Agglomerative-Based Flip-Flop Merging and Relocation for Signal Wirelength and Clock Tree Optimization

SEAN SHIH-YING LIU, National Chiao Tung University
WAN-TING LO, Taiwan Semiconductor Manufacturing Company, Limited (TSMC)
CHIEH-JUI LEE and HUNG-MING CHEN, National Chiao Tung University

In this article, we propose a flip-flop merging algorithm based on agglomerative clustering. Compared to previous state-of-the-art on flip-flop merging, our proposed algorithm outperforms that of Chang et al. [2010] and Wang et al. [2011] in all aspects, including number of flip-flop reductions, increase in signal wirelength, displacement of flip-flops, and execution time. Our proposed algorithm also has minimal disruption to original placement. In comparison with Jiang et al. [2011], Wang et al. [2011], and Chang et al. [2010], our proposed algorithm has the least displacement when relocating merged flip-flops. While previous works on flip-flop merging focus on the number of flip-flop reduction, we further evaluate the power consumption of clock tree after flip-flop merging. To further minimize clock tree wirelength, we propose a framework that determines a preferable location for relocated merged flip-flops for clock tree synthesis (CTS). Experimental results show that our CTS-driven flip-flop merging can reduce clock tree wirelength by an average of 7.82% with minimum clock network power consumption compared to all of the previous works.

Categories and Subject Descriptors: B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids—*Automatic Synthisis, Optimization*

General Terms: Algorithms, Design

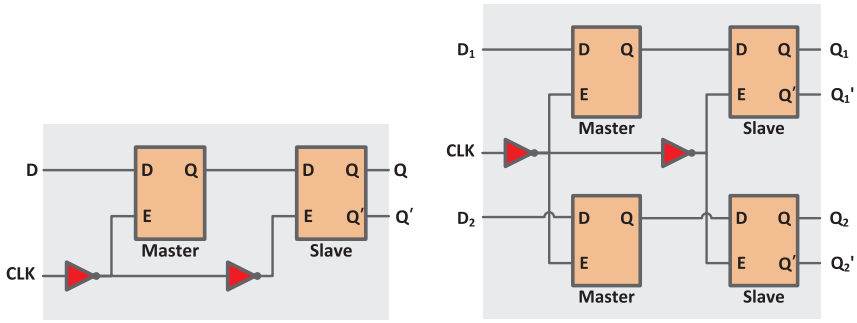
ACM Reference Format:
Liu, S. S.-Y., Lo, W.-T., Lee, C.-J., and Chen, H.-M. 2013. Agglomerative-based flip-flop merging and relocation for signal wirelength and clock tree optimization. ACM Trans. Des. Autom. Electron. Syst. 18, 3, Article 40 (July 2013), 20 pages.
DOI: http://dx.doi.org/10.1145/2491477.2491484

1. INTRODUCTION

In modern SoC design flow, recent researches in industry and academia have discovered that flip-flops in advanced technology require less driving power. Less driving power for flip-flops implies that multiple flip-flops can be driven by a single inverter. Thus, the possibility of multi-bit flip-flop emerges in nanometer design. Multibit flip-flops have the advantage of sharing single inverter, which reduces power consumption with more compact layout area. In addition, replacing original one-bit flip-flops with multibit flip-flops can reduce both sink number and sink capacitance. In this context,

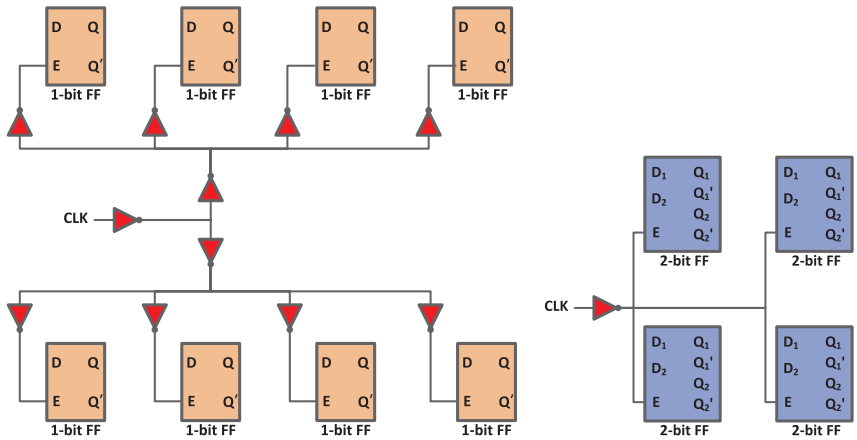
A preliminary version of this article [Liu et al. 2012] was presented in *Proceedings of the IEEE/ACM Design Automation and Test Conference in Europe.*
This work is supported in part by the National Science Council (NSC) of Taiwan under Grant No. NSC1002220E009045.
Authors' addresses: S. S.-Y. Liu and C.-J. Lee, Institute of Electronics, National Chiao Tung University, Hsinchu 300, Taiwan; emails: sniealu.ee96g@g2.nctu.edu.tw, jerrylee2929.ee92@gmail.com; W.-T. Lo, Process Design Kit Development Department, Taiwan Semiconductor Manufacturing Company (TSMC); email: soul9657@gmail.com; H.-M. Chen, Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu 300, Taiwan; email: hmchen@mail.nctu.edu.tw.

DOI: http://dx.doi.org/10.1145/2491477.2491484



(a) One-bit flip-flop. (b) Two-bit flip-flop.

Fig. 1. Illustration of a multibit flip-flop sharing one inverter.



(a) (b)

Fig. 2. Reduction of sinks in clock tree. (a) Illustration of a clock signal connected to eight one-bit flip-flops. (b) Illustration of clock wirelength reduction using multibit flip-flops to reduce the number of flip-flops.

sink capacitance refers to the clk-pin capacitance of a flip-flop. Figure 1 illustrates the implementation of a two-bit flip-flop, a single clk-pin can trigger both flip-flops using one inverter. Figure 2 illustrates how sink reduction affects the topology of a clock tree. It is experimented in Chen et al. [2010] that integrating a multibit flip-flop library into current commercial tools can significantly reduce clock tree power consumption.

According to Donno et al. [2004], the power consumption of a clock network is responsible for 40% of total power consumption. Thus, optimizing the clock network can effectively reduce total power consumption. Optimization techniques for clock tree topology include reduction on total clock tree wirelength by replacing the registers [Hou et al. 2009; Lee and Markov 2011; Wang et al. 2007], optimizing the size of buffers [Shelar 2007], or considering activity factor in placement of registers [Chen et al. 2002; Cheon et al. 2005].

Implementation of multibit flip-flops can also reduce clock tree power consumption by reducing the number of sinks. Less sinks implies shorter clock tree wirelength during clock tree synthesis. However, multibit flip-flops have the drawback of increasing in signal wirelength. For signals with a high activity factor, the improvement in power

reduction by greedily merging flip-flops may be saturated by the increase in dynamic power consumption due to an increase in signal wirelength.

1.1. Previous Works

Previous works on optimization of flip-flops can be categorized to optimizations during the placement stage and the post-placement stage.

1.1.1. Optimization during the Placement Stage. Regarding flip-flop replacement, Lee and Markov [2011] integrate a virtual clock tree with a force-directed placer SimPL [Kim et al. 2010]. In each iteration of placement, a virtual clock tree is constructed to identify the topology of the clock tree. A contraction force is then applied on each flip-flop to motivate flip-flops to group into clusters. Eventually, reshaping the clock tree can reduce the clock tree wirelength. Experimental results show that relocating the sink position during placement can reduce clock tree wirelength by 30% and dynamic power consumption by 7%. Cheon et al. [2005] proposed a methodology based on activity-based register clustering to replace registers with a higher-switching activity factor to same-leaf clusters in clock trees. Experimental results show that replacement of register by considering the activity factor of the register can reduce switching power by 25%.

1.1.2. Optimization at the Post-Placement Stage. In terms of optimization at the post-placement stage, previous state of the art on flip-flop merging [Jiang et al. 2011; Wang et al. 2011; Chang et al. 2010] shares similar objectives on reducing the number of flip-flops in order to reduce total power consumption. Reduction in the number of flip-flops is achieved by merging flip-flops to a higher-bit multi-bit flip-flop.

Chang et al. [2010] tackles the problem by applying progressive window-based optimization. As a window sweeps across a given layout, an intersection graph is constructed with a predefined window size. After the intersection graph is constructed, a maximal independent set of cliques is identified to select groups of flip-flops for merging. Similarly to Chang et al. [2010], Wang et al. [2011] applies techniques in minimum clique partitioning to identify a set of nonconflicting cliques.

In contrast to Wang et al. [2011] and Chang et al. [2010], Jiang et al. [2011] proposed a linear-size sequence representation for identifying several clustering combinations. An interval graph is constructed to record a flip-flop's feasible region on both the X and Y direction. The X-direction interval graph provides decision points for obtaining essential flip-flops, and the Y-direction interval graph provides a maximal clique for each essential flip-flop.

