Easyn: Resynthesis and Legalization Joint NIMCH Design Placement

Jiahao Xu, Yuan Pu, Yu Zhang, *Member, IEEE*, Fangzhou Liu, Wenjian Yu, *Fellow, IEEE*, Kai-Yuan Chao, and Bei Yu, *Senior Member, IEEE*

***Abstract*— Noninteger multiple cell height (NIMCH) libraries have been adopted by advanced manufacturers in the latest technology node to offer better power efficiency, performance optimization, and area occupancy. However, in order to meet manufacturing requirements, such as power supply, NIMCH placement must comply with several additional layout restrictions. Currently, the state-of-the-art method applies resynthesis and legalization to the initial placement result to achieve a minimum displacement. However, the two decoupled phases lead to a reduction in the solution space and an insensitivity to timing. Therefore, we propose an iterative algorithm with a corresponding data structure, which comprehensively considers displacement and timing information. Experiments demonstrate that our method can generate legalized NIMCH designs with a lower maximum delay and a reduced area.**

***Index Terms*— Circuits and systems, digital circuits, electronic design automation and methodology, design for manufacture, design for quality.**

I. Introduction

With advancements in VLSI technology nodes, noninteger multiple cell height (NIMCH) designs, which refer to layout methodologies where the cells in a circuit do not necessarily have the same heights, are getting more attention.

The larger cell height in design usually means better performance (e.g., pin accessibility and shorter delay) and higher cost (e.g., area and power consumption) for a single cell [1]. In traditional VLSI design, all cell heights are typically standardized as integer multiples of a single-row height to simplify the manufacturing process and ensure compatibility across different libraries and tools. However, this approach leads to area and power waste since cells on noncritical paths incur unnecessary expense. In contrast, using noninteger multiples for cell heights enables more flexible and efficient use of the chip area, potentially resulting in smaller area and lower power consumption. Dobre et al. [2] implemented a method to place designs with multiple heights, demonstrating that mixed-height designs achieve area and power reductions while maintaining performance compared to 8T or 12T-only designs. Moreover, TSMC, on the manufacturing side, has launched its NIMCH FinFlex technology to provide critical design flexibility for better power efficiency and performance optimization at the 3-nm node [3].

However, NIMCH designs also introduce complexity in terms of placement, routing, and manufacturing. For example, NIMCH layout constraints require that all cells in the same subblock have the same height, while subblocks of different heights must be separated by additional space. This requires sophisticated design tools and methodologies to handle the variations in cell heights effectively. Dobre et al. [2] presented the first work implementing NIMCH design at the subblock or subisland level. Zhao et al. [4] also produced an island-based solution by replacing and clustering higher cells based on the initial solution generated by a commercial tool, including a stage of revising the block range to ensure solution legality. Lin and Chang [5] explored row-based legalization to narrow the solution space and achieve smaller displacement and shorter delay.

These works, along with [6], [7], [8], [9], and [10], focus on the legalization process after the initial placement generated by a conventional tool, which may mix cells of different heights in the same row or subblock. However, this design flow could cause significant cell displacement to meet NIMCH constraints in the legalization phase, thus significantly extending the potential critical path length. Therefore, Pu et al. [11] implemented an additional phase of placement-aware logic resynthesis between initial placement and legalization. This method resynthesizes the circuit by modifying the gates (and thus the heights) assigned to the nodes based on the initial placement result. This procedure aims to regularize the spatial distribution of cell heights to align with NIMCH constraints, leading to smaller displacement and shorter maximum delay in the following legalization phase. Fig. 1 illustrates an example that resynthesis reduces the critical path detour. Fig. 1(a) illustrates a situation where direct legalization causes a significant detour. In Fig. 1(b), the resynthesis process is decoupled from legalization, so some timing-aware reassignments still cause great displacement after legalization. In Fig. 1(c), we illustrate the motivation of Easyn that by coupling resynthesis and legalization stages, we can assign high-performance gates to the critical paths as much as possible while trying to preserve the shape of the critical paths.

Received 12 June 2025; revised 27 September 2025 and 7 December 2025; accepted 21 December 2025. Date of publication 29 December 2025; date of current version 25 August 2026. This work was supported in part by the Research Grants Council of Hong Kong SAR under Grant CUHK14211824 and Grant CUHK14210723 and in part by the MIND Project under Grant MINDXZ202404. This article was recommended by Associate Editor A. Teman. *(Corresponding author: Bei Yu.)*

Jiahao Xu, Yuan Pu, Yu Zhang, Fangzhou Liu, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Wenjian Yu is with the Department of Computer Science and Technology, BNRist, State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University, Beijing 100084, China.

Kai-Yuan Chao is with Siemens Digital Industries Software, Willisonville, OR 97070 USA.

Digital Object Identifier 10.1109/TCAD.2025.3649428

1937-4151 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

Prior to the widespread adoption of NIMCH technology, resynthesis methods had already been extensively researched. The gate sizing and circuit restructuring techniques utilized in Lesyn [11] were implemented across various paradigms. Heuristic sensitivity-based methods [12], [13], for instance, assess individual gate sensitivity using predefined knowledge or physical-aware information, subsequently adapting gate sizes based on these sensitivity rankings. While these heuristics are efficient and yield considerable optimization on many benchmarks, concerns regarding their stability remain. Lagrangian relaxation [14], [15], [16] is a different way of optimization. They resorted to mathematically restricting the solution space. However, the core purpose is still to heuristically optimize the sensitivity function. By employing the Karush–Kuhn–Tucker (KKT) conditions, the search space is greatly constrained. However, it increases runtime and still produces suboptimal solutions. Dynamic programming (DP)-based methods calculate the optimal substructures related to delay and power at all the gates [17], [18] and determine the gate in backtracing. However, to achieve reasonable runtime, the DP-based methods have to apply suboptimal substructures or require constraints on the problem setting. This prevents DP methods from finding the optimal solution. Liang et al. [19] proposed a Monte Carlo-based cell version mitigation. This method focuses on alleviating spatial congestion in the legalization process. Learning-based methods [20], [21], [22] explored the possibilities of applying machine learning and generative AI to predict the value of gate sizing operations on each gate. However, the large amount of training time and the scalability to other libraries and designs undermine the implementation of learning-based methods. Recently, the flow that fuses resynthesis and placement phases was explored by Du et al. [23]. This flow gives an insight into leveraging physical information in a more effective way.

In the flow of Lesyn, the resynthesis and legalization phases are completely decoupled. The DP process is not aware of the congestion of the adjacent rows of certain heights. The legalization algorithm does not adjust gate assignments to reduce displacement. This further reduces the solution space. In addition, its DP strategy does not make sufficient use of the spatial and timing information in the initial placement.

To overcome these concerns and leverage the advantages in both phases, we propose a new NIMCH design resynthesis and legalization framework: Easyn. Specifically, this framework begins with the initial placement result and implements an iterative procedure to optimize several critical metrics (delay, area, displacement, and so on). In each iteration, a DP process first traverses the circuit in topological order to collect necessary information. Then, in the DP traceback process, each node is assigned an appropriate gate and position under the guidance of a placer. The heights of these cells can be modified simultaneously. The expected resource usage and delay requirements will be adapted to achieve a better balance between cost and performance in the next iteration. Finally, with the cooperation of the DP engine and the placer, a legalized solution is produced, which is supposed to have smaller displacements and a better tradeoff between area and delay. We summarize our major contributions as follows.

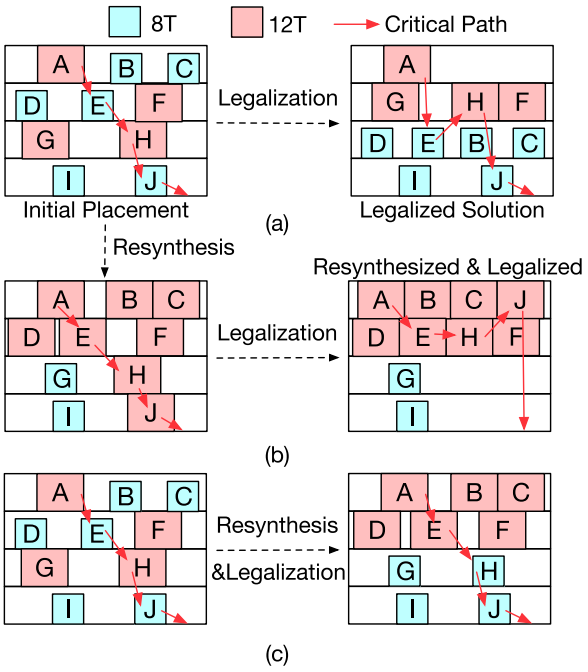


Fig. 1. NIMCH legalization with resynthesis. (a) Legalizing initial placement can cause an obvious detour. (b) Some resynthesized cells are still greatly displaced. (c) Coupling resynthesis and legalization can mitigate this effect.

1) We propose an open-sourced framework for the resynthesis and legalization of NIMCH circuit design.
2) We develop an incremental row-based placer to cooperate with the resynthesis procedure, which allows placing cells not necessarily in coordinate order.
3) We propose an iterative flow to achieve a better balance between area and delay.
4) Experimental results demonstrate that under the same area constraint, our method achieves 18% delay reduction.

