Alternative Approach of Low Power Multi Bit Flip-Flops in Integrated Circuits

M.Santhanaraj[1], K.Siddharthraju[2] , R.Dhivya Devi[3], S.Prabahar[4]

Assistant Professor-Department of Electronics and communication Engineering,
[1,2,3]KPR Institute of Engineering and Technology, Coimbatore.
[4]Chandy College of Engineering, Tuticorin

***Abstract* – In modern VLSI designs, power consumed by clocking is one of the major issue. Hence, in this paper 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 leading to violation of timing and placement capacity constraints. Some techniques are proposed to avoid this problem. Manhattan distance and co-ordinate transformation used to identify those flip-flops that can be merged and their legal regions. Then, a combination table is built to enumerate all possible combinations. Finally, the flip-flops are merged in hierarchical manner. According to the experimental results, our algorithm significantly reduces clock power by 20-30% and besides power reduction minimizing the total wire length 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 [10]. This is due to the high switching activity.

$$P_{clk} = C_{clk} V^2_{dd} f_{clk}$$

Where, *Pclk* is clock power, *fclk* is the clock frequency, *Vdd* is the supply voltage, and *Cclk* is the switching capacitance including the gate capacitance of flip-flops 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. Reducing the power consumption not only can enhance battery life but also can avoid the overheating problem, which would increase the difficulty of packaging or cooling [1], [2]. 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 [3]. Several methodologies [4], [5] have been proposed to reduce the power consumption of clocking. During clock tree synthesis, less number of flip-flops means less number of clock sinks.

Besides, once more smaller flip-flops are replaced by larger multi-bit flip-flops, device variations in the corresponding circuit can be effectively reduced. As CMOS technology progresses, the driving capability of an inverter-based clock buffer increases significantly. The driving capability of a clock buffer can be evaluated by the number of minimum-sized inverters that it can drive on a given rising or falling time

A. Existing System

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

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 flip-flop. 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 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. In the beginning, we have to identify a legal placement region for each flip-flop. First, the feasible placement regions 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 can be obtained by the overlapped area of these regions.

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.

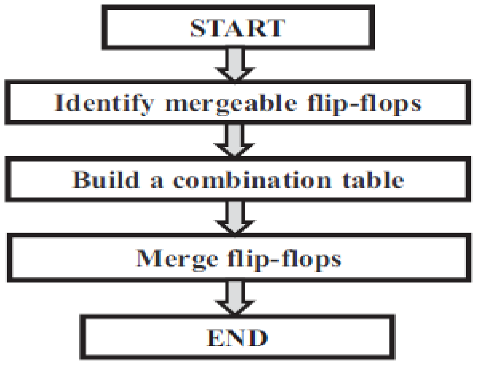


Fig. 1 Algorithm of Flow chart

After the legal placement regions of flip-flops are found and the combination table is built, we can use the third stage is 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 merge-able. Thus, we have to combine several bins into a larger bin and repeat this step until no flip-flop can be merged anymore.

A. Identify Mergable Flip-Flops

Identify the mergable flip-flops can be estimated by the shape of a feasible placement region in the IC associated with one pin $\boldsymbol{p}_i$ connecting to a flip-flop $\boldsymbol{f}_i$ would be diamond region. Since there may exist several pins connecting to $\boldsymbol{f_i}$**,** the legal placement region of $\boldsymbol{f_i}$ are the overlapping area of several regions. The replacement of some flip-flops with multi-bit flip-flops would change the routing length of the nets that connect to a flip-flop, it inevitably changes timing of some paths.

Manhattan distance

To avoid that timing is affected after the replacement, the Manhattan distance between pin $\boldsymbol{p}_i$ and flip-flop $\boldsymbol{f}_j$ cannot be longer than the given constraint $S(pi\,)$ defined on the pin pi [i.e., $\boldsymbol{M}(\boldsymbol{p}_i\,,\,\boldsymbol{f}_j\,) \le \boldsymbol{S}(\boldsymbol{p}_i\,)$]. Since there may exist several pins connecting to $\boldsymbol{f}_i$ **,** the legal placement region of $\boldsymbol{f}_i$ are the overlapping area of several regions.

