Design Implementation With Noninteger Multiple-Height Cells for Improved Design Quality in Advanced Nodes

Sorin Adrian Dobre, Andrew B. Kahng, and Jiajia Li

***Abstract*—Standard-cell libraries can be developed with different cell heights (e.g., in FinFET technology, corresponding to different numbers of fins). Larger cell heights provide higher drive strengths, but at the cost of larger area and power consumption as well as pin capacitance. Cells with smaller heights are relatively smaller in area, but have weaker drive strengths and are more likely to suffer from routing congestion and pin accessibility issues. Existing design methodologies and tool flows are able to mix cells with different heights at the block level (i.e., each block contains cells with heights being integer multiples of a particular "single row" cell height). To our knowledge, no design methodology in the literature or in production mixes cells with different, noninteger multiple heights in a fine-grained manner. In this paper, we propose a novel physical design optimization flow to implement design blocks with mixed cell heights in a fine-grained manner. Our optimization resolves the "chicken-and-egg" loop between floorplan site definition and the optimized choices of cell heights after placement with full comprehension of the constraints and costs of mixing cells of different heights (e.g., the "breaker cell" area overheads of row alignment between sub-blocks of 8 T and 12 T cell rows), our optimization achieves up to over 30% area and power reductions versus 12 T-only implementation while maintaining the same performance, and up to over 10% performance improvement along with power and area reductions versus 8 T-only implementation.**

***Index Terms*—Digital integrated circuits, mixed cell height, physical design, placement algorithms.**

I. Introduction

STANDARD cell-based implementation has been widely used for very large scale integration designs due to its relatively accurate abstraction and semi-regular layout. Cells are designed with different heights (e.g., different numbers of fins in FinFET node). Larger cell heights have higher drive strengths at the cost of larger area and power consumption as well as pin capacitance. Smaller cell heights result in relatively smaller area, but have weaker drive strengths and are more likely to suffer from routing congestion and pin accessibility issues. Although a cell with smaller height can be designed with large width to gain drive strength, the additional poly capacitance and metal capacitance can lead to area and power overheads as compared to a cell with larger height. Fig. 1 shows the delay and area tradeoff of buffers and inverters at foundry 28 nm linear programming (LP) technology.[1] Each circle represents a buffer or an inverter. In red are 12 T cells, and in blue are 8 T cells. As expected, 12 T cells can achieve smaller delay at the cost of larger area.

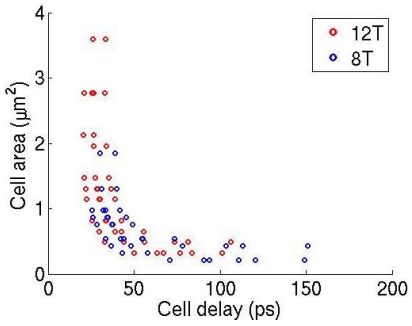

Fig. 1. Delay-area tradeoff of 8 T and 12 T buffers/inverters in 28 nm LP foundry libraries. Corner: [Slow (nMOS) slow (pMOS) process (SS), 0.95 V, 125 °C]. Load cap = FO4 + 20 μm M3 wire.

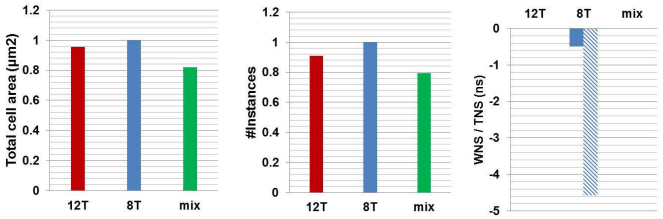

Fig. 2. Post-synthesis netlist with mixed cell heights has significant area reduction compared to 12 T-only and 8 T-only netlists. Technology: 28 nm LP. Design: AES. Frequency: 1.5 GHz. Corner: (SS, 0.95 V, 125 °C). Total cell area and number of instances are normalized to those of the 8 T-only case. In the right figure, the solid bar indicates worst negative slack (WNS), and the shaded bar indicates TNS. Implementations with 12 T-only and mixed cell heights have no timing violations.

Manuscript received November 28, 2016; revised March 15, 2017 and May 31, 2017; accepted June 21, 2017. Date of publication July 24, 2017; date of current version March 29, 2018. A preliminary version of this work appeared at the 2015 IEEE/ACM International Conference on Computer-Aided Design [3]. This paper was recommended by Associate Editor I. H.-R. Jiang. *(Corresponding author: Jiajia Li.)*

S. A. Dobre is with Qualcomm Technologies, Inc., San Diego, CA 92121 USA (e-mail: sdobre@qti.qualcomm.com).

A. B. Kahng and J. Li are with the University of California at San Diego, La Jolla, CA 92093 USA (e-mail: abk@ucsd.edu; jil150@ucsd.edu).

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

Digital Object Identifier 10.1109/TCAD.2017.2731679

[1]The area values of buffers and inverters are from the liberty models in a foundry 28LP technology. We further estimate the intrinsic delay of a buffer or an inverter based on delay lookup tables from liberty models. Specifically, we assume fanout-of-four load capacitance together with an extra capacitance of 20 μm M3 wire, and an initial input slew of 30 ps. We then iteratively feed the output slew of the cell to its input pin until the change of the output slew value is less than 1 ps. We then report the cell delay with the stabilized slew value in Fig. 1.

Given that cells of different heights exhibit different tradeoffs among performance, power, and area, mixing cells of different heights in a design is able to provide a larger solution space and—up to some practical limit—improved design quality. Fig. 2 shows the post-synthesis area and timing comparison among implementations of an open-source design AES [17] with 12 T-only, 8 T-only, and mixed cell heights. Due to generally larger areas of cell instances (particularly those with low drive strengths), 12 T-only implementation tends to have larger design area. On the other hand, weak drive strengths of 8 T cells result in a large number of buffer insertions to meet timing constraints, which also increases design area.[2] We observe from the example that mixing cell heights achieves 14% and 18% area reduction at the post-synthesis stage versus the 12 T-only and 8 T-only implementations, respectively. Note that since the example only compares post-synthesis solutions, where layout effects are not considered, the example only shows a (motivating) estimation of the potential block-level benefits from mixing cell heights.

Motivated by the above observations, in this paper we propose to mix cell heights at the sub-block level (i.e., within a single P&R block) of physical implementation, to achieve improved design quality—specifically, tradeoffs of achievable performance, power, and area.[3] However, optimizing a design by mixing noninteger multiple cell heights is highly nontrivial. The challenges include the following.

1) Current design methodologies and tool flows can only mix cells of different heights at the block level, i.e., each block of a design uses cells with a particular height, with fine-grained mixing not available with today's EDA tools.
2) There is a "chicken-and-egg" quandary: heights of cell rows are defined (in the placement site map) at the floorplan stage, but the optimized choices of cell heights are highly dependent on the placement outcome and timing constraints.
3) There are costs associated with mixing cells of different heights. For instance, "breaker cells" must be inserted for row alignment between sub-blocks of cell rows with different heights. We define a *breaker cell* as the space (i.e., placement and/or routing blockages) that must be inserted between the boundaries of regions of different cell heights.

The contributions of this paper are as follows.

1) To our knowledge, we are the first in the literature to propose mixed cell-height implementation with noninteger multiple heights at the sub-block or subisland level in advanced nodes.
2) We develop methodologies which can easily be integrated within existing physical design flow, using standard commercial tools, for mixed cell-height implementation.
3) We show that mixing 12 T and 8 T cells in a 28 nm LP foundry technology achieves up to over 30% area and power reductions versus 12 T-only implementation while maintaining the same performance, and up to over 10% performance improvement along with power and area reductions versus 8 T-only implementation.

[2]The larger total cell area of the 8 T-only netlist as compared to that of the 12 T-only netlist is due to tight timing constraints. We demonstrate in Section V that with loose timing constraints, 12 T-only implementations typically incur area overhead as compared to 8 T-only implementations.

[3]Since today's P&R tools already support placement of integer multiple-height standard cells (e.g., [20]), this paper mainly focuses on implementation with noninteger multiple-height cells, which has not been addressed in the literature.

II. Related Works

To our knowledge, there is no previous work on mixed cell-height design methodology that addresses noninteger multiple heights within a block. A recent work [12] optimizes cell placement with heights being integer multiples of a particular cell height. And, commercial placers have been capable of handling multicell row height cells (i.e., with heights that are integer multiples of a single-row height) for over two decades. However, placement of single-height, double-height, triple-height, etc. cells does not require site map change (i.e., the floorplan rows and placement sites are fixed during the optimization), whereas in our problem, different noninteger cell heights require different row and site definitions. Therefore, the placement problem involving cells with multiple noninteger heights cannot be solved by existing optimizations that handle the integer-multiples problem.

Notwithstanding the above, the mixed cell-height placement problem bears some similarity to the problem of voltage island placement, in that both problems try to assign a certain attribute with different values (e.g., cell height or supply voltage) to standard-cell instances, in order to improve performance or reduce power. Further, both problems must comprehend a type of incurred cost (e.g., area overhead of breaker cells or insertion of level shifters) when performing the assignment. We therefore review exemplary works from the literature on voltage island placement.

Ching *et al.* [2] and Wu *et al.* [15] proposed partitioning methodologies to divide post-placement die area into regions which will be assigned to different supply voltages. The objectives are, respectively, to minimize the number of partitions and to handle nonrectangular partitions. Wu *et al.* [16] further considered timing constraints during the voltage assignment. However, the interaction with standard-cell placement is missing in all of these works. An improved optimization proposed in [14] performs incremental placement to move timing-critical cells out from the low-voltage regions by adjusting net weights and cell delays. Guo *et al.* [6] embedded their voltage-island-aware placement optimization to a partitioning-based placement algorithm to minimize the number of level shifters.

Although the voltage island placement problem has been well-studied in the previous literature, there is still no available solution for mixed cell-height design implementation due to the existence of chicken-and-egg loop between floorplan and cell height selection, additional layout constraints, and area impact of cell height choices.

III. Problem Formulation

Table I gives notations used in the following discussion.

We state the noninteger multiple-height cell placement problem as follows.[4]

Given a design (i.e., gate-level netlist), timing constraints, liberty and technology models for cell libraries with multiple track heights, and P&R block area and aspect ratio, place the

[4]Since we focus on placement with noninteger multiple-height cells, in the following discussions "mixed cell heights" or "mixed heights" refers to "mixed noninteger multiple cell heights."

