Flip-flop Clustering by Weighted K-means Algorithm

Gang Wu*, Yue Xu†, Dean Wu‡, Manoj Ragupathy†, Yu-yen Mo† and Chris Chu*

*Department of Electrical and Computer Engineering, Iowa State University, IA, United States
†Oracle America, Santa Clara, CA, United States
‡RedMart, Singapore
Email: {gangwu, cnchu}@iastate.edu, {yue.x.xu, manoj.ragupathy, yuyen.mo}@oracle.com, dean.wu@gmail.com

ABSTRACT

This paper presents a novel flip-flop clustering and relocation framework to help reduce the overall chip power consumption. Given an initial legalized placement, our goal is to reduce the wirelength of the clock network by reducing distance between flip-flops and their drivers, while minimize the disturbance of original placement result. The idea is to form flip-flops into clusters, such that all flip-flops within each cluster can be placed near a single clock buffer and connected by a simple routing structure. Therefore, overall clock network wirelength can be greatly reduced and significant power savings can be achieved. In particular, we propose a modified K-means algorithm which effectively assigns flops into clusters at the clustering step. Then, at the relocation step, flops are actually relocated and regularly structured clusters are formed. Our framework is evaluated on real industrial benchmarks. We compare our framework with a flow without flop clustering and an industrial window based flop clustering flow. Experimental results show our framework can achieve significant dynamic power savings while has less disturbance of the original placement.

1. INTRODUCTION

Due to the more restrictive temperature constraints and increasing requirements of the battery life, power has become a very important optimization objective for modern VLSI designs. An effective way to reduce power consumption is to put more emphasis on the design and optimization of clock networks, since among the overall chip power consumption, more than 40% power can be consumed by the switching power of the clock network [1]. One reason that clock consumes so much power is because the clock signals switch much more frequently than regular signals. Another reason is that the clock network often drives a large number of flip-flops which create huge load capacitance.

Power optimization for clock network has been studied for decades and many techniques, such as clock gating [2], clock buffer sizing [3], dynamic voltage/frequency scaling [4], etc., have been developed. Recently, researchers try to optimize clock network by exploring better placement locations for flip-flops. One family of techniques perform flip-flop placement during the traditional global placement stage, through net weighting [5] or using the guidance of Manhattan rings [6]. However, these methods might increase routing congestion and also lead to significant signal wirelength increase, especially for large scale designs [7]. Another family of techniques try to adjust flip-flop locations after the placement stage [8–15]. The basic idea is to bring flip-flops closer to each other and form them into clusters. As an example, Fig. 1 shows part of the design after performing the post-placement flip-flop clustering using the framework proposed in this paper.

There are many benefits of performing flip-flop clustering after the conventional placement stage. First, since the number of flops per cluster can be controlled to optimize the use of a single clock buffer, the total number of clock buffers used in the design can be much less, and the reduction of the number of clock buffer at the first level can reduce the rest of clock tree. Second, after forming a regular placement structure for all the flops within one cluster, a simple routing structure, such as fishbone routing, will be able to route the leaf level of the clock tree. Thus, the overall clock wirelength can be effectively reduced [10]. In addition, since all the flops are placed very close to the clock buffer, the clock skew is reduced, which can help improve the timing of the circuit [15].

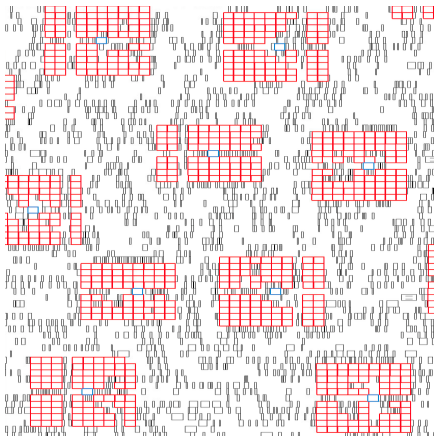


Figure 1: Part of the design after performing flip-flop clustering and relocation. Flip-flops are highlighted as red and clock buffers are highlighted as blue.

