Lesyn: Placement-Aware Logic Resynthesis for Non-Integer Multiple-Cell-Height Designs

Yuan Pu
CUHK

Fangzhou Liu
CUHK

Yu Zhang
CUHK

Zhuolun He
CUHK

Yibo Lin
Peking Univ

Kai-Yuan Chao
Siemens

Bei Yu
CUHK

Abstract

Non-integer multiple cell height (NIMCH) standard-cell libraries offer promising co-optimization for power, performance and area in advanced technology nodes. However, such non-uniform design introduces new layout constraints where any sub-region can only accommodate gates of the same cell height due to manufacturability concerns. The existing physical design flow for NIMCH circuits, which handles the layout constraint by clustering and relocating gates according to their cell heights, often leads to substantial gate displacement that harms circuit performance. To alleviate the above issue, this paper proposes a row-based logic resynthesis procedure that explicitly adjusts cell heights after initial placement without changing cell positions. Experiment results demonstrate that compared with the conventional NIMCH physical design flow, our proposed approach can reduce the maximal delay by 26.1%.

1 Introduction

With advancements in technology nodes, standard cell libraries are now being designed with varying cell heights to meet diverse power, performance, and area (PPA) requirements of digital circuits. Traditionally, multi-row-height cells were sized as integer multiples of a single-row height. However, this approach often results in unnecessary area cost and may not align optimally with the shrinking transistor sizes and minimal wiring pitch [1]. In contrast, standard cells with non-integer multiple cell heights (NIMCH) offer a more extensive solution space for circuit design, allowing for highly flexible and efficient power, performance, and area co-optimization. TSMC, the latest 3nm manufacturer, has adopted NIMCH in its FinFlex cell technology to achieve improved PPA through place-and-route co-optimization [2]. An illustrative example by Dobre *et al.* demonstrates that the utilization of non-integer-height standard cells, such as 8T and 12T, leads to significant area reduction and performance enhancement compared to using solely 8T/12T standard cells [3].

NIMCH libraries bring new layout constraints in circuit placement: standard cells are constrained to be located in the sub-regions based on their cell heights, with each sub-region only accommodating gates of the same cell height. The conventional NIMCH physical design flow [3, 4] starts with an initial placement, and employs an

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656243

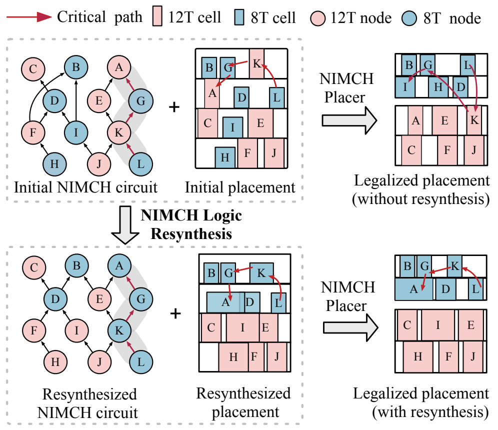


Figure 1: Illustration of the effect of logic resynthesis on the final NIMCH placement result. 8T/12T node in the circuit denotes that the node is mapped to a library gate with the height of 8T/12T.

NIMCH placer to form separate 8T/12T sub-regions, as is shown in the top part of Figure 1. Since 8T/12T cells are usually spatially mixed in the initial placement, the conventional flow could cause huge displacement of cells and significantly degrade the performance. One remedy to the conventional NIMCH physical design flow is the incorporation of **placement-aware logic resynthesis**, which resynthesizes cells in each separate sub-region to the same heights. As is shown in the lower part of Figure 1, the incorporation of placement-aware logic resynthesis significantly reduces cell displacement, thereby maintaining the circuit performance.

Although the literature has investigated NIMCH physical design flow, most of the studies focus on pure NIMCH placement. Based on the sub-region pattern, NIMCH placement approaches can be classified into two categories: island-based and row-based, as illustrated in Figure 2. For island-based NIMCH placement, a study by [3] employed Innovus to generate the NIMCH initial placement. An iterative timing-aware legalization process is then applied to move cells with the same heights to the same island. Chen *et al.* [5] formulated the NIMCH placement as a nonlinear optimization problem by introducing pseudo nets connected among gates with the same cell heights. For the row-based approach, Lin *et al.* [4] used Innovus to get the initial placement, followed by k-means-based row-height assignment and NIMCH-aware cell legalization to minimize cell displacement and achieve a legalized solution. They also demonstrated that row-based NIMCH placement outperforms the island-based approach regarding wirelength and total power consumption. Despite the efforts to advance placement algorithms, these approaches

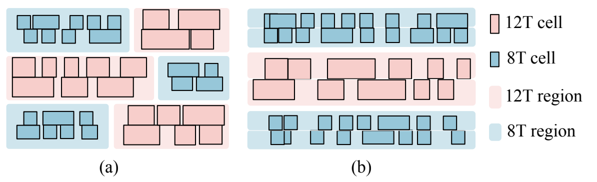


Figure 2: Illustration of (a) island-based and (b) row-based NIMCH sub-region patterns.

follow a pure placement flow. Given that cells with different heights are spatially mixed in the initial placement, these approaches usually end up with large cell displacement to meet the NIMCH layout constraints, and eventually degrade circuit performance. This scenario is illustrated in the top part of Figure 1, where the critical path $L \rightarrow K \rightarrow G \rightarrow A$ after NIMCH placement becomes considerably longer, resulting in a larger path delay.

