

Agglomerative-Based Flip-Flop Merging with Signal Wirelength Optimization

Sean Shih-Ying Liu, Chieh-Jui Lee and Hung-Ming Chen

Institute of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan

{sniealu.ee96g,jerrylee.ee92}@g2.nctu.edu.tw, hmchen@mail.nctu.edu.tw

Abstract—In this paper, an optimization methodology using agglomerative-based clustering for number of flip-flop reduction and signal wirelength minimization is proposed. Comparing to previous works on flip-flop reduction, our method can obtain an optimal tradeoff curve between flip-flop number reduction and increase in signal wirelength. Our proposed methodology outperforms [1] and [12] in both reducing number of flip-flops and minimizing increase in signal wirelength. In comparison with [9], our methodology obtains a tradeoff of 15.8% reduction in flip-flop’s signal wirelength with 16.9% additional flip-flops. Due to the nature of agglomerative clustering, when relocating flip-flops, our proposed method minimizes total displacement by an average of 5.9%, 8.0%, 181.4% in comparison with [12], [1] and [9] respectively.

I. INTRODUCTION

In modern SoC design flow, recent research in industry and academia discover that flip-flops in advance technology require less driving power. Less driving power for flip-flops imply that multiple flip-flops can be driven by a single inverter. Thus, the possibility of multi-bit flip-flop emerges in nanometer design. Multi-bit flip-flop has the advantage to share common inverter to reduce power consumption per bit and more compact flip-flop’s layout area. In addition, replacing original 1-bit flip-flops with multi-bit flip-flops can reduce both sink number and sink capacitance. Fig.1 illustrates the implementation of a 2-bit flip-flop and Fig.2 illustrates how sink reduction affects the topology of a clock tree. It is experimented in [3] that integrating multi-bit flip-flop library into current commercial tool can significantly reduce clock tree power consumption.

According to [7], clock tree power consumption is responsible for 40% of total power consumption. Thus, optimizing clock tree can effectively reduce total power consumption. Optimization techniques for clock tree including reduction of total clock tree wirelength by replacing registers [10][13][8], optimizing size of buffers [11] or considering activity factor when synthesizing clock tree [2][4].

Implementation of multi-bit flip-flop can also reduce clock tree power consumption by reducing sink number. Less sink number implies shorter clock tree wirelength during clock tree synthesis. However, multi-bit flip-flop has the drawback of increasing in flip-flop’s signal wirelength. For signals with high activity factor, the improvement in power reduction by greedily merging flip-flop will eventually saturate due to increase in flip-flop’s signal wirelength.

A. Previous Works

In terms of optimization of flip-flop at post-placement stage, all three [12][1][9] share same objective by optimizing

This work was supported in part by National Science Council(NSC) of Taiwan under Grant No. NSC1002220E009045.
978-3-9810801-8-6/DATE12/©2012 EDAA

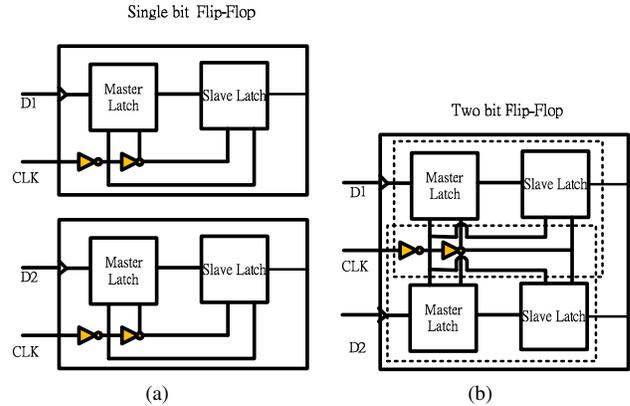


Fig. 1. Inverter sharing using multi-bit flip-flop. (a) are two 1-bit flip-flops. (b) is a 2-bit flip-flop

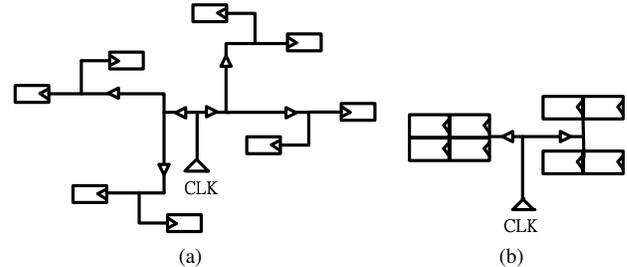


Fig. 2. Reduction of sinks in clock tree. (a) is an illustration of a clock signal connecting to eight 1-bit flip-flop. (b) is an illustration using multi-bit flip-flop.

between flip-flop and signal wirelength reduction. Given a set of flip-flop library, placement density constraint and slack constraint, flip-flop reduction is achieved by merging flip-flop to appropriate higher bit flip-flop. The main difference between the three works lies on selecting groups of flip-flop to be merged.

