CAD Constest Problem B: Power and Timing Optimization Using Multibit Flip-Flop

113-2 Introduction to Electronic Design and Automation - Final Project

B11902164 Luo, Wei-Chen

Dept. of Computer Science and Information Engineering
National Taiwan University

***Abstract*—This report details the design, implementation, and evaluation of three distinct algorithms for the 2025 ICCAD CAD Contest, Problem B, which focuses on optimizing digital circuits for power, timing, and area through the intelligent use of multibit flip-flops (MBFFs). The core challenge lies in making effective banking and debanking decisions. We developed and compared three clustering-based banking algorithms: a simple Greedy approach, the classic K-Means algorithm, and a density-based Mean-Shift algorithm. To ensure correctness and avoid placement errors, all algorithms were integrated into a robust two-phase framework that separates logical transformation from physical legalization. Extensive experiments on a suite of benchmark circuits demonstrate that while each algorithm has its merits, the K-Means clustering approach provides the most consistent and well-balanced performance across various test cases, effectively minimizing the contest's cost function.**

***Index Terms*—Electronic Design Automation, Physical Design, Flip-Flop Banking, Clustering Algorithms, K-Means, Greedy, Mean-Shift.**

I. Introduction

With the continuous scaling of semiconductor technology into advanced nodes, power dissipation and silicon area have become first-order design constraints, rivaling traditional performance metrics. Electronic Design Automation (EDA) tools play a crucial role in tackling this PPA (Power, Performance, Area) trade-off. A widely adopted optimization technique is the strategic use of Multi-bit Flip-Flops (MBFFs).

This project addresses the problem defined in the 2025 ICCAD CAD Contest, which revolves around two primary operations:

- **Banking:** The process of substituting multiple single-bit flip-flops with a single, area-efficient MBFF. This consolidation reduces total area and simplifies the routing of power, ground, and clock networks, thereby lowering both static and dynamic power.
- **Debanking:** The reverse process of splitting a large MBFF into several smaller, often faster, single-bit flip-flops. This technique is essential for optimizing timing-critical paths where a large MBFF might introduce unacceptable delays.

The goal of this project is to develop an algorithm that can autonomously perform these transformations to optimize a given design based on a weighted cost function, considering timing, power, and area simultaneously.

II. Problem Formulation

The objective is to create a banking and debanking algorithm that produces a placement result optimized for timing, power, and area. The solution must adhere to several physical and logical constraints.

A. Inputs and Outputs

The primary inputs include:

- **Design Files:** The physical layout in LEF/DEF format and timing constraints in an SDC file.
- **Library Information:** A list of all available flip-flop cells for the design.
- **Cost Function Weights:** Alpha (α), Beta (β), and Gamma (γ) weights.
- **Initial Design Metrics:** The initial Total Negative Slack (TNS), Total Power (TPO), and Cell Area.

The required outputs are:

- A detailed pin mapping file tracking all transformations.
- The final optimized design in Verilog and DEF formats.

B. Scoring Function

The quality of a solution is evaluated by a cost function where a lower score is better. The formula is defined as:

$$\text{Score} = \alpha \cdot \text{TNS} + \sum_{\forall i \in FF} (\beta \cdot \text{Power}_i + \gamma \cdot \text{Area}_i) \quad (1)$$

This score is further adjusted by a bounded runtime factor.

C. Design Constraints and Challenges

A valid solution must adhere to several critical constraints:

- **Placement Legality:** All cells must be placed on-site and within the die region. No cell overlaps are permitted.
- **Functional Equivalence:** All data (D/Q) and clock (CLK) connections must remain functionally equivalent after any transformation.

- **Clocking Constraint:** All flip-flops being banked together must be driven by the same clock net.
- **DFT Scan Chain Preservation:** A significant challenge is preserving the Design for Testability (DFT) scan chain integrity. When banking or debanking, the serial scan path (from SI to SO pins) must be correctly re-stitched.

