TIMBER: A Fast Algorithm for Timing and Power Optimization using Multi-bit Flip-flops

Aditya Das Sarma[1], Shui Jiang[2], Wan Luan Lee[3], Tsung-Yi Ho[2], Tsung-Wei Huang[3]
[1]*Department of Computer Sciences, University of Wisconsin–Madison, USA*
[2]*Department of Computer Science and Engineering, The Chinese University of Hong Kong, China*
[3]*Department of Electrical and Computer Engineering, University of Wisconsin–Madison, USA*
adassarma@cs.wisc.edu, sjiang22@cse.cuhk.edu.hk, wlee329@wisc.edu, tyho@cse.cuhk.edu.hk, tsung-wei.huang@wisc.edu

***Abstract*—Multi-bit flip-flop (MBFF) banking and debanking is a widely adopted technique for optimizing power and total negative slack (TNS) during the post-placement stage of digital design. While banking flip-flops can reduce both power and area, excessive banking may lead to increased TNS due to significant register displacement, as well as bin density violations (BDVs) caused by over-placing MBFFs in legalized regions. To address these challenges, the EDA community recently organized a CAD Contest seeking innovative solutions from both academia and industry. In response, we present *TIMBER*, a fast and effective optimization algorithm that balances competing objectives in MBFF placement. Unlike existing methods, TIMBER employs a bin-density-aware placement strategy that simultaneously minimizes BDVs and TNS, while also achieving gains in power and area efficiency. To further enhance the runtime performance, TIMBER incorporates a parallelization strategy. Experimental results on the official 2024 CAD Contest benchmarks demonstrate that TIMBER outperforms the first-place winner, delivering on average 13.08× better solution quality, zero BDVs, 5.06× faster single-threaded runtime, 3.56× lower memory usage and up to 72.49× speedup in multi-threaded execution.**

I. Introduction

Multi-bit flip-flop (MBFF) banking and debanking is a widely adopted technique for optimizing power and total negative slack (TNS) during the post-placement stage of digital design [1]. MBFF banking is widely used to reduce area and power consumption by merging single-bit FFs into shared clocking structures [4]. As a result, MBFF banking not only improves energy and area efficiency, but also reduces the need for additional clock and routing resources [2], [3]. Despite these advantages, excessive MBFF banking can degrade timing on critical paths due to displacement of registers from their original locations. On the other hand, MBFF debanking splits MBFFs to smaller-bit FFs in order to recover timing, albeit at a cost of increasing power and area [5]. Figure 1 gives an example of MBFF banking and debanking.

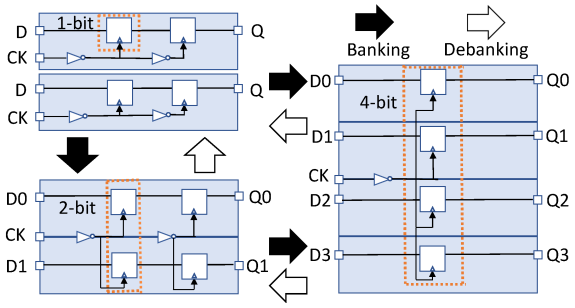


Figure 1: An example of MBFF banking and debanking to optimize timing, power, and area during the post-placement stage.

Due to the contrasting objectives, designing a good MBFF banking and debanking strategy is very challenging. This challenge is further exacerbated by additional resource constraints, such as limited bin capacity during MBFF placement. Exceeding the allowable bin capacity can lead to *bin density violation* (BDV). Specifically, BDVs occur when standard cells are unevenly distributed across placement bins, resulting in routing congestion, longer wirelengths, and degraded timing performance [8]. These violations can also hinder clock tree synthesis and strain the power delivery network, causing IR drops and localized thermal issues [9]. As a result, bin density control is a critical consideration in MBFF placement [10].

Several prior works have explored MBFF-based optimizations. For instance, [12] utilizes a weighted clustering approach based on k-means to group spatially close registers. Building on this, [13] applies a mean-shift algorithm to generate more stable and refined clusters. Other approaches, such as [14], employ mixed-driving MBFFs to optimize power by adjusting driving strengths based on local load conditions. Additionally, MBFFs are widely integrated in methods presented in [15]–[18]. However, these methods primarily emphasize banking, often overlooking the complementary role of debanking, which can limit their effectiveness in navigating trade-offs among power, area, and timing. In addition, they often neglect the bin density constraint as an optimization objective, which can lead to suboptimal placement results with increased routing congestion and degraded timing.

To address these problems, the EDA community organized a CAD Contest in 2024 ICCAD [6] to seek novel solutions for MBFF optimization from both academia and industry. However, as presented by [7], there remains substantial room for improvement, even in the first-place solution. Consequently, we propose *TIMBER*, a fast algorithm that generates a placement optimized for timing, area, and power under a bin density constraint. Unlike existing methods, TIMBER employs a bin-density-aware placement strategy that simultaneously minimizes BDVs and TNS, while also achieving gains in power and area efficiency. We summarize three key technical innovations below:

- **Balanced register selection:** We propose a fast, balanced selection strategy that evaluates multiple candidate groupings and selects the combination that best satisfies the overall design objectives.
- **Bin density-aware placement:** We efficiently explore all legal placements within a local window, prioritizing those that minimize timing degradation and BDV impact while preserving power and area benefits.
- **Parallel efficiency:** We adopt a multi-threaded design that partitions the die into independent regions, enabling concurrent execution and yielding substantial improvements in runtime efficiency.

We evaluate TIMBER on the ICCAD 2024 Contest Problem B benchmark suite [6], which includes designs with 101K–150K cells and approximately 20K registers. Compared with the first-place solution, TIMBER achieves an average speedup of 5.06× and 13.08× lower final cost score in single-threaded mode, while reducing peak memory usage by 3.56×. Using the default parameters given by the contest, TIMBER introduces zero BDVs and demonstrates strong robustness under varying bin utilization thresholds; achieving up to 42% fewer violations when the threshold is lowered to 80%. In parallel execution, TIMBER achieves a peak speedup of 72.49× using 16 threads.