Chang *et al.* [1] solves the problem by applying progressive window-based optimization. As window sweeps across given layout, an intersection graph will be constructed within the window. The intersection graph is constructed with node representing flip-flop and edge represents that two flip-flops can be safely merged without violating slack constraint. After intersection graph is constructed, maximal independent set of cliques can be identified.

Similar to [1], Wang *et al.* [12] also models the given problem as graph problem. The difference is that [12] applies minimum clique partition algorithm to identify a set of non-conflicting cliques. In addition, a clock tree synthesis using [5] is applied to demonstrate reduction in dynamic power and clock tree wirelength.

In contrast to [12] and [1], Jiang *et al.* [9] propose a linear-size sequence representation to quickly identify optimal clustering combinations. Based on coordinate transformation, [9] generate interval graph corresponding to each flip-flop's feasible region with both X and Y direction. The X direction interval graph provides decision points to obtain essential flip-flops and Y direction interval graph provides maximal clique for each essential flip-flops. Compared to [12] and [1], [9] can achieve most flip-flop reduction within fastest runtime.

B. Deficiency in Previous Work on Flip-Flop Merging and Relocation

In [1], both signal wirelength and flip-flop reduction trails behind [9] and [12]. The inherent characteristic of window-based optimization prevents a global view of the entire problem. In [9], although flip-flop number is greatly reduced, it is achieved with great expense in increasing signal wirelength. Moreover, more flip-flop reductions entail higher perturbation to original layout which add burden to final legalization stage.

C. Our Contributions

In this paper, we propose a post-placement flip-flop based on agglomerative clustering. In contrast to previous work, we merge flip-flop in a bottom-up fashion. Flip-flop merging is performed by choosing pair of flip-flop with least increase to signal wirelength. Such approach can guarantee that total increase in signal wirelength is minimized with equivalent flip-flop number reduction. Unlike [1] and [12], our approach avoids identifying maximum clique which is unnecessary when given multi-bit flip-flop library is limited.

The key concept of agglomerative clustering is to regard each given element as an independent cluster and then progressively cluster elements to a larger cluster. In flip-flop merging, the given elements are the location of flip-flops and each cluster of m flip-flops represents a m -bit multi-bit flip-flop. We formulate given flip-flop merging problem as a graph problem then apply agglomerative clustering to optimize between flip-flop number reduction and increase in flip-flop's signal wirelength.

Nearest Neighbor Selection (NNS) and agglomerative clustering shares similar quality for wirelength minimization. However, NNS restricts one edge for each node which is not suitable to implement flip-flop merging. In flip-flop merging, not every pair of sink can be merged due to slack constraint and limited types of flip-flop in flip-flop library. In addition, to cope with clock tree synthesis, edge cost is required to update constantly. Agglomerative clustering in this regard is much more flexible and its high resemblance to NNS making it an optimal choice for flip-flop merging.

The organization for rest of the paper is as follows. Section II will describe the construction of intersection graph using line sweep method. Section III will introduce the flow of flip-flop merge by selecting appropriate merging candidate. Section IV will present the experimental result and Section V concludes the paper.

II. PROBLEM FORMULATION

The problem of flip-flop merging and relocation for clock tree optimization can be defined as follows. Given a set of m -bit flip-flop library and a netlist of flip-flop with corresponding location, reduce the number of flip-flops by merging flip-flop with minimum increase in signal wirelength.

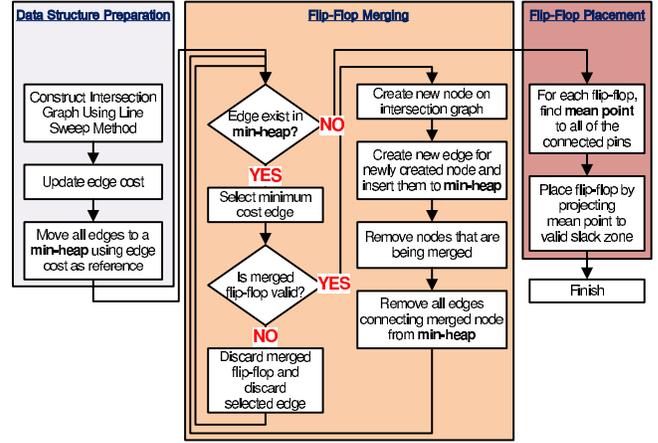


Fig. 3. Flow Chart for agglomerative flip-flop merging.