TABLE I
NOTATIONS USED IN OUR WORK

Term	Meaning
h_i	available cell heights $h_0 \leq h_1 \leq \ldots \leq h_N$
(X^l, Y^b, X^r, Y^t)	coordinates of block area (i.e., standard-cell placement region)
P_j	partition in which cells are of the same height, $(0 \leq j \leq M)$
$(x_j^l, y_j^b, x_j^r, y_j^t)$	coordinates of partition P_j, $(0 \leq j \leq M)$
H_j	height of partition P_j, $(0 \leq j \leq M)$
W_j	width of partition P_j, $(0 \leq j \leq M)$
t_j	cell height corresponding to partition P_j, $(0 \leq j \leq M)$
w_{site}	placement site pitch (width)
d	shift of cell rows in vertical direction to avoid cell overlap

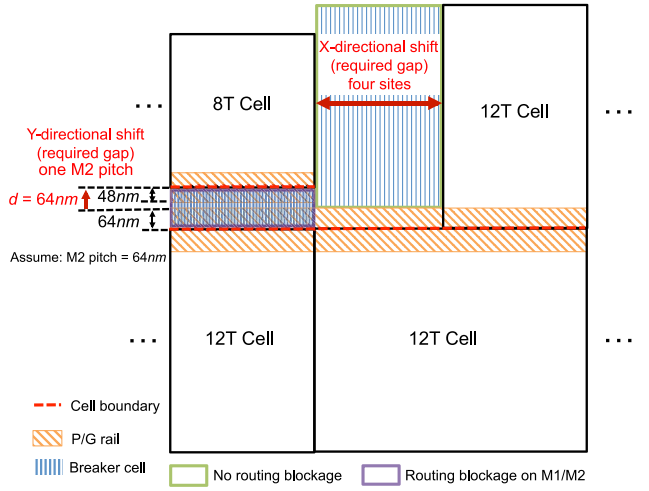

Fig. 3. Area cost of breaker cells.

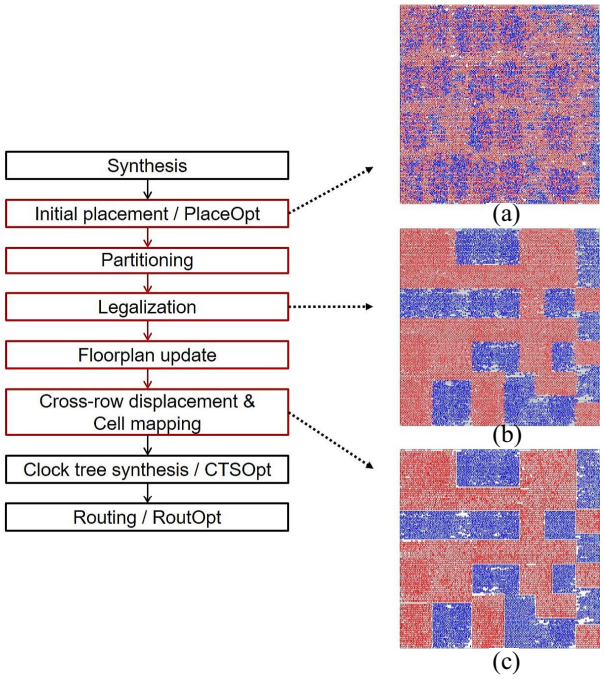

Fig. 4. Overall flow of our optimization. In the example, the maximum cut number (K) = 30. (a) Initial placement with mLEF, 8 T cell rows only. (b) Legalized placement with mLEF, 8 T cell rows only. (c) Optimized placement in updated floorplan with original LEF, mix of 12 T/8 T rows (with space for breaker cell).

design such that each cell instance is legally placed in row sites with corresponding height. The objective of the placement is to achieve minimum design power or area while maintaining the (same) target performance.

Additional layout constraints for mixed cell-height implementation applied in our studies below are as follows.[5]

- C_1: To ensure manufacturability, each region of a particular cell height must have at least two cell rows.
- C_2: Due to N-well sharing, each region of a particular cell height must have an even number of rows.
- C_3: Every region with a particular cell height must align with the block's overall metal and poly track definitions.
- C_4: The horizontal distance between two regions of different cell heights must be no less than four placement sites to honor the well-to-well spacing rule.
- C_5: The minimum vertical distance between two partitions of different cell heights must ensure that the power/ground (P/G) rail of one cell does not encroach beyond the P/G rail of another cell. In other words, there must be a vertical gap to avoid P/G track alignment issue with mixed-cell heights. (Fig. 3 shows an example with M2 pitch = 64 nm, and P/G rail width = 48 nm and 64 nm for 8 T and 12 T cells, respectively. Although the P/G rail width difference between 8 T and 12 T cells is less than one M2 pitch, to align cells to routing tracks, the minimum d in the example is 64 nm.)
- C_6: Breaker cells must be inserted to ensure the minimum horizontal and vertical distances between two regions of different cell heights.

IV. METHODOLOGY

We now describe our optimization methodology for mixed cell-height implementation. The overall optimization flow is shown in Fig. 4. Given an input design (i.e., register-transfer level netlist) and timing constraints, we first synthesize it with liberty files of all available cell heights having been made available to the logic synthesis tool. To resolve the chicken-and-egg loop between floorplan and cell height selection, we modify standard-cell library exchange format (LEF) files such that all cells have the same height (i.e., the minimum cell height among all the available heights), while maintaining the original area of each cell.[6] In the discussion below, we refer to such modified LEF files as modified LEF (mLEF). In this way, we break the chicken-and-egg loop and enable a commercial placement tool to "freely" place cells with timing-awareness. In other words, based on the synthesized netlist which contains various cell heights, we perform floorplanning with mLEF such that all cell rows have the same height (i.e., the minimum cell height among all the available cell heights). We then use a commercial placer (i.e., *Innovus*) to perform placement and post-placement optimization. Since we use the original liberty timing/power models and preserve the original area for each standard cell (although with modified aspect ratio for cells

[5]Our proposed approaches given below transparently handle other values of the parameters that define these constraints (e.g., minimum number of cell rows in a given-height region, or minimum separation between two different-height regions, etc.).

[6]In doing so, we round cell widths to the nearest whole site with no cell area reduction. E.g., given three libraries with heights 8 T, 9 T, and 12 T: 1) a 12 T, 6-site cell would be represented by an 8 T, 9-site cell and 2) a 9 T, 5-site cell would be represented by an 8 T, 6-site cell.

that originally had nonminimum height), this placement optimization is able to comprehend the tradeoff between timing constraints, power and area overheads. We emphasize that in the initial placement, we do not enforce any layout constraints or define regions for different cell heights, whereby the commercial placer is able to freely exploit the benefits from mixing cell heights. As a result, timing-critical cells tend to have larger heights (i.e., larger width with mLEF), while noncritical cells are smaller. An example initial placement solution of design AES is shown in Fig. 4(a), in which 12 T cells (with mLEF) are in red, and 8 T cells are in blue.

Based on the initial placement solution, we partition the block area into regions of particular cell heights with awareness of area cost due to breaker cells. Here, cell height refers to the original cell height as opposed to the cell height in mLEF. We then legalize the placement solution by: 1) displacement of cells (i.e., placement perturbation) and 2) swapping of cells across different heights (i.e., gate sizing). Here, we say that a placement solution is *legal* when each cell instance is placed in a region with the same height [e.g., as shown in Fig. 4(b)]. Once the placement solution is legal, we update the floorplan. Specifically, we perform refloorplanning such that the cell row height in each region is defined by the partitioning solution. We also insert space between regions with different cell heights to model the cost of breaker cells. We then map cells to the cell rows of the updated floorplan, using the original standard-cell LEF files. In the end, we perform clock tree synthesis and route the design [Fig. 4(c)]. In the following, Section IV-A describes our partitioning methodology, Section IV-B presents our placement legalization flow, and Section IV-C discusses cross-row displacement and cell mapping algorithms.

A. Floorplan Partitioning and Region Definition

We perform slicing-based partitioning using dynamic programming to divide the block area into regions of particular cell heights. Algorithm 1 shows our partitioning procedure. We first evaluate the cost of each candidate partition, i.e., $\text{cost}(x^l, y^b, x^r, y^t, 0)$, in which the fifth parameter indicates the number of cuts within the partition (line 1). If the height of a candidate partition is h_j, the cost of the partition is calculated as

$$\text{cost} = \beta \cdot \sum_{h_i \neq h_j} \text{area}_i + \lambda \cdot \Delta\text{power} + \eta \cdot \Delta\text{delay} \qquad (1)$$

where area_i is the total area of cells with height h_i located within the candidate partition; Δpower is the estimated total cell power increase by swapping cells with original cell height $h_i < h_j$ to height h_j[7]; Δdelay is the estimated total cell delay increase by swapping timing-critical cells (i.e., cells with slack $<20\%$ of the clock period) with original height $h_i > h_j$ to height h_j; and β, λ, and η are weighting factors.

Reducing the value of component $\sum_{h_i \neq h_j} \text{area}_i$ (i.e., the sum of areas of cells whose heights differ from the height of the partition) will help to minimize the cost of placement legalization (i.e., displacement and cell-height swapping) as well as the perturbation to the initial placement solution. Furthermore, to reduce the potential power and timing penalties due to legalization, we minimize the estimated cell power and delay increase within each candidate partition. More specifically, we first find the best candidate library cell (with height h_j) for each cell instance (assume that the original height of the cell instance is h_i), such that the best candidate library cell has the minimum cell power without any delay increase (resp. the minimum cell delay without any power penalty) if $h_j > h_i$ (resp. $h_j < h_i$). We then estimate Δpower and Δdelay values for each candidate partition accordingly.

Algorithm 1 DP-Based Partitioning

1: calculate $cost(x^l, y^b, x^r, y^t, 0)$ $\forall X^l \le x^l \le x^r \le X^r, Y^b \le y^b \le y^t \le Y^t$
2: **for** $k \coloneqq 1$ to K **do**
3: **for** $x^l \coloneqq X^l$ to $X^r - \Delta x$ **do**
4: **for** $y^b \coloneqq Y^b$ to $Y^t - \Delta y$ **do**
5: **for** $x^r \coloneqq x^l + \Delta x$ to X^r **do**
6: **for** $y^t \coloneqq y^b + \Delta y$ to Y^t **do**
7: $cost(x^l, y^b, x^r, y^t, k) = \min_{x^l \le x \le x^u, y^b \le y \le y^t} ($
$cost(x^l, y^b, x, y^t, k') + cost(x, y^b, x^r, y^t, k'') + 4 \cdot w_{site} \cdot (y^t - y^b),$
$cost(x^l, y^b, x^r, y, k') + cost(x^l, y, x^r, y^t, k'') + d \cdot (x^r - x^l)$
$) \quad \forall k',\ k''\ s.t.\ k' + k'' = k - 1$
8: **end for**
9: **end for**
10: **end for**
11: **end for**
12: **if** $cost(X^l, Y^b, X^r, Y^t, k) \ge cost(X^l, Y^b, X^r, Y^t, k-1)$ **then**
13: **return** $cost(X^l, Y^b, X^r, Y^t, k-1)$
14: **end if**
15: **end for**
16: **return** $cost(X^l, Y^b, X^r, Y^t, K)$

Furthermore, we set the cost of a candidate partition to infinity if it violates any of the constraints (e.g., constraints C_1 and C_2) described in Section III. More specifically, a partition (x^l, y^b, x^r, y^t) with height h_j must satisfy

$$y^t - y^b \ge 2 \cdot h_j \qquad (2)$$

$$\lfloor \frac{y^t - y^b}{2 \cdot h_j} \rfloor \cdot 2 \cdot h_j \cdot \left(x^r - x^l\right) \ge \left(y^t - y^b\right) \cdot \left(x^r - x^l\right) \cdot U_j. \qquad (3)$$

Inequality (2) forces each partition to have at least two rows. Inequality (3) ensures that partitions in the updated floorplan, after rounding to an even number of rows per partition according to constraint C_2, have enough sites to place cells; here, U_j is the placement utilization within the partition. Finally, for each candidate partition, we set the height of the partition as the height which leads to the minimum cost function value.

