Wire Retiming Problem With Net Topology Optimization

Dennis K. Y. Tong, Evangeline F. Y. Young, Chris Chu, and Sampath Dechu

***Abstract*—In this paper, we study the retiming problem of sequential circuits with net topology optimization. Both interconnect and gate delay are considered in retiming. Most previous retiming algorithms have assumed ideal conditions for the nonlogical portions of data paths, which are not sufficiently accurate to be used in high-performance circuits today. In our modeling, we assume that the delay of a wire is directly proportional to its length. This assumption is reasonable since the quadratic component of a wire delay is significantly smaller than its linear component when the more accurate Elmore delay model is used. A simple experiment was conducted to illustrate the validity of this assumption. We present two approaches to solve the retiming problem, both of which have polynomial time complexity. The first one can compute the optimal clock period, while the second one is an improvement over the first one in terms of practical applicability. The second approach gives solutions that are very close to the optimal (0.06% more than the optimal on average) but in a much shorter runtime. The optimally retimed circuit will then be realized physically by placing the registers and finding the net topologies. In contrast to many previous works [*Proc. IEEE Int. Conf. Comput.-Aided Des.*, p. 136, 1998], [*IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 22(7) Jul. 2003] that performed simple calculations to determine the register positions, our approach can preserve the optimal clock period that is obtained by the retiming step and utilize as few registers as possible. Minimization of register number saves both area and power in register and clock loading. Our topology optimization step is shown to be optimal for nets with four or fewer pins, and this type of nets constitutes over 90% of the nets in a sequential circuit on average. Using the ISCAS89 benchmark, we tested our algorithm with a 0.35-μm complementary metal–oxide–semiconductor standard cell library. Silicon Ensemble was used to layout the design with a row utilization of 50%. Experimental results showed that our algorithm could find the best sharing of registers for a net in most of the cases, i.e., using the minimum number of registers while preserving the target clock period that is obtained by the retiming step, within a minute run on an Intel Pentium IV 1.5 GHz PC with 512 MB RAM.**

***Index Terms*—Design automation, flip-flops, interconnect delay, placement, registers, retiming, very-large-scale integration.**

I. Introduction

RETIMING [3] is a useful and popular technique for performance optimization of sequential circuits. It relocates registers to reduce cycle time while preserving the functionalities of circuits. Much effort has been made to apply this technique in different areas such as power reduction [4], [5], testability [6], [7], logic resynthesis [8], circuit partitioning [9]–[11], and physical planning [12]. Some extended its applicability to large practical circuits efficiently [13]–[20]. However, most retiming algorithms have assumed ideal conditions for the nonlogical portions of data paths, specifically ignoring interconnect delay. As process technology gets down to deep submicrometer, interconnect delay becomes a major factor of path delay. Without including this delay component, existing retiming algorithms are not sufficiently accurate to be used in practical high-performance circuits today. Besides, it is very important to be able to realize a retimed circuit physically to achieve the optimal clock period that is obtained by the retiming step. In this paper, we study the problem of retiming with both interconnect and gate delay and propose a scheme to realize an optimally retimed circuit physically to achieve the target clock period.

The choice of an accurate interconnect delay model is important. In [21] and [22], interconnect delay was incorporated into the retiming process, but simplified assumptions were made such that the interconnect delay between adjacent registers on the same wire was neglected. Another approach to integrate retiming into detailed placement was presented in [1]. After an initial place and route, heuristics were used to estimate interconnect delay. Retiming and post retiming placement were then performed to optimize the circuit performance. A recent paper [23] of Tabbara *et al.* applied retiming in the deep submicrometer (DSM) domain, and interconnect delay was considered. It was done by having a lower bound on the number of registers on each wire e_{uv}, while the delay at nodes was irrelevant. Registers could be retimed into a node that represented a component and affected the total area of the components. Retiming was performed to satisfy the constraint on the number of registers on each wire while minimizing the total area of the components. In [15], a clock skew solution corresponding to an optimal clock period was converted into a retiming solution, which was guaranteed to be at most one gate delay larger than the optimal clock period. However, their current approach to perform this conversion considered only gate delay. Lin and Zhou [24]–[26] have considered the retiming problem with linear interconnect delay model, but they have formulated the problem differently on chip level with macroblocks, etc.

Manuscript received July 11, 2005; revised April 26, 2006 and October 27, 2006. This work was supported by the Direct Grant for Research of the Chinese University of Hong Kong under Projects 2050321 and 2050352. This paper was recommended by Associate Editor C. J. Alpert.

D. K. Y. Tong and E. F. Y. Young are with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong (e-mail: fyyoung@cse.cuhk.edu.hk).

C. Chu is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011-3060 USA (e-mail: cnchu@iastate.edu).

S. Dechu is with the Blaze DFM, Inc., Sunnyvale CA 94089 USA (e-mail: sampath@blaze-dfm.com).

Digital Object Identifier 10.1109/TCAD.2007.895583

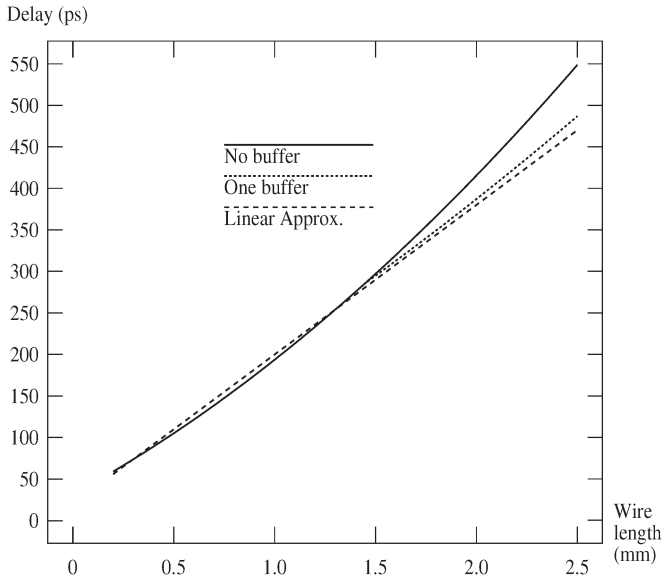


Fig. 1. Simple experiment to illustrate the relationship between wire delay and wire length.

In our model, the delay of a wire is assumed to be directly proportional to its length.[1] When a wire is short, the quadratic component of the wire delay is significantly smaller than its linear component. For a long wire, buffer insertion can be performed to break the wire into short segments. A simple experiment was conducted to illustrate the validity of this assumption, and the result was shown in Fig. 1. In this experiment, the Elmore delay model was used, and the parameters were based on the 0.07-μm technology. This graph shows the relationship between wire delay (y axis) and wire length (x axis). If the wire is shorter than 1.46 mm, the error of using a linear approximation is at most 5.48%. If the wire is longer than 1.46 mm, the delay can be reduced by inserting a buffer, and the error that resulted is even less.