Since flip-flop merge is conducted at post-placement stage, the location of pins can not be changed. Thus, change in total signal wirelength will only result in change in distance between each pin and flip-flop. The given input contains following information.

- Area and power value for flip-flop library. Generally, a m -bit flip-flop with larger m value has the benefit of lower power and area per bit.
- Connection of input and output pins to each flip-flop. A m -bit flip-flop has $2m$ pins. For example, a 1-bit flip-flop has one input and output pin and a 2-bit flip-flop has two input and output pins.
- Location of every pin and flip-flop.

The constraints for the given problem are described as follows.

A. Slack Constraint

There is a slack value for each pair of flip-flop and pin. 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.

B. Placement Density Constraint

The given layout is partitioned into different set of bins, each bin is given with a placement density constraint and pre-placed combinational logic. The final placement density after flip-flop clustering must not violate the placement density constraint for all bins. The summation of all flip-flop area and the combinational logic area in one bin can not exceed its pre-defined placement density constraint.

III. CONSTRUCTION OF INTERSECTION GRAPH

Fig. 3 is a flow chart on flip-flop merging. The flow is divided into three parts, first part constructs intersection graph using line sweep method to calculate all the available merging candidate. Then, second part will merge flip-flop based on agglomerative clustering. Third part will place flip-flop by projecting optimal location to valid mergeable zone follow by breadth-first search to search for a valid location.

The slack value for a flip-flop can be treated as a distance budget for a flip-flop. The farther away flip-flop moved from its original location, the less slack value a flip-flop has. In this regard, the slack value for a flip-flop can be modeled as a form

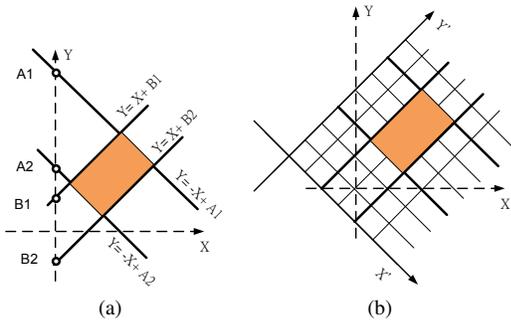


Fig. 4. The rotate coordinate system to describe flip-flop movable zone. (a) Illustration of the rotated coordinate system by 45 degrees. (b) Coordinates for rotated rectangle using y-intercept.

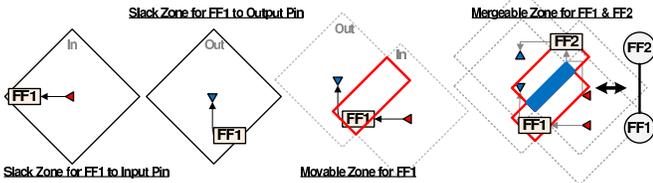


Fig. 5. Illustration of movable zone for a flip-flop and mergeable zone for multiple flip-flop. Movable zone is the overlapping region between slack zone for input and output pin. Mergeable zone is the overlapping region between two movable zones for two flip-flops.

of wire delay. The distance in this context refers to Manhattan distance, thus the region a flip-flop can be relocated without violating the given slack constraint forms a movable zone.

A mergeable zone for two flip-flops is the overlapping region of two flip-flop's movable zone. Fig. 5 is an illustration of mergeable zone to merge two flip-flops. The mergeable zone can be regarded as an edge in graph representation. To merge two flip-flops, the movable zone 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 constraint 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 zone of ff_i and ff_j overlaps one another.

The naive implementation of agglomerative clustering has time complexity $O(N^2)$ which is to compare every node with every other node. However, not every node has an edge connecting to every other node. A more efficient approach is to sort all the rectangles on X-coordinate and find overlapping segment in Y-coordinate which can reduce number of comparisons. This approach is derived from Line Sweep Method[6].

Algorithm 1 describes the procedure to construct graph representation for a N number of flip-flops. Each flip-flop n_i is represented by a 45-degree rotated rectangle corresponding to the movable zone illustrated in Fig. 5. To efficiently represent the rotated rectangles, B and A corresponding to y-intercept of the rectangle illustrated in Fig. 4 is used to represent x and y coordinate respectively.