1.2. Our Contributions

In this article, we propose a flip-flop merging algorithm based on agglomerative clustering. In contrast to previous works on post-placement, flip-flops are merged in a bottom-up fashion. Flip-flop merging is achieved by selecting pair of flip-flops with least increase to signal wirelength and least perturbation to original placement. Unlike those of Chang et al. [2010] and Wang et al. [2011], our approach avoids identifying a maximal clique, which is unnecessary when the multibit flip-flop library is limited. In addition to flip-flop merging, we further investigate the effects of the relocation of merged flip-flops on clock network power consumption. In brief, our key contributions are summarized as follows.

—The proposed flip-flop merging algorithm can efficiently minimize increase in signal wirelength and perturbation to original placement. Compared to previous works on flip-flop merging, the signal wirelength connecting to flip-flops can be reduced up to 16.2% and the displacement of flip-flops up to 155.1%.

—Even with additional consideration of the increase in signal wirelength and perturbation to original placement, the proposed flip-flop merging algorithm can still reduce number of flip-flops by an additional 4.8% and 8.6% compared to Wang et al. [2011] and Chang et al. [2010], respectively.
—In consideration of the subsequent clock tree synthesis (CTS), we propose a CTS-driven flip-flop merging which can reduce wirelength in the clock network by 7.82%.

The organization for the rest of the article is as follows. Section 2 formulates the problem of flip-flop merging and relocation. Section 3 describes the construction of the intersection graph using the line sweep method. Section 4 introduces the methodology of flip-flop merging by selecting appropriate merging candidates. Section 5 demonstrates how to relocate merged flip-flops in favor of clock tree synthesis. Section 6 presents experimental results and Section 7 concludes.

2. PROBLEM FORMULATION

Each flip-flop is connected to an input pin and an output pin. The location of pins cannot be changed. Thus, change in total signal wirelength can only result from change in distance between each pair of pin and flip-flop. The signal wirelength of a given flip-flop is defined as the summation of the Manhattan distance from a flip-flop to all the pins it connects to. Here, the given input and the problem of flip-flop merging are defined as follows.

—*Area and power value for each flip-flop*. Generally, an m-bit flip-flop with a larger m value has the benefit of lower power and area per bit.
—*A netlist of input and output pins to each flip-flop*. An m-bit flip-flop has $2m$ pins. For example, a one-bit flip-flop has one input and output pin and a two-bit flip-flop has two input pins and two output pins.
—Location of every pin and flip-flop.
—A slack constraint that is converted to distance for each pair of flip-flop and pin.
—The given layout is partitioned into a set of bins: each bin is given with a predefined placement density constraint.

The Multibit Flip-Flop Merging Problem. Given a placed design, a set of m-bit flip-flop libraries, the netlist of flip-flops, and the position of the flip-flops and input/output pins connected to the flip-flops, minimize the total number of flip-flops by merging flip-flops to higherbit multibit flip-flop with minimal increase to signal wirelength and minimal perturbation to original placement. The placement of flip-flops needs to satisfy the following two constraints.

—*Slack Constraint.* The final placement of the merged flip-flop must be placed in a position that does not violate any slack constraint for every pin it connects to.
—*Placement Density Constraint.* The final placement density after flip-flop merging must not violate the placement density constraint for all bins.

The primary objective of this work still focuses on the number of flip-flop reductions. However, increase in signal wirelength and disruption to original placement is taken into consideration when determining merging priority. Note that the given input and constraints are entirely different from the benchmark used in Lee and Markov [2011]. Since the ISPD 2005 benchmark does not specify the gate type of the input node, Lee and Markov [2011] randomly select a set of nodes to be flip-flops and consider the location of flip-flops during placement stage. In this work and the aforementioned previous work, flip-flop merging [Jiang et al. 2011; Wang et al. 2011; Chang et al. 2010] is performed at the post-placement stage. The given input is a set of placed designs

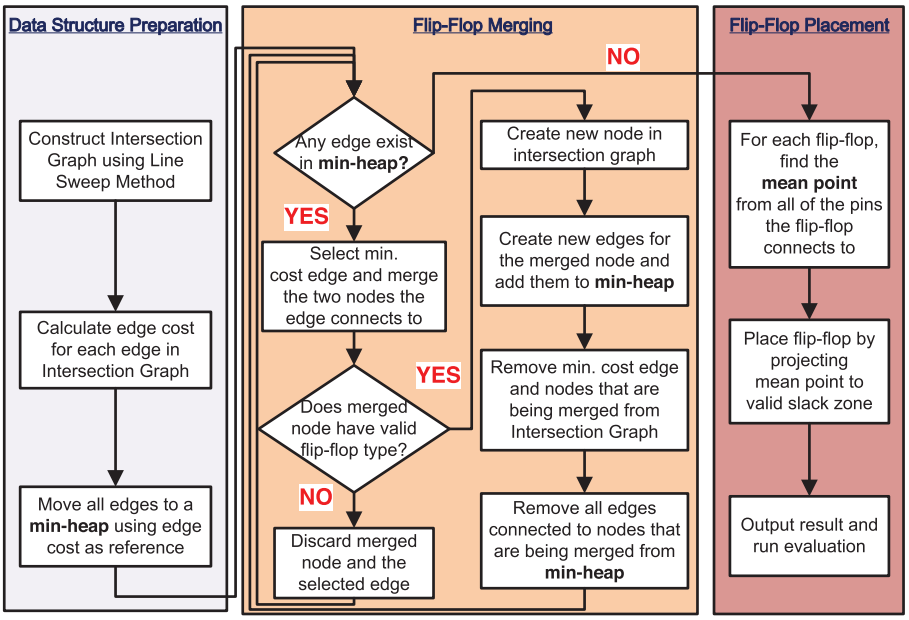


Fig. 3. Flowchart of flip-flop merging based on agglomerative clustering.

where only the location of flip-flops can be changed while not violating the given slack and density constraints.

3. CONSTRUCTION OF THE INTERSECTION GRAPH

Figure 3 is a flowchart on flip-flop merging. The proposed methodology on flip-flop merging is divided to three main parts; the first part constructs an intersection graph using the line sweep method and determines edge cost for all edges. The second part merges flip-flops based on agglomerative clustering by selecting mergeable candidate with the lowest cost. Finally, the third part places flip-flops by projecting optimal locations to valid mergeable zones followed by a breadth-first search to look for a valid location in which to place the merged flip-flop.

The slack value for a flip-flop can be treated as a distance budget. The farther away a flip-flop is moved from its input pins or output pins, the less slack value a flip-flop has. In this work, the slack value for a flip-flop is modeled as a distance budget. We define the *movable zone* of a flip-flop as the region where a flip-flop can be arbitrary positioned without violating any of the slack constraints. Since distance between two points in a chip design is measured in Manhattan distance, the movable zone of a flip-flop is a 45-degree tilted rectangle. Variable A and variable B are used to represent the four y-intercepts of the tilted rectangle. The tilted rectangle is illustrated in Figure 4.

The *mergeable zone* of two flip-flops is defined as the overlapping region of two flip-flops' movable zone. Figure 5 is an illustration of a mergeable zone of two flip-flops. The mergeable zone can be regarded as an edge in graph representation. To merge two flip-flops, the movable zones of the two flip-flops must overlap such that the merged flip-flop can be placed inside the overlapping region without violating any of the slack constraints for both flip-flops. For any pair of flip-flops ff_i and ff_j, an edge e_{ij} exists if and only if the movable zones of ff_i and ff_j overlap one another.

To find all of the merge zones, the naive implementation has time complexity of $O(N^2)$, which is compared to every movable zone of a flip-flop with every other movable

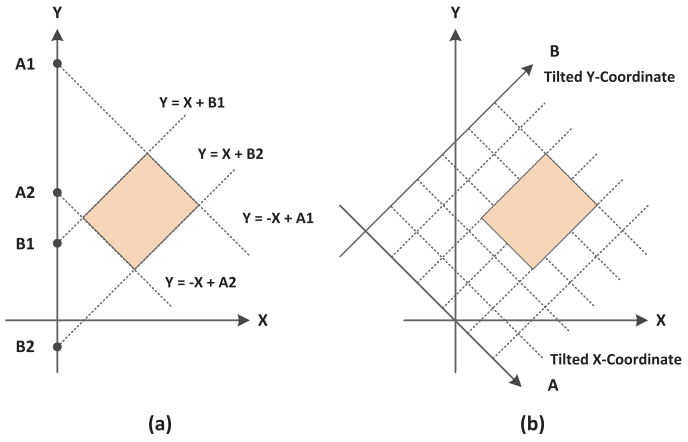


Fig. 4. The rotated coordinate system for representing movable and mergeable zones of a flip-flop. (a) Illustration of the rotated coordinate system by 45 degrees. (b) Coordinates for the rotated rectangle using y-intercept.

