ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

**International Journal of Innovative Research in Computer and Communication Engineering**

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

**Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)**

**Organized by**

**Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014**

# Low Power Multi Bit Flip Flops Design for VLSI Circuits

L.Angel Prabha[1], S.Joy[2]

M.E Applied Electronics, V.S.B Engineering College, Karur, Tamilnadu, India[1, 2]

**ABSTRACT:** In this paper we present a power optimization technique to reduce clock power by using multi bit flip flop method. We have proposed the several techniques to overcome the problems of flip- flops replacement without timing and placement capacity constraints violation. First we perform the co –ordinate transformation to convert the diamond shape legal region into rectangular region and Building the Combination table to identify the Mergeable Flip-flops' Moreover, by judiciously merging and placing the MBFFs, the total wire length is also significantly reduced.

**KEYWORDS**— Clock power reduction, merging, multi-bit flip-flop, replacement, wirelength.

## I. INTRODUCTION

Due to the popularity of portable electronic products, Low power system has attracted more attention in recent years. 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. Moreover, in modern VLSI designs, power consumed by clocking has taken a major part of the whole design.. 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. Fig. 1 shows the clock diagrams of 1- and 2-bit flip-flops. 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 be considering the area capacity of the region. 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.

### A.RELATED WORK

They use the graph-based approach. 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 correspond- ing 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.

B.OUR CONTRIBUTIONS

The difficulty of this problem has been illustrated in the above descriptions. 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.
1) 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.
2) 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.
3) 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.

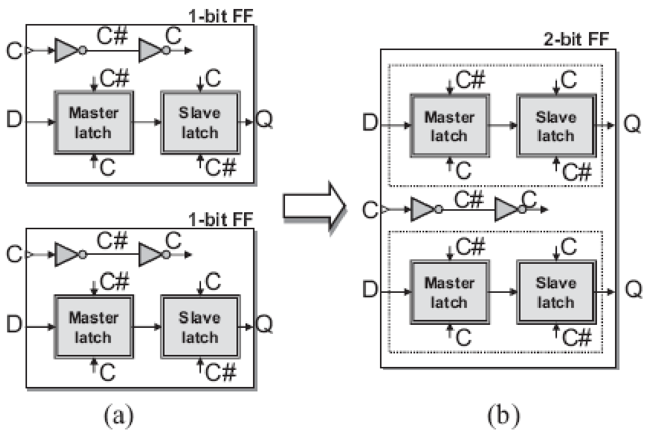


Fig. 1. Example of merging two 1-bit flip-flops into one 2-bit flip-flop.
(a) Two 1-bit flip-flops (before merging). (b) 2-bit flip-flop (after merging).

## II. PROPOSED METHOD

DESIGN FLOW

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 *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

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

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.

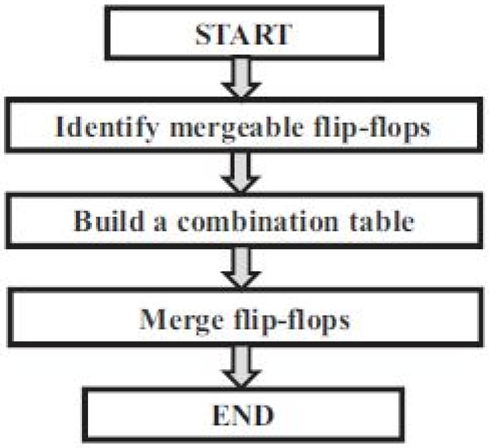


FIG. 2. FLOW CHART OF OUR ALGORITHM

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.

## III. BUILD A COMBINATION TABLE

we may add pseudo types, which denote those flip-flops that are not provided by the library, For example, assume that a library only supports two kinds of flip-flops whose bit widths are 1and 4, respectively. 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.