First, left and right X-coordinates for all rectangles n_i are stored in an array X . The X-coordinates is sorted in non-decreasing order using heap sort. After the array is sorted, an imaginary line starts to sweep from the beginning of the array X . If the x value is the left X-coordinate of a rectangle, the rectangle is stored into a red-black tree P and compared with all of the rectangles stored in P to check whether there exists overlapping region between the two rectangles. Since all rectangles are already sorted by their X-coordinates, if two

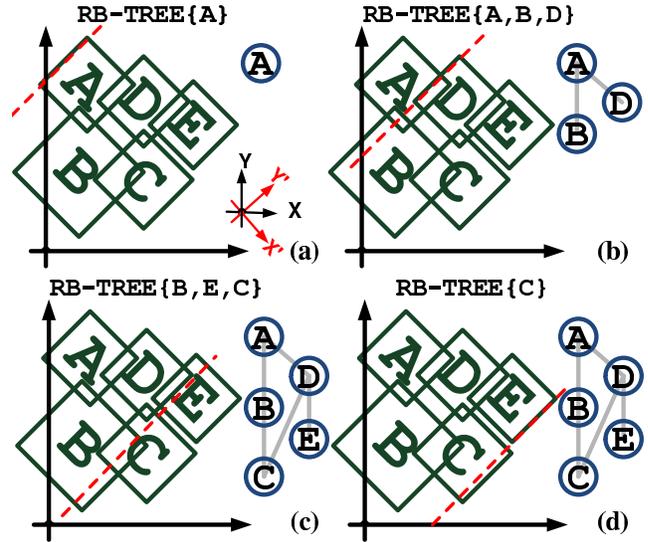


Fig. 6. The sweep line method first sorts the rectangle based on their X'-coordinate. When the sweep line touches a rectangle, the rectangle will compare with all the rectangles stored in the RB-Tree. When a sweep line leaves a rectangle, that rectangle is removed from the RB-Tree.

Algorithm 1 Line Sweep Method to Identify Overlapping Rectangle

```

1: for  $i = 0 \rightarrow i \leq N - 1$  do
2:    $X \leftarrow X \cup n_{i,leftX} \cup n_{i,rightX}$ 
3: end for
4: Sort  $X$  in non-decreasing order
5: for  $x \in X$  do
6:   if  $x$  is leftX for  $n_i$  then
7:     for  $n_j \in P$  do
8:       if Segment-Overlap( $n_i, n_j$ )=true then
9:         Create new edge  $e_{ij}$  between  $n_i$  and  $n_j$ 
10:      end if
11:    end for
12:     $P \leftarrow P \cup n_i$ 
13:  else
14:     $P \leftarrow P - n_i$ 
15:  end if
16: end for

```

rectangles n_i and n_j were to be compared, then it implies n_i and n_j must be overlapped in X-coordinate. Hence, to determine whether two rectangles exist overlapping region only needs to check whether the segments of two rectangle are overlapped in Y-coordinate. If the x value is the right X-coordinate of a rectangle, the rectangle is removed from the red-black tree P . When the sweep line reaches the end of the array X , all the edges can be generated.

Fig. 6 demonstrates a simple example using Line Sweep Method. In Fig. 6(a), sweep line first enters rectangle A and rectangle A is stored in the red-black tree. In Fig. 6(b), sweep line enters rectangle D. Rectangle A is then compared with rectangle B to check whether if there exist overlapping region. In Fig. 6(c), sweep line leaves rectangle D which is removed from the red-black tree. Finally, in Fig. 6(d), the sweep line leaves rectangle E and rectangle E is removed.

In worst case scenario in which every rectangle overlaps with every other rectangles, the graph representation of such circumstance is a complete graph. The time complexity for Line Sweep Method in a complete graph is $O(N^2)$ since

no rectangles will be removed from red-black tree and every inserted rectangle must compare with every other rectangles stored in the red-black tree.

IV. FLIP-FLOP MERGING

After intersection graph is constructed, the edge with minimum cost will be selected and checked whether it is mergeable. To determine the cost between two nodes i and j , the mean point M_i and M_j to all connected pins for flip-flop i and j will be first calculated. The cost between node i and node j is the minimum distance between M_i and M_j . Let n be the total number of pins connecting to flip-flop i , Equation 1 describes the mean point for all pin p_i connected to flip-flop i . Equation 2 describes the edge cost connecting flip-flop i and flip-flop j .