The distance between two points in a grid based on a strictly horizontal and/or vertical path, as opposed to the diagonal or distance. The Manhattan distance is the simple sum of the horizontal and vertical components, whereas the diagonal distance might be computed by applying the Pythagorean theorem

Then, we can find which flip-flops are merge-able according to whether their feasible regions overlap or not. In Fig. 2, 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).

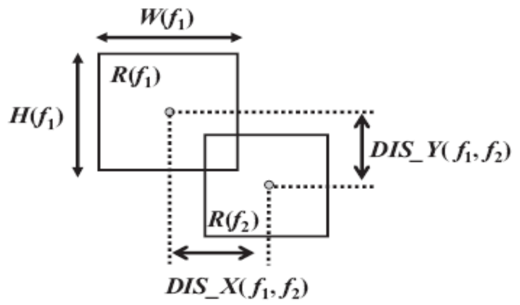


Fig. 2 Overlapping region of two diamond shapes. (b) Rectangular shapes obtained by rotating the diamond shapes in (a) by 45°.

Since the feasible placement region of each flip-flop can be easily identified after the coordinate transformation, we simply use (1) and (2) to determine whether two flip-flops overlap or not.

$$\text{DIS\_X}(f_1,f_2) < \tfrac{1}{2}\,(W(f_1) + W(f_2)) \;\ldots\ldots\ldots\ldots\; (1)$$

$$\text{DIS\_Y}(f_1,f_2) < \tfrac{1}{2}\,(H(f_1) + H(f_2)) \;\ldots\ldots\ldots\ldots\; (2)$$

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.

B. Build A Combinational Table

If we want to replace several flip-flops by a new flip-flop f_i', we have to make sure that the new flip-flop fi' is provided by the library $\boldsymbol{L}$ 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.

Step1: Initialize the Library and the Combination Table

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. 3(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. 3(a)].

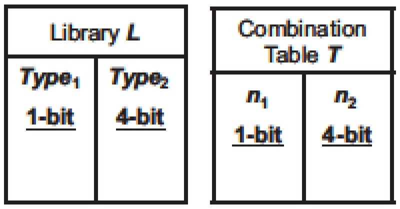


Fig. 3(a) Initialize the library *L* and the combination table *T*.

Step2: Insert Pseudo Type

Next, the function Insert Pseudo Type 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. 3(b)].

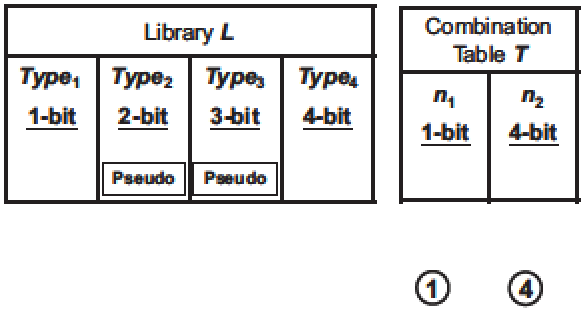


Fig. 3(b) Pseudo types are added into *L*, and the corresponding binary tree is also build for each combination in *T*.

Step3: create New Combination n3

By combining two1-bit flip-flops in the first and combination of two n1 and make a new Combination *n*3 can be obtained [see Fig. 3(c)].

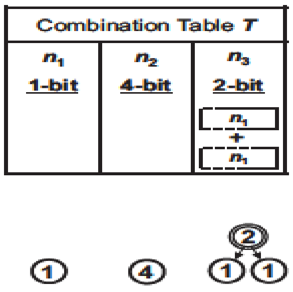


Fig. 3(c) New combination *n*3 is obtained from combining two *n*1s

Step4: create New Combination *n*4 and *n*5

Similarly, we can get a new combination *n*4 (*n*5) by combining *n*1 and *n*3(two *n*3's) [see Fig. 3.3 (d)]. Finally, *n*6 is obtained by combing *n*1 and *n*4.