II. Preliminaries

A. NIMCH Logic Resynthesis

A Boolean network can be represented by a directed acyclic graph (DAG). Each node is bound with a gate in the gate library. After placement, all the nodes, represented by cells, are placed without overlapping in the range of a block. A cut c of a node n is a set of nodes in the transitive fan-in of n such that every path from a PI to n passes through at least one node in c. Each cut c is associated with a truth table representing the logic function of n in terms of the leaves. A library gate g matches a cut c if their truth tables are identical. The resynthesis algorithm in Lesyn [11] begins by standardizing cell heights, primarily by unifying the heights of noncritical cells across most rows. A DP algorithm then traverses the circuit in topological order. For each node n and candidate gate g, it calculates the minimum delay, taking into account the gates g_i assigned to cut nodes $c_i \in c$. In the subsequent traceback stage, the algorithm assigns each node n the gate that minimizes area while satisfying the delay constraints. Notably, for any node n_i, the newly selected gate may utilize a different fan-in set, provided that its truth table

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

preserves the original logic function. While this resynthesis modifies gate assignments and connectivity without altering functionality, the resulting circuit may still violate NIMCH placement constraints, necessitating further legalization.

B. NIMCH Legalization Methods

Dobre et al. [2] first designed an island-based flow for the NIMCH placement problem to resolve the "chicken-and-egg" loop between floorplan and cell height selection. Initially, all cells are standardized to a uniform height, enabling the use of conventional placement tools for generating an initial placement. Subsequently, the design space is divided into subblocks, with each subblock assigned a specific height. The cells are then resized and replaced to ensure uniform height within each block while minimizing the cost defined by area, power, delay, and so on of the whole legalization process.

Though the experimental results in [2] showed a great area and power saving while preserving the delay performance, compared to 8T or 12T-only designs, there are still concerns such that the process above leads to a too large solution space and a large displacement along with the wirelength. Therefore, Lin and Chang [5] proposed a row-based algorithm to reduce the solution space and cut down displacement, which proved to be both more effective and efficient.

The row-based legalization approach also starts with unifying all cell heights and generating an initial placement using a commercial tool. A k-means clustering algorithm is then employed to determine the optimal height for each row, taking into account the original cell heights. Following this, cells are sorted and placed using the Abacus method [24], with each cell being assigned exclusively to a row matching its height requirement. Finally, the cell heights that have been unified are changed back to their original states, and the floorplan is updated accordingly.

C. Problem Formulation

In the previous works, the NMICH placement constraints have been defined formally. We apply the following essential constraints in our legalization process to keep the problem concise without losing generality [2], [4].

- C_1: To honor the well-to-well spacing rule, two cells of different heights must be separated by at least a specific horizontal distance.
- C_2: To avoid P/G rail encroachment, a minimum vertical distance is also in need between two cells of different heights.
- C_3: To ensure manufacturability, each region of a particular cell height must have at least two cell rows.
- C_4: Due to N-well sharing, each region of a particular cell height must have an even number of rows.

Since this work focuses on row-based legalization, we do not have to consider the horizontal constraints as [2] stated.

We formally define the NIMCH resynthesis and placement problem as follows.

Problem 1 (NIMCH Resynthesis and Placement): Given a circuit C with n gates, a gate library L, and an initial placement result R_0, resynthesize the cells in C and obtain a placement solution that satisfies constraints C_1–C_4. The functionality of the circuit should not be changed. The delay of the design should be minimized by the reassignment of gates and the placement solution. Since there is a significant tradeoff between area and delay, the total area should be constrained.

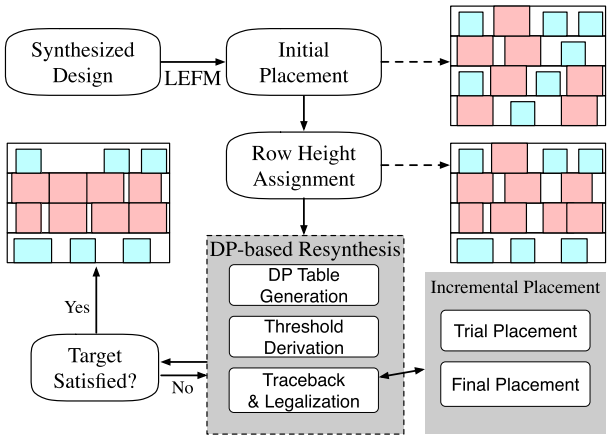


Fig. 2. Overall workflow of fusing resynthesis and legalization placement to leverage physical information. The slack/tighten loop is iteratively performed.

III. Methodology

In this section, we propose our new NIMCH resynthesis and placement joint working flow. The flow considers the constraints as defined in Section II-C and obtains a legalized solution with high PPA quality. We now describe the overall optimization flow as illustrated in Fig. 2.

First, before our algorithm starts, we generate an initial placement result R_0 with innovus, as mentioned in Lesyn [11]. This process does not involve NIMCH placement constraints, so most rows are placed with mixed heights.

Based on the initial placement R_0 and the corresponding timing report, a row height assignment algorithm is applied to roughly modify the cell heights and determine the height of each row, which will be further discussed in Section III-A.

Subsequently, the resynthesis and placement workflow is executed iteratively. In each iteration, the mode is determined by the current total area: if it exceeds the specified maximum, the iteration is designated as a "slack" phase; otherwise, it is a "tighten" phase. Each round initiates with a topological DP algorithm, detailed in Section III-B, which generates a delay-gate table used for the subsequent threshold determination. Following this, a traceback step assigns the most appropriate gate to each node, aiming to balance area consumption, displacement, and delay requirements. This assignment strategy relies on the integration of the DP results with our incremental placer, introduced in Section III-C.

After the back tracing, we obtain a legalized solution. If another iteration is required, the parameters will be modified based on the process of this round, which will be demonstrated in Section III-D. If the total area or WNS requirement is met, we update the floorplan based on the final resynthesis and ensure the NIMCH placement constraints are satisfied. Our solution is then evaluated by innovus.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

A. Row Height Assignment

Given the initial placement R_0, we apply a rough row height assignment strategy to adjust the distribution of cell heights to the row-based subregion pattern. To comply with the NIMCH constraints C_3, C_4, we form every two adjacent rows as a pair (e.g., rows $2n$ are paired with $2n+1$). For each pair of rows rows_i, denote the set of cells of 8T/12T height as $C_{i,8T}/C_{i,12T}$ and the ratio of them as

$$\text{ratio}_i = \frac{|C_{i,8T}|}{|C_{i,12T}| + |C_{i,8T}|}. \tag{1}$$

We define two hyperparameters α and β, which by default are set to $\alpha = 0.2, \beta = 0.8$. For all the rows_i with

$$\alpha \leq \text{ratio}_i \leq \beta. \tag{2}$$

We collect them in a set S and sort S in the ratio_i order. In S, the rows with $\text{ratio}_i \geq \delta$ ($\text{ratio}_i < \delta$) will be assigned as 8T (12T) for the entire row. Here, δ is adjusted to make the total area approximately reach the original area. This is a bit counterintuitive to the common sense that we should and only should unify the heights in the rows with very high (low) ratio_i. However, according to our observations in the experiments, a very high (low) ratio_i means that only a few illegal cells should be moved to other rows in legalization, while a row with evenly distributed height causes great displacement and detour of routing if the illegal cells in this row are moved to adjacent rows. Therefore, we prefer to unify the heights of the cells in a row with evenly distributed cell heights ($\alpha \leq \text{ratio}_i \leq \beta$).

After unifying the heights of the cells in these rows, we refer to the slack report to modify the heights of some cells again. Since the unifying height operation can assign 8T cells to a lot of cells on the critical path, we refer to the slack report and change those 8T cells that have a higher delay than our required value back to 12T. At the same time, we change 12T cells with the most slack to 8T to keep the total area unchanged.

After the above cell heights reassignment process, each cell is assigned a new height, denoted as CH_c for each cell c, and the distribution of cell height generally presents a row-based subregion distribution. Then, we apply a heuristic algorithm to determine the height of each row, to assign a height RH_i for each row row_i. In [5], k-means was applied to assign row heights. However, we notice that k-means tends to create very large and compact clusters, failing to leave 12T (8T) rows in some fairly large ranges. Therefore, we apply a heuristic method to create a more evenly distributed row height assignment to reduce displacement.

1) $\text{RH}_i = 12T$, for those rows_i with $\text{ratio}_i \leq \alpha$.
2) Traverse all unassigned rows from bottom to top, and sum up the width of 12T cells.
3) When the total width summed exceeds a threshold (by default, $2\beta\times$ width of row), we assign the pair of rows at the center of these accumulated rows to 12T. The center is defined as the center of gravity using the width of cell as weight. If a row in the pair is already 12T, select the nearest pair of rows.
4) Clear the summed width and repeat 2–4 to find the next pair of 12T rows.

Note that to comply with the constraints C_3, C_4, a pair of rows is always assigned with the same height. The other rows are automatically assigned as 8T. Now, each row may still have cells of mixed heights. If the height of a cell is changed in this stage, the gate assignment in the original design cannot be used as the initial status for the DP process. We will assign it with a gate of this changed height with minimal area as its original gate. In the following iteration process, the gates corresponding to some cells may be further modified, and the final position of each cell will be ensured to meet the constraints.

B. DP-Based Resynthesis

In this section, we propose a DP-based resynthesis algorithm to remodel the circuit by assigning appropriate gates to each node. This process may involve changing the topology structure of the circuit but should preserve the logic function.

Before identifying the DP table structure, we need to define our global optimization target. We can formulate it in several ways.

1) $\min \sum_{n=1}^{N} A_{g_n}$, such that $\text{Delay} \leq T$.
2) $\min \text{Delay}$, such that $\sum_{n=1}^{N} A_{g_n} \leq A$.
3) $\min D$, such that $\text{CH}_n = \text{RH}_{y_n}$.