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

$$Cost_{i,j} = \sqrt{(M_{i,x} - M_{j,x})^2 + (M_{i,y} - M_{j,y})^2} \quad (2)$$

For two flip-flops to be mergeable, there must exist corresponding type in given flip-flop library. For example, if FF-A is 1-bit, FF-B is 2-bit and there is no 3-bit flip-flop available, FF-A and FF-B can not be merged. In each iteration, each merge will select two flip-flops with the least edge cost. If two flip-flops are successfully merged, new node representing the merged flip-flop will be created and added to the graph with corresponding edges. Original flip-flops and edges connecting to two original flip-flops will be removed from the graph and new edges connecting to merged flip-flop will be added.

The algorithm will terminate until there is no edge left in the graph. In Fig. 7(a), node A and B are picked to be merged, edges connected to node A and node B are removed. In Fig. 7(b), new node M_{AB} is created with two new edges added to the graph. Fig. 7(c) illustrates the same concept of merging node M_{AB} and node M_{CD} to a 4-bit node M_{ABCD} . New edge is created between two nodes if and only if all the nodes including nodes included in merged node forms a clique in graph. Since there is no edge between M_{AB} and E , no new edge is created between M_{ABCD} and E . The algorithm terminates in Fig. 7(d) since there is no more edge in the graph.

A. Placement of Flip-Flop

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 merged flip-flop must not overlap with other merged flip-flops.

The given layout is partitioned into a set of bins with given placement 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 mean point is projected onto the boundary of the mergeable zone of merged flip-flop.

The projection point is selected as the relocation point. Before placing merged flip-flop at the relocation point, it will first check whether the relocation point is being occupied and whether the placement density constraint of the corresponding bin is violated. If violated, a breadth-first search will be conducted to search for a nearest unoccupied point in which

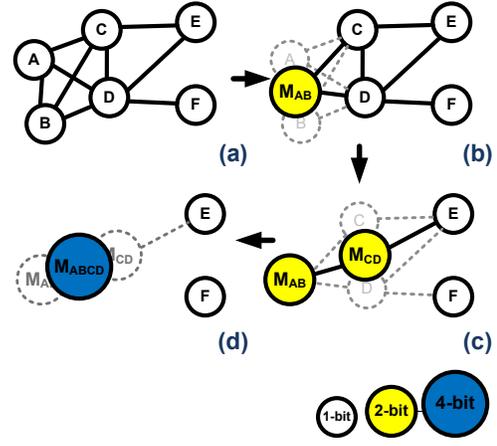


Fig. 7. Example of agglomerative clustering in flip-flop merging. (a) Intersection graph. (b) Node A and B are merged to a 2-bit node M_{AB} , new edges are created between node M_{AB} to C and M_{AB} to D (c). Node C and D are merged to a 2-bit node M_{CD} , new edge are created between M_{AB} and M_{CD} (d) Node M_{AB} and M_{CD} are merged to 4-bit node M_{ABCD} .

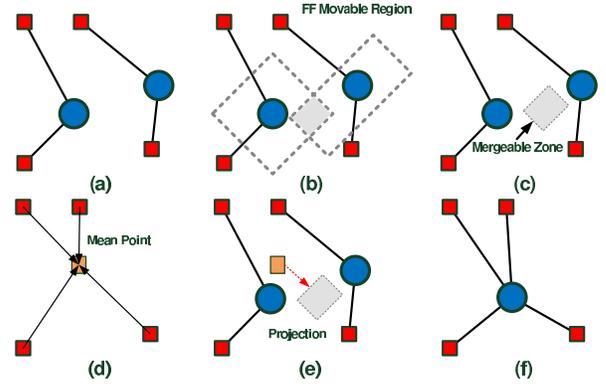


Fig. 8. Placement of Merged Flip-Flop. Blue circle denotes flip-flop and red square denotes pin. (a) Original Flip-Flop location. (b) Calculate Flip-Flops movable zone. (c) Mergeable zone to merge two flip-flops. (d) Calculate mean point for all pins connecting to two flip-flops. (e) Project mean point to mergeable zone. (f) Placement of merged flip-flop.

both slack and placement density constraint are not violated. If such point does not exist, the merged flip-flop will split and return to its original state.

