Rewired Retiming for Flip-flop Reduction and Low Power without Delay Penalty

JIANG, Mingqi

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Computer Science and Engineering

The Chinese University of Hong Kong

September 2009

Thesis/Assessment Committee

Professor XU Qiang (Chair)

Professor Prof. WU Yu Liang David (Thesis Supervisor)

Professor YOUNG Fung Yu (Committee Member)

Professor Huang Shi Yu (External Examiner)

Abstract of the thesis entitled:

Rewired Retiming for Flip-flop Reduction and Low Power without Delay Penalty

Submitted by JIANG Mingqi
for the degree of Master of Philosophy
at The Chinese University of Hong Kong in (July, 2009)

Abstract

Retiming is an optimization technique for sequential circuits by repositioning flip-flops across the combinational elements of the circuit. It has been applied to different areas such as logic synthesis, circuit partitioning, power reduction etc. However, due to the intrinsic difference between fan-in and fan-out counts of a retimed component, the number of flip-flops tends to be undesirably increased in a conventional retiming procedure, which can cause a significant area/power penalty on the retimed circuit. Moreover, because of the higher dominance on interconnect delays, without a mechanism to reflect real physical design accurately, the clock period produced by a retiming scheme will be unrealistic.

To overcome these two major drawbacks of the conventional retiming technique, we propose a novel retiming flow combined with rewiring, being able to largely cut down flip-flops (FFs) while with the original retimed clock period uncompromised. For a more accurate delay estimation, all interconnect delays are formulated and calculated based on real placements.

Experimental results show that this novel rewired retiming scheme can bring a reduction of 18.7% averagely on the number of flip-flops compared to the original retiming without rewiring. This large FF reduction can be considered a free gain as the retimed clock period can still be kept without compromise. And meanwhile, due to such FF reduction, about 8.26% of the total dynamic power can be saved.

摘　要

时序重排是一种通过在组合逻辑之间对触发器进行重新定位的针对时序电路的优化技术。它已经被广泛应用于不同领域，比如逻辑综合，电路割分，功耗降低等等。然而，由于被时序重排的逻辑单元输入和输出的数目不一样，在传统的时序重排中触发器的数目通常会大量增加，这会引起面积和功耗的巨大增加。另外，由于线上的时延占了主导地位，没有一个机制来准确地反映物理层的设计，在时序重排中得出的最优化时钟周期会难以实现。

为了克服现有的时序重排的这两点缺陷，我们提出一种结合了逻辑再接线技术的时序重排流程，可以大量地减少触发器的数目，并且维持原本的时序重排可得的最优化时钟周期。为了实现更准确的时延估计，所有的线上时延都被模型化并在真正的电路布局上计算出来。

实验结果显示，较之原有的时序重排技术，这种新型的结合逻辑再接线技术的时序重排技术可以平均减少 18.7%的触发器。这么显著的触发器数目的下降是完全没有代价的获益，因为时钟周期的优化没有任何改变。同时，由于触发器数目的减少，8.26% 的总动态功耗可以被节省。

Acknowledgement

I would like to express my highest gratitude to my supervisor Prof. David Yu-Liang Wu, for his consistent support and guidance throughout my 2 years' M.Phil. study. His earnest attitude, endless endeavor towards research and never-give-up spirit has been leading my research as well as my daily life.

Besides, I must thank Prof. Evangeline F.Y. Young for her support and comments on the work. Also, my fellow colleagues in the VLSI CAD group, they are so very much helpful and provide me with every care and support along the way, with so much inspiring discussion and suggestions.

Table of Contents

list of Figures

list of Tables

Chapter 1

Introduction

Following the Moore's Law, the Very Large Scale Integration (VLSI) technology has soared over the last decades, which brings the Electronic Design Automation (EDA) tools to an even more significant and indispensable role in the industry. As the process technology advances, the requirements for EDA techniques become more and more sophisticated. EDA optimization techniques mainly include reducing the circuit area, delay and power from different stages of the VLSI design. As technology advances to deep submicron, people's concerns have gradually shifted from area to timing and power dissipation.

Retiming is an EDA optimization technique which is originally designed for timing optimization [1]. It minimizes the circuit clock period for sequential circuits by repositioning flip-flops across the combinational elements of the circuit. The later works in retiming involve optimizing the cycle time [1] [2], reducing the area by minimizing the number of flip-flops [3] [4] etc. It has been also applied to different practical applications, such as logic synthesis [4][5], circuit partition[6][7], power reduction [8][9]and testability[10].

In most of the existing approaches, the problem of flip-flop placement is viewed from a purely graph-based perspective, with all logic information about the circuit being discarded during retiming. Approaches which incorporate retiming with logic re-synthesis are thus proposed in [4] to try to exploit the possibility of improvement by utilizing the extra freedom.

In [4], the authors proposed a circuit optimization approach in which all the flip-flops are temporarily moved to the boundaries of the combinational network using retiming. Re-synthesis is then performed on the combinational logic between the flip-flops. This is one of the first attempts to couple the movement of flip-flops by retiming and combinational re-synthesis techniques to achieve circuit optimization goals. However, their technique is primarily targeted at minimizing the number of literals of the circuit. No real placement information is utilized to handle the interconnect delay factor that dominates in circuit design nowadays. Clearly, any logic synthesis flow would be more accurate and effective if physical information obtained from real place and route can be integrated together.

As the VLSI process technology scales down to a deep submicron era, traditional retiming algorithm which ignores the interconnect delay is no longer accurate enough, because the interconnect delay can be much dominating and larger than the logic/gate delay. In [11], the retiming problem is re-formulated to include both gate and interconnect delays, in which the interconnect delay is assumed to be proportional to the wire length.

However, as demonstrated in [12], the optimal clock period gained from retiming may not be feasible after the circuit is really placed. As a large number of flip-flops are relocated and the flip-flop number will usually increase after retiming, a traditionally retimed optimal clock period might not be close to reality in a legalized placement. Moreover, a larger amount of power consumption can be introduced due to the increased flip-flops. Therefore, besides delay improvement, it's also important to cut down the retiming-induced flip-flops for both area and power reductions, which is a

problem not addressed in [11].

In this work, we will integrate an interconnect delay based retiming with rewiring to achieve better and more accurate circuit optimizations. Rewiring [13-16] is an optimization technique in logic re-synthesis, a powerful tool for combinational logic transformation and circuit optimization. We demonstrate that with the application of logic transformation using rewiring, we can further reduce the number of flip-flops on an interconnect delay retiming (18.7%). In addition, as a good by-result, due to the reduction on flip-flops, the power consumption estimated by Power Compiler gives a considerable reduction (8.26%) on the total dynamic power of the circuit.

This thesis is organized as follows. Chapter 2 introduces the rewiring backgrounds and algorithms. Chapter 3 reviews previous work in retiming, including Min-clock period retiming, Min-area retiming, retiming for low power, and interconnect retiming. Chapter 4 presents the rewired retiming optimization technique for flip-flop reduction. Chapter 5 analyzes the power reduction due to the use of the optimization scheme. Chapter 6 gives the final conclusion.

☐ **End of chapter.**

Chapter 2

Rewiring Background

Rewiring, originally proposed in [13] and [14], is a powerful technique for combinational optimization. Rewiring can be viewed as a procedure of logic transformation on the combinational part of the circuit. It transforms the circuit through replacing certain wires by adding some extra wires to the circuit, while maintaining the logic function of the circuit unchanged. The wires being removed are called target wires (TWs), while the extra wires added are called alternative wires (AWs). Guided by a suitable cost function, the appropriate target wires and alternative wires can be selected to achieve different optimization objectives, including logic minimization [13] [15], post layout timing optimization [14], technology mapping [16][17], FPGA routing [18][19] and circuit partitioning [20]. From all the previous works, we can learn that rewiring algorithms provide flexible and powerful logic transformation which can be used to optimized circuits' performance for different goals. This strongly motivates us to apply rewiring to improve retiming

An example of rewiring is shown in Fig.2.1.

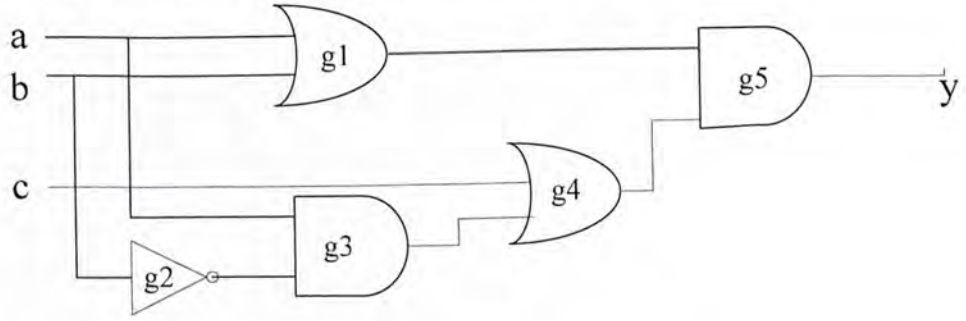


Fig.2.1 (a) Original Circuit

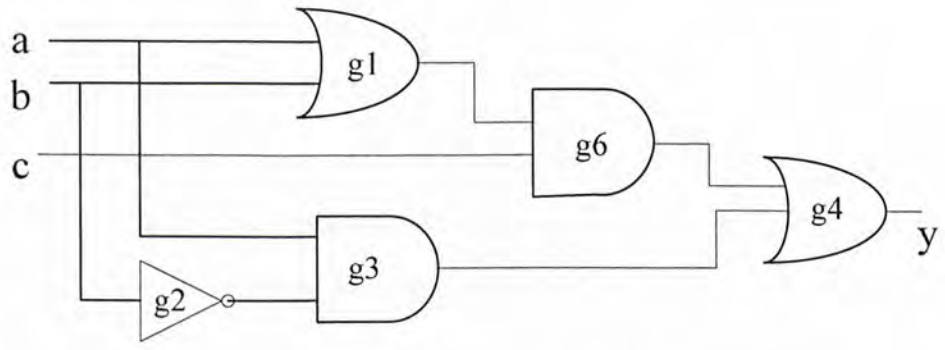


Fig.2.1 (b) Rewired Circuit

The original circuit is shown in Fig.2.1(a). Suppose we want to remove the wire g1→g5(red line, TW), the rewiring is done as the follows:

1. add a gate g6 to connect the output of g1 and c to the input of g4. (by doing this, actually a wire is added from g1 to g4).
2. remove the wire g1→g5
3. the gate g5 has only one input left and becomes removable

Finally, the rewired circuit is shown in Fig.2.1(b). The function $y = (a+b)c + a\bar{b} = ac + bc + a\bar{b}$ remains the same as Fig.2.1.(a).