We present two retiming approaches in this paper, both of which have polynomial time complexity. The first one is extended from the mixed integer linear programming (MILP) approach in [3] to consider both gate and wire delay and can solve the retiming problem optimally, i.e., relocating the registers in a circuit to give the smallest possible clock period. The second one transforms the problem into a single-source longest path problem and then applies a technique to reduce the size of the graph for the longest path computation. It is an improvement over the first one in terms of practical applicability. It gives solutions very close to the optimal (0.06% more than the optimal on average) but in a much shorter runtime.

After a circuit is retimed, we need to realize it physically. A net is represented as a branch of edges in a retiming graph model that does not bear any information about the net topology and register positions. It is unknown whether the clock period that is obtained by retiming can be realized in the design. Being able to obtain the net topologies and place the registers to preserve the target clock period is important, or it will make the retiming optimization meaningless. Minimizing the number of registers that are used is also essential, as the size of a register is usually several times larger than that of a simple gate, regardless of the process technology being used. There are several previous works on postretiming register placement, but many of them suffer from the problem of oversimplification when wire delay dominates. For example, in [1], the authors assume that a register is located at the geometric center of the connected gates. A similar problem occurs in [2] in which the authors determine the position of a register in such a way that the sum of the net lengths that are connected to that register is minimized.

We devised a scheme to realize a retiming solution physically to achieve a target clock period, given the gate positions. This problem involves two main subproblems, namely: 1) *topology finding* and 2) *register placement*. As we have mentioned before, a net is modeled as a branch of edges in the retiming graph; topology finding refers to the problem of finding an optimal sharing of registers among the fan-out edges of a net given the geometric positions of the connected gates. After topology finding, we need to compute an appropriate position for each register given the constraints in placement (some occupied areas do not allow register insertion), and this problem is known as register placement. Given a circuit with its placement (we used standard cell design in our experiments), retiming is first performed on the circuit to obtain the optimal clock period; then, topology finding and register placement will be performed to realize the retimed solution physically. Our approach can find the optimal topology, i.e., using the minimum number of registers while preserving the clock period, for four or fewer pin nets. Since nets with four or fewer pins constitute, on average, over 90% of the nets in a circuit, our proposed algorithm offered an agreeable performance in the experiments. Nearly all the nets had their best topologies found, and registers were inserted successfully to achieve the target clock period.

The remainder of this paper is organized as follows: We present the problem statement in Section II. The optimal and the fast approaches for the retiming problem are presented in Sections III-A and III-B, respectively. The topology finding and register placement step are discussed in Section IV-A and Section IV-B, respectively. Experimental results are shown in Section V. A conclusion follows in Section VI.

II. Problem Formulation

Given sequential circuit C and its placement $\wp$, we want to retime C to obtain the optimal clock period and implement this retimed solution in $\wp$ by inserting registers into $\wp$ and finding the connection topologies between the gates/registers. This problem can be divided into two parts: 1) retiming and 2) topology optimization. We will describe these two subproblems in detail in the following sections.

III. Retiming With Interconnect and Gate Delay

A sequential circuit C can be represented by a directed graph $G(V, E)$, where each node v corresponds to a combinational gate and each directed edge e_{uv} represents a connection from

[1]Please note that the retiming result in Section III can also be applied to other delay models since the result is independent of how the interconnect delay d_{ij} between two gates i and j changes with the length of the wire from i to j.

the output of gate u to the input of gate v, through zero or more registers. Without loss of generality, we assume that G is strongly connected. If not, we can add a source node s and connect it to all primary inputs, add a target node t and connect all primary outputs to it, and connect t to s. Then, the resulting graph is strongly connected. If we set the delay of s, t, and all the added edges to zero, and set the number of registers on e_{ts} to one and that on the other added edges to zero, a retiming solution of the modified graph will also be a valid retiming solution of the original graph as long as e_{ts} still has one register. Let w_{uv} be the number of registers on edge e_{uv}. Let d_{uv} be the interconnect delay of edge e_{uv} if all the registers are removed. Note that the delay of an interconnect segment is assumed to be proportional to the length of the segment. Let d_u be the gate delay of node u.

Traditionally, interconnect delay is ignored during retiming. A retiming solution can be viewed as a labeling of the nodes $r : V \rightarrow Z$, where Z is the set of integers [3]. The retiming label $r(v)$ for a node v represents the number of registers that were moved from its outputs toward its inputs. After retiming, the number of registers $\hat{w}_{uv}$ on an edge e_{uv} is given by $\hat{w}_{uv} = r(v) + w_{uv} - r(u)$.

As interconnect delay is dominating in the Very DSM technology, the exact position of each register will affect the clock period. A *retiming solution* should specify both retiming label $r(v)$ for each node v and the exact positions of the $\hat{w}_{uv}$ registers on each edge e_{uv}. Retiming should be formulated as a problem of determining a *feasible retiming solution*, i.e., a retiming solution in which the number of registers $\hat{w}_{uv}$ on each edge e_{uv} is nonnegative, such that the clock period of the retimed circuit is minimized. In the following, we show how to check whether a particular clock period T can be achieved by a feasible retiming solution. The minimum achievable clock period T_{opt} can then be found by binary search.

A. Exact Approach

This approach is extended from the MILP approach in [3]. In the original formulation, only gate delay is considered, and there is thus no differences between having one or more than one registers on a wire. Their technique can be extended to solve the problem with both gate and interconnect delay optimally by modifying some of the constraint formulation. In order to formulate the problem as a MILP, for each gate v, we need to define a term $a(v)$ that represents the maximum arrival time at the output of gate v. An example to illustrate this definition is shown in Fig. 2. We can then formulate the problem as the following MILP:

$$d_v \leq a(v) \qquad \forall v \in V \tag{1}$$

$$a(v) \leq T \qquad \forall v \in V \tag{2}$$

$$r(v) + w_{uv} - r(u) \geq 0 \qquad \forall e_{uv} \in E \tag{3}$$

$$a(v) \geq a(u) + d_{uv} + d_v - T\,(r(v) + w_{uv} - r(u)) \qquad \forall e_{uv} \in E \tag{4}$$

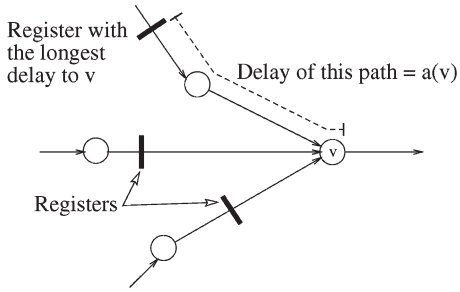


Fig. 2. Example to illustrate the meaning of $a(v)$.