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2898025

The reduction of clock network wirelength comes at a cost of the increase of signal wirelength. However, since a significant portion of the power is consumed by clock wires [1], the clock power reduction can be larger than the overhead in signal power. Another concern is that flop clustering might hurt the timing of the circuit, as the clustering process might cause some flops to move a very long distance, and the combinational cells can also be moved because of the legalization. However, the timing degradation can be effectively controlled by minimizing the disturbance of the original placement and limiting the maximum displacement of flip-flops during the clustering process. In addition, considering the timing information at flop clustering stage is rough, there are still chances to improve the timing in later stages such as the routing stage. Therefore, flip-flop clustering is able to produce significant power savings with tolerable delay impact.

Many works have been done on the post-placement flip-flop clustering problem. In [8] [9], the groups of flip-flops or latches to be formed into clusters are either determined by some simple heuristic criterion or by greedily splitting big clusters. Thus, the clustering results obtained by these approaches can be far from optimal. In [10], a genetic algorithm based latch clustering approach is proposed. However, genetic algorithms usually have long runtime and are not scalable, which makes it not practical for large scale circuits. In [11–14], the authors explored the intersection graph based clustering approach which helps replace a group of flops into a multi-bit flip-flop (MBFF). The idea is to form an intersection graph based on the intersection of the feasible movement regions of flip-flops. Then, the clustering problem is transformed into the problem of finding all maximal cliques in the intersection graph. However, this approach is suitable only when feasible movement region of each flip-flop is very small, in which case each formed MBFF only contains very few number of flip-flops. In our case, the feasible movement region is much larger and each formed cluster contains many flip-flops. Therefore, the obtained intersection graph will be very dense and the runtime of these algorithms will not be acceptable. In [15], a clustering approach adapting K-means algorithm is proposed, which is similar to our framework. However, the proposed approach does not have control on the number of flip-flops within each cluster, which can create very unbalanced clustering results and violating the maximum drive strength of the clock buffer. Also, they do not have constraint on the maximum displacement of flip-flops, and might cause timing degradation when flops move a very long distance.

In this paper, we are focusing on the problem of reducing power consumption by performing post-placement flip-flop clustering and relocation. The input to our framework is a a design which has already been placed and legalized. We want to group and relocate the flip-flops to form them into regularly structured clusters. Our goal is to minimize the total displacement of all the flip-flops which in turn reduces the disruption of the original placement results, and minimize the number of clock buffers used, therefore reduce the rest of clock tree. In addition, we enforce a hard constraint on the maximum allowable displacement for each flip-flop to avoid timing degradation caused by critical flops moved very far away from its original position. We also enforce an upper bound on the maximum number of flops allowed within each cluster to help meet the maximum the drive strength of the clock buffers. Other design constraints, such as clock domains, enable signals and placement blockages are also considered in our framework.

Our framework decomposes the flip-flop clustering problem into two steps: flip-flop clustering and flip-flop relocation. The first step finds the groups of flops to be clustered by a modified K-means algorithm. Since the standard K-means algorithm does not enforce any constraints, we developed methods which can be combined with the K-means algorithm to guarantee the clustering results satisfy the maximum displacement constraint for each flip-flop and the cluster size constraint for each cluster. In particular, since the sizes of the clusters generated by the standard K-means algorithm are very unbalanced, we add weights on each cluster at the cluster assignment step of K-means to help balance the number of flops within each cluster. In the flip-flop relocation step, we actually moves the flops into legal locations with respect to the placement blockages and form them into regularly structured clusters.

The effectiveness of our framework is evaluated on real industrial designs which contain 400K cells on average. Our framework is compared with a physical design flow without performing any flip-flop clustering and an existing window based clustering flow which has already been used in the production. In terms of the total switching power, our framework has achieved 9.4% savings compared with the flow without flip-flop clustering and 4.8% savings compared with the window based flip-flop clustering flow.

The rest of this paper is organized as follows. In Section II, we describe preliminaries about the K-means algorithm and formally define the problem solved in this paper. In Section III, we present our flip-flop clustering framework. Finally, the experimental results are presented in Section IV.

