International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:268
Vol.2, No.5, May 2013 DOI:10.15693/ijaist/2013.v2i5.5-12

Low Power High Speed Memory Architecture Using Multi Bit Flip Flops

G.Barathi[1] , S.Ramesh Kumar[2] Dr.R.Ganesan[3]

[1] PGScholar, Sethu Institute of technology, Madurai.
[2] Assistant Professor, Sethu Institute of technology , Madurai.
[3] Professor/HOD,

Abstract – **In modern VLSI designs, power consumed by clocking has taken a major part of the whole design especially for those designs using deeply scaled CMOS technologies. Hence in this paper we propose an algorithm for reducing the power consumption by replacing some flip-flops with fewer multi-bit flip-flops without affecting the performance of the original circuit. The flip-flop replacement will change the location of some flip-flops, leading to violation of timing and placement capacity constraints. To avoid this problem, several techniques are proposed. Utilizing the properties of manhattan distance and coordinate transformation, first we identify those flip-flops that can be merged and their legal regions. Next, a combination table is built to enumerate all possible combinations. Finally, the flip-flops are merged in hierarchial manner. According to the experimental results, our algorithm significantly reduces clock power by 20-30% and besides power reduction minimizing the total wirelength is also considered.**

KEY WORDS – **Clock power, multi-bit flip-flop manhattan distance, merging.**

I. INTRODUCTION

A clock system and a logic part consumes dominant part of the total chip power. The clock system itself consumes 20–45% of the chip power. In this clock system power, 90% is consumed by the flip-flops themselves and the last branches of the clock distribution network which directly drives the flip-flops [1]. This is due to the high switching activity.

$$P_{clk} = C_{clk} V^2_{dd} f_{clk} \qquad (1)$$

where P_{clk} is clock power, f_{clk} is the clock frequency, V_{dd} is the supply voltage, and C_{clk} is the switching capacitance including the gate capacitance of flip-flops (sequential elements) controlled by the clock signal, the interconnect capacitance of the clock network, and the capacitance associated with the buffers/inverters used in the clock network. Several methodologies [2], [3] have been proposed to reduce the power consumption of clocking. Given a design that the locations of the cells have been determined, the power consumed by clocking can be reduced further by replacing several flip-flops with multi-bit flip-flops. During clock tree synthesis, less number of flip-flops means less number of clock sinks. Thus, the resulting clock network would have smaller power consumption and uses less routing resource.

Applying MBFFs may have the following advantages:
1) smaller design area due to shared clock drivers and clock gating cells;
2) less delay and power of the clock network due to fewer clock sinks and smaller capacitive load on the clock net;
3) controllable clock skew because of common clock and enable signals for a group of flip-flops and reduced depth of a clock tree;
4) improved routing resource utilization especially when considering design for testability. The required routing resource for a scan chain is greatly reduced because of fewer cells in a scan chain.

Fig. 1 shows an example of merging two 1-bit flip-flops into one 2-bit flip-flop. If we replace the two 1-bit flip-flops as shown in Fig. 1(a) by the 2-bit flip-flop as shown in Fig. 1(b), the total power consumption can be reduced because the two 1-bit flip-flops can share the same clock buffer. However, the locations of some flip-flops would be changed after this replacement, and thus the wirelengths of nets connecting pins to a flip-flop are also changed. To avoid violating the timing constraints, we restrict that the wirelengths of nets connecting pins to a flip-flop cannot be longer than specified values after this process. Besides, to guarantee that a new flipflop can be placed within the desired region, we also need to consider the area capacity of the region.

A. *Related work*

Chen *et al.* [4] and Hou *et al.* [5] leverage on register banking at logic synthesis and at early physical synthesis, respectively. However, the subsequent timing and routing cost of the clustered result may somewhat deviate from what is expected at such early stages.

On the other hand, Yan and Chen [7], Chang *et al.* [8], and Wang *et al.* [9] postponed this task to postplacement to further consider the timing and even routing issues. Yan and Chen [7] analyzed the timing-safe region for each flipflop and then constructed an intersection graph to record the pairwise overlapping of these regions. They reduced MBFF clustering to minimum clique partitioning and solved it byiteratively merging flip-flops with fewest compatible flip-flops. However, they assumed the available bit numbers of the given MBFF library are contiguous and unlimited.

Considering a discrete and finite MBFF library, Chang *et al.* [6] proposed the problem of using multi-bit flip-flops to reduce power consumption in the post-placement stage. They use the graph-based approach to deal with this problem. In a graph, each node represents a flip-flop. If two flip-flops can be replaced by a new flip-flop without violating timing and capacity constraints, they build an edge between the corresponding nodes. After the graph is built, the problem of replacement of flip-flops can be solved by finding an *m*-clique in the graph. The flip-flops corresponding to the nodes in an *m*-clique can be replaced by an *m*-bit flipflop. They use the branch-and-bound and backtracking algorithm [8] to find all *m*-cliques in a graph. Because one node (flip-flop) may belong to