where T is the clock period that we want to check whether it is achievable. Since $a(v)$ is the longest delay to the output of gate v from a register that is connected directly to an input of v, this delay must be at least the delay of gate v, so $d_v \leq a(v)$, as stated in (1). Besides, this delay cannot exceed clock period T, as required in (2). Constraint (3) is needed for a feasible retiming solution. Constraint (4) is to ensure that enough registers are on each edge e_{uv} to achieve a clock cycle T. As the largest possible delay between two adjacent registers is T, the right-hand side of constraint (4) is reduced by T for each register on edge e_{uv}. Note that this constraint also captures the scenario when there is no registers on edge e_{uv}. In that case, the arrival time at node u contributes directly to the arrival time at node v. In [3], wire delay is not considered, so we only need to differentiate the cases when a wire has zero or nonzero registers on it. Therefore, the inequality (4) is written as $a(v) \geq a(u) + d(v)$ whenever $e_{uv} \in E$ and $r(u) - r(v) = w(e_{uv})$, i.e., whenever an edge $e_{uv} \in E$ has no registers on it.

By introducing a variable $R(v)$ at each node v that is defined as $a(v)/T + r(v)$, the preceding set of constraints (1)–(4) can be rewritten as a set of difference constraints as follows:

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

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

$$r(u) - r(v) \leq w_{uv} \qquad \forall e_{uv} \in E \tag{7}$$

$$R(v) - R(u) \geq \frac{d_{uv}}{T} + \frac{d_v}{T} - w_{uv} \qquad \forall e_{uv} \in E. \tag{8}$$

Notice that (5)–(8) is a set of difference constraints involving both integer and real variables. There are $|V|$ real variables $R(v)$, $|V|$ integer variables $r(v)$, and $2|V| + 2|E|$ constraints. This can be solved in polynomial time of $O(|V||E| + |V|^2 \lg |V|)$ if Fibonacci heap is used as the data structure [27].

If the preceding set of constraints is solvable, the values of $r(v)$ and $a(v)$ for all $v \in V$ are known. We can then find the exact position of each register on a wire one by one as follows: For each edge e_{uv}, if there are registers that are retimed on it, i.e., $r(v) + w_{uv} - r(u) > 0$, the first register on this edge will be placed at a distance of delay $T - a(u)$ from the output of gate u. Other registers are then placed as far from each other as possible, i.e., at a distance of delay T from the previous one, until reaching gate v. All the remaining registers on this edge are then placed right before v.

B. Fast Approximate Approach

In this approach, we first replace each gate by a wire of the same delay and then solve the problem with only interconnect delay optimally and efficiently. Those registers that are retimed "into" a gate are moved either to the input or the output wires of the gate. The exact positions of the registers on the wires are then determined by a linear program to minimize the clock period. The solution that is obtained by this approach is very close to the optimal, on average, as shown by the experimental results. In the following, we first show how the retiming problem with interconnect delay only can be solved optimally. Then, we describe in detail how gate delay can be handled simultaneously.

1) Retiming With Interconnect Delay Only: In this section, we assume that $d_v = 0$ for all $v \in V$. This problem with zero gate delay is the same as the maximum cycle ratio problem, which has been studied in many previous works [28]–[31] before. We first show that the clock period feasibility problem can be reduced to a single-source longest path problem. We then present a fast algorithm to solve the longest path problem. We solve the set of constraints (5)–(8) with the help of the following lemma.

Lemma 1: Assume that $d_v = 0$. Given $R(v)$ for all $v \in V$ satisfying constraint (8), we can obtain a solution to constraints (5)–(8) by setting $r(v) = \lfloor R(v) \rfloor$ for all $v \in V$.

Proof: It is clear that $0 \le R(v) - \lfloor R(v) \rfloor < 1$ for all $v \in V$. Therefore, (5) and (6) are satisfied. For any $e_{uv} \in E$,

$$\begin{aligned} r(u) - r(v) &\le R(u) - r(v) \text{ as } r(u) \le R(u) \\ &\le \left(\frac{d_{uv}}{T} + R(u)\right) - r(v) \text{ as } \frac{d_{uv}}{T} > 0 \\ &\le (w_{uv} + R(v)) - r(v) \text{ by constraint (8)} \\ &< w_{uv} + 1 \text{ as } R(v) - r(v) < 1. \end{aligned}$$

As $r(u) - r(v)$ is an integer, it must be less than or equal to w_{uv}. Hence, constraint (7) is also satisfied. ■

Lemma 1 implies that we can first solve constraint (8) to find $R(v)$, and it is then easy to find $r(v)$ to satisfy the other three constraints. Notice that if $d_v \neq 0$ for some $v \in V$, Lemma 1 does not hold as constraint (5) is not satisfied. This technique is similar to that used in [30] to find an approximately optimal retiming in a nonunit-delay circuitry with gate delay only. The problem of finding $R(v)$ for all $v \in V$ to satisfy constraint (8) can be viewed as a single-source longest path problem on G with length l_{uv} equals $d_{uv}/T - w_{uv}$ for each $e_{uv} \in E$. As G is strongly connected, we can pick an arbitrary node as the source node s.[2] Note that edge lengths can be positive. If G has a positive cycle, the set of constraints has no solutions. It means that clock period T is infeasible.

The single-source longest path problem in Section III-B1 can be solved by the Bellman–Ford algorithm [32], and the time complexity is $O(|V||E|)$. This algorithm may still be slow in practice. An interesting idea of using small feedbacks to speed up the Bellman–Ford algorithm is found in [33] with time complexity $O(|E||E^-|)$, where E^- is the set of edges in G with negative weights. In this section, we present a single-source longest path algorithm, which is faster in practice. The basic idea is to reduce the size of G by compacting some paths into edges before the Bellman–Ford algorithm is applied. The details are given here. We first transform graph $G(V, E)$ into a directed acyclic graph (DAG) $G'(V', E')$ by performing a depth-first traversal [32] starting from the source node s. The depth-first traversal defines a tree in G. Those nontree edges running from a node u to an ancestor v of u are called back edges. If we point all incoming back edges of a node v to an extra node v', the resulting graph will be a DAG because every simple cycle in G involves at least one back edge. Formally, we use E_b to denote the set of back edges and V_b to denote the set of nodes with an incoming back edge. For each node v in V_b, we introduce an extra node v'. The back edge e_{uv} is removed from the graph, and the edge $e_{uv'}$ is added. The resulting DAG is $G'(V', E')$, where $V' = V \cup \{v' | v \in V_b\}$ and $E' = (E - E_b) \cup \{e_{u,v'} | e_{u,v} \in E_b\}$. We set length $l_{uv'}$ of edge $e_{uv'}$ to l_{uv}. To illustrate the transformation, consider graph G in Fig. 3(a) with source node A. Suppose that the depth-first traversal visits the nodes in the order *ACDEFB*. Then, $E_b = \{e_{DA}, e_{CA}, e_{FC}, e_{FA}\}$, and $V_b = \{A, C\}$. We introduce two extra nodes A' and C', and replace the four edges e_{CA}, e_{DA}, e_{FA}, and e_{FC} with the edges $e_{CA'}$, $e_{DA'}$, $e_{FA'}$, and $e_{FC'}$, respectively. The resulting DAG is shown in Fig. 3(b).