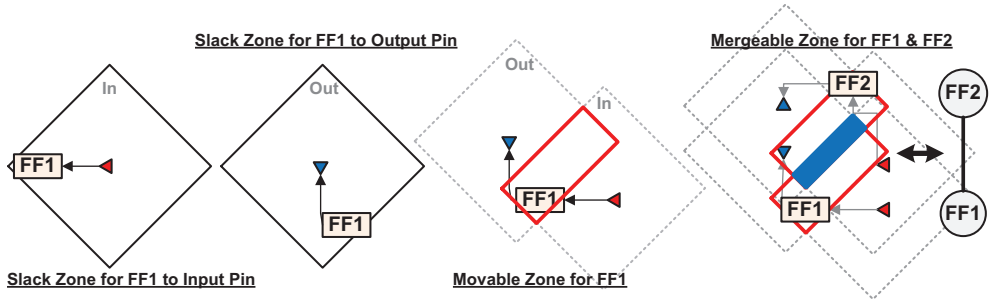


Fig. 5. Illustration of the movable zone for a flip-flop and the mergeable zone for multiple flip-flops. The movable zone is the region where a flip-flop can be arbitrary placed without violating any of the slack constraints. The mergeable zone is the overlapping region between two movable zones for two flip-flops.

zones. However, this approach iterates through all possible scenarios, which is inefficient. A more efficient approach is to sort all the of movable zones on the X-coordinate and then find overlapping segments in the Y-coordinate. This approach, derived from Cormen et al. [2001], can effectively reduce the number of comparisons. A similar technique for identifying mergeable zones is also adopted in Wang et al. [2011].

Algorithm 1 describes the procedure for constructing intersection graph for N number of the flip-flops. First, the sorted left and right X-coordinates of all movable zones are stored in an array X. A sweep line begins to scan the elements in array X. During line sweep, if the selected element corresponds to the left X-coordinate of a certain movable zone, the movable zone is stored to a container P and checks for overlap with all other movable zones stored in P. Since all movable zones are already sorted by their X-coordinates, only the Y-coordinate needs to be examined to determine whether two movable zones have overlap in Y-coordinates. When the right X-coordinate of a movable zone is selected during line sweep, the movable zone is removed from container P. The construction of the intersection graph is complete when the line sweep reaches the end of array X.

Figure 6 demonstrates an example using the line sweep method. In Figure 6(a), the sweep line first enters movable zone A, and movable zone A is stored in P. In

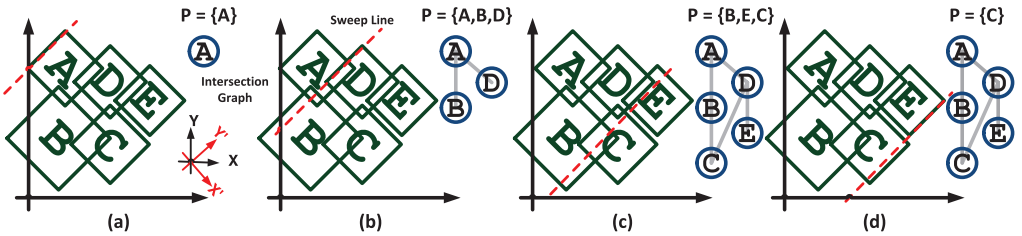


Fig. 6. The sweep line method first sorts the rectangle based on its X-coordinate. When the sweep line touches a rectangle, the rectangle is compared with all the rectangles stored in P. When a sweep line leaves a rectangle, that rectangle is removed from P.

ALGORITHM 1: Line Sweep Method to Identify Overlapping Rectangle

```
for i = 0 → i ≤ N − 1 do
    X ← X ∪ n_{i,leftX} ∪ n_{i,rightX}
end
Sort X in non-decreasing order for x ∈ X do
    if x is leftX for n_i then
        for n_j ∈ P do
            if Segment-Overlap(n_i, n_j)=true then
                Create new edge e_ij between n_i and n_j
            end
        end
        P ← P ∪ n_i
    end
    else
        P ← P - n_i
    end
end
```

Figure 6(b), the sweep line enters movable zone D. Movable zone D is then compared with movable zone A and B to check whether if there exists an overlapping region. In Figure 6(c), the sweep line leaves movable zone D, which is then removed from P. Finally, in Figure 6(d), the sweep line leaves movable zone E, and movable zone E is removed from P.

In the worst-case scenario in which every movable zone overlaps with every other movable zone, the graph representation of such a scenario is a complete graph. The time complexity for the line sweep method in a complete graph is $O(N^2)$, since no movable zone will be removed from P during the line sweep, and every inserted movable zone must compare with every other movable zone stored in P. Table I analyzes the distribution of flip-flops for obtained testcases. Each flip-flop has a movable zone. During the line sweep, each scan needs to compare with an average 4.00% of flip-flops. The maximum size of P or the maximum overlaps among all movable zones ranges from 1.18% to 43.88% of the total number of flip-flops.

4. FLIP-FLOP MERGING BASED ON AGGLOMERATIVE CLUSTERING

After the intersection graph is constructed, the main challenge lies in determining an appropriate objective cost for agglomerative clustering for optimization [Ward 1963]. In order to achieve more flip-flop reduction while considering increases in signal wirelength, the edge cost should reflect the following two criteria: (1) number of edges removed from the intersection graph and (2) increase in signal wirelength. The first

Table I. Maximum and Average Size of P

Testcase	FF#	Max. Size P	Avg. Size P	Max. Size P/FF#	Avg. Size P/FF#
c1	98	43	12	43.88%	12.24%
c2	423	92	24	21.75%	5.67%
c3	1,692	187	50	11.05%	2.96%
c4	5,128	349	86	6.81%	1.68%
c5	10,575	472	124	4.46%	1.17%
c6	169,200	1,993	499	1.18%	0.29%
Avg.	**—**	**—**	**—**	**14.86%**	**4.00%**

criteria aims to minimize perturbation to the intersection graph, and the second criteria aims to minimize perturbation to original placement.

4.1. Agglomerative Clustering of Flip-Flops

To simultaneously consider both criteria into edge cost, the Euclidean distance between two flip-flops is used to determine the edge cost. Although signal wirelength is measured in Manhattan distance, it does not accurately reflect the magnitude of displacement. Using Euclidean distance is more accurate in reflecting the distance between two flip-flops. Displacement of flip-flops can be minimized when flip-flops are merged within the shortest distance. Minimal displacement implies less increase to signal wirelength and less disruption to original placement.

To calculate the edge cost between flip-flop i and flip-flop j, mean points M_i and M_j of all pins connected to flip-flop i and flip-flop j are calculated. The edge cost between flip-flop i and flip-flop j is set to the Euclidean distance between M_i and M_j. In Equation (1), M_i is the mean point of all pins connected to flip-flop i, where n is the total number of pins connected to flip-flop i. In Equation (2), $Cost_{i,j}$ is the edge cost between flip-flop i and flip-flop j.

$$M_i = (M_{i,x}, M_{i,y}) = \left(\frac{\sum_{i=0}^{i \le n-1} p_{i,x}}{n}, \frac{\sum_{i=0}^{i \le n-1} p_{i,y}}{n} \right); \tag{1}$$

$$Cost_{i,j} = \sqrt[2]{(M_{i,x} - M_{j,x})^2 + (M_{i,y} - M_{j,y})^2}. \tag{2}$$

For two flip-flops to be mergeable, a valid n-bit flip-flop in a given flip-flop library[1] is required. For example, if FF-A is one-bit, FF-B is two-bit, and a three-bit flip-flop does not exist in the given flip-flop library, FF-A and FF-B can not be merged. During flip-flop merging, the edge with minimum cost is selected. If two flip-flops can be successfully merged, a new node representing the merged flip-flop is added to the intersection graph. Flip-flops being merged and edges connected to them are removed from the intersection graph. New edges connected to merged flip-flops are added to the intersection graph.

The algorithm terminates when there is no more edge left in the intersection graph. In Figure 7(a), the edge that connects node A and node B, is selected. Node A, node B, and all edges connected to node A and node B are removed from the intersection graph. Then, the new node M_{AB} is created and added to the intersection graph. Figure 7(c) illustrates an example that merges M_{AB} and node M_{CD} to a four-bit node M_{ABCD}. A new edge is created between two nodes if and only if all the nodes form a clique in the intersection graph. Since there is no edge between M_{AB} and E, no new edge is created

[1]Agglomerative clustering can handle situations in which a certain flip-flop type is not available in the given library. A pseudo flip-flop type can be added to overcome this issue. When clustering is completed, clusters with this pseudo flip-flop type is recursively declustered until no pseudo flip-flop type is found for all clusters.