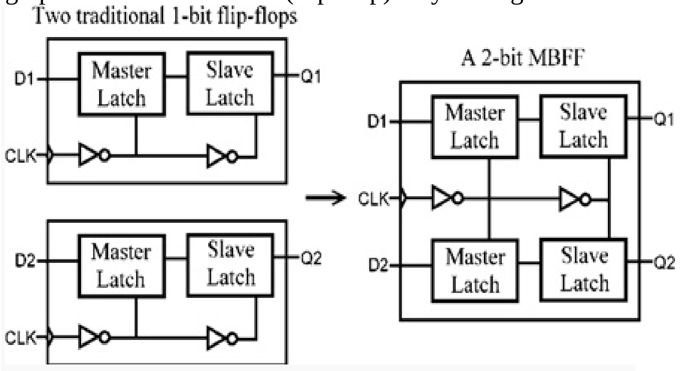


Fig. 1. Replacing two traditional FFs by a 2-bit MBFF

several *m*-cliques (*m*-bit flip-flop), they use greedy heuristic algorithm to find the maximum independent set of cliques, which every node only belongs to one clique, while finding *m*-cliques groups. However, if some nodes correspond to *k*-bit flip-flops that $k>=1$, the bit width summation of flip-flops corresponding to nodes in an *m*-clique, *j* , may not equal *m*. If the type of a *j* -bit flip-flop is not supported by the library, it may be time-wasting in finding impossible combinations of flip-flops.

II. OUR ALGORITHM

Our design flow can be roughly divided into three stages. Please see Fig. 5 for our flow. In the beginning, we have to identify a **legal placement region** for each flip-flop *fi* . First, the feasible placement region of a flip-flop associated with different pins are found based on the timing constraints defined on the pins. Then, the legal placement region of the flip-flop f_i can be obtained by the

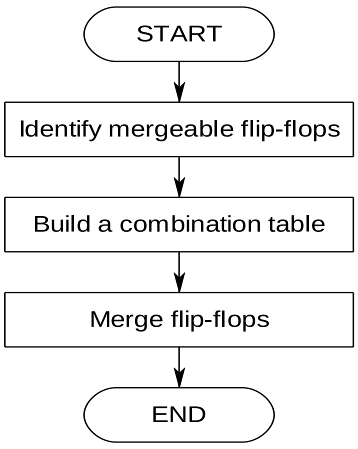
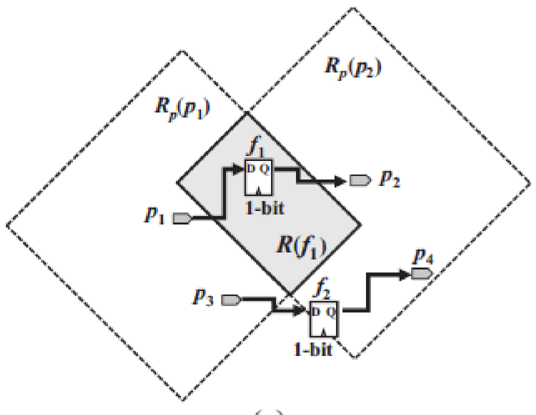


Fig. 2. Flow chart of our algorithm

overlapped area of these regions. However, because these regions are in the diamond shape, it is not easy to identify the overlapped area. Therefore, the overlapped area can be identified more easily if we can transform the coordinate system of cells to get rectangular regions. In the second stage, we would like to build a **combination table,** which defines all possible combinations of flip-flops in order to get a new multi-bit flip-flop provided by the library. The flip-flops can be merged with the help of the table. After the legal placement regions of flip-flops are found and the combination table is built, we can use them to **merge flip-flops**. To speed up our program, we will divide a chip into several bins and merge flip-flops in a local bin. However, the flip-flops in different bins may be mergeable. Thus, we have to combine several bins into a larger bin and repeat this step until no flip-flop can be merged anymore.

A. *Transformation of placement space*

The replacement of some flip-flops with multi-bit flip-flops would would change the routing length of the nets that connect to a flip-flop, it inevitably changes timing of some paths. To avoid that timing is affected after the replacement, the Manhattan distance between pin p_i and flip-flop $f_j$cannot be longer than the given constraint $S(p_i)$ defined on the pin p_i [i.e., $M(p_i , f_j) \leq S(p_i)$]. Since there may exist several pins connecting to ***f** i* , the legal placement region of ***f** i* are the overlapping area of several regions. As shown in Fig. 3(a), there are two pins *p*1 and *p*2 connecting to a flip-flop *f*1, and the feasible placement regions for the two pins are enclosed by dotted lines, which are denoted by *Rp(p*1*)* and *Rp(p*2*),* respectively. Thus, the legal placement region *R(f*1*)* for *f*1 is the overlapping part of these regions. In Fig. 3(b), *R(f*1*)* and *R(f*2*)* represent the legal placement regions of *f*1 and *f*2. Because *R(f*1*)* and *R(f*2*)* overlap, we can replace *f*1 and *f*2 by a new flip-flop *f*3 without violating the timing constraint, as shown in Fig. 3(c).