2. PRELIMINARIES

2.1 K-means algorithm

K-means algorithm [16] is one of the most widely used algorithms for clustering, due to its simplicity, efficiency and empirical success [17]. The standard K-means algorithm finds a partition such that the sum of Euclidean distance between the cluster center and the instances is minimized. Here, the cluster center is calculated as the mean location of all the instances within the cluster.

Let N be the total number of instances to be clustered. We denote the x-coordinates of instances by a vector $\mathbf{x} = (x_1, x_2, \cdots, x_N)$. We denote their y-coordinates by a vector $\mathbf{y} = (y_1, y_2, \cdots, y_N)$. Let $\mathbf{C} = (C_1, C_2, \cdots, C_K)$ be a set of K clusters of instances. Let $\mu_x(C_k)$ and $\mu_y(C_k)$ be the x and y coordinate of the center of cluster C_k. The problem solved by K-means algorithm can be formally written as:

$$\text{Min} \quad \sum_{k=1}^{K} \sum_{(x_i, y_i) \in C_k} (||x_i - \mu_x(C_k)||^2 + ||y_i - \mu_y(C_k)||^2)$$

The steps of the standard K-means algorithm which solves the above problem are as follows:

- Step 1: Choose K initial cluster center locations.
- Step 2: Assign each instance to the cluster which provides the smallest cost.
- Step 3: Recompute the center location of each cluster.

- Step 4: Repeat steps 2 and 3 until there is no further change in costs of all instances.

Here, the cost of assigning an instance locating at (x_i, y_i) to cluster C_k is defined as:

$$Cost = ||x_i - \mu_x(C_k)||^2 + ||y_i - \mu_y(C_k)||^2$$

The runtime of the standard K-means algorithm is $O(t * N*K)$, where t is the number of iterations until convergence. In practice, t is often small and the results only improve slightly after few iterations, which makes K-means algorithm to be very fast compared with other clustering methods, especially for very large scale data sets [18].

2.2 Problem formulation

In our problem, the instances to be clustered are flip-flops. The flop displacement cause by the clustering process can be approximated as the Manhattan distance between the flip-flop and the cluster center. Then, the flip-flop clustering problem which minimize the total sum of flop displacement and K, while satisfies the cluster size constraints and flop displacement constraints can be formulated as:

$$\text{Min} \sum_{k=1}^{K} \sum_{(x_i,y_i)\in C_k} (|x_i - \mu_x(C_k)| + |y_i - \mu_y(C_k)|) + \alpha * K$$

$$\text{Subject to } |C_k| \leq size\_limit \qquad \forall k$$

$$|x_i - \mu_x(C_k)| + |y_i - \mu_y(C_k)| \leq disp\_limit_i \qquad \forall k \text{ and } \forall (x_i, y_i) \in C_k$$

Here, α is a constant value adjusting the effort between minimizing displacement and K. $size\_limit$ is a given constant value denote the cluster size limit. $disp\_limit_i$ is the maximum allowable displacement for flop i according to its timing criticality.

It can be seen that the standard K-means algorithm cannot be directly applied to our problem due to the differences in objective function and the extra constraints. We will discuss how we handle these differences by our weighted K-means algorithm in Sec. III-A.

3. OUR PROPOSED FRAMEWORK

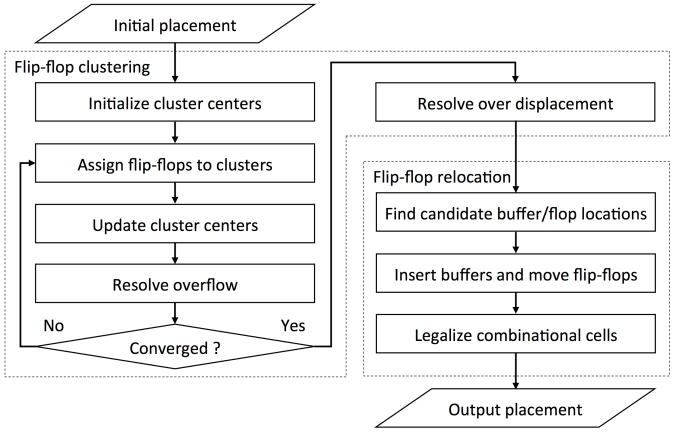


Figure 2: The proposed flip-flop clustering and relocation framework.

