CLOCK POWER MITIGATION OF MULTI-BIT FLIP-FLOPS USING MERGING TECHNIQUE

T.Bhuvaneswari[1] | C.Prema[2]

[1](Dept of ECE, Assistant Professor, KGiSL Institute of Technology, bhuvaneswari3008@gmail.com)

[2](Dept of ECE, Assistant Professor, KGiSL Institute of Technology, c.prema@kgkite.ac.in)

Abstract— *In modern VLSI design power has been become a burning issue. Reducing power consumption in design it enables better and cheaper products to be designed and power-related chip failures to be minimized. The power consumed by clocking gradually takes a dominant part. Multi-bit flip-flop is an effective method for clock power consumption reduction. The underlying idea behind multi-bit flip-flop method is to eliminate the total inverter number by sharing the inverters in the flip-flops. To deal with the difficulty efficiently, we have proposed several techniques. First, we perform a co-ordinate transformation to identify those flip-flops that can be merged and their legal regions. Besides, we show how to build a combination table to enumerate possible combinations of flip-flops provided by a library. Finally, we use a hierarchical way to merge flip-flops. The time complexity of our algorithm is (n1.12) less than the empirical complexity of (n2).Our algorithm significantly reduces clock power by 20–30% and the running time is very short. In the largest test case, which contains 1 700 000 flip-flops, our algorithm only takes about 5 min to replace flip-flops and the power reduction can achieve 21%.*

Keywords— VLSI; CMOS; SOC

1. INTRODUCTION

The main objective of this circuit to implement on low power VLSI, to design a multi-bit flip-flop using shared inverters concepts. This facilitates a reduces wire-length and reduces the power.

2. OVERVIEW

Due to the popularity of portable electronic products, low power system has attracted more attention in recent years. As technology advances, systems-on-a-chip (SoC) design can contain more and more components that lead to a higher power density. 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. Therefore, the consideration of power consumption in complex SOCs has become a big challenge to designers. Moreover, 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. Thus, several methodologies, 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. 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. Because of this phenomenon, several flip-flops can share a common clock buffer to avoid unnecessary power waste.

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 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.

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 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.

To deal with this problem, the direct way is to repeatedly search a set of flip-flops that can be replaced by a new multi-bit flip-flop until none can be done. However, as the number of flip-flops in a chip increases dramatically, the complexity would increase exponentially, which makes the method impractical. To handle this problem more efficiently and get better results, we have used the following approaches.

- To facilitate the identification of mergeable flip-flops, we transform the coordinate system of cells. In this way, the memory used to record the feasible placement region can also be reduced.

- To avoid wasting time in finding impossible combinations of flip-flops, we first build a combination table before actually merging two flip-flops. For example, if a library only provides three kinds of flip-flops, which are 1-, 2-, and 3-bit, we first separate the flip-flops into three groups. Therefore, the combination of 1- and 3-bit flip-flops is not considered since the library does not provide the type of 4-bit flip-flop.

- We partition a chip into several sub regions and perform replacement in each sub region to reduce the complexity. However, this method may degrade the solution quality. To resolve the problem, we also use a hierarchical way to enhance the result.

3. MODIFIED SINGLE-BIT FLIP-FLOPS

A. INTRODUCTION

Cell library L and a placement which contains a lot of flip-flops, target is to merge as many flip-flops as possible in order to reduce the total power consumption. If we want to replace some flip-flops $f1,..., f j-1$ by a new flip-flop $f j$, the bit width of $f j$ must be equal to the summation of bit widths in the original ones (i.e., $\sum bi = b j$, $i = 1$ to $j-1$). Besides, since the replacement would change the routing length of the nets that connect to a flip- flop, it inevitably changes timing of some paths. Finally, to ensure that a legalized placement can be obtained after the replacement, there should exist enough space in each bin. To consider these issues, define two constraints as follows.