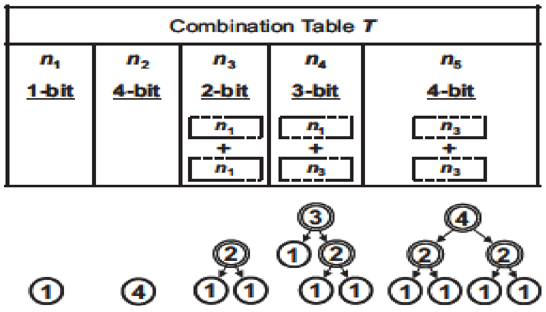


Fig. 3(d) New combination *n*4 is obtained from combining *n*1 and *n*3, and the combination *n*5 is obtained from combining two *n*3s.

Step5: create New Combination *n6*

All possible combinations of flip-flops are shown in Fig 3(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. 3(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.

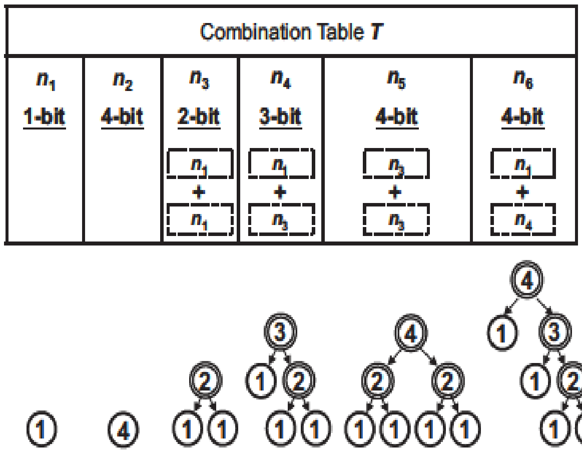
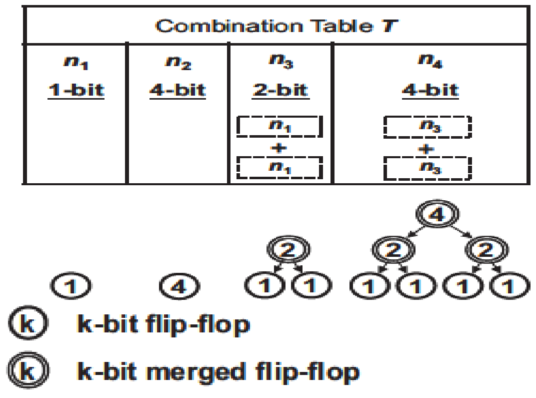


Fig. 3(e) New combination *n*6 is obtained from combining *n*1 and *n*4.

Step6: Deleting Unused Combination

After deleting *n*6, *n*4 is also need to be deleted. The last combination table ***T*** is shown in Fig. 3.3 (f).



Combination Table T			
n_1 1-bit	n_2 4-bit	n_3 2-bit	n_4 4-bit
		n_1 + n_1	n_3 + n_3

Fig. 3(f) Last combination table is obtained after deleting the unused combination in (e).

C. Merge Flip-Flops

In this module, more number of flip-flops can be merged into multi bit flip-flops. Fig. 2.5 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. 4(a) by the 2-bit flip-flop as shown in Fig. 4(b), the total power consumption can be reduced because the two 1-bit flip-flops can share the same clock buffer.

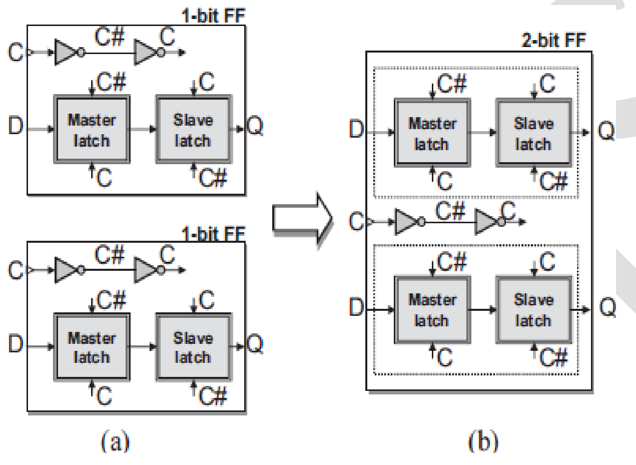


Fig 4(a) Two 1-bit flip-flops (before merging), 4(b) 2-bit flip-flop (after merging).