Here, delay represents the maximal delay from any primary inputs (PIs) to any primary outputs (POs) of the circuit. T is the target clock period we want to meet. A_{g_n} is the area of gate g_n assigned to n, and A is the target total area that we should not exceed. D is the total displacement that is used to estimate the possible routing detour. CH_n and RH_{y_n} mean that the cell height of n must match the height of the row where it is to be placed in.

The area and delay always have a tradeoff, so our optimization target should be achieving minimal one side without exceeding the limit of the other side. This corresponds to 1) and 2) and is mainly handled by the DP algorithm. The displacement is more difficult to control and has a more complex impact on the final routing result. We mainly handle it through communication and collaboration between the DP process and the placement process, which will be covered in depth in Section III-C.

Lesyn [11] introduced a DP-based resynthesis technique, where the DP entry stores the cut status of the nodes and generates a delay–area Pareto frontier. However, it is impossible to save the complete information from the input of the circuit to the current node. Therefore, the "area" metric there only considers the area of the current node and its cut (i.e., the sum of the area of the gate assigned to the current node and the adjacent fan-in gates). This design requires a large amount of time and space to enumerate the combinations of the cut gates while still providing a suboptimal solution.

Therefore, we propose a DP table formulation with the corresponding back tracing strategy. The results of the DP table are mainly used to provide reference information for subsequent traceback and placement strategies and greatly

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

save the calculation time itself. Specifically, the DP table is formulated as follows.

Definition 1 (DP Table Formulation): For each node n, there is a set G_n of available gates selected from the gate library L. The entry $\mathrm{DP}_{n,g}$ represents the delay at n when a gate $g \in G_n$ is assigned to n (assuming that the gates from input to n remain unchanged). Slack ratio table $\mathrm{ST}_{n,g}$ is the ratio between the reduced delay and the extra area. Specifically, the following conditions hold.

1) $\mathrm{ST}_{n,g} = \dfrac{\mathrm{DP}_{n,g} - \mathrm{DP}_{n,o}}{A_o - A_g}$, in slack iteration.
2) $\mathrm{ST}_{n,g} = \dfrac{\mathrm{DP}_{n,g} - \mathrm{DP}_{n,o}}{A_g - A_o}$, in tighten iteration.

Here, o denotes the original gate of n. $\mathrm{CP}_{n,g}$ counts the number of critical paths that pass n. Specifically, $\mathrm{CP}_{n,g,\mathrm{forward}}$ and $\mathrm{CP}_{n,g,\mathrm{back}}$ represent the number of critical path from PI to n and n to PO, respectively. Their product $\mathrm{CP}_{n,g}$ is the number of critical paths through n.

DP and ST can be generated by going through the circuit once in topological order, while $\mathrm{CP}_{n,g}$ needs to be calculated separately by traversing the graph both forward and backward. We define $G_n \subseteq L$ as the set of valid candidate gates for node n. A gate g is included in G_n if and only if there exists a cut c of n such that: 1) g functionally matches c (preserving circuit logic) and 2) since a large change of circuit connectivity may cause a large extra wire length and thus bad timing, if the connectivity of the circuit is changed, $\forall c_i \in c$, c_i is constrained to be spatially close to n.

DP Table Generation. Given the initial placement result and the row height assignment result, we initiate the DP table in topological order. For n that is not a PI, the set G_n is calculated by enumerating gates satisfying the constraints following Definition 1. Then, we go through the circuit as demonstrated in Algorithm 1.

Algorithm 1 DP Table Generation

Require: Circuit C, gate library L, candidate gate sets G_n
Ensure: Delay table $DP_{n,g}$, slack ratio table $ST_{n,g}$

```
1:  Record the required time RT_n of all the nodes;
2:  Set up a sequence Q of nodes sorted in topological order;
3:  for n in Q do
4:      for g in G_n do
5:          c ← cut that matches g;
6:          Remap n to g in the timer;
7:          for c_i in c do
8:              delay_i ← DP_{c_i,AG_{c_i}} + d(c_i, n);
9:              DP_{n,g} ← max(DP_{n,g}, delay_i);
10:         end for
11:         Update ST_{n,g};
12:     end for
13: end for
```

Before the DAG is traversed in topological order, we record the required time RT_n at each n with the original gate assignments for all nodes. In lines 3–5, for each n, we enumerate $g \in G_n$. A g can have a different cut c from the current cut modeled in the timer. (This means a change in circuit structure.) If this happens, the RC tree should be locally remodeled in the timer to generate $d(c_i, n)$, the delay from each cut node to n. Note that in order to save runtime, we assume that only the gate assignment of n is modified and all other gates remain unchanged when calculating $d(c_i, n)$. In line 6, the gate assignment status of n is updated in the timer. From line 7 to line 10, we enumerate $c_i \in c$ and generate the delay at n by

$$DP_{n,g} = \max_{c_i \in c} DP_{c_i,\mathrm{AG}_{c_i}} + d\,(c_i, n)\,. \tag{3}$$

$\mathrm{ST}_{n,g}$ is then updated based on the equation in Definition 1. However, only a subset of (n, g) entries in ST will be considered for gate replacement in DP traceback. First, the original gate is skipped. Then, if it is a slack iteration, we require $\mathrm{DP}_{n,g} \le \mathrm{RT}_n$ and $A_g \le A_o$. If it is a tighten iteration, we require $\mathrm{DP}_{n,g} \le \mathrm{DP}_{n,o}$. In both cases, the smaller the value of $\mathrm{ST}_{n,g}$, the more effectively the area is used.

In this process, we also calculate $\mathrm{CP}_{n,g,\mathrm{forward}}$. We have

$$CP_{n,g} = \begin{cases} \displaystyle\sum_{c_i \in c} CP_{c_i,\mathrm{AG}_{c_i}}, & \text{if } DP_{n,g} > RT_n \\ 0, & \text{if } DP_{n,g} \le RT_n \end{cases} \tag{4}$$

where AG_{c_i} stands for the current gate assigned to c_i and c is the cut corresponding to g. Here, $\mathrm{CP}_{n,g,\mathrm{forward}}$ and $\mathrm{CP}_{n,g,\mathrm{back}}$ can be calculated in the opposite direction by (4). In the "back" direction, c is changed to the set of fan-out nodes of n.

For clarity, the flow in which the calculation results from the DP process and the placement process interact is shown in Fig. 3. Starting from the original cell position X and Y and gate assignment AG (the gray rectangle), DP and RT are first generated in the DP Table generation process. Then, ST and CP are derived as introduced above; 1) and 2) both take place in the traceback process, which will be explained in Algorithm 2. Finally, the legalized cell positions X and Y and the newly assigned gates AG are updated as the solution of this iteration.

$\mathrm{ST}_{n,g}$ and $\mathrm{CP}_{n,g}$ will be used as a reference to identify a slack (tighten) threshold TH for the following traceback and placement phase. To identify this threshold, we set a score for each pair of (n, g). This score is the sum of the percentiles of $\mathrm{ST}_{n,g}$ and $\mathrm{CP}_{n,g}$ in all pairs, where $\mathrm{ST}_{n,g}$ is sorted in ascending order and $\mathrm{CP}_{n,g}$ is sorted in descending order. For example, if $\mathrm{ST}_{n,g}$ is the 5% smallest and $\mathrm{CP}_{n,g}$ is the 3% largest, then the score for (n, g) is 8.

We sort this score $\mathrm{score}_{n,g}$ in ascending order and accumulate the area reduction (consumption). Once the accumulated change of area exceeds $\gamma\times$ total area, we set TH as the current score value. Here, γ is a ratio that changes as the number of iterations increases, which will be discussed in Section III-D.

Generally, this slack/tighten threshold TH helps us determine the most effective resynthesis choices. This threshold works like the other heuristic sensitivity-based methods. Since the calculation of the critical path is affected by the legalization process, we do not need to pursue the exact delay calculation at this stage, but choose to use this estimation method as an alternative.

The scale of DP entry for all n, g is $O(n * \mathrm{size}_G)$, where size_G denotes the average size of available gate set of all

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

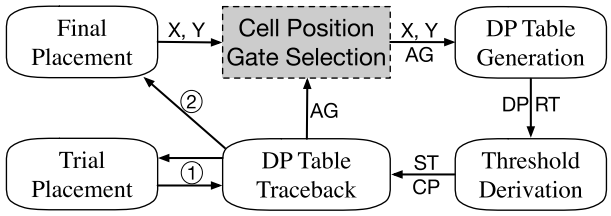


Fig. 3. Detailed flow of the DP-based resynthesis.

nodes. Remodeling the RC tree involves nodes connected to n, but this step is rarely triggered since structural change hardly happens. For each pair of (n, g), enumerating the entries of the cut has the scale of $|c| * \text{size}_G$. Here, size_G and $|c|$ are small constants, so the DP table generation takes $O(n)$ time. Note that we modified OpenTimer to make it support local updates, which reduced the total time consumption.

DP Traceback and Placement. In this section, we describe the traceback procedure for the DP table, which selects the most appropriate gate for each node. This process is guided by both the DP solution, which considers delay and area, and the placement information provided by the placer. Gate assignments and their legalized positions are determined simultaneously, yielding a legalized layout immediately upon completion of the traceback. The placer, treated as a callable module in this context, is detailed in Section III-C. The full traceback procedure is outlined in Algorithm 2.