Over the years, a lot of effort has been made in developing rewiring algorithms. There are now existing three main rewiring algorithms, namely the Automated Test Pattern Generation (ATPG)-based, graph-based, and Set of Pairs of Functions to be Distinguished (SPFD)-based algorithms. In our work the ATPG-based rewiring is adopted, and thus it will be described in detail in the following sections with brief introduction of graph-based rewiring

.

2.1 REWIRE

The most commonly used rewiring technique is Automatic Test Pattern Generation (ATPG) based. It converts the problem of finding target-alternative wire pairs into a problem of seeking undetectable stuck-at faults where ATPG technique is applied. The basic idea of the ATPG-based rewiring technique is to add a redundant wire/gate to make other wires/gates redundant and removable. Redundancy inside a circuit means that the logic value of a connection or a component has no effects on the circuit outputs.

REWIRE is an ATPG-based rewiring algorithm which utilizes the undetectable stuck-at-fault inside the circuit to find alternative wires and to make the target wire redundant. For a given target wire, the algorithm computes the Mandatory Assignments (MA) [13] for the test of the target wire. Mandatory Assignment is a set of values that assigned to the inside of the circuit such that the fault at the target wire can propagate to the primary output. If a set of consistent MA of the target wire does not exist, it means that the stuck-at-fault of the target wire is not detectable at the primary output, which means that the target wire is redundant and removable. If a set of MA for the target wire originally exists, we can try to add a redundant wire to the circuit, so that the MA becomes inconsistent and the target wire becomes undetectable and removable. Such a redundant wire is called alternative wire. After adding an alternative wire to the circuit, we have to check whether it is redundant as well, we can do this by a similar process, i.e. assigning the newly added alternative wire as a target wire and check its MA, if a consistent set of MA does not exist, it is a redundant wire.

The example in Fig. 2.2 shows how the rewiring works. g3 → g7 is a

candidate wire to make g1 → g5 redundant and removable. We test the stuck-at-1 fault at g1 → g5. First, we set {a = 0, b = 0} to make g1 = 0. To propagate the fault to the primary output o1, the side inputs to g5, g6, g7, and g9 should have non-controlling values, i.e. {e = 1, g4 = 0, g3 = 1, f = 1, g = 0}. g4 = 0 requires {g2 = 0, b = 0}. So g1 has to be 1 to make g3 = 1, but we have set g1 = 0. The conflict means that there is no test vector to detect this fault. Hence g1 → g5 is redundant and removable.

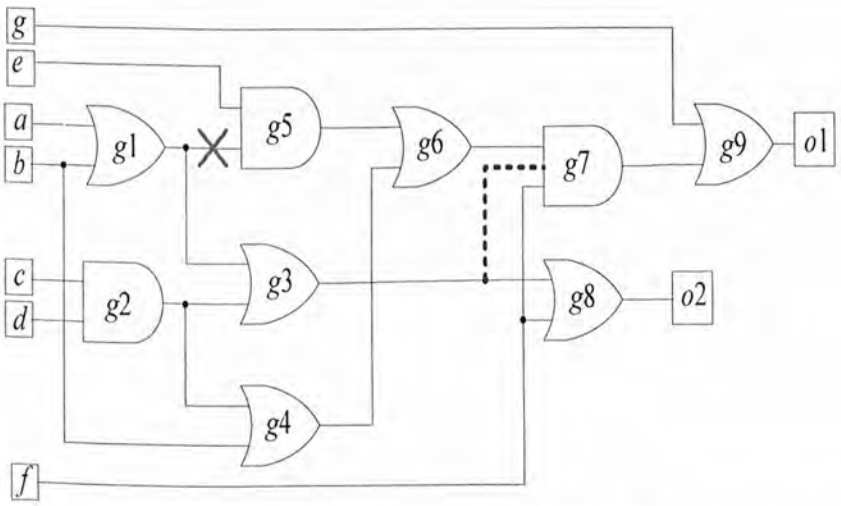


Fig.2.2 example of ATPG-based rewiring

2.2 GBAW

Graph-based rewiring uses graph pattern matching to find target wires and alternative wires. GBAW is a graph based rewiring algorithm [22]. It uses a set of graph configurations, which are called Patterns. Patterns are pre-defined graph representations of sub-circuits which contains alternative wires. Figure 2.3 is an example of a pattern. The target wire to the NOR gate can be replaced by the alternative wire to the AND or NAND gate. Figure 2.4 shows a circuit containing a pattern local 13 (the pattern is embraced with the dashed box).

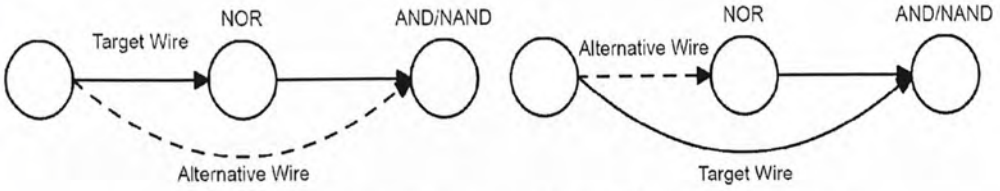


Figure 2.3 Example of GBAW Patterns

GBAW transforms the problem of finding target wires and alternative wires into matching patterns. It searches for alternative wires by performing pattern matching on the circuit from the library of patterns. GBAW is time efficient in finding alternative wires, on average it is around 150 times faster than REWIRE. However, the number of alternative wires found by GBAW is much less than that of REWIRE.

☐ **End of chapter.**

Chapter 3

Retiming

The previous chapter has introduced the background of the rewiring technique, with specific details in ATPG-based rewiring. Next, we are going to see how this technique can be incorporated into retiming, and before that, this chapter will review the retiming technique, including its original formulation, advancement, with particularly its application to power optimization and its drawbacks in today's technology. Retiming involves optimizing the cycle time (Min-Clock Period Retiming) [1] [2], reducing the area by minimizing the number of flip-flops (Min-area Retiming) [3] [4] etc. It has been also applied to different practical applications, such as logic synthesis [4][5], circuit partition[6][7], power reduction [8][9]and testability[10]. We are going to see why we need rewiring for the improvements of retiming, in terms of delay and placement estimation, traditional area reduction, as well as power reduction, which has captured much attention in today's technology.

3.1 Min-Clock Period Retiming

The earliest retiming formulation is given by Leiserson and Saxe [1]. It is a graph-based optimization technique to get the feasible minimal clock cycle by repositioning the flip-flops in a sequential circuit without violating the circuit's function and timing constraints.

In a classical retiming formulation, A sequential circuit C is represented by a directed graph $G(V, E, d, w)$. Each node v corresponds to a combinational gate and each directed edge $e(u, v)$ represents a connection from the output of

gate u to the input of gate v. For each combinational element v in the circuit. there is a propagation delay $d(v)$. The number of flip-flops are modeled as weight $w(u, v)$ on the edge $e(u, v)$. If there are n flip-flops on the edge e(u, v), $e(u,v)$ has a weight $w(u, v)$ = n.

An example is shown in Fig.3.1 (c). Consider a circuit composed of two comparators, one adder and two flip-flops. A comparator has a function

$\delta(x,a) = 1$ if $x = a$,

else $\delta(x,a) = 0$

an adder has a function:

$adder(x, y) = x + y$

A comparator has a delay of 3ns; an adder has a delay of 7ns. The primary input and output of the circuit is represented as a "Host" element with 0 delay. Originally there are two flip-flops at the wire from the "Host" to the first adder.