However, the locations of some flip-flops would be changed after this replacement, and thus the wire lengths of nets connecting pins to a flip-flop are also changed.

To avoid violating the timing constraints, we restrict that the wire lengths of nets connecting pins to a flip-flop cannot be longer than specified values after this process. Besides, to guarantee that a new flip-flop can be placed within the desired region, we also need to consider the area capacity of the region.

Region Partition

To speed up our problem, we divide the whole chip into several sub regions. By suitable partition, the computation complexity of merging flip-flops can be reduced significantly. As shown in Fig. 5, we divide the region into several sub regions, and each sub region contains six bins, where a bin is the smallest unit of a sub region.

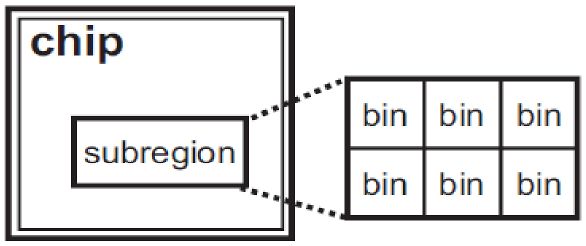


Fig. 5 Region partition with six bins in one sub region.

Replacement of Flip-Flops In Each Sub-region

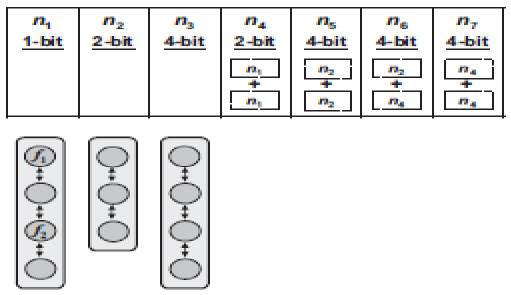
Before illustrating the procedure to merge flip-flops, first an equation is given to measure the quality if two flip-flops are going to be replaced by a new flip-flop as follows:

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

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 wire length. 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.

<u>Step1: Sets of Flip-Flops Before Merging</u>

A library containing three types of flip-flops (1-, 2-, and 4-bit), we first build a combination table ***T*** as shown in Fig. 6(a).

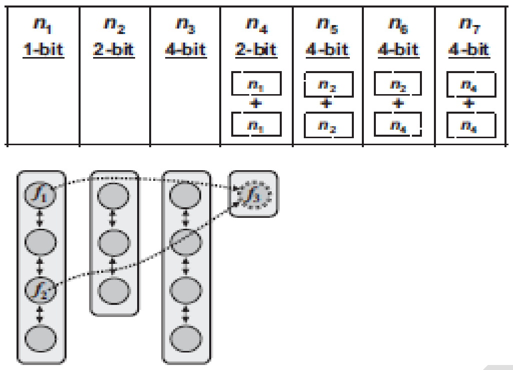


n_1 1-bit	n_2 2-bit	n_3 4-bit	n_4 2-bit	n_5 4-bit	n_6 4-bit	n_7 4-bit
			n_1 + n_1	n_2 + n_2	n_2 + n_4	n_4 + n_4

Fig 6(a) Sets of flip-flops before merging

Step2: Replacing 2-Bit Flip-Flop *f*3.

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 flip-flop in *n*4, which needs two 1-bit flip-flops according to the combination table. Each pair of flip-flops in *n*1 are 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. 5(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. 6(b)].