Algorithm 2 DP Solution Traceback

Require: Circuit C, DP table $DP_{n,g}$
Ensure: Assigned gates AG_n, modified placement information
1: Initialize the placer P;
2: Set up a sequence Q of nodes in reverse-topological order;
3: Initialize the required delay RT_n at each PO;
4: **for** n in Q **do**
5: $g* \leftarrow AG_n$, $cost \leftarrow INF$;
6: **for** g in G_n **do**
7: **if** `select`(n, g, DP, ST, RT, TH) **then**
8: **if** P.`tryPlace`$(n, g) < cost$ **then** ▷ Algorithm 4
9: $g* \leftarrow g$;
10: $cost \leftarrow P$.`tryPlace`(n, g);
11: **end if**
12: **end if**
13: **end for**
14: P.`finalPlace`$(n, g*)$;
15: $area_slack \leftarrow area_slack - area_{g*} + area_{AG_n}$;
16: $AG_n \leftarrow g*$;
17: Remap n to AG_n in the timer;
18: **for** $c_i \in c$ **do**
19: $RT_{c_i} \leftarrow \min(RT_{c_i}, DP_{n,g*} - \text{delay}(c_i, n))$;
20: **end for**
21: **end for**

Starting from each PO, we traverse the circuit following the reverse topological order. The placer is initialized since it guides the resynthesis and generates the position of each cell dynamically. AG_n represents the gate assigned to a single node. It is initially the gate assigned to n in the input design or preprocessing. It varies as the iterations proceed.

The function `select` verifies several properties of gate n. Specifically, if this is a slack iteration, then select is true.

1) $\text{DP}_{n,g} \leq \text{RT}_n$.
2) $\text{score}_{n,g} \leq \text{TH}$.

And if this is a tight iteration, select is true.

1) $\text{DP}_{n,\text{AG}_n} > \text{RT}_n$.
2) $A_g - A_{\text{AG}_n} \leq \text{area_slack}$.
3) $\text{score}_{n,g} \leq \text{TH}$.

Here, area_slack is the current difference between the total area and the limitation of area. P.`tryPlace`(n, g) calculates the minimum possible displacement if g is assigned to n. By cooperating with the Placer, we ensure that the displacement caused by the change in cell height is well controlled. If a gate must be placed in a very remote row, the placer will simply output INF and g will not be considered then. If no candidate g* is updated in this process, the original gate AG_n is kept. More details will be introduced in Section III-C.

After the gate assigned to n is determined, we place it immediately in the placer (line 14), because this will influence the displacement of the cells to be placed later. This final placement operation is similar to Algorithm 4 and will also be introduced in Section III-C. The corresponding variables and the status in timer are then updated (lines 15–17). The required time RT of the fan-in nodes is renewed depending on the current gate assignment (lines 18–19). This is actually another DP process in the reverse topological order.

Handling Placement Failure. In the above DP traceback and placement process, we notice that placement failure can happen if the following conditions are met.

1) All the g values that satisfy select in Algorithm 2 get INF in trial placement.
2) The original gate AG_n has a different height from the original row of n and also gets INF in trial placement.

In this situation, we simply assign a gate with the same height as the original row of n so that n can be placed nearby. During the experiments, thanks to our relatively uniform row height assignment, this situation almost never occurs.

C. Incremental Placement

In this section, we propose an incremental placer based on the balanced binary tree. This placer can insert a cell at any position (instead of the right end) of any row with logarithmic complexity.

In [24], a heuristic legalization algorithm was proposed. The cells are first sorted by their x coordinates in the initial placement. After that, for each cell, it will be placed at the rightmost end of a nearby row that causes the smallest displacement. This method proves to be effective in controlling costs and has good efficiency. In [5], Abacus was modified and used to finally legalize cells considering their heights.

However, this kind of method cannot meet our needs. This is because, in order to control the amortized complexity of each placement, Abacus requires sorting the initial placement results first. However, in our scenario, we hope to place the

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

TABLE I
TERMINOLOGY AND NOTATION FOR CALCULATION IN THE INCREMENTAL PLACEMENT PROCESS

Variable	Definition	Variable	Definition
x_i	Optimal x of cell i	x'_i	Original x of cell i
e_i	Weight of cell i	w_i	Width of cell i
D_R	Displacement in row R	D_L	Displacement in cluster L
S_{ex}	$\sum_i e_i x'_i$	M_{ew}	$\sum_i e_i \sum_{k=1}^{i-1} w_k$
S_e	$\sum_i e_i$	S_w	$\sum_i w_i$
S_{ex2}	$\sum_i e_i x'^2_i$	S_{exw}	$\sum_i e_i x'_i \sum_{k=1}^{i-1} w_k$
S_{ew2}	$\sum_i e_i (\sum_{k=1}^{i-1} w_k)^2$	N_L	The number of cells in cluster L

current cell immediately during the DP traceback process, in order to provide more accurate displacement information for the subsequent placement process. Therefore, we propose an incremental placer that supports immediate placement in the DP traceback process.

We first derive the formulas for calculating the optimal placement of the clusters. Next, we support inserting a single cell into any row (final placement) by maintaining the formulas on a balanced tree. Finally, we introduce how to estimate the cost of inserting a cell using trial placement.

Incremental Placement in a Cluster. We first introduce our algorithm that places a cell in any position of a single row with logarithmic complexity. This method involves calculating the optimal position on a balanced tree.

In Abacus, the horizontal displacement of the cells in one row is defined as

$$\min \ D_R = \sum_{i=1}^{N_R} e_i \left(x_i - x'_i\right)^2 \tag{5}$$

where D_R is the total displacement of cells in row R, N_R is the number of cells in row R, and i enumerates all the cells in R. The meaning of the variables that appear in the following is listed in Table I.

For a cluster of cells within a row, the cells must abut to ensure a nonoverlapping placement with minimal displacement. In the following analysis, we focus on deriving the positions within each cluster. We define indices i and j to enumerate the cells in the cluster, sorted by their original x-coordinates. The positions of these cells must satisfy [24]

$$x_i = x_1 + \sum_{j=1}^{i-1} w_j. \tag{6}$$

Therefore, the optimal position of the cluster can be calculated by solving

$$\sum_{i=1}^{N_L} e_i x_1 = \sum_{i=1}^{N_L} e_i \left(x'_i - \sum_{j=1}^{i-1} w_j\right). \tag{7}$$

The authors further simplified the formula into a form that can be recursively calculated when a cell is appended to the right end of a cluster. This makes the operation of appending cells have amortized linear complexity.

However, as mentioned above, this calculation requires that the cells be sorted first. We modify the recursive formula so that it can be merged and split on a balanced tree to achieve out-of-order insertion of cells.

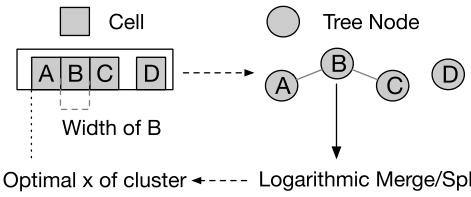


Fig. 4. Adjacent cells are merged into a cluster. Each cluster is formed as a balanced tree to achieve logarithmic merge and split.

In our program, each cluster of cells is arranged as a balanced binary tree. As illustrated in Fig. 4, each cell becomes a node on the tree, with its original x-coordinate serving as the key. In order to adapt the calculation of optimal position and displacement to the recursive process on the balanced tree, we can derive the following:

$$x_1 = \frac{S_{\text{ex}} - M_{\text{ew}}}{S_e} \tag{8}$$

$$D_L = S_{\text{ex2}} - 2S_{\text{ex}}x_1 - 2S_{\text{exw}} + 2x_1 M_{\text{ew}} + x_1^2 S_e + S_{\text{ew2}} \tag{9}$$

where x_1 and D_L are the optimal left end coordinate and the total displacement of the cells in cluster L, respectively. Other intermediate values can be calculated as the operations in the balanced tree proceed (merge, split, and so on). At node n with two sons, left child l and right child r, these intermediate values can be merged in O(1) time. As a result, we have

$$S_{\text{ex}}(n) = S_{\text{ex}}(l) + S_{\text{ex}}(r) + e_n x'_i \tag{10}$$

$$M_{\text{ew}}(n) = M_{\text{ew}}(l) + M_{\text{ew}}(r) + (S_e(r) + e_n) S_w(l) + S_e(r) w_n \tag{11}$$

$$S_{\text{ex2}}(n) = S_{\text{ex2}}(l) + S_{\text{ex2}}(r) + e_n x'^2_n \tag{12}$$

$$S_{\text{exw}}(n) = S_{\text{exw}}(l) + S_{\text{exw}}(r) + \left(S_{\text{ex}}(r) + e_n x'_n\right) S_w(l) + S_{\text{ex}}(r) w_n \tag{13}$$

$$S_{\text{ew2}}(n) = S_{\text{ew2}}(l) + e_n S_w(l)^2 + S_{\text{ew2}}(r) + S_e(r)(S_w(l) + w_n)^2 + 2(S_w(l) + w_n) M_{\text{ew}}(r). \tag{14}$$

The basic definitions of the notations above are also listed in Table I. Notations assigned with n, l, and r limit the calculation scope to a node and its subtree. For example, $S_e(n)$ sums the e_i in the subtree of n (including n).

By maintaining the intermediate values on the balanced tree, we can insert a cell in any cluster with amortized logarithmic complexity. We can immediately calculate x_1 and D_L of a cluster using the intermediate values at the root node.

Incremental Placement in a Single Row. With the above data structure, we can perform the placement in any single row efficiently as described in Algorithm 3. This function places a cell c in a row R and returns the corresponding displacement. `Ind` indicates whether this insertion should be revoked after calculating the cost. For clarity, we first assume that this insertion takes place in a final placement and will not be revoked.