Fig. 5 shows contour maps of power and delay costs, as well as partitioning solutions with various weighting factors of design AES.[8] The contour maps are plotted based on the sum of power or delay costs within each grid. As the value of λ decreases and the value of η increases, the area of 12 T regions and power both increase, while timing slack improves. In our experiments, we empirically set (β, λ, η) as (1, 1, 1), (1, 0.1, 1), (1, 1, 0.1), (1, 0.1, 0.1) and select the outcome with minimum power that satisfies timing constraints.

The heart of the dynamic programming recurrence (i.e., in determining the partitioning solution with minimum cost) is given in lines 2–16. We recursively search for the minimum-cost partitioning solution of a rectangular region with k cuts, and increase the value of k in each iteration up to a given maximum allowable number of cuts, K, which is a

[7]Since dynamic power dominates in our testcases, we use the gate capacitance ratio between different cell heights to estimate the power increase.

[8]We measure area, power, and delay in units of μm^2, μW, and ps for cost function estimation. A small value of β will result in large perturbation of the initial placement, such that the power, area, and timing costs due to legalization can be high. On the other hand, a large value of β will limit the timing- and power-awareness in the partitioning optimization. We empirically use $\beta = 1$ and vary the values of λ and η between 0 and 1 to explore the tradeoff between power and timing during partitioning optimization.

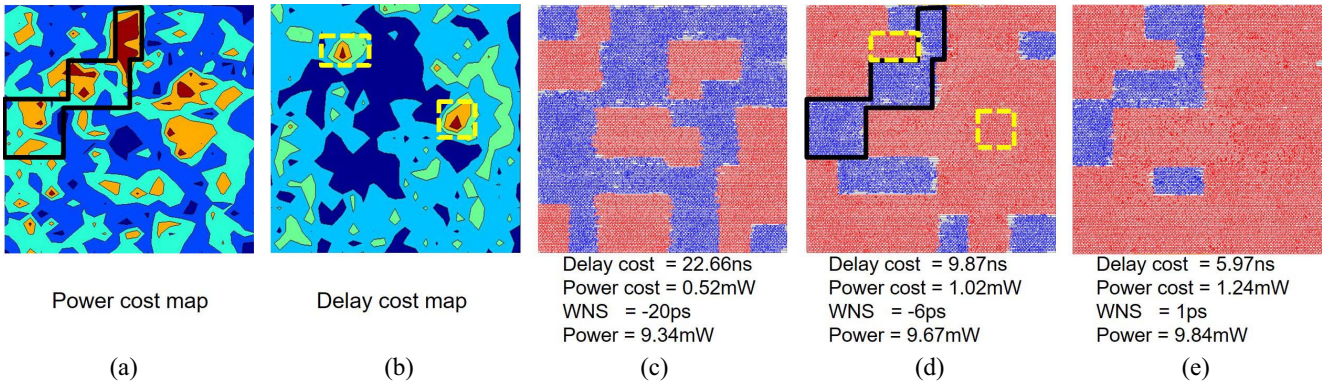

Fig. 5. (a) Contour map of power cost function. (b) Contour map of delay cost function. (c) Partitioning solution with $\beta = 1$, $\lambda = 0.8$, and $\eta = 0.2$. (d) Partitioning solution with $\beta = 1$, $\lambda = 0.7$, and $\eta = 0.3$. (e) Partitioning solution with $\beta = 1$, $\lambda = 0.6$, and $\eta = 0.4$. Cell area, power, and delay are, respectively, measured in units of μm^2, μW, and ps for cost estimation. In red are 12 T cells, and in blue are 8 T cells. Design: AES. Technology: 28 nm LP. By comprehending power and timing penalties, our partitioning optimization defines the height of regions with large power penalty as 8 T (e.g., region defined by black boundaries) and the height of regions with large timing penalty as 12 T (e.g., regions in yellow-dotted boxes).

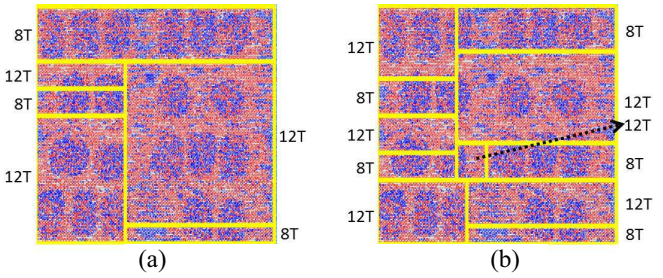

Fig. 6. Examples of partitioning solutions for the AES testcase. In red are 12 T cells (with mLEF); and in blue are 8 T cells. Yellow lines are cuts. The cell height of a partition is marked on its side. $\beta = 1$, $\lambda = 0$, and $\eta = 0$. (a) Cut number = 5 and cost = 4818 μm^2. (b) Cut number = 10 and cost = 4584 μm^2.

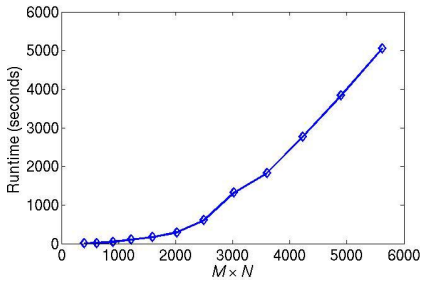

Fig. 7. Runtime of DP-based partitioning with number of grids (i.e., $M \times N$) varying from 400 to ~6000 using a single thread on a 2.5 GHz Intel Xeon server. The maximum cut number is fixed at 40.

user-defined parameter.[9] Specifically, the for loop in line 2 sweeps the number of cuts. The for loops in lines 3–6 further enumerate all regions to optimize with a given minimum grid size (i.e., $\Delta x \times \Delta y$), where (x^l, y^b) and (x^r, y^t) are, respectively, the coordinates of the lower-left and upper-right corners of a region, and (X^l, Y^b) and (X^r, Y^t) are, respectively, the coordinates of the lower-left and upper-right corners of the design block. Fig. 6 and its caption tell us that the total cost within partitions reduces as the number of cuts increases.[10] However, the area cost of breaker cells increases with the number of cuts. We therefore sweep the number of cuts during our partitioning optimization and select the solution with the minimum total cost.

To find the best partitioning solution of a region (x^l, y^b, x^r, y^t) using exactly k cuts, we observe that such a solution can always be seen as a single "top-level" cut, along with the best solutions of the two subregions induced by that cut. Hence, to find the best k-cut solution, we enumerate all potential vertical and horizontal cuts of the region (where x and y in line 7, respectively, indicate the location of the vertical and horizontal cuts), and select the solution that minimizes the sum of the costs of the two separate parts (subregions)—with respective number of cuts k' and k'' satisfying $k'+k'' = k-1$—plus the cost of the single vertical or horizontal top-level cut.

[9]In our experiments, we set K to a large value (e.g., 30 for a 100 μm×100 μm floorplan) in order to ensure good solution quality.

[10]For clarify of illustration, we only consider the component of area cost in Fig. 6, which is indicated by the amount of red (12 T) or blue (8 T) area in the figure. Therefore, the costs are measured in units of μm^2.

Note that the proposed partitioning comprehends the area cost of breaker cells, for which width $= 4 \cdot w_{\text{site}}$ for a vertical cut, and height $= d$ for a horizontal cut (line 7). For the example shown in Fig. 3, d must be larger than 64 nm. The procedure terminates when the cost does not decrease with an increased cut number (line 13), or the maximum cut number K is achieved (line 16). To improve the scalability, we divide the block area into $M \times N$ grids (where M and N are also user-defined parameters), and perform the proposed partitioning method on these grids. The runtime complexity of the procedure is $O((M+N)(M \cdot N \cdot K)^2)$. Fig. 7 shows the empirical runtime of our dynamic programming (DP)-based partitioning. We observe that partitioning with no larger than 50×50 grids and 40 cuts requires less than 10 min using a single thread on a 2.5 GHz Intel Xeon server. It must be noted that, as an initialization step, we traverse all cell instances in the design to estimate the potential cost of each grid. The corresponding runtime complexity of this initialization step is $O(G)$, where G is the total number of cell instances. In practice, this initialization requires less than one minute for a design with 60 K instances.

B. Timing-Aware Placement Legalization

Based on the partitioning solution, we perform iterative optimization to achieve a legal placement. Note that we still use mLEF at this optimization stage, but boundaries and cell heights of regions have been defined. We apply two knobs in our iterative heuristic: 1) displacement of a cell (e.g., moving a 12 T cell from an 8 T region to a 12 T region) and

Algorithm 2 Heuristic to Legalize Placement

```
1: while there exists a cell with a different height than its partition do
2:     list ← ∅
3:     for all cell g with a different height than its partition do
4:         calculate cost function of g
5:         add g to list
6:     end for
7:     sort list in order of decreasing cost
8:     swap_cnt ← 0
9:     for all g ∈ list do
10:            apply displacement/swapping based on cost function
11:        incremental timing analysis
12:        if slack of g < min(0, original slack of g) || whitespace_of_grid < ω then
13:            undo change
14:        else
15:            ++swap_cnt
16:        end if
17:        if swap_cnt ≥ γ · total_gate_count then
18:            apply ECOs in Innovus
19:            correlate internal timer with Innovus
20:            for all cell g in the design do
21:                downsize g
22:                incremental timing analysis
23:                if slack of g < min(0, original slack of g) then
24:                    undo change
25:                end if
26:            end for
27:            if WNS ≤ −θ · clock_period then
28:                fix maximum transition violations
29:                timing recovery
30:                apply ECOs in Innovus
31:                correlate internal timer with Innovus
32:            end if
33:        end if
34:    end for
35: end while
```

2) cell-height swapping (e.g., assign an 8 T cell to a 12 T cell master in a 12 T region via gate sizing). Both of these knobs affect timing, and cell-height swapping also affects area. Thus, to ensure that the optimization does not lead to large design quality degradation, we evaluate the timing and area impacts of each potential move (one move is a cell displacement or a cell-height swap).

Because timing analysis with commercial P&R tools is typically slow, our optimization approach requires a relatively accurate and fast timing engine. We have developed an internal timing analysis engine (i.e., *internal timer*) to guide the optimization. Our internal timer estimates gate delay and slew at an output pin based on the liberty lookup tables. It further uses D2M [1] and PERI [11] models that, respectively, estimate wire delay and slew propagation along the interconnect. Wirelength change due to cell displacement is measured by net half-perimeter wire length (HPWL), and wire capacitance and resistance are scaled correspondingly. More specifically, we scale a given net's latest correlated (with the P&R tool) wire parasitics by a ratio $WL_{\text{cur}}/WL_{\text{corr}}$, where WL_{cur} is the net's current HPWL and WL_{corr} is the net's HPWL in the placement corresponding to the latest correlation with the P&R tool. This gives us an approximate estimation of updated (i.e., current) wire parasitics, and hence of the net's wire parasitic change. To correlate the internal timer with the P&R tool (line 19 in Algorithm 2), we update cell locations, sizes, and threshold voltage (VT) types in the P&R tool through a Tcl socket, and use the P&R tool to perform placement legalization, trial route, parasitic extraction, and timing update. We then update the cell locations, wire parasitics in our internal timer correspondingly. We apply a slack offset at each pin to match the internal timing slacks to those from the P&R tool following the general approach of [9] and [13].