II. Problem Formulation

We build TIMBER atop the problem formulation of 2024 ICCAD Contest Problem B [6], which provides a rigorous benchmarking environment for us to evaluate the solution quality of an MBFF placement algorithm. The input data consists of combinational gates and sequential elements (i.e., flip-flops (FFs)). Combinational gates are fixed in both structure and location as they cannot be modified or relocated. In contrast, FFs can be moved, merged (banked), or split (debanked) to optimize the placement objective in terms of timing, power, area, and bin density. Specifically, the objective is a weighted cost function defined as follows:

$$\sum_{i \in \text{FF}} (\alpha \cdot \text{TNS}(i) + \beta \cdot \text{Power}(i) + \gamma \cdot \text{Area}(i)) + \lambda \cdot D \tag{1}$$

where α, β, γ, and λ are weights, i denotes an FF, $\text{TNS}(i)$ is the total negative slack associated with an FF i, $\text{Power}(i)$ is its power consumption, and $\text{Area}(i)$ is its area. D represents the number of violated bins under the given utilization threshold ϵ. The die region is organized into rows of placement sites, which is referred to as placement rows. To accommodate routing congestion, the die region is partitioned into uniform bins each of size $B_W \times B_H$, defined per testcase. A bin is violated if its utilization exceeds the given threshold ϵ.

A. Placement Constraints

The output placement must satisfy the following constraints:

1) All instances must be placed entirely within the die region.
2) Instances must not overlap and must align with the available placement sites within the placement rows.
3) Nets connected to FFs must preserve functional equivalence relative to their original data input behavior.

A standard cell library is provided, specifying the dimensions, area, power, and timing characteristics of all available FFs that can be utilized in the final placement.

B. Timing Calculation

Timing is computed based on the displacement of FFs from their original locations and can be estimated by [21], [22]. In the initial placement, the D pin of each register is associated with an input slack value, and the corresponding Q pin has an intrinsic delay of δ_0. As illustrated in Figure 2, consider a pair of FFs, FF_1 (launching register) and FF_2 (capturing register). If FF_1 and FF_2 are moved or modified such that their wirelengths change from W_1 to W_1' and W_3 to W_3', respectively, and the delay of FF_1 changes from δ_0 to δ_0', the slack at FF_2 is updated as:

$$\begin{aligned} Slack(FF_2) = Slack(FF_2) + (\delta_0 - \delta_0') + \\ \tau \times (W_1 - W_1') + \tau \times (W_3 - W_3') \end{aligned} \tag{2}$$

where τ denotes the displacement delay factor, provided as part of the testcase.

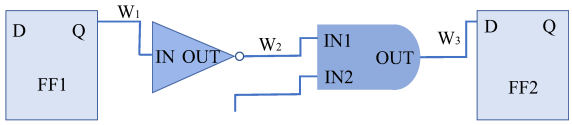


Figure 2: A pair of launching (FF1) and capturing (FF2) FFs. W_i denotes the wirelength of the net between Q/OUT and D/IN.

C. MBFF Banking and Debanking

The D pin and Q pin of each register are the fundamental unit for banking (merging) and debanking (splitting) MBFFs. FFs can be merged to form larger instances or split into smaller ones. Banking FFs generally improves power and area efficiency due to shared clocking resources, but this often comes at the cost of timing degradation and bin density violations. Debanking, on the other hand, can alleviate pressure on timing and bin constraints, but increases power consumption and area overhead. Therefore, the central challenge of the CAD Contest is to strike an effective balance between MBFF banking and debanking while meeting all specified placement constraints. Furthermore, all resulting FF configurations must conform to the standard cell library definitions provided in each input testcase.

III. Algorithm

In this section, we discuss TIMBER, a fast and efficient algorithm for timing and power optimization using MBFFs. TIMBER consists of three key steps, *register candidate formation step*, *placement candidate selection step*, and *legalization step*. First, in the register candidate formation step, all feasible register groupings are generated for each register pin, as permitted by the standard library. Next, in the placement location candidate selection step, each combination is evaluated and assigned to the most suitable subregion within the defined window, based on a cost function that balances power, area, timing and BDV penalty. Finally, in the legalization step, the registers are minimally displaced to eliminate overlaps while preserving timing integrity and avoiding bin density violations. Table I lists some symbols that we will use frequently in the following sections.

TABLE I: Frequently Used Notations

Symbol	Description
α	Penalty coefficient for TNS
β	Penalty coefficient for power
γ	Penalty coefficient for area
λ	Penalty coefficient for bin density violations
ϵ	Maximum bin utilization ratio
W_X	Width of the placement window
W_Y	Height of the placement window
T_M	Maximum number of window shift trials

A. Register Candidate Formation

1) Forming Clock Groups: To preserve functional correctness, which is a strict requirement outlined in the problem formulation, only register pins sharing the same clock net are eligible to be grouped together. Therefore, as a preprocessing step, all register pins are first partitioned according to their associated clock nets.

2) Sorting the Cell Library: The cell library consists of several multi-bit registers varying in size, area, and power consumption. For each available size, we select the register candidate that maximizes the value of $\beta \cdot \text{power} + \gamma \cdot \text{area}$. This ensures that each chosen candidate contributes the most toward improving the cost function defined in Equation 1.

3) Balanced Nearest Neighbor Combination Formation: Each unvisited register pin is considered as a seed for forming a combination using the candidates selected in the previous step. If a candidate supports k bits, the algorithm identifies the k nearest neighboring register pins to form a potential grouping. Once a valid combination is formed, all k participating register pins are marked as visited to avoid reuse. Combinations are attempted in descending order of size, and the process halts as soon as a legal placement location candidate is found, as described in the subsequent subsections.

