PROBLEM B: POWER AND TIMING OPTIMIZATION USING MULTIBIT FLIP-FLOP

Meng-Chen Wu, Hung-Ling Liu, Min-Hsin Liu

1. Introduction

In modern technology nodes, minimizing power and area is crucial in semiconductor design. A common technique is replacing single-bit flip-flops with multibit flip-flops to save space and reduce routing complexity. However, this can sometimes negatively impact timing in critical paths, requiring us to revert to single-bit flip-flops, known as multibit flip-flop debanking.

In this contest, we will simulate banking and debanking decisions in virtual designs. Contestants must balance timing, power, and area to find the best optimization solutions for each testcase.

Next, we will introduce Previous Research, followed by our overall flow and the three-step proposed method to reduce cost: Cluster, Banking&Debanking, and Legalization. Then, we will present the experimental results, including observations and future works derived from these results. Finally, we will conclude with our findings.

2. Previous Research

To optimize timing and power of a given initial placement, many different approaches have been proposed. Wu *et al.* [1] proposed a framework of register clustering by weighted K-means following by register relocation to reduce the number of clock buffer and clock wire length, thus reduce the clock power and improve timing. Chang *et al.* [2] improved the classic mean shift method, which is an old pattern recognition method, to search for the stationary point of register clusters. Liu *et al.* [3] presented a multi-bit flip-flop (MBFF) generation approach considering mixed-driving MBFFs whose certain bits have a higher driving strength than the other bits. They utilized the timing slacks of flip-flop's pins to construct timing-feasible regions and found maximum intersection of those regions for further MBFF merging. We choose to develop our method mainly referring to [2] to find appropriate cluster points for registers, since the performance of their method is relatively good under most metrics. In the following we introduce necessary background and details of their work.

2.1 Classic Mean Shift

Classic mean shift was introduced by Fukunaga *et al.* [4], and applied to cluster analysis in various fields, e.g. Computer Vision [5]. Given n data points $\mathbf{x}_i, i = 1, ..., n$ in the d-dimensional space R^d, for doing clustering, we are interested only in a special class of radially symmetric kernels (*density functions*),

$$K(\mathbf{x}) = c \cdot k(||\mathbf{x}||^2), \tag{1}$$

where the univariate function $k(x)$ is called the *profile* of the kernel function, with only $x \geq 0$. c is the normalization constant which makes $K(\mathbf{x})$ integrate to one. In this project we use the Gaussian kernel with profile

$$k(x) = e^{-\frac{1}{2}x}, \tag{2}$$

the corresponding kernel density estimator (*density surface*) with one bandwidth parameter h is

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \\ &= \frac{c}{nh^d} \sum_{i=1}^{n} k\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right). \end{aligned} \tag{3}$$

Considering each data point $\mathbf{x}_i$ as the coordinate of a flip-flop, local maximums of $f(\mathbf{x})$ can be viewed as proper cluster points for nearby flip-flops. For each flip-flop, to find its corresponding cluster, we iteratively perform gradient ascent starting from its original coordinate to find the local maximum it belongs to. The gradient of $f(\mathbf{x})$ is

$$\begin{aligned} \nabla f(\mathbf{x}) &= \frac{c}{nh^d} \sum_{i=1}^{n} \nabla k\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \\ &= \frac{c}{nh^d} \sum_{i=1}^{n} k'\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \cdot \nabla\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \\ &= \frac{2c}{nh^{d+2}} \sum_{i=1}^{n} k'\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \cdot (\mathbf{x} - \mathbf{x}_i). \end{aligned} \tag{4}$$

Notice that $k'(x)$ is the total derivative of univariate function $k(x)$. For convenience, we define an univariate function $g(x)$ as

$$g(x) = -k'(x). \tag{5}$$

Then the gradient of density surface $f(\mathbf{x})$ can be written as

$$\begin{aligned}
&\nabla f(\mathbf{x}) \\
&= \frac{2c}{nh^{d+2}} \sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right) \cdot (\mathbf{x}_i - \mathbf{x}) \\
&= \frac{2c}{nh^{d+2}} \left(\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)\mathbf{x}_i - \sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)\mathbf{x}\right) \quad (6) \\
&= \frac{2c}{nh^{d+2}} \sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right) \left[\frac{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)\mathbf{x}_i}{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}\right].
\end{aligned}$$