To overcome the concerns above, we propose a new NIMCH physical design flow, which incorporates an additional step of placement-aware logic resynthesis. The core idea involves integrating the logic resynthesis procedure between the initial placement and NIMCH placement stages. By preserving the cell positions in the initial placement, our proposed procedure resynthesizes cells in the NIMCH circuit, such that the placement of the resynthesized circuit aligns with the row-based sub-region pattern. Figure 1 shows one example of incorporating our proposed logic resynthesis algorithm into the physical design flow of an NIMCH circuit. The resynthesized placement aligns with the row-based sub-region pattern, resulting in small cell displacement by the NIMCH placer and better timing performance (reflected by the shorter critical path). We summarize our major contributions as follows:

- This paper represents the first work proposing a placement-aware logic resynthesis approach tailored for NIMCH circuits.
- We develop a timing-aware strategy for row height assignment and a dynamic programming-based algorithm for row-based logic resynthesis.
- We propose two distinct strategies: (1) conservative logic resynthesis, which is computationally efficient; (2) structural logic resynthesis, which explores larger solution space.
- Experimental results on the EPFL arithmetic benchmark demonstrate that Lesyn reduces the displacement during NIMCH placement by 99.1%, and reduces the maximal delay by 26.1%.

2 Preliminaries

2.1 Basics of Logic Synthesis and Resynthesis

In logic synthesis, a Boolean network is a directed acyclic graph (DAG). Nodes in the Boolean network correspond to library gates, and edges represent wires. The terms Boolean network and circuit are used interchangeably. Primary Inputs (PIs) and Primary Outputs (POs) are nodes without fanins and fanouts, respectively. A cut of a node n is a set of nodes that must be traversed to reach n from PIs. A cut is K-feasible if its size does not exceed K. Each cut c of a node n can be associated with a truth table representing the function at n considered from its leaves. A library gate g **matches** a cut c if they have the same truth table (function). In a Boolean network, a cut c **covers** node n if n is mapped to a library g and g matches c. Given a node n and its associated cut set $Cut^n = \{c_1, c_2, ..., c_n\}$, assume n is originally mapped by a library gate g_i and g_i matches a cut c_i ($c_i \in Cut^n$), we can **remap** n with another gate g_j such that g_j matches another cut c_j ($c_j \in Cut^n$ and $i \neq j$). The remap operation will modify the circuit connectivity but not the functionality.

Logic resynthesis involves modifying the original circuit by substituting its sub-circuits or gates with logically equivalent yet different components. This modification can be categorized as **conservative** or **structural**. For conservative logic resynthesis, only the gate type is modified while the circuit connectivity remains unchanged. For structural logic resynthesis, the circuit connectivity is modified.

2.2 Row-Based NIMCH Logic Resynthesis

Problem 1 (Row-Based NIMCH Logic Resynthesis)**.** Given a circuit C with n gates, a set of library gates L and the initial placement of C, resynthesize each gate in C to minimize the combined cost of total cell area and maximal delay, and ensure the placement of the resynthesized circuit aligns with the row-based sub-region pattern.

By aligning the resynthesized placement with the row-based sub-region pattern, little displacement is introduced by the subsequent NIMCH placer, implicitly improving the timing performance. There are two reasons for minimizing total cell area: (1) Large cell area may cause cell inflation on some placement rows, leading to larger displacement for NIMCH legalization and worsening timing performance. (2) Large cell area increases the utilization rate of the core, causing congestion in the subsequent routing flow.

3 Conservative Logic Resynthesis

In this section, we introduce the algorithm for NIMCH row-based conservative logic resynthesis. Assuming a node n_i in the original NIMCH circuit is covered by a cut c_i, for conservative logic resynthesis, we exclusively select library gates matching cut c_i as candidate library gates for the resynthesis of n_i. Conservative logic resynthesis can ensure that the circuit connectivity (topology) remains unchanged after resynthesis.

Figure 3 shows the overall flow of our proposed algorithm: Starting with the NIMCH circuit and its initial placement, the process commences with row height assignment, which assigns a distinct cell height to each gate in the circuit to form the row-based sub-region pattern. Following this and preserving the cell positions in the initial placement, dynamic programming (DP) is employed to resynthesize the circuit referring to the assigned cell heights, wherein the resynthesized solution candidates are tabulated. The ultimate logic resynthesis solution is ascertained by retracing the computed DP table. During the solution traceback, the problem of path reconvergence arises when a node has multiple fanouts, and more than one candidate library gates are selected for that node. A heuristic strategy is proposed to tackle this issue. The resynthesized placement is then fed into the SOTA row-based NIMCH placer [4] to obtain a legalized placement satisfying the NIMCH layout constraints. In the remainder of this section, we use two non-integer cell heights 8T and 12T to illustrate our algorithm for convenience. It is noteworthy that our algorithm can handle NIMCH circuits with any pair of non-integer cell heights.