V. EXPERIMENTAL RESULT

TABLE I
BIT, POWER AND AREA FOR EACH FLIP-FLOP TYPE

Name	Bit#	Power	Area	Power per bit	Area per bit
FF1	1	100	100	100	100
FF2	2	172	192	86	96
FF4	4	299	398	74.75	99.5

In this paper, to make fair comparison with [1], [12] and [9]. We obtained six testcases(C1~C6) and given flip-flop library from [1] which are also input benchmarks for [12] and [9]. Executables from [12], [9] and our algorithm are performed under Intel Xeon CPU 5160 Cent OS workstation running at 3.0 GHz. Executable from [1] is performed under Intel Core i3 CPU 550 Ubuntu workstation running at 3.2 GHz since it is compiled with g++ 4.5.4.

TABLE II
COMPARISON ON NUMBER OF FLIP-FLOP REDUCTION USING 1, 2 AND 4-BIT MULTI-BIT FLIP-FLOP

#	Original			[9]			[12]			[1]			Our		
	1/2/4-bit	Total	Norm.	1/2/4-bit	Total	Norm.	1/2/4-bit	Total	Norm.	1/2/4-bit	Total	Norm.	1/2/4-bit	Total	Norm.
C1	76,22.0	98	3.063	0.4,28	32	1.000	6,7.25	38	1.188	8,10.23	41	1.281	2,9.25	36	1.125
C2	366,57.0	423	3.440	0.6,117	123	1.000	16,30.101	147	1.195	24,36.96	156	1.268	8,36.100	144	1.171
C3	1464,228.0	1692	3.474	0.14,473	487	1.000	70,125,400	595	1.221	84,146,386	616	1.265	28,142,402	572	1.175
C4	4378,751.0	5128	3.460	2.21,1459	1482	1.000	232,402,1211	1845	1.245	242,469,1175	1886	1.273	80,450,1225	1755	1.179
C5	9150,1425.0	10575	3.504	2.33,2983	3018	1.000	484,806,2476	3766	1.248	480,920,2420	3820	1.266	160,880,2520	3560	1.180
C6	146400,22800.0	169200	3.520	18,113,47939	48070	1.000	7580,13508,39351	60439	1.257	7320,14780,38780	60880	1.266	2440,14020,40380	56840	1.182
Avg.	-	-	3.410	-	-	1.000	-	-	1.226	-	-	1.270	-	-	1.169

TABLE III
COMPARISON ON FLIP-FLOP'S SIGNAL WIRELENGTH AND EXECUTION TIME

Circuit #	[9]			[12]			[1]			Our		
	WL(nm)	WL(Norm.)	sec.	WL(nm)	WL(Norm.)	sec.	WL(nm)	WL(Norm.)	sec.	WL(nm)	WL(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	-	1.006	11.012	-	1.068	27.988	-	1.004	6.604

TABLE IV
COMPARISON ON AVERAGE DISPLACEMENT DISTANCE FROM ORIGINAL POSITION TO RELOCATED POSITION OF MERGED FLIP-FLOPS

Circuit Testcase	[9]		[12]		[1]		Our	
	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

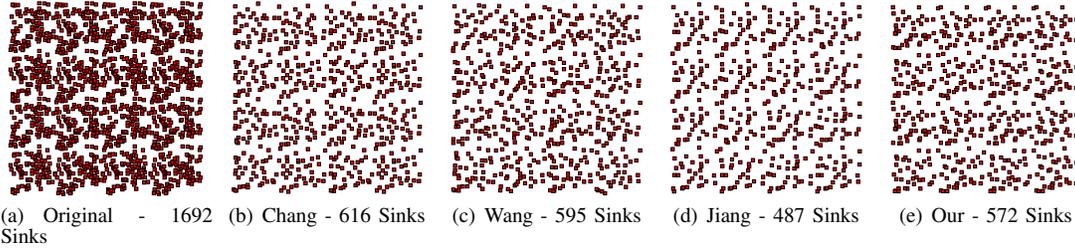


Fig. 9. Placement of flip-flop for testcase C3 after flip-flop merging. (a) Original (b)Chang et al. [1] (c) Wang et al. [12] (d) Jiang et al. [9] (e) Our Work.

An evaluator is implemented to evaluate the flip-flop's power reduction, increase in signal wirelength and displacement of flip-flop before and after flip-flop merging. The evaluator also examine whether if the given slack and placement density constraint is violated. The evaluated results for all of the obtained executables are verified with corresponding authors in [1][12][9].

The detail of the given multi-bit flip-flop library is described in Table I. Higher-bit flip-flop has the advantage of lower power consumption and lower area per bit.