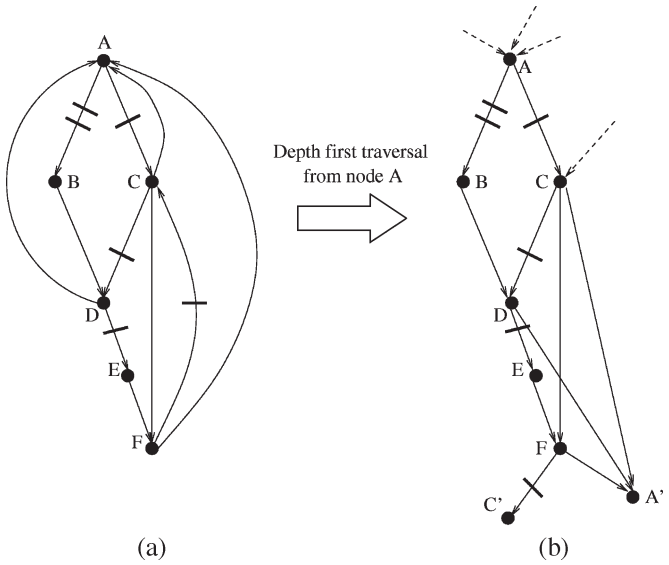


Fig. 3. Example to illustrate the transformation to a DAG. (a) Original graph G. (b) DAG G'.

We then construct graph H with node set V_b. The edge set E_H contains edge e_{uv} for $u, v \in V_b$ if there exists a path from u to v in G with either no back edge or one back edge at the end. The length of edge e_{uv} in $H(l^H_{uv})$ is the longest path distance among those paths. Note that the longest path distance from u to v in G with no back edge (respectively, with one back edge at the end of the path) equals the longest path distance from u to v (respectively, from u to v') in G'. Hence, l^H_{uv} for all $u, v \in V_b$ can be computed by solving $|V_b|$ single-source longest path problems in G' for different source nodes in V_b. As G' is a DAG, each single-source longest path problem can be

[2] If the original circuit is not strongly connected, a source node s has already been added.

solved in linear time by visiting the nodes in topological order. The time complexity to construct H is therefore $O(|V_b||E|)$.

It is obvious that every path in H corresponds to at least one path in G of the same length. Therefore, if H contains a positive cycle, G will also contain a positive cycle. On the other hand, if G contains a positive cycle, the cycle can be broken up into a set of paths $p_1, p_2, \ldots, p_k$ such that both endpoints of each path p_i are in V_b. Notice that each path p_i corresponds to an edge in H of at least the same length. So, H must also contain a positive cycle. Therefore, we can solve the positive cycle detection problem on H instead of on G. If H has no positive cycles, $R(v)$ for all $v \in V_b$ can be found from H. $R(v)$ for all $v \in V - V_b$ can then be found in linear time by propagating $R(v)$ for all $v \in V_b$ through G' in topological order.

The most time-consuming steps are steps 7 and 8 inside the binary search loop. Step 7 can be done in $O(|V_b||E|)$ time as discussed previously. Step 8 can be done in $O(|V_b||E_H|)$ time by the Bellman–Ford algorithm. As V_b contains much fewer nodes than V and E_H usually contains comparable or fewer edges than E, this technique is usually more efficient than applying the Bellman–Ford algorithm to G directly. The total time complexity is $O(|V_b| \max\{|E|, |E_H|\} \lg(K/(\epsilon T_{\text{opt}})))$, where ϵ is the error bound for the binary search, K is the difference between the upper and lower bounds of the clock period initially, and T_{opt} is the optimal clock period. Notice that the number of iterations in the binary search, i.e., the logarithmic term, can be reduced by finding the maximum delay-to-register ratio, which is a lower bound to the minimum clock period [30].

Algorithm *I-Retiming*()
/* Retime a sequential circuit with interconnect delay only to */
/* achieve the minimum possible clock cycle with an error*/
/* bound ϵ.*/
Input: A sequential circuit C with interconnect delay only
Output: An optimally retimed circuit of C
1. Build graph $G(V, E)$ from C
2. Build DAG G' by DFS(G)
3. C_{up} = a feasible clock, C_{low} = an infeasible clock
4. Do
5. $T = (C_{\text{up}} + C_{\text{low}})/2$
6. Update edge lengths of G' according to T
7. Build graph $H(V_b, E_H)$ with
$E_H = \{e_{uv} | u \in anc(v) \cup anc(v')$ in $G'\}$
by finding single-source longest paths in G'
8. If H does not have any positive cycle then
9. $C_{\text{up}} = T$
10. Else
11. $C_{\text{low}} = T$
12. while $(C_{\text{up}} - C_{\text{low}})/C_{\text{up}} > \varepsilon$
13. $T = C_{\text{up}}$ //C_{up} is always a feasible clock period
14. Compute $R(v)$ and $r(v)$ for each node $v \in V$
15. Compute the exact position of each register on a wire

2) Retiming With Interconnect and Gate Delay: In this section, we discuss how to consider interconnect and gate delay simultaneously based on the preceding algorithm for interconnect delay only. To consider gate delay, we first represent gate v with delay d_v by a wire $e_{v_1 v_2}$ with delay $d_{v_1 v_2} = d_v$. This transformation for the circuit in Fig. 3(a) is shown in Fig. 4(b). We can then obtain an optimal retiming on this transformed circuit $\tilde{G}$ using the algorithm in Section III-B-1. However, the retiming solution that is obtained on $\tilde{G}$ may not be feasible for the original circuit G because some registers may be retimed into a wire that represents a gate. Therefore, we need to perform a postprocessing step to get back a feasible retiming solution for G from the optimal retiming solution for $\tilde{G}$. This is done by linear programming.

