of rows n_{h_m} and n_{h_M} for each height can be determined by

$$n_{h_m} = \left\lfloor \frac{T_{h_m} H}{T_{h_m} h_m + T_{h_M} h_M} \right\rfloor, \quad n_{h_M} = \left\lceil \frac{T_{h_M} H}{T_{h_m} h_m + T_{h_M} h_M} \right\rceil \tag{1a}$$

$$T_{h_m} = \frac{W_{h_m}}{W_{h_m} + W_{h_M}}, \; T_{h_M} = \frac{W_{h_M}}{W_{h_m} + W_{h_M}} \tag{1b}$$

where W_{h_m} and W_{h_M} are the total widths of cells with height h_m and h_M, respectively. The average height h^* can be computed as

$$h^* = \frac{n_{h_m} h_m + n_{h_M} h_M}{n_{h_m} + n_{h_M}} \tag{2}$$

We initialize totally $n_{h_m} + n_{h_M}$ rows[1] in the layout with row height h^*. The modified LEF file is generated by modifying the heights of all cells to h^* while every cell maintains its area and timing characteristics. Then, we use a commercial placement tool to perform placement and post-placement optimization to produce an initial placement.

B. *Row Configuration Determination*

1) Previous Approach: K-means clustering based on the y-coordinates of the cells is applied in [7] to determine the distribution of the minority rows (line 2 in algorithm 1). Initially, it partitions minority cells into k clusters by k-means clustering where k is the estimated number of minority rows required. For each cluster, if the total width of the minority cells in the cluster is greater than a pre-defined upper bound, the corresponding cluster center will be replaced by two new cluster centers. Then k-means clustering is repeated until there is no change in the number of clusters. Afterwards, for each cluster, the row closest to the center of the cluster (i.e., the mean y-coordinate of the minority cells in the cluster) is selected to be a minority row. All remaining rows will become the majority rows.

But there are a number of drawbacks with the k-means clustering-based method. Firstly, it cannot cope with macros. It assumes the same upper bound on the total width of cells for all minority cell clusters but the available width of each row for cell placement varies greatly in the presence of macros in a design. To select the row closest to the center of a minority cell cluster as a minority row, the selected row may not be able to accommodate all minority cells in that cluster when the row is partially blocked by macros. More importantly, there is no guarantee that the total available width of all rows selected as minority rows will be sufficient to accommodate all minority cells. In other words, it may result in a row configuration with no feasible placement solution. Even if there is no macro, the k-means clustering-based method totally ignores the effects of the majority cells. It does not consider whether enough space is left for placing the majority cells when the number of minority cell clusters is increased repeatedly. And taking the mean of a minority cell cluster to select a minority row entirely neglects the majority cell displacement which is detrimental to the solution quality.

2) DP Approach: Given an initial placement without row type restriction, we propose a dynamic programming approach to determine a row configuration that guarantees all minority cells can be placed in the minority rows and all majority cells can be placed in the majority rows whether there are macros or not, and it targets to optimize the total displacement of both majority cells and minority cells. This is done by tentatively assigning each cell to a row that minimizes the total vertical displacement of all cells from their initial locations to their assigned rows subject to the following constraints. (1) All cells assigned to the same row must have the same height[2]. (2) The total width of cells assigned to row r_i must not exceed its available capacity $w(r_i)$. (3) No minority cell is moved by more than β rows[3] from its original row. However, to try assigning each cell to every single row will result in an unacceptably long runtime. Hence, we make a number of observations to come up with a simplified dynamic program formulation below.

Observe that most rows will be designated as majority rows, so there are many good ways to move the majority cells to the final majority rows. Hence, we will not spend extra time to assign the majority cells to specific rows during row planning. Instead, for every majority cell originally located within a row selected as a minority row, it must be moved away and a

[1]Estimating the total number of rows in this way always work regardless of the presence of macros. Besides, we are not fixing the number of minority rows to n_{h_m}, the final number of minority rows will be determined by Algorithm 2.

[2]The cells are changed back to their original heights and widths.

[3]β is set to 4 in out experiments.