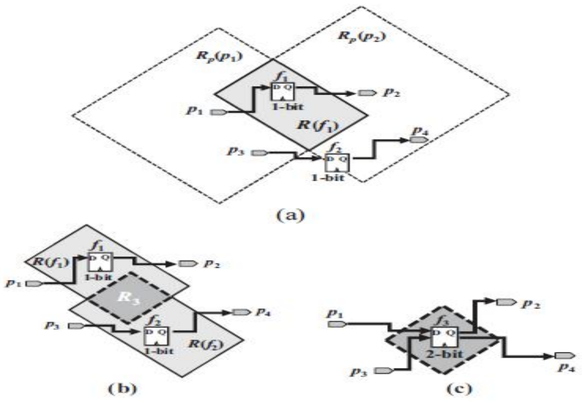


Fig. 3. (a) Feasible regions*Rp (p*1*)* and *Rp(p*2*)* for pins *p*1 and *p*2 which are enclosed by dotted lines, and the legal region *R( f*1*)* for *f*1 which is enclosed by solid lines. (b) Legal placement regions *R( f*1*)* and *R( f*2*)* for *f*1 and *f*2, and the feasible area *R*3 which is the 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 p1,p2,p3 and p4

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

**International Journal of Innovative Research in Computer and Communication Engineering**

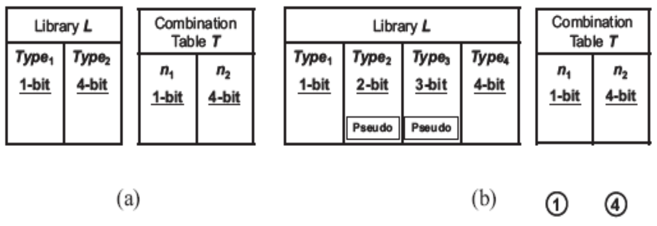
(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

**Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)**

**Organized by**

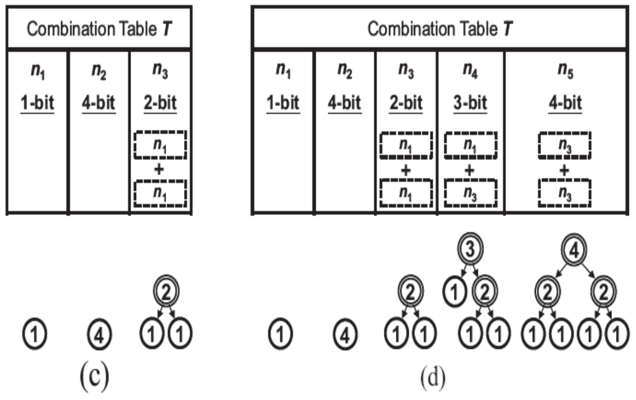
**Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014**

Let *b*max and *b*min denote the maximum and minimum bit width of flip-flops in L. In InsertPseudoType, it inserts all flip-flops whose bit widths are larger than *b*min and smaller than *b*max into *L* if they are not provided by *L* originally. After this procedure, all combinations in *L* are sorted according to their bit widths in the ascending order. At present, all combinations are represented by binary trees with 0-level. Thus, we would assign NULL to its right and left child. Finally, for every two kinds of combinations in *T*, we try to combine them to create a new combination. If the new combination is the flip-flop of a feasible type (this can be checked by the function Type Verify), we would add it to the table *T*. In the function Type Verify, we first add the bit widths of the two combinations together and store the result in *b*sum. Then, we will add a new combination *n* to *T* with bit width *b*sum if *L* has such kind of a flip-flop. After these procedures, there may exist some duplicated or unused combinations in *T*. Thus, we have to delete them from the table and the two functions Duplicate Combination Delete and Unused Combination Delete are called for the purpose In DuplicateCombinationDelete, it checks whether the duplicated combinations exist or not. If the duplicated combinations exist, only the one with the smallest height of its corresponding binary tree is left and the others are deleted. In Unused Combination Delete, it checks the combinations whose corresponding type is pseudo in L. If the combination is not included into any other combinations, it will be deleted.

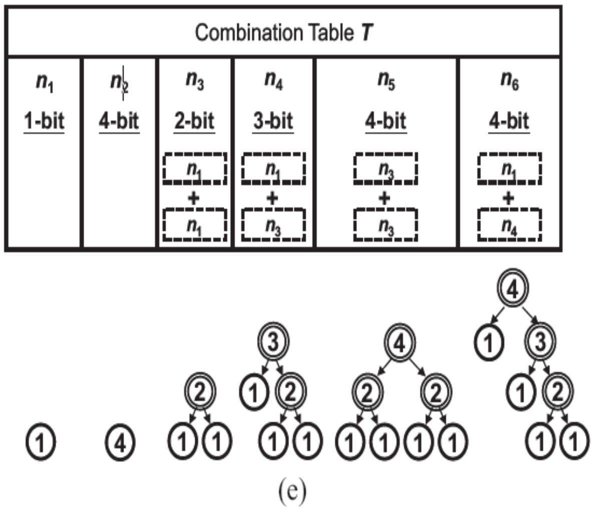


.