To comprehend wire congestion effects, we add a penalty in the form of wire resistance and capacitance scaling, based on routing demand versus supply overflows within the bounding box of a given net. We estimate overflow based on the trial routing solution from *Cadence Innovus Implementation System v16.1* [20]. More specifically, if the average horizontal (resp. vertical) routing congestion within the bounding box of a net is $X\%$, we penalize the horizontal (resp. vertical) portion of HPWL by a multiplicative factor of ($X\%$-$X_{\text{th}}\%$) whenever $X > X_{\text{th}}$. Here, $X_{\text{th}}\%$ is a threshold that we set to 95% based on separate studies. The value $X_{\text{th}}\% = 95\%$ is used in all experiments reported below.[11] To maintain the accuracy of our internal timer, we correlate timing slack, wire capacitance, and overflow information during the optimization through a Tcl socket with *Cadence Innovus Implementation System v16.1* [20]. Fig. 8 shows our optimization framework. We believe that our internal timer approach most closely resembles that of the previous work [9]; however, our internal timer better comprehends the impact of cell displacement on timing by considering both wirelength change and routing congestion information.

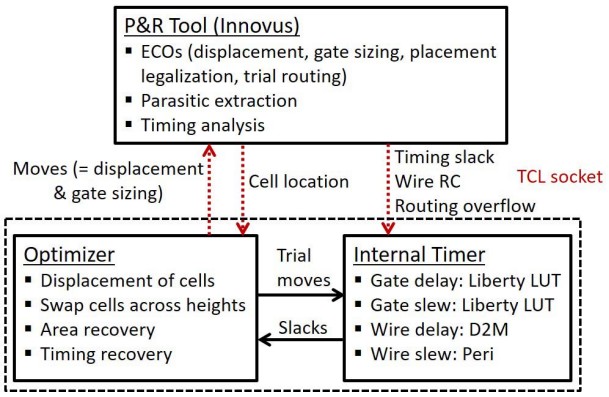

Fig. 8. Framework of our optimization.

Algorithm 2 describes our heuristic to legalize the placement. We first evaluate the cost (in terms of area and timing) of each potential move (i.e., cell displacement or swapping) (line 4). We consider cell displacement in eight directions (i.e., {N, S, E, W, NE, NW, SE, SW}) with the maximum movement distance of D ($D = 15$ μm in our experiments). The set of candidate cell displacements is similar to what is applied in the local optimization of [7]. For cell-height swapping, we consider candidate library cells whose heights match that of the partition. We use the cost function shown

$$\text{cost} = \alpha \cdot \frac{\max(0, -\Delta\text{slack})}{\max\left(1\text{ps},\ \text{slack}_{\text{orig}}\right)} + (1-\alpha) \cdot \frac{\max(0, \Delta\text{area})}{\max\left(1\mu\text{m}^2, \text{whitespace}_{\text{orig}}\right)} \quad (4)$$

where Δslack and Δarea are, respectively, the timing slack and cell area changes due to displacement and/or swapping. $\text{slack}_{\text{orig}}$ and $\text{whitespace}_{\text{orig}}$ are the original timing slack of the cell and whitespace of the corresponding grid. We divide the block area into an $M \times N$ mesh of grids. For each grid, we estimate whitespace based on placement utilization. The parameter α is a weighting factor, which has an initial value of 0.5. We adaptively change the value of α for each cell during the iterative optimization, such that when an attempt

[11] For example, if the average horizontal congestion is 98%, we multiply the x-component of HPWL by $1.03 = 0.98/0.95$.

leads to timing violation (resp. placement utilization violation), we increase (resp. decrease) α of the cell by $1.5\times$.

The cost function (4) only considers area and timing impacts due to each move. In separate studies, we have also included input pin and wire capacitance, as well as leakage power, into our cost function for legalization moves. However, the resultant solutions show negligible improvement (e.g., $<1\%$) in terms of power, area, and timing. This might be due to the positive correlations among area, pin capacitance, and leakage power. Furthermore, we adaptively change the value of α during our optimization. As a result, we observe from our experiments that moves with small area and power costs are typically selected during the early steps. As the value of α increases, moves with small delay penalties are selected during the late-stage legalization.

We sort all cells which have different height than their partition in decreasing order of cost, and apply moves to legalize the placement (line 7). When a move results in timing failure or violation of placement density, we undo the move (lines 12-13); here ω is the required whitespace according to the area of breaker cells and maximum placement density constraints.[12] We rely on the commercial P&R tool (i.e., *Innovus*) to perform incremental placement legalization. By considering the placement density, we minimize the impact of potential displacement on the cells that are already legalized. To ensure the convergence of the flow (i.e., that optimization can lead to a legalized placement), we commit the move of a cell which has been visited F times, regardless of its impact on timing and area. We use $F = 5$ in our optimization.[13] In addition, we apply a form of Tabu search [5] during the optimization to increase the likelihood of finding feasible solutions for cells. Specifically, we record the latest three attempts and forbid these moves for the current move of optimization. During the optimization, we (re-)correlate our internal timer with *Innovus* in terms of timing slack/slew, cell location, wire parasitic, and routing, overflow after every $\gamma\%$ of the total number of cells has been changed (lines 17–19), so that cost function terms will be estimated based on accurate timing and placement density information. We use $\gamma = 2$ in our optimization.[14] We also include area recovery (lines 20–26) and timing recovery (lines 27–32) in our optimization to maintain timing and area quality. The parameter θ is a threshold of slack violation that triggers timing recovery; we empirically set this to 0.15. Note that during the timing recovery, we perform backward (in which we downsize fanout cells) and forward (in which we upsize cells) maximum transition violation fixes, which enhance the timing recovery quality.

[12]In our experiments, we set the maximum placement density of the entire block as the placement density from the initial placement plus 5%.

[13]We observe in our experiments that the number of cells which have been visited six times without a feasible solution is quite small, e.g., less than 60 in a design with 15 K cells. Moreover, although such a move will cause density violation within a grid, the neighbor grids typically have enough whitespace for placement legalization (i.e., the placement density constraint of a partition can still be met). However, for a high-density design, the move of a cell must ensure that the placement density of a partition does not exceed the density constraint. Otherwise, a repartitioning step might be required.

[14]In our experiments, each correlation of our internal timer with *Innovus* takes $\sim$20 s on a design with 15 K instances and $\sim$42 s on a design with 60 K instances. The number of correlations highly depends on the initial partitioning solution (i.e., number of cell instances to be legalized). We empirically observe from our experiments that the correlation takes $\sim$15% of the total runtime of our optimization, which includes partitioning, placement legalization, and cell mapping.

TABLE II
USER-DEFINED PARAMETERS

Term	Meaning
β, λ, η	weighting factors used in partitioning cost function
K	maximum number of cuts
$M \times N$	number of grids
D	maximum displacement distance
γ	parameter to determine frequency of timing correlation
θ	slack violation tolerance threshold
F	maximum number of visits of a cell before a move is applied

We observe from our experimental results that the ratio between the number of cells being swapped and the number of cells being displaced ranges from 1.2 to 5.4. This ratio seems highly dependent on the partitioning solution, timing constraints, netlist structure, etc. For instance, fewer partitions and/or tighter timing constraints can lead to more swaps relative to displacements.

We list all the user-defined parameters applied during partitioning and placement legalization in Table II.

C. Mapping From mLEF to Original LEF in Assigned Regions

As discussed above (e.g., in the context of Fig. 4), we use mLEF with adjusted aspect ratios for cell layouts during the initial placement, partitioning, and legalization stages, where all cells have the same height (i.e., the minimum cell height h_0) but scaled cell widths. When the placement solution is legalized (i.e., each cell instance is placed in a region with the same height), we update the floorplan to insert cell rows according to the *actual* cell height of each partition. We also allocate space to model the area cost of breaker cells in the updated floorplan. Thus, there are two distinct optimizations that we apply—*cross-row cell displacement* and *cell mapping*—which we describe next.

Cells with the minimum height have their original layout aspect ratios in mLEF, and therefore no aspect ratio changes are needed to map these cells in the updated floorplan. In a partition with the minimum cell height, we only remove overlaps between cell instances and breaker cells. In other words, we perform *cross-row cell displacement* to ensure that the total cell width (i.e., total number of placement sites) in a row does not exceed the available number of placement sites with the existence of breaker cells (which are modeled as spaces in our experiments).

Algorithm 3 describes our cross-row cell displacement procedure. We perform four iterations of cross-row cell displacements. The first and the third iterations are top-to-bottom, traversing from the topmost row to the bottommost row, and optimizing one row at a time. The second and the fourth iterations optimize similarly, but in a bottom-to-top manner. Furthermore, in the first and second iterations, we move cells to adjacent rows only if there is enough space in the adjacent rows. In the third (resp. fourth) iteration, we force cells to move to the lower (resp. upper) adjacent row regardless of available space in the adjacent row. This strategy enables chained moves of cells across rows. In Algorithm 3, k is the row index, which ranges from 1 (the bottommost row) to R (the topmost row); W_i is the total cell width in the ith row; $W_i^{\max}$ is the maximum allowed cell width in the ith row; $w(g)$ is the width of cell g; Δ records the HPWL increase due to cross-row cell displacement; and function move(g, dir) moves

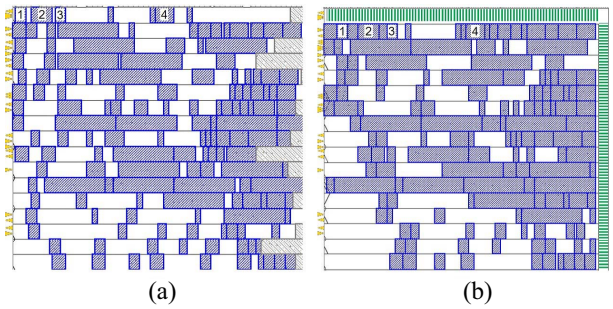

Fig. 9. Example of cross-row cell displacement (an 8 T region located at the lower-left corner of the DMA block). (a) Before cross-row cell displacement. In blue are 8 T cells. In gray are 12 T cells. (b) After cross-row cell displacement. In green are breaker cells. Breaker cells introduce routing blockages on M1/M2, whereas ports (in yellow) are placed on M3 and above.

Algorithm 3 Cross-Row Cell Displacement

1: **for** $k := 1$ to 4 **do**
2: **for all** i^{th} row in the partition **do**
3: **while** i^{th} row has capacity/overlap violation **do**
4: $\Delta_{min} \leftarrow +\infty$; $g_{move} \leftarrow \emptyset$
5: **for all** $g \in i^{th}$ row **do**
6: **if** $i \neq 1$ && $(W_{i-1} + w(g) \leq W_{i-1}^{max} \mid\mid k == 3)$ **then**
7: $\Delta \leftarrow$ HPWL increase by moving g to $(i-1)^{th}$ row
8: **if** $\Delta < \Delta_{min}$ **then**
9: $\Delta_{min} \leftarrow \Delta$; $g_{move} \leftarrow g$; dir = down
10: **end if**
11: **end if**
12: **if** $i \neq R$ && $(W_{i+1} + w(g) \leq W_{i+1}^{max} \mid\mid k == 4)$ **then**
13: $\Delta \leftarrow$ HPWL increase by moving g to $(i+1)^{th}$ row
14: **if** $\Delta < \Delta_{min}$ **then**
15: $\Delta_{min} \leftarrow \Delta$; $g_{move} \leftarrow g$; dir = up
16: **end if**
17: **end if**
18: **end for**
19: **if** $g_{move} == \emptyset$ **then**
20: **break**
21: **else**
22: $move(g_{move}, dir)$
23: **end if**
24: **end while**
25: **end for**
26: **end for**

cell g to the row that is adjacent in direction dir. After the cross-row cell displacements, we perform single-row incremental placement optimization using dynamic programming to further reduce wirelength.[15] Fig. 9 shows an example of our cross-row displacement optimization.