(a)

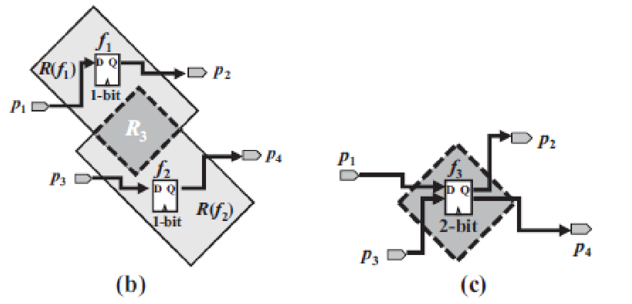


Fig. 3. (a) Feasible regions*Rp (p*1*)* and *Rp(p*2*)* for pins *p*1 and *p*2 which are enclosed by dotted lines, and the legal region *R(f*1*)* for *f*1 which is enclosed by solid lines. (b) Legal placement regions *R(f*1*)* and *R(f*2*)* for *f*1 and *f*2, and the feasible area *R*3 which is the overlap region of *R(f*1*)* and *R(f*2*)*. (c) New flip-flop *f*3 that can be used to replace *f*1 and *f*2 without violating timing constraints for all pins *p*1, *p*2, *p*3, and *p*4.

However, it is not easy to identify and record feasibleplacement regions if their shapes are diamond. Moreover, four coordinates are required to record an overlapping region [see Fig. 4(a)]. Thus, if we can rotate each segment 45°, the shapes of all regions would become rectangular, which makes identification of overlapping regions become very simple. For example, the legal placement region, enclosed by dotted lines in Fig. 4(a), can be identified more easily if we change its original coordinate system [see Fig. 4(b)]. In such condition, we only need two coordinates, which are the left-bottom corner and right-top corner of a rectangle, as shown in Fig. 4(b), to record the overlapped area instead of using four coordinates.

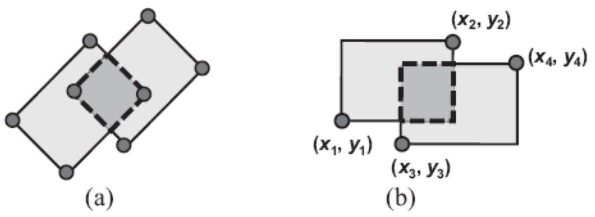


Fig. 4.(a) Overlapping region of two diamond shapes. (b) Rectangular shapes obtained by rotating the diamond shapes in (a) by 45°.

The equations used to transform coordinate system are shown in (1) and (2).

$$x' = \frac{x+y}{\sqrt{2}} \Rightarrow x'' = \sqrt{2}\cdot x' = x + y \qquad (2)$$

$$y' = \frac{-x+y}{\sqrt{2}} \Rightarrow y'' = \sqrt{2}\cdot y' = -x + y. \qquad (3)$$

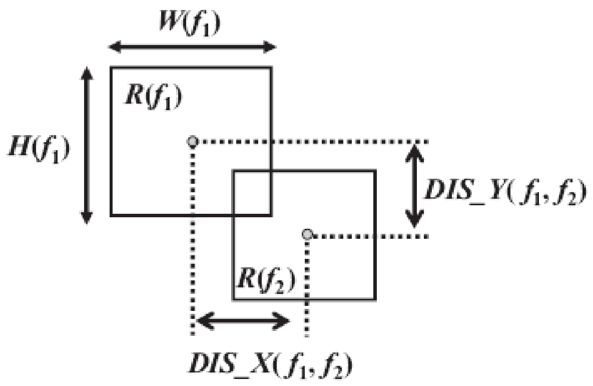


Fig. 5(a)Overlapping region of two diamond shapes.
(b)Rectangular shapes obtained by rotating the diamond shapes in (a) by 45°.

Then, we can find which flip-flops are mergeable according to whether their feasible regions overlap or not. Since the feasible placement region of each flip-flop can be easily identified after the coordinate transformation, we simply use (3) and (4) to determine whether two flip-flops overlap or not.

$$\text{DIS\_}X(f_1, f_2) < \frac{1}{2}(W(f_1) + W(f_2)) \qquad (4)$$

$$\text{DIS\_}Y(f_1, f_2) < \frac{1}{2}(H(f_1) + H(f_2)) \qquad (5)$$

where *W(f*1*)* and *H(f*1*)* [*W(f*2*)* and *H(f*2*)*] denote the width and height of *R(f*1*)* [*R(f*2*)*], respectively, in Fig. 8, and the function DIS_*X*(*f*1, *f*2*)* and (DIS_Y(*f*1, *f*2*))* calculates the distance between centers of *R(f*1*)* and *R(f*2*)* in *x*-direction (*y*-direction).

B. Build a Combination table

If we want to replace several flip-flops by a new flip-flop f_i'(note that the bit width of f_i' should equal to the summation of bit widths of these flip-flops), we have to make sure that the new flip-flop f_i' is provided by the library ***L*** when the feasible regions of these flip-flops overlap**.** Now a combination table is to be built, which records all possible combinations of flip-flops to get feasible flip-flops before replacements. Thus, we can gradually replace flip-flops according to the order of the combinations of flip-flops in this table.