B. Placement Location Candidate Formation

1) Window-based interval generation: We define a fixed-size rectangular window with dimensions $W_X \times W_Y$. For each register combination, we compute the average of the D pin coordinates from all constituent register pins, and center the window around this point. The objective is to identify an optimal placement location for the register combination within this window that minimizes negative timing slack while satisfying bin density constraints. If no legal placement is found within the current window, it is iteratively shifted horizontally until a valid region is identified.

To accomplish this, we adopt an approach similar to that of [11]. The windowed region comprises unoccupied areas and areas already occupied by registers or blockages (e.g., combinational cells or the window boundary). Each row in the window is divided into segments, where a segment refers to a continuous stretch of unoccupied space between two blockages.

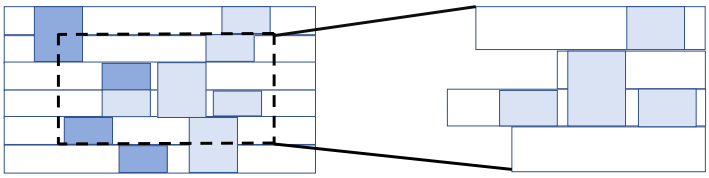


Figure 3: Extraction of segments within a local window for constructing a placement location candidate.

A segment may contain registers that are entirely located within its boundaries. We refer to these as local cells. The union of such segments across multiple rows forms a placement region candidate. For each row, one segment is selected, and if multiple segments exist, the one closest to the window center is chosen. This process is illustrated in Figure 3.

Next, we derive intervals from these segments. An interval is defined as a tuple: (left neighbouring cell, right neighbouring cell, $x_{\min}$, $x_{\max}$), where the left(right) neighbouring cell refers to the local cell immediately adjacent to the interval boundary, and $x_{\min}$ and $x_{\max}$ represent the start and end x-coordinates of the interval, respectively. If there are no left or right neighbouring cells, then we mark it as window boundary.

To prepare these intervals for candidate placement, we perform a right-shift of all local cells to their rightmost legal positions by processing them in a reverse topological (right-to-left) order. This procedure is described in Algorithm 1 and illustrated in Figure 4.

Algorithm 1 Extreme Right Placement of Local Cells

Input: Window region W
Output: Extreme right placement L of local cells
1: $V \leftarrow$ Extract local cells inside W
2: $topo \leftarrow$ Reverse topological sort of V
3: $L \leftarrow [\,]$ `// List to store placement states`
4: **for** each register $r \in topo$ **do**
5: Move r to the rightmost legal position
6: $L \leftarrow$ Current window state
7: **end for**
8: **return** L

Next, we perform a topological sort on the local cells to preserve their left-to-right ordering. Starting from this order, we incrementally shift each cell to the left, one at a time, while maintaining legality. For every valid position encountered during this process, we record the corresponding interval. This process continues until the leftmost legal position is reached. The complete procedure is detailed in Algorithm 2.

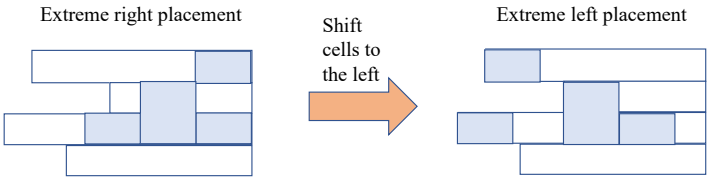


Figure 4: The figure on the left illustrates the extreme right valid placement of the placement region candidate from Figure 3.

Algorithm 2 Interval generation algorithm

Input: Extreme left placement configuration P
Output: Set of intervals C
1: $C \leftarrow \emptyset$
2: $topo \leftarrow$ topological sort of cells in P
3: **while** $topo$ is not empty **do**
4: $curr \leftarrow topo.dequeue()$
5: Shift $curr$ to its leftmost legal position
6: Create new interval I based on $curr$'s new position
7: $C \leftarrow I$
8: **end while**
9: **return** C

2) Generating placement location candidates: The intervals generated in the previous step are first sorted in increasing order based on their left endpoints. We then iterate through these intervals, and as soon as we accumulate enough vertically aligned segments to match the height of the register candidate, we identify a valid placement location candidate. Figure 5 illustrates some potential placement location candidates for our running example.

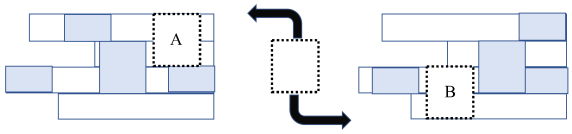


Figure 5: Inserting register candidate (black dotted boundary) into placement location candidates (A and B) generated from the intervals. The candidate with the lowest cost is selected.

If no suitable placement locations are found within the initial window, the window is shifted horizontally by W_X units in both left and right directions, up to T_M times. If a valid location still cannot be found, the algorithm falls back to the next smaller register size and repeats the process until a feasible placement is identified.

3) Evaluating placement location candidates: We evaluate the quality of each placement location candidate using the procedure described in Algorithm 3. Given that we maintain grid density information in an appropriate data structure, we first compute the additional area introduced by the current register candidate across the bins it spans. For each bin that exceeds its capacity threshold ϵ, we increase the placement cost by a penalty factor λ.

To account for timing, our aim is to minimize the distance between each register pin and its corresponding driver (precursor) pin. Reducing this distance inherently improves timing by lowering the TNS, as signal propagation delays are shortened.

This cost function, which penalizes both bin overflows and poor timing, ensures that placement candidates with fewer bin density violations and better timing characteristics are favored. Once all candidates are evaluated, the one with the lowest overall cost is selected. The register candidate is then placed at that location, and the grid density data structure is updated accordingly.