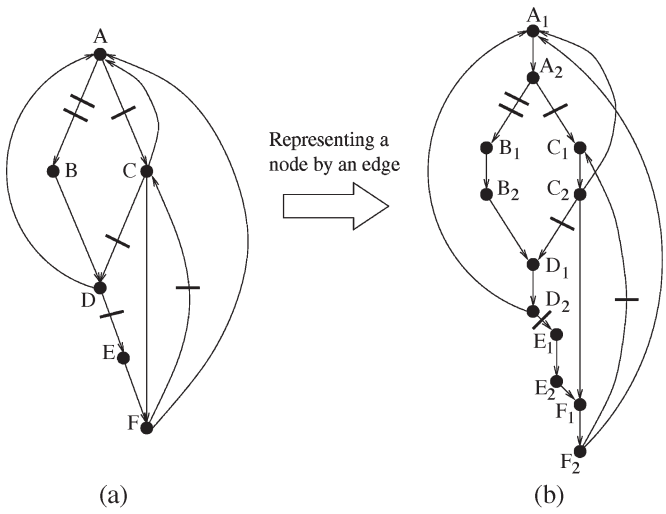


Fig. 4. Representation of gates by wires. (a) Original graph G. (b) Transformed graph $\tilde{G}$.

First, we move the registers in a gate either backward to the input wires or forward to the output wires of the gate, depending on which direction has a shorter distance. An example showing the relocation of registers is given in Fig. 5. After this relocation step, the number of registers $\hat{w}_{uv}$ on each edge e_{uv} is fixed. A linear program is used to determine the exact positions of the registers on the edges. Alternatively, the method in [26] can be used to minimize the clock period when the r values are unchanged in $O(|V|^2|E|)$ time. The objective of the linear program is to minimize the clock period T subject to the constraints in register count on each edge. In the following, we use x_{uv}^k to denote the delay from the kth register to the $k+$ first register of the wire from node u to node v in G for $k = 0, 1, \ldots, \hat{w}_{uv}$. Notice that, when $\hat{w}_{uv} = 0$, x_{uv}^0 is the delay of the whole wire and, when $k = 0$ and $k = \hat{w}_{uv} > 0$, x_{uv}^k are the delays of the wire from node u to the first register and from the last register to node v, respectively. The linear program is formulated as follows:

Minimize T

$$\text{Subject to} \quad \sum_{k=0}^{\hat{w}_{uv}} x_{uv}^k = d_{uv} \quad \forall e_{uv} \in E \tag{9}$$

$$x_{uv}^{\hat{w}_{uv}} + d_v \le a(v) \qquad \forall e_{uv} \in E \text{ s.t. } \hat{w}_{uv} > 0 \tag{10}$$

$$a(u) + x_{uv}^0 \le T \qquad \forall e_{uv} \in E \text{ s.t.} \hat{w}_{uv} > 0 \tag{11}$$

$$a(u) + d_{uv} \le a(v) \qquad \forall e_{uv} \in E \text{ s.t. } \hat{w}_{uv} = 0. \tag{12}$$

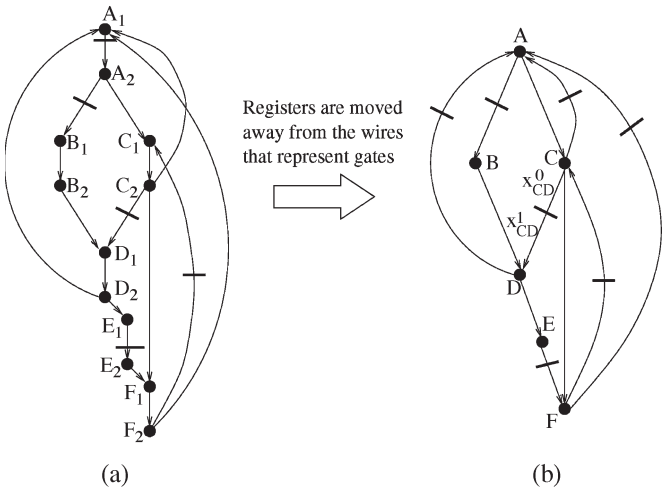


Fig. 5. Relocation of registers retimed into a gate. (a) Retimed solution in $\tilde{G}$. (b) Registers are relocated in G.

For the circuit in Fig. 5(b), example constraints are $x^0_{CD} + x^1_{CD} = d_{CD}$ for type (9), $x^1_{CD} + d_D \leq a(D)$ for type (10), $a(C) + x^0_{CD} \leq T$ for type (11), and $a(B) + d_{BD} \leq a(D)$ for type (12). We can solve this linear program to obtain the best possible clock period T^* under the register count constraint on each edge. Notice that this linear program can solve the subproblem of finding the best possible position of each register on a wire to optimally minimize the clock cycle *only when the register count on each edge is fixed*, but the overall approach of handling both interconnect and gate delay is not optimal. The overall algorithm $IG\text{-}Retiming()$ to handle both interconnect and gate delay is summarized as follows:

Algorithm *IG-Retiming*()
/* Retime a sequential circuit with both interconnect and gate*/
/* delay to achieve a clock cycle very close to the minimum. */
Input: A sequential circuit C with both interconnect and gate delay
Output: A retimed circuit of C
1. Build graph G from C
2. Build $\tilde{G}$ by replacing each gate in G by a wire of the same delay
3. Solve the retiming problem of $\tilde{G}$ by $I\text{–}Retiming()$
4. Move registers away from wires that represent gates
5. Set up a linear program based on the register count on each edge
6. Solve the linear program to obtain a feasible retiming solution and the smallest possible clock period T^*

IV. Flop Topology Optimization

After retiming, we need to realize the circuit physically, so that the optimal clock period that is obtained by retiming can be achieved. Given a retiming solution of the circuit (i.e., a target clock period T, a retiming label $r(v)$ at each gate v, and the maximum arrival time $a(v)$ at the output of gate v) and the positions of its gates, we want to find the topologies of the nets and place the registers to realize the circuit, preserving the target clock T as much as possible.

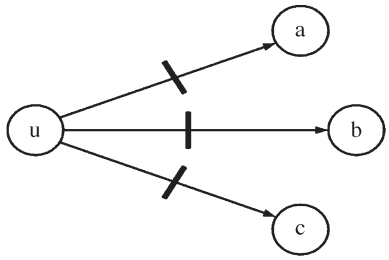


Fig. 6. Graph model of a four-pin net obtained after retiming in which each edge has a register.

Now, consider a net $N(s, D, L)$ in the retimed circuit, where s denotes the driving gate, D denotes the set of all driven gates, and L denotes the set of interconnections between s and each of the gates $d_i \in D$. Obviously, $\{s\} \cup D \subseteq V$ and $L \subseteq E$. For each edge $e_{sd_i} \in L$, we have a value $\hat{w}_r(s, d_i)$ representing the number of registers along edge e_{sd_i} after retiming. The problem is to insert the minimum number of registers for this net into the circuit such that the target clock period is preserved as much as possible. This problem comprises two main subproblems known as *topology finding* and *register placement*. Topology finding is the problem of finding a topology Υ_N of a net N given the exact geometric positions of the gates such that the minimum number of registers will be used and the target clock period will be preserved. Register placement is the problem of finding a position for each register given the topology Υ_N of net N.