On the other hand, cells originally having large height (i.e., larger than the minimum cell height) become shorter and wider in mLEF. We must recover the original aspect ratio of cell layouts in the updated cell rows with actual cell heights. For example, assume that there are 20 10 T cells uniformly placed in five 8 T cell rows (i.e., as a 5 × 4 mesh). To update the floorplan, we maintain the same partition area and place cell rows according to the height of the partition. We therefore have four 10 T cell rows. Given that the layout of these 10 T cells (with the same cell area) are scaled back to their original height with a reduced cell width, five cells now can fit into one row in the updated floorplan. The mapped cell placement becomes a 4 × 5 mesh. Note that in this mapping procedure, the region outlines are fixed, and we only update the cell row height within each region [as illustrated in Figs. 4(b)–(c) and 11]. As shown in the example, cell mapping in the updated floorplan

[15]Our dynamic programming formulation is the same as the "minimum HPWL" formulation in [10]. Formulation details are given in [10].

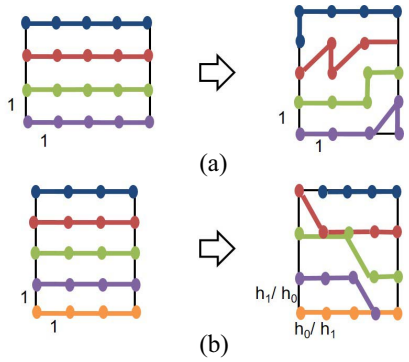

Fig. 10. Illustration of graph embedding (a) from [4] and (b) for proposed cell mapping. Vertical connections are not shown.

can be viewed as embedding a graph to another graph with a different aspect ratio (e.g., embed a 5 × 4 mesh to a 4 × 5 mesh). We therefore revisit the graph embedding literature.

Ellis [4] showed that to embed a 2-D mesh of size $w \times h$ (with unit distance between every two adjacent nodes in both horizontal and vertical directions) to another 2-D mesh of size $w' \times h'$, where $w' < w$ and h' is the smallest integer satisfying $w' \cdot h' \geq w \cdot h$, if (w/w') is no larger than 2, the maximum wirelength of a two-pin net (in Manhattan distance) in the embedded graph is no more than two units. An example with $(w/w') = (5/4)$ is shown in Fig. 10(a): the wirelength of each connection in the original graph is one, and the maximum wirelength in the embedded graph (i.e., the diagonal connection) is two. Note that our optimization of cell mapping differs from [4], in that [4] varies the area of the graph (i.e., mesh) while our optimization assumes a fixed mesh area (i.e., area of a partition).

Following the discussions in [4], we can show that if we map a 2-D-mesh placement with cell height h_0, in which all cells have the same cell area, to another 2-D-mesh placement with cell height h_1, the maximum wirelength scaling of a mesh edge (i.e., two-pin net) according to the mapping is no more than $(h_0/h_1) + (h_1/h_0)$.[16] Fig. 10(b) shows an example with 10 T and 8 T cells. Assuming unit wirelength for each two-pin connection between any horizontally or vertically adjacent cells in the original 2-D-mesh placement, the maximum wirelength increase is 1.05.

Inspired by the graph-embedding theory, we propose an approach to map cells to cell rows with original cell heights for general cases, in which cells have different widths and are not necessarily placed in a 2-D mesh. Algorithm 4 shows our procedure to map cells from an initial floorplan with R rows of height h_0 to an updated floorplan with R' rows of height h_j. We first estimate the average total cell width of each cell row in the updated floorplan (line 1), in which g is a cell in partition P_j; $w(g)$ is the actual width of cell g corresponding to height h_j. We then store cells in the ith row from the initial floorplan into list_1 and sort them by increasing order of their X-coordinates (lines 5–6). For each cell on the $(i+1)^{\text{th}}$ row of the initial floorplan, we estimate the wirelength (i.e., HPWL) difference between the case of assigning the cell to the (i')th row of the updated floorplan versus the case of assigning the cell to the $(i'+1)$th row of the updated floorplan (lines 9-12). We then sort the cells on the $(i+1)$th row of the initial floorplan by the corresponding delta wirelength

[16]Proof details are given in [4].

Algorithm 4 Cell Mapping

1: $W_{avg} = (\sum_{g \in P_j} w(g))\ /R'$
2: $i = 1$
3: $list' \leftarrow$ cells in i^{th} row from the initial floorplan
4: **for** $i' := 1$ to R' **do**
5: $\quad list_1 \leftarrow list'$
6: $\quad$ sort $list_1$ in order of increasing cells' X-coordinate
7: $\quad W \leftarrow$ total width of $list_1$
8: $\quad ++i$
9: $\quad$ **for all** cell g in i^{th} row from the initial floorplan **do**
10: $\quad\quad \Delta(g) \leftarrow HPWL\_diff(g, i', i'+1)$
11: $\quad\quad list'.push(g)$
12: $\quad$ **end for**
13: $\quad$ sort $list'$ in order of increasing $\Delta(g)$
14: $\quad$ **while** $list' \neq \emptyset$ && $W \leq Min(1.05 \cdot W_{avg},\ W_{i'}^{max})$ **do**
15: $\quad\quad g \leftarrow list'.pop()$
16: $\quad\quad list_2.push(g)$
17: $\quad\quad W \leftarrow W + w(g)$
18: $\quad$ **end while**
19: $\quad$ sort $list_2$ in order of increasing cells' X-coordinate
20: $\quad$ **if** $list' == \emptyset$ **then**
21: $\quad\quad ++i$
22: $\quad\quad list' \leftarrow$ cells in i^{th} row from the initial floorplan
23: $\quad$ **end if**
24: $\quad DPPlace(i', list_1, list_2)$
25: **end for**

values (line 13). We iteratively add these cells to list_2 until the total width of cells in list_1 and list_2 exceeds $1.05\times$ of the average total cell width of each row (W_{avg}) or the maximum allowed total width in the (i')th row ($W_{i'}^{\text{max}}$, where the width of breaker cells is considered) (lines 14–18). If all cells from list' are added to list_1 and list_2, we fill list' with cells from the next row in the initial floorplan (lines 21–22). Finally, we use dynamic programming to order and place cells from list_1 and list_2 onto the (i')th row in the updated floorplan. In summary, lines 1–23 of Algorithm 4 determine Y-coordinates of cells and optimize the vertical component of HPWL; while the $DPPlacer(i', \text{list}_1, \text{list}_2)$ further minimizes the horizontal component of HPWL. As an improvement to [8] which also uses dynamic programming to optimally order two rows of cells into a single row with minimized wirelength, our formulation also determines the optimal cell locations. The recurrence relation in our dynamic programming optimization is

$$\text{sol}(i,j,k) = \text{Min}\begin{cases} \text{sol}(i,j,k-1) \\ \text{sol}(i-1,j,k-w(g_i)) + \text{cost}(g_i,k) \\ \text{sol}(i,j-1,k-w(g_j)) + \text{cost}(g_j,k) \end{cases}$$

where $\text{sol}(i,j,k)$ is the wirelength corresponding to the optimal placement solution of the first i cells in list_1 and the first j cells in list_2 within the first k placement sites in the updated cell row; g_i is the ith cell in list_1; g_j is the jth cell in list_2; $w(g)$ is cell width (i.e., in terms of the number of sites) of g; and $\text{cost}(g,k)$ is the HPWL increase by allocating g (i.e., right edge of g) at the kth placement site.

Fig. 11 shows an example of our cell mapping optimization. With a fixed region outline, we map 21 rows of 12 T cells in 8 T height (i.e., using mLEF) to 14 rows of 12 T cells in 12 T height. Fig. 11 also shows inserted breaker cells around the 12 T region.

Fig. 12 shows the optimized wirelength (i.e., HPWL) comparison between our proposed method (using dynamic programming) versus a greedy method proposed in [3]. We observe that our proposed optimization achieves up to 16% wirelength reduction compared to the greedy method. Furthermore, it is obvious that the proposed algorithm can achieve the mapping solution or a solution with the same total wirelength shown in Fig. 10(b). We note that the procedure

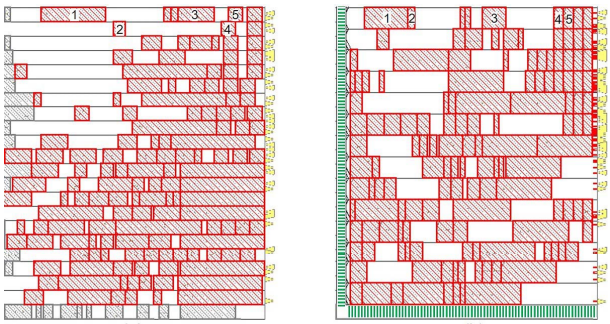

Fig. 11. Example of cell mapping (a 12 T region located at upper-right corner of the block). (a) Before mapping optimization. 12 T cells (in red) are of 8 T cell height. In gray are 8 T cells. (b) After mapping optimization. 12 T cells (in red) are of 12 T cell heights. (8 T cells are moved to neighboring 8 T regions.) In green are breaker cells. The mapping of five specific cell instances is shown by labels 1–5.

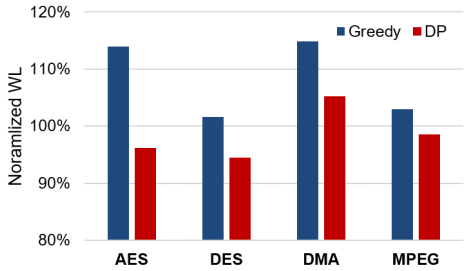

Fig. 12. Wirelength comparison between our dynamic programming-based optimization versus a greedy optimization in [3]. Wirelength values are normalized to the wirelength before cell mapping.

TABLE III
BENCHMARKS

Design	#Instances	#Flip-flops	Clock period range
AES	∼16K	530	650ps – 800ps
DES	∼23K	1984	600ps – 750ps
JPEG	∼60K	4512	750ps – 900ps
MPEG	∼16K	3193	550ps – 700ps

described in Algorithm 4 only applies to the case, where the ratio between h_j and h_0 is no larger than two. We can easily extend our mapping procedure to address cases with height ratio greater than two by extending our dynamic programming formulation to optimize more than two lists of cells.

V. Experimental Results

We perform experiments in a 28 nm LP foundry technology with dual-VT libraries, 0.95 V nominal supply voltage, and cell height choices 12 T and 8 T. To confirm that our optimization can perform a fine-grained mixed cell-height implementation, we select four design blocks (AES, DES, JPEG, MPEG) from the OpenCores [17] website. Parameters of these four testcases are shown in Table III. For each design, we determine a range of clock periods starting from a clock period with relative loose timing constraint, up to the clock period at which the 8 T-only implementation shows setup timing violations. These designs are synthesized using *Synopsys Design Compiler vI-2013.12-SP3* [18] and then placed and routed using *Cadence Innovus Implementation System v16.1* [20].

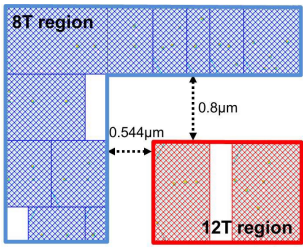

Fig. 13. Inserted space on the boundaries between 12 T and 8 T regions to model the cost of breaker cells.