Algorithm 3 Evaluating Placement Location Candidate

Input: Global cost gc
Input: Placement location candidate P
Input: Density penalty λ
Input: TNS penalty α
Input: Displacement delay ω
Input: Maximum bin utilization ratio ϵ
Output: Updated global cost gc

$lc \leftarrow 0$ `// local cost`
$x_{\min} \leftarrow \min$(left endpoints of intervals in P)
$x_{\max} \leftarrow \min$(right endpoints of intervals in P)
for each placement site $p \in P$ such that $x_{\min} < p < x_{\max}$ **do**
    $lc \leftarrow lc + \lambda \times$ No. of Bin Violations in p
end for
for each register pin $r \in$ Register Candidate **do**
    $lc \leftarrow lc + \alpha \times \omega \times$ distance$(r, \text{precursor}(r))$
end for
$gc \leftarrow \min(gc, lc)$
return gc

C. Legalization

Once a register candidate is placed at its optimal location, a legalization step is performed to eliminate any overlaps with previously placed registers. This is done while preserving the original left-to-right ordering of local cells. To achieve minimal displacement, we use queues to propagate shifts outward from the newly placed register: local cells to its left are pushed leftward, and those to its right are pushed rightward.

Because the placement candidate is selected within a confined window region, we guarantee that all legalized cells remain within this window. This localized adjustment ensures overlap-free placement without compromising bin density or timing constraints. Figure 6 illustrates legalization in our running example. The full procedure is detailed in Algorithm 4.

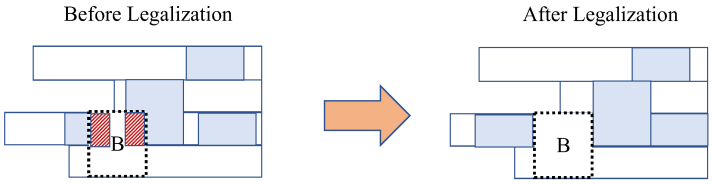


Figure 6: Inserting register candidate into placement location candidate B from Figure 5. Legalization ensures minimal displacement of the cells in the original placement in Figure 3 to ensure no overlaps (marked in red stripes). The relative order of the local cells is also maintained.

D. Time Complexity Analysis

1) **Clock Group Formation:** Linear scan over register pins takes $O(N)$, where N is the number of register pins.
2) **Cell Library Sorting:** Sorting m register types by weighted cost takes $O(m \log m)$ time. As $m \ll N$, this step is not a bottleneck.
3) **Segment and Interval Generation:**
   - Segment formation within a window takes $O(W_X \times W_Y)$, where W_X and W_Y are the window's dimensions.
   - Topological sorting and interval generation over L local cells each take $O(L)$ time.

   This process repeats for each unvisited register pin, resulting in a total complexity of $O(N \times (W_X W_Y + 2L))$.

Algorithm 4 Legalization Algorithm

Input: Unlegalized placement L
Input: Current cell candidate C
Output: Legalized placement L

$Q \leftarrow [\,]$ `// Queue to store local cells`
for $lc \in$ left neighbouring local cells of C **do**
    $Q.enqueue(\{lc, C\})$
end for
while Q is not empty **do**
    $\{new, curr\} \leftarrow Q.dequeue()$
    $w \leftarrow$ width of new
    $x \leftarrow$ lower-left x coordinate of $curr$
    $x_c \leftarrow$ lower-left x coordinate of new
    $x^* \leftarrow x_c + w - x$
    $x_c \leftarrow x_c - x^*$
    $Q.enqueue(\{\text{local cell to the left of } new, new\})$
end while
for $rc \in$ right neighbouring local cells of C **do**
    $Q.enqueue(\{rc, C\})$
end for
while Q is not empty **do**
    $\{new, curr\} \leftarrow Q.dequeue()$
    $w \leftarrow$ width of $curr$
    $x \leftarrow$ lower-left x coordinate of $curr$
    $x_c \leftarrow$ lower-left x coordinate of new
    $x^* \leftarrow x + w - x_c$
    $x_c \leftarrow x_c + x^*$
    $Q.enqueue(\{\text{local cell to the right of } new, new\})$
end while
return L

4) **Placement Evaluation:** Constant-time lookups yield $O(1)$ per candidate.
5) **Legalization:** Adjusting overlapping cells takes $O(L)$ per window. Since $L \ll N$, impact is minor.

Overall Time Complexity: The algorithm is dominated by the segment and interval generation phase. Therefore, the total time complexity is given as follows:

$$\mathcal{O}\left(N \cdot ((W_X \cdot W_Y) + 2L)\right)$$

IV. Experimental Results

We implemented TIMBER in C++17 and compiled it using `g++` version 11.4.0 with the `-O2` optimization flag enabled. All experiments were conducted on a Linux workstation equipped with a 13$^{\text{th}}$ Gen Intel® Core™ i5-13500 CPU and 80 GB of RAM. We evaluated our solution using the benchmark suite provided by the 2024 ICCAD Contest on Power and Timing Optimization with MBFFs [6]. This suite comprises designs containing 100K–150K standard cells and approximately 20K registers per design. Detailed statistics of the benchmark suite are outlined in Table II. For brevity, we rename the benchmarks as `case1`–`case7`, corresponding to `testcase1`–`testcase3` and `hiddencase01`–`hiddencase04`, respectively.

Since there is no universally optimal value for the parameters W_X, W_Y and T_M (defined in Table I) for all benchmarks, we expose them to the user as tunable parameters, allowing greater flexibility to meet the needs of the user's design objectives. In TIMBER, we set the window dimension to $W_X = 15$ and $W_Y = 15$, with T_M set to 15 for all, except `case4`, where T_M was increased to 25 to ensure successful placement of all register pins on the die. These values produce the best results for our objective.

TABLE II: Benchmark Statistics. α, β, γ and λ represent the weights associated with timing, area, power, and BDV, respectively (see Equation 1). ϵ is the maximum bin utilization threshold. #C is the number of cells, and #R is the number of registers.