3.1 Row Height Assignment

During row height assignment, a specific cell height is assigned to each cell such that the row-based sub-region pattern is established on the initial placement. In the later stage of resynthesis, each cell can be only resynthesized to the library gate with the assigned cell height.

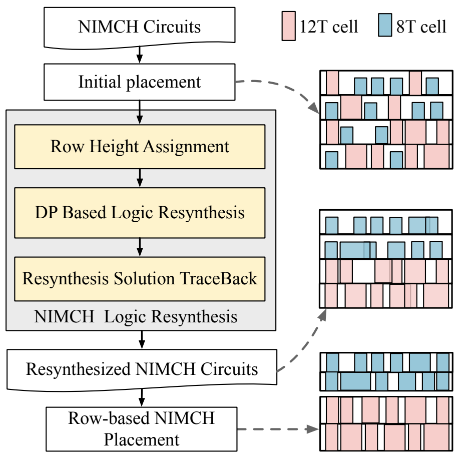

NIMCH Circuits
12T cell
8T cell
Initial placement
Row Height Assignment
DP Based Logic Resynthesis
Resynthesis Solution TraceBack
NIMCH Logic Resynthesis
Resynthesized NIMCH Circuits
Row-based NIMCH Placement

Figure 3: The overall flow of Lesyn.

We propose a majority-guided algorithm for row height assignment. Denote a placement row by r_i and the set of 8T/12T cells on r_i by C_i^{8T}/C_i^{12T}. The ratio between $|C_i^{8T}|$ and $|C_i^{12T}|$ is then denoted by ratio$_i$. The ratio$_i$ is multiplied by a predefined parameter β to control the overall 8T/12T ratio of the circuit, as shown in Equation (1).

$$\text{ratio}_i = \beta \frac{|C_i^{8T}|}{|C_i^{12T}|}. \tag{1}$$

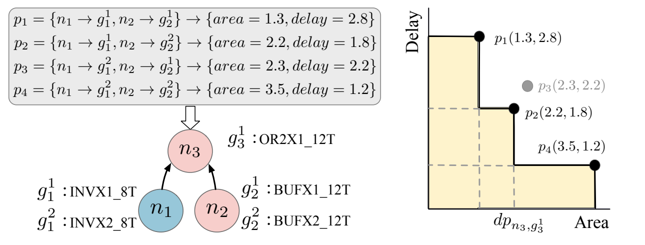
If ratio$_i$ > 1, the heights of all cells on r_i are set to 8T, and vice versa. However, the majority-guided row height assignment may cause issues such as bad timing performance (assigning the height of timing-critical gate to 8T) or area waste (assigning the height of non-timing-critical gate to 12T).

To resolve the issues above, we allow some of the placement rows to contain both 8T and 12T gates during the process of row height assignment (the NIMCH layout constraint violations will be resolved by row-based NIMCH placement in the later stage). We propose a timing-aware strategy for row height assignment: After applying the majority-guided algorithm above, we refer to the timing report of the initial NIMCH placement and sort all gates by their **slack** values. For all gates assigned to 8T, we select the top x gates exhibiting the smallest slack values and reassign them to 12T; Similarly, for gates assigned to 12T, the top y gates with the largest slack values are reassigned to 8T. In this work, x and y are set to be small values to reduce the displacement introduced by the row-based NIMCH placer. After row height assignment, for each node n_i, its assigned cell height is denoted by $RHA[n_i]$.

3.2 DP Based Logic Resynthesis

Referring to the row height assignment RHA, we employ dynamic programming to tabulate the resynthesized solution. We formulate the NIMCH row-based logic resynthesis problem in Equation (2). The objective is to minimize the total cell area (explained in Section 2.2). There are two constraints of the problem: (1) The maximal delay between any primary input and primary output is less than the target delay α; (2) The cell height of each gate g_i satisfies the row height assignment solution RHA.

$$\min \quad \sum_{i=1}^{N} A_i \quad \text{s.t.} \quad \text{Delay} \le \alpha, \quad g_i \in L, \quad h(i) = RHA[i], \tag{2}$$


