POWER AND TIMING OPTIMIZATION USING MULTIBIT FLIP-FLOP

Chen Ying Tung

National Taiwan University
Taipei, Taiwan
jennydong1005@gmail.com

Yun Hwa Tsou

National Taiwan Univeristy
Taipei, Taiwan
yunhwa.tsou@gmail.com

ABSTRACT

In this research, we aim to find a good solution to replace single-bit flip flops with multi-bit flip flops to have better performance in total area, power, and timing constraints. We first cluster the flip-flops by effective mean shift algorithm, then use dynamic programming to decide how to merge the single-bit flip-flops into multi-bit flip-flops based on area and power constraints. For simplification, we have ignored the placement rows and non-overlap constraints and chose only one kind of flip-flop for each number of bits as the flip-flop library. Our result shows that the area and the number of areas can be greatly reduced, while the power stays the same due to the limited library of flip-flops.

1. INTRODUCTION

In the modern semiconductor industry, minimizing power consumption and area in circuit design has become a highly researched topic. One commonly applied method is to merge several single-bit flip-flops into multi-bit flip-flops, which is referred to as the "multi-bit flip-flop banking" technique. In general, flip-flop banking helps save areas in the circuit. Replacing single-bit flip-flops with multi-bit flip-flops can also reduce clock power and clock network complexity.

However, with the development of the complexity in power, area, and timing optimization, flip-flop banking no longer guarantees absolute improvements. In some cases with timing critical nets, multi-bit flip-flops would possibly result in worsening timing performance, which requires the opposite technique referred to as "flip-flop debanking", to divide multi-bit flip-flops into several single-bit flip-flops.

Therefore, the target of our work aims to find a better configuration of flip-flops for circuit design with substantial registers and combinational gates. We have researched a common algorithm in this field which is called the "Mean Shift Algorithm" and its several transforms. Also, a special data structure called Rtree is used to save the absolute and relative positions of flip-flops in our work. In deciding which types of flip-flops to use, we apply dynamic programming to analyze the power efficiency of different flip-flops. The final method we proposed can be divided into two parts: register clustering, where we cluster the flip-flops according to their density distribution by effective mean shift algorithm, and flip-flop banking, in which in this stage we decide which flip-flops to merge and which to divide.

2. PROBLEM FORMULATION

The problem is based on Problem B of ICCAD 2024 CAD Contest. Given an original circuit including multiple nets, different kinds of flip-flops and several combinational gates, we are asked to find a new configuration of flip-flops that can obtain the least cost. The cost of any circuit design is measured by (1), where TNS means total negative slack and i is a flip-flop. The exact definition of the cost can be found in [1]. There are different kinds of flip-flops with different numbers of bits, different areas, and different delays given by the problem. We are allowed to change the type or the position of all flip-flops while other gates are not allowed to alter. Our goal is to find a better configuration of flip-flops, including their position so that the total cost can decrease.

While changing flip-flops, several constraints must be satisfied. First, flip-flops in different nets must not be combined since they have different clock signals. Second, all flip-flops have to be placed in the lower left corner of placement rows. Third, for every bin, the utilization rate must not exceed the limit given by the problem. The details of constraints for the problem can be found in [1].

$$Cost = \alpha \times TNS(i) + \beta \times Power(i) + \gamma \times Area(i) \quad (1)$$

3. PRELIMINARIES

3.1. Mean Shift Algorithm

Fukunaga and Hostetler have proposed classic mean shift[4], which places a Gaussian kernel on each data point and all the kernels generate a density surface. The kernel density estimator for a d-dimensional data point x is

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^{n} k(\frac{x - x_i}{h}) \quad (2)$$

A dense region will form a local hill, and the data point iteratively shifts until it reaches the nearest peak. The shifting vector of each data point can be computed by gradient ascent until it reaches a stationary point.

$$m(x) = \frac{\sum_{i=1}^{n} x_i g(\| \frac{x-x_i}{h} \|)^2}{\sum_{i=1}^{n} g(\| \frac{x-x_i}{h} \|)^2} - x \quad (3)$$

After the shifting, we can assign each data point to a cluster by a stable matching problem. However, the complexity of the classic mean shift is $O(Tn^2)$, where T is the number of iterations, and n is the number of data points. In contrast to the classic mean shift, in which all data points have the same kernel bandwidth h, an effective mean shift can vary the bandwidth by selecting K nearest neighbor. This can be a better choice of bandwidth since a small bandwidth will lead to the result that every data point becomes a cluster, and with a large bandwidth, there will be only one cluster.