Library L	
Type 1	Type 2
1 bit	4 bit

Combinational table	
n1	n2
1 bit	4 bit

Fig.6(a)Initialize the library *L* and the combination table *T* .

Combination Table T				
n1 1 bit	n2 4 bit	n3 2 bit n1 + n1	n4 3 bit n1 + n3	n5 4 bit n3 + n3

Fig.6(b) Pseudo types are added into *L*, and the corresponding binary tree is also build for each combination in *T*.

For example, consider a library ***L*** that provides two types of flip-flops, whose bit widths are 1 and 4 (i.e., b_{min} = 1 and b_{max} = 4), in Fig. 6(a). We first initialize two combinations n_1 and n_2 to represent these two types of flip-flops in the table ***T*** [see Fig. 6(a)]. Next, the function **InsertPseudoType** is performed to check whether the flip-flop types with bit widths between 1 and 4 exist or not. Thus, two kinds of flip-flop types whose bit widths are 2 and 3 are added into ***L***, and all types of flip-flops in ***L*** are sorted according to their bit widths [see Fig. 6(b)]. Now, for each combination in ***T***, we would build a binary tree with 0-level, and the root of the binary tree denotes he combination. Next, we try to build new legal combinations according to the present combinations. By combing two1-bit flip-flops in the first combination,a new

Combination Table T		
n1 1 bit	n2 4 bit	n3 2 bit n1 + n1

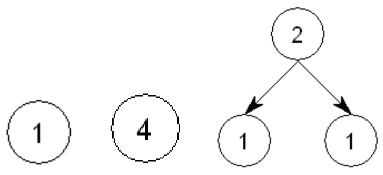


Fig. 6(c) New combination *n*3 is obtained from combining two *n*1s.

Combination n_3 can be obtained [see Fig. 6(c)]. Similarly, we can get a new combination n_4 (n_5) by combining n_1

Library L			
Type 1 1 bit	Type 2 2 bit (pseudo)	Type 3 3 bit (pseudo)	Type 4 4 bit

Combination Table	
n1 1 bit	n2 4 bit

and n_3(two n_3's) [see Fig. 6(d)].
Finally, n_6 is obtained by combing n_1 and n_4.

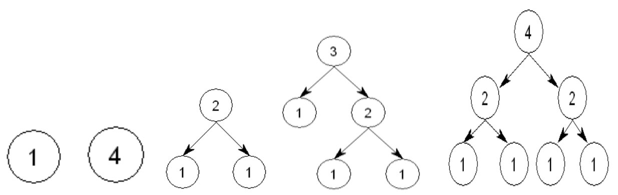


Fig. 6(d) New combination *n*4 is obtained from combining *n*1 and *n*3, and the combination *n*5 is obtained from combining two *n*3s.

All possible combinations of flip-flops are shown in Fig.6(e). Among these combinations, *n*5 and *n*6 are duplicated since they both represent the same condition, which replaces four 1-bit flip-flops by a 4-bit flip-flop. To speed up the process, *n*6 is deleted from ***T*** rather than *n*5 because its height is larger. After this procedure, *n*4 becomes an unused combination [see Fig. 6(e)] since the root of binary tree of *n*4 corresponds to the pseudo type, *type*3, in ***L*** and it is only included in *n*6. After deleting *n*6, *n*4 is also need to be deleted. The last combination table ***T*** is shown in Fig. 6(f).

Combination Table T					
n1 1 bit	n2 4 bit	n3 (2 bit) n1 + n1	n4 (3 bit) n1 + n3	n5 (4 bit) n3 + n3	n6 (4 bit) n1 + n4

Fig. 6(e) New combination *n*6 is obtained from combining *n*1 and *n*4.

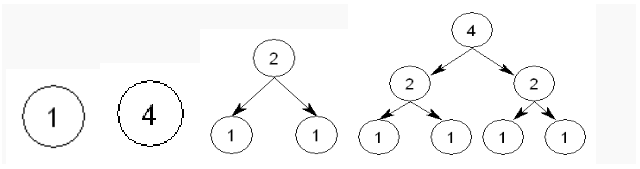


Fig.6(f) Last combination table is obtained after deleting the unused combination in (e).

C. *Merge Flip-Flops.*

After the combination table is built the combination of flip-flops are used for merging and replacing. To reduce the complexity the whole placement region is divided into several sub regions. Then, several subregions are combined into a larger subregion and the flip-flops are replaced again so that those flip-flops in the neighboring subregions can be replaced further. Finally, those flip-flops with pseudo types are deleted in the last stage because they are not provided by the supported library.

1) Region Partition: To speed up our problem, we divide the whole chip into several subregions. By suitable partition, the computation complexity of merging flip-flops can be reduced significantly. As shown in Fig. 11, we divide the region into several subregions, and each subregion contains six bins, where a bin is the smallest unit of a subregion.

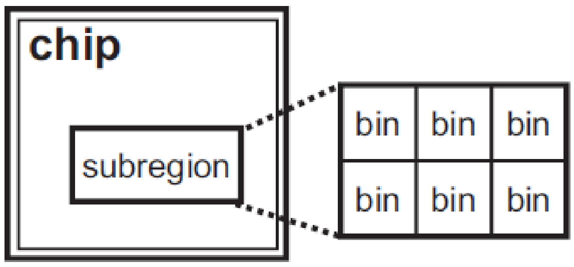


Fig. 7 Example of region partition with six bins in one subregion.