$p_1 = \{n_1 \rightarrow g_1^1, n_2 \rightarrow g_2^1\} \rightarrow \{area = 1.3, delay = 2.8\}$
$p_2 = \{n_1 \rightarrow g_1^1, n_2 \rightarrow g_2^2\} \rightarrow \{area = 2.2, delay = 1.8\}$
$p_3 = \{n_1 \rightarrow g_1^2, n_2 \rightarrow g_2^1\} \rightarrow \{area = 2.3, delay = 2.2\}$
$p_4 = \{n_1 \rightarrow g_1^2, n_2 \rightarrow g_2^2\} \rightarrow \{area = 3.5, delay = 1.2\}$
n_3
g_3^1 : OR2X1_12T
g_1^1 : INVX1_8T
g_1^2 : INVX2_8T
n_1
n_2
g_2^1 : BUFX1_12T
g_2^2 : BUFX2_12T
Delay
$p_1(1.3, 2.8)$
$p_3(2.3, 2.2)$
$p_2(2.2, 1.8)$
$p_4(3.5, 1.2)$
dp_{n_3,g_3^1}
Area

(a) Illustration of fanin mappings of n_3 **(b) AD-curve of n_3 and g_3^1**

Figure 4: Illustration of the dynamic programming approach.

where N denotes the number of nodes in the circuit, L denotes the set of library gates, g_i represents the library gate resynthesized for node n_i, A_i and h_i denote the area and height of gate g_i, and $RHA[i]$ denotes the assigned cell height for node n_i.

While acquiring the cell area for any given node in the circuit is intuitive, the calculation of the maximum delay from a PI to any given node, n_i, can be decomposed into sub-problems concerning the computation of maximal delay for its fanin nodes. We can store the delay value of each node in a table, which can then be directly used to calculate the maximal delay(s) from PI (s) to its fanout node(s). This strategy eliminates the need for delay recalculation. Given these insights, we can solve the NIMCH row-based logic resynthesis problem by dynamic programming, and the DP table calculation follows the topological order (from PI to PO).

Before diving into the dynamic programming formulation, the definition of fanin mapping is first given in Definition 1.

Definition 1. (fanin mapping) For a node n in a circuit, denote its fanin nodes by Fanins(n) and for each fanin node k ($k \in$ Fanins(n)), Gates(k) denotes a set of library gates that can be mapped to k. A **fanin mapping** for node n, denoted by p_n, refers to a library gate mapping assignment for all fanin nodes of n ($p_n = \{k \rightarrow g_k, \forall k \in \text{Fanins}(n), g_k \in \text{Gates}(k)\}$).

There are three nodes $\{n_1, n_2, n_3\}$ in the circuit of Figure 4(a): n_1 and n_2 are fanin nodes of n_3, there are two candidate library gates for n_1 (g_1^1 and g_1^2) and two candidates for n_2 (g_2^1 and g_2^2). According to definition 1, there are four fanin mappings of node n_3: $p_1 = \{n_1 \rightarrow g_1^1, n_2 \rightarrow g_2^1\}$, $p_2 = \{n_1 \rightarrow g_1^1, n_2 \rightarrow g_2^2\}$, $p_3 = \{n_1 \rightarrow g_1^2, n_2 \rightarrow g_2^1\}$ and $p_4 = \{n_1 \rightarrow g_1^2, n_2 \rightarrow g_2^2\}$.

A 3-D table dp is introduced to describe the dynamic programming state. Given a node n_i, its candidate library gate g_i and its fanin mapping p_i, the dp entry dp_{n_i,g_i,p_i} stores two attributes, namely, *area* and *delay*. The attribute *area* represents the average area of gate g_i and its all fanin gates in p_i, and *delay* denotes the maximal delay from PI to the output pin of n_i (given that n_i is mapped to the library gate g_i, and its fanin mapping is p_i). Integrating all fanin mapping entries of dp_{n_i,g_i}, we derive an area-delay-curve (AD-curve), and each point on the AD-curve represents the area-delay characteristic of one fanin mapping of n_i. Note that the AD-curve is a Pareto curve, which means any point on the curve is not dominated by any other point (Definition 2 gives the definition of point dominance). Figure 4(b) gives one example of AD-curve of node n_3 and library gate g_3^1 (dp_{n_3,g_3^1}). p_3 is dominated by p_2 and is excluded.

Definition 2. (Point Dominance) In an AD-curve, a point i dominates another point j is $i.area < j.area$ and $i.delay < j.delay$.

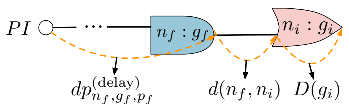


Figure 5: Illustration of delay notations mentioned in the subsection DP Entry Calculation.

To apply the dynamic programming approach on a circuit, we need to go through all nodes in the circuit in topological order (from PI to PO): For each node n_i and each candidate library g_i, if g_i matches the cut of n_i in the original circuit and the cell height of g_i satisfies the row height assignment *RHA*, we calculate the dp entries of dp_{n_i,g_i} for each possible fanin mapping p_i. Then, all fanin mapping entries of dp_{n_i,g_i} are integrated and pruned to construct the AD-curve. This process is propagated until the AD-curves of all nodes and their candidate library gates are constructed. Next, we delve into further elaboration on **dp entry calculation** and **AD-curve construction**.