An overview of our two-step flip-flop clustering framework is shown in Fig. 2. Our framework starts with a timing optimized, legalized placement. At the flip-flop clustering step, we first initialize K cluster center locations. Then, a clustering solution satisfying the cluster size constraints and flop displacement constraints are generated by our weighted K-means algorithm. At the flip-flop relocation step, we first find legal locations for clock buffers and flops. Then, buffers are inserted per cluster and flops are relocated. In the end, we legalize the combinational cells with flop locations fixed.

3.1 Flip-flop Clustering

3.1.1 Initialize cluster centers

Finding a proper K value can be difficult, since increasing K will result in a smaller total flop displacement, but also increase the number of clock buffers used in the design. A trivial solution would be driving each flop by one clock buffer. Here, we use a large α value in the objective function to minimize K. After we decide K, we also need to find K initial cluster center locations, which can affect the clustering results and the number of iterations required to converge. One commonly used idea is to randomly pick K instance locations from the data set and use them as the initial center locations. However, we do not want to introduce randomness into our framework, which might cause troubles for the physical design convergence. Here, we propose the following recursive bipartition approach to help us find an initial K value and deploy K center locations on the placement region, as shown in Algorithm 1:

Algorithm 1 Initialize K Cluster Centers

```
1: function InitCenter(S, K);
2:     if |S| ≤ size_limit then
3:         Initiate a center at ( Σ_{x_i∈S} x_i/|S|, Σ_{y_i∈S} y_i/|S|);
4:         return
5:     end if
6:     Bipartite S into S_1, S_2
7:       with |S_1| = |S| * ⌊K/2⌋ /|K|, |S_2| = |S| * ⌈K/2⌉ /|K|;
8:     InitCenter(S_1, ⌊K/2⌋);
9:     InitCenter(S_2, ⌈K/2⌉);
10: end function
```

We use S to denote the set of flip-flops to be partitioned. Since α is large, it is the best to generate a solution with K as small as possible. Initially, we roughly set $K = |S|/size\_limit$. The function returns when the number of flip-flops to be partitioned is no more than $size\_limit$. Otherwise, we split the flip-flops into two partitions with one partition has $|S| * \lfloor K/2 \rfloor /|K|$ flops and the other has $|S| * \lceil K/2 \rceil /|K|$ flops. This makes the number of flip-flops assigned at each partition be proportional to the number of clusters at each partition. In particular, we sort the flops based on their x or y coordinates depending on whether we perform vertical or horizontal partition at this iteration. Then, we assign flip-flops to S_1 based on their sorted order until we reach the desired number of flops for this partition. The rest of flops will be assigned to S_2.

3.1.2 Assign flip-flops to clusters

The standard K-means algorithm assigns a flip-flop to the cluster whose center yields the smallest Euclidean distance. Considering wires can only be horizontal or vertical during

the routing, here we use Manhattan distance instead of Euclidean distance. Thus, the cluster can be picked based on the following cost function:

$$Cost = |x_i - \mu_x(C_k)| + |y_i - \mu_y(C_k)| \quad (1)$$

However, if we generate the clustering results using the above cost function, the sizes of the clusters can be very unbalanced, which makes it very difficult to satisfy the cluster size constraints required by our problem formulation. An example is shown in Fig. 3, where X axis lists the index of each cluster and is sorted based on the cluster size. Y axis shows the number of flops within each cluster. Considering the maximum allowable cluster size to be 80, it can be seen that there are many clusters which are over the size limit.