- Timing Constraint for a Net Connecting to a Flip-Flop f j from a Pin pi : To avoid that timing is affected after the replacement, the Manhattan distance between *pi* and *f j* cannot be longer than the given constraint *S(pi)* defined on the pin *pi* [i.e., $M(pi , f j) \leq S(pi)$].Based on each timing constraint defined on a pin, we can find a feasible placement region for a flip-flop ***fj*** . See Fig.6.1for example. Assume pins *p*1 and *p*2 connect to a 1-bit flip-flop *f*1. Because the length is measured by Manhattan distance, the feasible placement region of *f*1 constrained by the pin *pi* [i.e., $M(pi , f1) \leq S(pi)$] would form a diamond region, which is denoted by *Rp(pi)*, *i* = 1 or 2. See the region enclosed by dotted lines in the figure. Thus, the legal placement region of *f*1 would be the overlapping region enclosed by solid lines, which is denoted by *R(f*1*)*. *R(f*1*)* is the overlap region of *Rp(p*1*)* and *Rp(p*2*)*.

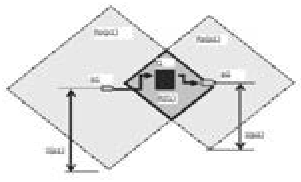


Figure.3.1. Defined slack region of the pin

Capacity Constraint for Each Bin **B**k : The total area of flip-flops intended to be placed into the bin *Bk* cannot be larger than the remaining area of the bin *Bk* (i.e., $\sum A(fi) \leq RA(Bk))$.

4. PROPOSED ARCHITECTURE

A. DESIGN FLOW

Our design flow can be roughly divided into three stages. 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 *fi* can be obtained by the 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. In this section, we would detail each stage of our method. In the first subsection, we show a simple formula to transform the original coordination system into a new one so that a legal placement region for each flip-flop can be identified more easily. The second subsection presents the flow of building the combination table. Finally, the replacements of flip-flops will be described in the last subsection.

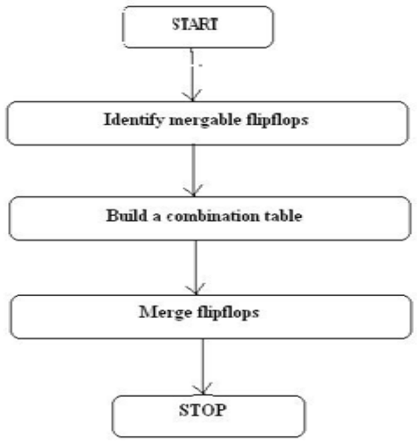


Figure: 4.1 Flow chart of our algorithm

B. TRANSFORMATION OF PLACEMENT SPACE

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.7.1.1(a), there are two pins p1 and p2 connecting to a flip-flop f1, and the feasible placement regions for the two pins are enclosed by dotted lines, which are denoted by Rp(p1) and Rp(p2), respectively. Thus, the legal placement region R(f1) for f1 is the overlapping part of these regions. In Fig. 7.1.1(b), R(f1) and R(f2) represent the legal placement regions of f1 and f2. Because R(f1) and R(f2) overlap, we can replace f1 and f2 by a new flip-flop f3 without violating the timing constraint, as shown in Fig. 7.1.1(c). However, it is not easy to identify and record feasible placement regions if their shapes are diamond. Moreover, four coordinates are required to record an overlapping region [see Fig. 7.1.2(a)]. Thus, if we can rotate each segment 45°, the shapes of all regions would become 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.7.1.3 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).

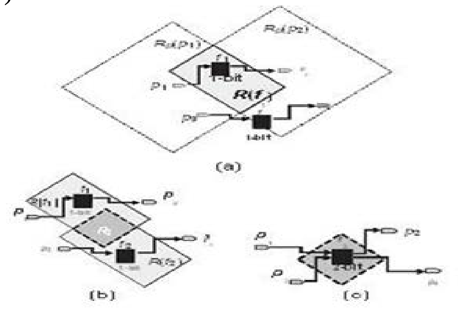


Figure 4.1. (a) Feasible regions*Rp (p*1*)* and *Rp(p*2*)* for pins *p*1 and *p*2 which are enclosed by dotted lines, and the legal region *R(f*1*)* for *f*1 which is enclosed by solid lines. (b) Legal placement regions *R(f*1*)* and *R(f*2*)* for *f*1 and *f*2, and the feasible area *R*3 which is the 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.

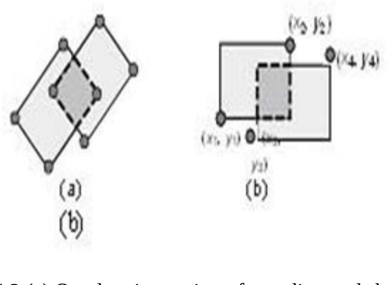


Figure 4.2 (a) Overlapping region of two diamond shapes. (b) Rectangular shapes obtained by rotating the diamond shapes in (a) by45°.