Circuit	α	β	γ	λ	ϵ	#C	#R
case1	10	$2e3$	$2e-3$	$1e8$	92.93	108685	19879
case2	10	$4e2$	$8e-7$	$1e8$	98.27	153457	21164
case3	10	$1e4$	$2e-3$	$1e8$	97.44	101221	19789
case4	10	$2e5$	$2e-6$	$1e8$	92.93	108685	19879
case5	10	$4e4$	$8e-7$	$1e8$	98.27	153457	21164
case6	10	$4e2$	$8e-5$	$1e8$	98.27	153457	21164
case7	10	$1e4$	$2e-3$	$1e8$	97.44	101221	19789

A. Baseline and Scoring

We use the contest's first-place solution as our baseline. Since only the executable was provided, we cannot reason its algorithm but consider it as a black box. To ensure fair comparison, we normalize power, timing, and area across all testcases. Normalization is necessary due to the wide variation in raw metric values across different circuit sizes, which can span several orders of magnitude. While a common approach uses global min and max values from all contest submissions (over 90 registered teams for problem B alone), this is impractical since we don't have access to every team's executable as the contest organizers. Instead, we estimate the metric ranges individually. For power and area, we use the standard cell library to determine the registers with the lowest and highest values. For TNS, we assume a minimum of zero and estimate the maximum via Monte Carlo simulation by randomly placing registers and recording the worst-case result. This normalization allows us to quantify how far TIMBER and the baseline deviate from the potential best and worst values of each metric.

Since TIMBER targets the contest formulation, we focus on comparing with the first-place solution. We do not compare with other MBFF works (e.g., [11]–[14]) since they target a different problem formulation. Additionally, we do not compare with industrial tools, as they are designed to support a wide range of practical design constraints, such as power, timing, routing, and legal placement, which often come with nontrivial runtime and quality trade-offs.

B. Overall performance comparison

Table III presents a comparative analysis of key performance metrics between TIMBER and the baseline (the first place). As shown in Table III, the geometric means of the power, area, and timing metrics for both TIMBER and the baseline are generally comparable. TIMBER demonstrates a slight advantage in timing and power efficiency. Notably, in `case2` and `case6`, TIMBER achieves area reductions that are two orders of magnitude smaller than those of the baseline. We attribute this significant improvement to TIMBER's balanced register selection strategy, which prioritizes combinations that yield the greatest reduction in the overall cost function.

Next, we compare number of BDVs. TIMBER achieves *zero* bin violations across all testcases, whereas the baseline incurs several violations, with a maximum of 14 in `case4`. As highlighted in the contest as a constraint, having too many violated bins can incur severe problems, such as routing congestion, longer wirelength, and degraded timing performance, and thus should be avoided. We attribute TIMBER's zero BDVs to its bin-density-aware placement strategy, which carefully balances optimization objectives while strictly adhering to bin density constraints.

We further compare the peak memory usage and runtime of both solutions. On average, TIMBER consumes 3.56× less memory than the baseline, with a peak reduction of 5.15× in `case6`. In terms of runtime, TIMBER achieves an average speedup of 5.06×, with significant gains of 14.92× and 14.8× for `case2` and `case5`, respectively. These improvements are largely due to TIMBER's lightweight and efficient implementation. The algorithm employs a quick neighbor selection strategy to maximize area savings and a highly optimized legalization subroutine that ensures overlap-free placements.

Finally, we examine the overall final scores collected by the official checker program [6], where lower values indicate better performance. TIMBER outperforms the baseline in all testcases, except `case2`, achieving an average improvement of 13.08× and a maximum of 41.41× in `case4`. In `case2`, although the score is slightly worse due to an unusually low value of γ that heavily prioritizes timing over other objectives, it is noteworthy that TIMBER is still approximately 15× faster for this testcase. Figure 7 illustrates the outputs produced by TIMBER and the baseline when `case4` is the input.

(a) Placement solution of TIMBER. **(b) Placement solution of the first place.**

Figure 7: Placement solutions for `case4`. Red regions depict non-movable sequential logic while blue regions represent the registers. TIMBER produces a 41.41× better solution based on the contest score.

C. Solution Quality under different Bin Utilization Ratios (ϵ)

As bin density is emphasized as a constraint by the contest, different utilization ratios can significantly impact solution quality. To further evaluate TIMBER's robustness to BDVs, we conducted an experiment by running both algorithms under a smaller bin density threshold (ϵ). In particular, lowering ϵ increases white space (e.g., beneficial for routing) but also raises the risk of BDVs, making it an effective stress test for different placement strategies. This setup enables us to assess the resilience of both TIMBER and the baseline under tighter bin density constraints.

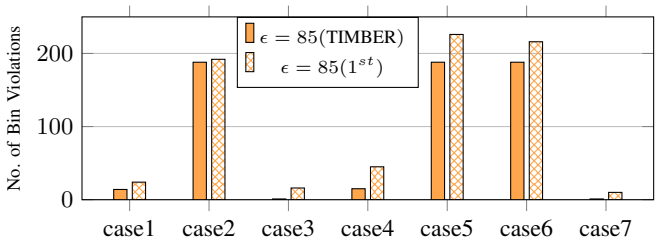
Figure 8 presents the comparison for $\epsilon = 85$ and $\epsilon = 80$ across all testcases. TIMBER consistently results in fewer BDVs than the baseline across the board. For $\epsilon = 85$, TIMBER achieves a 22.5% reduction in BDVs (Figure 8a), while for $\epsilon = 80$, the reduction is 16.5% (Figure 8b). Notably, under $\epsilon = 85$, TIMBER achieves 16× and 10× fewer BDVs in `case3` and `case7` respectively, demonstrating its effectiveness in maintaining legal placements even under more restrictive density conditions. We attribute this improvement to our bin-density-aware placement strategy in TIMBER, which actively monitors the current bin occupation levels during candidate insertion. If a prospective grid exceeds the bin density threshold ϵ, TIMBER redirects placement to less congested regions, thereby avoiding violations while maintaining placement quality.