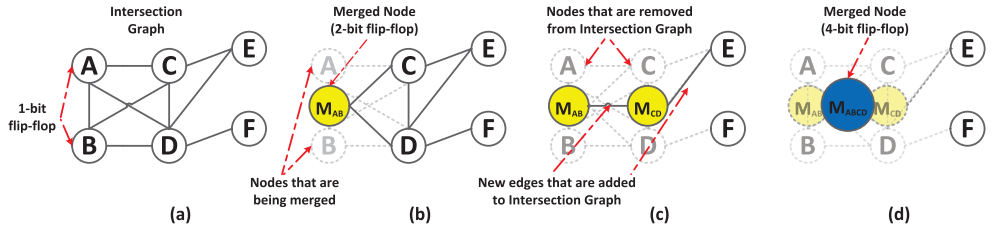


Fig. 7. Example of agglomerative clustering in flip-flop merging. (a) Intersection graph; (b) nodes A and B are merged to a two-bit node M_{AB}, and new edges are created between node M_{AB} to C and M_{AB} to D; (c) nodes C and D are merged to a two-bit node M_{CD}, and a new edge is created between M_{AB} and M_{CD}; (d) node M_{AB} and M_{CD} are merged to a four-bit node M_{ABCD}.

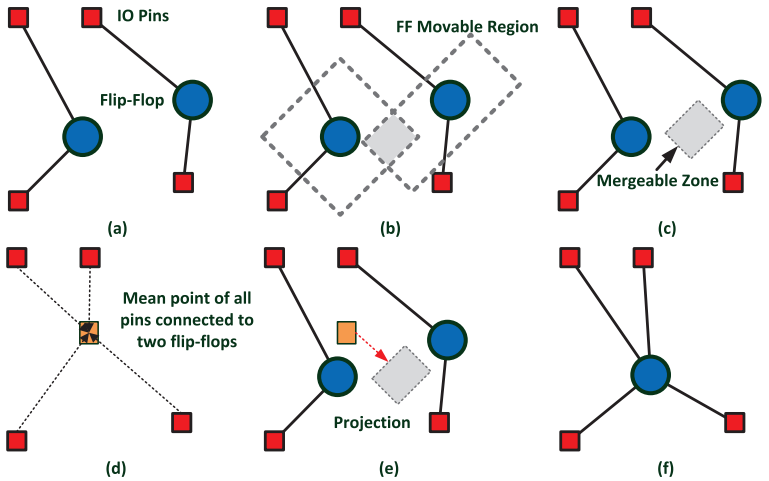


Fig. 8. Placement of merged flip-flops. Blue circle denotes flip-flop and red square denotes pin. (a) Original flip-flop location; (b) movable zone of flip-flops; (c) mergeable zone of two flip-flops; (d) mean point of all pins connected to flip-flops that are being merged; (e) projection of mean point to mergeable zone; (f) placement of merged flip-flop.

between M_{ABCD} and E. The algorithm terminates in Figure 7(d), since there is no more edge in the intersection graph.

4.2. Placement of Merged Flip-Flops

For a merged flip-flop to be successfully relocated, three conditions must be satisfied.

—Slack constraint must be met for all flip-flops.
—Placement density for all bins can not be violated.
—The position of the merged flip-flop must not overlap with other merged flip-flops.

The given layout is partitioned into a set of bins with a given placement density constraint. Figure 8 illustrates an example of merged flip-flop placement while satisfying the density constraint. To place a merged flip-flop, the mean point of all the pins connected to merged flip-flop is first calculated. Then the location of the mean point is projected onto the boundary of the mergeable zone of two flip-flops.

The projection point is selected as the relocation point. Before placing the merged flip-flop at the relocation point, the relocation point is first checked to determine whether it is being occupied and whether the placement density constraint of the placed bin is

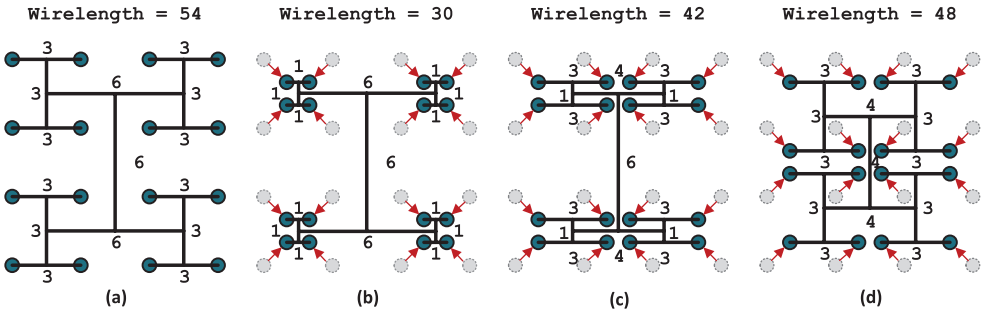


Fig. 9. Illustration on reduction in clock network wirelength by moving sinks to different locations. (a) Original position of sinks, wirelength = 54 unit; (b) position of sinks after moving sinks toward the position of the grandparent node in binary tree representation of merging points, wirelength = 30 unit; (c) Position of sinks after moving sinks toward the position of the great-grandparent node in binary tree representation of merging points, wirelength = 42 unit; (d) position of sinks after moving sinks toward the mean point of all sinks, wirelength = 48 unit.

violated. If violated, a breadth-first search is conducted to search for the nearest unoccupied point in which both slack and placement density constraints are not violated.

5. CLOCK TREE SYNTHESIS-DRIVEN FLIP-FLOP MERGING AND RELOCATION

The relocated position of a merged flip-flop is essential to the power consumption of the clock tree. Without considering clock tree topology, a merged flip-flop only needs to be placed in a position that minimizes increase in signal wirelength. However, within the scope of clock tree synthesis, the positions of flip-flops are preferably placed in clusters. Thus, when relocating merged flip-flops, the merging points in the clock tree should be taken into consideration. Thus, as an extended version to the original flip-flop merging problem, the problem of CTS-driven flip-flop merging is defined as follows.

Clock Tree Synthesis-Driven Flip-Flop Merging Problem. Given a set of m-bit flip-flops with corresponding location and the netlist of flip-flops, place the merged flip-flops in clusters where clock tree wirelength can be reduced.

To consider clock tree topology during flip-flop merging, a pseudo clock tree based on Chang et al. [2012] is constructed every time after flip-flop merging. Chang et al. [2012], constructed clock tree in a bottom-up fashion, and the merging points of the clock tree are represented by a binary tree. The organization of this section is as follows. We first explain on how to choose an appropriate sink position and then describe the framework integrating CTS with flip-flop merging.

5.1. Selecting the Appropriate Position to Relocate Merged Flip-Flops

A Deferred Merge Embedding (DME) [Edahiro 1993] based clock tree can be represented by a binary tree in which each sink is represented by a leaf node and nodes in each hierarchy level of binary tree represents merging points at different stages. A common practice in clock tree optimization is to position sinks in cluster such that the clock tree wirelength can be reduced [Papa et al. 2011]. Moving sinks towards the merging points in clock tree can orient placement of flip-flop such that they are placed in compact fashion. In this work, the clustering of sinks is achieved by moving sinks closer to the merging points of clock tree. Figure 9 illustrates different scenarios on moving sinks towards different merging points. Through empirical analysis, moving sinks toward the position of grandparent node in binary tree representation of merging points is most effective in reducing wirelength of clock network.

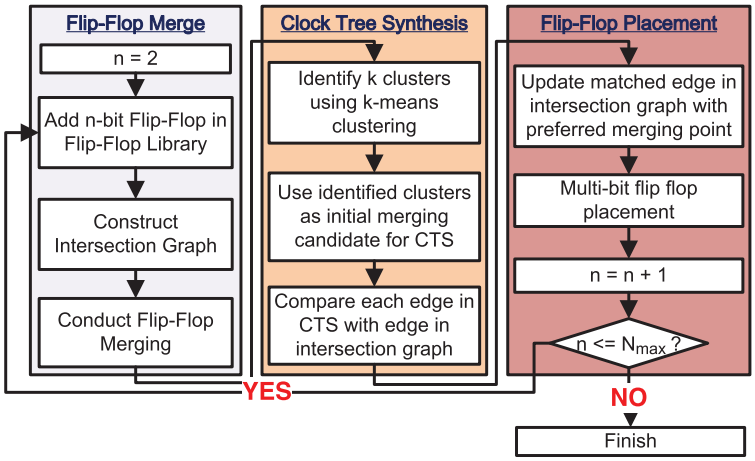


Fig. 10. Flowchart of integrating clock tree synthesis in flip-flop merging. N_{max} denotes the given maximum bit number of multibit flip-flop.

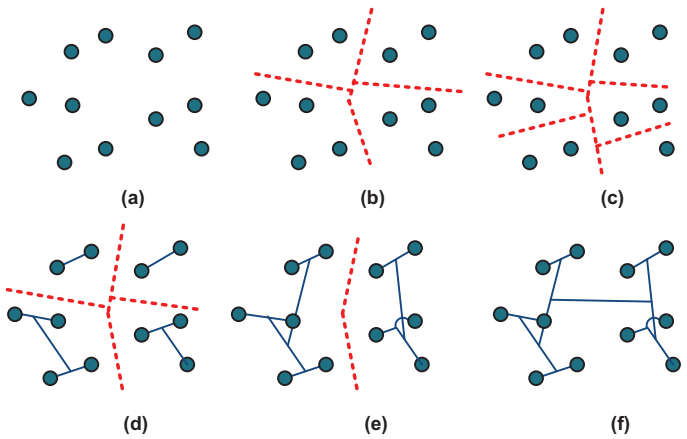


Fig. 11. Illustration on selecting an initial merging candidate of sinks using k-means clustering.