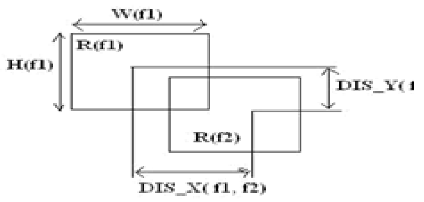
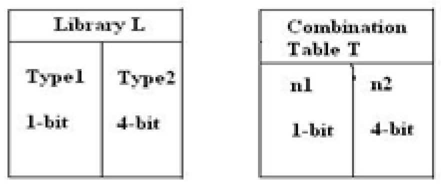


Figure 4.3 Overlapping relation between available placement regions of f 1and f 2.

C. BUILD A COMBINATION TABLE

If we want to replace several flip-flops by a new flip-flop fi'(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 fi' is provided by the library L when the feasible regions of these flip-flops overlap. Build a combination table, 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. Since only one combination of flip-flops needs to be considered in each time, the search time can be reduced greatly. We use a binary tree to represent one combination for simplicity. Each node in the tree denotes one type of a flip-flop in L. The types of flip-flops denoted by leaves will constitute the type of the flip-flop in the root. For each node, the bit width of the corresponding flip-flop equals to the bit width summation of flip-flops denoted by its left and right child [please see Fig. 7.1.4(e) for example].

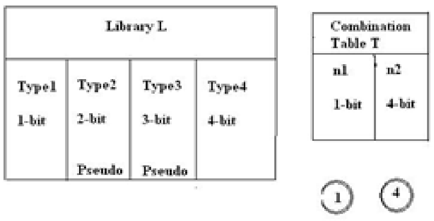
In order to use a binary tree to denote a combination whose bit width is 4, there must exist flip-flops whose bit widths are 2 and 3 in L . If the combination is not included into any other combinations, it will be deleted. For example, suppose a library L only provides two types of flip-flops, whose bit widths are 1 and 4 (i.e., bmin = 1 and bmax = 4), in Fig. 9(a). We first initialize two combinations n1 and n2 to represent these two types of flip-flops in the table T [see Fig. 7.1.4(a)]. 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. 7.1.4(b)]. Now, for each combination in T, we would build a binary tree with 0-level, and the root of the binary tree denotes the combination. Next, we try to build new legal combinations according to the present combinations. By combing two 1-bit flip-flops in the first combination, a new combination n3 can be obtained [see Fig. 7.1.4(c)]. Similarly, we can get a new combination n4 (n5) by combining n1 and n3(two n3's) [see Fig. 7.1.4(d)]. Finally, n6 is obtained by combing n1 and n4. All possible combinations of flip-flops are shown in Fig. 9(e). Among these combinations, n5 and n6 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 our program, n6 is deleted from T rather than n5 because its height is larger. After this procedure, n4 becomes an unused combination [see Fig. 7.1.4(e)] since the root of binary tree of n4 corresponds to the pseudo type, type3, in L and it is only included in n6. After deleting n6, n4 is also need to be deleted. The last combination table T is shown in Fig. 7.1.4(f)



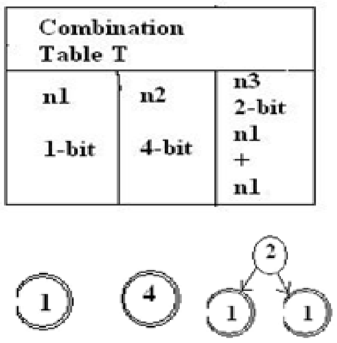
Library L	
Type1	Type2
1-bit	4-bit

Combination Table T	
n1	n2
1-bit	4-bit

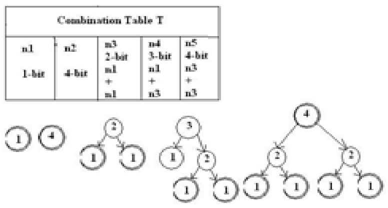
(a)



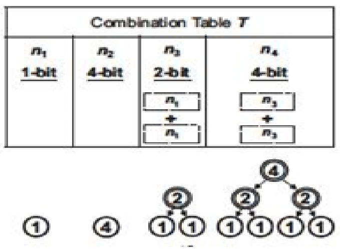
(b)



(c)