It should be noted that all cells always maintain the same order as their initial x-coordinates in ascending order, regardless of the order in which they were placed. Therefore, when

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

a cell c is placed in a row R, we can efficiently search if there is a cluster that overlaps with it (line 3).

Algorithm 3 Place a Cell in a Row

```
1:  function PLACECELL(Row index R, Cell c, Ind)
2:      d ← 0;
3:      if ∃cluster L s.t. x_c ≤ x_L + w_L & x_c + w_c ≥ x_L then
4:          rank ← The rank of c in L; d ← d − D_L;
5:          Split(L, rank, root_1, root_2);
6:          L ← Merge(root_1, c);
7:          L ← Merge(c, root_2);
8:          d ← d + D_L;
9:          while True do
10:             L_l ← The cluster on the left side of L;
11:             if x_L ≤ x_{L_l} + w_{L_l} then
12:                 d ← d − D_{L_l} − D_L;
13:                 L ← Merge(L_l, L);
14:                 d ← d + D_L;
15:             end if
16:             L_r ← The cluster on the right side of L;
17:             if x_L + w_L ≥ x_{L_r} then
18:                 d ← d − D_{L_l} − D_L;
19:                 L ← Merge(L_l, L);
20:                 d ← d + D_L;
21:             end if
22:             Break if L_l and L_r are not merged;
23:         end while
24:     else
25:         c forms a new cluster at its original position;
26:     end if
27:     Return d/size(L);
28: end function
```

If such a cluster L exists, we split the cluster (the balanced tree of it) and insert the cell (lines 5–7). The ranks of the cells are decided based on their original x-coordinate to produce the optimal displacement, sorted in ascending order in the horizontal direction. Here, the Split operation means dividing the cluster L into L_1 and L_2. The size of L_1 is rank. The rest cells of L are in L_2. The Merge operation fuses two clusters. Both are basic operations of balanced tree with amortized logarithmic complexity, and the intermediate values we want will be maintained in this process. Any balanced tree implementation can work, such as Splay [25].

In lines 9–23, the program will continuously check whether the current cluster overlaps with the clusters on the left and right. If so, the two clusters will be merged on the balanced tree. The optimal placement of this new cluster will also be calculated and checked again until it does not overlap with other clusters. Finally, we return the estimated change in displacement. It is scaled by the size of the cluster since we want to spread the cost evenly across each cell in the cluster; otherwise, a cluster will be easily rejected only because it is large.

Fig. 5 illustrates an example of incremental placement. Consider an iteration where cell C is converted to a 12T height and relocated to the nearest 12T row. At this stage, cells A, B, and D have been placed at their original positions, with B and D already forming a cluster. When the placer inserts the 12T cell C between B and D, it identifies that a new cluster is formed. Based on the sort order of their original x-coordinates, the placer first splits the existing B and D cluster and merges them with C. As this new cluster overlaps with A, A is also merged into the cluster. The total displacement and the optimal x-coordinate for the entire cluster are calculated using the balanced tree structure. Since the process relies solely on the tree's merge and split operations, the computational complexity remains logarithmic relative to the number of cells involved.

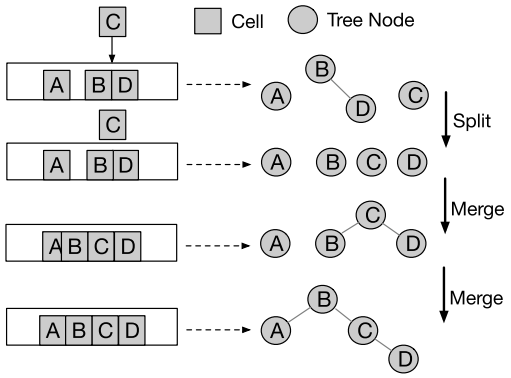


Fig. 5. Incremental placement process. When a cell is placed in a certain row and conflicts with an existing cluster, the cluster is split in the balanced tree representation. The cell is then inserted in the appropriate position and merged into a cluster with adjacent cells.

If Ind indicates that `placeCell` is triggered in a trial placement, the algorithm should be modified. This modified `placeCell` still inserts c in R. However, we record the specific places where c is merged in every corresponding cluster and where these clusters are split. After calculating the displacement change caused by this trial operation, the whole process will be revoked. Hence, the global placement status is not changed. To save space, we did not describe this undo process in Algorithm 3.

Algorithm 3 allows us to insert a cell at any position in any row and efficiently calculate its optimal position and displacement. However, to cooperate with the resynthesis DP traceback process, we need a multirow placement strategy, i.e., the trial placement algorithm for cost estimation, and the final placement operation to modify the global placement state.

Trial placement. In Algorithm 2, we mainly use two operations of the placer: P.`tryPlace`(n, g) and P.`finalPlace`(n, g), which represent trial placement and final placement, respectively. Trial placement temporarily places a cell in a row, returns the displacement cost, and undoes this placement. Final placement immediately places a cell in a row after the gate assignment and the target row of the cell is determined. In this part, we introduce trial placement and how to use our placer to assist the resynthesis and control displacement.

The trial placement flow is described in Algorithm 4. For each row R with sufficient space and matching height to gate g, we try to place the cell c (representing node n with assigned gate g) into R (line 4).

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

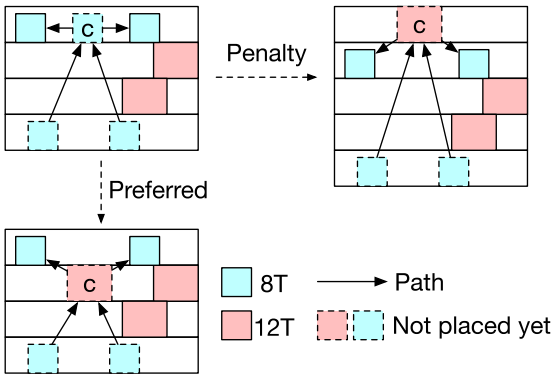


Fig. 6. Constraint of movement. Movements that increase both input and output wire distance will be penalized significantly. Moving in a range to adjust row height constraint is preferred.

To prevent significant routing detours in extreme cases, we constrain the vertical movement of each cell during placement (line 5). Specifically, when placing a cell c, we consider its fan-in set, C_{in}, and fan-out set, C_{out}. Since our traceback process follows a reverse topological order, cells in C_{out} have already been placed at their new positions, while cells in C_{in} remain at their original ones. We add a vertical detour penalty to the displacement cost. This penalty is derived from dy(c, C), which represents the sum of vertical distances between cell c and $c' \in C$. Here, rowheight is the physical height of row R_n. As illustrated in Fig. 6, moving c upward increases its distance to both fan-in and fan-out cells, thus increasing the penalty. This mechanism incentivizes placing c in a row situated between its fan-in and fan-out connections. Furthermore, if the vertical penalty for placing cell c in a row R exceeds a predefined threshold, we discard R as a candidate location (line 6). This hyperparameter, $\text{threshold}_{\text{trial}}$, is set to approximately three times the square of the row height. This rule effectively prevents cells from being assigned to rows that are excessively far from their connected logic.

In line 7, the horizontal displacement cost is calculated in a `placeCell` operation with `Ind` being Trial.

The optimal total displacement is then updated (lines 8–10). Note that the values obtained at this stage are only used as a temporary reference. The final placement position and cost can only be obtained after all cells are placed. The row R' with the minimum cost is returned to Algorithm 2 to identify the gate assignment.

Final placement. The final placement process simply inserts the cell c in the row with the minimum displacement cost, as demonstrated in Algorithm 5. Note that we do not allow placement failure in the final placement. The cell must be placed in a row, even if the cost value is large.

Now, with Algorithms 4 and 5, we have implemented a callable placer for the DP-based resynthesis algorithm. The placer is able to dynamically estimate displacement cost and place the cells through the DP traceback process, with amortized logarithmic complexity.

D. Iteration Strategy and Hyperparameters

Before each iteration, we will modify the hyperparameters to generate a better solution in the next iteration.

Algorithm 4 Trial Placement

1: **function** TRYPLACE(Node n, gate g)
2: $x_c \leftarrow x_n$, $w_c \leftarrow w_g$, $R' \leftarrow$ original row R_n;
3: $dis \leftarrow INF$;
4: **for** R in all Rows that satisfy g **do**
5: $d \leftarrow (dy(R, C_{in}) - dy(R_n, C_{in}) + dy(R, C_{out}) - dy(R_n, C_{out})) \times rowheight$;
6: Continue if $d >$ threshold$_{trial}$;
7: $d+ =$ `placeCell`$(c, R, Trial)$; ▷ Algorithm 3
8: **if** $d < dis$ **then**
9: $dis \leftarrow d$, $R' \leftarrow R$;
10: **end if**
11: **end for**
12: **Return** dis;
13: **end function**

Algorithm 5 Final Placement

1: **function** FINALPLACE(Node n, gate g)
2: $x_c \leftarrow x_n$, $w_c \leftarrow w_g$, $R' \leftarrow$ original row R_n;
3: $dis \leftarrow INF$;
4: **for** R in all Rows that satisfy g **do**
5: $d \leftarrow (dy(R, C_{in}) - dy(R_n, C_{in}) + dy(R, C_{out}) - dy(R_n, C_{out})) \times rowheight$;
6: $d+ =$ `placeCell`$(c, R, Trial)$; ▷ Algorithm 3
7: **if** $d < dis$ **then**
8: $dis \leftarrow d$, $R' \leftarrow R$;
9: **end if**
10: **end for**
11: `placeCell`(c, R', Final);
12: **end function**