DP Entry Calculation. For any node n_i in the circuit with library gate g_i and fanin mapping p_i, to determine the *delay* and *area* values of dp_{n_i,g_i,p_i}, we first determine the minimal *delay* value for each fanin of n_i: For each $(n_f, g_f) \in \text{Fanins}(n_i)$, where n_f denotes one fanin node of n_i and g_f represents the library gate selected for n_f, we select the fanin mapping point p_f with the minimal *delay* value from the AD-curve of dp_{n_f,g_f}, and store the value of $dp^{(\text{delay})}_{n_f,g_f,p_f}$ plus the wire delay from n_f to n_i ($d(n_f, n_i)$) in a table $d^{(i)}_{\text{min}}$, as shown in Equation (3).

$$d^{(i)}_{\text{min}}[(n_f, g_f)] = \min_{p_f \in dp_{n_f,g_f}} dp^{(\text{delay})}_{n_f,g_f,p_f} + d(n_f, n_i) \tag{3}$$

Then, we set the *delay* value of dp_{n_i,g_i,p_i} to be the maximal delay among d^i_{min} plus the gate delay of g_i ($D(g_i)$). The attribute *area* stores the average area of gate g_i and all its fanin gates in p_i.

We use a state transition function to depict the process of dp entry calculation, as formulated in Equation (4).

$$\begin{aligned} dp^{(\text{delay})}_{n_i,g_i,p_i} &= D(g_i) + \max_{n_f,g_f} d^{(i)}_{\text{min}}[(n_f, g_f)], \\ dp^{(\text{area})}_{n_i,g_i,p_i} &= \frac{A(g_i) + \sum_{g_f \in p_i} A(g_f)}{1 + |\text{Fanins}(n_i)|}, \end{aligned} \tag{4}$$

where $D(g_i)$ and the $A(g_i)$ denote the gate delay and area of library gate g_i, $d(n_f, n_i)$ denotes the wire delay from node n_f to node n_i. Figure 5 illustrates the meaning of $dp^{(\text{delay})}_{n_f,g_f,p_f}$, $d(n_f, n_i)$ and $D(g_i)$, and Figure 6 gives one example of dp entry calculation: Assume n_j and n_k are two fanin nodes of n_i, to calculate $dp_{n_i,g_i,\{n_j\to g_j,n_k\to g_k\}}$ (library gates g_j and g_k are mapped to n_j and n_k, respectively), we first determine the minimal delay d_j/d_k in the AD-curve of $dp_{n_j,g_i}/dp_{n_k,g_k}$. Then, $dp^{(\text{delay})}_{n_i,g_i,\{n_j\to g_j,n_k\to g_k\}}$ can be calculated as the maximal delay between d_j and d_k plus the gate delay of g_i.

AD-curve Construction. For any node n_i and its candidate library gate g_i, after calculating the dp entry of each fanin mapping, we need to construct the AD-curve of dp_{n_i,g_i}: Treating the dp entry of each fanin mapping as a 2D point with the corresponding *delay* and *area* value as the point coordinate, we prune the dominated points,

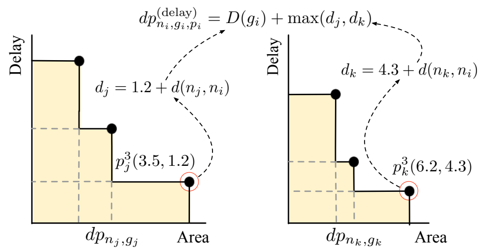


Figure 6: Illustration of dp entry calculation of dp_{n_i,g_i,p_i}, where $p_i = \{n_j \to g_j, n_k \to g_k\}$.

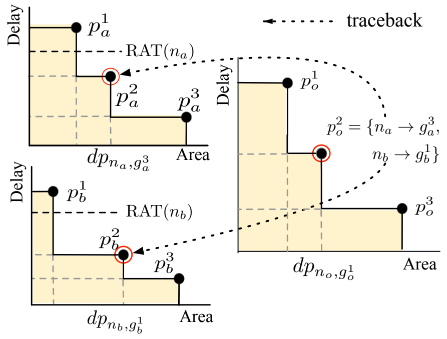


Figure 7: Illustration of solution traceback for fanin nodes (n_a and n_b) of n_o: Assume library gate g^1_o is mapped to n_o and the corresponding fanin mapping selected is $\{n_a \to g^3_a, n_b \to g^1_b\}$, we trace back to the AD-curves of dp_{n_a,g^3_a} and dp_{n_b,g^1_b}, and select the fanin mapping which satisfies the required arrival time constraint and has the minimal area (p^2_a and p^2_b).

and the pareto frontier constituted by the remaining points is the AD-curve of dp_{n_i,g_i}.

3.3 Resynthesis Solution Traceback

This sub-section details the process of solution traceback in the calculated DP table, and proposes a heuristic strategy to solve the issue of path reconvergence encountered during the traceback.