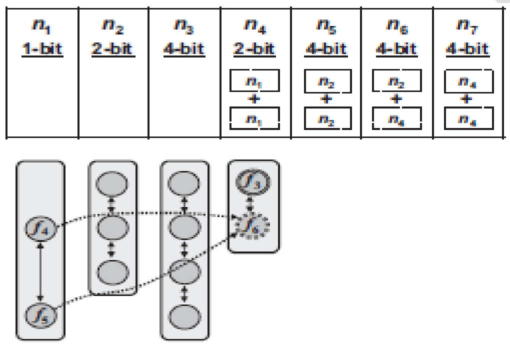


n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_1
f_2
f_3

Fig 6(b) Two 1-bit flip-flops, *f*1 and *f*2, are replaced by the 2-bit flip-flop *f*3.

Step3: Replaced 2-Bit Flip-Flop *f*6.

Similarly, *f*4 and *f*5 are combined to obtain a new flip-flop *f*6, and the result is shown in Fig. 6 (c)

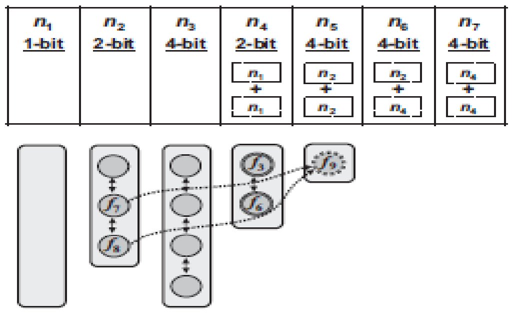


n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_4
f_5
f_3
f_6

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

Step4: Replacing 4-Bit Flip-Flop *f*9

After all flip-flops in the combinations of 1-level trees (*n*4 and *n*5*)* are obtained as shown in Fig. 6(d), we start to form the flip-flops in the combinations of 2-level trees (*n*6, and *n*7*)*.

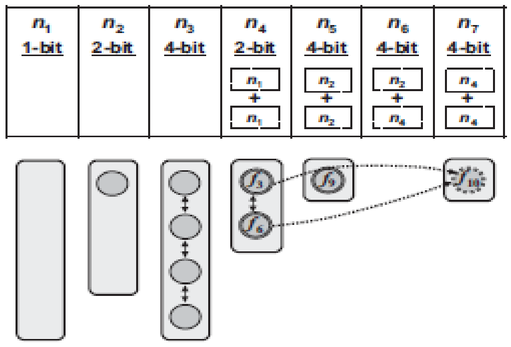


n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_7
f_8
f_3
f_6
f_9

Fig 6(d) Two 2-bit flip-flops, *f*7 and *f*8, are replaced by the 4-bit flip-flop *f*9.

Step5: Replacing 4-bit flip-flop *f*1

In Fig. 3.6 (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 flip-flops 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 merge-able, we can combine them to obtain a 4-bit flip-flop *f*10 in *n*7.

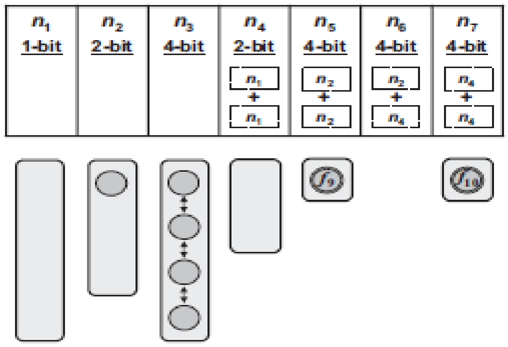


n_1 1-bit
n_2 2-bit
n_3 4-bit
n_4 2-bit n_1 + n_1
n_5 4-bit n_2 + n_2
n_6 4-bit n_2 + n_4
n_7 4-bit n_4 + n_4
f_3
f_6
f_9
f_{10}

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

Step6: Sets of Flip-Flops After Merging

Finally, there exists no couple of flip-flops that can be combined further, the procedure Finishes as shown in Fig. 6(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_9
f_{10}

Fig 6(f) Sets of flip-flops after merging.