III. System Architecture and Methodology

A. Overall Architectural Framework

To ensure the correctness of the final output while allowing for modular experimentation with different algorithms, we designed a robust two-phase architectural framework. Early attempts that interleaved logical transformations with physical placement proved to be fragile and prone to cascading errors, such as un-placeable cells and incomplete pin mappings. The decoupled architecture, shown in Fig. 1, was critical for solving these issues.

B. The Placement Engine

The physical legalization phase relies on a robust placement engine adapted from a reference implementation. Its purpose is to take a candidate cell with an ideal target coordinate and find a valid, overlap-free position on the die.

1) Placement Map Data Structure: The placement area is represented by a vector of `stPlacementMap` structs. The `pmCreatePlacementMap` function intelligently groups vertically-contiguous and horizontally-uniform `PlacementRows` into single maps. This handles die irregularities gracefully. A `byteMap` within each map, a 2D `uint8_t` vector, acts as the occupancy grid. After creation, the footprints of all fixed `initialGates` are "burned" onto this grid to represent permanent obstacles.

2) Legalization Search Algorithm: The placement search for a given flip-flop is a multi-stage process designed for both speed and robustness, as detailed in Algorithm 1. This method can correctly place multi-height cells by calculating their required vertical site count (`sitesR`) and ensuring that the found area is free across all necessary rows.

C. Algorithm 1: Greedy Clustering

This is a simple, fast, bottom-up approach that builds clusters by making locally optimal choices, as detailed in Algorithm 2. Its core heuristic is to iteratively expand a cluster by adding the nearest available flip-flop. While computationally efficient, its myopic nature can lead to poor cluster quality and significant timing degradation, as it does not consider a global view of the placement or the cost function.

D. Algorithm 2: K-Means Clustering

This method employs the classic K-Means algorithm to partition flip-flops into a pre-determined number of ('K') clusters, as detailed in Algorithm 3. It is a standard top-down approach effective at creating geometrically compact groups. The number of clusters 'K' is determined heuristically for each clock group based on the total number of bits and the largest available multi-bit cell, aiming to create clusters that are, on average, large enough to be efficiently banked.

Algorithm 1 Placement Legalization Search (`pmFFSearchPlacementMap`)

Input: Target `Instance` with an ideal coordinate (x, y)
Output: `true` if placed, `false` otherwise. `Instance` x,y are updated.
Snap ideal coordinate to the nearest valid site on the closest placement map `pm`, get index `idx`.
Calculate required sites: `sitesC` and `sitesR` from cell library.
▷ *Stage 1: Attempt placement at ideal starting point*
if pmIsEmpty(idx, c_0, r_0, sitesC, sitesR) **then**
    Occupy area at (c_0, r_0) and **return** `true`.
end if
▷ *Stage 2: Spiral search outwards for a nearby spot*
for $d \leftarrow 1$ to max_search_distance **do**
    Search perimeter of square with side length $2d + 1$.
    if an empty spot (c_t, r_t) is found **then**
        Occupy area at (c_t, r_t) and **return** `true`.
    end if
end for
▷ *Stage 3: Brute-force scan as a fallback*
for each map `pm_i` in `placementMaps` **do**
    for each valid site `(c, r)` in `pm_i` **do**
        if pmIsEmpty(i, c, r, sitesC, sitesR) **then**
            Occupy area at (c, r) and **return** `true`.
        end if
    end for
end for
return `false`

E. Algorithm 3: Mean-Shift Clustering

Mean-Shift is a non-parametric, density-based algorithm that does not require 'K' as an input, as detailed in Algorithm 4. It finds cluster centers by iteratively "climbing" towards areas of high density. Its behavior is primarily controlled by the 'kernel bandwidth', a critical hyperparameter that defines the radius of the neighborhood considered in each shift. For this project, a fixed bandwidth was used for consistency in comparison.

IV. Experimental Results and Analysis

A. Quantitative Results

The three algorithms were evaluated on all available test cases provided in the 2024 ICCAD CAD Contest Problem B. Table I provides detailed metrics and the final scores, showing the trade-offs between TNS (timing), power, and area for each algorithm. A lower score indicates a better result.