5.2. Integrating Clock Tree Synthesis with Flip-Flop Merging

In this section, we present the CTS-driven flip-flop merging algorithm. Two graphs are constantly reconstructed. One graph is the intersection graph in flip-flop merging, and the other graph is the binary tree representation of merging points in clock tree. Figure 10 is a flow chart of the proposed framework.

A pseudo CTS is performed every time after flip-flop merging. During construction of the clock tree, the selection of pairs of sinks to be merged significantly affects the quality of the clock tree. To achieve zero skew, a poor selection of merging candidates creates additional snaking wirelength, which increases the power consumption. To obtain a global view over the distribution of sinks, k-means clustering is performed to identify cluster of sinks within near proximity. The identified clusters then serve as the initial merging candidates for CTS. Figure 11 is an example using k-means clustering to construct clock tree topology. A similar technique is also applied in Papa et al. [2011] for latch bundling. When pseudo CTS is completed, each connected pair of nodes in CTS is compared with the edges in the intersection graph. If there is a match between

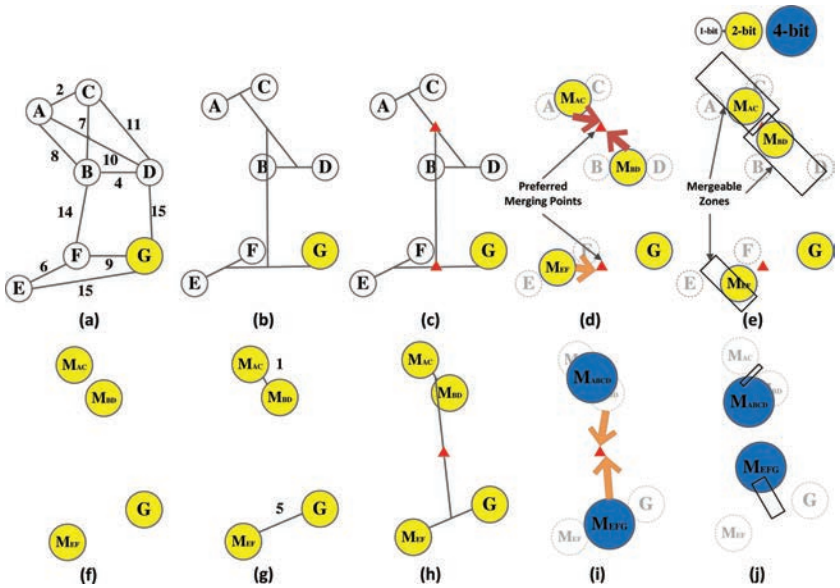


Fig. 12. An example of CTS-driven flip-flop merging. (a) Original intersection graph; (b) synthesized clock tree of the initial sink position; (c) identified preferable merging points for relocated merged flip-flops; (d)–(e) the merged two-bit flip-flop is moved as close to its preferable merging point as possible; (f)–(g) intersection graph after merging two-bit flip-flops; (h) identified preferable merging point; (i)–(j) the merged four-bit flip-flop is moved as close to its preferable merging point as possible.

a connected pair of sinks in CTS with an edge in the intersection graph, the edge in the intersection graph will be updated with a preferred merging point.

In the original flip-flop merging problem, flip-flops are iteratively merged until there is no more edge left in the intersection graph. To integrate with CTS, pseudo CTS needs to be performed to determine a preferable merging location before flip-flops can be merged in the next iteration. The preferable merging location for a pair of sinks is the position of the mutual grandparent node in a binary tree representation of a DME-based clock tree.

The detail of the framework is described as follows. The flip-flop library begins with one-bit or two-bit flip-flop type. One iteration of CTS is performed to construct the binary tree. Flip-flop merging is performed after one iteration of CTS; the maximum bit number of the multibit flip-flops is limited to two-bit. If there is a matched edge between the binary tree of the DME-based clock tree and the intersection graph, the merged flip-flop is positioned as close to the preferable merging point as possible. Figure 12 is an example of CTS driven flip-flop merging. Figure 12(a) is the intersection graph and Figure 12(b) is the constructed clock tree. Figure 12(c) identifies the position of the merging point, as illustrated in Figure 12(d). In Figure 12(e), the merged flip-flop is positioned as close to the preferred merging point without violating slack constraint. Figure 12(g)–(j) repeat the process by adding four-bit flip-flops to the flip-flop library. The process terminates when all flip-flop types are added to the flip-flop library.

6. EXPERIMENTAL RESULT

To evaluate the effectiveness of our proposed algorithm, our algorithm is compared with previous works on flip-flop merging [Jiang et al. 2011; Wang et al. 2011; Chang et al. 2010]. Six testcases (c1∼c6) and flip-flop libraries are obtained

Table II. Comparison with Original Input and Jiang et al. [2011] on Number of Flip-Flop Reductions

	Original			[Jiang et al. 2011]			Our work on FF merging		
#	1/2/4-bit FF	Total	Norm.	1/2/4-bit FF	Total	Norm.	1/2/4-bit FF	Total	Norm.
c1	76,22,0	98	2.722	0,4,28	32	0.889	2,9,25	36	1.000
c2	366,57,0	423	2.938	0,6,117	123	0.854	8,36,100	144	1.000
c3	1464,228,0	1692	2.958	0,14,473	487	0.851	28,142,402	572	1.000
c4	4378,751,0	5128	2.922	2,21,1459	1482	0.844	80,450,1225	1755	1.000
c5	9150,1425,0	10575	2.971	2,33,2983	3018	0.848	160,880,2520	3560	1.000
c6	146400,22800,0	169200	2.977	18,113,47939	48070	0.846	2440,14020,40380	56840	1.182
Avg.	—	—	**2.914**	—	—	**0.856**	—	—	**1.000**

Table III. Comparison with Wang et al. [2011] and Chang et al. [2010] on Number of Flip-Flop Reductions

	[Wang et al. 2011]			[Chang et al. 2010]			Our work on FF merging		
#	1/2/4-bit FF	Total	Norm.	1/2/4-bit FF	Total	Norm.	1/2/4-bit FF	Total	Norm.
c1	6,7,25	38	1.056	8,10,23	41	1.139	2,9,25	36	1.000
c2	16,30,101	147	1.021	24,36,96	156	1.083	8,36,100	144	1.000
c3	70,125,400	595	1.040	84,146,386	616	1.077	28,142,402	572	1.000
c4	232,402,1211	1845	1.051	242,469,1175	1886	1.075	80,450,1225	1755	1.000
c5	484,806,2476	3766	1.058	480,920,2420	3820	1.073	160,880,2520	3560	1.000
c6	7580,13508,39351	60439	1.063	7320,14780,38780	60880	1.071	2440,14020,40380	56840	1.182
Avg.	—	—	**1.048**	—	—	**1.086**	—	—	**1.000**

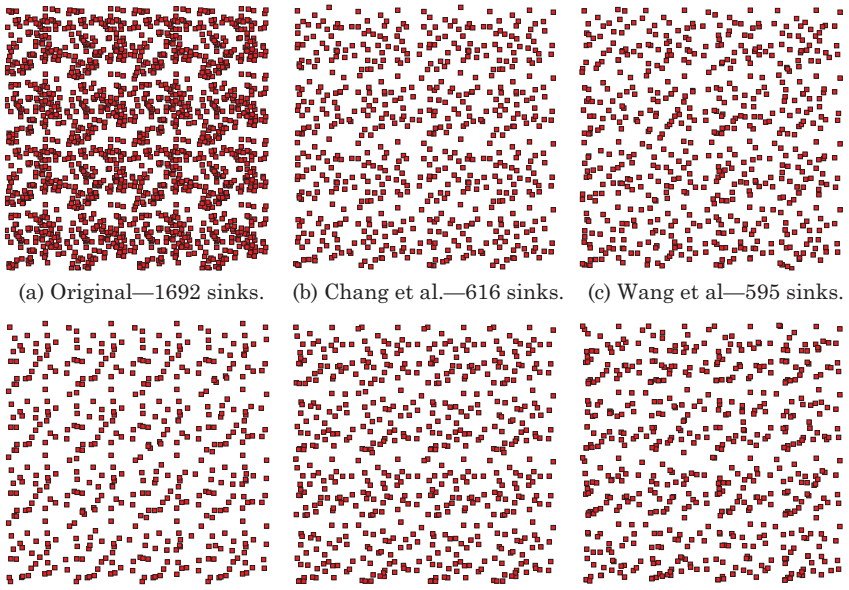
[Chang et al. 2010]. Binary executables of the works [Jiang et al. 2011; Wang et al. 2011; Chang et al. 2010] are obtained from their respective corresponding authors. Executables from Jiang et al. [2011], Wang et al. [2011], and our work on flip-flop merging are performed under an Intel Xeon CPU 5160 Cent OS workstation running at 3.0 GHz. Executables from Chang et al. [2010] are performed under an Intel Core i3 CPU 550 Ubuntu workstation running at 3.2 GHz, since it is compiled with g++ 4.5.4.