Advantages of Merging Multi Bit Flip-Flops

- Smaller design area due to shared clock drivers and clock gating cells.
- Less delay and power of the clock network due to fewer clock sinks and smaller capacitive load on the clock net;
- Controllable clock skew because of common clock .
- The required routing resource for a scan chain is greatly reduced because of fewer cells in a scan chain.

III. RESULTS AND DISCUSSION

SIMULATION OF TWO 1-BIT FLIP-FLOP

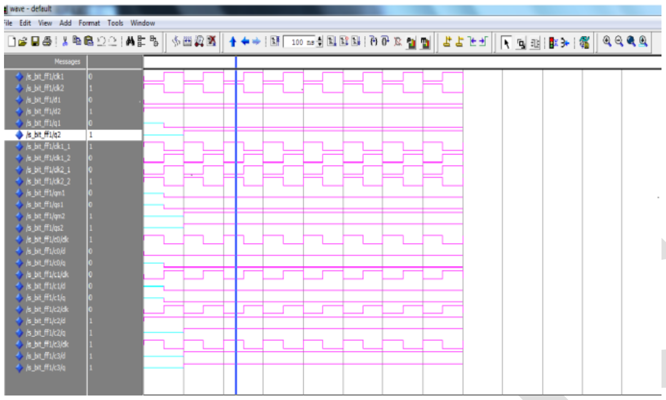


Fig. 8(a) Two single bit flip-flop output

Figure the simulation result of two 1-bit flop-flops. It requires two clock pulses to operate the flip-flop. When clk1 is enable, the input d1 is assign to the output q1. Otherwise, it stable in previous state. When clk2 is enable, the input d2 is assign to the output q2. Otherwise, it stable in previous state.

SIMULAION OF 2-BIT FLIP-FLOP OUTPUT

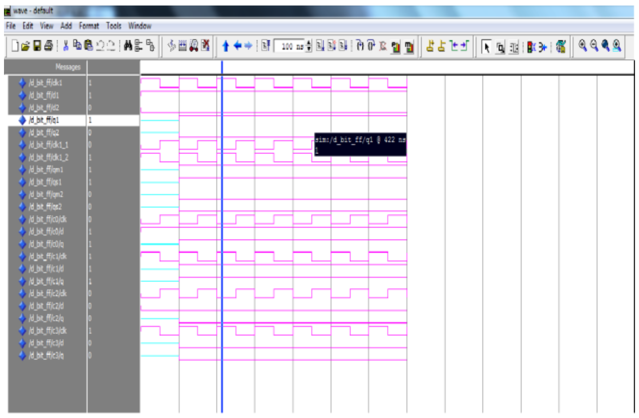


Fig. 8(b) double bit flip-flop simulation output

Figure shows the simulation result of 2-bit flop-flops. It needs one clock pulses to operate the flip-flop. When clk1 is enable, the input d1 is assign to the output q1 and also d2 is assign to q2. Otherwise, it stable in previous state.

SIMULATION OF COMBINATIONAL TABLE

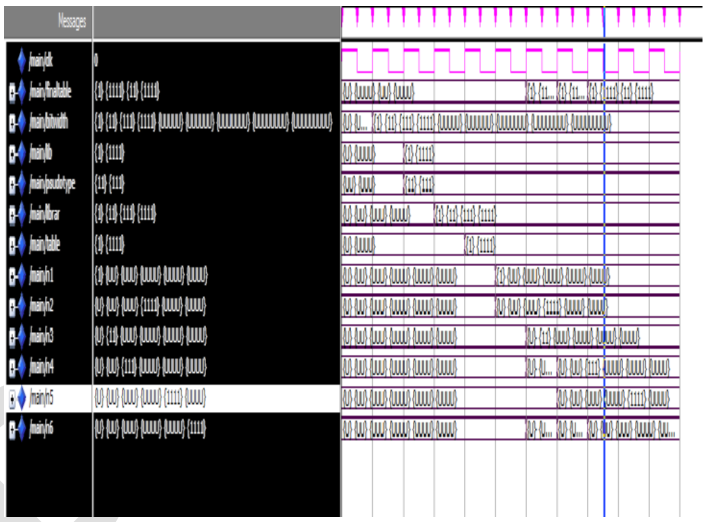


Fig. 8(c) Combinational Table output

Figure shows the simulation of combinational table. It has library and combinational table. Library has 1-bit and 4-bit flip-flop. Pseudo type of 2-bit and 4-bit flip-flops are added to the library. The combinational table creates each combinational of flip-flop response of each clock cycles. Then. Eliminate unused flip-flops and create final table.