Solution Traceback. The solution traceback of the calculated DP table follows the reverse-topological order, and starts with the nodes whose fanouts are POs (output nodes for short). To determine the resynthesized gate selection of an output node n_o from a list of candidate library gates ($G_o = \{g^1_o, g^2_o, ..., g^k_o\}$), we merge all AD-curves of dp_{n_o,g^i_o} ($g^i_o \in G_0$) into a single AD-curve, and among the points whose delays are smaller than the required arrival time of n_o (RAT(n_o)), the point with the minimal *area*, say, dp_{n_o,g_o,p_o}, is selected as the resynthesized solution of n_o (that is, library gate g_o is selected as the resynthesis solution of n_o).

Referring to the fanin mapping p_o, we can further determine the library gate selection of the fanin nodes of n_o. Assume node n_f is one fanin node of n_o, and the library gate g_f is selected for n_f, a fanin mapping point p_f on the AD-curve of dp_{n_f,g_f} is required to be determined to **trace back** the resynthesis solutions for fanins of n_f: We first calculate the required arrival time RAT(n_f) of n_f. Then, the points on the AD-curve of dp_{n_f,g_f} whose *delays* are larger than RAT(n_f) are pruned, and the fanin mapping among the remaining points with the minimal *area* is selected. This traceback procedure is recursively leveraged until the gate-selection is completed for each node in the circuit. Figure 7 illustrates solution traceback.

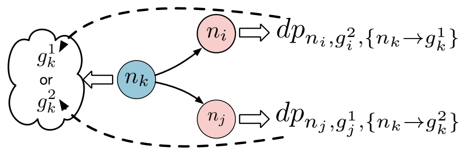

g_k^1
or
g_k^2
n_k
n_i
n_j
$dp_{n_i, g_i^2, \{n_k \to g_k^1\}}$
$dp_{n_j, g_j^1, \{n_k \to g_k^2\}}$

Figure 8: Illustration of path reconvergence: Assume $dp_{n_i,g_i^2,\{n_k \to g_k^1\}}$ and $dp_{n_j,g_j^1,\{n_k \to g_k^2\}}$ represent the resynthesis result for n_i and n_j, with fanin mappings $n_k \to g_k^1$ and $n_k \to g_k^2$ respectively. Two library gates (g_k^1 and g_k^2) are selected for n_k.

Path Reconvergence. Since the underlying topology of the gate-level netlist is a DAG instead of a tree, path reconvergence occurs in the process of solution traceback when two nodes (n_i and n_j) share the same fanin node (n_k), and the dp table entry determination of n_i and n_j leads to gate-type-selection conflict of n_k, as illustrated in Figure 8. The issue of path reconvergence is discussed in several works involving DP-like systematic solution tracing [6–8]. We propose a heuristic strategy to handle the path reconvergence issue. During the process of solution traceback, if the gate-selection conflict occurs for node n_k (that is, there are two candidate gates for n_k), we calculate the required arrival time of n_k (RAT(n_k)), and between the two candidate library gates, we choose the candidate with the minimal area whose *delay* is smaller than RAT(n_k) as the resynthesis solution of n_k.

By the timing-aware solution traceback and the heuristic strategy for path reconvergence fix, the total area is minimized and the timing constraint is satisfied for the resynthesis result.

4 Structural Logic Resynthesis

In this section, we extend the conservative logic resynthesis by allowing circuit topological modification, for larger solution space and potentially better resynthesis result. We propose structural logic resynthesis by only modifying the step of DP table calculation (Section 3.2): For library gate selection of any node n_i, instead of sticking to library gates which match the original cut of n_i, any library gate g_i which matches any cut in Cut_i (Cut_i denotes all cuts of node i) is expected to be selected as the candidate gate (as long as the cell height of g_i satisfies the row height assignment *RHA*). The dp entries and AD-curve are calculated and constructed for each g_i.

However, exhausting all library gates of all cuts may be computationally prohibitive for DP table calculation and resynthesis solution traceback. Moreover, for any node n_i and its cut c_j ($c_j \in Cut_i$), if the nodes in c_j are spatially distant from n_i in the initial placement, resynthesizing n_i to the library gate matching c_j may cause a large wire delay and thus bad timing.

To overcome the two concerns above, we constrain the maximal size of the cut to be 4 (4-feasible cut), and prune the spatially-distant cut(s) for each node in the step of DP table calculation. Figure 9 illustrates the process of structural resynthesis. Compared with conservative logic resynthesis, the structural approach explores the variants of the circuit connectivity and topological structure, enlarging the resynthesis solution space at the cost of a larger runtime.

5 Experimental Results

We implement the logic resynthesis algorithm in C++, with the logic synthesis library mockturtle [9] used for cut enumeration and technology mapping. OpenTimer [10], a static timing analysis (STA) tool, is used for timing evaluation. All experiments were performed on a 64-bit Linux machine with Intel Core i7 2.5GHz CPU and 64GB memory. By modifying the 15nm FinFET-based Open Cell Library (OCL) [11], we made our technology library contain the information of 8T and 12T cells. Figure 10 demonstrates the delay-area trade-off between 8T and 12T cells in our technology library.