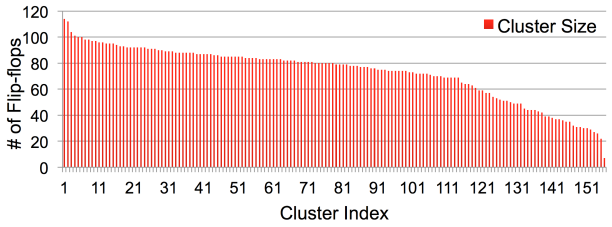


Figure 3: Sizes of clusters by standard K-means algorithm.

In order to have a more balanced clustering results, we add a weight to each cluster based on its current size. The basic idea is to set a higher weight to a cluster if it contains more flip-flops. Thus, flip-flops will have a lower tendency to be assigned to this cluster, since the cost of choosing the cluster is set to be the original cost multiply the current weight of this cluster. However, when we choose a proper weight setting method, we also need to consider the trade-off between cell displacement and the balancing of cluster sizes. In particular, a higher weight or history based weight provides us less overflow but larger total flip-flop displacement. Here, we use a smaller and non-history based weight as shown below, which provides a better total displacement. The overflowed clusters can be effectively handled at our resolve overflow step.

$$Cost = (|x_i - \mu_x(C_k)| + |y_i - \mu_y(C_k)|) * max(\ (|C_k|/size\_limit),\ 1\ ) \quad (2)$$

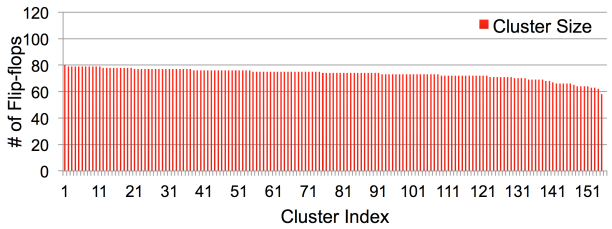


Figure 4: Sizes of clusters by weighted K-means algorithm.

Fig. 4 shows the cluster sizes after applying the above cost function. It can be seen that all the cluster sizes are around the *size_limit*. The effectiveness of the weighted K-means algorithm can also be seen in Fig. 5, where X axis shows the number K-means iteration and Y axis shows the percentage of overflowed clusters. After we use the weighted cost function, the percentage of overflowed clusters becomes less and less when more iterations of K-means algorithm are performed.

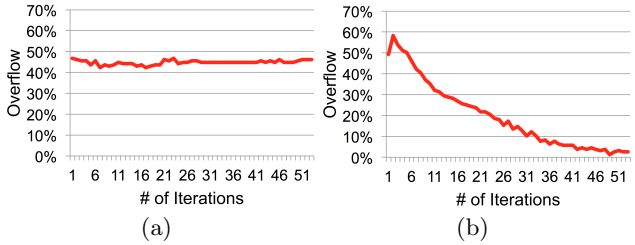


Figure 5: Percentage of overflow clusters in (a) standard K-means algorithm (b) weighted K-means algorithm.

In the first iteration of the K-means algorithm, we still use Equation (1) to calculate the cost at flip-flop assignment step, since all clusters are empty in the beginning. In the rest of the K-means iterations, we update the cluster assignment of each flip-flop at the flip-flop assignment step, based on the cost calculated by Equation (2).

One thing we noticed is that it is very important to update the weight of the cluster immediately, which means whenever we move a flip-flop from one cluster to the other, we need to update the weight of the corresponding two clusters. Otherwise, oscillation problems can happen: in one iteration, many flip-flops are moved into one cluster, but in next iteration, all these flip-flops move away due to the huge weight of this cluster caused at the previous iteration. This can make the K-means algorithm become very difficult to converge.

3.1.3 *Update cluster centers*

Same as the standard K-means algorithm, at this step, centers of each cluster are recalculated as the mean value of the flip-flop locations:

$$\mu_x(C_k) = \sum_{x_i \in C_k} x_i/|C_k|, \quad \mu_y(C_k) = \sum_{y_i \in C_k} y_i/|C_k| \quad \forall k$$

3.1.4 *Resolve overflow*

For some designs such as the one in Fig. 4, simply adding weights in the cost function will make all clusters satisfy the size constraints. However, this cannot be guaranteed for all the designs. Thus, we add the resolve overflow step within the K-means iteration which guarantees all cluster sizes are under the *size_limit* when our weighted K-means algorithm terminates.