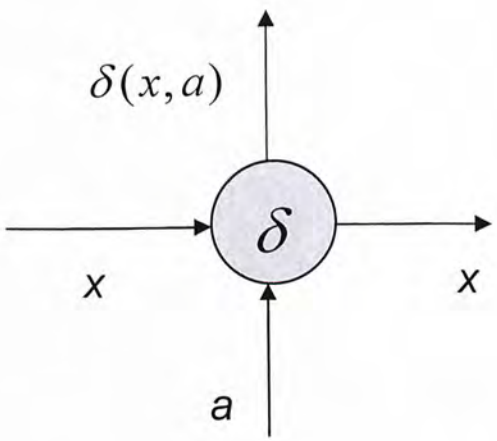


Fig. 3.1 (a) Comparator

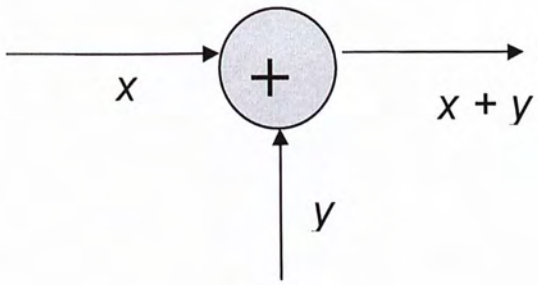


Fig.3.2 (b) Adder

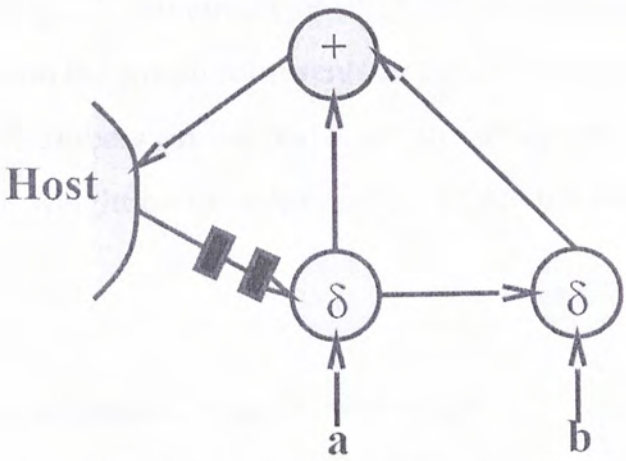


Fig. 3.1 (c) original circuit

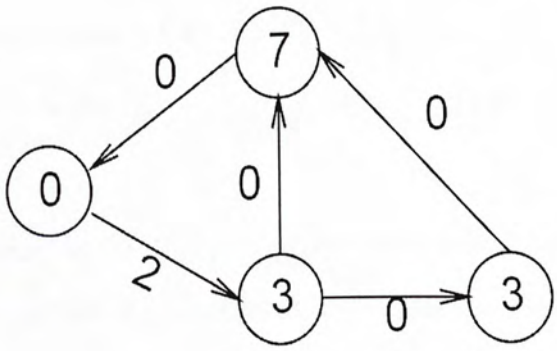


Fig. 3.2 Graph representation of the original circuit

r(v) = -1 r(v) = +1

Fig.3.3 Basic retiming operations

As shown in Fig.3.2, the circuit of Fig.3.1(c) is represented by a retime graph. Each node on the graph represents a combinational element (adder or comparator). The numbers on the nodes are the delays of the combinational elements *d(v)*. The weight on the edges (*w(u, v)*) are the number of flip-flops on the edges.

For a path *p*, from vertex v_0 to v_k, with edges e_0, $e_1 \ldots e_{k-1}$

$$d(p) = \sum_{0}^{k} d(v_i)$$

$$w(p) = \sum_{0}^{k-1} w(e_i)$$

The clock period is defined as:

$c = \max\{d(p)\}$ $(p: w(p)=0)$

As the clock period is the longest delay from one flip-flop to another. The original circuit has a delay of 13, which is calculated as 3+3+7 = 13, the sum of the longest path delay.

A retime value of integer type *r(v)* is defined for each node v to represent the flip-flop movements across the node, as shown in Fig.3.3. *r(v)* of a positive

value m stands that there will be m flip-flops moved from every output edges of v to every input edges of v. Similarly, a negative $r(v)$ value of –m stands for the opposite moving direction. The weight $w'(u,v)$ after retiming is $w'(u,v) = w(u, v) + r(v) - r(u)$

To represent the retiming operation of Fig.3.2, each of the nodes in Fig.3.2 has its retime value $r(v)$, which is shown in Fig. 3.4.

From Fig.3.4 we can see the two comparators have a -1 value, which means that 1 flip-flop is retimed from their input to their outputs. As a result, the graph after retiming is shown in Fig. 3.5. In Fig.3.5, the new clock period is reduced to 7, as the original critical path of 13 is broken by flip-flops.

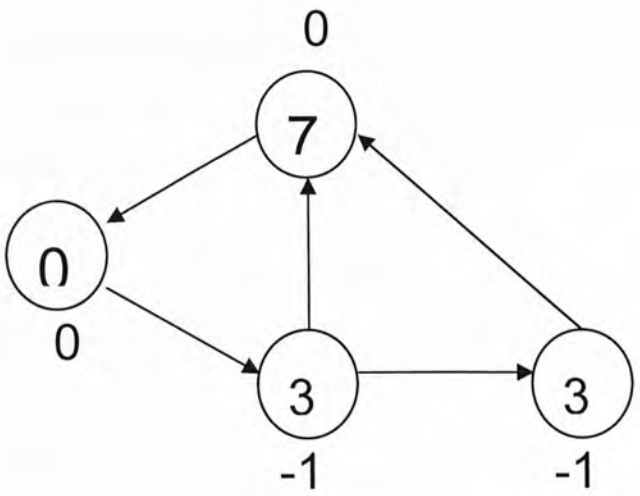


Fig.3.4 Retime value r(v)

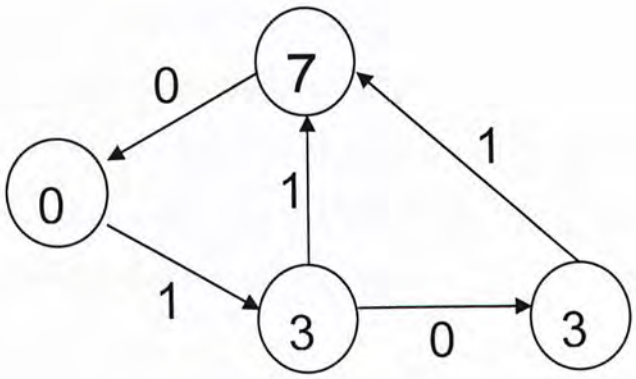


Fig.3.5 Graph representation of the retimed circuit

Therefore, classical retiming can be viewed as an integer value vertex labeling of the retime value on the graph such that by the specified adding and removing of flip-flops, the new graph has the minimal achievable clock period while the structure of the graph is unchanged.

In [2], the algorithm for minimizing the clock period of a circuit is based on two quantities defined as:

$W(u, v)= \min\{w(p): u \rightarrow v\}$

$D(u, v)= \max\{d(p): u \rightarrow v \text{ and } w(p)=W(u, v)\}$

W(u,v) is the minimum number of flip-flops on any path from vertex u to v, D(u, v) is the maximum total propagation delay on any critical path from u to v.

Based on these two quantities, [2] developed the following lemmas, which are the basic tool needed to solve to min-clock-period retiming problem.

For a given clock period c, the retimed circuit has a clock period $c' \leqq c$ if and only if:

(1) $r(u) - r(v) \leqq w(u, v)$ for every edge e(u,v) of G

(2) $r(u) - r(v) \leqq W(u, v) - 1$ for all vertices such that $D(u, v) > c$

The constraints on the unknowns *r(v)* are linear inequalities involving only differences of the unknowns, and thus they can be regarded as an instance of a linear programming problem. Using the Bellman-Ford algorithm to test whether a given clock period c is feasible takes only $O(|V|^3)$ for the $O(|V|^2)$ inequalities. The algorithm to solve the minimum clock period is summarized below:

1. Compute all W(u, v) and D(u, v) for all u, v $\in$ V such that u is connected to v
2. Sort the elements in the range of D
3. Binary search among D(u, v) for the minimum achievable clock period. Use the Bellman-Ford algorithm to test whether the clock period is achievable.
4. For the minimum clock period found in step 3, use the values for the *r(v)* found as the optimal retiming solution.

Yet, [2] proposed a more efficient algorithm to determine whether a given clock period is feasible, by iteratively relaxing the constraints for each tentative retiming. This efficient algorithm takes only $O(|V||E|)$ time, which is a significant improvement of the original $O(|V|^3)$. Combined with the Binary search, the author gave a $O(|V||E| \lg|V|)$ algorithm which can solve the min-clock-period problem.

The retiming problem can also be formulated as a Mixed-Integer Linear Programming (MILP) problem.

For a graph *G(V, E, d, w)*, there exist a legal retiming such that $c' \leqq c$ if the following constraints are satisfied:

For every vertex v, if there exist a real value s(v) and an integer value r(v) such that

$-s(v) \leqq -d(v)$ for every vertex $v \in V$

$s(v) \leqq c$ for every vertex $v \in V$

$r(u) - r(v) \leqq w(u, v)$ for every edge $e \in E$

$s(u) - s(v) \leqq -d(v)$ for every edge wherever $r(u) - r(v) = w(u, v)$

Based on the above constraints, we can solve the retiming problem using mathematical programming approach. An algorithm is derived from this MILP basis. The basic steps of the algorithm are shown as follows:

1. Compute all W(u, v) and D(u, v) for all $u, v \in V$ such that u is connected to v
2. Sort the elements in the range of D
3. Binary search among D(u, v) for the minimum achievable clock period. Use the MILP to test whether the clock period is achievable.
4. For the minimum clock period found in step 3, use the values for the r(v) found as the optimal retiming solution.

Step 1 runs in $O(|V||E| + |V|^2 \lg|V|)$ if the Fibonacci heap data structure by Fredman and Tarjan [23] is used for the all-pairs shortest paths algorithm [24]. Step 2 runs at $O(|V|^2 \lg|V|)$ for the $O(|V|^2)$ elements. Every iteration in the binary search of step 3 requires solving a MILP with $|V|$ integer variables, $|V|$ real variables, and $2|V|+2|E|$ inequalities. The total time of step 3 is thus $O(|V||E| \lg|V| + |V|^2 \lg^2|V|)$. Therefore, the total runtime of the algorithm is $O(|V||E| \lg|V| + |V|^2 \lg^2|V|)$.

Though the theoretical formulation by Leiserson and Saxe to solve the retiming problems have polynomial complexity, the implementations of the algorithm is not considered and lead to high complexity for large circuit with

more than 500 combinational cells. Shenoy et al. [25] addresses the implementation issues required to exploit the sparsity of circuit graphs to allow min-period retiming as well as constrained min-area retiming to be applied to circuits with as many as 10,000 combinational cells. Recently, Zhou [26] proposed an efficient incremental algorithm for min-period retiming which iteratively moves FFs to decrease the clock period while guarantees to find the optimal solution in a short time.

3.2 Min-Area Retiming

There can be more than one solution to reposition the flip flops, while achieving the same optimal clock period. Min-area retiming is therefore targeted at minimizing the number of flip-flops, such that the total area of the circuit can be minimized. Min-area retiming minimizes the FF area under a given clock period, thus could be used to minimize the FF area even under the minimum clock period.

Based on the classical Min-clock period retiming, Min-area retiming is a technique which further solve for a solution with minimum total weights. The basic formulation of the problem is the same, a sequential circuit C is represented by a directed graph G(V, E), each combinational element is represented by a node v, and each connection is represented by and edge e(u, v). The flip flops on the edges are denoted by the weight on the edges. $r(v)$ is also used to represent the number of flip-flops that are retimed backward across the node v. The weight $w'(u,v)$ after retiming is $w'(u,v) = w(u, v) + r(v) - r(u)$. The min-area retiming objective is to minimize the total number of

flip-flops, i.e., $\Sigma w'$ is min. Using (3), this leads to the following optimization problem:

$$r(v) \cdot (|FI(v)| - |FO(v)|) \rightarrow \text{minimum}$$

where FI(v) and FO(v) represent the set of fanin and fanout gates of gate v.