The *mean shift vector* is defined as

$$\mathbf{m}(\mathbf{x}) = \frac{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)\mathbf{x}_i}{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x}. \quad (7)$$

Observe that $g(x)$ is always positive. For an example, the Gaussian kernel $k(x) = e^{-\frac{1}{2}x}$ results in $g(x) = -k'(x) = \frac{1}{2}e^{-\frac{1}{2}x}$. Therefore, mean shift vector $\mathbf{m}(\mathbf{x})$ can be viewed as $\nabla f(\mathbf{x})$ being divided with a positive factor, thus we can directly use $\mathbf{m}(\mathbf{x})$ to perform gradient ascent process on the density surface to find the proper local maximum for $\mathbf{x}$, in other words, for the initial coordinate of each flip-flop.

2.2 Effective Mean Shift

Chang *et al.* [2] mainly proposed two improvements for applying classic mean shift algorithm on register clustering problem, we will introduce them in sections below.

1) Identifying Effective Neighbors

Computing the mean shift vector of input coordinate $\mathbf{y}^t$ is equivalent to directly computing the shifted coordinate $\mathbf{y}^{t+1}$, where

$$\mathbf{y}^{t+1} = \mathbf{y}^t + \mathbf{m}(\mathbf{y}^t) = \frac{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{y}^t-\mathbf{x}_i}{h}\right\|^2\right)\mathbf{x}_i}{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{y}^t-\mathbf{x}_i}{h}\right\|^2\right)}. \quad (8)$$

In equation (8), we need to compute the weighted mean of n flip-flop initial coordinates, which is tended to be a pretty large number, e.g. at most 260k flip-flops in ICCAD-2015 CAD contest incremental timing-driven placement benchmark suite [2]. However, for $\mathbf{x}_i$ which is far from $\mathbf{y}^t$, its contribution to $\mathbf{y}^{t+1}$ is extremely small, thus they identify K Nearest Neighbor (KNN) of each flip-flop at the beginning and only consider the weighted mean of KNN flip-flops when calculating $\mathbf{y}^{t+1}$.

2) Variable Bandwidth Selection

In classic mean shift, there is only one bandwidth parameter h in density function $f(\mathbf{x})$, but since we prefer making larger cluster as long as the displacement delay induced is acceptable, assigning variable bandwidth to each flip-flop respectively will be more reasonable. For an example, if there is an isolated flip-flop and we can accept quite large displacement, setting larger bandwidth will help us explore the opportunity to cluster this isolated flip-flop. Furthermore, they suggest to take slack information of flip-flop into bandwidth setting consideration because smaller bandwidth will prevent the flip-flop from clustering, which is a good behavior for timing critical flip-flops. Combining above two improvements, the shifted coordinate equation of effective mean shift is

$$\mathbf{y}^{t+1} = \frac{\sum_{i\in KNN'(\mathbf{y}^t)} \frac{\mathbf{x}_i}{h_i^{d+2}} g\left(\left\|\frac{\mathbf{y}^t-\mathbf{x}_i}{h_i}\right\|^2\right)}{\sum_{i\in KNN'(\mathbf{y}^t)} \frac{1}{h_i^{d+2}} g\left(\left\|\frac{\mathbf{y}^t-\mathbf{x}_i}{h_i}\right\|^2\right)}, \quad (9)$$

where $KNN'(\mathbf{y}^t)$ is the KNN of $\mathbf{y}^t$ after eliminating some distant neighbor, h_i is the bandwidth for flip-flop $\mathbf{x}_i$.

3. Proposed Method

We propose a three-step method (Fig. 1) to reduce the total cost of input placement. First, we adopt effective mean shift algorithm to find register clusters. Second, flip-flop banking and debanking is performed within clusters. Lastly, to satisfy place-on-site and non-overlapping constraint of the contest problem, we conduct breadth-first search (BFS) based legalization of flip-flops considering slack importance. We describe the details of each step in following sections.

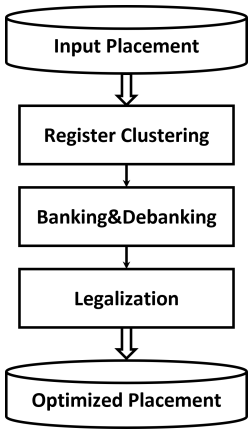

Fig. 1: Overall flow

3.1 Register Clustering

In this step, we take the initial coordinate of all flip-flops as input, and conduct register clustering by effective mean shift. The output is clusters of registers. We divide our register clustering procedure into four section.