Fig. 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 *n*3 is obtained from combining two *n*1s. (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.



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

## IV. MERGE FLIP-FLOPS

In merging of flip flops to reduce the complexity, we first divide the whole placement region into several subregions, and use the combination table to replace flip-flops in each subregion. 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 library.

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 we divide the region into several subregion, each subregion contains sixbins.bin is a small unit of subregion.

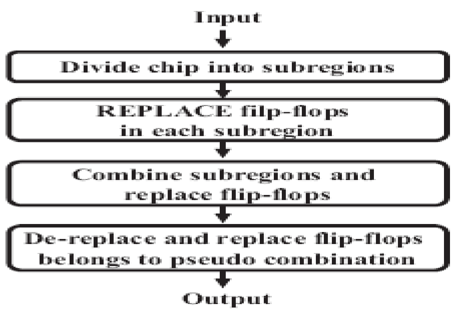
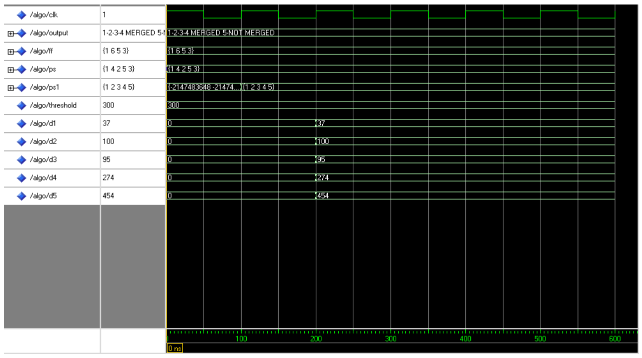

Input
Divide chip into subregions
REPLACE filp-flops in each subregion
Combine subregions and replace flip-flops
De-replace and replace flip-flops belongs to pseudo combination
Output

Fig 5 detailed flow to merge flipflops

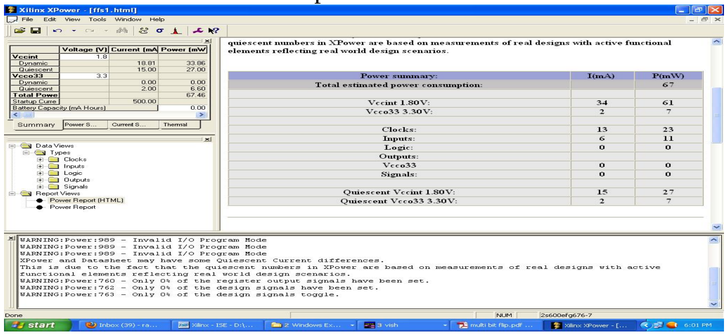
## V. SIMULATION RESULTS

Flip flop Merging output



| | | |
|---|---|---|
| /algo/clk | 1 | |
| /algo/output | 1-2-3-4 MERGED 5-N | 1-2-3-4 MERGED 5-NOT MERGED |
| /algo/ff | {1 6 5 3} | {1 6 5 3} |
| /algo/ps | {1 4 2 5 3} | {1 4 2 5 3} |
| /algo/ps1 | {1 2 3 4 5} | {-2147483648 -21474... {1 2 3 4 5} |
| /algo/threshold | 300 | 300 |
| /algo/d1 | 37 | 0 37 |
| /algo/d2 | 100 | 0 100 |
| /algo/d3 | 95 | 0 95 |
| /algo/d4 | 274 | 0 274 |
| /algo/d5 | 454 | 0 454 |

Proposed Power



Xilinx XPower - [ffs1.html]