The early Min-area retiming basically follows the Min-clock period retiming idea of Leiserson and Saxe. Later, a lot of work has been done to study the Min-area retiming algorithm. Shenoy et al. [25] were among the first to consider a practical implementation of the min-area retiming algorithm. They proposed techniques to prune away redundant constraints, which lead to higher efficiency in time and space usage to solve the problem. Singh et al. [27] also proposed to incrementally move FFs in the circuit to overcome the expenses of previous approaches to min-area retiming, however, their approach is a heuristic which only looks for better moves and may end up with sub-optimal solution. Recently, Jia Wang et.al. [28] proposed an efficient algorithm iMinArea which can solve the Min-area retiming problem incrementally and optimally, and the runtime is proved to be much faster than all existing approaches.

3.3 Retiming for Low Power

Power dissipation has been receiving more and more concerns today, a lot of optimization techniques have been applied to this area, and retiming is one of

them. As retiming is a technique which can lead to strong effect on the critical path delay, number and position of the flip-flops, etc., which are all significant factors of a circuit's power dissipation.

Previous works in this field mainly consider reducing the switching activities of the circuit. As switching activities are largely due to the position of flip-flops. Switching activity is a significant cause of power dissipation in combinational and sequential circuits. Logic synthesis has been used to improve the power dissipation of a circuit. In [29], a new cost function for combinational logic synthesis targeting low power was presented. A method to speed up a sequential circuit using retiming and lowering power dissipation (and increasing delay) by scaling down the power supply voltage was presented in [30]. There are also methods that lowered power dissipation by restructuring the combinational logic. In [31], the authors investigated the application of retiming to modify the switching activities on internal wires of a circuit and demonstrate the impact of these techniques on average power dissipation.

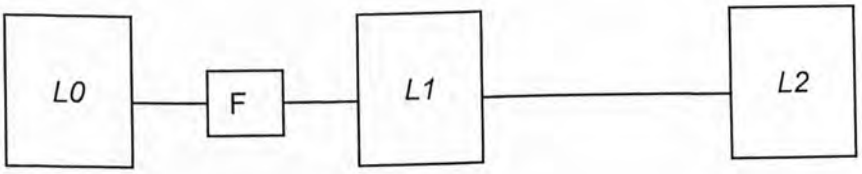


Fig.3.6 (a) Switching activity and power dissipation

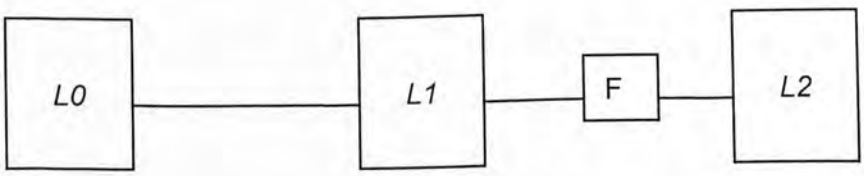


Fig.3.6 (b) Switching activity and power dissipation after retiming

The power dissipation of a gate g is proportional to the output switching activity E_g times the load capacitance of its fanout C, i.e. E_g C. The effect of retiming on switching activity is demonstrated in Fig.3.6. L0, L1 and L2 and combinational elements of the circuit, the power dissipation of (a) is $E_0C_F + E_FC_1 + E_1C_2$, the power dissipation of (b) is $E_0C_1 + E_1'C_F + E_F'C_2$, the power dissipation of (a) and (b) is different, generally E_F is less than E_1', because the output of FF changes at most once every clock period, by the same token, E_F' is less than E_1. Therefore, changing the position of FF can have an influence on the total power dissipation. A general idea is to retime the FFs to the wires with high switching activities.

In [31], the nodes are selected by a cost function, based on the switching activity at their output and the number of fanouts.

$Weight(i) = P(i) * (ni(i) + no(i))$, where $P(i)$ is the power estimated by switching activity and load capacitance. *ni* and no are the numbers of fanins and fanouts.

The switching activity is estimated by the probability that a transition propagates through its transitive fanout. Retiming is then executed based on the cost function, trying to place FFs at the fanouts of high switching gates.

In [32], the authors conduct an empirical study of power dissipation when different levels of pipelining are added to a circuit. They discovered that by adding more levels of pipelining to a sequential circuit, more gates are likely to have balanced paths and so there is a power reduction.

In [33], the authors propose an algorithm for pipeline insertion in a sequential circuit. Since this is the only known algorithm for low power retiming, it warrants close inspection. The first step of their algorithm is to perform a detailed power estimation of all the gates in the network. Each gate is then weighted by three factors: (1) The amount of glitching activity at the gate (the dierence between the switching activity under a general delay model and the zero delay model) (2) The probability that a transition at a gate results in transitions in the gate's transitive fanout (at most two levels ahead) (3) The number of fanouts of the gate. Latches are then placed in a greedy fashion in front of the gates based upon their weights. In an attempt to reduce the total latch count, the nal step of their algorithm is to forward retime gates which are not on the same paths as the gates that were greedily retimed. The approach in [33] has three limitations: (1) Their algorithm requires a costly simulation for each level of pipelining that is inserted. (2) Placing a latch in front of a gate may prevent the propagation of a glitch at the cost of the generation of a new glitch. (3) Their algorithm may introduce large latch counts.

As reported in [34], in reality the power consumed by a single latch is considerably greater than the power consumed by a single transition. It is very likely that the power dissipation can be cut down further if the number of FFs can be reduced.

So far, the effect of using retiming to reduce power is not obvious applicable, simulating the switching activity and combined into retiming is a tedious task, yet, it is obvious that reducing the number of FFs is critical for reducing the total power dissipation and beneficial for clock tree generation.

3.4 Retiming with Interconnect Delay

As the VLSI process technology scales down to a deep submicron era, a major drawback of the traditional retiming approaches (both Min-period retiming and Min-area retiming) is that in all the retiming algorithms above, the interconnect delay (or wire delay) is not considered into a path delay. However, in today's technology, interconnect delay has become a predominant factor which can be much more than the delay of a gate. Hence, not only gate delay should be considered but more importantly, the interconnect delay. The traditional formulation which ignores the interconnect delay is therefore not accurate enough regarding delay estimation.

The work of [11] gives us a new retiming formulation including both gate delay and interconnect delay. Based on experiments conducted in [11], the interconnect delay is nearly proportional to the interconnect wire length, therefore, the formulation assumes that the interconnect delay is proportional to the wire length. The wire length can be estimated by the Manhattan distance between the connected cells in the given placement.

Similar to the Min-clock period retiming, a circuit C is represented by a corresponding graph $G(V, E, w, d)$. Each node v corresponds to a combinational gate and each directed edge $e(u, v)$ represents a connection from the output of gate u to the input of gate v. For each combinational element v in the circuit, there is a propagation delay $d(v)$. The number of flip-flops are modeled as weight $w(u, v)$ on the edge $e(u, v)$. Besides, the interconnect delay on an edge $e(u, v)$ is represented by $d(u, v)$, which is estimated from the Manhattan distance between the corresponding cells in the given placement.

It is assumed that G is strongly connected. If not, [11] stated that a source node s can be added to the circuit and connect it to all primary inputs, and a target node t can be added as well and connect all primary outputs to it, and connect t to s. Then the resulting graph is strongly connected. The delay of s, t and all the added edges are set to zero; the number of registers on the edge from t to s is set to one and that on the other added edges set to zero. Then a retiming solution of the modified graph will also be a valid retiming solution of the original graph.

One method [11] for the problem is extended from the Mixed Integer Linear Programming method introduced in 3.1. This method can guarantee the optimal solution. In the original Min-clock period retiming formulation, only gate delay is considered, however, the MILP mathematical programming approach can be extended to solve the problem with both gate and interconnect delay optimally by modifying some of the constraint formulation.

A variable $a(v)$ is defined for each node v to represent the maximum arrival time of v. As shown in Fig.3.7, $a(v)$ is the time given for a signal to travel from a flip-flop fanin to a node v to the output of the node v. If there are multiple flip-flops, $a(v)$ is given for the one with longest path delay to v.

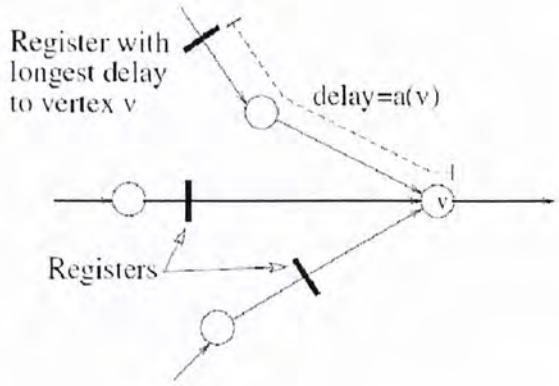


Fig.3.7 example of the arrival time [11]

The MILP is formulated as follows:

$d(v) \leq a(v) \quad \forall v \epsilon V \quad (1)$

$a(v) \leq T \qquad \forall v \epsilon V \quad (2)$

$r(v) + w(u,v) - r(u) \geq 0 \quad \forall e \epsilon E \qquad (3)$

$a(u) + d(u,v) + d(v) - T(r(v) + w(u,v) - r(u)) \leq a(v) \quad \forall e \epsilon E \qquad (4)$

Constraints (1) and (2) are obvious. (3) is the number of flip-flops after retiming on the edge e(u, v). For a legal retiming, the weight should never be negative. (4) is a new constraint considering the interconnect delay. (4) can be interpreted as the arrival time of a signal from a flip-flop to a node v must be larger than or equal to the delay of the path from the flip-flop to the node v.