Our method to resolve overflow is like this: at every certain K-means iterations, we pick one cluster which has most number of flip-flops among all the clusters violating the size constraints. Then, a new center is inserted near the center of this cluster and a new empty cluster is created accordingly. Next, if a smaller cost can be achieved, the flip-flop in the overflowed cluster will be moved to this new cluster. The weights of these two clusters are also updated accordingly.

The K-means iteration continues until all the clusters satisfy the size constraints and there is no improvement on costs of all the flip-flops within certain iterations.

3.1.5 *Resolve over displacement*

If the number of clusters (K) is sufficient and the $disp\_limit_i$ is not too small, most of the flip-flops will satisfy the dis-

placement constraint for the clustering solution generated by our weighted K-means algorithm. However, there are some corner cases, which one flop can be extremely far away from other flops in the original legalized placement. Thus, it is necessary to develop a post-processing step to fix the over displacement problems for these particular flip-flops.

The method we used to fix over displacement is to insert a new cluster centered at the location of the violating flip-flop. Then, we assign the violating flip-flop to this new cluster. To take the most advantage of this new cluster, we will also assign nearby flip-flops to this new cluster, if smaller costs can be achieved. Different from resolving overflow, we cannot resolve the over displacement within the K-means iteration, since the resolve displacement step inserts a small weight cluster which can be pulled away from the violating flip-flop by other flops during the K-means iteration.

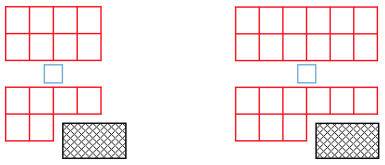
An example of the flip-flop clustering results are shown in Fig. 8 (a), where each flip-flop is assigned to one cluster which is denoted by the fly lines (blue) connecting the flip-flops to the center of the cluster.

3.2 Flip-flop Relocation

3.2.1 *Find candidate buffer and flip-flop locations*

The desired clock buffer location is the mean center location generated by our algorithm. However, it is possible that this location is overlapping with some placement blockages. In this case, we simply search around and find the nearest legal location as the candidate buffer location.

We form the flops within one cluster into a wing structure which has an empty column over the clock buffer, just as the cluster structure used in the window based industrial flow. To find candidate flop locations, a default configured wing structure is formed first, according to the location of the clock buffer. Then, candidate locations which are overlapping with the blockages will be removed, as shown in Fig. 6 (a). If the remaining candidate locations are not enough to allocate all the flops within this cluster, we use a new configuration to enlarge the wing structure until sufficient candidate flop locations are found, as shown in Fig. 6 (b).



(a) (b)

Figure 6: (a) A 4×4 configuration for the wing structure with blockage overlapping locations removed. (b) An enlarged 4×6 configuration with sufficient candidate locations.

3.2.2 *Insert buffers and relocate flip-flops*

First, buffers are inserted at the candidate buffer location. Then, flops are sequentially moved to the candidate flop locations as shown in Fig. 7. In particular, for each flop, we try all candidate locations within the wing structure and pick the one which provides the smallest displacement. After we relocates the flop to the candidate location, this location will no long be available for other flops. The order we used to relocate the flop is based on their timing criticality and the flop which is more timing critical will be moved first.

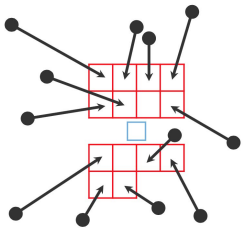
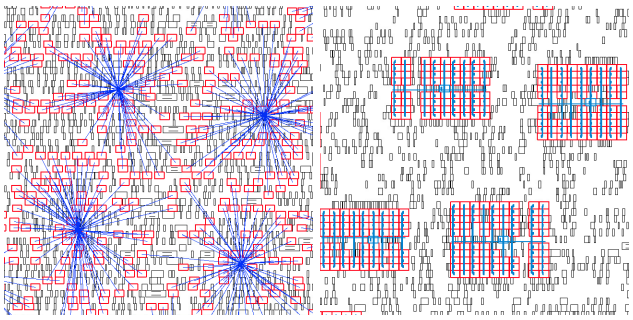


Figure 7: Move flip-flops into candidate locations.