SIMULATION OF MERGING MANY FLIP-FLOPS

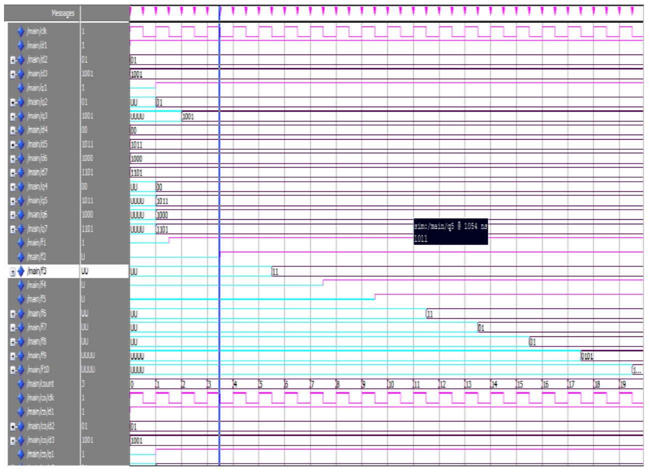
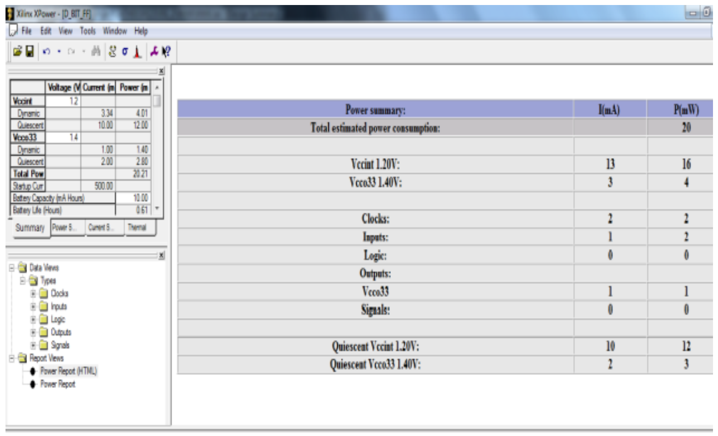


Fig 8(d) merging many flip-flops

Figure shows the simulation of merging many flip-flops by using one clock pulse. When the clock is enable, the input d1,d2,d3,d4,d5,d6 and d7 is assign to the output q1,q2,q3,q4,q5,q6 and q7. Otherwise it remains the previous state.

POWER ESTIMATION OF DOUBLE BIT FLIP-FLOP



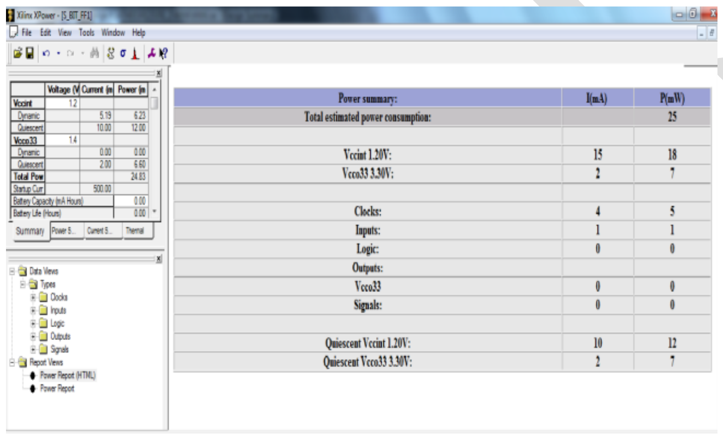
	Voltage (V	Current (m	Power (m
Vccint	1.2		
Dynamic		3.34	4.01
Quiescent		10.00	12.00
Vcco33	1.4		
Dynamic		1.00	1.40
Quiescent		2.00	2.80
Total Pow			20.21
Startup Cur		500.00	
Battery Capacity (mA Hours)			10.00
Battery Life (Hours)			0.61