TABLE I: Notation

R	list of rows $(r_1, r_2, \ldots, r_{\|R\|})$ in increasing y-coordinates
$w(r)$	total available width of row r not blocked by macros
$y(r)$	y-coordinate of center of row r
$nMaj(r)$	number of majority cells originally in row r
$maxRW$	maximum available row width $(= max_r\{w(r)\})$
$MinC$	list of minority cells $(m_1, m_2, \ldots, m_{\|MinC\|})$ sorted in non-decreasing y-coordinates
w_{m_i}	width of minority cell m_i
y_{m_i}	y-coordinate of center of minority cell m_i
$minCW$	minimum minority cell width $(= min_i\{w_{m_i}\})$
β	vertical displacement bound of a cell in rows

fixed penalty cost will be incurred. Next, if for two minority cells m_i and m_j, $y_{m_i} \leq y_{m_j}$ but $row(m_i) > row(m_j)$ where $row(c)$ is the index of the row to which cell c is assigned to, then we can swap the row assignment of m_i and m_j which will reduce or maintain the same total displacement to their assigned rows except when swapping them will violate the available capacity of either row $row(m_i)$ or $row(m_j)$. So, we assume that $row(m_i) \leq row(m_j)$ if $y_{m_i} \leq y_{m_j}$ in our formulation which will dramatically reduce the space and time complexities. In particular, we first sort the minority cells in non-decreasing order of the y-coordinates of their centers[4] (i.e., $y_{m_1} \leq \cdots \leq y_{m_{|minC|}}$) and obtain a sorted list $MinC$ of minority cells, then we require that only consecutive minority cells in $MinC$ can be assigned to the same minority row.

We will use the notation in Table I to describe our algorithm. We define some key terms and show how to compute them next.

- $Cost[k][j]$ denotes the minimum cost incurred when k minority rows are optimally selected to accommodate the first j minority cells (i.e., $m_1, m_2, \ldots, m_j$) which includes the cost of moving the first j minority cells to the selected minority rows and moving the majority cells originally in the selected minority rows away.
- $Row[k][j]$ denotes the row index of the k-th minority row when k minority rows are optimally selected to accommodate the first j minority cells to minimize $Cost[k][j]$.
- $FirstmCell[k][j]$ denotes the first minority cell in the k-th minority row when k minority rows are optimally selected to accommodate the first j minority cells to minimize $Cost[k][j]$.
- $Row\_Cost(t, i, j)$ denotes the cost of moving minority cells $m_i, m_{i+1}, \ldots, m_j$ to row r_t and moving the majority cells originally in row r_t away.
- $Ideal\_Row[i][j]$ denotes the index t of the ideal row r_t for minority cells m_i, m_{i+1}, ... , m_j to be assigned to in order to minimize $Row\_Cost(t, i, j)$.
- $Choose\_Row(last\_t, i, j)$ denotes the index t of the best row r_t for minority cells m_i, m_{i+1}, ... , m_j to be assigned to in order to minimize $Row\_Cost(t, i, j)$ under the condition that it must be larger than $last\_t$.

We note that all of the above are subject to the available width of individual rows which can be different for different rows due to blockage by macros. For simplicity, estimating the cost by the amount of vertical cell displacement is good enough in practice which is supported by the experimental results. To determine a subset of rows to be turned into minority rows, we have to find the value k^* of k that minimizes $Cost[k][|minC|]$ and then trace back the corresponding k^* optimal minority rows from $Row[k^*][|minC|]$ and $FirstmCell[k^*][|minC|]$. We will compute $Cost[\,][\,]$, $Row[\,][\,]$, and $FirstmCell[\,][\,]$ by dynamic programming.

$Row\_Cost(t, i, j)$, the cost of moving minority cells m_i, m_{i+1}, ... , m_j to row r_t, and moving the majority cells originally in row r_t away, is given by (3). α is the penalty cost for moving a majority cell in row r_t to a nearby majority row.[5] We set $Row\_Cost(t, i, j)$ to ∞ if row r_t is more than β rows away from the original rows of m_i or m_j, or if the available width of row r_t is insufficient to accommodate minority cells m_i to m_j, or if it will leave too much wasted space in row r_t to accommodate only minority cells m_i to m_j. By controlling the minimum targeted utilization λ of a minority row[6], we guarantee that there will be enough space left for placing the majority cells.