An evaluator is implemented to evaluate the number of flip-flop reductions, signal wirelength, and displacement of flip-flops. The evaluated results for executables are verified with the corresponding authors [Chang et al. 2010; Wang et al. 2011; Jiang et al. 2011]. Note that Wang et al. [2011] has a parameter λ to adjust weighting between the number of clock sinks and the signal wirelength and Jiang et al. [2011] have an additional signal wirelength-driven mode. Both binaries we obtained are configured to maximize the number of flip-flop reductions.

The CTS engine from Chang et al. [2012] is used to perform clock tree synthesis for all binaries after flip-flop merging is completed. This CTS engine is a DME-based CTS engine which can meet stringent slew and slack constraints considering process variation in the ISPD 2011 Clock Tree Synthesis Contest [Sze 2010]. A converter is implemented to convert the result after flip-flop merging to ISPD 2011 CTS benchmark format. In this work, process variation is set to zero, since the effect of process variation is not our primary concern.

6.1. Flip-Flop and Signal Wirelength Reduction

Tables II and III present the results of flip-flop number reduction. Compared with the original benchmark without flip-flop merging, our algorithm can reduce the number of flip-flops by 65.7%. In comparison with Chang et al. [2010] and Wang et al. [2011], our proposed algorithm has 8.6% and 4.9% less flip-flops, respectively. In comparison with Jiang et al. [2011], our algorithm have an additional 16.9% more flip-flops. Figure 13 illustrates distribution of sinks after flip-flop merging.



(a) Original—1692 sinks. (b) Chang et al.—616 sinks. (c) Wang et al—595 sinks.

(d) Jiang et al.—487 sinks. (e) Liu et al.—572 sinks. (f) Clock–Driven FF-Merge—585 sinks.

Fig. 13. Placement of flip-flops for testcase c3 after flip-flop merging. (a) Original placement of flip-flops; (b) placement of flip-flops after flip-flop merging in Chang et al. [2010]; (c) placement of flip-flops after flip-flop merging in Wang et al. [2011]; (d) placement of flip-flops after flip-flop merging in Jiang et al. [2011]; (e) placement of flip-flops after our work on flip-flop merging; (f) placement of flip-flops after our work on CTS-driven flip-flop merging.

Table IV. Comparison on Increase in Signal Wirelength and Execution Time after Flip-Flop Merging

	[Jiang et al. 2011]			[Wang et al. 2011]			[Chang et al. 2010]			Our FF merging		
#	WL(nm)	Norm.	Sec.	WL(nm)	Norm.	Sec.	WL(nm)	Norm.	Sec.	WL(nm)	Norm.	Sec.
c1	8606500	1.061	0	8215500	1.013	0	8196500	1.010	0.01	**8112500**	1.000	0
c2	35495000	1.162	0	31017000	1.016	0.06	33032000	1.081	0.03	**30534000**	1.000	0.02
c3	144232000	1.174	0.04	123585000	1.006	0.29	132321000	1.078	0.07	**122790000**	1.000	0.13
c4	445319500	1.174	0.11	379692500	1.001	0.79	405594500	1.069	0.2	**379287500**	1.000	0.5
c5	911912000	1.207	0.25	**755695000**	1.000	1.95	827580000	1.095	0.49	769890000	1.019	1
c6	14654546000	1.191	3.31	**12309247000**	1.000	36.45	13245195000	1.076	92.64	12338170000	1.002	21.86
Avg.	—	**1.162**	**1.00**	—	**1.006**	**11.01**	—	**1.068**	**27.99**	—	**1.004**	**6.60**

In Table IV, regarding an increase in signal wirelength, greedy flip-flop number reduction in Jiang et al. [2011] results in most increase in signal wirelength on all six testcases. In contrast, our work have 16.2% less signal wirelength compared to Jiang et al. [2011]. In comparison to Wang et al. [2011] and Chang et al. [2010], our work has 0% to 6.8% less signal wirelength while having more flip-flop number reductions.

When switching activity is high, an additional increase in dynamic power of signal wirelength may saturate power reduction in clock network. Table V presents total capacitance, including clock network and signal wirelength, in response to different adjustment of activity factor. Unit wire capacitance is set to 0.0002fF/nm, which follows the configuration in the ISPD 2010 CTS Contest [Sze 2010]. Switching activity is set to 1.00 for clock network and incrementally adjusted from 0.00 to 0.20 for signal

Table V. Total Capacitance Including Clock Network Capacitance (Buffer + Sink + Wire) and Capacitance of Signal Wirelength Multiplied by Different Activity Factor

		[Jiang et al. 2011]			[Wang et al. 2011]			[Chang et al. 2010]			[Liu et al. 2012]		
		Capacitance (fF)			Capacitance (fF)			Capacitance (fF)			Capacitance (fF)		
	α	Sig.	Ttl.	Nm.	Sig.	Ttl.	Nm.	Sig.	Ttl.	Nm.	Sig.	Ttl.	Nm.
c1	0.00	0.00	1210.71	1.01	0.00	1202.28	**1.00**	0.00	1224.43	1.02	0.00	1202.88	1.00
	0.05	86.07	1296.77	1.01	81.97	1284.24	1.00	86.07	1310.50	1.02	81.13	1284.01	**1.00**
	0.10	172.13	1382.84	1.01	163.93	1366.21	1.00	172.13	1396.56	1.02	162.25	1365.13	**1.00**
	0.15	258.20	1468.90	1.02	245.90	1448.17	1.00	258.20	1482.63	1.03	243.38	1446.26	**1.00**
	0.20	344.26	1554.97	1.02	327.86	1530.14	1.00	344.26	1568.69	1.03	324.50	1527.38	**1.00**
c2	0.00	0.00	4685.12	**1.00**	0.00	4891.21	1.04	0.00	4905.12	1.05	0.00	4815.78	1.03
	0.05	354.95	5040.07	**1.00**	310.17	5201.38	1.03	330.32	5235.44	1.04	305.34	5121.12	1.02
	0.10	709.90	5395.02	**1.00**	620.34	5511.55	1.02	660.64	5565.76	1.03	610.68	5246.46	1.01
	0.15	1064.85	5749.97	1.00	930.51	5821.72	1.02	990.96	5896.08	1.03	916.02	5731.80	**1.00**
	0.20	1419.80	6104.92	1.01	1240.68	6131.89	1.02	1321.28	6226.40	1.03	1221.36	6037.14	**1.00**
c3	0.00	0.00	18282.07	**1.00**	0.00	19053.72	1.04	0.00	19221.31	1.05	0.00	18799.74	1.03
	0.05	1442.32	19724.39	**1.00**	1235.85	20289.57	1.03	1323.21	20544.52	1.04	1227.90	20027.64	1.02
	0.10	2884.64	21166.71	**1.00**	2471.70	21525.42	1.02	2646.42	21867.73	1.03	2455.80	21255.54	1.00
	0.15	4326.96	22609.03	1.01	3707.55	22761.27	1.01	3969.63	23190.94	1.03	3683.70	22483.44	**1.00**
	0.20	5769.28	24051.35	1.01	4943.40	23997.12	1.01	5292.84	24514.15	1.03	4911.60	23711.34	**1.00**
c4	0.00	0.00	51366.42	**1.00**	0.00	54474.26	1.06	0.00	55419.02	1.08	0.00	53382.78	1.04
	0.05	4453.20	55819.62	**1.00**	3796.93	58271.16	1.04	4055.95	59474.96	1.07	3792.88	57175.65	1.02
	0.10	8906.39	60272.81	**1.00**	7593.85	62068.09	1.03	8111.89	63530.91	1.05	7585.75	60968.53	1.01
	0.15	13359.59	64726.01	**1.00**	11390.78	65865.01	1.02	12167.84	67586.85	1.04	11378.63	64761.40	1.00
	0.20	17812.78	69179.20	1.01	15187.70	69661.94	1.02	16223.78	71642.80	1.05	15171.50	68554.28	**1.00**
c5	0.00	0.00	114012.97	**1.00**	0.00	119078.82	1.04	0.00	120844.52	1.06	0.00	118176.71	1.04
	0.05	9119.12	123132.09	**1.00**	7556.95	126635.77	1.03	8275.80	129120.32	1.05	7698.90	125875.61	1.02
	0.10	18238.24	132251.21	**1.00**	15113.90	134192.72	1.01	16551.60	137396.12	1.04	15397.80	133574.51	1.01
	0.15	27357.36	141370.33	1.00	22670.85	141749.67	1.00	24827.40	145671.92	1.03	23096.70	141273.41	**1.00**
	0.20	36476.48	150489.45	1.01	30227.80	149306.62	1.00	33103.20	153947.72	1.03	30795.60	148972.31	**1.00**

Note: α stands for switching activity for signal wirelength, Nm. for normalize, Sig. for capacitance of signal wirelength multiplied by α, Ttl. for summation of capacitance of clock network and signal wirelength multiplied by α.

wirelength. Our work on flip-flop merging has the least total capacitance on 10 out of 25 settings, while Jiang et al. [2011] has the least capacitance on 13 out 25 settings. In general, out work is more suitable for designs when switching activity is 0.15 or above, and Jiang et al. [2011] is more suitable for designs when the activity factor is 0.10 or less.