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		20
Vccint 1.20V:	13	16
Vcco33 1.40V:	3	4
Clocks:	2	2
Inputs:	1	2
Logic:	0	0
Outputs:		
Vcco33	1	1
Signals:	0	0
Quiescent Vccint 1.20V:	10	12
Quiescent Vcco33 1.40V:	2	3

Fig 8(e) Power Estimation of Double Bit Flip-Flop

Figure shows the power Estimation of 2-bit flip-flop. The total power estimation of 2-bit flip flop is 20mW. This is smaller than using two 1-bit flip-flops. In this 2-bit flip-flop, we are using 1 clock is used for operating two flip-flop. Due to sharing clock tree, the power becomes reduced.

POWER ESTIMATION OF TWO SINGLE BIT FLIP-FLOP



	Voltage (V	Current (m	Power (m
Vccint	1.2		
Dynamic		5.19	6.23
Quiescent		10.00	12.00
Vcco33	1.4		
Dynamic		0.00	0.00
Quiescent		2.00	6.60
Total Pow			24.83
Startup Cur		500.00	
Battery Capacity (mA Hours)			0.00
Battery Life (Hours)			0.00

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		25
Vccint 1.20V:	15	18
Vcco33 3.30V:	2	7
Clocks:	4	5
Inputs:	1	1
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 1.20V:	10	12
Quiescent Vcco33 3.30V:	2	7

Fig 8(f) Power Estimation of Two Single Bit Flip-Flop

Figure shows the power Estimation of two 1-bit flip-flop. The total power estimation of two 1-bit flip flop is 25mW. This is greater than using 2-bit flip-flops. So, the 2-bit flip-flop is more benefit-able compared to two 1-bit flip-flops.

V. CONCLUSION AND FUTURE WORK

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 wire length reduction. Besides power reduction, the objective of minimizing the total wire length is also considered. to the cost function. The experimental results show that our algorithm can achieve a balance between power reduction and wire length reduction.

In future, the multi bit flips (or) shared clock pulse of the IC replaces to double edge triggered multi-bit flip-flop. It improves the overall performance and power consumption. Then, It also reduces the delay greatly. The total wire length is also minimized.

REFERENCES

[1] 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.

[2] W. Hou, D. Liu, and P.-H. Ho, "Automatic register banking for lowpower clock trees," in *Proc. Quality Electron. Design*, San Jose, CA, Mar. 2009, pp. 647–652.

[3] 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.

[4] H. Kawagachi and T. Sakurai, "A reduced clock-swing flip-flop (RCSFF) for 63% clock power reduction," in *VLSI Circuits Dig. Tech. Papers Symp.*, Jun. 1997, pp. 97–98.

[5] Y. Cheon, P.-H. Ho, A. B. Kahng, S. Reda, and Q. Wang, "Power-aware placement," in *Proc. Design Autom. Conf.*, Jun. 2005, pp. 795–800.

[6] W. Hou, D. Liu, and P.-H. Ho, "Automatic register banking for low power clock trees," in *Proc. Quality Electron. Design*, San Jose, CA, Mar. 2009, pp. 647–652.

[7] 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.

[8] . 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. IEEE/ACM Comput.-Aided Design Int. Conf.*, San Jose, CA, Nov. 2010,pp. 218–223.

[9] . J.-T. Yan and Z.-W. Chen, "Construction ofconstrained multi-bit flipflops for clock power reduction," in *Proc. ICGCS*, 2010, pp. 675–678.

[10] . 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 *Proc. ICCAD*, 2010, pp. 218–223.

[11] . 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.

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