$Ideal\_Row[i][j]$, the index t of the ideal row r_t for minority cells m_i, m_{i+1}, ... , m_j to be assigned to in order to minimize $Row\_Cost(t, i, j)$, is given by (4).[7] We set $Ideal\_Row[i][j]$ to 0 if no row is suitable to accommodate minority cells m_i to m_j. (Indexes for valid rows are 1 to $|R|$.)

$Cost[k][j]$, the minimum cost incurred when k minority rows are optimally selected to accommodate the first j minority cells, is expressed recursively in (6). At first glance, one may think that $Cost[k][j]$ should be equal to the minimum of $Cost[k-1][i-1]+$ $Row\_Cost(Ideal\_Row[i][j], i, j)$ over all feasible choices of i, but it is incorrect since it is possible that $Ideal\_Row[i][j]$ is less than or equal to the selected $(k-1)$-th minority row index that yielded $Cost[k-1][i-1]$. To ensure the index of the k-th minority row is larger than that of the $(k-1)$-th minority row, we make use of $Choose\_Row(last\_t, i, j)$. $Choose\_Row(last\_t, i, j)$, the index t of the best row for minority cells m_i, m_{i+1}, ... , m_j to be assigned to in order to minimize $Row\_Cost(t, i, j)$ under the condition that it must be larger than $last\_t$, is given by (5). If $Ideal\_Row[i][j]$ is larger than $last\_t$, then $Choose\_Row(last\_t, i, j)$ simply returns $Ideal\_Row[i][j]$. Otherwise, we search for a row close to the original rows of m_i and m_j with enough available width to accommodate minority cells m_i to m_j with the minimum cost whose index is greater than $last\_t$.

When we determine $Cost[k][j]$, the first cell $FirstmCell[k][j]$ in the k-th minority row when k minority rows are optimally selected to accommodate the first j minority cells can be determined simultaneously. The mathematical expression for $FirstmCell[k][j]$ is given by (7). We set $FirstmCell[k][j]$ to 0 if there is no feasible way to select k rows to accommodate the first j minority cells. Finally, $Row[k][j]$, the index of the k-th minority row if k minority rows are optimally selected to accommodate the first j minority cells, is given by (8). To compute $Row[k][j]$, we choose the optimal k-th minority row to accommodate minority cells $m_{FirstmCell[k][j]}$ to m_j subject to the condition that its index must be larger than the $(k-1)$-th minority row selected for accommodating the preceding $FirstmCell[k][j]-1$ minority cells. We set $Row[k][j]$ to 0 if there is no feasible way to select k rows to accommodate the first j minority cells.

Hence, we have the final algorithm for selecting a set of rows as minority rows as shown in Algorithm 2. In lines 1 and

[4]If the y-coordinates of the centers of two or more minority cells are the same, we order them in non-decreasing order of the x-coordinates of their centers.

[5]α is set to $2 \times \frac{1-p^8}{1-p^2} \times \frac{\#minority}{\#majority}$ times of the minority row height in our experiments where p is the probability of a row is selected to be a minority row.

[6]λ is set to 0.25 in our experiments.

[7]We note that if one ignores the displacement of the majority cells and the presence of macros, then the ideal row r_t must satisfy $y_{m_i} \leq y(r_t) \leq y_{m_j}$. However, it is not necessarily true when the displacement of majority cells are considered and there are macros.

$$Row\_Cost(t,i,j) = \begin{cases} \sum_{p=i}^{j} |y(r_t) - y_{m_p}| + nMaj(r_t) * \alpha & \text{if } y_{m_j} - \beta h^* \le y(r_t) \le y_{m_i} + \beta h^* \wedge \sum_{p=i}^{j} w_{m_p} \le w(r_t) \wedge \frac{\sum_{p=i}^{j} w_{m_p}}{w(r_t)} \ge \lambda \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