1) If the area_slack is negative, the next iteration is slack. Otherwise, it will be a tight iteration. In our experiments, in order to align with the baseline, we force the program to ensure that the total area does not exceed the baseline. Therefore, the initial area_slack is the difference between the total area of the baseline result and that of our initial solution.
2) If it is a tight iteration, check whether the timing requirements are satisfied. If the timing requirement is satisfied and the area has slack, we reduce the clock period by the current WNS. This forces the algorithm to convert cells on critical paths into 12T.
3) The generation of threshold TH in Algorithm 2 requires a hyperparameter γ. The slack and tighten iterations have their own γ, respectively, both starting from 0.02 and doubling in their own iteration. This threshold defines the "cost-effective" gate resizing choices in each iteration. Similar to other heuristic iteration methods guided by the critical path distribution, this allows us to roughly allocate resources to the most important gates. It enables us to reassign a number of gates in one iteration, which is efficient and proves to be effective.
4) The iteration ends when the area_slack stops changing or the iteration number reaches the predefined limit. By default, we set the maximum number of iterations

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

TABLE II
BENCHMARK SCALE

Case	Cells	Nets	Area	Case	Cells	Nets	Area
adder	891	1147	184.844	max	2287	2799	486.13
bar	2188	2323	753.074	multi	16941	17069	4466.54
div	48343	48471	9669.61	sin	4203	4227	1243.09
hyp	168035	168291	38368.1	sqrt	21832	21960	5523.95
log2	21148	21180	5193.71	square	12985	13050	2709.18

to 8. More iterations do not necessarily lead to better results. We explored the area–delay optimization curve through the iteration process in Section IV-C.

IV. EXPERIMENTS

We implement our algorithm in C++ to ensure optimal performance and memory efficiency for large-scale designs. The experiments of baselines and our method were all conducted on a Debian server with Intel[1] Xeon[1] Silver 4310 CPU. The implementation leverages the mockturtle [26] logic network library, which serves as the foundational framework for maintaining the netlist graph representation. This library provides data structures for DAGs and offers interfaces for cut set generation and technology mapping, supporting our gate sizing and restructure process. For static timing analysis (STA), we employ OpenTimer [27], [28]. However, the original OpenTimer lacks native interfaces to directly calculate point-to-point delay between arbitrary nodes in the timing graph. We modified OpenTimer's APIs with Flute3 [29]. It will now only rebuild the RC tree locally after we modify the technology mapping of a node, enabling an efficient DP process.

We construct our NIMCH technology library based on the 15-nm FinFET-based Open Cell Library [30]. The method of library modification was demonstrated in [31]. The benchmark generation process follows the same parameter specifications as described in the Lesyn framework [11]. The size of the benchmarks is listed in Table II. "Area" is the area of the original design after initial placement in (μm^2).

Circuit designs are collected from the EPFL combinational arithmetic benchmark suite [32]. Our experiments do not involve sequential circuits. As mentioned in Lesyn [11], the designs are synthesized with library consisting of both 8T and 12T cells. Then, we apply Library Exchange Format (LEF) modification to the designs, unifying the heights of all cells to 8T. The area and other parameters of the 12T cells will not be changed in this process. Subsequently, we utilize the commercial tool innovus to generate the high-quality initial placement result. Finally, the shape of the 12T cells is recovered, and we get R_0 mentioned in Table III. The wire parasitics (SPEF) of the initial placement are also extracted and will be fed into OpenTimer later for timing calculation. The library files, including Liberty timing models (.lib) and physical libraries (LEF/DEF), along with timing reports containing critical path information, are sent to the main program of Easyn for row height assignment first. Our resynthesis–legalization engine will then optimize technology mapping and placement iteratively, as illustrated in Fig. 2. Finally, the legalized design will be evaluated by innovus.

The main optimization targets include power, performance, and area (PPA). Since there is always a tradeoff between delay and power (area), a small extra area consumption can lead to a significantly lower delay. Therefore, we force our program to end the iteration with both a lower delay and area compared to the baseline. It should be noted that our algorithm does not introduce randomness. The output will always remain the same if the hyperparameters are not changed.

A. Overall Performance

The overall performance results are tabulated in Table III. We compare our method with direct legalization [5], which only employs a row-based legalization algorithm, and resynthesis and legalization in sequence (Lesyn [11]), which decouples the logic optimization and physical placement stages. Area, power, delay, and time in the table represent the total area, the leakage power, the delay of the worst path, and the runtime of the program. These metrics collectively represent the fundamental PPA tradeoffs. In most of the evaluated test cases, Easyn achieves significant improvements across all metrics, demonstrating superior optimization capability through its integrated approach that simultaneously considers logic structure modifications, technology mapping decisions, and placement constraints. In "square," Easyn does not dominate the baseline. The tradeoff between delay and area led to a higher delay. Our area control only limited the upper bound of area. Therefore, there was difficulty to perfectly align PPA with baseline. We believe that the overall experimental results have proven the effectiveness of our method.

The improvements can be attributed to the synergistic optimization between gate sizing and displacement-aware legalization, which enables better exploration of the solution space compared to traditional decoupled methodologies.

We also present a comprehensive comparison of delay metrics with the legalization-free results in Table IV. The w/o Legal column represents the design after initial placement without legalization constraints (i.e., they are not subject to the NIMCH placement constraints C_1–C_4 and can be placed with arbitrary vertical positions). The area usage of it is the same as the direct legalization baseline listed in Table III. The columns of Lesyn and Easyn still report the delay after complete optimization, including legalization. Decoupled represents the result of a modified version of Easyn without the physical-aware property, disabling the feedback between placement metrics and resynthesis decisions, which serves as an observation for the ablation study to validate our integrated optimization approach.

The initial layout, developed without legalization constraints, predictably exhibits minimal delay. The legalization process introduces placement perturbations that result in routing detours, thereby increasing interconnect delay. Our algorithm optimizes the PPA of the circuit by assigning appropriate gates and placement. Figs. 7–9 illustrate the experimental results that will be discussed below. Fig. 10 provides a visual illustration of the initial layout and our legalized layout

[1]Registered trademark.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

TABLE III
PPA AND RUNTIME COMPARISON WITH BASELINES

Case	Legalization [5]				Lesyn [11]				Easyn			
	area (um^2)	power (mW)	delay (ns)	time (s)	area (um^2)	power (mW)	delay (ns)	time (s)	area (um^2)	power (mW)	delay (ns)	time (s)
adder	197.2	0.012	4.359	28	199.066	0.013	3.704	36	184.7	0.011	1.847	24
bar	744.1	0.043	1.731	22	742.835	0.043	0.522	44	707.2	0.036	0.412	124
div	9930.8	0.549	327.512	421	9905.2	0.532	220.435	1375	9606.3	0.472	213.439	1617
hyp	38830.1	2.071	4804.936	2346	38956.1	2.093	1850.964	3758	38476.2	1.981	1327.967	1659
log2	5638.9	0.334	23.154	369	5689	0.352	18.315	724	5587.2	0.326	16.354	853
max	494	0.025	3.879	32	507.489	0.026	3.836	56	479.1	0.022	2.974	149
multi	4415	0.248	12.582	178	4822.73	0.273	9.823	485	4397.7	0.241	8.712	457
sin	1212.7	0.072	4.351	40	1269.94	0.079	3.908	157	1253.6	0.071	3.209	172
sqrt	5962.3	0.355	259.114	371	5616.98	0.314	238.684	688	5510.5	0.297	194.465	856
square	2705.1	0.135	8.625	164	2771.36	0.141	7.308	214	2693.1	0.132	7.405	324
Normalize	1.000	1.000	1.000	1.000	1.033	1.015	0.744	2.167	**0.975**	**0.912**	**0.610**	2.916

TABLE IV
DELAY COMPARISON WITH LEGALIZATION-FREE PLACEMENT AND ABLATION STUDY

Case	w/o Legal	Lesyn	Decoupled	Easyn
adder	1.462	3.704	1.894	1.847
bar	0.573	0.522	0.437	0.412
div	92.017	220.435	232.561	213.439
hyp	313.51	1850.964	1461.325	1327.967
log2	8.688	18.315	18.152	16.354
max	2.231	3.836	3.440	2.974
multi	3.391	9.823	9.078	8.712
sin	2.233	3.908	3.209	3.209
sqrt	41.964	238.684	202.645	194.465
square	3.103	7.308	7.552	7.405
Normalize	1.000	2.826	2.425	2.278

for the `div` benchmark, with the 8T cells in blue and the 12T cells in red.

Overall, our method achieves a delay reduction of 18% compared to Lesyn while simultaneously maintaining a smaller area occupancy. The optimization mainly comes from two aspects.

Reduced Routing Detour. Easyn implements a tightly integrated resynthesis and legalization joint optimization methodology, where the techniques employed to control cell displacement and minimize perturbation distance significantly reduce the final delay caused by routing detours. Our row height assignment method also alleviates the problem of excessive concentration of the same row height generated by k-means in some cases, achieving better distribution of mixed-height cells across the whole placement region. Note that this effect is a comparison with the baselines. Some rows of the same height still have a certain degree of concentration as illustrated in Fig. 10, because of the NIMCH placement constraints.