(d)



(e)

Figure 4.4. Example of building the combination table. (a) Initialize the library
L and the combination table T . (b)
Pseudo types are added into L, and the corresponding binary tree is also build for each combination in T. (c) New combination n3 is obtained from combining two n1s. (d) New combination n4 is obtained from combining n1 and n3, and the combination n5 is obtained from combining two n3s.(e) Last combination table is obtained after deleting the unused combination

D. MERGE FLIP-FLOPS

Now, show how to use the combination table to combine flip-flops in this subsection. To reduce the complexity, we first divide the whole placement region into several sub-regions, and use the combination table to replace flip-flops in each sub-region. Then, several sub-regions are combined into a larger sub-region and the flip-flops are replaced again so that those flip-flops in the neighboring sub-regions 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. Fig.4.5 shows this flow.

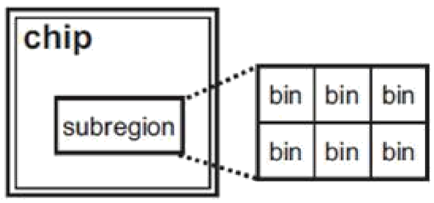


Figure.4.5 Region partition with six bins in one region

• Region partition (optional): 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.

• Replacement of flip flops in each sub-region: corresponding to their types in the merge flip-, we first give an equation to cost = routing length– α x (available _area) ^1/2 Where routing _ length denoted 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 as is a weighting factor. The cost function includes the term routing length to favor a replacement that induces shorter wavelength. 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 future and more power reduction. Thus, we will give it a smaller cost. Once the flip flop cannot be merged to a higher –bit type we ignore the available _area in the cost function, and hence are set to 0.After combination table. First, we link flip flops below the combinations measure library. Then, quality factor for each if two combination flip-flops as linked the left child and right child of the root. For example, given a library containing three types of flip-flops(1-,2-, and 4-bit), we first build a combination table T as shown in fig7.16(a). In the beginning, the flip-flops with various types are, respectively, linked below n1, n2, and n3 in T according to their types. Suppose we want to form a flip-flop in n4, which needs two 1-bit flip-flops according to the combination table. Each pair of flip-flops in n1 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.7.1.6(a), f1 and f2 are chosen because their combination gains the smallest cost. Thus, we add a new node f3 in the list below n4, and then delete f1 and f2 from their original list [see Fig. 7.1.6(b)]. Similarly, f4 and f5 are combined to obtain a new flip-flop f6, and the result is shown in Fig.7.1.6(c). After all flip-flops in the combinations of 1-level trees (n4 and n5) are obtained as shown in Fig.7.1.6(d), we start to form the flip-flops in the combinations of 2-level trees (n6, and n7). In Fig.7.1.6(e), there exist some flip-flops in the lists below n2 and n4, and merge them to get flip-flops in n6 and n7, respectively. Suppose there is no overlap region between the couple of flip-flops in n2 and n4. It fails to form a 4-bit flip-flop in n6. Since the 2-bit flip-flops f3 and f6 are mergeable, we can combine them to obtain a 4-bit flip-flop f10 in n7. Finally, because there exists no couple of flip-flops that can be combined further, the procedure finishes as shown in Fig.7.1.6(f). If the available overlap region of two flip-flops exists, assign a new one to replace those flip-flops.

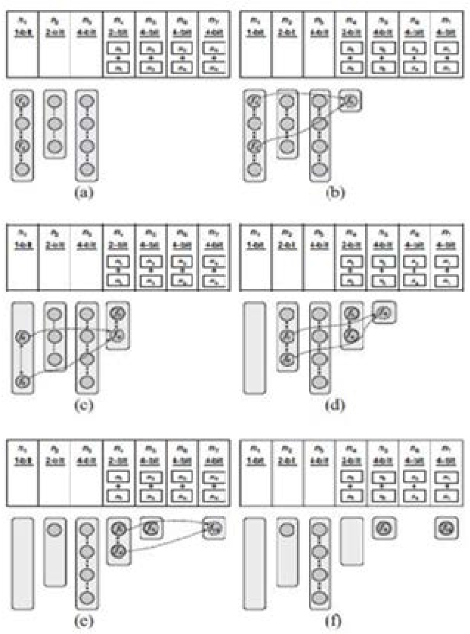


Figure46. Example of replacements of flip-flops. (a) Sets of flip-flops before merging. (b) Two 1-bit flip-flops, f1 and f2, are replaced by the 2-bit flip-flopf3. (c) Two 1-bit flip-flops, f4 and f5, are replaced by the 2-bit flip-flop f6. (d) Two 2-bit flip-flops, f7 and f8, are replaced by the 4-bit flip-flop f9. (e) Two 2-bit flip-flops, f3 and f6, are replaced by the 4-bit flip-flop f10.(f)Sets of flip-flops after merging