In addition, we can reduce clock power while minimizing timing degradation by effective mean shift. First, the flip-flop distribution is mapped to a density surface, where dense regions form hills. During the shifting process, each flip-flop will climb up to the nearest peak in the specified bandwidth. The bandwidth varies from the distribution of each hill. Ya-Chu Chang[2] proposed to consider the k-nearest neighbors as the reference of bandwidth. The density function and shift vectors become:

$$f(x) = \frac{1}{nh_i^d} \sum_{i=1}^{n} k(\frac{x - x_i}{h_i}) \tag{4}$$

$$m(x) = \frac{\sum_{i=1}^{n} \frac{x_i}{h_i^{d+2}} g(\| \frac{x-x_i}{h_i} \|)^2}{\sum_{i=1}^{n} \frac{1}{h_i^{d+2}} g(\| \frac{x-x_i}{h_i} \|)^2} - x \tag{5}$$

3.2. Flip-flop Banking Algorithm

In the previous section, we relocated the flip-flop to form clusters. The next step is to "merge" the single-bit flip-flop into the multi-bit flip-flop. Here we did not use stable matching[2] to save the composition of each cluster. Instead, we use a fixed bounding box to define the flip-flops that need to be merged. Then based on the given multi-bit flip-flop library, a dynamic programming table can be constructed to save the best selection of multi-bit flip-flop for a certain number of bits. The cost function and other details will be described in section 4.2.

4. OUR PROPOSED METHOD

The format of the input files are text files that include all the information of the circuit. After reading the data, the flip-flops of the circuit will go through a mean shift algorithm, and flip-flop banking is expected to improve total power and area.

4.1. Effective Mean Shift Algorithm

The goal of an effective mean shift algorithm is to cluster the flip-flops based on their density distribution. Each flip-flop will shift to the nearest local hill to reduce the wire length. We adapted the source code of [2] to accomplish the mean shift algorithm. Firstly, all the flip-flops will be stored in an R-tree, which is a data structure for spatial access methods. With the R-tree, we can store the K-nearest neighbors for each flip-flop. The distance to the Kth element of the neighbor(sorted by squared Euclidean distance) will be set as the bandwidth during the computation of the mean-shift algorithm.

4.1.1. Build R-tree

We use the C++ library: boost for the R-tree structure. For the convenience of the R-tree operations, each flip-flop is represented by its left-bottom coordinate and an ID number. Then each of the flip-flops can be directly inserted into the R-tree with the build-in function. R-tree can also be used to detect intersection, disjoint, or overlap.

4.1.2. Find the K-nearest neighbors for each flip-flop

Since the R-tree has been built, we can directly use the built-in function to query the K-nearest neighbors of each flip-flop and save them by vectors. Before adding the neighbors to the vector, the squared Euclidean distance of the flip-flop and its neighbors have to be smaller than the maximum displacement, which is a parameter that can prevent large bandwidth during shifting. As a result, some of the flip-flops may have less than K-neighbors.

4.1.3. Computation of shifting vector

For each flip-flop, the shifting vector is computed by the Gaussian with all of its neighbors. Each flip-flop will shift for several iterations until it does not shift anymore.

4.2. Flip-Flop Banking

After register clustering, we merge and divide flip-flops in each cluster into either single-bit or multi-bit flip-flops. Here again, we construct a new R-tree to find the nearest cluster. Before we perform flip-flop banking/debanking, we apply cost function analysis to flip-flops to find the one with the least cost in each number of bits. Then, to find the best combination of flip-flops in each cluster, dynamic programming is performed based on power and area analysis. After deciding the constitution of each cluster, we relocate flip-flops and map their corresponding pins correctly. The details of our method are explained in the following paragraphs.

4.2.1. Neighbors Searching Using R-Tree

After flip-flops have been clustered together as a group, we reconstruct the R-tree to update the relative locations of flip-flops. To identify clusters in the circuit, we use a query box from the R-tree data structure that can find all flip-flops (or say, boxes) within the box given the boundaries of it. We execute the query box on every flip-flop in the circuit and delete those searched in the list of flip-flops. Every time after each search, we perform either banking or debanking on the neighbors we found, then we go on to search for the next cluster until there is no flip-flop in the list.