In the end, we also adjust the orientation of the flip-flops to make sure their clock pins are properly aligned to help reduce the clock wirelength. Part of the design with routed clock nets after flop relocation is shown in Fig. 8 (b).



(a) (b)

Figure 8: Part of the design: (a) after performing flip-flop clustering (b) after clock routing.

4. EXPERIMENTS

Our flip-flop clustering and relocation framework are evaluated on 8 real industrial designs ranging from 55K to 795K cells. These designs are placed using the state-of-art commercial physical design tool as an input to both the window based flip-flop clustering flow and our framework. In particular, the window based flip-flop clustering flow look for flops to group window by window. All the flops within a window are greedily moved together to form a cluster. This flow has already been used in real production and is able to obtain sufficient power savings with minor timing degradation.

We set the $size\_limit$ to be 80 and the $disp\_limit_i$ to be 60 μm for all the flops, which is same as the value used in the window based industrial flow. The flop clustering is performed at each group of flops having the same clock domain and sharing a common enable signal. In addition, the resolve overflow step is performed at every 5 K-means iteration and the loop terminates when there is no improvement within 10 iterations. After the flip-flop relocation, a commercial physical design tool is used to legalize the combinational cells if they are overlapping with the relocated flops. Finally, rest of the clock tree is constructed by commercial CTS tool and the design is routed to get the wire load.

Since the static power consumption will not be affected by the flip-flop locations, we focus on comparing the switching power among all the flows. The switching power for both clock and signal nets are estimated using the traditional $\beta * C_{load} * Vdd^2 * f_{clock}$ which is a good approximation for interconnect power. Here, β denotes the switching activity factor.

Table I. Comparison on industrial benchmarks

	# of Cells	# of Flops	Disp. x 10^3 (μm)		Total WL x 10^6 (μm)			Clk Switching Power (mW)			Total Switching Power (mW)		
			WB	Ours	NC	WB	Ours	NC	WB	Ours	NC	WB	Ours
D1	55K	9K	67.58	74.44	1.60	1.68	1.70	8.04	6.23	4.50	23.28	22.29	20.81
D2	172K	36K	237.68	190.74	4.94	5.26	5.20	24.11	18.66	17.04	71.05	69.05	66.86
D3	229K	39K	365.85	311.79	12.44	12.86	12.71	34.12	21.30	19.29	153.79	145.66	142.26
D4	322K	58K	371.03	310.82	7.44	8.48	7.90	40.99	28.08	28.17	111.47	109.39	103.83
D5	399K	73K	1018.83	441.55	10.76	13.03	11.32	53.18	34.11	33.43	155.44	159.47	142.17
D6	668K	123K	934.80	859.02	20.04	21.26	20.89	102.75	67.06	59.88	293.07	271.06	260.65
D7	537K	127K	716.12	637.18	16.09	17.05	16.90	88.19	69.04	60.61	240.72	231.87	222.40
D8	795K	166K	1171.14	979.31	21.22	22.87	22.67	124.72	88.32	79.99	325.49	306.96	297.09
		Norm.	1.283	1.000	0.952	1.032	1.000	1.572	1.099	1.000	1.094	1.048	1.000