5. RESULTS

A. *POWER ANALYSIS: COMPARISON TABLE*

The table shows the various test circuits and experimental results. Case 5 is the largest circuit of about 1 700 000 flip-flops. Because the execution time is dominated by the number of flip-flops in the circuit, Case 5 is applied to help to demonstrate the efficiency and robust of our algorithm. Row 1 in the table lists all test cases and row 2 shows types of different flip-flops that can be used in each test case. Rows 3 and 4 respectively, show numbers of flip-flops and total power consumption in original test cases. After some flip-flops are replaced by our algorithm, the power consumption of each design is shown in row 5, and row 6 computes the ratio of power reduction by our algorithm, which is denoted by PR_ Ratio. From rows 7 to 9, it shows the wire-length reduction by our algorithm. Rows 7 and 8 show the original wire-length and the wire-length after our program is applied. Finally, the ratio of wire-length reduction, which is denoted by WR_Ratio, is shown in row 9.The values of PR_ Ratio in all cases are between 20 and 30. Besides, the wire-length are less than the original circuit in all cases, and the best value of WR_Ratio can achieve 42.18% improvement. Row 10 shows the execution time of each case. Because of the long execution time of parser, show the execution time of parser in row 11.

6. CONCLUSION

This paper has proposed 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. Besides power reduction, the objective of minimizing the total wire length also be considered to the cost function. The experimental results show that our algorithm can achieve a balance between power reduction and wire length reduction. Moreover, even for the largest case which contains about 1 700 000 flip-flops, our algorithm can maintain the performance of power and wire length reduction in the reasonable processing time.

TABLE I. POWER ANALYSIS: COMPARISON TABLE

	Case 1	Case 2	Case 3	Case 4	Case 5
Library	1,2,4	1,2,4,8,6	1,2,4,13	1,2,4,8	1,2,4,8
Flip-flop number	120	953	5524	60000	1728000
Power (unit 10^3)	12	95	552	6000	172800
Power merged(unit 10^3)	9	67	430	4208	136509
PR RATIO(%)	20.97	28.80	22.11	29.87	21.00
WLori(unit 10^3)	83	577	3563	53625	1199304
WL merged(unit 10^3)	71	506	2189	31008	1068961
WR RATIO(%)	85.62	87.77	61.44	57.82	89.13
TIMES (S)	0.08	0.24	1.07	36.7	2377
Times of parser	0.07	0.15	0.29	3.8	2153

REFERENCES

[1] Ya-Ting Shyu, Jai-Ming Lin, Chun-Po Huang, Cheng-Wu Lin, Ying-Zu Lin, and Soon-Jyh Chang, Member, IEEE" Effective and Efficient Approach for Power Reduction by Using Multi-Bit Flip-Flops," IEEE transactions on very large scale integration (VLSI) systems, vol. 21, no. 4, april 2013
[2] P. Gronowski, W. J. Bowhill, R. P. Preston, M. K. Gowan, and R. L.Allmon, "High-performance microprocessor design," IEEE J. Solid-StateCircuits, vol. 33, no. 5, pp. 676–686, May1998.
[3] 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.
[4] 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.
[5] H.Kawagachi and T.Sakuari,"a reduced clock-swing flip flop (RCSFF) for 63% clock power reduction ,'in VLSI circuits Dig .Tech.PapersSymp,jun.1997,pp.97-98.
[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. IEEE/ACM Comput.-Aided Design Int.Conf.,San Jose,CA,NOV.2010, pp.218-223.
[7] Faraday Technology Corporation[Online].Available :http://www. Faraday-tech.com/index.html
[8] C.Bron and J.Kerbosch,"Algorithm 457:Finding all cliques of an undirected graph,''ACMCommun,vol.16,no.9,pp.575-577,1973.
[9] CAD Contest of Taiwan[online].Available: http://cad_contest.cs.Nctu.eu.tw/cad11