4.2.2. Cost Function Analysis

Before the banking/debanking stage, it is crucial to decide what kinds of flip-flops to use in order to lower the cost. For there are substantial types of flip-flops with the same number of bits in the problem, we consider that it is more efficient if we use only one type of flip-flop with the least cost in each number of bits. To find this flip-flop, we use the cost function defined in the problem to calculate the cost of each flip-flop according to its power and area. The formula is illustrated in (6), where i is a flip-flop.

$$\beta \times Power(i) + \gamma \times Area(i) \tag{6}$$

Once we find the flip-flop with the least cost in each number of bits, we only use that flip-flop in the following steps of flip-flop banking/debanking.

4.2.3. Flip-flop Banking/Debanking Using Multi-bit Flip-flop Power and Area Analysis

To find the best configuration of flip-flops in each cluster, we focus on the total power and area consumption of each different combination of flip-flops. We aim to find the best combination of flip-flops for each cluster such that the cost of power and area is the least. The problem can reduced to the 0-1 Knapsack problem [3], which is solvable by using dynamic programming. In the table of

Set of Flip-flop \Bits	0	1	2	3	4
$\emptyset$	0	INF	INF	INF	INF
{1}	0	100	200	300	400
{1, 2}	0	100	172	272	344
{1, 2, 4}	0	100	172	272	312

Table 1: Example of Multibit flip-flop cost analysis

dynamic programming, instead of calculating power solely as performed in [3], we calculate the total cost of both power and area by the formula (6) under different configurations of flip-flops, so we can get more accurate analysis matching with our problem. The table size is $(N + 1) \times (b_{lcm} + 1)$, where N is the total number of types of number of bits and b_{lcm} is the least common multiple of the numbers of bits. We also use another table of the same size to save the combination of different flip-flops in each entry, so that we can know which flip-flops to use in the following sections.

An example of the table is shown in Table. 1. In this example, there are in total three kinds of flip-flops after performing cost function analysis, including one single-bit, one 2-bit, and one 4-bit flip-flop. After simple calculation, we can know the least common multiple of them is 4, so the size of the table is 4×5. The leftmost column represents the set of flip-flops that can be used for each row, while the top row is the number of bits. After applying dynamic programming, we can get every combination of flip-flops for bits ranging from one to four having the minimum of cost.

We recursively divide the total number of bits in a cluster (which is found by the query box in 4.2.1) by the number of bits in the table in decreasing order. At the same while, we accumulate the number of used flip-flops for each division to get the final number of each kind of flip-flops in a cluster. For instance, if there are in total of 7 bits in the cluster, since $7 \div 4 = 1 \cdots 3$, we know the least cost of the cluster would be $312 \times 1 + 272 \times 1 = 584$. The corresponding combination of flip-flops can be found in the other table.

4.2.4. Relocation and Pin Mapping

The relocation of multi-bit flip-flops after banking is decided by its original location in the cluster after the Effective Mean Shift algorithm. We put the flip-flop back to the original coordinates according to their original location in the cluster so that it should not overlap with others in general cases.

The last step is to map the original pins to newly created flip-flops following the criteria of the problem. This action is executed every time a new flip-flop, either single-bit or multi-bit, is created. We count the accumulated number of bits among the flip-flops in the cluster and map the original pins by their original sequence in the cluster. The naming rule follows the flip-flop list given by the problem.

5. RESULT

We implemented the above method in the C++ programming language and executed the program on a platform under Ubuntu 22.04.2 LTS. Our proposed method applies several parameters in the algorithm that can be modulated to obtain better results. These parameters include the number of nearest neighbors in the Mean Shift algorithm and the search range of the query box in the flip-flop banking stage.

We experiment with our algorithm with different values of parameters to get the best result. The experiment result is presented in Table 2. For Table 2, we can see that the total cost grows under the same number of nearest neighbors during mean shifting as the search range increases. However, the increase is mainly due to the rise of area while the power stays almost the same. Similarly, the cost with different numbers of nearest neighbors, which is shown in Table 3, reveals the same result.

We suppose one of the possible explanations is the magnitude of gate power given in the test case is too minor compared to that of area, resulting in unnoticeable changes. Since we only select one kind of flip-flop for each number of bits, the difference in power between different kinds of flip-flops is only about 10%, which makes the difference even smaller.