$$Ideal\_Row[i][j] = \begin{cases} \operatorname*{argmin}_{t} Row\_Cost(t,i,j) & \text{if } \sum_{p=i}^{j} w_{m_p} \le maxRW \wedge \exists t : Row\_Cost(t,i,j) \ne \infty \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$Choose\_Row(last\_t,i,j) = \begin{cases} Ideal\_Row[i][j] & \text{if } \sum_{p=i}^{j} w_{m_p} \le maxRW \wedge Ideal\_Row[i][j] > last\_t \\ \operatorname*{argmin}_{t:t>last\_t} Row\_Cost(t,i,j) & \text{if } \sum_{p=i}^{j} w_{m_p} \le maxRW \wedge Ideal\_Row[i][j] \le last\_t \wedge \\ & \quad \exists t : (t > last\_t \wedge Row\_Cost(t,i,j) \ne \infty) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$Cost[k][j] = \begin{cases} \min_{\sum_{p=i}^{j} w_{m_p} \le maxRW} \{Cost[k-1][i-1] + Row\_Cost(Choose\_Row(Row[k-1][i-1],i,j),i,j)\} & \text{if } \sum_{p=1}^{j} w_{m_p} \le k * maxRW \\ \infty & \text{otherwise} \end{cases} \tag{6}$$

$$FirstmCell[k][j] = \begin{cases} \operatorname*{argmin}_{i: \sum_{p=i}^{j} w_{m_p} \le maxRW} \{Cost[k-1][i-1] + Row\_Cost(Choose\_Row(Row[k-1][i-1],i,j),i,j)\} & \text{if } \sum_{p=1}^{j} w_{m_p} \le k * maxRW \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

$$Row[k][j] = \begin{cases} Choose\_Row(Row[k-1][FirstmCell[k][j]-1], FirstmCell[k][j], j) & \text{if } FirstmCell[k][j] \ne 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

2, we initialize $maxRW$ (maximum available width of a row among all rows) and set $maxK$ (maximum number of minority rows) as 1.125 times the estimated number of minority rows in sec. II-A. We sort all minority cells in non-decreasing order of y-coordinates in line 3. To speed up the computation, we pre-compute $Ideal\_Row[][]$ in line 4. In lines 5 and 6, we initialize $Cost[][]$, $FirstmCell[][]$, and $Row[][]$. The base case when one row has to be selected to accommodate the first j minority cells corresponds to lines 7 to 14. Since there is no previous selected minority row, the best row to accommodate minority cells m_1 to m_j must be $Ideal\_Row[1][j]$ (line 12). $Cost[1][j]$ and $FirstmCell[1][j]$ can be determined directly (lines 10 to 11).

In lines 15 to 31, we iterate k from 2 to $maxK$ to compute $Cost[k][j]$, $FirstmCell[k][j]$, and $Row[k][j]$. To compute $Cost[k][j]$, $FirstmCell[k][j]$, and $Row[k][j]$ for a particular pair of values of k and j in lines 21 to 29, we have to choose a suitable row to accommodate minority cells m_i to m_j considering the last row chosen for the minority cells preceding m_i and the resultant total cost for every feasible i (lines 22 and 23). If the total cost is less than the current value of $Cost[k][j]$, then we update $Cost[k][j]$, $FirstmCell[k][j]$, and $Row[k][j]$ (lines 24 to 27). In the end, we use $FirstmCell[][]$ and $Row[][]$ to trace back a set of minority rows that minimizes $Cost[k][|MinC|]$ and can accommodate all minority cells (lines 32 to 38).

3) Implementation Details and Run Time Analysis: We analyze the run time complexity of algorithm 2. First, we note that the number of minority cells a row can accommodate is bounded by $N = maxRW/minCW$ where $maxRW$ is maximum available width of a row and $minCW$ is the minimum minority cell width. To speed up the computation of $Row\_Cost(t,i,j)$, for each row r_t, we pre-compute the number of minority cells whose y-coordinates are smaller and greater than the y-coordinate of r_t, respectively. We also pre-compute the summation of the y-coordinates of any q consecutive minority cells for $q = 1, 2, \ldots, N$ in $O(|MinC|N)$ time. Consequently, each call to procedure $Row\_Cost(t,i,j)$ will only take $O(1)$ time. For two minority cells m_i and m_j, if $j - i \ge N$, then $Ideal\_Row[i][j]$ must be 0. Thus we only need to compute $O(|MinC|N)$ entries in $Ideal\_Row[][]$. The time to compute an entry $Ideal\_Row[i][j]$ is $O(|R|)$ since there are at most $|R|$ rows which can accommodate minority cells m_i to m_j and the time for each call to procedure Row_Cost is $O(1)$. Hence, the time required to construct $Ideal\_Row[][]$ is $O(|MinC|N|R|)$. The time to access a value in $Ideal\_Row[][]$ subsequently is $O(1)$.