6.2. Average Displacement Distance for Merged Flip-Flops

In addition to analyzing a flip-flop's power and signal wirelength, we also analyze the average displacement of each merged flip-flop. Minimizing displacement of flip-flops will try to create least perturbation to original placement. Table VI presents the results of the average displacement of merged flip-flops. For each merged flip-flop, we can identify which of the original flip-flops it merged together. Then displacement between the remembering flip-flop's original location and its final relocation position can be calculated. In all six testcases, the original flip-flop is given in one-bit or two-bit. We perform analysis on all of the testcases for each executable and present the results by taking the average displacement for one-bit and two-bit flip-flops. Based on observation, the benefit of agglomerative clustering is two fold: it increases the least signal wirelength and creates the least perturbation to the original layout. In

Table VI. Comparison on Average Displacement Distance of Flip-Flops from Original Position to Relocated Position

Circuit	[Jiang et al. 2011]		[Wang et al. 2011]		[Chang et al. 2010]		Our	
Testcase	1-bit(nm)	2-bit(nm)	1-bit(nm)	2-bit(nm)	1-bit(nm)	2-bit(nm)	1-bit(nm)	2-bit(nm)
c1	**37658**	49309	41875	19500	41426	22333	38270	**19477**
c2	40208	47922	38999	19760	38827	21350	**36989**	**18907**
c3	40717	48353	39820	19761	38399	21121	**37222**	**18907**
c4	41044	48129	40006	20361	38363	21098	**37433**	**19041**
c5	41027	48662	39429	20155	38159	21090	**37362**	**18907**
c6	41021	48898	39810	20395	38047	21103	**37430**	**18907**
Avg.	**40279**	**48546**	**39990**	**19989**	**38870**	**21349**	**37451**	**19027**
Norm.	**1.076**	**2.551**	**1.068**	**1.050**	**1.038**	**1.122**	**1.000**	**1.000**

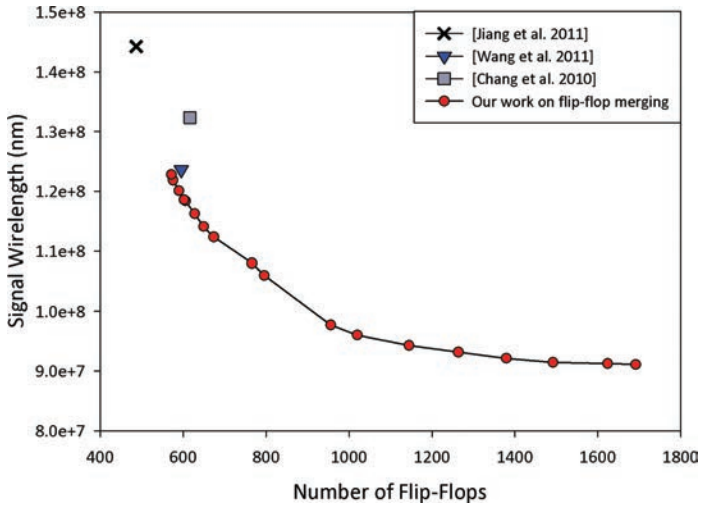


Fig. 14. Illustration on the tradeoff curve between the number of flip-flops and signal wirelength. Our proposed algorithm can obtain an empirically smooth trade-off between signal wirelength reduction and number of flip-flop reductions.

contrast, clique-based approaches [Wang et al. 2011; Chang et al. 2010] greedily reduce the number of flip-flops and neglect information on the physical location of flip-flops. Regarding Table VI, our algorithm has the least displacement out of all works on flip-flop merging.

Our agglomerative-based approach offers flexibility in optimizing between signal wirelength and flip-flop number reduction. An upper bound on flip-flop displacement can be established by limiting the maximum displacement of flip-flops. Figure 14 is a trade-off curve between signal wirelength and flip-flop number reduction by altering the upper bound of flip-flop displacement for test-case c3; it can be observed that our work on flip-flop merging can obtain an empirical smooth trade-off curve. Points on the upper right of the curve [Wang et al. 2011; Chang et al. 2010] are less optimal points.

6.3. Result of Clock Tree Synthesis after Flip-Flop Merging

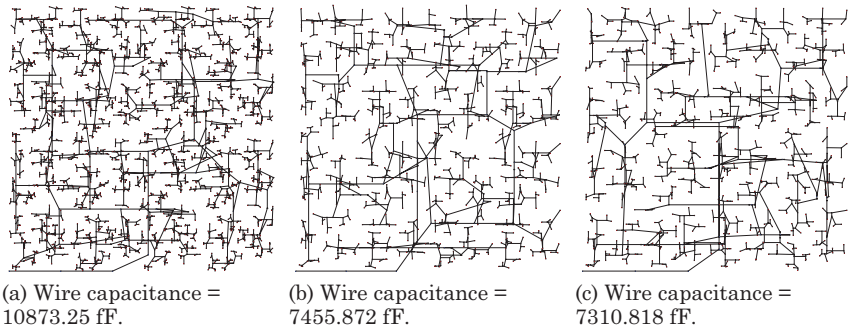
In this section, flip-flop location after flip-flop merging is converted to the ISPD 2010 benchmark format. Slew limit is set to 100 ps; skew limit is set to 7.5 ps; and process

Table VII. Comparison on Capacitance of Clock Network Using Location of Merged Flip-Flops as Input to CTS Engine [Chang et al. 2012]

Circuit	Testcase	c1	c2	c3	c4	c5	Norm.
Original	Skew (ps)	0.200	1.586	2.104	2.069	2.860	—
	Sink (fF)	398.440	1624.140	6496.560	19844.020	40603.500	—
	Buffer(fF)	485.800	1927.800	7036.600	31911.100	65983.500	—
	Wire (fF)	586.426	2604.800	10873.250	34426.621	70841.548	—
	Total (fF)	**1470.666**	**6156.740**	**24406.410**	**86181.740**	**177428.548**	**1.556**
[Jiang et al. 2011]	Skew (ps)	0.408	0.490	0.992	2.063	1.997	—
	Sink (fF)	327.320	1313.760	5249.440	16065.700	32780.020	—
	Buffer(fF)	485.800	1763.000	6377.400	14699.800	38431.000	—
	Wire (fF)	397.587	1608.361	6655.228	20600.921	42801.949	—
	Total (fF)	**1210.707**	**4685.121**	**18282.068**	**51366.421**	**114012.969**	**1.000**
[Wang et al. 2011]	Skew (ps)	0.252	1.145	0.870	2.834	1.829	—
	Sink (fF)	336.140	1339.520	5365.500	16456.160	33584.000	—
	Buffer(fF)	485.800	1763.000	6377.400	16183.000	39883.300	—
	Wire (fF)	380.335	1788.710	7310.818	21835.078	45611.517	—
	Total (fF)	**1202.275**	**4891.230**	**19053.718**	**54474.238**	**119078.817**	**1.037**
[Chang et al. 2010]	Skew (ps)	0.037	0.824	0.681	2.796	1.967	—
	Sink (fF)	339.360	1349.040	5388.040	16501.380	33644.800	—
	Buffer(fF)	485.800	1763.000	6377.400	16183.000	39749.400	—
	Wire (fF)	399.274	1793.078	7455.872	22734.639	47450.315	—
	Total (fF)	**1224.434**	**4905.118**	**19221.312**	**55419.019**	**120844.515**	**1.050**
Our FF-Merge	Skew (ps)	0.015	1.097	0.716	2.407	1.658	—
	Sink (fF)	334.180	1336.720	5342.680	16366.000	33376.000	—
	Buffer(fF)	485.800	1763.000	6377.400	15523.800	39749.400	—
	Wire (fF)	382.901	1716.058	7079.658	21492.976	45051.305	—
	Total (fF)	**1202.881**	**4815.778**	**18799.738**	**53382.776**	**118176.705**	**1.025**
Our CTS-Driven FF-Merge	Skew (ps)	0.297	0.662	2.392	2.324	2.024	—
	Sink (fF)	334.180	1341.200	5357.240	16392.880	33440.960	—
	Buffer(fF)	485.800	1763.000	4811.800	14123.000	39749.400	—
	Wire (fF)	346.219	1557.159	6342.141	20601.140	42482.960	—
	Total (fF)	**1166.199**	**4661.359**	**16511.181**	**51117.020**	**115673.320**	**0.974**

variation is set to 0.00. Capacitance of the sink for a one-bit flip-flop is set to 3.5 fF, 6.02 fF for a two-bit flip-flop, and 10.92 fF for a four-bit flip-flop. Table VII presents the total capacitance of clock network for all works, and Figure 15 presents the synthesized clock tree after flip-flop merging on testcase c3.