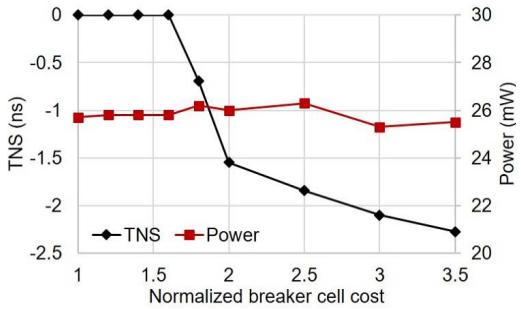

Fig. 14. Results with different breaker cell area cost values. Design: AES. Frequency: 1.5 GHz.

In our experiments, we set the gate density at the floorplan stage as 60% and the aspect ratio of the floorplan as 1. We use the same commands and options for placement, post-placement optimization, clock tree synthesis (CTS), post-CTS optimization, routing and post-routing optimization for single-height and mixed-height cases. We, respectively, use *Cadence Innovus Implementation System* and *Synopsys PrimeTime-PX vH-2013.06-SP3* (PT-PX) [19] for timing and power analysis at the post-routing stage and wire parasitics (standard parasitic exchange format) obtained from Innovus. We use *Synopsys PrimeTime vH-2013.06-SP2* [19] to search for the minimum supply voltage that satisfies a given frequency target. Our optimization flow is implemented in C++. Functions used in P&R tools and the socket between our optimizer and the P&R tool are implemented in Tcl. We conduct our experiments on a 2.5 GHz Intel Xeon server.

Modeling Breaker Cell Costs: The placement site pitch (width) and the M2 metal pitch in the 28 nm LP technology that we use are, respectively, 0.136 μm and 0.1 μm. Based on the discussion in Section III, the horizontal and vertical shifts between any 8 T and 12 T regions must be no less than 0.544 μm and 0.1 μm, respectively. In our experiments, we shift cell rows by 0.8 μm in the vertical direction between any 12 T and 8 T regions (Fig. 13).[17] We also insert placement and routing blockages correspondingly. In FinFET technology, layout constraints for mixed-height design become more complicated (e.g., fin alignment). To evaluate the impact of breaker cell cost, we perform experiments with up to 3.5X area cost of the breaker cell in 28 nm technology. Results in Fig. 14 show small performance and power degradation when breaker cell area is $<\sim$2X. When breaker cell area increases beyond $\sim$2X, timing violation increases due to placement and routing congestion.

A. Performance-Area Tradeoff Comparison

We implement our benchmark designs using our proposed flow with mixed 8 T/12 T cells. We also perform conventional synthesis, placement, clock tree synthesis, and routing (SP&R) with 12 T-only cells and 8 T-only cells for comparison. The designs are implemented with clock periods shown in Table III. We use the nominal voltage 0.95 V at (SS, 125 °C) corner for design implementation and timing analysis. Table IV shows our experimental results, in which the clock period is the clock period used for implementation and each benchmark design is implemented in four clock periods. Total power values are reported at the sign-off frequency. We divide the block area of each design into grids of size around 6 μm $\times$ 6 μm for partitioning (i.e., $\Delta x = 6$ μm and $\Delta y = 6$ μm) as illustrated in Section IV-A, and for placement density constraint evaluation as illustrated in Section IV-B. Fig. 15 further shows the Pareto curves illustrating tradeoffs between performance and area of implemented designs at the post-routing stage, where the frequency given is the maximum achievable operating frequency.

Results show that by mixing 8 T and 12 T cells, our optimization achieves significant area reduction (e.g., over 20% on design AES) compared to designs with only single-height cells, especially for comparison to 12 T-only designs (e.g., over 30% area reduction on design AES).[18] This is because mixed cell heights provide a wider range of tradeoff between performance and area such that 8 T cells are applied to timing paths with large slacks for area reduction, and 12 T cells are used in timing-critical paths to meet timing constraints. Furthermore, with loose timing constraints, the area benefit of mixed cell-height designs over 8 T-only designs reduces. On the other hand, mixed cell heights also have similar or even higher maximum achievable performance compared to designs with only single-height cells. For instance, we observe up to 13% performance improvement from mixed cell heights over 8 T-only designs (i.e., on design AES). This is because the maximum achievable performance of an 8 T-only design is limited by the weak drive strengths of 8 T cells. Moreover, mixed cell heights are able to have more compact area, which reduces wire capacitance, as well as smaller pin capacitance (i.e., by using 8 T cells) on nontiming critical fanouts of a timing-critical driver (which is typically a 12 T cell) compared to 12 T-only designs.

Experimental results also show that our optimization with mixed cell heights offers comparable routed wirelength compared to 8 T-only designs and smaller routed wirelength

[17]Due to the lack of mixed-height implementation flow in commercial P&R tools, we implement a mixed-height design using a multiple power domain flow in *Innovus* [20], where regions with different cell heights are defined as different power domains. Such a flow allows different cell row heights within a design block. Further, due to the limitation of the commercial P&R tools, we insert space between regions (by defining the "minGap" attribute) to model the breaker cell area cost. As a result, cell rows must be aligned and the minimum vertical shift is 0.8 μm instead of 0.1 μm.

[18]Note that our mixed-height optimization on design AES (clock period = 800 ps) results in a 8 T-only design, but with 27% area reduction compared to that of the 8 T-only implementation. This might be due to pessimism and a large number of inserted buffers during the synthesis stage of the 8 T-only flow. Specifically, at the post-synthesis and initial placement stages, the mixed-height design (which contains 18% 12 T cells) achieves 5% area reduction and 7% power reduction compared to the 8 T-only design. Moreover, our partitioning solution swaps all 12 T cells in the mixed-height design to 8 T cells, which results in 25% load pin capacitance reduction on average for each cell, and hence further reductions of area and power (but without timing violation due to the loose timing constraints).

TABLE IV
PARAMETERS AND RESULTS OF IMPLEMENTED DESIGNS

Flow	12T	8T	mix	12T	8T	mix	12T	8T	mix	12T	8T	mix
Design (clk period)	AES (650ps)			DES (600ps)			JPEG (750ps)			MPEG (550ps)		
#Instances	14466	16056	14967	19896	23699	24738	52887	60137	57756	14117	15709	14820
12T/8T	14466/0	0/16056	14185/782	19896/0	0/23699	471/24267	52887/0	0/60137	4304/53452	14117/0	0/15709	1261/13559
LVT/RVT	11090/3376	14249/1807	2835/12132	10730/9166	18991/4708	12012/12726	20844/32043	32746/27391	20541/37215	3655/10462	7342/8367	2800/12020
Setup WNS (ps)	-54	-119	1	-0	-18	1	-0	-20	-0	-10	-143	2
Setup TNS (ns)	-2.755	-18.963	0.000	-0.000	-1.083	0.000	-0.000	-2.741	-0.000	-0.107	-18.861	0.000
#Hold violations	0	0	0	0	0	0	0	0	0	0	0	0
Area (μm^2)	15481	13582	12190	20470	18218	15598	59998	49156	41703	21055	15494	13730
Utilization	85%	89%	81%	61%	69%	56%	63%	66%	56%	62%	70%	63%
WL (mm)	160	151	157	183	173	177	535	512	510	150	148	152
Leakage (mW)	0.489	0.366	0.158	0.346	0.385	0.203	1.017	0.959	0.511	0.216	0.206	0.090
Total power (mW)	35.0	30.0	25.7	57.2	52.9	44.9	67.1	58.6	51.4	34.4	30.2	27.7
Clock area (μm^2)	15	13	12	53	41	39	158	131	112	102	167	64
Clock WL (μm)	2017	1841	1463	8050	6834	6490	23977	19549	17912	11678	10620	10470
Runtime (min)	146	147	224	113	140	257	237	265	514	57	91	163
Design (clk period)	AES (700ps)			DES (650ps)			JPEG (800ps)			MPEG (600ps)		
#Instances	14126	15688	14594	18504	22708	22321	54439	57310	56084	13776	15957	13655
12T/8T	14126/0	0/15688	437/14157	18504/0	0/22708	1692/20629	54439/0	0/57310	106/55978	13776/0	0/15957	614/13041
LVT/RVT	8361/5765	13432/2256	6226/8368	8905/9599	16021/6687	7675/14646	20489/33950	27695/29615	19556/36528	2621/11155	5217/10740	2105/11550
Setup WNS (ps)	-3	-60	1	-0	-13	0	-2	-8	-0	1	-82	-5
Setup TNS (ns)	-0.009	-7.452	0.000	-0.000	-0.367	0.000	-0.005	-0.182	-0.000	0.000	-8.482	-0.048
#Hold violations	0	0	0	0	0	0	0	0	0	0	0	0
Area (μm^2)	14021	13241	8755	19586	16186	14539	59529	44998	41240	20171	14849	13268
Utilization	72%	93%	62%	62%	65%	59%	66%	64%	59%	63%	68%	62%
WL (mm)	166	141	135	180	159	158	518	477	468	147	139	136
Leakage (mW)	0.387	0.350	0.128	0.290	0.302	0.144	1.038	0.760	0.516	0.164	0.153	0.081
Total power (mW)	29.4	27.5	17.7	51.0	44.6	39.1	59.6	50.9	46.3	30.9	26.4	24.9
Clock area (μm^2)	15	13	11	49	35	36	109	100	93	81	102	65
Clock WL (μm)	2079	1915	1363	7823	6107	6019	21308	22895	16612	11631	11270	9724
Runtime (min)	136	158	128	91	151	176	239	310	495	37	65	107
Design (clk period)	AES (750ps)			DES (700ps)			JPEG (850ps)			MPEG (650ps)		
#Instances	13805	14981	14122	18123	22206	21782	52401	58055	57304	13421	14490	13478
12T/8T	13805/0	0/14981	282/13840	18123/0	0/22206	842/20940	52401/0	0/58055	418/56886	13421/0	0/14490	1088/12390
LVT/RVT	6844/6961	11385/3596	4013/10109	7677/10446	14688/7518	5899/15883	18120/34281	25312/32743	15373/41931	2144/11277	5181/9309	1320/12158
Setup WNS (ps)	-3	-24	3	0	-8	-0	0	-11	0	1	-56	2
Setup TNS (ns)	-0.003	-1.740	0.000	0.000	-0.091	-0.000	0.000	-0.241	0.000	0.000	-5.163	0.000
#Hold violations	0	0	0	0	0	0	0	0	0	0	0	0
Area (μm^2)	12202	11901	8334	19158	15040	13767	57128	44073	40678	19717	14635	13127
Utilization	66%	91%	58%	65%	66%	61%	65%	69%	63%	62%	70%	61%
WL (mm)	157	137	131	171	154	156	539	425	423	149	143	135
Leakage (mW)	0.275	0.290	0.096	0.253	0.243	0.104	0.915	0.679	0.423	0.141	0.147	0.063
Total power (mW)	24.0	22.7	16.3	46.1	39.1	34.9	54.5	48.1	44.1	28.5	23.3	22.9
Clock area (μm^2)	19	10	12	47	29	40	100	106	95	82	58	51
Clock WL (μm)	1960	1683	1959	7605	5933	6326	20133	18441	16055	11709	10027	9734
Runtime (min)	49	73	113	97	111	93	189	244	579	59	91	63
Design (clk period)	AES (800ps)			DES (750ps)			JPEG (900ps)			MPEG (700ps)		
#Instances	12883	14480	14043	17636	20178	19788	51991	57388	56287	11977	14514	13445
12T/8T	12883/0	0/14480	0/14043	17636/0	0/20178	1427/18361	51991/0	0/57388	212/56075	11977/0	0/14514	253/13192
LVT/RVT	5453/7430	9557/4923	4907/9136	6754/10882	12355/7823	5087/14701	14054/37937	25056/32332	14807/41480	1774/10203	4144/10370	1079/12366
Setup WNS (ps)	0	-11	2	-0	-6	1	0	-11	0	1	-12	3
Setup TNS (ns)	0.000	-0.280	0.000	-0.000	-0.024	0.000	0.000	-0.375	0.000	0.000	-0.455	0.000
#Hold violations	0	0	0	0	0	0	0	0	0	0	0	0
Area (μm^2)	11294	10244	7402	18637	14359	13447	54647	44758	40400	19434	14038	12794
Utilization	63%	74%	54%	65%	64%	61%	60%	71%	64%	63%	67%	60%
WL (mm)	155	133	128	165	150	153	591	483	479	159	138	140
Leakage (mW)	0.220	0.224	0.078	0.220	0.207	0.094	0.711	0.699	0.413	0.129	0.116	0.054
Total power (mW)	21.2	19.2	13.7	41.8	35.5	32.3	52.0	42.7	38.4	26.8	21.4	21.0
Clock area (μm^2)	13	11	11	47	30	32	111	85	93	101	50	46
Clock WL (μm)	2004	1566	1570	7397	6451	7270	21171	17433	17862	11823	9717	9818
Runtime (min)	74	107	76	81	120	76	139	285	424	61	83	58