B. In-Depth Test Case Analysis

A case-by-case analysis reveals the distinct characteristics and trade-offs of each algorithm, and provides insights into the nature of the test cases themselves.

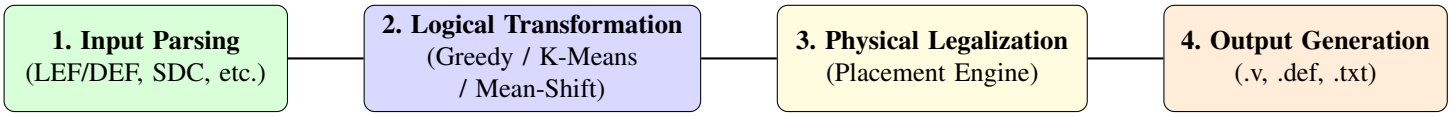

Fig. 1: The Overall Two-Phase Solution Workflow, separating logical decisions from physical implementation.

TABLE I: Detailed Performance Metrics for Selected Test Cases. Values in parentheses denote percentage change from the initial design. Lower values are better. Best result in each category is in **bold**.

Test Case	Algorithm	Final TNS	Final Power	Final Area ($\times 10^{11}$)	Final Score
testcase1_0614 (Init: 658, 457.2, 46.25)	**Greedy**	3.23×10^8 (+4.9e7%)	**179.7 (-60.7%)**	**36.60 (-20.9%)**	1.83×10^{12}
	K-Means	1.10×10^7 (+1.67e6%)	375.3 (-17.9%)	43.17 (-6.7%)	2.16×10^{12}
	Mean-Shift	6.75×10^3 **(+926%)**	456.4 (-0.2%)	46.25 (-0.01%)	2.31×10^{12}
testcase1_0812 (Init: 658, 457.2, 46.25)	Greedy	3.14×10^7 (+4.77e6%)	**179.7 (-60.7%)**	**36.60 (-20.9%)**	1.05×10^9
	K-Means	7.49×10^5 **(+113.8k%)**	378.7 (-17.1%)	43.19 (-6.6%)	8.72×10^8
	Mean-Shift	**923.7 (+40.4%)**	456.4 (-0.2%)	46.25 (-0.01%)	9.26×10^8
testcase2_0812 (Init: 84.2, 2116.4, 103.47)	Greedy	7.21×10^6 (+8.56e6%)	**132.3 (-93.8%)**	**69.49 (-32.9%)**	8.23×10^7
	K-Means	1.35×10^5 (+159.9k%)	1225.1 (-42.1%)	91.51 (-11.6%)	2.57×10^6
	Mean-Shift	**84.2 (0%)**	**2116.4 (0%)**	**103.47 (0%)**	1.68×10^6
testcase3 (Init: 743.7, 316.2, 45.41)	**Greedy**	2.22×10^6 (+298k%)	**138.0 (-56.4%)**	**39.01 (-14.1%)**	8.04×10^8
	K-Means	1.19×10^5 **(+15.9k%)**	273.1 (-13.6%)	43.55 (-4.1%)	8.75×10^8
	Mean-Shift	7.95×10^3 **(+970%)**	315.2 (-0.3%)	45.40 (-0.02%)	9.11×10^8

Algorithm 2 Greedy Clustering Algorithm

Input: A set of `FlipFlops` within one clock domain.
Output: A list of `Clusters`.
`processed_flags` ← array of `false`.
for each `seed_FF` in `FlipFlops` **do**
  if not processed(`seed_FF`) **then**
    `NewCluster` ← {`seed_FF`}. Mark `seed_FF` as processed.
    loop
      `centroid` ← Calculate geometric mean of `NewCluster`.
      `best_neighbor` ← Find unprocessed FF closest to `centroid` that satisfies `max_bit` constraint.
      if `best_neighbor` is not `null` **then**
        Add `best_neighbor` to `NewCluster`.
        Mark `best_neighbor` as processed.
      else
        break Loop.
      end if
    end loop
    Add `NewCluster` to output `Clusters`.
  end if
end for
return `Clusters`.