1) Finding KNN of Each Flip-flop

A brute force method is for each flip-flop, compute the Euclidean distance of every pair of flip-flops, and select $k_{neighbor}$ flip-flops with the smallest distance as KNN. By brief testing, this method takes about 7000 second for $n = 300,000$, which is clearly not acceptable. Therefore, we use the R tree data structure to store the 2-dimensional data points and managed to find KNN rather efficiently. For $n = 300,000$, the R tree method only takes 5 second.

2) Timing Aware Bandwidth Setting

We set the bandwidth h_i according to below equation,

$$h_i = min(h_{max},\ c_h r_h(s)\left\|\mathbf{x}_i - \mathbf{x}_{i,k_{neighbor}}\right\|). \tag{10}$$

h_{max} is the bandwidth upper bound for all flip-flops, set this value according to maximum displacement allowed. The second entry of min function is the variable bandwidth of each flip-flop. $\left\|\mathbf{x}_i - \mathbf{x}_{i,k_{neighbor}}\right\|$ is the maximum Euclidean distance of $\mathbf{x}_i$ and all flip-flops in its KNN. c_h is a self-defined scaling constant. $r_h(s)$ is the timing aware bandwidth ratio funtion taking slack s of flip-flop $\mathbf{x}_i$ as input. $r_h(s)$ is defined as below,

$$\begin{aligned} r_h(s) &= log_b\left(b^{r_{h,min}} + \frac{(v(s) - s_{min})}{s_{max} - s_{min}} \cdot (b^{r_{h,max}} - b^{r_{h,min}})\right), \\ v(s) &= min(\ max(s_{min},\ s),\ s_{max}\ ). \end{aligned} \tag{11}$$

$v(s)$ is the function mapping s to valid range $[s_{min},\ s_{max}]$. $[r_{h,min},\ r_{h,max}]$ is the range of $r_h(s)$. b is the base of log function. Take Fig. 2 as an example, $r_h(s)$ maps the slack of [-5, 50] to bandwidth ratio of [0.01, 2], other value of slack is truncated to between [-5, 50]. We choose log function to reflect the criticality of small slack, when slack is close to s_{min}, the ratio $r_h(s)$ will decrease faster, and when slack is rather large, the ratio grows slower to prevent too much displacement.

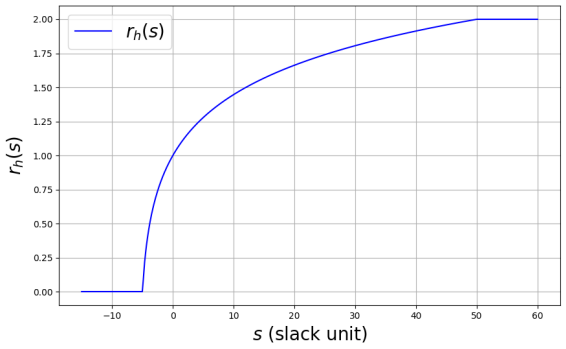

Fig. 2: Timing critical bandwidth function, $r_h(s)$

3) Shift to Local Maximum

For each flip-flop, we perform the gradient ascent process by computing equation (9) iteratively until $|\mathbf{y}^{t+1} - \mathbf{y}^t| < \delta$, where δ is a self-defined threshold.

4) Identify Cluster of Register

We arbitrarily choose the shifted coordinate of one register as the cluster center, and traverse all register in some certain order, if a register is not clustered and its shifted coordinate has distance $\leq \epsilon$ with the current cluster center, then it joins the cluster.

3.2 Banking&Debanking

Given many clusters and their flip-flops, bank or debank the flip-flops based on slack information. We divide this process into four simple stages:

1) Classify

Considering that flip-flops in different clock domains cannot be banked together, we first classify them. Separate the flip-flops in the same cluster according to their clock domains.

For example, in the Fig.3, reg1 and reg3 belong to the same clock domain, while reg2 is different. Thus, banking {reg1,reg2} and banking {reg2,reg3} are invalid banking combinations.