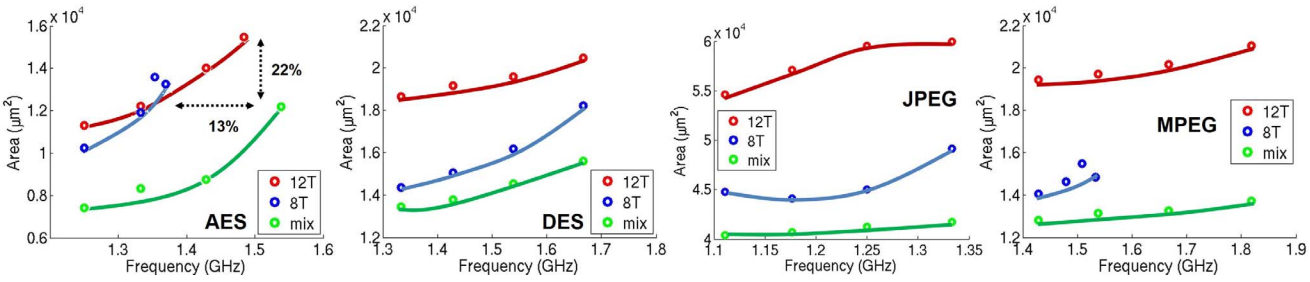

Fig. 15. Pareto curves of performance-area tradeoff for implementations with 8 T-only, 12 T-only and mixed cells.

compared to 12 T-only designs, which is mainly due to our wirelength-aware cell mapping and reduced total cell area (i.e., a more compact layout). Furthermore, in our experimental flow we use command *ccopt_design* from *Innovus* to perform clock tree synthesis. Results show that our optimized designs with mixed cell heights have similar clock tree metrics

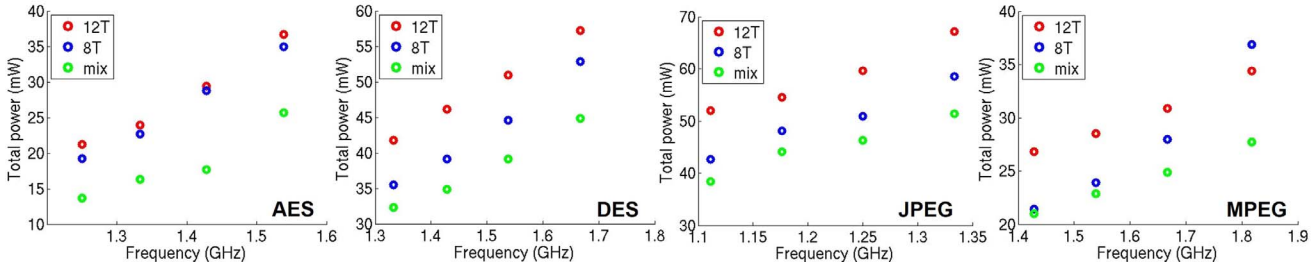

Fig. 16. Iso-performance power comparison with voltage scaling among implementations with 8 T-only, 12 T-only, and mixed cells.

(i.e., total buffer area and clock tree wirelength) compared to designs with only single-height cells. This indicates that our optimization does not incur any power and area penalties with respect to the clock tree synthesis optimization. As described in Section IV-A, we use four sets of (β, λ, and η) values to perform optimization and select the minimum-power outcome. In our experiments, we perform the four implementations in parallel and report the runtime corresponding to the minimum-power solution in Table IV. On the downside, we observe in our experiments that including more libraries (i.e., with mixed cell heights) typically increases the runtime of commercial SP&R tools. Furthermore, although we apply our internal timer during the placement legalization, the incremental timing analysis, placement legalization, and trial routing used for correlation also incur runtime overhead. Therefore, our mixed-height implementations are achieved at the cost of larger runtimes compared to single-height cases.

B. Iso-Performance Power Comparison

Given that certain designs have timing violations, to achieve a fair power comparison we perform voltage scaling on each design so that all designs meet the timing constraints. We then compare power at the scaled supply voltage. In our experiments, we define *scaling_lib_group* in the *PT-PX* tool to enable such comparisons. Note that to compensate the slack discrepancy between *Innovus* and *PrimeTime*, we apply a constant slack shift (i.e., the difference between the WNS from *PrimeTime* versus the WNS from *Innovus*) of the entire block to correlate the post-routing worst slack values, then perform voltage scaling. When the difference between the scaled voltage and the signoff voltage is larger than 30 mV, we perform SP&R with the scaled voltage and use the smaller power value between that of the initial implementation and that of the additional implementation in our comparison.

Fig. 16 shows the iso-performance power comparison. We observe that 8 T-only designs typically have smaller power compared to 12 T-only ones. The exception of design MPEG with frequency = 1.8 GHz might be due to a larger number of buffer insertion as well as voltage scaling in the 8 T-only design to meet the performance constraints. Moreover, our optimized designs with mixed cell heights provide power reduction compared to 8 T-only and 12 T-only designs. Such power reduction mainly comes from reduced cell area and wirelength, as well as power-awareness in our partitioning optimization. Similarly to area benefit, power benefit from mixed heights over 8 T-only designs also reduces at a loose timing constraint.

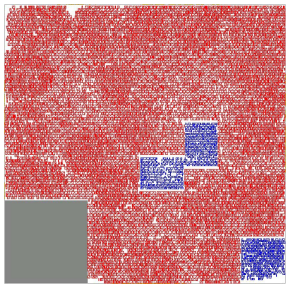

Fig. 17. Layout of AES_macro with mixed-height optimization. In gray is the placement blockage. In red are 12 T cells. In blue are 8 T cells.

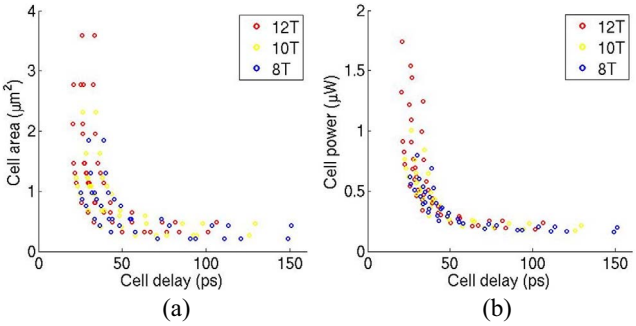

Fig. 18. (a) Delay-area tradeoff and (b) delay-power tradeoff, of 8 T, 10 T, and 12 T buffers/inverters in 28 nm LP foundry libraries. Corner: (SS, 0.95 V, 125 °C). Load cap = FO4 + 20 μm M3 wire.

C. Additional Validations

We further validate our optimization flow on three variants of the design AES: 1) design AES with tight timing constraints (AES_tight); 2) design AES with blockages to model existence of macro blocks (AES_macro); and 3) a design containing eight AES blocks (AES_x8). For AES_tight, we increase the maximum frequency of AES by 20% compared to that in Table IV. Results in Table V show that the mixed-height design achieves more than 10% power and area reductions, as well as better timing, compared to the 12 T-only design. Although the 8 T-only design shows smaller power and area compared to the mixed-height design, the 8 T-only design has huge timing violations. Furthermore, we implement design AES with inserted placement blockage (i.e., AES_macro) to model the existence of macro blocks (as illustrated in Fig. 17). We extend our optimization flow in several ways to comprehend the existence of macro blocks: 1) we treat a grid that is occupied by a macro block as having zero cost during our DP-based partitioning step; 2) we comprehend placement blockages during our placement legalization using placement density constraints; and

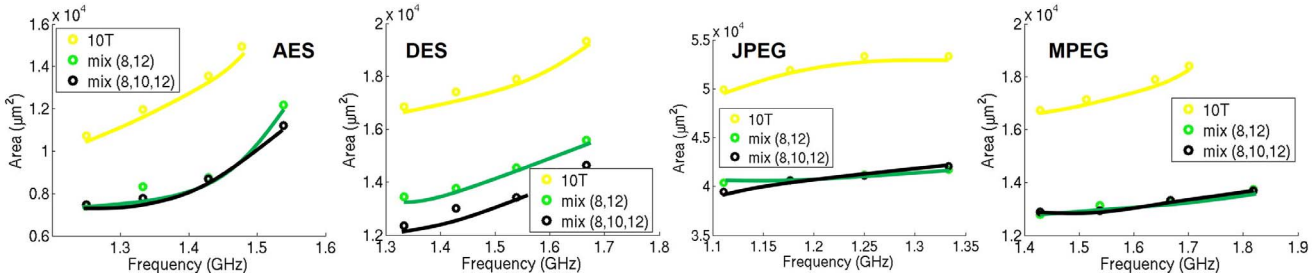

Fig. 19. Pareto curves of performance-area tradeoff for implementations with 10 T-only and mixed (8 T+12 T and 8 T+10 T+12 T) cells.

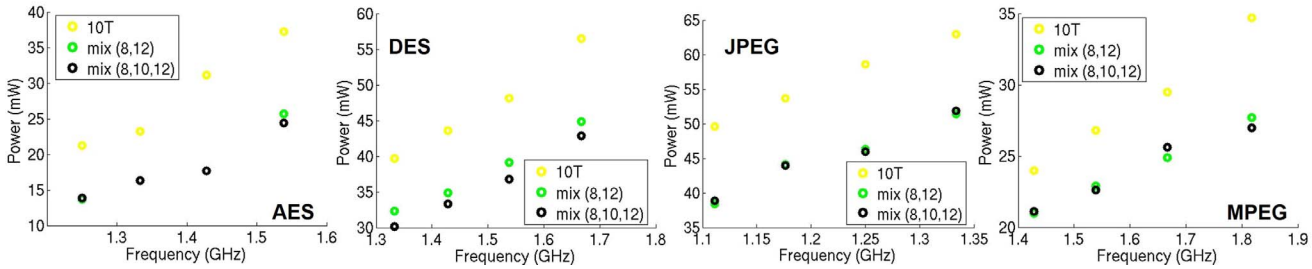

Fig. 20. Iso-performance power comparison with voltage scaling among implementations with 10 T-only and mixed (8 T + 12 T and 8 T + 10 T + 12 T) cells.