A. Flip-Flop Number and Signal Wirelength Reduction

Table II presents the result of flip-flop number reduction. Compared with original benchmark without flip-flop merging, our algorithm can reduce flip-flop number by 65.7%. In comparison with [1] and [12], our proposed algorithm outperforms by 8.6% and 4.9% respectively. In comparison with [9], our algorithm trails behind by 16.9%. Fig. 9 illustrates distribution of sinks after flip-flop merging. Fig. 10 illustrates the flip-flop's signal path corresponding to Fig. 9.

In Table III, regarding to increase in signal wirelength connecting to each flip-flop, [9] increases most signal wirelength

in all 6 testcases. In contrast, although our algorithm trails behind [9] by 16.9% in flip-flop reduction but outperforms [9] by 16.2% in flip-flop's signal wirelength reduction. Additional increase in signal wirelength in [9] implies that flip-flop's power reduction will be counteracted by increase in dynamic power consumption in signal wirelength if switching activity factor is high. Regarding power saved due to flip-flop number reduction, power consumption of flip-flop can be calculated using Table I.

B. Average Displacement Distance for Merged Flip-Flop

In addition to analyzing flip-flop's power and signal wirelength, we also analyze average displacement of each merged flip-flop. Minimizing displacement of flip-flops will try to create least perturbation to original placement. Table IV presents the result of average displacement of merged flip-flops for all of the obtained executables. For each merged flip-flop, we can identify which of the original flip-flops it consists. Then displacement between flip-flop's original location and its final relocation position can then be calculated. In all 6 testcases, original flip-flop is given in 1-bit or 2-bit, we perform analysis on all of the testcases for each executable and present the result

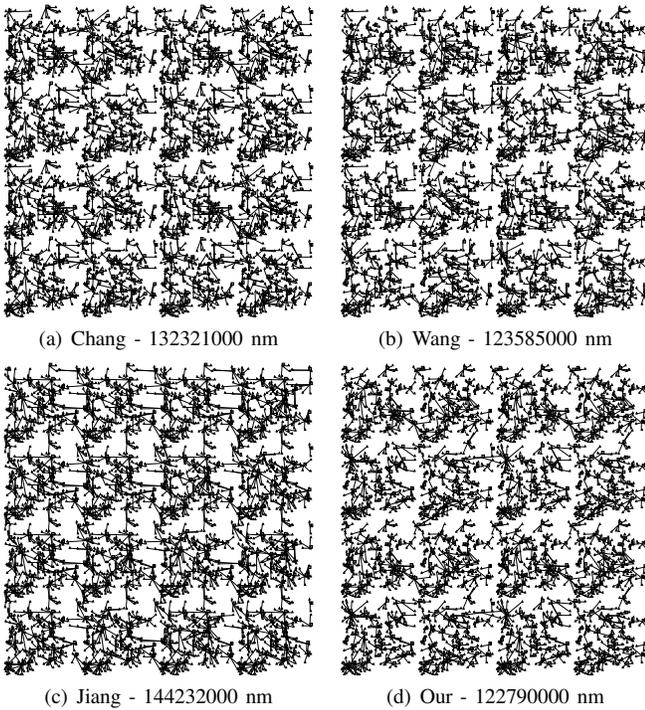


Fig. 10. Signal wirelength after flip-flop merging for testcase c3.(a) Chang et al. [1](b) Wang et al. [12] (c) Jiang et al. [9] (d) Our Work.

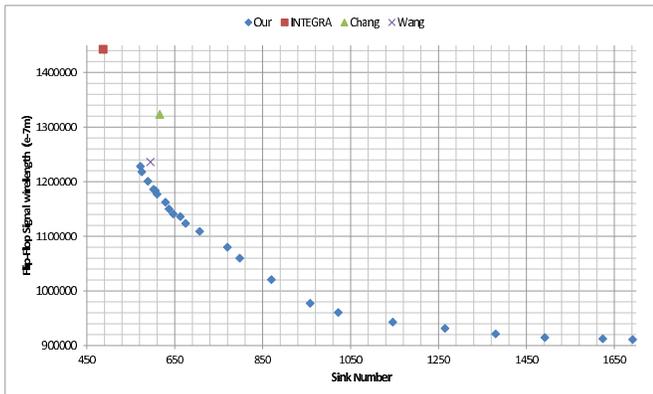


Fig. 11. Tradeoff curve of sink number and signal wirelength. By limiting maximum displacement of flip-flops, our proposed method can obtain a smooth tradeoff between signal wirelength reduction and flip-flop reduction.

by taking the average displacement for 1-bit and 2-bit flip-flop.