For a call to procedure $Choose\_Row(last\_t,i,j)$, it takes

Algorithm 2: Minority Row Selection

```
   Output: Set $S$ of selected minority rows
1  $maxRW \leftarrow max(\{w(r_i)|r_i \in R\})$
2  $maxK \leftarrow \lceil 1.125 \times n_{h_m} \rceil$
3  Sort minority cells in non-decreasing order of $y$-coordinates
4  Compute $Ideal\_Row[][]$ based on equation (4)
5  Initialize $Cost[][]$ with all entries set to $\infty$
6  Initialize $FirstmCell[][]$ and $Row[][]$ with all entries set to 0
7  $sum \leftarrow 0$
8  $j \leftarrow 1$
9  while $sum + w_{m_j} \le maxRW$ do
10     $Cost[1][j] \leftarrow Row\_Cost(Ideal\_Row[1][j], 1, j)$
11     $FirstmCell[1][j] \leftarrow 1$
12     $Row[1][j] \leftarrow Ideal\_Row[1][j]$
13     $sum \leftarrow sum + w_{m_j}$
14     $j \leftarrow j + 1$
15 for $k = 2$ to $maxK$ do
16     $sum1 \leftarrow 0$
17     $j \leftarrow 1$
18     while $sum1 + w_{m_j} \le k \times maxRW$ do
19         $sum2 \leftarrow 0$
20         $i \leftarrow j$
21         while $sum2 + w_{m_i} \le maxRW$ and $i \ge k$ do
22             $t \leftarrow Choose\_Row(Row[k-1][i-1], i, j)$
23             $c \leftarrow Cost[k-1][i-1] + Row\_Cost(t, i, j)$
24             if $c < Cost[k][j]$ then
25                 $Cost[k][j] \leftarrow c$
26                 $FirstmCell[k][j] \leftarrow i$
27                 $Row[k][j] \leftarrow t$
28             $sum2 \leftarrow sum2 + w_{m_i}$
29             $i \leftarrow i - 1$
30         $sum1 \leftarrow sum1 + w_{m_j}$
31         $j \leftarrow j + 1$
32 $K \leftarrow \operatorname*{argmin}_{k:k \le maxK} Cost[k][|MinC|]$
33 $S \leftarrow \{\}$
34 $Lastmcell \leftarrow |MinC|$
35 while $K > 0$ do
36     $S = S \cup \{Row[K][Lastmcell]\}$
37     $Lastmcell \leftarrow FirstmCell[K][Lastmcell] - 1$
38     $K \leftarrow K - 1$
39 return $S$
```

$O(1)$ time if $Ideal\_Row[i][j]$ is greater than $last\_t$. Otherwise, we need to call procedure Row_Cost at most $2\beta + 1$ times (when m_i and m_j are in the same row) to search for the best row with index greater than $last\_t$ and between $y_{m_j} - \beta h^*$ and $y_{m_i} + \beta h^*$. So, the worst case complexity for a call to $Choose\_Row(last\_t, i, j)$ is still $O(1)$. In each iteration of the innermost loop in lines 22 to 29 of algorithm 2, we have to call $Choose\_Row$ once and Row_Cost once. Thus the time complexity for one iteration of the innermost loop is $O(1)$. For every particular pair of values for k and j, the innermost loop will execute at most N times. For every particular value for k, there are no more than $|MinC|$ feasible values for j. The outermost while loop in line 15 iterates k from 2 to $maxK$. So, the total time spent by the outermost while loop is $O(maxK|MinC|N)$ which is also the overall complexity of the algorithm. Since $maxK < |R|$ and $N|R| \simeq |MinC|$, the worst case time complexity of algorithm 2 is $O(|MinC|^2)$. Note that to perform row planning for very large designs, one may choose to divide the layout region horizontally into a few subregions and perform row planning for each separately to reduce the runtime.