D. Parallel Efficiency

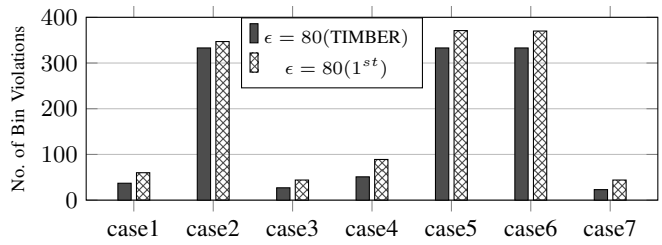
For our final experiment, we implemented a parallel version of TIMBER and evaluated its runtime performance and solution quality at different numbers of CPU threads [19], [20]. Specifically, we partitioned the placement region into fixed-size grids, matching the dimensions of the bin structure. This design ensures that each grid becomes an

TABLE III: Comparison between TIMBER and the first-place solution (1^{st}) of 2024 CAD Contest and TIMBER based on normalized power, area, and timing, as well as bin density violations (BDVs), memory, runtime, and final score values. All values are generated by the official checker program of 2024 CAD Contest.

Circuit	Power		Area		Timing		BDV		Memory(MB)		Runtime(s)		Final Score	
	1^{st}	TIMBER	1^{st}	TIMBER	1^{st}	TIMBER	1^{st}	TIMBER	1^{st}	TIMBER	1^{st}	TIMBER	1^{st}	TIMBER
case1	0.054	0.074	0.230	0.240	0.016	0.028	2	0	262.444	67.092	2.012	1.120	9.39e8	**7.43e8**
case2	0.089	0.025	0.002	0.00007	0.001	0.014	0	0	728.704	126.692	25.66	1.720	**7.47e5**	1.43e6
case3	0.002	0.009	0.00046	0.006	0.234	0.022	1	0	234.684	65.956	2.135	1.680	8.29e8	**7.33e8**
case4	0.022	0.038	0.300	0.412	0.006	0.003	14	0	250.040	67.068	1.898	1.980	1.43e9	**3.46e7**
case5	0.021	0.025	0.00006	0.00007	0.094	0.142	6	0	649.640	126.648	26.21	1.770	6.15e8	**1.59e7**
case6	0.035	0.025	0.001	0.00007	0.011	0.014	5	0	363.984	126.724	3.445	1.770	5.56e8	**5.65e7**
case7	0.002	0.017	0.00049	0.003	0.190	0.054	1	0	234.952	66.016	2.133	1.960	8.28e8	**7.33e8**
Geomean	**0.017**	0.025	0.00278	**0.00255**	0.025	**0.022**	-	-	348.431	**87.689**	4.548	**1.690**	3.03e8	**3.34e7**

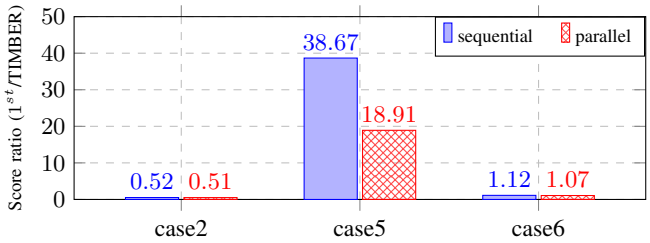


(a) No. of BDVs when $\epsilon = 85$.

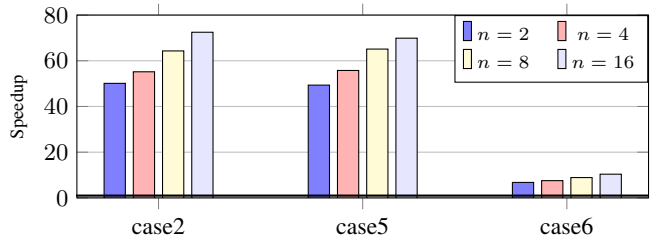


(b) No. of BDVs when $\epsilon = 80$.

Figure 8: Comparison of BDVs between TIMBER and the baseline across all testcases under different bin density thresholds (ϵ). Regardless of the bin density threshold, TIMBER always achieves fewer BDVs than the baseline.



(a) Score ratio (1^{st}/TIMBER) for sequential and parallel.



(b) Speedup of TIMBER for different thread counts (n).

Figure 9: Parallel efficiency of TIMBER on three largest circuits, `case2`, `case5`, and `case6`. (a) Impact on the final score, and (b) runtime speedup over the baseline on different thread counts. Although parallelization may slightly degrade the overall score, our result still outperforms the baseline while achieving a significant speedup up to 72.49×.

isolated instance of the placement problem, albeit on a smaller scale, with registers constrained to remain within their assigned grid. Consequently, each grid can be processed independently by a separate thread executing the core TIMBER algorithm. While this reduces the effective search space, it allows for massive parallelism with minimal compromise in solution quality.

Figure 9a presents the ratio of the final score of the baseline to that of TIMBER under both sequential and parallel implementations. Here, we focus on the three largest testcases in the benchmark, `case2`, `case5`, and `case6`. The parallel version of TIMBER exhibits comparable performance to the sequential implementation in `case2` and `case6`, and still achieves a substantial improvement of 18.91× in final score over the baseline in `case5`. The corresponding runtime improvements are shown in Figure 9b. For `case6`, the parallel implementation with 16 threads achieves a 10.37× speedup. On the other hand, TIMBER achieves 72.49× and 69.89× speedup for `case2` and `case5`, respectively, compared to the single-threaded baseline. We attribute this significant speedup to TIMBER's parallel design, where each thread independently processes a distinct region of the die without expensive synchronization. This design makes TIMBER highly scalable and well-suited for benchmarks with larger numbers of registers and increased placement complexity.