If we define a real variable R(v) as R(v) = a(v)/T + r(v), the constraints (1) to (4) can be transformed to :

$$R(v) - r(v) \geq \frac{d(v)}{T} \qquad \forall v \in V \qquad (5)$$

$$R(v) - r(v) \leq 1 \qquad \forall v \in V \qquad (6)$$

$$r(u) - r(v) \leq w(u,v) \qquad \forall e(u,v) \in E \qquad (7)$$

$$R(v) - R(u) \geq \frac{d(u,v)}{T} + \frac{d(v)}{T} - w(u,v) \qquad \forall e(u,v) \in E \qquad (8)$$

There are |V| real variables R(v), |V| integer variables r(v), and 2|V|+2|E| constraints. This problem can be solved in O(|V||E| lg|V| +

$|V|^2 \lg^2 |V|$)if the Fibonacci heap [23] is used as the data structure. If these set of constraints can be solved, which means that the clock period T and the variable r(v) is decided, we can determine the exact position of the flip-flops on the wire. Besides, [11] also proposed a fast near optimal approach (0.13% more than the optimal clock) for the interconnect delay retiming, by first reducing the problem to the single source longest path problem.

□ **End of chapter.**

Chapter 4

Rewired Retiming for Flip-flop Reduction

The last chapter has reviewed the previous works of retiming, from the most classical Min-period retiming, to new approaches in Min-area retiming, interconnect delay retiming and the application to power reduction. Throughout the whole picture, we can see retiming is a topic which closely follows the step of technology advancement and has attracted much attention due to its potential for area, timing and power optimization. However, there is still plenty of space to explore in this area as previous optimization related to retiming seldom consider interconnect delay, and there is a need to develop optimization tools for area, timing and power based on this more accurate delay model. Considering with logic synthesis is a good opportunity for us to achieve further improvement on this subject. This chapter is going to talk about the rewired retiming technique in detail as proposed by the author.

4.1 Motivation and Problem Formulation

As demonstrated in the previous chapter, in today's deep submicron technology, traditional retiming algorithm which ignores the interconnect delay is no longer accurate enough, because the interconnect delay can be much dominating and larger than the logic/gate delay. However, most existing retiming related applications and their optimization don't consider a more accurate delay model with interconnect delay, until this problem is addressed in [5], where the retiming problem is re-formulated to include both gate and interconnect delays, in which the interconnect delay is assumed to

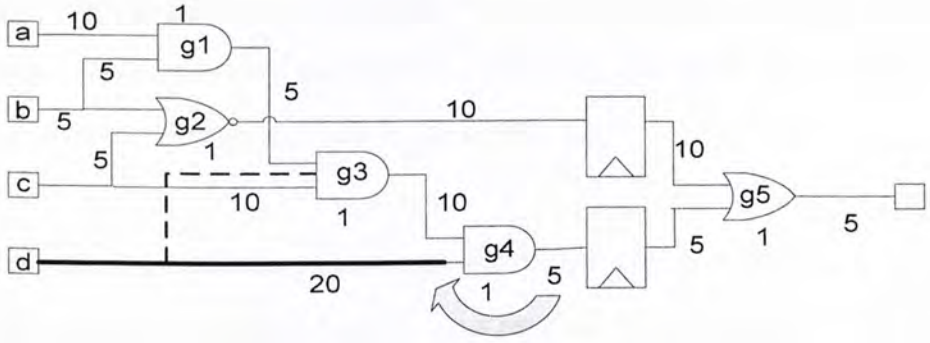
be proportional to the wire length.

Nevertheless, as demonstrated in [12], the optimal clock period gained from retiming may not be feasible after the circuit is really placed. As a large number of flip-flops are relocated and the flip-flop number will usually increase after retiming, a traditionally retimed optimal clock period might not be close to reality in a legalized placement. Moreover, a larger amount of power consumption can be introduced due to the increased flip-flops. Therefore, besides delay improvement, it's also important to cut down the retiming-induced flip-flops for both area and power reductions, which is a problem not addressed in [11].

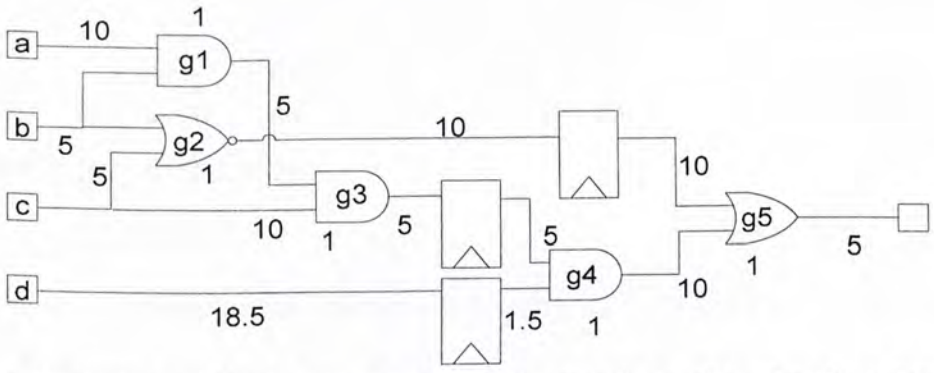
Therefore, we try to integrate the interconnect based retiming with rewiring to achieve better and more accurate circuit optimizations. Our experiment demonstrate that with the application of logic transformation using rewiring, we can further reduce the number of flip-flops based on the models introduced in [11].

The example in Fig. 4.1 shows how the rewiring helps in reducing the number of flip-flops through logic transformation. Assuming that each logic gate has a delay of 1 unit, while the interconnect delay is proportional to the (shortest) Manhattan distance between the placed logic elements. In Fig. 1 (a), the initial circuit has a clock period of 33 with two FFs. A conventional retiming would produce a solution (Fig. 1(b)) with a reduced clock period of 22 however with an increased FF count of three. As a new FF is added to the wire d → g4, we would try to see if we can remove this wire. Using rewiring technique, we know that d → g3 is an alternative wire for the target wire d→g4. After a rewiring transformation, the retiming solution uses only two

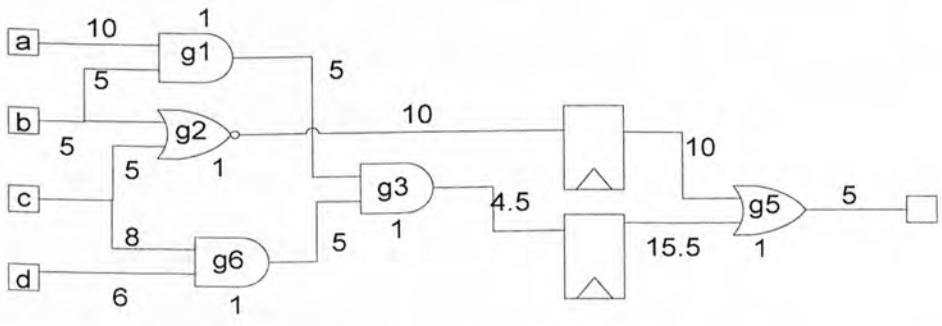
FFs, with clock period of 21.5. The rewired circuit after retiming is shown in (c).



(a) The initial circuit: clock period is 33. A FF is going to move across g4 guided by retiming. By rewiring implication, d→g3 is an alternative wire to replace d→g4



(b) Retiming without rewiring: the clock period is reduced to 22, while FFs are increased.



(c) Retiming after rewiring: after replacing d→g4 by d→g3, the number of FF is reduced compared to (b), the retimed clock period is 21.5.

Fig. 4.1: Flip-flop reduction using retiming and rewiring

The rewired retiming problem can be formulated as follows:

Given a sequential circuit C and its placement P, based on an interconnect delay model, we can compute its initial retiming solution: the minimum clock period T and the number of FFs n after retiming.

Applying several iterations on rewiring transformations, we want to find a functionally equivalent circuit C′ and its corresponding placement P′, such that after retiming for C′ based on P′, the number of FF n′ is considerably cut down from n, while the clock period T′ is not worse than T.

4.2 Retiming Indication

In order to achieve better delay estimation and make our technique more practical, here we adopt the delay model proposed in [11], in which the interconnect delays are assumed to be proportional to wire lengths, i.e. interconnect delays are estimated from the shortest Manhattan distance between the connected cells in the given placement. Gate delay is assumed to be 1 unit of the wire delay. Though there can be many other different delay models, we assume that a general flow responsive to a reasonable cost function can also be effective to others. We implement the Mixed Integer Linear Programming approach to solve the interconnect delay retiming problem.

As discussed in chapter 3, a graph G(V, E, w, d), is used to represent the sequential circuit. Each node v corresponds to a combinational gate and each directed edge $e(u, v)$ represents a connection from the output of gate u to the

input of gate *v*. For each *v* in the circuit, there is a propagation delay *d(v)*. The number of flip-flops are modeled as the weight *w(u, v)* on the edge *e(u, v)*. *w(u, v)* is a non-negative integer. The interconnect delay of edge *e(u, v)* without any FF is represented by *d(u, v)*.

A retime value of integer type *r(v)* is defined for each node v to represent the flip-flop movements across the node, as shown in Fig.4.2. An *r(v)* of a positive value m stands that there will be m flip-flops moved from every output edges of v to every input edges of v. Similarly, a negative *r(v)* value of -m stands for the opposite moving direction. Beware that in some retiming works involving placement, multiple fanouts can share the same FF when the circuit is placed, whose results can reduce FFs needed a bit but are highly dependent on the actual placement tool applied. In order to see a fairer comparison on the FF reductions produced by our flow, in this paper we assume that in this case each fanout should take one FF, i.e. we disallow the possibility of FF sharing on fanouts. When the circuit is finally placed, we place the same number of FFs as produced by the retiming procedure.