C. *Legalization*

After determining the row configuration in section II-B, each row is changed to its assigned height according to the row configuration. Then, we legalize the cell placement with matching row height by modifying the legalization approach in [11] which minimizes the required cell displacement. In [11], the legalization of a cell is done by comparing the results of inserting the cell in each row, then the cell is inserted into the row which results in the minimum cost. Here we limit the cell can only be inserted into some row which has the same height as the cell.

III. Experimental results

We implemented our method in C++ and it was executed on a 64-bit Ubuntu server with AMD ThreadRipper 3970X CPU and 256GB memory. We modified the 15nm Nangate open cell library [8] to make our cell library contain cells with two distinct heights, 8T and 12T, where the height of all cells in the 15nm Nangate open cell library is 12T. The cells with lower driving strength in the 15nm Nangate open cell library are selected to be the short cells and the remaining cells are tall cells. The width and height of each tall cell remain the same as in the 15nm Nangate open cell library. The width of each short cell is not changed but the height is modified to 8T. Since the area of short cells became smaller, we also modified the capacitance and power of the short cells in the liberty file. Figure 1 shows the trade-off between cell delay and cell area of the inverters and buffers in our cell library[8].

Six designs from [9] were chosen to be the benchmarks as shown in Table II where "des", "aes", "fpu", and "jpeg" were used in [7]. Note that we did not use the well-known ISPD 2015 benchmark set since no timing information is available. $\#Min.\ Cells$, $\#Maj.\ Cells$, $\#Nets$, $Clock$, and W stand for number of minority cells, number of majority cells, number of nets, clock period, and layout width, respectively. It is possible that the number of minority cells is more than the number of majority cells but the total width of minority cells must be less than the total width of majority cells. We used Synopsys Design Compiler R-2020.09 to perform logic synthesis. In the initial placement stage, we used Cadence Innovus Implementation System v21.13-s100_1 to perform placement. After selecting some rows to be minority rows, we updated all row heights. Then, we legalized the cell placement with matching row height. Finally, the legalized placement is routed using Cadence Innovus Implementation System v21.13-s100_1.

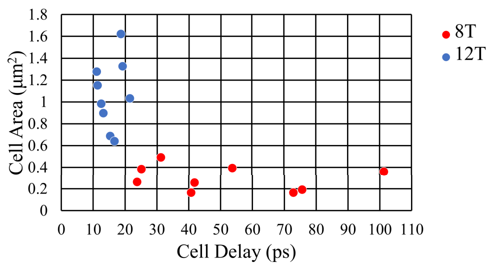


Fig. 1: The trade-off between cell area and cell delay of different buffers and inverters in our cell library.

TABLE II: Benchmark information.

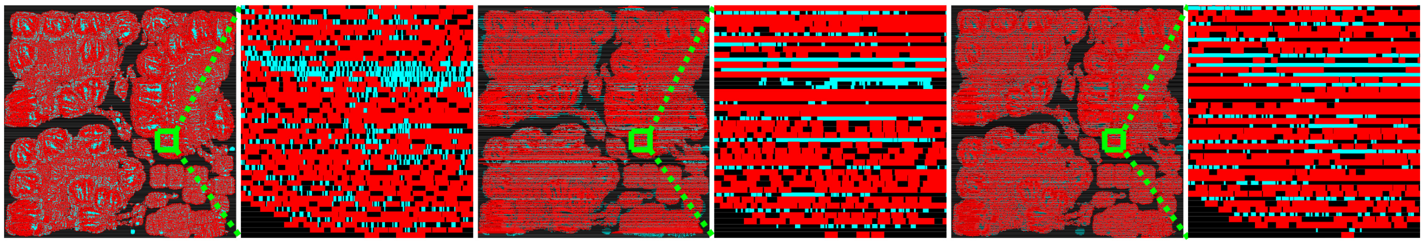
Design	#Min. Cells	#Maj. Cells	#Nets	Clock(ps)	W(μm)
des	127	1633	2000	250	27.328
aes	960	5284	6305	300	59.968
gng	3562	931	4747	500	54.592
fpu	22229	6931	31074	700	145.536
vga	48734	17052	65884	1200	225.152
jpeg	150103	62963	238408	1000	413.504