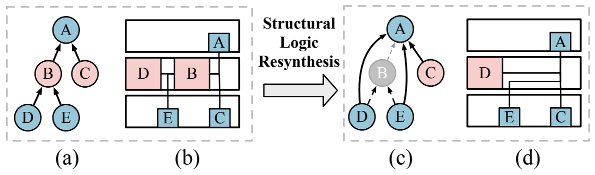


Figure 9: Illustration of structural logic resynthesis: in circuit (a), the node A has two cuts, namely, $\{B, C\}$ and $\{C, D, E\}$, and A is covered by the cut $\{B, C\}$. (b) is the placement before structural resynthesis. After structural resynthesis, in the circuit of (c), A is remapped to cut $\{C, D, E\}$, and B is dangling and thus removed. The corresponding placement of (c) is shown in (d).

To evaluate the effectiveness and efficiency of our method, we conduct our experiments on the EPFL combinational arithmetic benchmark suite [12]. Firstly, these designs are synthesized (technology mapped) by mockturtle. Then, we modify the standard-cell LEF files such that all cells share the same height (8T) with area unaltered (the reason for LEF modification is that existing commercial placement tools can only handle designs with the same cell heights). The circuits are then placed by Innovus in timing-driven mode to obtain the initial placement. Next, our proposed row-based logic resynthesis methods are leveraged on the initial placement to generate the resynthesized placement. Finally, we apply the row-based NIMCH placement [4] on both the initial placement (for conventional flow) and the resynthesized placement to eliminate potential cell overlaps and satisfy the NIMCH layout constraints. Note that during the row-based NIMCH placement, each gate reverts to its original height, with a corresponding update to the floorplan (height of each row).

In the implementation of the dynamic programming-based logic resynthesis, we apply Cadence Innovus to generate the wire parasitics (SPEF) of the initial placement, and feed the generated SPEF to Opentimer for delay calculation. For wire delay calculation of the structural resynthesis, whenever the resynthesis updates the circuit connectivity, we apply flute [13] to generate Steiner trees for the affected nets. Based on the Steiner trees, we construct corresponding RC trees and feed them back to Opentimer for wire delay calculation.

Since no clocks are defined for the combinational circuits, it is impossible to evaluate common timing metrics (such as WNS and TNS) and dynamic power consumption. Therefore, we follow the convention of logic synthesis to report experimental results in Table 1: 'area' denotes the total area of all gates in the circuit, 'power' reports the leakage power consumption, and 'delay' reports the maximal path delay in the placed circuit. Besides, 'disp' reports the displacement between the initial placement (generated by Innovus) and the layout-constraint-satisfied placement. The runtime of 'conventional flow' consists of the runtime of Innovus for initial placement and the runtime of row-based NIMCH placement [4] for layout-constraint-satisfied placement, while the runtime of 'conventional flow + Lesyn' sums up the runtime of initial placement generation, structural logic resynthesis and row-based NIMCH placement.

Table 1: Post-Place PPA results on the EPFL combinational arithmetic benchmark suite. 'Conventional flow' involves directly applying the row-based NIMCH placer [4] on the initial placement. For 'conventional flow +Lesyn', our proposed resynthesis approach is incorporated between the stage of initial placement generation and row-based NIMCH placement.

Circuit	Metrics		conventional flow [4]					conventional flow + Lesyn (Ours)				
	# Cells	# Nets	area (um^2)	power (mW)	delay (ns)	disp (um)	time (s)	area (um^2)	power (mW)	delay (ns)	disp (um)	time (s)
adder	891	1147	197.198	0.012	1.865	2410.82	47	197.231	0.011	1.715	923.65	59
bar	2188	2323	744.096	0.043	1.525	4281.73	54	727.213	0.039	0.485	311.30	77
div	48343	48471	9930.834	0.549	245.849	185512.02	1248	9850.830	0.525	208.065	6223.10	2182
hyp	168035	168291	38847.320	2.181	1154.000	4623408.61	3943	38923.400	2.082	795.058	30046.210	6505
log2	21148	21180	5638.914	0.334	21.330	102082.70	704	5546.227	0.324	12.821	5296.64	1307
max	2287	2799	493.994	0.025	2.839	2267.18	83	490.160	0.024	2.652	515.07	103
multi	21832	21960	4415.000	0.248	9.996	395585.00	493	4443.712	0.246	5.646	2186.75	836
sin	4203	4227	1212.678	0.072	3.710	13039.92	165	1219.117	0.072	3.269	540.42	252
sqrt	21832	21960	5962.334	0.355	147.273	168311.07	870	5942.411	0.361	142.552	2803.20	1377
square	12985	13050	2705.146	0.135	6.155	35430.74	240	2704.163	0.134	5.748	1944.83	361
Normalize	-	-	1.000	1.000	1.000	1.000	**1.000**	**0.999**	**0.965**	**0.739**	**0.009**	1.538