Conserved Critical Paths. Easyn seeks to preserve 12T gate assignment for timing-critical paths during both the row height assignment phase and the resynthesis process. This timing-aware gate selection property proves effective in scenarios where the total gate delay dominates over interconnect delay in the critical path composition. For instance, in the benchmarks `max` and `sin`, Lesyn does not exhibit a significant delay increase, indicating that wiring detour is not the primary timing bottleneck in these designs. Nevertheless, we still achieve substantial optimization gains through our strategy, which demonstrates the effectiveness of our timing-aware technology mapping optimizations.

To verify the scalability of our method, we have included a scalability analysis to evaluate both the runtime and optimization effectiveness of our method on benchmarks of increasing scale.

Fig. 7 compares the runtime of our method against the Lesyn baseline. Note that the `div` and `hyp` benchmarks were omitted from the plot to maintain a clear visual scale, but their runtime trends are consistent with the other results. While our method exhibits a longer absolute runtime, its growth rate remains manageable for larger benchmarks, avoiding the complexity explosion. This empirical result aligns with the theoretical complexity analysis presented in Table III. That is, the scale of the DP table has an upper bound of $O(n)$, and each placement operation has an upper bound of $O(\log(N_r))$, where N_r stands for the cell placed in the row. Therefore, the time complexity of each iteration is bounded by $O(n\log(N_r))$. In the largest benchmark `hyp`, it appears that the iteration number is only 75% of the average, finishing faster than expected.

To assess optimization effectiveness, we compare the circuit delay against the baseline in Fig. 8. The y-axis represents the ratio between the delay of our method and that of Lesyn. Since the total area is comparable between our method and Lesyn across all test cases, delay serves as the primary performance metric. As illustrated by the delay ratio in the figure, the optimization quality is consistently maintained as the circuit scale increases.

Together, these analyses confirm the scalability of our approach. Our theoretical framework and experimental results provide strong evidence that the effectiveness of our algorithm is not diminished by increasing circuit scale.

B. Ablation Study

Cooperation between DP Traceback and Placement. Table IV also serves as an ablation study. `Decoupled` represents a modified implementation based on Easyn's core algorithms, where the primary modification involves disrupting the bidirectional communication flow illustrated in Algorithm 2. Specifically, this method eliminates the

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

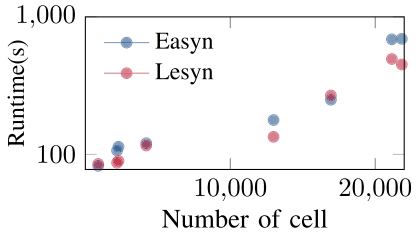


Fig. 7. Time consumption as scale grows.

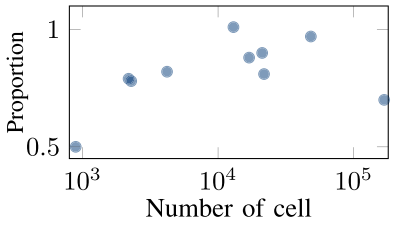


Fig. 8. Optimization effect on delay as scale grows.

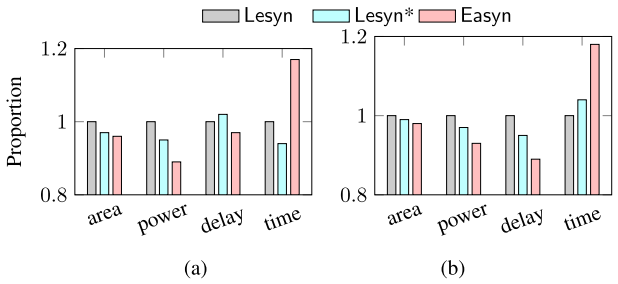


Fig. 9. Contribution of row height assignment. (a) `div`. (b) `sqrt`.

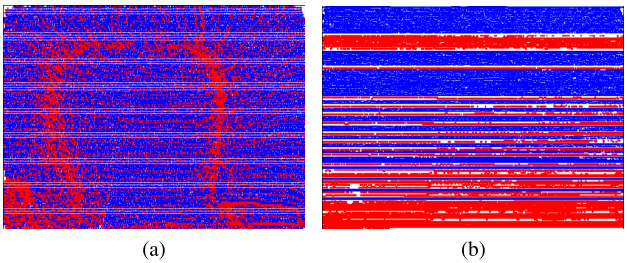


Fig. 10. In case `div`. (a) Initial placement result which violates the NIMCH placement constraints and (b) Easyn result after resynthesis and legalization.

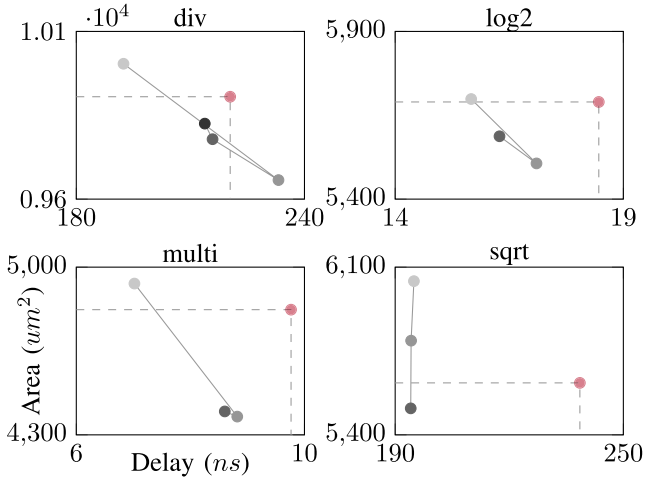


Fig. 11. Area–delay tradeoff through iterative rebalancing. Each dot is an iteration and the dot gets darker as the iteration proceeds. The area usage is reduced to below the limit while the timing is still better. In case `sqrt`, room for optimization exists in the initial synthesized design, so both area and delay are reduced.

feedback mechanism from the placement engine during the DP traceback process for resynthesis decisions. The cost calculation procedures that are dynamically supported by the incremental placer in lines 8, 10, and 14 of the algorithm, which originally incorporate accurate displacement costs, are replaced by a simple estimation based solely on the vertical Manhattan distance to the nearest row with matching height constraints. After the resynthesis process is complete, all cells are legalized in a single batch operation. This legalization process essentially degenerates to applying traditional Abacus-style legalization directly, losing the benefits of incremental placement refinement and resynthesis–placement co-optimization.

The experimental results clearly demonstrate that the decoupled version is more unstable compared to Easyn. Although the overall performance still surpasses Lesyn across the benchmark suite, it performs substantially worse than Easyn in terms of both consistency and optimization quality. In several benchmark cases, the decoupled version produces longer critical path delays than Lesyn. This empirical observation conclusively validates the effectiveness of our integrated optimization flow and underscores the crucial role of the incremental placer in dynamically providing displacement cost that can guide resynthesis decisions.

Contribution of Row Height Assignment. We conducted an ablation study to evaluate the contribution of the row height assignment stage. In this experiment, we modified the row height assignment process of the baseline, Lesyn, to function identically to that of Easyn, and then reran the standard test flow. As illustrated in Fig. 9, this modified baseline (denoted as Lesyn*) yielded improved PPA in some cases, such as `sqrt`. However, in many other instances, it resulted in tradeoffs, such as lower area/power at the cost of higher delay or vice versa. Consequently, this experiment did not demonstrate a dominant optimization trend.

C. Iteration Performance

Fig. 11 illustrates the iterative evolution of the area–delay optimization curve across several representative benchmark cases, providing visual insights into our algorithm's convergence behavior and solution space exploration. Each dot represents the result after an iteration and progressively darkens after successive iterations. Since our optimization framework enforces a strict constraint that the final area must not exceed that achieved by Lesyn (indicated by the orange marker), the program will first apply slack iterations to save area consumption. Then, in the tight iterations, the program seeks to reduce the worst path delay. Specifically, in `div`, `log2`, and `multi`, the slack iterations first reclaimed area consumption. Then, the tightening iterations optimized the critical paths within the area constraint. In `sqrt`, the original design seems to leave too much slack in noncritical paths,

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

so the program achieves both area and delay reduction in iterations. Since area and delay have a tradeoff, our program converges to the lower left of baseline, indicating that Easyn can find a better solution under a certain area constraint.

D. About Hyperparameter Selection

We used certain hyperparameters in row height assignment and DP process. They are assigned a default value as demonstrated above. For example, α and β are set to 0.2 and 0.8 in Section III-A, respectively, and γ starts from 0.02 in Section III-D. A series of experiments was performed in advance to determine these parameters. For example, if (α, β) were $(0.1, 0.9)$ or $(0.4, 0.6)$, the results were generally worse than the current setting. In a random setting in the range, no one setting dominates all other settings. The tradeoff among PPA also hinders the comparison. Therefore, we just selected a stable setting for these parameters on all benchmarks.

V. Conclusion and Future Work

In this article, we introduce a novel resynthesis and legalization joint optimization framework for the NIMCH design placement flow. We propose an iterative DP-based resynthesis algorithm tightly coupled with a corresponding incremental placer. It efficiently legalizes an NIMCH design under certain area constraints while achieving significant critical path delay reduction. This is realized by minimizing cell displacement to control wire length and routing detours, and distributing high-performance cell to timing-critical paths while applying downsizing in noncritical regions. The experiments show that our workflow achieves better PPA than state-of-the-art methods, with the maximum delay reduced by 18%. The code and benchmarks are shared at https://github.com/xjhthu/Easyn

Several valuable directions for future work can be explored. First, we did not consider constraints related to manufacturability, such as pin accessibility. The manufacturing legality metrics can be incorporated into the optimization objectives, or treated as constraints like C_1–C_4. Second, the combinational part of a sequential circuit is naturally compatible with our method. As long as the circuit has a clear topological order, our algorithm can be naturally extended to comply with the sequential cells (e.g., flip-flops). The induced extra challenges and constraints to the retiming algorithm may be tackled by parallel computing. Finally, the proposed method may be extended to encompass synthesis, macroplacement, and cell placement to enable its adoption in commercial flows.