We compared with [7] which is the only previous work that addressed the row planning problem of hybrid-row-height design.[9] We implemented the k-means clustering algorithm of [7] in C++ and it was executed on the same machine with the assumption that no extra spacing is required between rows of differing heights as in [10]. Our implementation is approximately ten times faster than the implementation in [7]. We did not compare our placement result with Innovus native placement solution since Innovus can only perform placement for designs with single cell height or cell heights as integer multiples of a single type of row height. The results are listed in Table III and Table IV. The $Disp.$, $Min.\ Disp.$, and $Maj.\ Disp.$ columns in Table III give the total displacement of the cells, the total displacement of the minority cells, and the total displacement of the majority cells in the final placement compared to the initial placement, respectively. The WL column in Table IV gives the final routed wirelength after detailed routing.

It can be seen that our DP is able to produce a superior row configuration than k-means clustering in each case. Our row planning resulted in 30.7% less total displacement than k-means clustering on average, and the final routed wirelength is reduced by 7.4% on average. Since our DP-based method takes the displacement of majority cells into account while the k-means clustering algorithm does not, it resulted in 30.1% less total displacement of majority cells on average compared to k-means clustering. Our DP resulted in all designs satisfying the timing constraints. The power is also reduced by 6.7% on average. Fig. 2(b) and 2(c) show the final legalized placements of jpeg through the two different methods. We can see that the final legalized placements through our DP bear a higher resemblance to the initial placements in Fig. 2(a). In our experiment, we divided the benchmark "jpeg" horizontally into three subregions and performed row planning for each separately with DP.

[8]We extract the cell delay from the liberty model under the condition that the input transition time is 30 ps and the total output load capacitance is 100 fF.

[9]We did not compare with [5] since it solved a different problem, namely, placement legalization with cell version change to adapt to the row configuration given by the user.



(a) Initial placement of jpeg and the close-up view of the section enclosed by the green rectangle.

(b) Final placement of jpeg using k-means clustering and the close-up view of the section enclosed by the green rectangle.

(c) Final placement of jpeg using our DP approach and the close-up view of the section enclosed by the green rectangle.

Fig. 2: The initial placement and the final legalized placements of jpeg through the two different methods. (The majority cells are shown in red.)

TABLE III: Comparing the post-legalization total cell displacement, displacement of minority cells, displacement of majority cells, and runtime of two different row planning methods.

	k-means clustering				Ours (DP)			
Design	Disp. (μm)	Min. Disp. (μm)	Maj. Disp. (μm)	Time (s)	Disp. (μm)	Min. Disp. (μm)	Maj. Disp. (μm)	Time (s)
des	1073.33	546.65	526.69	0.01	977.66	171.23	806.43	<0.01
aes	5062.12	2143.58	2918.53	0.21	3711.83	1301.52	2410.31	0.02
gng	9878.3	4056.18	5822.12	0.91	5881.4	4322.71	1558.69	0.27
fpu	41295.82	25435.19	15860.63	3.42	28580.52	23277.8	5302.72	6.47
vga	45570.44	31437.23	14133.21	5.0	37302.86	29697.84	7605.02	31.49
jpeg	424954.46	384516.41	40438.05	30.84	172450.06	144346.71	28103.36	55.1
Norm.	1	1	1		0.693	0.704	0.699	

TABLE IV: Comparing the routed wirelength, power, worst negative slack(WNS), and total negative slack(TNS) reported by Innovus through two different row planning methods.

	k-means clustering				Ours (DP)			
Design	WL (μm)	Power (mW)	WNS (ns)	TNS (ns)	WL (μm)	Power (mW)	WNS (ns)	TNS (ns)
des	9259	15.44	0.009	0.0	8872	14.068	0.044	0.0
aes	42168	46.11	0.026	0.0	40067	42.716	0.057	0.0
gng	43466	17.009	0.036	0.0	39490	15.508	0.071	0.0
fpu	248451	45.951	0.014	0.0	228267	43.159	0.052	0.0
vga	487388	49.406	0.087	0.0	476892	48.322	0.108	0.0
jpeg	1752926	199.084	0.097	0.0	1482693	185.328	0.234	0.0
Norm.	1	1			0.926	0.933		

TABLE V: Final results through our DP row planning method for modified benchmarks with macros.