Fig.4.2 retiming flip-flops forward and backward

Finally, based on the mixed integer linear programming (MILP) approach (constraints (5) - (8) in chapter 3.4, by introducing a variable R(v) = a(v)/T + r(v) for each node v, a set of constraints are formulated as follows:

$$R(v) - r(v) \geq \frac{d(v)}{T} \qquad \forall v \in V \qquad (5)$$

$$R(v) - r(v) \leq 1 \qquad \forall v \in V \qquad (6)$$

$$r(u) - r(v) \leq w(u,v) \qquad \forall e(u,v) \in E \qquad (7)$$

$$R(v) - R(u) \geq \frac{d(u,v)}{T} + \frac{d(v)}{T} - w(u,v) \qquad \forall e(u,v) \in E \qquad (8)$$

This problem can be solved in $O(|V||E|\lg|V| + |V|^2\lg^2|V|)$. By solving $r(v)$ for each node, the weight of each edge after retiming is decided. Therefore, in our iterative optimization process, the variable $r(v)$ is used to predict the movement of FFs. Once we find a retiming solution, the optimal clock period T, $r(v)$ of each node is produced as a side effect, and we make use of this value to indicate the movement of FFs. Once this information is obtained, some heuristics can be developed to guide the rewiring optimization process.

4.3 Target Wire Selection

Given a target wire, rewiring is able to find a list of alternative wires which when added to the circuit, can make the target wire redundant and thus removable. For a sequential circuit, the logic implication [13] is done only within the combinational sub-network bounded by FFs. Therefore, the alternative wires are within the same combinational sub-network of the target

wire. A wire with FFs is treated as a primary input or output of the sub-network and cannot be a target wire.

By selecting TWs which have an effect on the movement of the FFs, the FFs produced by retiming could be different. Here, a few heuristics are developed based on some typical conditions observed to guide the selection of TWs.

Consider an edge *e(u, v)* (*u* is a fanin to *v*) as shown in Fig.4.3, where *r(v)* denotes the number of FFs moved from *v*'s fanouts to its fan-ins, and *r(u)* denotes the number of FFs moved from *u*'s fanouts to its fanins. There are two conditions which are effective in FF reduction:

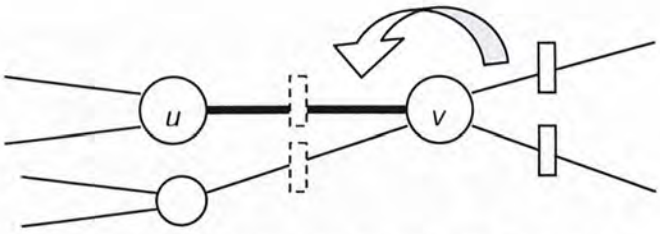


Fig.4.3 (a) Condition 1: *e(u, v)* is selected

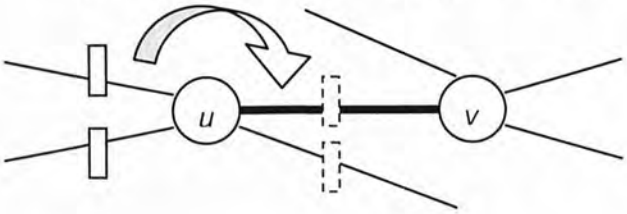


Fig.4.3 (b) Condition 2: *e(u, v)* is selected

Condition 1: $r(v) - r(u) > 0$ ($r(v) > 0$), there is FF moved from the fanout(s) of v to this edge and the other input edge(s) of v. Therefore, by replacing *e(u, v)* by an alternative wire is likely to reduce FF after retiming, *e(u, v)* is selected as a target wire.

Condition 2: $r(v) - r(u) > 0$ ($r(u) < 0$), there is FF moved from the fanin edges of u to the fanout(s) of u. If u has more than 1 fanout edges, replacing $e(u, v)$ by an alternative wire is likely to reduce FF after retiming, $e(u, v)$ is selected as a target wire.

On the above two conditions, FFs in this edge will have a net increase after retiming as they will be moved in either from the fanout(s) of v and/or fanin(s) of u. Therefore, selecting $e(u, v)$ as a target wire and replacing it by its alternative wire (before applying retiming) is likely to reduce the number of FFs of the retiming result.

Consider other conditions, if $r(v)-r(u)<0$, $e(u, v)$ must already have FF before retiming (because the weight of an edge can never be negative, which is guaranteed by the retiming constraints). A wire with FFs is treated as a primary input or output during logic implication [13] and cannot be a target wire. If $r(v) - r(u) = 0$, $e(u, v)$ is not selected as target wire, since it has no change to the number of FF on this edge.

In this way, retiming gives us a hint to direct the rewiring transformations. Indicated by the retime value $r(v)$, the wires that have a direct effect on the number of FFs are identified as target wires. By replacing them with their alternative wires, the FFs after retiming are very likely to be reduced. The TW-AW list should be updated whenever a rewiring transformation is accepted.

4.4 Incremental Placement Update

After rewiring, we must give an updated placement to support the next iteration of retiming evaluation. To avoid the undesirable influence of the randomness rooted from re-placement for the whole circuit, we estimate the position of the new cells and update the placement incrementally after rewiring. Our approach is applicable to both 2-input gates and multiple input gates, for simplicity we use 2-input gates in our experiment. The update is done as follows:

When a target wire is removed from the network, a corresponding sink cell having only one input left is removable. In this case, we drop the cell from the placement, and the positions of all other cells remain unchanged. When an alternative wire is added to the circuit, a new gate is added between the alternative wire's sink node and its fanout(s), as shown in Fig.4.4

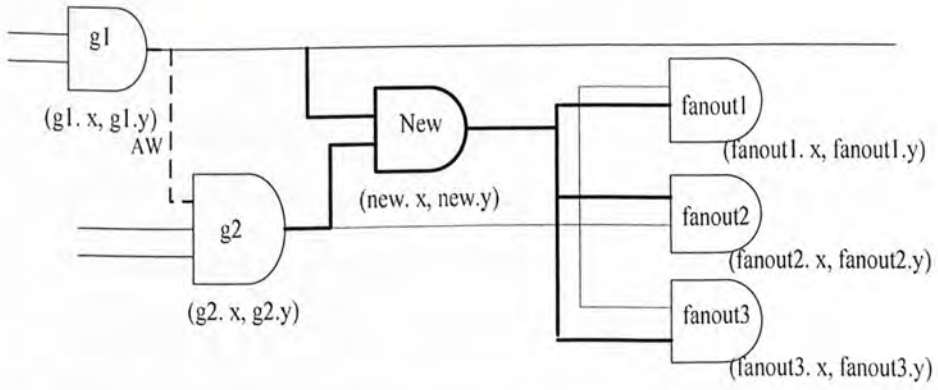


Fig.4.4 Example of placement estimation for adding new wire

In Fig. 4.4, g1 and g2 are the source and sink of the alternative wire AW. fanout1, fanout2 and fanout3 are the fanouts of g2. In order to add AW, a

new cell New is added to the circuit, connecting g1, g2 and the fanouts of g2 (bold lines). After adding the new cell and the new net (bold lines), the net g2→fanout1, g2→fanout2 and g2→fanout3 are removed.

Assume that the positions of all other cells remain the same, to find an optimal position for the new gate, such that the total Manhattan distance among the g1, g2, New, fanout1, fanout2 and fanout3 is minimum, is a difficult problem (similar to finding a Steiner Minimum Tree, which is NP-hard). To avoid the complexity, we adopt an arithmetic average position to estimate the New gate's position.

$$new.x = (g1.x + g2.x + fanout1.x + fanout2.x + fanout3.x) / 5$$
$$new.y = (g1.y + g2.y + fanout1.y + fanout2.y + fanout3.y) / 5$$

In general, the position (x, y) for the new cell is estimated as:

$$new.x = (AWsrc.x + AWdst.x + \sum fanouts.x) / N$$
$$new.y = (AWsrc.y + AWdst.y + \sum fanouts.y) / N$$

where AWsrc and AWdst are the source and sink of the alternative wire, N is the total number of fanouts plus 2 (AWsrc and AWdst).

As a rewiring step usually only injects a small perturbation on a local area, based on our experiments this calculated position provides a reasonable estimation close to the real placement change, thus can be used for the next retiming and rewiring iterations. After all rewiring iterations, the rewired circuit is retimed and placed again, and the clock period is calculated from the final placement.

4.5 Optimization Flow

An optimization scheme combining ATPG based rewiring and retiming is developed. As shown in Fig. 4.5, the optimization scheme consists of the following basic steps:

(1) Initial retiming: select the initial TW-AW list based on the initial retiming indication.

(2) Perform rewiring using a TW-AW pair from the TW-AW list.

(3) Incrementally update the placement according to the rewiring transformation.

(4) Retiming evaluation: evaluate whether the rewiring is beneficial, if yes, go to (5); if not, go to (6)

(5) If the number of accepted transformations is within N (a predefined number of total iterations), accept the rewiring transformation, and update the TW-AW list, go to (2) to perform the next iteration; if N is reached, go to (8)

(6) Discard the change, if there is still TW-AW pair in the TW-AW list, go back to (2) with next TW-AW pair; if TW-AW list is empty, go to (7)

(7) If the number of perturbations performed is within a predefined M, perturb the circuit by randomly performing a rewiring transformation, update the TW-AW list and go back to (2); if M is reached, go to (8)

(8) Final retiming and placement, get the clock period.

The initial retiming and retiming evaluation both solve the MILP for T and r(v) to predict the FF movements and resulting number of FFs, but the FFs are not actually moved.