V. Conclusion

In this work, we presented TIMBER, a fast algorithm for multi-bit flip-flop (MBFF) banking and debanking. Aimed at addressing the limitations of current approaches, TIMBER introduces a bin-density-aware placement strategy that jointly minimizes power, area, TNS, and bin density violations. By incorporating a multi-threaded parallelization scheme and a region-based partitioning strategy, TIMBER also achieves significant runtime acceleration. Our evaluation on the official 2024 CAD Contest benchmarks highlights TIMBER's ability to outperform the contest's first-place winner. Inspired by our research [23]–[38], we plan to enhance the performance of TIMBER using GPU.

Acknowledgment

The authors acknowledge team cadb0027 in 2024 ICCAD CAD Contest for sharing their winning solution binary. This project is supported by NSF grants 2235276, 2349144, 2349143, 2349582, and 2349141. This research was also partially conducted by ACCESS – AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.

REFERENCES

[1] S. Borkar, "Design Challenges of Technology Scaling," IEEE Micro, vol. 19, no. 4, pp. 23–29, 1999.

[2] K. Usami and M. Horowitz, "Clustered voltage scaling technique for low-power design," Proceedings of the International Symposium on Low Power Electronics and Design, 1995.

[3] Y. -T. Shyu, J. -M. Lin, C. -P. Huang, C. -W. Lin, Y. -Z. Lin and S. -J. Chang, "Effective and Efficient Approach for Power Reduction by Using Multi-Bit Flip-Flops," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 4, pp. 624-635, April 2013, doi: 10.1109/TVLSI.2012.2190535.

[4] Y. Ye, S. Borkar, and V. De, "A new technique for standby leakage reduction in high-performance circuits," Symposium on VLSI Circuits, 1998.

[5] J. Yang and T. Kim, "Debanking Techniques on Multi-bit Flip-flops for Reinforcing Useful Clock Skew Scheduling," 2023 IEEE 36th International System-on-Chip Conference (SOCC), Santa Clara, CA, USA, 2023, pp. 1-6, doi: 10.1109/SOCC58585.2023.10256966.

[6] Sheng-Wei Yang, Jhih-Wei Hsu, Ting-Wei Lee, Tzu-Hsuan Chen, and Chin-Fang Cindy Shen. 2025. 2024 ICCAD CAD Contest Problem B: Power and Timing Optimization Using Multibit Flip-Flop. In Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24). Association for Computing Machinery, New York, NY, USA, Article 197, 1–6. https://doi.org/10.1145/3676536.3689911

[7] Sheng-Wei Yang, Jhih-Wei Hsu, Ting-Wei Lee, Tzu-Hsuan Chen, and Chin-Fang Cindy Shen.2024 ICCAD CAD Contest Problem B Slides. https://docs.google.com/presentation/d/1CA62_VESTlL9MAsQrAejqOw FeXowiMf6

[8] N. H. E. Weste and D. M. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed. Boston, MA, USA: Addison-Wesley, 2011.

[9] S. Natarajan et al., "A 14nm logic technology featuring 2nd-generation FinFET, air-gapped interconnects, self-aligned double patterning" in IEEE Int. Electron Devices Meeting (IEDM), San Francisco, CA, USA, Dec. 2014, pp. 3.7.1–3.7.3.

[10] Q. Zhou, J. Hu and Q. Zhou, "An effective iterative density aware detailed placement algorithm," 2014 IEEE International Symposium on Circuits and Systems (ISCAS), Melbourne, VIC, Australia, 2014, pp. 1444-1447, doi: 10.1109/ISCAS.2014.6865417.

[11] W.-K. Chow, C.-W. Pui, and E. F. Y. Young. Legalization algorithm for multiple-row height standard cell design. In Proceedings of the 53rd Annual Design Automation Conference, DAC '16, pages 83:1–83:6, 2016.

[12] Gang Wu, Yue Xu, Dean Wu, Manoj Ragupathy, Yu-yen Mo, and Chris Chu. 2016. Flip-flop clustering by weighted K-means algorithm. In Proceedings of the 53rd Annual Design Automation Conference (DAC '16). Association for Computing Machinery, New York, NY, USA, Article 82, 1–6. https://doi.org/10.1145/2897937.2898025

[13] Ya-Chu Chang, Tung-Wei Lin, Iris Hui-Ru Jiang, and Gi-Joon Nam. 2019. Graceful Register Clustering by Effective Mean Shift Algorithm for Power and Timing Balancing. In Proceedings of the 2019 International Symposium on Physical Design (ISPD '19). Association for Computing Machinery, New York, NY, USA, 11–18. https://doi.org/10.1145/3299902.3309753

[14] Meng-Yun Liu, Yu-Cheng Lai, Wai-Kei Mak, and Ting-Chi Wang. 2022. Generation of Mixed-Driving Multi-Bit Flip-Flops for Power Optimization. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22). Association for Computing Machinery, New York, NY, USA, Article 75, 1–9. https://doi.org/10.1145/3508352.3549473

[15] Y.-T. Chang, C.-C. Hsu, M. P.-H. Lin, Y.-W. Tsai, and S.-F. Chen, "Post-placement power optimization with multi-bit flip-flops," in Proc. IEEE/ACM ICCAD, 2010, pp. 654–661.

[16] C.-C. Hsu, Y.-C. Chen, and M. P.-H. Lin, "In-placement clock-tree aware multi-bit flip-flop generation for power optimization," in Proc. IEEE/ACM ICCAD, 2013, pp. 474–481.

[17] M. P.-H. Lin, C.-C. Hsu, and Y.-T. Chang, "Recent research in clock power saving with multi-bit flip-flops," in Proc. IEEE MWSCAS, 2011, pp. 1–4.