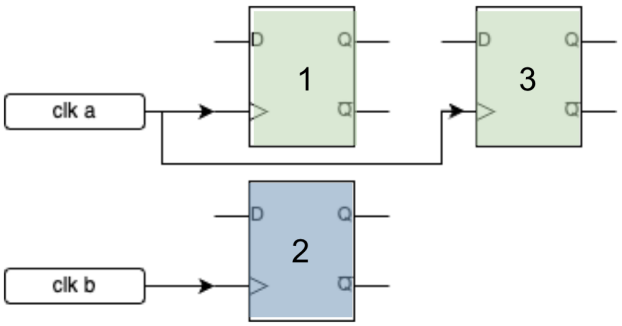

Fig. 3: Classify flipflops by clock domains

2) Debanking

Identify flip-flops with any pins having negative slack and debank them into single-bit flip-flops.

3) Collecting

After debanking, some single-bit flip-flops might have positive slack. To reduce costs, we collect these flip-flops for banking in the next stage.

4) Multi-level Banking

For these flip-flops, we use a multilevel approach, starting from level 1. At level n, merge the flip-flops into 2^n bits. The maximum value of n depends on the largest multibit size provided by the library. If the slack remains positive after banking, continue banking the flip-flop at the next level until any pin's slack becomes negative.

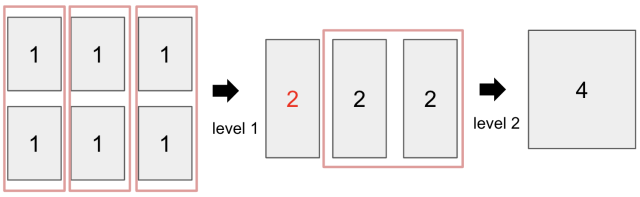

Fig. 4: Multi-level banking

3.3 Legalization

1) Definition of legal

After clustering and banking/debanking, the flip-flops may not be on the legal position. An instance is legal if its lower left corner aligns to the site grid of the placement rows. In Fig.5, the green instances are placed on-site, which is legal. As for the red instances, they are placed on the illegal locations.

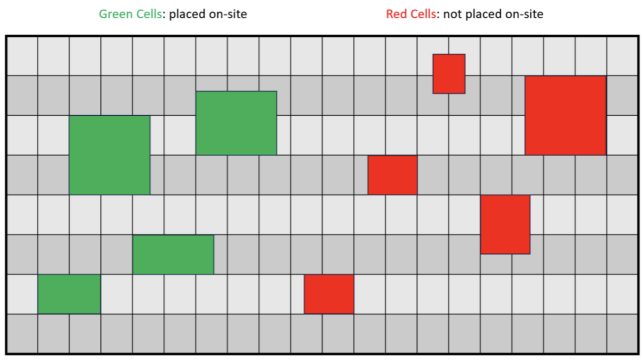

Fig. 5: Legal Explanation

2) Flow of legalization

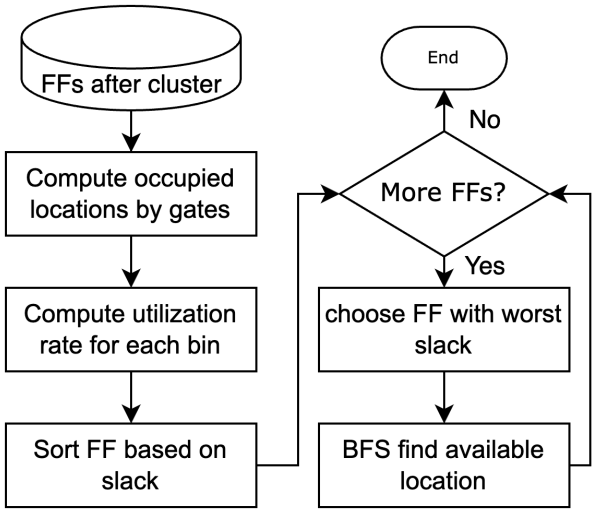

Fig. 6: Legalization Flow

The overall flow for legalization is shown in Fig.6. Given a set of flip-flops after clustering and banking/debanking, we first build a matrix to record if a location is occupied or not. Then, we build another matrix to record the utilization rates for each bin, which is related to the cost function. At this point, we only consider gates for these two matrices because the locations for all flip-flops will be decided later. After we build the two matrices, we sort all flip-flops based on their slack and start to iterate all flip-flops. In each iteration, we choose a flip-flop with the worst slack, and give this flip-flop an exact legal location with breadth-first search method, which will be elaborated in the next part. When all locations of flip-flops are decided, we finish the legalization process and parse output file.