The experimental results are shown in Table I. "NC" denotes the non-clustering flow. "WB" denotes the window based flip-flop flow. "Disp." column shows the total flip-flop displacement. "Total WL" column shows the total wirelength which includes clock nets and regular signal nets. Compared with the flop displacement, our framework is 28.3% better than the window based flow. This indicates our framework has much less disturbance on the original placement results and should be much easier to achieve timing closure compared with the window based flow. For the clock switching power, our framework is 57.2% better than the flow without any flip-flop clustering and 9.9% better than the window based flow. For the total switching power, our framework is 9.4% better than the non-clustering flow and 4.8% better than the window based flow. These show that our framework is very effective on reducing dynamic power consumption. The average number of flops per cluster is around 73 for all our clustering results, which indicates the clock buffer being used is close to minimum. Since the window based flow is implemented using Tcl scripts while our framework is implemented using C++, it is not fair to compare the runtime between these two flows. In general, our framework runs much faster than the window based clustering flow and the proposed weighted K-means algorithm converges within minutes even for very large designs.

5. CONCLUSIONS

This paper has proposed a novel flip-flop clustering framework to help reduce power consumption at post-placement stage. The weights in the cost function of K-means algorithm is essential for us to generate more balanced clustering results, which makes the K-means algorithm suitable for the flip-flop clustering problem. In addition, we develop efficient steps guaranteeing the clustering results satisfying the size and displacement constraints. Our framework is evaluated on large scale industrial designs and compared with industrial flows. The significant improvement has demonstrated the practicability and the effectiveness of our framework.

6. REFERENCES

[1] D. Papa, C. Alpert, C. Sze, Z. Li, N. Viswanathan, G.-J. Nam, and I. L. Markov, "Physical synthesis with clock-network optimization for large systems on chips," *Micro, IEEE*, vol. 31, no. 4, pp. 51–62, 2011.

[2] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Trans. Circuits Syst. I, Fundam. Theory*, vol. 47, no. 3, pp. 415–420, 2000.

[3] K. Wang and M. Marek-Sadowska, "Buffer sizing for clock power minimization subject to general skew constraints," in *DAC 2004*.

[4] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *ICCAD 2002*.

[5] Y. Cheon, P.-H. Ho, A. B. Kahng, S. Reda, and Q. Wang, "Power-aware placement," in *DAC 2005*.

[6] Y. Lu, C. Sze, X. Hong, Q. Zhou, Y. Cai, L. Huang, and J. Hu, "Navigating registers in placement for clock network minimization," in *DAC 2005*.

[7] D.-J. Lee and I. L. Markov, "Obstacle-aware clock-tree shaping during placement," *TCAD*, vol. 31, no. 2, pp. 205–216, 2012.

[8] W. Hou, D. Liu, and P.-H. Ho, "Automatic register banking for low-power clock trees," in *ISQED 2009*.

[9] C. J. Alpert, Z. Li, G.-J. Nam, D. A. Papa, C. N. Sze, and N. Viswanathan, "Latch clustering with proximity to local clock buffers," 2013. US Patent 8,458,634.

[10] S. I. Ward, N. Viswanathan, N. Y. Zhou, C. C. Sze, Z. Li, C. J. Alpert, and D. Z. Pan, "Clock power minimization using structured latch templates and decision tree induction," in *ICCAD 2013*.

[11] I. H.-R. Jiang, C.-L. Chang, and Y.-M. Yang, "INTEGRA: Fast multibit flip-flop clustering for clock power saving," *TCAD*, vol. 31, pp. 192–204, 2012.

[12] S.-H. Wang, Y.-Y. Liang, T.-Y. Kuo, and W.-K. Mak, "Power-driven flip-flop merging and relocation," *TCAD*, vol. 31, pp. 180–191, 2012.

[13] 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 *ICCAD 2010*.

[14] C. Xu, P. Li, G. Luo, Y. Shi, and I. H.-R. Jiang, "Analytical clustering score with application to post-placement multi-bit flip-flop merging," in *ISPD*, pp. 93–100, ACM, 2015.

[15] R. Puri, H. Qian, C. N. Sze, and J. Warnock, "Regular local clock buffer placement and latch clustering by iterative optimization," 2012. US Patent 8,104,014.

[16] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, pp. 129–137, 1982.

[17] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[18] S. Har-Peled and B. Sadri, "How fast is the k-means method?," *Algorithmica*, vol. 41, pp. 185–202, 2005.