References

[1] H. Mehdi and P. Paul, "Non-integer height standard cell library," US Patent 8788998, Current Assignee, Avago Technol. Int. Sales Pte Ltd., 2014.

[2] S. A. Dobre, A. B. Kahng, and J. Li, "Design implementation with noninteger multiple-height cells for improved design quality in advanced nodes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 4, pp. 855–868, Apr. 2018.

[3] S.-Y. Wu et al., "A 3 nm CMOS FinFlex platform technology with enhanced power efficiency and performance for mobile SoC and high performance computing applications," in *IEDM Tech. Dig.*, Dec. 2022, pp. 1–4.

[4] Y.-C. Zhao et al., "A mixed-height standard cell placement flow for digital circuit block," in *IEEE/ACM Proc. Design, Autom. Test Eurpoe*, Jun. 2019, pp. 328–331.

[5] Z.-Y. Lin and Y.-W. Chang, "A row-based algorithm for non-integer multiple-cell-height placement," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–6.

[6] J. Chen, Z. Zhu, W. Zhu, and Y. Chang, "Toward optimal legalization for mixed-cell-height circuit designs," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2017, pp. 1–6.

[7] J. Chen, Z. Zhu, W. Zhu, and C. Yao-Wen, "A robust modulus-based matrix splitting iteration method for mixed-cell-height circuit legalization," *ACM Trans. Design Autom. Electron. Syst.*, vol. 26, no. 2, pp. 1–28, Mar. 2021.

[8] W.-K. Chow, C.-W. Pui, and E. F. Young, "Legalization algorithm for multiple-row height standard cell design," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jul. 2016, pp. 1–6.

[9] X. Li, J. Chen, W. Zhu, and Y.-W. Chang, "Analytical mixed-cell-height legalization considering average and maximum movement minimization," in *Proc. Int. Symp. Phys. Design*, Apr. 2019, pp. 27–34.

[10] Y. Lin et al., "MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 6, pp. 1237–1250, Jun. 2018.

[11] Y. Pu et al., "Lesyn: Placement-aware logic resynthesis for non-integer multiple-cell-height designs," in *Proc. 61st ACM/IEEE Design Autom. Conf.*, Jun. 2024, pp. 1–6.

[12] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim, and I. L. Markov, "Sensitivity-guided metaheuristics for accurate discrete gate sizing," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2012, pp. 233–239.

[13] A. B. Kahng, S. Kang, H. Lee, I. L. Markov, and P. Thapar, "High-performance gate sizing with a signoff timer," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2013, pp. 450–457.

[14] C.-P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design. Dig. Tech. Papers*, Jul. 1998, pp. 617–624.

[15] S. Daboul, N. Hähnle, S. Held, and U. Schorr, "Provably fast and near-optimum gate sizing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3163–3176, Dec. 2018.

[16] A. Sharma, D. Chinnery, T. Reimann, S. Bhardwaj, and C. Chu, "Fast Lagrangian relaxation-based multithreaded gate sizing using simple timing calibrations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1456–1469, Jul. 2020.

[17] S. Hu, M. Ketkar, and J. Hu, "Gate sizing for cell library-based designs," in *Proc. 44th ACM/IEEE Design Autom. Conf.*, Jun. 2007, pp. 847–852.

[18] Y. Liu and J. Hu, "A new algorithm for simultaneous gate sizing and threshold voltage assignment," in *Proc. Int. Symp. Phys. Design*, Mar. 2009, pp. 27–34.

[19] S.-H. Liang, T.-L. Hsiung, W.-K. Mak, and T.-C. Wang, "Hybrid-row-height design placement legalization considering cell variants," in *Proc. Great Lakes Symp. VLSI*, Jun. 2023, pp. 363–367.

[20] S. Nath, G. Pradipta, C. Hu, T. Yang, B. Khailany, and H. Ren, "TransSizer: A novel transformer-based fast gate sizer," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2022, pp. 1–9.

[21] P. Pham and J. Chung, "AGD: A learning-based optimization framework for EDA and its application to gate sizing," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2023, pp. 1–6.

[22] Y. Ye et al., "Learning-driven physically aware large-scale circuit gate sizing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 5, pp. 1901–1914, May 2025.

[23] Y. Du, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Fusion of global placement and gate sizing with differentiable optimization," in *Proc. 43rd IEEE/ACM Int. Conf. Comput.-Aided Design*, Oct. 2024, pp. 1–9.

[24] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: Fast legalization of standard cell circuits with minimal movement," in *Proc. Int. Symp. Phys. Design*, 2008, pp. 47–53.

[25] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *J. ACM*, vol. 32, no. 3, pp. 652–686, Jul. 1985.

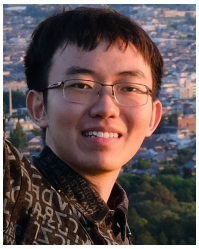
[26] M. Soeken et al., "The EPFL logic synthesis libraries," 2018, *arXiv:1805.05121*.

[27] T.-W. Huang and M. D. Wong, "Opentimer: A high-performance timing analysis tool," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Jun. 2015, pp. 895–902.

[28] T.-W. Huang, C.-X. Lin, and M. D. F. Wong, "OpenTimer v2: A parallel incremental timing analysis engine," *IEEE Design Test*, vol. 38, no. 2, pp. 62–68, Apr. 2021.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.

[29] M. Pan, Y. Xu, Y. Zhang, and C. Chu, "FastRoute: An efficient and high-quality global router," in *Proc. Int. Conf. VLSI Design*, 2012, pp. 1–18.
[30] M. Martins et al., "Open cell library in 15 nm FreePDK technology," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2015, pp. 171–178.
[31] Y. Zhang et al., "Multi-electrostatics based placement for non-integer multiple-height cells," in *Proc. Int. Symp. Phys. Design*, Mar. 2024, pp. 161–168.
[32] L. Amarú, P. Gaillardon, and G. D. Micheli, "The EPFL combinational benchmark suite," *Hypotenuse*, vol. 256, no. 128, 2015, Art. no. 214335.



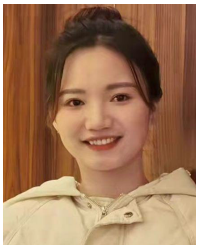
Jiahao Xu received the B.S. degree in information and computing science from Tsinghua University, Beijing, China, in 2024. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK), Hong Kong, under the supervision of Prof. Bei Yu.

His research interests include placement and routing algorithms in EDA.



Yuan Pu received the B.S. degree in computer science from The Chinese University of Hong Kong (CUHK), Hong Kong, in 2022, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, under the supervision of Prof. Bei Yu.

His research interests include combinatorial algorithms/AI in EDA.



Yu Zhang (Member, IEEE) received the B.S. degree in statistics from Central South University, Changsha, Hunan, China, in 2019. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

She was a Research Intern with Peng Cheng Laboratory, Shenzhen, China, from June 2020 to January 2021. Her main research interests include federated learning and machine learning in EDA.



Fangzhou Liu received the B.Eng. degree from the School of Electronic Science and Engineering, Nanjing University, Nanjing, China, in 2023. She is currently pursuing the Ph.D. degree with the Chinese University of Hong Kong, Hong Kong, under the supervision of Prof. Bei Yu.

Her research focuses on applying machine learning techniques to EDA and logic synthesis optimization.



Wenjian Yu (Fellow, IEEE) received the B.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 1999 and 2003, respectively.

He is currently a Professor with the Department of Computer Science and Technology, Tsinghua University. He has co-authored four books and over 200 papers in refereed journals and conferences. His current research interests include electronic design automation, high-performance numerical algorithms, and machine learning.



Kai-Yuan Chao (Kevin) (Senior Member, IEEE) received the M.S.E. and Ph.D. degrees in electrical and computer engineering from The University of Texas at Austin, Austin, TX, USA, in 1992 and 1995, respectively.

He has worked at Intel, Hillsboro, OR, USA, for many leading CPUs since P6B to 14-nm Broadwell, as well as at the Data Center Group Cloud Silicon Division, focusing on cloud platform chips and manufacturing technology co-optimization for chip and platform/system (DTCO and STCO) before his retirement in 2018. He has worked in Synopsys as the RD Group Director for Technology Pathfinding in Strategic Programs. He also advised few successful companies in AI software, EDA, integrated voltage regulator, memory-logic 3DIC, microLED and VCSEL/CPO mass transfer, and data center thermal solution. Since 2024, he has been working at Siemens EDA. His research interests include 3DIC, cross-domain technology co-optimization, and design automation.



Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently a Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received the 11 Best Paper Awards from ICCAD 2013, 2021, and 2024, the IEEE TSM 2022, DATE 2022, ASPDAC 2012 and 2021, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, six ICCAD/ISPD Contest Awards, IEEE CEDA Ernest S. Kuh Early Career Award in 2021, the DAC Under-40 Innovator Award in 2024, and Hong Kong RGC Research Fellowship Scheme (RFS) Award in 2024. He has served as the TPC Chair for the ACM/IEEE Workshop on Machine Learning for CAD and in many journal editorial boards and conference committees.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on September 13,2026 at 14:25:28 UTC from IEEE Xplore. Restrictions apply.