3) BFS for legalization

To put a flip-flop on a legal location, we modify classic breadth-first search. Given a ideal location which may be illegal, we first find the nearest location. A location is unavailable if this location is occupied by other instances or putting the flip-flop at the location violates utilization rate of some bin. We decide a location is available by maintaining two matrices mentioned in 3.3-(2). If a location is unavailable, we then consider four location around it. By doing it recursively, we can finally get a legal location for a flip-flop. Here's is a simple example. In Fig.7, the gray block means the fixed instances. Suppose that max utilization rate of a bin is 50% and there is a ideal location P1 for a flip-flop. Since P1 is not on grid, we find the nearest legal location P2. However, if we put it at P2, it will violate utilization rate of Bin 2. Thus, we then take the four location around into consideration, which are P3 P6. P3 violates utilization rate of Bin 2 as well. P4 and P5 both are occupied by other instances. Only P6 meets all the constraints. Thus, we assign this location to the flip-flop and updates the two matrices accordingly. The final location is the green block in Fig.7.

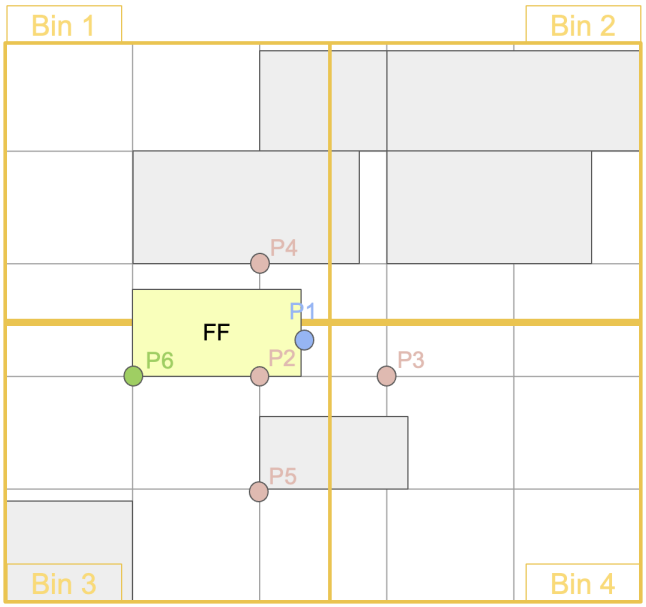

Fig. 7: BFS Example

4. Experimental Results

4.1 Setting

Meanshift parameter:

- $k_{neighbor}$: 140
- h_{max}: 1e+04
- c_h: 100
- s_{min}: -50
- s_{max}: 300
- $r_{h,min}$: 1e-06
- $r_{h,max}$: 2

The following experiments are based on the only released test case from ICCAD contest.

4.2 Evaluation Function

$$Cost = \sum_{i \in \text{FF}} (\alpha \cdot \text{TNS}(i) + \beta \cdot \text{Power}(i) + \gamma \cdot \text{Area}(i)) + \lambda \cdot D \tag{12}$$

In this formula:

- α: the weight for TNS,
- β : the weight for Power,
- γ : the weight for Area,
- λ : the weight for D,
- D : the number of bins that violates the utilization rate threshold

For the following experiments, the values for each of them are 1, 5, 5,and 10.

4.3 Experiments

In our implementation, the slack limit decides if we can bank this flip-flop with another. Since it related to the banking decision, the result would be directly affected by it. Thus, we try different value for slack limit to see how the results change with slack limit. The results of experiments are shown in Table 1. Since the smaller slack limit means that flip-flops are more easily to be banked, the numbers of multi-bit flip-flops become larger when the slack limit is smaller. Furthermore, the multi-bit flip-flops usually consumes less power in cost of larger delay so that the power is less and TNS is worse when the slack limit is smaller. From the experiments, we can observe that if we don't bank any flip-flops, the result is the best. The reason is that the current test case has many defects which makes the area cost dominates the others. We can directly change the type of all flip-flop to the type of one bit flip-flop with smallest area. In general, the slack limit of a value near zero is more reasonable. Thus, to evaluate our method, we think that we can choose slack limit to be zero.