[18] H. Moon and T. Kim, "Design and allocation of loosely coupled multi-bit flip-flops for power reduction in post-placement optimization," in Proc. IEEE ASPDAC, 2016, pp. 1–6.

[19] Tsung-Wei Huang, Dian-Lun Lin, Chun-Xun Lin, and Yibo Lin, "Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System," IEEE Transactions on Parallel and Distributed Systems (TPDS), vol. 33, no. 6, pp. 1303-1320, June 2022

[20] Tsung-Wei Huang, Chun-Xun Lin, Guannan Guo, and Martin Wong, "Cpp-Taskflow: Fast Task-based Parallel Programming using Modern C++," IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 974-983, Rio de Janeiro, Brazil, 2019

[21] Tsung-Wei Huang and Martin Wong, "OpenTimer: A High-Performance Timing Analysis Tool," IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 895-902, Austin, TX, 2015

[22] Tsung-Wei Huang, Guannan Guo, Chun-Xun Lin, and Martin Wong, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol. 40, no. 4, pp. 776-789, April 2021

[23] Zizheng Guo, Tsung-Wei Huang, and Yibo Lin, "Accelerating Static Timing Analysis using CPU-GPU Heterogeneous Parallelism," IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems (TCAD), vol. 32, no. 12, pp. 4973-4984, Dec. 2023

[24] Guannan Guo, Tsung-Wei Huang, Yibo Lin, and Martin Wong, "GPU-accelerated Critical Path Generation with Path Constraints," IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Germany, 2021

[25] Zizheng Guo, Tsung-Wei Huang, and Yibo Lin, "HeteroCPPR: Accelerating Common Path Pessimism Removal with Heterogeneous CPU-GPU Parallelism," IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Germany, 2021

[26] Guannan Guo, Tsung-Wei Huang, Yibo Lin, and Martin Wong, "GPU-accelerated Path-based Timing Analysis," IEEE/ACM Design Automation Conference (DAC), CA, 2021

[27] Zizheng Guo, Tsung-Wei Huang, and Yibo Lin, "GPU-accelerated Static Timing Analysis," IEEE/ACM International Conference on Computer-aided Design (ICCAD), CA, 2020

[28] Shiju Lin, Guannan Guo, Tsung-Wei Huang, Weihua Sheng, Evangeline Young, and Martin Wong, "GCS-Timer: GPU-Accelerated Current Source Model Based Static Timing Analysis," ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, 2024

[29] Zizheng Guo, Tsung-Wei Huang, Jin Zhou, Cheng Zhuo, Yibo Lin, Runsheng Wang, and Ru Huang, "Heterogeneous Static Timing Analysis with Advanced Delay Calculator," IEEE/ACM Design, Automation and Test in Europe Conference (DATE), Valencia, Spain, 2024

[30] Yi-Hua Chung, Shui Jiang, Wan Luan Lee, Yanqing Zhang, Haoxing Ren, Tsung-Yi Ho, and Tsung-Wei Huang, "SimPart: A Simple Yet Effective Replication-aided Partitioning Algorithm for Logic Simulation on GPU," International European Conference on Parallel and Distributed Computing (Euro-Par), Dresden, Germany, 2025

[31] Jie Tong, Wan-Luan Lee, Umit Yusuf Ogras, and Tsung-Wei Huang, "Scalable Code Generation for RTL Simulation of Deep Learning Accelerators with MLIR," International European Conference on Parallel and Distributed Computing (Euro-Par), Dresden, Germany, 2025

[32] Wan-Luan Lee, Shui Jiang, Dian-Lun Lin, Che Chang, Boyang Zhang, Yi-Hua Chung, Ulf Schlichtmann, Tsung-Yi Ho, and Tsung-Wei Huang, "iG-kway: Incremental k-way Graph Partitioning on GPU," ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, 2025

[33] Wan-Luan Lee, Dian-Lun Lin, Cheng-Hsiang Chiu, Ulf Schlichtmann, and Tsung-Wei Huang, "HyperG: Multilevel GPU-Accelerated k-way Hypergraph Partitioner," IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, 2025

[34] Boyang Zhang, Che Chang, Cheng-Hsiang Chiu, Dian-Lun Lin, Yang Sui, Chih-Chun Chang, Yi-Hua Chung, Wan-Luan Lee, Zizheng Guo, Yibo Lin, and Tsung-Wei Huang, "iTAP: An Incremental Task Graph Partitioner for Task-parallel Static Timing Analysis," IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, 2025

[35] Boyang Zhang, Dian-Lun Lin, Che Chang, Cheng-Hsiang Chiu, Bojue Wang, Wan-Luan Lee, Chih-Chun Chang, Donghao Fang, and Tsung-Wei Huang, "G-PASTA: GPU Accelerated Partitioning Algorithm for Static Timing Analysis," ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, 2024

[36] Wan-Luan Lee, Dian-Lun Lin, Tsung-Wei Huang, Shui Jiang, Tsung-Yi Ho, Yibo Lin, and Bei Yu, "G-kway: Multilevel GPU-Accelerated k-way Graph Partitioner," ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, 2024

[37] Wan-Luan Lee, Dian-Lun Lin, Shui Jiang, Cheng-Hsiang Chiu, Yibo Lin, Bei Yu, Tsung-Yi Ho, and Tsung-Wei Huang, "G-kway: Multilevel GPU-Accelerated k-way Graph Partitioner using Task Graph Parallelism," ACM Transactions on Design Automation of Electronic Systems (TODAES), 2025

[38] Chedi Morchdi, Cheng-Hsiang Chiu, Wan-Luan Lee, Tsung-Wei Huang, and Yi Zhou, "Enhancing Graph Partitioning with Reinforcement Learning-based Initialization", IEEE High-performance and Extreme Computing Conference (HPEC), virtual, 2025