Based on observation, the benefit of merging nearest neighbor flip-flops is two-fold, it increases least signal wirelength and creates least perturbation to original layout. The nature of agglomerative clustering which orients to merge nearest flip-flop that will increase least signal net wirelength. In contrast, clique based approach [1][12] greedily merges flip-flop to maximize sink number reductions. Regarding Table IV, our algorithm has the least displacement with better flip-flop reduction comparing to [1] and [12]. In comparison with [9], since it has the most flip-flop reduction, it also creates most perturbation to original placement.

Agglomerative-based approach offers flexibility in optimizing between signal wirelength and flip-flop number reduction. An upper bounds on flip-flop displacement can be established to limit displacement of flip-flops. Fig. 11 is a tradeoff curve between signal wirelength and flip-flop number reduction by

altering the upper bound for flip-flop displacement for testcase c3, it can be observed that our proposed method obtains a near pareto front curve. Points on upper right of the curve [12][1] are obvious less optimal points. [9] is positioned at top left corner by achieving maximum flip-flop reduction, however, is less flexible when signal wirelength is primary concern.

VI. CONCLUSIONS

In this paper, we proposed an agglomerative based flip-flop merging algorithm. According to observation, agglomerative clustering can achieve a smooth tradeoff curve between sink number reduction and increase in signal wirelength. Our proposed algorithm outperforms all previous published flip-flop merging algorithms in terms of increase in total signal wirelength with least perturbation to original placement.

VII. ACKNOWLEDGMENT

The authors would like to thank Yao-Tsung Chang and Prof. Mark Po-Hung Lin From National Chung Chen University on providing binary executable for [1], Shao-Huan Wang and Prof. Wai-Kei Mak from National Tsing Hua University on providing binary executable for [12] and Chih-Long Chang and Prof. Iris Hui-Ru Jiang from National Chiao Tung University on providing binary executable for [9]. Many thanks to anonymous reviewers for their tremendous effort on identifying most detailed errors to improve this paper. Your comments are greatly appreciated.

REFERENCES

- [1] 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 *Proceedings of the International Conference on Computer Aided Design*, pages 218–223, 2010.
- [2] C. Chen, C. Kang, and M. Sarrafzadeh. Activity-sensitive clock tree construction for low power. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 279–282, 2002.
- [3] L. Chen, A. Hung, H.-M. Chen, E. Tsai, S.-H. Chen, M.-H. Ku, and C.-C. Chen. Using multi-bit flip-flop for clock power saving by designcompiler. In *Proceeding Synopsys User Group*, 2010.
- [4] Y. Cheon, P.-H. Ho, A. Kahng, S. Reda, and Q. Wang. Power-aware placement. In *Proceedings of the Design Automation Conference*, pages 795–800, June 2005.
- [5] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao. Bounded-skew clock and steiner routing. *Transaction Design Automation Electronic System*, 3:341–388, 1998.
- [6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
- [7] M. Donno, E. Macii, and L. Mazzone. Power-aware clock tree planning. In *Proceedings of the International Symposium on Physical Design*, pages 138–147, 2004.
- [8] W. Hou, D. Liu, and P.-H. Ho. Automatic register banking for low-power clock trees. In *Proceedings of the International Symposium on Quality of Electronic Design*, pages 647–652, 2009.
- [9] I. H.-R. Jiang, C.-L. Chang, Y.-M. Yang, E. Y.-W. Tsai, and L. S.-F. Chen. INTEGRA: fast multi-bit flip-flop clustering for clock power saving based on interval graphs. In *Proceedings of the International Symposium on Physical Design*, pages 115–122, 2011.
- [10] D.-J. Lee and I. L. Markov. Obstacle-aware clock-tree shaping during placement. In *Proceedings of the International Symposium on Physical Design*, pages 123–130, 2011.
- [11] R. S. Shelar. An efficient clustering algorithm for low power clock tree synthesis. In *Proceedings of the International Symposium on Physical Design*, pages 181–188, 2007.
- [12] S.-H. Wang, Y.-Y. Liang, T.-Y. Kuo, and W.-K. Mak. Power-driven flip-flop merging and relocation. In *Proceedings of the International Symposium on Physical Design*, pages 107–114, 2011.
- [13] Y. Wang, Q. Zhou, X. Hong, and Y. Cai. Clock-tree aware placement based on dynamic clock-tree building. In *Proceedings of the International Symposium on Circuits and Systems*, pages 2040–2043, May 2007.