6. DISCUSSION

Overall, our proposed method can efficiently reduce the total area and power of the flip-flops while the number of flip-flops can be greatly reduced. However, we omit some of the constraints to simplify the problem first to obtain the prototype of our solution.

First, we do not consider the Placement Rows constraint. That is, flip-flops may only be placed in the lower left corner being on the site grid. We consider that the problem is extremely complex for every placement row could have a different width and height, resulting in irregular qualified points. Therefore, in the current version of our algorithm, flip-flops are placed directly in the location calculated by the Mean Shift algorithm. Possible improvements can be made by setting more constraints during the calculation of positions during the Mean Shift algorithm and using appropriate data structures for grinding placement.

Second, in the Mean Shift algorithm, we consider flip-flops as dots using the coordinates of their lower left corner. We place flip-flops without considering the widths and heights of flip-flops, which means there is the possibility of overlapping. To further improve this problem, boundary constraints are needed in the Mean Shift algorithm to make sure that no flip-flop is placed in the region of other flip-flops.

Last, in our proposed method, we choose only one flip-flop among flip-flops with the same number of bits. The main reason is the flip-flop with the least cost should then be the best choice. However, in some special cases, we might want to expand our library of flip-flops. For instance, some flip-flops with a specific size could be more suitable although it has more cost than others under the case that choosing other flip-flops with smaller cost could induce problems such as overlapping, resulting in even greater cost to fix the problem. In this kind of circumstance, it would be more flexibility if we had more choices of flip-flops with the same number of bits. In addition, the timing analysis can be further implemented to make sure that the resulting circuit satisfies the timing constraint.

7. CONCLUSION

Our proposed method is applicable without the placement of rows and overlapping constraints. For further improvement, the connection between every pin should be wisely saved to easily implement timing analysis and calculate total negative slack. Also, the placement of the flip-flop should be stored on a map that can prevent overlapping. In this research, we only consider the banking algo-

Search Range	Area	Power	Cost	1-bit	2-bit	4-bit	Total # of flip-flops
Before banking	4.63E+11	457.185	2.31E+12	19879	0	0	19879
500	2.69E+11	2.87638	1.34E+12	3775	2783	1114	7672
1000	2.80E+11	2.87638	1.40E+12	3125	2662	1520	7307
2000	2.97856e+11	2.87638	1.48928e+12	2283	2197	2243	6723
3000	3.10021e+11	2.87638	1.55E+12	1812	1811	2746	6369
4000	3.19E+11	2.87638	1.59475e+12	1496	1467	3139	6102

Table 2: Cost with different search range

K	Area	Power	Cost	1-bit	2-bit	4-bit	Total # of flip-flops
Before banking	4.63E+11	457.185	2.31E+12	19879	0	0	19879
7	2.84E+11	2.87638	1.42E+12	2804	2686	1651	7141
14	2.97856e+11	2.87638	1.48928e+12	2283	2197	2243	6723
30	3.22E+11	2.87638	1.61E+12	1465	1338	3261	6064
50	3.39E+11	2.87638	1.70E+12	830	815	3951	5596
70	3.46E+11	2.87638	1.73E+12	596	577	4236	5409

Table 3: Cost with different number of nearest neighbor

rithm. Debanking algorithm is also important for the flexibility of this clustering problem.

8. JOB DIVISION

Task \Member	Chen Ying	Yun Hwa
Paper survey	V	V
Mean Shift Alg. implementation	V	
Flip-flop Banking implementation		V
Report	V	V

9. REFERENCES

[1] Sheng-Wei Yang, Tzu-Hsuan Chen, Jhih-Wei Hsu, Ting Wei Li, Cindy Chin-Fang Shen Synopsys, Inc, *Power and Timing Optimization Using Multibit Flip-Flop*.

[2] Ya-Chu Chang, Tung-Wei Lin, Iris Hui-Ru Jiang, Gi-Joon Nam, *Graceful Register Clustering by Effective Mean Shift Algorithm for Power and Timing Balancing*, San Francisco, USA, April, 2019.

[3] Iris Hui-Ru Jiang, Chih-Long Chang, and Yu-Ming Yang, *INTEGRA: Fast Multibit Flip-Flop Clustering for Clock Power Saving*, February 2012.

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