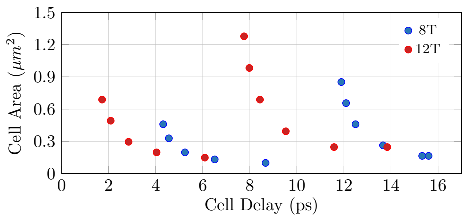
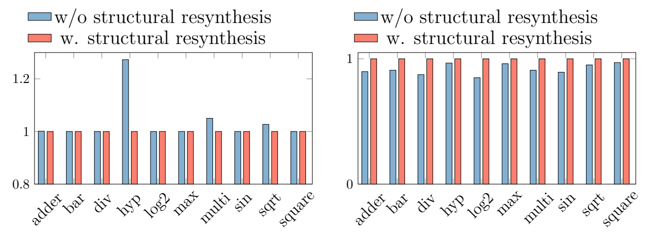


Figure 10: The delay-area trade-off between 8T and 12T cells in our technology library.



(a) maximal delay comparison **(b) runtime comparison**

Figure 11: Comparison of normalized maximal delay and runtime, with and without structural logic resynthesis.

As is shown in Table 1, by adjusting the parameter β in row height assignment and the target delay α in Equation (2), the total cell area of the initial circuit and resynthesized circuit remain almost the same. Compared with directly applying row-based NIMCH placer on the initial placement by Innovus, Lesyn reduces the maximal delay by 26.1%, at the cost of 55.1% additional runtime. Moreover, since our proposed resynthesis algorithm preserves the cell locations of the initial placement, the displacement of Lesyn is reduced by 99.1%, compared with the conventional NIMCH physical design flow. Little displacement introduced by our proposed approach guarantees that the timing-optimization by the initial placement is well preserved, leading to a smaller delay.

Figure 11 showcases a comparison between conservative and structural logic resynthesis on the EPFL arithmetic benchmark suite. Since the variance in total cell area, leakage power and displacement for the conservative and structural approaches are within 2%, we only compare the normalized maximal delay and runtime. Compared with the conservative logic resynthesis, the structural approach explores larger solution space and achieves 15.7% extra maximal delay reduction (exemplified by the case hyp, multi and sqrt), at the cost of 8.8% additional runtime.

6 Conclusion

In this paper, we introduce a new flow for NIMCH physical design, with the incorporation of placement-aware logic resynthesis. Our proposed row-based logic resynthesis algorithm starts with a timing-aware row height assignment strategy, which assigns specific cell heights to each row of the initial placement. Then, we develop a dynamic-programming-based algorithm for logic resynthesis, and a heuristic method is proposed to solve the path reconvergence issue of the solution traceback. Experimental results demonstrate that by keeping the total cell area unchanged, our algorithm can reduce the maximal delay by 26.1%.

References

[1] M. Hatamian and P. Penzes, "Non-integer height standard cell library," Patent, uS Patent 8,788,998.

[2] S.-Y. Wu, C. Chang, M. Chiang, C. Lin, J. Liaw, J. Cheng, J. Yeh, H. Chen, S. Chang, K. Lai *et al.*, "A 3nm CMOS FinFlex™ Platform Technology with Enhanced Power Efficiency and Performance for Mobile SoC and High Performance Computing Applications," in *Proc. IEDM*, 2022.

[3] S. A. Dobre, A. B. Kahng, and J. Li, "Design implementation with noninteger multiple-height cells for improved design quality in advanced nodes," *IEEE TCAD*, vol. 37, no. 4, pp. 855–868, 2017.

[4] Z.-Y. Lin and Y.-W. Chang, "A row-based algorithm for non-integer multiple-cell-height placement," in *Proc. ICCAD*, 2021.

[5] J. Chen, Z. Huang, Y. Huang, W. Zhu, J. Yu, and Y.-W. Chang, "An efficient epist algorithm for global placement with non-integer multiple-height cells," in *Proc. DAC*, 2020.

[6] Y. Liu and J. Hu, "A new algorithm for simultaneous gate sizing and threshold voltage assignment," in *Proc. ISPD*, 2009.

[7] K.-M. Lai, T.-W. Huang, P.-Y. Lee, and T.-Y. Ho, "ATM: A High Accuracy Extracted Timing Model for Hierarchical Timing Analysis," in *Proc. ASPDAC*, 2021.

[8] M. M. Ozdal, S. Burns, and J. Hu, "Gate sizing and device technology selection algorithms for high-performance industrial designs," in *Proc. ICCAD*, 2011.

[9] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. T. Calvino, D. S. Marakkalage *et al.*, "The epfl logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.

[10] T.-W. Huang and M. D. Wong, "Opentimer: A high-performance timing analysis tool," in *Proc. ICCAD*, 2015.

[11] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proc. ISPD*, 2015.

[12] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proc. IWLS*, 2015.

[13] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE TCAD*, vol. 27, no. 1, pp. 70–83, 2007.