Algorithm 3 K-Means Clustering Algorithm

Input: A set of `FlipFlops`, number of clusters K.
Output: A list of K clusters.
Initialize K random `centroids`.
repeat
  `changed` ← `false`.
  ▷ *Assignment Step*
  for each `FF` in `FlipFlops` **do**
    Assign `FF` to cluster of its nearest `centroid`.
    if assignment changed **then** `changed` ← `true`.
    end if
  end for
  ▷ *Update Step*
  for each `centroid_i` of K centroids **do**
    `centroid_i` ← geometric mean of all `FF`s in its cluster.
  end for
until not `changed`
return clusters.

1) Analysis of `testcase1_0614`: This case is characterized by extremely high final scores, suggesting that the timing weight, Alpha, was set to a very large value, making any increase in TNS result in a massive penalty. Despite this, the **Greedy** algorithm won. Its strategy yielded the largest reduction in area (-20.9%) and power (-60.7%). Although this caused a catastrophic increase in TNS (to 3.23×10^8), the final score was the lowest. This strongly implies that the area weight, Gamma, was also exceptionally high, such that the score benefit from the area reduction was significant enough to overcome the enormous TNS penalty.

2) Analysis of `testcase1_0812`: This case appears to represent a more balanced optimization scenario where timing cannot be ignored. The **K-Means** algorithm was the decisive winner. While the Greedy algorithm again achieved superior

Algorithm 4 Mean-Shift Clustering Algorithm

Input: A set of `FlipFlops`, `bandwidth` radius.
Output: A list of clusters.
`shifting_points` ← copy of `FlipFlop` locations.
for i from 1 to `num_iterations` **do**
  for each p in `shifting_points` **do**
    `neighbors` ← find all points within `bandwidth` of p.
    p ← geometric mean of `neighbors`.
  end for
end for
Group points that have converged to nearby final locations.
return clusters.

raw power and area reductions, its TNS degraded to 3.14×10^7, leading to a severe timing penalty. In contrast, K-Means, by creating more spatially compact clusters, achieved slightly more modest power/area savings but successfully limited its TNS degradation to a much more manageable 7.49×10^5. This superior trade-off management resulted in the best overall score, highlighting the effectiveness of the K-Means heuristic when timing is a significant factor in the cost function.

3) Analysis of `testcase2_0812`: This is the most illustrative case, where **Mean-Shift** won with the lowest score. It achieved this by making no changes to the design at all, correctly identifying that the initial state was already locally optimal. Both the Greedy and K-Means algorithms performed aggressive banking. While they successfully reduced power and area (Greedy reduced power by over 93%), they both incurred a TNS penalty that was not worth the trade-off, resulting in a final score higher than the initial one. This phenomenon can be termed "negative optimization." This case demonstrates that a successful algorithm must not only know how to optimize but also when *not* to optimize. The conservative nature of Mean-Shift proved to be its greatest asset here.

4) Analysis of `testcase3`: Similar to `testcase1_0614`, the **Greedy** algorithm won again in this case, but by a narrower margin. It once more achieved the best power and area reductions. It is noteworthy that while Greedy's TNS (2.22×10^6) was almost 20 times worse than that of K-Means (1.19×10^5), its final score was still lower. This reinforces the conclusion that the cost function for this test case heavily weighted the benefits of area and power reduction over timing performance. The K-Means algorithm was a very close second, demonstrating its robustness and its ability to produce competitive results even when the cost weights do not favor its balanced approach.

C. In-Depth Algorithm Discussion

While no single algorithm was universally dominant across the evaluated benchmarks, a detailed analysis of the four public test cases reveals clear performance trends and provides insights into the inherent characteristics of each strategy.