	Ours (DP)								
Design	#macro	Disp. (μm)	Min. Disp. (μm)	Maj. Disp. (μm)	Time (s)	WL (μm)	Power (mW)	WNS (ns)	TNS (ns)
des_{m}	2	1054.24	198.84	855.40	<0.01	8950	13.924	0.042	0.0
aes_{m}	3	4477.57	1082.41	3395.16	0.02	42533	44.354	0.051	0.0
gng_{m}	3	8153.68	6040.02	2113.65	0.19	43551	16.208	0.054	0.0
fpu_{m}	4	30905.73	24007.42	6898.30	5.77	240587	42.928	0.051	0.0
vga_{m}	5	37635.83	29763.50	7872.32	29.84	488461	48.478	0.133	0.0
jpeg_{m}	6	184631.53	155360.09	29271.44	52.05	1535862	186.795	0.118	0.0

In the second experiment, we modified each benchmark by introducing some blockage areas in the layout region to mimic pre-placed macros. We only report the results through our DP approach for the modified benchmarks in Table V since the k-means clustering method is not applicable. The final routed wirelengths are slightly increased compared to Table IV which is expected. Again all timing constraints are satisfied.

IV. Conclusion

We considered the row planning and placement problem for hybrid-row-height design. We proposed an effective and flexible dynamic programming method for row planning. Our DP approach produced superior hybrid row configurations compared to [7], and resulted in 30.7% and 7.4% reduction in the total cell displacement and final routed wirelength, respectively. Moreover, our DP method can handle designs with macros while satisfying the timing constraints.

References

[1] J. Chen, Z. Huang, Y. Huang, W. Zhu, J. Yu, and Y.-W. Chang. An efficient epist algorithm for global placement with non-integer multiple-height cells. In *2020 ACM/IEEE DAC*, pages 1–6, 2020.

[2] K.-Y. Chen, H.-C. Hsu, W.-K. Mak, and T.-C. Wang. Hybridgp: Global placement for hybrid-row-height designs. In *2022 ASP-DAC*, pages 294–299, 2022.

[3] S. A. Dobre, A. B. Kahng, and J. Li. Design implementation with noninteger multiple-height cells for improved design quality in advanced nodes. *IEEE TCAD*, 37(4):855–868, 2018.

[4] A. B. Kahng. Advancing placement. https://ispd.cc/ispd2022/slides/2021/protected/3_1_Kahng.pdf, 2021.

[5] S.-H. Liang, T.-L. Hsiung, W.-K. Mak, and T.-C. Wang. Hybrid-row-height design placement legalization considering cell variants. In *2023 GLSVLSI*, GLSVLSI '23, page 363–367, 2023.

[6] Y. Lin, C. Wang, and Y. Hou. Variant cell height integrated circuit design, U.S. Patent 10878157, Dec. 2020.

[7] Z.-Y. Lin and Y.-W. Chang. A row-based algorithm for non-integer multiple-cell-height placement. In *2021 IEEE/ACM ICCAD*, pages 1–6, 2021.

[8] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen. Open cell library in 15nm freepdk technology. In *2015 ISPD*, page 171–178, 2015.

[9] Online. Opencores. https://opencores.org/.

[10] S.-Y. Wu, C. Chan, et al. A 3nm cmos finflex™ platform technology with enhanced power efficiency and performance for mobile soc and high performance computing applications. In *2022 International Electron Devices Meeting (IEDM)*, pages 27.5.1–27.5.4, 2022.

[11] P. Spindler, U. Schlichtmann, and F. M. Johannes. Abacus: Fast legalization of standard cell circuits with minimal movement. In *2008 ISPD*, page 47–53, 2008.

[12] Y.-C. Zhao, Y.-C. Lin, T.-C. Wang, T.-H. Wang, Y.-R. Wu, H.-C. Lin, and S.-Y. Kao. A mixed-height standard cell placement flow for digital circuit block. In *2019 DATE*, pages 328–331, 2019.