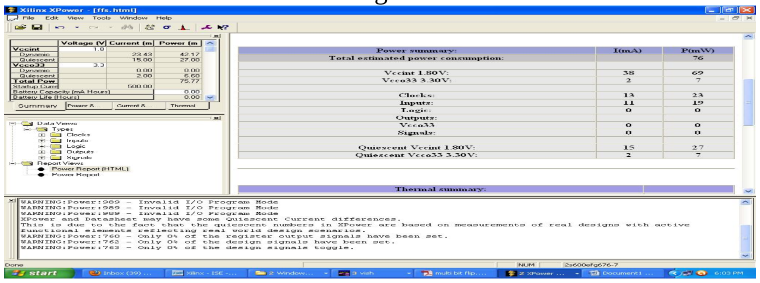
| | Voltage (V) | Current (mA | Power (mW |
|---|---|---|---|
| Vccint | 1.8 | | |
| Dynamic | | 18.81 | 33.86 |
| Quiescent | | 15.00 | 27.00 |
| Vcco33 | 3.3 | | |
| Dynamic | | 0.00 | 0.00 |
| Quiescent | | 2.00 | 6.60 |
| Total Powe | | | 67.46 |
| Startup Curre | | 500.00 | |
| Battery Capacity (mA Hours) | | | 0.00 |

quiescent numbers in XPower are based on measurements of real designs with active functional elements reflecting real world design scenarios.

| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 67 |
| | | |
| Vccint 1.80V: | 34 | 61 |
| Vcco33 3.30V: | 2 | 7 |
| | | |
| Clocks: | 13 | 23 |
| Inputs: | 6 | 11 |
| Logic: | 0 | 0 |
| Outputs: | | |
| Vcco33 | 0 | 0 |
| Signals: | 0 | 0 |
| | | |
| Quiescent Vccint 1.80V: | 15 | 27 |
| Quiescent Vcco33 3.30V: | 2 | 7 |

```
WARNING:Power:989 - Invalid I/O Program Mode
WARNING:Power:989 - Invalid I/O Program Mode
WARNING:Power:989 - Invalid I/O Program Mode
XPower and Datasheet may have some Quiescent Current differences.
This is due to the fact that the quiescent numbers in XPower are based on measurements of real designs with active
functional elements reflecting real world design scenarios.
WARNING:Power:760 - Only 0% of the register output signals have been set.
WARNING:Power:762 - Only 0% of the design signals have been set.
WARNING:Power:763 - Only 0% of the design signals toggle.
```

Existing Power



Xilinx XPower - [ffs.html]

| | Voltage (V | Current (m | Power (m |
|---|---|---|---|
| Vccint | 1.8 | | |
| Dynamic | | 23.43 | 42.17 |
| Quiescent | | 15.00 | 27.00 |
| Vcco33 | 3.3 | | |
| Dynamic | | 0.00 | 0.00 |
| Quiescent | | 2.00 | 6.60 |
| Total Pow | | | 75.77 |
| Startup Curre | | 500.00 | |
| Battery Capacity (mA Hours) | | | 0.00 |
| Battery Life (Hours) | | | 0.00 |

| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 76 |
| | | |
| Vccint 1.80V: | 38 | 69 |
| Vcco33 3.30V: | 2 | 7 |
| | | |
| Clocks: | 13 | 23 |
| Inputs: | 11 | 19 |
| Logic: | 0 | 0 |
| Outputs: | | |
| Vcco33 | 0 | 0 |
| Signals: | 0 | 0 |
| | | |
| Quiescent Vccint 1.80V: | 15 | 27 |
| Quiescent Vcco33 3.30V: | 2 | 7 |

Thermal summary:

```
WARNING:Power:989 - Invalid I/O Program Mode
WARNING:Power:989 - Invalid I/O Program Mode
WARNING:Power:989 - Invalid I/O Program Mode
XPower and Datasheet may have some Quiescent Current differences.
This is due to the fact that the quiescent numbers in XPower are based on measurements of real designs with active
functional elements reflecting real world design scenarios.
WARNING:Power:760 - Only 0% of the register output signals have been set.
WARNING:Power:762 - Only 0% of the design signals have been set.
WARNING:Power:763 - Only 0% of the design signals toggle.
```

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

## REFERENCES

[1] 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, , May 1998.

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

[4] 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.

[5] 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. 201pp. 218–223.