Our results are shown in Table 2. Compared to the input data, we achieve about 91% cost reduction when slack limit equals to 1000. However, our total negative slack and power are both increase. Only cost of area is reduced in this

Slack limits	-1000	-100	-10	-1	0	1	10	100	1000
# of 1 bit ff	905	953	2543	2643	2705	3003	5249	14561	19879
# of 2 bit ff	839	883	1884	2288	2321	2250	1787	597	0
# of 4 bit ff	4324	4290	3392	3165	3133	3094	2764	1031	0
TNS	330289	329439	283365	262297	258798	258903	242035	233659	232045
Power	920.447	932.691	1291.78	1349.10	1362.49	1403.45	1719.48	3104.09	3903.24
Area	1.75e+09	1.74e+09	1.63e+09	1.62e+09	1.62e+09	1.59e+09	1.41e+09	6.46e+08	2.13e+08
Total cost	1.75e+09	1.74e+09	1.63e+09	1.62e+09	1.62e+09	1.59e+09	1.41e+09	6.47e+08	2.13e+08

Table 1: Slack Limits and Corresponding Data

	TNS	Power	Area	D	Initial score	Runtime
Input Data	657.88	2285.93	2.30e+09	0	2.30e+09	6.27s
Best Result	232045	3903.24	2.13e+08	0	2.13e+08	
Reasonable Result	258798	1362.49	1.62e+09	0	1.62e+09	

Table 2: Final Result

case. From the result, we can clearly see that the area impact dominates the others so that we can achieve such improvement. It is worth noticing that the increment of TNS. Since the current released test case has some unclear and defective parts, the TNS calculation may not be as accurate as it should be. Another problem in the test case is that the cost factors for TNS, power, and area make the final cost totally dominated by the area cost. It is unrealistic because timing constraints are critical for most designs. To evaluate our method effectively, we choose slack limit to be zero, which is more reasonable. In this case, we can achieve about 30% cost reduction. Only TNS becomes worse while power and area are both improved.

5. Observation

Regarding the testcase and cost function, we noted some observations and inconsistencies:

1. The positions of flip-flop pins are unusual, extending beyond the flip-flop's boundaries.
2. Some gates have output wires without input wires.
3. The displacement delay in the test data is too large, making slack calculations sensitive to small displacements, which significantly increases TNS.
4. The coefficients in the cost function are not balanced, causing the area to dominate the cost and minimizing the impact of other factors. In reality, TNS can significantly affect subsequent timing convergence.

6. Future work

Ignoring the aforementioned issues, we believe there are two points for further improvement to make our algorithm more robust:

6.1 Consider the Impact of the Cost Function

Consider the Impact of the Cost Function for Timing, Area, and Power.Each test case has different weightings (alpha, beta, gamma). We can prioritize based on these coefficients or pre-calculate cost changes when making decisions. This approach allows us to fit the cost function more generally, achieving better optimization.

6.2 Incremental Optimization

Currently, each of the three steps considers a single factor in isolation. We can try incremental optimization, such as reanalyzing banking/debanking after legalization. By making small adjustments, we can approach the optimal solution more closely.

7. Conclusion

In this project, we modify the effective mean shift algorithm [2] to achieve power and timing optimization with multi-bit flip-flop. We apply three steps flow including clustering, banking & debanking, and legalization. In clustering, we find KNN for each flip-flop and shift each of them to local maximum. In banking & debanking, we bank the flip-flops with multi-level strategy and debank the flip-flops with bad slack. In legalization, we legalize all flip-flops by BFS under utilization rate and overlapping constraints. Our method can achieve at most 91% cost reduction. However, this drastic improvement comes from the defective test case. To evaluate our method, we can achieve about 30% cost reduction under reasonable settings. To improve our method, we will consider the impact of whole cost function rather than focus on certain

one. Besides, incremental optimization between banking & debanking and legalization can be applied to achieve lower cost.

8. Contribution

Below are our work distributions for this project:

1. Meng-Chen Wu : register clustering
2. Hung-Ling Liu : IO handling, banking&debanking
3. Min-Hsin Liu : legalization

All of us worked together on the paper research, presentation, and report writing.

9. References

[1] Gang Wu, Yue Xu, Dean Wu, Manoj Ragupathy, Yu-yen Mo, and Chris Chu. Flip-flop clustering by weighted k-means algorithm. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[2] Ya-Chu Chang, Tung-Wei Lin, Iris Hui-Ru Jiang, and Gi-Joon Nam. 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, page 11–18, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Meng-Yun Liu, Yu-Cheng Lai, Wai-Kei Mak, and Ting-Chi Wang. Generation of mixed-driving multi-bit flip-flops for power optimization. In *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2022.

[4] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[5] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.