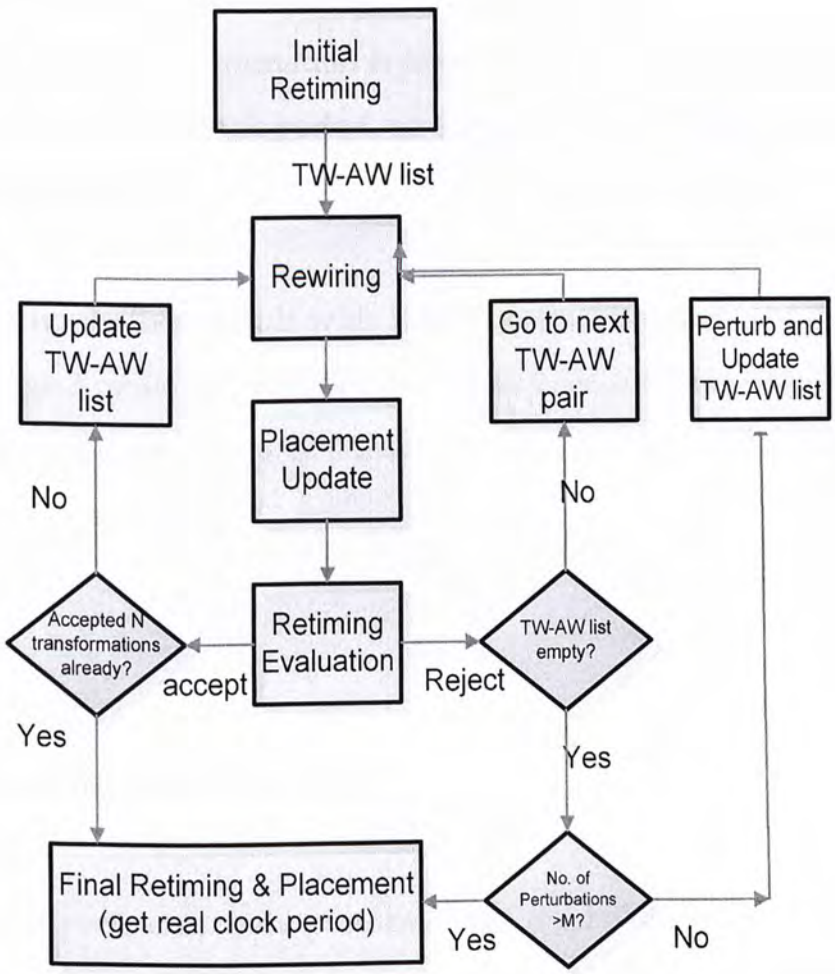


Fig.4.5 Overall optimization flow

To make the scheme more efficient, N iterations are divided into three stages (each stage has N/3 iterations) with a different definition of whether a rewiring is beneficial or not.

Stage 1: the transformation is beneficial if the clock period is less than or equal to the initial clock period, and the number of FFs is less than the retiming solution with the least FFs so far. This is a greedy stage

Stage 2: the transformation is beneficial if the clock period is within 1.25 times of initial clock period, and the number of FFs is less than the retiming solution with the least FFs so far.

Stage 3: the transformation is beneficial if the clock period is less than or equal to the initial clock period, and the number of FFs is less than the best result of stage 1.

Finally, the best result with least FFs and no larger clock period comes from stage 3 or stage 1 will be taken. The final circuit is retimed and placed, and the clock period is obtained from placement under the same delay model.

4.6 Experimental Results

The experiments were performed for the ISCAS89 benchmark suite. The benchmark circuits were first mapped with a library consisting of common logic gates (Invertors, AND gates, OR gates, NAND gates, NOR gates). The maximum number of fanins for a gate is 2. The initial placements of the circuits were generated by the Capo 10.5 placer. The program was implemented in C language, the retiming MILP constraints are solved by ILog Solver 6.0. Experiments were run on a Sun Blade 2500 (2 x 1.6GHz US-IIIi) 2GB RAM Solaris 8 machine.

The number of iterations for each stage is set to be 10 (totally N=30); for stage 1 and 2, the predefined number of perturbation is 5, for stage 3, the perturbation is performed at most 15 times (totally M=25).

Table 4.1 shows the reduction of FFs of our approach compared to initial (original) pure retiming [5]. On average, our optimization scheme can achieve a 18.7% cut down on the total number of FFs. Basically, rewiring can be done in polynomial time, and experiments show that averagely 98.2% of the time is taken by solving the retiming constraints, where the simplex algorithm is used. This can be improved by more effective implementation of retiming, however, it does not affect our optimization scheme.

The final clock period is calculated on a real placement based on the same delay estimation model. To examine the effect on the clock period, we use the average and the best clock period obtained from 10 times of final placements.

Table 4.2 shows that from 10 placement results, a pure retiming has an average of 7.97% cut down on the initial clock period, while combining with rewiring, we also come up with an average of 7.96%. Note that the clock period reduction could be negative in this new flow because the new FF topology after the rewired retiming could be much different from that of the original pure retiming.

Table 4.3 shows the best clock period from 10 times of final placements. Compared to the best reduction of 18.01%, our optimization scheme also has a reduction of 17.74%.

	# of FF in pure retiming	# of FF in Rewired retiming	FF Reduction (%)	Runtime (minutes)	%of runtime spent in retiming
S298	133	119	10.5	2	99.1
S344	59	53	10.2	5	99.3
S349	57	54	5.26	6	99.5
S382	81	77	4.94	22	99.2
S444	93	75	19.4	5	99.4
S510	117	103	12	4	98.5
S526	181	166	8.29	107	99.6
S820	207	82	60.4	53	98.1
S832	67	17	74.6	76	96.2
S953	96	83	13.5	9	98.4
S1238	32	31	3.13	27	97.3
S1488	484	387	20	19	97.1
S1494	417	415	0.48	25	95.8
average			18.7		98.2

Table 4.1 Experiment Result of FF Reduction

	Clk of initial circuit	Clk of pure retiming	Reduction from initial (%)	Clk of rewired retiming	Reduction of from initial (%)
S298	41	30.5	25.6	29.1	29.0
S344	55	50.8	7.64	50	9.09
S349	54	49.3	8.7	51.7	4.25
S382	48	39.3	18.1	40	16.6
S444	52	48.4	6.92	47.3	9.03
S510	61	65.1	-6.7	64	-4.9
S526	49	42.4	13.4	45.2	7.75
S820	77	75.1	2.46	71.7	6.88
S832	81	71.7	11.5	76	6.17
S953	88	77.3	12.16	79.2	10
S1238	172	148.3	13.8	150	12.7
S1488	98	97.5	0.51	100.3	-2.34
S1494	95	104.9	-10.4	95.8	-0.84
average			7.97		7.96

Table 4.2 Experimental Result of Average Clock Period

	Clk of initial circuit	Clk of pure retiming	Reduction from initial (%)	Clk of rewired retiming	Reduction from initial (%)
S298	41	27	34.1	26	36.6
S344	55	42	23.6	44	20
S349	54	42	22.2	43	20.4
S382	48	34	29.2	33	31.2
S444	52	43	17.3	41	21.2
S510	61	58	4.9	60	1.63
S526	49	38	22.4	41	16.3
S820	77	67	12.9	65	15.5
S832	81	68	16.04	71	12.3
S953	88	70	20.4	71	19.3
S1238	172	138	19.7	138	19.7
S1488	98	92	6.12	89	9.18
S1494	95	90	5.26	88	7.36
average			18.01		17.74

Table 4.3 Experimental Result of Best Clock Period

The experimental data demonstrate that though being a quite simple scheme, this rewired retiming flow can stably further cut a significant 18.7% of FFs used in the retimed circuit without paying any compromise on the delay improvements produced by the original pure retiming, a result quite opening a new dimension for the retiming applications.

☐ **End of chapter.**

Chapter 5

Power Analysis for Rewired Retiming

Low power is a common crucial issue nowadays, and it is also a high concern of us to find out the impact on low power yielded by our proposing flow. Though previously there are some contributions in this field, as discussed in chapter 3, the major concern of them is the reduction of switching power itself but not cutting the power by reducing flip-flops. The cost function and power estimation of them is relatively too complicated and thus may not be accurate enough for practical use. Hence, in this chapter, we are going to analyze how the rewired retiming technique, as a relatively simple power cutting approach, impacts on the dynamic power of the circuit.

5.1 Power Model

After the final placement, we have a physical design with less clock period and minimized FFs. The circuit is then analyzed by Power Compiler to estimate its power dissipation, and compare with the circuit produced by pure retiming. Here, we adopt the Power Compiler as it is a well-acknowledged industrial tool for design and power analysis.

Power Compiler is a commercial tool widely used for power analysis and design optimization. Its power analyst engine provides detailed gate level power report by capturing switching activity, mapping the design to gates, and annotating the design.

The power dissipated in a circuit falls into two broad categories: static power and dynamic power. Static power is the power dissipated by a gate

when it is not switching, that is, when it is inactive or static. Static power is often called leakage power. Dynamic power is the power dissipated when the circuit is active, which is composed of switching power and internal power.

The switching power of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the cell outputs. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitances on the driving output. Internal power is the power dissipated within the boundary of a cell. During a signal switching, a circuit dissipates internal power by the charging or discharging of any existing capacitances internal to the cell. Internal power includes power dissipated by a momentary short circuit between the P and N transistors of a gate, called short-circuit power.

Power Compiler uses a zero-delay model for internal simulation and for propagation of switching activity during power analysis. This zero-delay model assumes that the signal propagates instantly through a gate with no elapsed time, and the switching activity propagating would make certain statistical assumptions.

In our experiment, we adapt the power model by Power Compiler, as well as the defaults of switching activity estimation at primary inputs as such:

P1 = 0.5 (the signal is in the logic "1 state" 50 percent of the time), where P1 is the probability that input P is at logic state 1.

TR = 0.5 fclk (the signal switches once every 2 clock cycles), where fclk is the frequency of the input's related clock in the design.

5.2 Experimental Results

In our experiment, the VTVT Standard Cell library (developed by the VTVT Group, Virginia Tech.) which targets the TSMC 0.18um, 1.8V CMOS process is adopted.

The leakage power, switching power, internal power and total dynamic power (sum of switching power and internal power) are estimated by Power Compiler with the above model and shown in the following.