Topology $\Upsilon_N = (P, K)$ is a tree (an acyclic graph with no designated root yet) that describes the structure of net N on the plane. Each node $p \in P$ corresponds to either a combinational gate or a register, and each edge $k_{uv} \in K$ represents a physical connection between gate/register u and gate/register v. Each node $p \in P$ that has only one adjacent node in Υ_N, i.e., $deg(p) = 1$ represents a combinational gate, while an internal node $p \in P$ that has more than one adjacent nodes, i.e., $deg(p) > 1$, represents a register. In Fig. 6, an example of a four-pin net in which each source-to-sink edge has a register after retiming is shown. There are five possible register sharing topologies in this example: 1) all the edges share a single register (maximum sharing), as shown in Fig. 7(a); 2) each edge has its own register (no sharing), as shown in Fig. 7(b); and 3) for the rest of the three equivalent cases, two edges share a single register, while the other one has a separate register, as shown in Fig. 7(c).

Although we can always identify the topology tree that has the maximum sharing of registers for a net, it is not always possible to place the registers on a chip using that topology while preserving the target clock period. Using case (a) in Fig. 7 as an example, suppose that the clock period that resulted from retiming T equals 1.5 units and the positions of the gates u, a, b, and c are (0, 0), (−3, 0), (0, 3), and (3, 0), respectively, as depicted in Fig. 8. Obviously, it is impossible to share a single register among the three edges without clock violation. Three separate registers have to be allocated and inserted exactly at

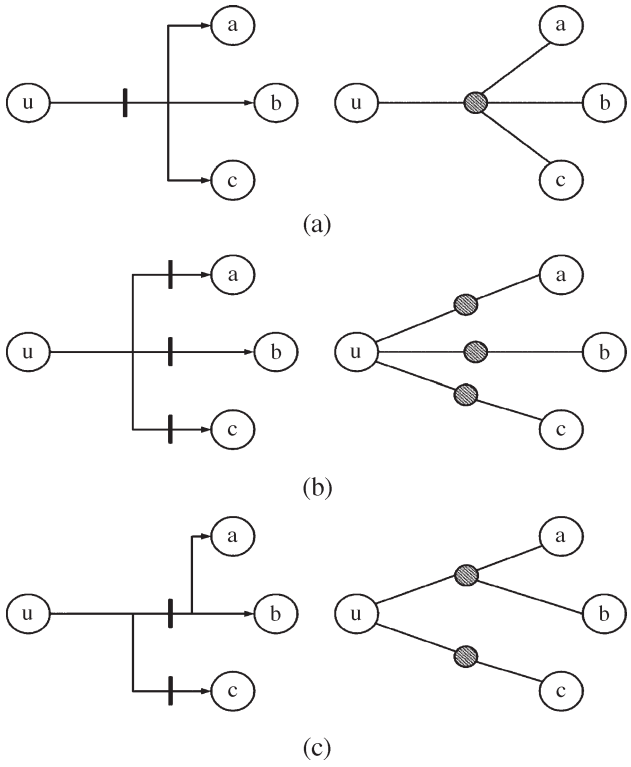


Fig. 7. Five possible register sharing topologies. The topology tree of each configuration is shown on the right. (a) Single register is shared between the edges (maximum sharing). (b) Each edge has a separate register (no sharing). (c) Two edges share a register, while the other edge has a separate one.

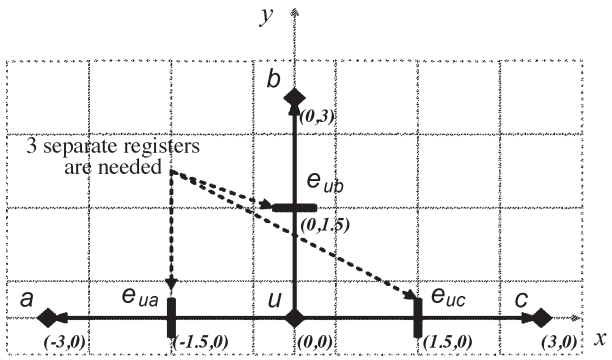


Fig. 8. Situation in which the registers cannot be shared in order to preserve the clock period $T = 1.5$ units.

$(-1.5, 0)$, $(0, 1.5)$, and $(1.5, 0)$ for edge e_{ua}, e_{ub}, and e_{uc}, respectively, in order to achieve the optimal clock period T.

Even if we have a feasible topology tree, it can happen that the suggested position for a register has been occupied, and we have to look for another appropriate position. The following sections will address how a feasible topology tree can be found and how the positions of the registers can be obtained.

A. Topology Finding

In this section, an algorithm is proposed to find the topology of a net given the constraints in placement such that the maximum sharing of registers is achieved and the clock period is preserved. This method can find the optimal topology for a net with four or fewer pins and can give near-optimal solution for a net with five or more pins according to the experimental results.

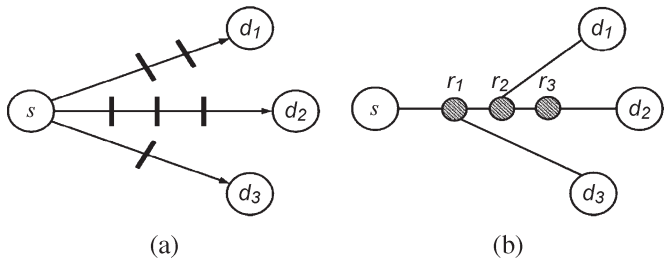


Fig. 9. (a) Retiming graph model and (b) the corresponding best possible topology $\Upsilon_{N_{\rm opt}}$ of a four-pin net example.

Given a net $N(s, D, L)$, a clock period T, and the maximal arrival time at the output of gate v, i.e., $a(v)$, we can obtain a feasible topology tree of N, i.e., Υ_N, as follows. First, we construct the best possible topology $\Upsilon_{N_{\rm opt}}$ for N, i.e., a topology having the minimum number of internal nodes (an internal node represents a register). Obviously, the number of internal nodes in $\Upsilon_{N_{\rm opt}}$ equals $Q = \max_{d_i \in D}\{\hat{w}_r(s, d_i)\}$, where $\hat{w}_r(s, d_i)$ denotes the number of registers on edge e_{sd_i} after retiming. We label each internal node as f_i representing the ith register on the net counting from source s for $1 \leq i \leq Q$. An example of the retiming graph model and the corresponding best possible topology $\Upsilon_{N_{\rm opt}}$ for a four-pin net is shown in Fig. 9.