2) Replacement of Flip-flops in Each Subregion: Before illustrating the procedure to merge flip-flops, first an equation is given to measure the quality if two flip-flops are going to be replaced by a new flip-flop as follows:

$$\text{cost} = \text{routing\_length} - \alpha \times \sqrt{(\text{available\_area})} \quad (6)$$

where routing_length denotes the total routing length between the new flip-flop and the pins connected to it, and available_ area represents the available area in the feasible region for placing the new flip-flop. α is a weighting factor.The cost function includes the term routing_length to favor a replacement that induces shorter wirelength. Besides, if the region has larger available space to place a new flip-flop, it implies that it has higher opportunities to combine with other flip-flops in the future and more power reduction. Thus, we will give it a smaller cost. Once the flip-flops cannot be merged to a higher-bit type we ignore the *available_area* in the cost function, and hence α is set to 0. First the flip-flops are linked below the combinations corresponding totheir types in the library. Then, for each combination ***n*** in ***T***, we serially merge the flip-flops linked below the left child and the right child of ***n*** from leaves to root. Based on the binary tree, we can find the combinations associated with the left child and right child of the root. Based on the binary tree, we can find the combinations associated with the left child and right child of the root. Hence, the flip-flops in the lists named *l*left and *l*right, linked below the combinations of its left child and its right child are checked. Then, for each flip-flop ***f i*** in l_{left}, the best flip-flop f_{best} in l_{right}, which is the flip-flop that can be merged with ***f i*** with the smallest cost recorded in C_{best}, is picked. For each pair of flip-flops in the respective list, the combination cost is computed if they can be merged and the pair with the smallest cost is chosen. Finally, we add a new flip-flop ***f '*** in the list of the combination *n* and remove the picked flip-flops which constitutes the ***f '.***

For example, given a library containing three types of flipflops (1-, 2-, and 4-bit), we first build a combination table ***T*** as shown in Fig. 8(a). In the beginning, the flip-flops with various types are, respectively, linked below *n*1, *n*2, and *n*3 in ***T*** according to their types. Suppose we want to form a flipflop in *n*4, which needs two 1-bit flip-flops according to the combination table. Each pair of flip-flops in *n*1 are

Combination Table T			
n1	n2	n3	n4
1 bit	4 bit	2 bit	4 bit
		n1	n3
		+	+
		n1	n3

selected and checked to see if they can be combined If there are several possible choices, the pair with the smallest cost value is chosen to break the tie. In Fig. 8(a), f1 and f2 are chosen because their combination gains the smallest cost. Thus, we add a new node f_3 in the list below n_4, and then delete f_1 and f_2 from their original list [see Fig. 8(b)].

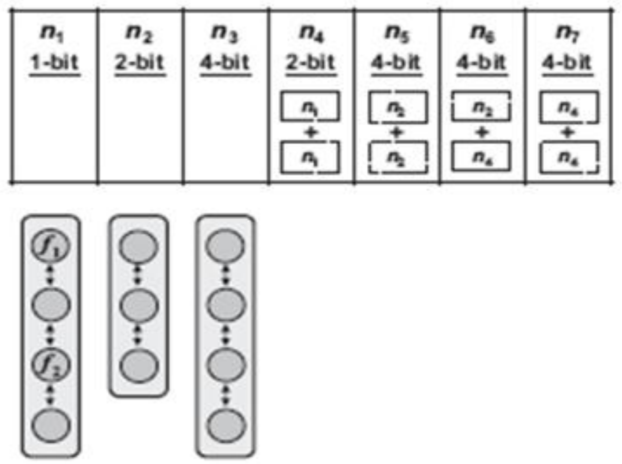

n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_1
f_2

Fig. 8(a) Sets of flip-flops before merging.

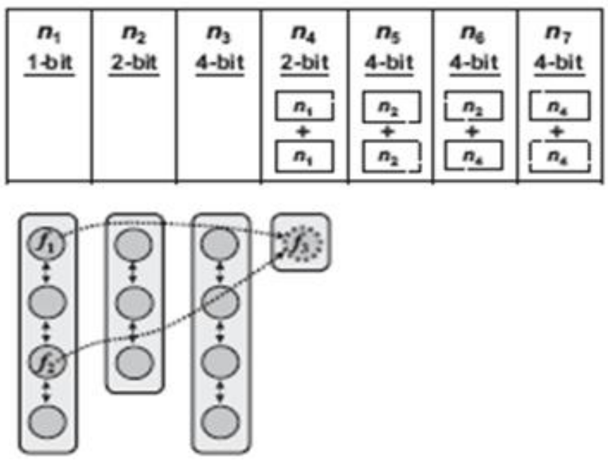

n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_1
f_2
f_3

Fig. 8(b) Two 1-bit flip-flops, *f*1 and *f*2, are replaced by the 2-bit flip-flop *f*3.