	Cell Internal (uW)	Net Switching (uW)	Total Dynamic (uW)	Leakage (nW)
s298	240.8	56.5	297.3	7.57
s344	189.2	75.7	264.9	5.7
s349	257.3	100.7	357.9	5.7
s444	474	197.1	671.2	10.6
s526	476.3	163.8	640.2	12.9
s820	406	370.1	776.2	10.1
s832	436.5	439.8	876.3	11.3
s1238	1400	1561	2961	17.89
s382	538.5	171.8	710.4	12.1
s386	153.2	134.9	288.1	5
s953	761.6	497	1258.7	14.4
s1488	506.1	621.6	1128	16.2
s1494	523.6	653.6	1177.3	16.4

Table 5.4 Power Estimation of Pure Retiming Circuits

	Cell Internal (uW)	Net Switching (uW)	Total Dynamic (uW)	Leakage (nW)
s298	201.2	47.5	248.7	6.4
s344	223.7	72.9	296.6	5.4
s349	231.6	76.7	308.3	5.6
s444	509.2	157.9	667.1	11.2
s526	384.9	124.4	509.4	12.4
s820	407.9	395	802.9	10.6
s832	366.2	303.4	669.6	11.12
s1238	1220	1337	2560	16.3
s382	423.2	158.6	581.8	9.7
s386	153.1	134.9	288.1	5
s953	735.8	485.8	1221	14.2
s1488	500	591	1091	16.9
s1494	487.3	571.4	1059	16.4

Table 5.5 Power Estimation of Rewired Retiming Circuits

	Pure Retiming	Rewired Retiming	Reduction %
s298	297.3	248.7	16.35
s344	264.9	296.6	-12
s349	357.9	308.3	13.86
s444	671.2	667.1	0.611
s526	640.2	509.4	20.43
s820	776.2	802.9	-3.44
s832	876.3	669.6	23.59
s1238	2961	2560	13.54
s382	710.4	581.8	18.1
s386	288.1	288.1	0
s953	1258.7	1221	2.995
s1488	1128	1091	3.28
s1494	1177.3	1059	10.05
Average			8.26

Table 5.6 Total Dynamic Power Reduction of Rewired Retiming Circuits

Table 5.4 and Table 5.5 show the power estimation results of the circuits for pure retiming and for rewired retiming respectively. The results generated by Power Compiler include cell internal power, net switching power, total dynamic power (sum of cell internal and net switching power) and the leakage power.

Table 5.6 shows the reduction on total dynamic power by using our optimization scheme. For the benchmarks used in the experiment, our approach can achieve an average power cut down of 8.26%, compared to the results produced by pure retiming.

In general, in our experimented rewired retiming scheme a reduction of 18.7% in the number of FFs can be achieved with a simultaneous power cut of 8.26%, while with the improved clock period remains totally un-sacrificed.

□ **End of chapter.**

Chapter 6

Conclusion

The thesis has studied the retiming and rewiring techniques and proposed an optimization scheme combining the two techniques to improve the interconnect delay retiming in terms of flip-flops reduction and power reduction. Placement with close relation to delay estimation, post-placement delay estimation and power analysis are also studied for the research.

A simple while very effective scheme integrating rewiring and retiming is developed to minimize the number of FFs while without scarifying the original delay reduction. The whole flow is placement-aware and works tightly with a real placement thus makes the retiming result more practically realizable. The real clock period is calculated from the final placement.

Experimental results show a remarkable reduction on the number of FFs (18.7%), which can be considered a free gain because the retimed clock period is kept unchanged.

Besides a significant area cut, this extra FF reduction produced by our approach can also lead to a desirable savings on power (8.26%) and ease the clock tree generation because of less clock skews.

The work is a new attempt to use logic re-synthesis technique to further improve the retiming technique with real placement-based interconnect delays. Our approach not only reduces the area but also keeps retiming close to a real placement, and cuts down the adverse effect of the increased FFs

commonly introduced by a conventional retiming process. Particularly, this remarkable flip-flop and power cut-down is achieved without sacrificing any original retimed clock period.

☐ **End of chapter.**

Bibliography

[1] C. Leiserson, F. Rose, and J. B. Saxe, "Optimizing Synchronous Circuitry by Retiming," in *Proc. 3rd Caltech Conf*, 1983, pp. 87-116.

[2] C. Leiserson and J. B. Saxe, " Retiming Synchronous Circuitry " , *Algorithmatica*, vol. 6, pp. 5-31, 1991.

[3] N. Maheshwari and S. Sapatnekar, " Efficient Retiming of Large Circuits" , *IEEE Trans. VLSI Systems*, vol. 6, pp. 74-83, March 1998.

[4] S. Malik, E. M. Sentovich, R. K. Brayton and A. Sangiovanni-Vincentelli, " Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques" , *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 74-84, January 1991.

[5] Rajeev K. Ranjan, Vigyan Singhal, Fabio Somenzi, and Robert K. Brayton. On the Optimization Power of Retiming and Resynthesis Transformation. In Proc. ICCAD, pages 402–407, 1998

[6] Jason Cong, Honching Li, and Chang Wu. Simultaneous Circuit, Partitioning/Clustering with Retiming for Performance Optimization.In Proc. DAC, pages 460–465, 1999

[7] Jason Cong, Sung Kyu Lim, and Chang Wu. Performance Driven Multi-level and Multiway Partitioning with Retiming. In Proc. DAC, pages 274–279, 2000.

[8] Monteiro, J. ,Devadas, S., Ghosh, A., Retiming sequential circuits for low power, Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993.

[9] C. V. Schimpfle, Sven Simon, and Josef A. Nossek. Optimal Placement of Registers in Data Paths for Low Power Design. In Proc. ISCAS, pages 2160–2163, 1997

[10] A. El-Maleh, T. E. Marchok, J. Rajski, and W. Maly. Behavior and Testability Preservation under the Retiming Transformation. IEEE TCAD, 16:528–542, 1997

[11] Dennis K.Y.Tong, Evangeline F.Y.Young, Chris Chu, and Sampath Dechu, " Wire Retiming Problem With Net Topology Optimization, *IEEE Transactions of Computer-Aided Design of Integrated Circuits and Systems, Vol.26, No.9,September 2007*

[12] Neumann, I. Kunz, W., "Tight coupling of timing driven placement and retiming" , *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium* , Volume: 5, On page(s): 351-354 vol.

[13] S. C. Chang, L. P. P. P. van Ginneken, and M. Marek-Sadowska, "Fast Boolean Optimization by Rewiring," in *Proc. Int'l Conf. Computer-Aided Design*, Nov. 1996, pp. 262–269.

[14] Y. M. Jiang, A. Krstic, K. T. Cheng, and M. Marek-Sadowska, Post-layout Logic Restructuring for Performance Optimization," in *Proc. of Design Automation Conf.*, 1997, pp. 662–665.

[15] L. A. Entrena and K. T. Cheng, "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal" , *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 909-916, July 1995.

[16] S.C. Fu, T.K. Lam, and Y.L. Wu, "On Improved Scheme for Digital Circuit Rewiring and Application on Further Improving FPGA Technology Mapping," *Proc. ASP-DAC 2009. pp.197-202*

[17] W.C. Tang, W.H. Lo, and Y.L. Wu, "Further Improve Excellent Graph-Based FPGA Technology Mapping by Rewiring," *Proc. ISCAS 2007.*

[18] L. Zhou, W.C. Tang, W.H. Lo, and Y.L. Wu, "How Much Can Logic Perturbation Help from Netlist to Final Routing for FPGAs," *Proc. IEEE/ACM Design Automation Conference (DAC'07), 2007.*

[19] L. Zhou, W.C. Tang, and Y. L. Wu, "Fast Placement-Intact Logic Perturbation Targeting for FPGA Performance Improvement," *Proc. IEEE Southern Conference on Programmable Logic (SPL'07)* pp. 63-68, 2007.

[20] D. I. Cheng, C. C. Lin, and M. Marek-Sadowska, "Circuit Partitioning with Logic Perturbation," in *Proc. Int. Conference on Computer Aided Design*, Nov. 1995, pp. 650–655.

[21] Chris C.N. Chu, Evangeline F.Y. Young, Dennis K.Y. Tong and Sampath Dechu, "Retiming with Interconnect and Gate Delay" ,*Proceedings IEEE International Conference on Computer-Aided Design, pp.221-226, 2003*

[22] Y. L. Wu, W. Long, and H. Fan. A fast graph-based alternative wiring scheme for Boolean networks. In International VLSI Design Conference, pages 268–273, 2000

[23] M.L. Fredaman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms. Proceedings of the 25th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, October 1984, pp.338-346.

[24] D.B. Johnson, Efficient algorithms for shortest paths in sparse networks, Journal of the Association for Computing Machinery, Vol.24,No.1, January 1997, pp1-13.

[25] N. Shenoy and R. Rudell. Efficient implementation of retiming. In ICCAD, pages 226–233, 1994.

[26] H. Zhou. Deriving a new efficient algorithm for min-period retiming. In ASPDAC, pages 990–993, 2005.

[27] D. P. Singh, V. Manohararajah, and S. D. Brown. Incremental retiming for FPGA physical synthesis. In DAC, pages 433–438, 2005

[28] Jia Wang, Hai Zhou, An efficient incremental algorithm for min-area retiming, Annual ACM IEEE Design Automation Conference, Proceedings of the 45th annual conference on Design automation, Pages 528-533

[29] A. Shen, S. Devadas, A. Ghosh, and K. Keutzer. On Average Power Dissipation and Random Pattern Testability of Combinational Logic Circuits. In Proceedings of the Int '1 Conference on Computer-Aided Design, pages 402-407, November 1992.

[30] P. Duncan, S. Swamy, and R. Jain. Low-Power DSP Circuit Design Using Retimed Maximally Parallel Architectures. In Proceedings of the I't Symposium on Integrated Systems, pages 266–275, March 1993.

[31] Monteiro, J. ,Devadas, S., Ghosh, A., Retiming sequential circuits for low power, Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993.

[32] Jeroen Leitjen, Jef van Meerbergen, and Jochen Jess. Analysis and reduction of glitches in synchronous network. Received:1996.

[33] Jose Monteiro and Srinivas Devadas. Computer-Aided Design Techniques for Low Power Sequential Logic Circuits. Kluwer Academic Publishers, 1997.

[34] Narayanan, U. , Peichen Pan, Liu, C.L. "Low Po wer Logic Synthesis under a General Delay Model", 1998 International Symposium on Low Power Electronics and Design, 1998. Proceedings. 10-12 Aug 1998 On page(s): 209- 214