We call the region on the plane where a register f can be placed the *candidate region* of f and is denoted by $D(f)$. For consistency, the candidate region $D(v)$ of a combinational gate v is the position of v itself, i.e., its coordinates (x_v, y_v), since v is already fixed in the placement. An *δ-extended region* of region $\Re$, which is denoted by $R^{+\delta}(\Re)$, is the region on the plane at distance δ or less from some point in $\Re$, assuming that the distance between two points is measured by their shortest Manhattan distance.

Besides, we define an *adjacent-gate region* for each node p in a topology tree, which is denoted by $A(p)$, as an δ-extended region from its candidate region $D(p)$, i.e., $A(p) = R^{+\delta}(D(p))$, where δ is defined differently for different types of nodes. The physical meaning of $A(p)$ refers to the region on the plane that encompasses *all* the possible positions of an adjacent gate of p. The value δ for the $A(p)$ of a node p is described as follows. If node p is an internal node, δ equals T. If node p represents a driven gate, δ equals $a(p) - d_p$. Otherwise, node p represents a driving gate, and we set δ to $T - a(p)$. Notice that all these regions are 45°-rotated rectangles on the rectilinear plane because of the Manhattan distance measurement.

Starting from the best possible topology $\Upsilon_{N_{\rm opt}}$, we will modify the topology tree incrementally until an optimal feasible topology Υ_N is obtained for net N. First, we choose the node that represents driving gate s as the root in $\Upsilon_{N_{\rm opt}}$ and direct all the edges away from s. Then, we will process each internal node f_i in $\Upsilon_{N_{\rm opt}}$ from $i = Q$ to $i = 1$, i.e., from the furthest register to the closest one, in the following manner. For each internal node f_i with a set of children $q_1, \ldots, q_m$, find a minimal set of the overlapping regions between $A(q_j)$ for $1 \leq j \leq m$, which

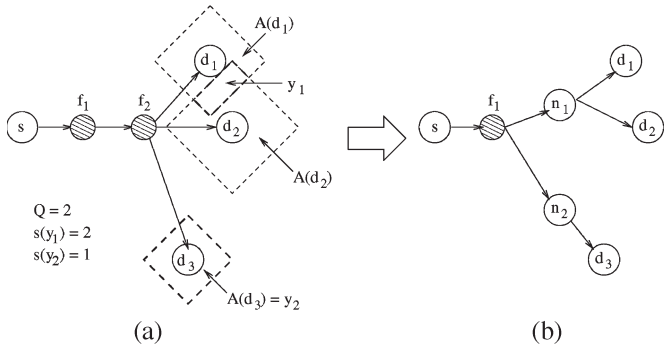


Fig. 10. Example illustrating the $SetY$ and $ChangeTree$ process.

is denoted by $Y_{\min} = \{y_1, \ldots, y_k\}$, such that the union of the elements in $Y_{\min}$ covers at least one point from each region $A(q_j)$. For each y_l in $Y_{\min}$, we call the number of regions that has at least one point in y_l as the *size* of y_l, which is denoted by $s(y_l)$. An example is show n in Fig. 10(a). The elements in $Y_{\min}$ are then sorted in a nonascending order of their sizes. The set $Y_{\min}$ can be found by the following procedure $SetY$:

Algorithm $SetY(f_i, \Upsilon_N)$
/* Find the $Y_{\min}$ of an internal node f_i in a topology tree Υ_N.*/
Input: An internal node f_i with children $q_1, \ldots, q_m$ in Υ_N
Output: $Y_{\min}$
1. $Y_{\min} = \emptyset$
2. Add $A(q_1)$ to $Y_{\min}$, i.e., $y_1 = A(q_1)$
3. For $j = 1$ to m
4. $overlapped = false$
5. For $l = 1$ to $|Y_{\min}|$
6. If $(y_l \cap A(q_j) \neq \emptyset)$
7. $y_l = (y_l \cap A(q_j)$
8. Sort the elements in $Y_{\min}$ in a nonascending order of their sizes
9. $overlapped = true$
10. Break
11. End if
12. End for
13. If $(overlapped = false)$
14. Increment $|Y_{\min}|$ by 1
15. Add $A(q_j)$ to $Y_{\min}$ at the end, i.e., $y_{|Y_{\min}|} = A(q_j)$
16. End if
17. End for

Notice that the union of the elements in $Y_{\min}$ covers at least one point from each region $A(q_j)$ for $1 \leq j \leq m$. Next, we can remove all the edges from f_i to its children $q_1, \ldots, q_m$ in $\Upsilon_{N_{\text{opt}}}$ and split node f_i into k new internal nodes $n_1, \ldots, n_k$, where node n_l corresponds to element y_l in $Y_{\min}$ for $1 \leq l \leq k$. In addition, we will assign region y_l as the candidate region of n_l, i.e., $y_l = D(n_l)$, for $1 \leq l \leq k$. Starting from the y_l with the largest size in $Y_{\min}$, an edge is added from n_l to each q_j that has no parent node yet and has $A(q_j)$ covered by y_l. This step is repeated until all the y_l's have been processed. Finally, an edge is added from the parent node of f_i to every newly created internal nodes n_l, and f_i will then be removed from the topology tree. An example is shown in Fig. 10(b). The preceding operations are described by the procedure $ChangeTree$ as follows:

Algorithm $ChangeTree(f_i, Y_{\min}, \Upsilon_N)$
/* Modify a topology tree Υ_N by replacing an internal node f_i by several other nodes.*/
Input: An internal node f_i in Υ_N and the corresponding $Y_{\min}$
Output: A modified topology tree
1. Remove all the edges from f_i to its children $q_1, \ldots, q_m$ in Υ_N
2. Instantiate k new internal nodes $n_1, \ldots, n_k$, where $k = |Y_{\min}|$
3. Assign region y_l as the candidate region of n_l, i.e., $y_l = D(n_l) \forall 1 \leq l \leq k$
4. For $l = 1$ to k
5. For $j = 1$ to m
6. If $(y_l \cap A(q_j) \neq \emptyset$ and q_j has no parent node yet)
7. Add an edge from n_l to q_j
8. End if
9. End for
10. Add an edge from the parent node of f_i to n_l
11. End for
12. Remove f_i
13. Output(Υ_N)

After visiting all the internal nodes f_i in $\Upsilon_{N_{\text{opt}}}$ and modifying the topology as described previously, we will get a new topology tree Υ_N at the end. The whole algorithm of *topology finding* of net N is described in the following procedure $TopTree$.