Similarly, *f*4 and *f*5 are combined to obtain a new flip-flop *f*6, and the result is shown in Fig. 8(c). After all flip-flops in the combinations of 1-level trees (*n*4 and *n*5*)* are obtained as shown in Fig. 8(d), we start to form the flip-flops in the combinations of 2-level trees (*n*6, and *n*7*)*.

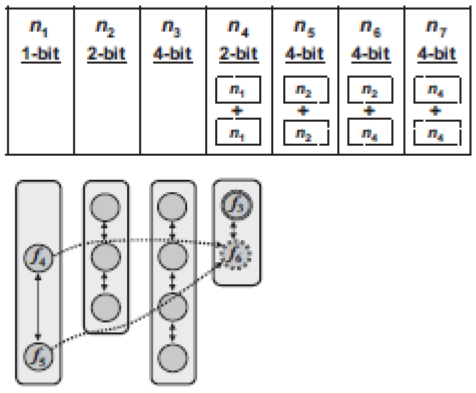

n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_3
f_4
f_5
f_6

Fig. 8(c) Two 1-bit flip-flops, *f*4 and *f*5, are replaced by the 2-bit flip-flop *f*6.

In Fig. 8(e), there exist some flip-flops in the lists below *n*2 and *n*4, and we will merge them to get flip-flops in *n*6 and *n*7, respectively. Suppose there is no overlap region between the couple of flipflops in *n*2 and *n*4. It fails to form a 4-bit flip-flop in *n*6. Since the 2-bit flip-flops *f*3 and *f*6 are mergeable, we can combine them to obtain a 4-bit flip-flop *f*10 in *n*7. Finally, because there exists no couple of flip-flops that can be combined further, the procedure finishes as shown in Fig. 8(f).

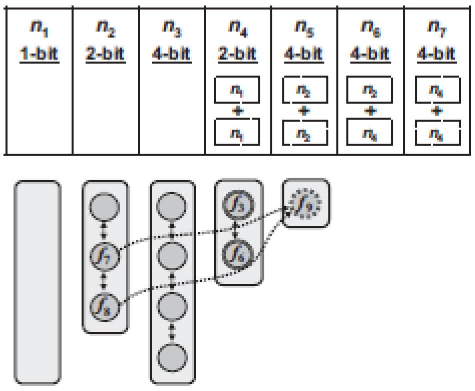

n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_7
f_8
f_3
f_6
f_9

Fig. 8(d) Two 2-bit flip-flops, *f*7 and *f*8, are replaced by the 4-bit flip-flop *f*9.

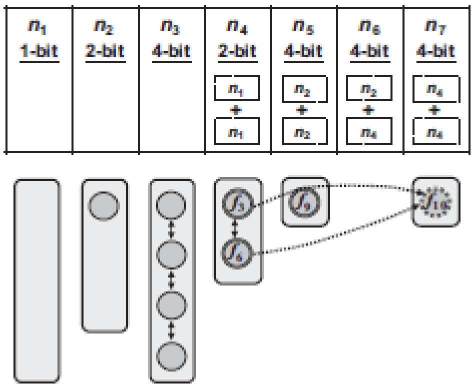

n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_3
f_6
f_9
f_{10}

Fig.8(e) Two 2-bit flip-flops, *f*3 and *f*6, are replaced by the 4-bit flip-flop *f*10.

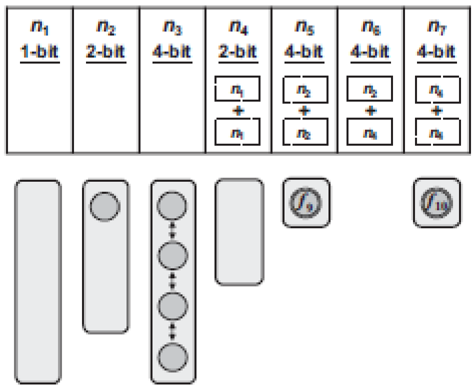

n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_9
f_{10}

Fig.8(f) Sets of flip-flops after merging.

If the available overlap region of two flip-flops exists, we can assign a new one to replace those flip-flops. Once there is sufficient space to place the new flip-flop in the available region, the algorithm will perform the replacement, and the new generated flip-flop will be placed in the grid that makes the wirelength between the flip-flop and its connected pins smallest. If the capacity constraint of the bin, *Bk*, which the grid belongs to will be violated after the new flip-flop is placed on that grid, we will search the bins near *Bk* to find a new available grid for the new flip-flop. If none of bins which are overlapped with the available region of new flip-flop can satisfy the capacity constraint after the placement of new flip-flop, the program will stop the replacement of the two flip-flops.