TABLE V
RESULTS OF VARIOUS AES DESIGNS

Flow	12T	8T	mix
Design (clk period)	AES_tight (550ps)		
#Instances 12T/8T	15442 15442/0	16241 0/16241	14691 541/14150
Setup WNS (ps)	-94	-230	-38
Setup TNS (ns)	-10.981	-53.379	-2.753
Area (μm^2)	19248	14144	16745
WL (mm)	198	158	181
Total power (mW)	49.5	37.1	44.6
Runtime (min)	152	116	237
Design (clk period)	AES_macro (650ps)		
#Instances 12T/8T	14679 14679/0	16354 0/16354	14985 13921/1064
Setup WNS (ps)	-26	-127	1
Setup TNS (ns)	-0.593	-16.428	0.000
Area (μm^2)	16610	14173	12032
WL (mm)	167	162	169
Total power (mW)	37.4	31.4	27.0
Runtime (min)	124	118	206
Design (clk period)	AES_x8 (650ps)		
#Instances 12T/8T	126594 126594/0	114592 0/114592	116095 19865/96230
Setup WNS (ps)	-167	-54	-52
Setup TNS (ns)	-186.59	-32.30	-13.61
Area (μm^2)	112075	129868	88016
WL (mm)	1564	1629	1459
Total power (mW)	250.6	281.8	215.0
Runtime (min)	319	521	1515

3) we avoid overlaps with macro blocks during our cell mapping optimization. Results in Table V show more than 14% reductions in power and area with our mixed-height optimization, compared to 12 T- and 8 T-only implementations. Last, to test the scalability of our optimization flow, we generate a design containing eight AES blocks with ∼120K instances.

D. Comparison to 10 T-Only Designs

We generate 10 T cell libraries by performing interpolation on timing and power tables of 12 T and 8 T libraries, which are from foundry 28LP technology.[19] We also generate cell LEF by scaling area of cells proportional to cell drive strengths according to area and drive strength information of 8 T and 12 T cells. In other words, we use the 8 T cell layouts from foundry technology, and scale their area accordingly to the 10 T cells' driving strength. Therefore, 10 T cells offer averaged performance-area and performance-power tradeoff points between 8 T and 12 T cells. Fig. 18 shows the delay-area and delay-power tradeoffs of 8 T, 12 T and our generated 10 T buffers and inverters. We use the methodology described in Footnote 1 to estimate cell area and delay. We estimate cell power assuming an operating frequency of 1 GHz and 10% switching activity.

Figs. 19 and 20, respectively, show performance-area Pareto curves and iso-performance comparisons between 10 T-only designs and mixed-height designs with 12 T and 8 T cells. We observe significant power and area reductions from the optimized designs with mixed heights over the 10 T-only designs. This indicates that single-height designs with an optimized performance-power/area tradeoff are not able to provide the similar performance, power, and area benefits compared to mixed cell-height designs which are able to explore a wider range of tradeoff among performance, power, and area.[20] Moreover, we perform mixed-height optimization with 8 T, 10 T, and 12 T cells as an example to demonstrate the scalability of our methodology to more than two cell heights. The black curves and dots in Figs. 19 and 20, respectively, show performance-area and performance-power tradeoffs of the optimized design with three cell heights. Results show similar design quality between the mixed-height solutions with

[19]As an example, with given input slew and output load capacitance, a 10 T 1X buffer's delay is the average of the delay of an 8 T 1X buffer and the delay of a 12 T 1X buffer with the same input slew and load capacitance.

[20]For our benchmark designs and selected clock periods, 8 T (resp. 12 T) cells typically lead to timing violations (resp. power and area overheads). We therefore consider 10 T cells to have an optimized performance-power/area tradeoff.

8 T and 12 T cells versus the solutions with 8 T, 10 T, and 12 T cells for most of the designs.[21] On design DES, adding 10 T cells reduces power and area. The slight power increase on design MPEG (frequency = 1.66 GHz) might come from the noise of SP&R tools as well as our optimization.

VI. Conclusion

In this paper, we have proposed a novel physical design optimization flow (which includes synthesis, placement, clock tree synthesis, and routing) to mix cells with different, noninteger multiple heights in a fine-grained manner within a single place-and-route block. Our flow addresses the chicken-and-egg loop between floorplan site definition and the post-placement choice of cell heights, and correctly models (based on industry feedback from 20SOC and 16FF design experience) breaker cell overheads of the mixed-height placement. Our optimization, applied to production 12 T and 8 T libraries in a 28LP foundry technology, can achieve over 30% area and power reductions while maintaining performance, as compared to a 12 T-only design flow. Moreover, our optimized mixed-height designs can achieve significant performance increase along with area and power reductions as compared to designs with 8 T-only cells.

Since there are recent studies on mixing integral multiple-height cells (e.g., [12]), one interesting future direction might be to thoroughly compare and contrast: 1) mixing of integral multiple-height cells, which has less area penalty but coarser-grained performance, power, and area tradeoff, versus 2) mixing of noninteger cell heights, which has relatively finer-grained performance, power, and area tradeoff but area cost due to breaker cells. Furthermore, to improve the scalability of our optimization flow, additional future directions might include: 1) an improved internal timer with more accurate incremental timing estimation with respect to the golden timer and 2) a parallel optimization framework to optimize different regions of a design simultaneously.

References

[1] C. J. Alpert, A. Devgan, and C. Kashyap, "A two moment RC delay metric for performance optimization," in *Proc. ISPD*, San Diego, CA, USA, 2000, pp. 73–78.
[2] R. L. S. Ching, E. F. Y. Young, K. C. K. Leung, and C. Chu, "Post-placement voltage island generation," in *Proc. ICCAD*, San Jose, CA, USA, 2006, pp. 641–646.
[3] S. Dobre, A. B. Kahng, and J. Li, "Mixed cell-height implementation for improved design quality in advanced nodes," in *Proc. ICCAD*, Austin, TX, USA, 2015, pp. 854–860.
[4] J. A. Ellis, "Embedding rectangular grids into square grids," *IEEE Trans. Comput.*, vol. 40, no. 1, pp. 46–52, Jan. 1991.
[5] F. Glover and M. Laguna, *Tabu Search*. Boston, MA, USA: Kluwer Acad., 1999.
[6] L. Guo, Y. Cai, Q. Zhou, and X. Hong, "Logic and layout aware voltage island generation for low power design," in *Proc. ASP DAC*, Yokohama, Japan, 2007, pp. 666–671.

[21] We understand that adding 10 T cells increases the layout solution space and would ideally provide more fine-grained tradeoffs of performance, power, and area, thus leading to potentially better solution quality. However, perhaps as a result of how we generate 10 T cells (i.e., by averaging the performance, power, and area of 8 T and 12 T cells), adding 10 T cells does not significantly expand the available area-delay or power-delay tradeoff (which can also be observed from Fig. 18), nor does it lead to improved design quality in our experiments. Another explanation could be the not-unexpected "limit of incremental benefit" that one would expect from additional available cell heights—similar to how added VT flavors or added cell sizes ("rich libraries") also eventually show diminishing or zero incremental returns.

[7] K. Han, A. B. Kahng, J. Lee, J. Li, and S. Nath, "A global-local optimization framework for simultaneous multi-mode multi-corner clock skew variation reduction," in *Proc. DAC*, San Francisco, CA, USA, 2015, pp. 1–6.
[8] S.-W. Hur and J. Lillis, "Mongrel: Hybrid techniques for standard cell placement," in *Proc. ICCAD*, San Jose, CA, USA, 2000, pp. 165–170.
[9] A. B. Kahng, S. Kang, H. Lee, I. L. Markov, and P. Thapar, "High-performance gate sizing with a signoff timer," in *Proc. ICCAD*, San Jose, CA, USA, 2013, pp. 450–457.
[10] A. B. Kahng, I. L. Markov, and S. Reda, "On legalization of row-based placements," in *Proc. GLSVLSI*, Boston, MA, USA, 2004, pp. 214–219.
[11] C. V. Kashyap, C. J. Alpert, F. Liu, and A. Devgan, "PERI: A technique for extending delay and slew metrics to ramp inputs," in *Proc. TAU*, Monterey, CA, USA, 2002, pp. 57–62.
[12] Y. Lin *et al.*, "MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," in *Proc. ICCAD*, Austin, TX, USA, 2016, pp. 1–8.
[13] C. W. Moon, P. Gupta, P. J. Donehue, and A. B. Kahng, "Designing a digital circuit by correlating different static timing analyzers," U.S. Patent 7 823 098, 2010.
[14] H. Wu and M. D. F. Wong, "Improving voltage assignment by outlier detection and incremental placement," in *Proc. DAC*, San Diego, CA, USA, 2007, pp. 459–464.
[15] H. Wu, I.-M. Liu, M. D. F. Wong, and Y. Wang, "Post-placement voltage island generation under performance requirement," in *Proc. ICCAD*, San Jose, CA, USA, 2005, pp. 309–316.
[16] H. Wu, M. D. F. Wong, and I.-M. Liu, "Timing-constrained and voltage-island-aware voltage assignment," in *Proc. DAC*, San Francisco, CA, USA, 2006, pp. 429–432.
[17] *OpenCores*. Accessed on Aug. 11, 2014. [Online]. Available: http://opencores.org
[18] *Design Compiler User Guide vI-2013.12-SP3*, Synopsys, Mountain View, CA, USA, 2013.
[19] *PrimeTime User Guide vH-2013.06-SP3*, Synopsys, Mountain View, CA, USA, 2013.
[20] *Innovus User Guide v16.1*, Cadence Design Syst., San Jose, CA, USA, 2016.

Sorin Adrian Dobre received the M.S. degree in microtechnology from the Faculty of Electronics and Telecommunication, Politehnica University, Bucharest, Romania.

He is a Senior Director of technology with Qualcomm Technologies, Inc., San Jose, CA, USA. His current research interests include advanced timing and power methodologies for implementation of ultralarge scale integrated system on chips, physical design and low power optimization methodologies, design for manufacturing and the development of next generation computer-aided design tools using HPC and deep learning (AI).

Andrew B. Kahng received the Ph.D. degree in computer science from the University of California at San Diego, La Jolla, CA, USA.

He is a Professor with the Computer Science Engineering Department and the Electrical and Computer Engineering Department, University of California at San Diego. His current research interests include IC physical design, the design-manufacturing interface, combinatorial optimization, and technology roadmapping.

Jiajia Li received the B.S. degree in software engineering from Shenzhen University, Shenzhen, China, in 2011, and the M.S. degree in electrical engineering from the University of California at San Diego, La Jolla, CA, USA, in 2013, where he is currently pursuing the Ph.D. degree.

He joined the Very Large Scale Integration Computer-Aided Design Laboratory, University of California at San Diego, in 2012. His current research interests include physical design and signoff optimization, margin reduction, and low-power design.