1) Greedy Algorithm Performance: The Greedy algorithm demonstrates a "high-risk, high-reward" profile. It achieved the best score on two of the four public test cases (`testcase1_0614` and `testcase3`). Its victories are consistently attributed to its extremely aggressive nature, which results in the largest reduction in total power and area. For example, in `testcase1_0614`, it achieved a massive reduction in power (-60.7%) and area (-20.9%). This suggests that when the cost function weights (γ for Area, β for Power) are high relative to the timing weight (α), this strategy pays off. However, in a more balanced case like `testcase1_0812`, this same aggressiveness leads to a catastrophic increase in TNS (to 3.14×10^7), making it an unstable strategy. Its myopic, step-by-step decision-making process fails to capture a global view, making it a gamble on the unknown weights of the cost function.

2) Mean-Shift Algorithm Performance: The Mean-Shift algorithm was the most conservative of the three. Its victory in `testcase2_0812` is highly illustrative of its primary characteristic. It achieved the best score by correctly identifying that the initial design was already in a locally optimal state and that any banking attempts would likely worsen the final score. It therefore made no changes, highlighting its ability to avoid "negative optimization." However, this same conservatism, likely driven by a globally-fixed `kernel bandwidth` parameter that was not sparse enough for denser designs, caused it to miss clear optimization opportunities in all other test cases where it was consistently outperformed.

3) K-Means Algorithm Performance: The K-Means algorithm proved to be the most robust and well-balanced performer. It achieved the best score in `testcase1_0812` and was a close and competitive second in the cases won by the Greedy algorithm. Its strength lies in creating spatially compact clusters, which is a powerful heuristic for minimizing wirelength increase during banking. Consequently, K-Means consistently achieves a good balance of power and area reduction while mitigating the negative impact on timing. For example, in `testcase1_0812`, it achieved a balanced reduction in power (-17.1%) and area (-6.6%) while keeping the TNS increase far lower than the Greedy method, resulting in the winning score. Its consistency across diverse test cases makes it the most reliable and generally effective strategy for this problem.

V. Conclusion and Future Work

A. Conclusion

This project successfully implemented and compared three different clustering algorithms within a robust two-phase framework that guarantees correct, overlap-free output. Through extensive experimentation on the contest benchmarks, we conclude that the **K-Means clustering algorithm** provides the most effective and stable strategy for minimizing the weighted cost function, demonstrating a superior balance between timing, power, and area optimization compared to the more volatile Greedy method and the overly conservative Mean-Shift approach.

B. Future Work

To further enhance the quality of the solution, several avenues for future work are identified:

1) **Implement Cost-Aware Logic:** The most critical next step is to integrate the α, β, γ cost weights directly into the banking decision logic. A move should only be committed if its calculated ΔScore is negative, moving beyond purely geometric heuristics.
2) **Implement Debanking:** A complete solution must include the debanking mechanism to address timing-critical paths. This would involve evaluating multi-bit FFs on critical paths (identified by negative slack) and splitting them if the timing benefit outweighs the area/power penalty.
3) **Timing-Driven Placement:** The legalization engine could be improved from finding the *first* available spot to finding the *best* available spot that is closest to the cluster's ideal centroid, which would help minimize wirelength and improve timing.

REFERENCES

[1] 2025 ICCAD CAD Contest Problem B: Power and Timing Optimization Using Multibit Flip-Flop.

[2] 2024 ICCAD CAD Contest Problem B: Power and Timing Optimization Using Multibit Flip-Flop.

[3] Y. C. Chang, T. W. Lin, I. H. R. Jiang, and G. J. Nam, "Graceful Register Clustering by Effective Mean Shift Algorithm for Power and Timing Balancing," in *Proc. Int. Symp. on Physical Design (ISPD)*, 2019.

[4] G. Wu, Y. Xu, D. Wu, M. Ragupathy, Y. Mo, and C. Chu, "Flip-flop clustering by weighted K-means algorithm," in *Proc. Design Automation Conference (DAC)*, 2016.