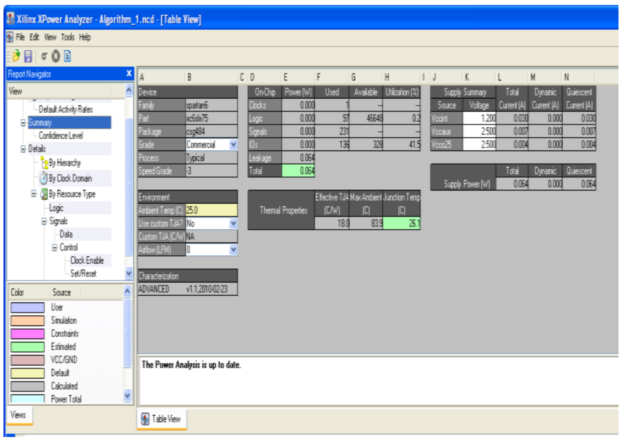
III. RESULTS AND DISCUSSIONS

In this algorithm, there exist two values which would affect our results: the first one is the dimension of a subregion since we would partition a chip into several subregions. The second one is the parameter used in the cost function of (6). Thus, we first do some experiments to explore better values for these two parameters.

1) Influence of Region Size on Performance: In this part, we first determine a suitable size for each subregion during partitioning. Since the execution time is actually dominated by the average number of flip-flops included in a subregion, we use the number of flip-flops in a single subregion to represent the size of a subregion, which can be obtained from multiplying the number of bins in a subregion by the average number of flip-flops in a bin. We sweep the number of flip-flops included in a subregion to observe its effect on power consumption and execution. While a subregion gets larger, the execution time becomes longer. However, the power consumption does not decrease proportionally. On the contrary, if the subregion size becomes very small, the power consumption will increase significantly.
To balance execution time and power consumption, we select 600 as the number of flip-flops in a single subregion (the normalized power and execution time are about 83% and 0.8% if the number of flip-flops in a single subregion is 600.

2) Influence of Weighting Factor α on Performance: Since the parameter α used by (6) would affect our results, it is necessary to find a suitable value for getting better results. In this experiment, we sweep α from 0 to 3 to get the data of power consumption and wirelength. While the value of α becomes larger, the power reduction ratio gets larger.

IV. SIMULATION RESULTS AND DISCUSSIONS



Device	
Family	spartan6
Part	xc6slx75
Package	csg484
Grade	Commercial
Process	Typical
Speed Grade	-3

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	1	---	---
Logic	0.000	97	46648	0.2
Signals	0.000	231	---	---
IOs	0.000	136	328	41.5
Leakage	0.064			
Total	0.064			

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.030	0.000	0.030
Vccaux	2.500	0.007	0.000	0.007
Vcco25	2.500	0.004	0.000	0.004

	Total	Dynamic	Quiescent
Supply Power (W)	0.064	0.000	0.064

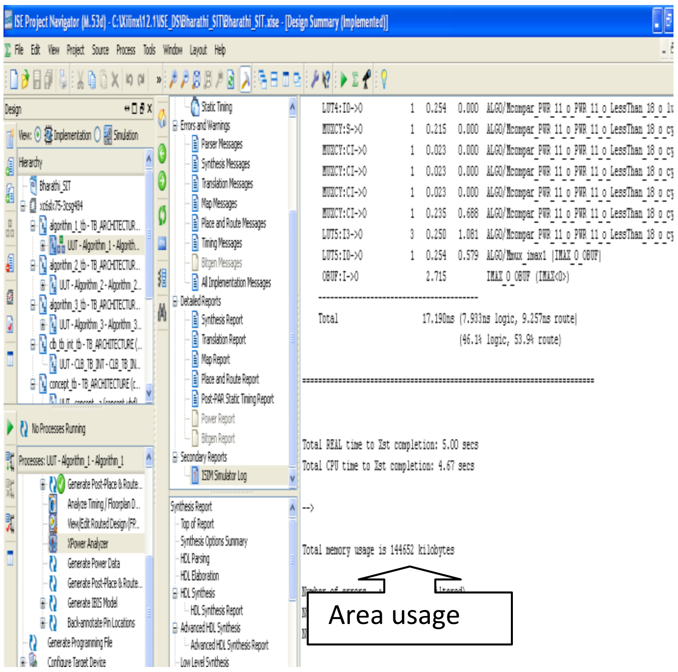
Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
Thermal Properties	18.0	83.9	26.1

Characterization	
ADVANCED	v1.1,2010-02-23

The Power Analysis is up to date.

Fig. 9. Power consumed by the algorithm.



```
LUT4:I0->O        1   0.254   0.000  ALGO/Mcompar_PWR_11_o_PWR_11_o_LessThan_18_o_lu
MUXCY:S->O        1   0.215   0.000  ALGO/Mcompar_PWR_11_o_PWR_11_o_LessThan_18_o_cy
MUXCY:CI->O       1   0.023   0.000  ALGO/Mcompar_PWR_11_o_PWR_11_o_LessThan_18_o_cy
MUXCY:CI->O       1   0.023   0.000  ALGO/Mcompar_PWR_11_o_PWR_11_o_LessThan_18_o_cy
MUXCY:CI->O       1   0.023   0.000  ALGO/Mcompar_PWR_11_o_PWR_11_o_LessThan_18_o_cy
MUXCY:CI->O       1   0.235   0.688  ALGO/Mcompar_PWR_11_o_PWR_11_o_LessThan_18_o_cy
LUT5:I3->O        3   0.250   1.081  ALGO/Mcompar_PWR_11_o_PWR_11_o_LessThan_18_o_cy
LUT5:I0->O        1   0.254   0.579  ALGO/Mmux_imax1 (IMAX_0_OBUF)
OBUF:I->O             2.715          IMAX_0_OBUF (IMAX<0>)
   ----------------------------------------
   Total                  17.190ns (7.933ns logic, 9.257ns route)
                                   (46.1% logic, 53.9% route)

=========================================================================

Total REAL time to Xst completion: 5.00 secs
Total CPU time to Xst completion: 4.67 secs

-->

Total memory usage is 144652 kilobytes
```