According to observations, the biggest gain from sink number reduction comes from power reduction in clock tree wirelength, rather than reduction in sink capacitance. Regarding testcase c5 in Table VII, our work on flip-flop merging yields 542 additional sinks, which reflects a 595.98 fF increase in sink capacitance and 2249.356 fF increase in wire capacitance. The increase in wire capacitance is roughly three times more than the increase in sink capacitance. We can overcome this gap by carefully relocating merged flip-flops in favor of clock tree synthesis. As seen in Table VII, our work on CTS-driven flip-flop merging can reduce wire capacitance in the clock network by an average of 7.82% and can reduce buffer capacitance in the clock network by an average of 6.71%. Figure 16 plots the sink capacitance and wire capacitance of the synthesized clock network. In Figure 16, the magnitude on the reduction of sink capacitance due to flip-flop merging is not as obvious as the reduction in wire capacitance. In Figure 16(b), wire capacitance can be effectively reduced by carefully choosing the relocation positions of merged flip-flops.



(a) Wire capacitance = 10873.25 fF.
(b) Wire capacitance = 7455.872 fF.
(c) Wire capacitance = 7310.818 fF.
(d) Wire capacitance = 6655.228 fF.
(e) Wire capacitance = 7079.658 fF.
(f) Wire capacitance = 6342.141 fF.

Fig. 15. Synthesized clock tree using Chang et al. [2012] for testcase C3 after flip-flop merging. (a) Synthesized clock tree on original location of sinks; (b) synthesized clock tree after flip-flop merging in Chang et al. [2010]; (c) synthesized clock tree after flip-flop merging in Wang et al. [2011]; (d) synthesized clock tree after flip-flop merging in Jiang et al. [2011]; (e) Synthesized clock tree after our work on flip-flop merging; (f) synthesized clock tree after our work on CTS-driven flip-flop merging.

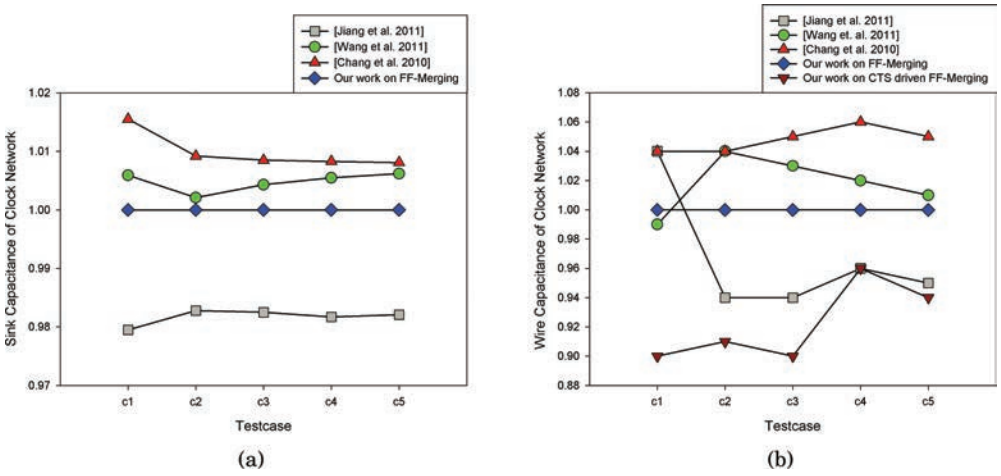


Fig. 16. Comparison of sink capacitance and wire capacitance in a synthesized clock network. The values of sink capacitance and wire capacitance are normalized to our work in flip-flop merging. (a) Comparison of sink capacitance of a clock network normalized to our work on flip-flop merging; (b) comparison of wire capacitance of a clock network normalized to our work on flip-flop merging.

7. CONCLUSIONS

In this article, we proposed a flip-flop merging algorithm based on agglomerative clustering. Compared to previous works, our proposed algorithm can effectively minimize disruption to original placement while having more precise control over the increase in signal wirelength. In spite of additional considerations regarding increase in signal wirelength and disruption to original placement, our proposed flip-flop merging algorithm still outperforms those of Chang et al. [2010] and Wang et al. [2011] on all aspects in terms of number of flip-flop reductions, increase in signal wirelength, and execution time. Out of all binaries we have obtained, our proposed algorothm has minimum displacement of merged flip-flops. In consideration of clock tree power consumption, we further investigate the effect of flip-flop relocation on clock tree wirelength. Our proposed CTS-driven flip-flop merging can effectively reduce power consumption of the synthesized clock tree after flip-flop merging.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments on improving the quality of this work. Special thanks also go to Professor Mark Po-Hung Lin from National Chung Cheng University for providing binary executable for Chang et al. [2010], Shao-Huan Wang and Professor Wai-Kei Mak from National Tsing Hua University for providing binary executable for Wang et al. [2011], and Professor Iris Hui-Ru Jiang from National Chiao Tung University for providing binary executable for Jiang et al. [2011].

REFERENCES

Chang, Y.-C., Wang, C.-K., and Chen, H.-M. 2012. On construction low power and robust clock tree via slew budgeting. In *Proceedings of the International Symposium on International Symposium on Physical Design*. 129–136.

Chang, Y.-T., Hsu, C. C., Lin, M. P. H., Tsai, Y. W., and Chen, S. F. 2010. Post-placement power optimization with multi-bit flip-flops. In *Proceedings of the International Conference on Computer Aided Design*. 218–223.

Chen, C., Kang, C., and Sarrafzadeh, M. 2002. Activity-sensitive clock tree construction for low power. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 279–282.

Chen, L., Hung, A., Chen, H.-M., Tsai, E., Chen, S.-H., Ku, M.-H., and Chen, C.-C. 2010. Using multi-bit flip-flop for clock power saving by DesignCompiler. In *Proceedings of the Synopsys User Group*.

Cheon, Y., Ho, P.-H., Kahng, A., Reda, S., and Wang, Q. 2005. Power-aware placement. In *Proceedings of the Design Automation Conference*. 795–800.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. 2001. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Donno, M., Macii, E., and Mazzoni, L. 2004. Power-aware clock tree planning. In *Proceedings of the International Symposium on Physical Design*. 138–147.

Edahiro, M. 1993. A clustering-based optimization algorithm in zero-skew routings. In *Proceedings of the Design Automation Conference*. 612–616.

Hou, W., Liu, D., and Ho, P.-H. 2009. Automatic register banking for low-power clock trees. In *Proceedings of the International Symposium on Quality of Electronic Design*. 647–652.

Jiang, I. H.-R., Chang, C.-L., Yang, Y.-M., Tsai, E. Y.-W., and Chen, L. S.-F. 2011. INTEGRA: Fast multi-bit flip-flop clustering for clock power saving based on interval graphs. In *Proceedings of the International Symposium on Physical Design*. 115–122.

Kim, M.-C., Lee, D.-J., and Markov, I. L. 2010. SimPL: An effective placement algorithm. In *Proceedings of the International Conference on Computer-Aided Design*. 649–656.

Lee, D.-J. and Markov, I. L. 2011. Obstacle-aware clock-tree shaping during placement. In *Proceedings of the International Symposium on Physical Design*. 123–130.

Liu, S. S.-Y., Lee, C.-J., and Chen, H.-M. 2012. Agglomerative-based flip-flop merging with signal wirelength optimization. In *Proceedings of the Design Automation Test in Europe Conference Exhibition*. 1391–1396.

Papa, D., Viswanathan, N., Sze, C., Li, Z., Nam, G.-J., Alpert, C., and L.Markov, I. 2011. Physical synthesis with clock-network optimization for large systems on chips. *IEEE Micro*. *31,* 4, 51–62.

SHELAR, R. S. 2007. An efficent clustering algorithm for low power clock tree synthesis. In *Proceedings of the International Symposium on Physical Design*. 181–188.

SZE, C. N. 2010. ISPD 2010 High performance clock network synthesis contest: Benchmark suite and results. In *Proceedings of the International Symposium on Physical Design*. 143–143.

WANG, S.-H., LIANG, Y.-Y., KUO, T.-Y., AND MAK, W.-K. 2011. Power-driven flip-flop merging and relocation. In *Proceedings of the International Symposium on Physical Design*. 107–114.

WANG, Y., ZHOU, Q., HONG, X., AND CAI, Y. 2007. Clock-tree aware placement based on dynamic clock-tree building. In *Proceedings of the International Symposium on Circuits and Systems*. 2040–2043.

WARD, J. H. 1963. Hierarchical grouping to optimize an objective function. *J. Am. Stat. Assoc. 58,* 301, 236–244.

Received July 2012; revised October, December 2012; accepted January 2013