Fig. 10 Area Usage of the algorithm

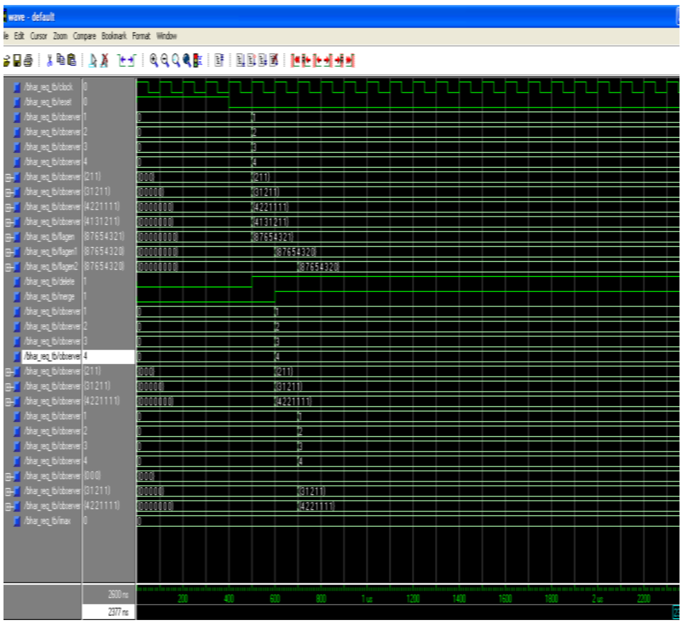


Fig. 10. Simulation result of the algorithm

V. CONCLUSION

This paper has proposed an algorithm for flip-flop replacement for power reduction in digital integrated circuit design. The procedure of flip-flop replacements is depending on the combination table, which records the relationships among the flip-flop types. The concept of pseudo type is introduced to help to enumerate all possible combinations in the combination table. By the guidelines of replacements from the combination table, the impossible combinations of flip-flops will not be considered that decreases execution time. The experimental results show that our algorithm can achieve a balance between power reduction and wirelength reduction. Besides power reduction, the objective of minimizing the total wirelength is also considered.

References

[1] L.-T. Wang, Y.-W. Chang, and K.-T. Cheng, Eds., *Electronic Design Automation: Synthesis, Verification, and Test*. Burlington, MA: Elsevier/ Morgan Kaufmann, 2009.

[2] D. Duarte, V. Narayanan, and M. J. Irwin, “Impact of technology scaling in the clock power,” in *Proc. IEEE VLSI Comput. Soc. Annu. Symp.,* Pittsburgh, PA, Apr. 2002, pp. 52–57.

[3] P. Gronowski, W. J. Bowhill, R. P. Preston, M. K. Gowan, and R. L. Allmon, “High-performance microprocessor design,” *IEEE J. Solid-State Circuits, vol. 33, no. 5, pp. 676–686, May 1998.*

[4] L. Chen, A. Hung, H.-M. Chen, E. Y.-W. Tsai, S.-H. Chen, M.-H. Ku, and C.-C. Chen, “Using multi-bit flip-flop for clock power saving by DesignCompiler,” in *Proc. Synopsys User Group (SNUG)*, 2010 [Online].Available:http://www.synopsys.com.cn/information/snug/2010/ using-multi-bit-flip-flop-for-clock-power-saving-by-designcompiler.

[5] W. Hou, D. Liu, and P.-H. Ho, “Automatic register banking for lowpower clock trees,” in *Proc. ISQED*, 2009, pp. 647–652.

[6] Y.-T. Chang, C.-C. Hsu, P.-H. Lin, Y.-W. Tsai, and S.-F. Chen, “Post-placement power optimization with multi-bit flip-flops,” in *Proc. EEE/ACM Comput.-Aided Design Int. Conf.*, San Jose, CA, Nov. 2010, pp. 218–223.

[7] J.-T. Yan and Z.-W. Chen, “Construction of constrained multi-bit flipflops for clock power reduction,” in *Proc. ICGCS*, 2010, pp. 675–678.

[8] Y.-T. Chang, C.-C. Hsu, M. P.-H. Lin, Y.-W. Tsai, and S.-F. Chen, “Postplacement power optimization with multi-bit flip-flops,” in *Proc. ICCAD*, 2010, pp. 218–223.

[9] S.-H. Wang, Y.-Y. Liang, T.-Y. Kuo, and W.-K. Mak, “Power-driven flip-flop merging and relocation,” in *Proc. ISPD*, 2011, pp. 107–114.