Algorithm $TopTree(N)$
/* Construct a feasible topology tree Υ_N of a net N.*/
Input: A net N in a circuit
Output: A topology tree Υ_N for N
1. Construct the best possible topology tree $\Upsilon_{N_{\text{opt}}}$ for net N
2. $\Upsilon_N = \Upsilon_{N_{\text{opt}}}$
3. For $i = Q$ to 1 where Q is the number of internal nodes in $\Upsilon_{N_{\text{opt}}}$
4. $Y_{\min} = GetY(f_i, \Upsilon_N)$
5. $\Upsilon_N = ChangeTree(f_i, Y_{\min}, \Upsilon_N)$
6. End for
7. Output(Υ_N)

To prove the correctness of the above algorithm, i.e., the statement of Theorem 1, we need to prove the following three lemmas first.

Lemma 2: Given a set of n 45°-rotated rectangles $R_1, \ldots, R_n$ on a rectilinear plane, if $R_1 \cap \cdots \cap R_n \neq \emptyset$, then $R^{+x}(R_1) \cap \cdots \cap R^{+x}(R_n) \neq \emptyset$, where x is a nonnegative real number.

Proof: Since $R_1 \cap \cdots \cap R_n \subseteq R^{+x}(R_1) \cap \cdots \cap R^{+x}(R_n)$, the argument follows. ■

Lemma 3: Given a set of n 45°-rotated rectangles $R_1, \ldots, R_{n-1}$ and S on a rectilinear plane, if $S \cap R_i \neq \emptyset$ for $1 \leq i \leq n-1$ and $R_1 \cap \cdots \cap R_{n-1} \neq \emptyset$, then $S \cap (R_1 \cap \cdots \cap R_{n-1}) \neq \emptyset$.

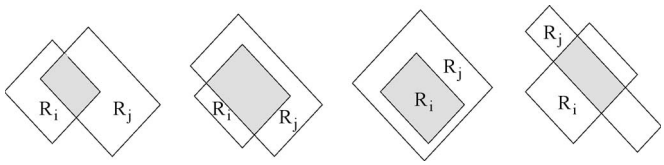


Fig. 11. Four possible ways that two 45°-rotated rectangles overlap.

Proof: It can be proven by induction. For the base case when $n = 3$, consider three 45°-rotated rectangles R_1, R_2, and S on a rectilinear plane. If $R_1 \cap R_2 \neq \emptyset$, there are only four ways that R_1 and R_2 overlap with each other, as shown in Fig. 11. It is easy to see that, in each case, if $S \cap R_1 \neq \emptyset$ and $S \cap R_2 \neq \emptyset$, $S \cap R_1 \cap R_2 \neq \emptyset$. When $n > 3$, let $R'_{n-2} = R_1 \cap \cdots \cap R_{n-2} \neq \emptyset$. By the inductive hypothesis, $S \cap R'_{n-2} \neq \emptyset$. Since $S \cap R_{n-1} \neq \emptyset$ and $R'_{n-2} \cap R_{n-1} \neq \emptyset$, we can conclude that $S \cap R'_{n-2} \cap R_{n-1} \neq \emptyset$, i.e., $S \cap (R_1 \cap \cdots \cap R_{n-1}) \neq \emptyset$, following a similar argument as in the case for $n = 3$. ■

Lemma 4: Given two 45°-rotated rectangles A and B on a rectilinear plane, we denote the *n times T-extended regions* of A and B as A_n and B_n, respectively, i.e., $A_n = R^{+(n\times T)}(A)$ and $B_n = R^{+(n\times T)}(B)$. Suppose that $A \cap B = R_{AB} \neq \emptyset$, we denote the *$n$ times T-extended region* of R_{AB} by $(R_{AB})_n$, i.e., $(R_{AB})_n = R^{+(n\times T)}(R_{AB})$. It is claimed that if there exists a point $x \in A_n \cap B_n$, $x \in R^{+T}((R_{AB})_{n-1})$ for all $n \geq 1$.

Proof: We prove by induction on n.

Base case: Consider the case when $n = 1$. Suppose that $x \in A_1 \cap B_1$, the T-extended region from the position of x is given by $R^{+T}(x)$. Obviously, $R^{+T}(x) \cap A_0 \neq \emptyset$ and $R^{+T}(x) \cap B_0 \neq \emptyset$ because $x \in A_1 \cap B_1$. Since $A_0 \cap B_0 \neq \emptyset$ (because $A_0 = A$, $B_0 = B$ and $A \cap B \neq \emptyset$), $R^{+T}(x) \cap (A_0 \cap B_0) \neq \emptyset$ by Lemma 3. Therefore, $x \in R^{+T}((R_{AB})_0)$, and the claim is true.

Inductive Step: Assume that the claim is true for $n = j - 1$, where j is a positive integer ≥ 2, i.e., if there exists a point $x \in A_{j-1} \cap B_{j-1}$, $x \in R^{+T}((R_{AB})_{j-2})$. Consider the case when $n = j$. Given a point $x \in A_j \cap B_j$, the T-extended region from x is denoted by $R^{+T}(x)$. Obviously, $R^{+T}(x) \cap A_{j-1} \neq \emptyset$ and $R^{+T}(x) \cap B_{j-1} \neq \emptyset$ because $x \in A_j \cap B_j$. By Lemma 2, since $A_0 \cap B_0 \neq \emptyset$, $A_{j-1} \cap B_{j-1} \neq \emptyset$. Therefore, $R^{+T}(x) \cap (A_{j-1} \cap B_{j-1}) \neq \emptyset$ by Lemma 3. By the induction hypothesis, if $R^{+T}(x) \cap (A_{j-1} \cap B_{j-1}) \neq \emptyset$, $R^{+T}(x) \cap R^{+T}((R_{AB})_{j-2}) \neq \emptyset$. Therefore, $x \in R^{+T}((R_{AB})_{j-1})$. ■

Theorem 1: The proposed algorithm $TopTree()$ can find a topology that maximizes the sharing of registers for an i-pin net, where $2 \leq i \leq 4$, and the target clock period T is preserved.

Proof: We prove the three possible cases one by one.

Case 1: $i = 2$—This case is trivial because there is only one source s, one sink t_1, and one edge e_{st_1} in a two-pin net, and there are no other edges to share registers with. The algorithm will start from the furthest internal node f_Q and take the adjacent-gate region of t_1 $A(t_1) = R^{+(a(t_1)-d_{t_1})}(D(t_1))$ as the candidate region of f_Q, i.e., $D(f_Q) = A(t_1)$. Next, the algorithm will process node f_{Q-1} and take the adjacent-gate region of f_Q, $A(f_Q) = R^{+T}(D(f_Q))$ as the candidate region of f_{Q-1}, i.e., $D(f_{Q-1}) = A(f_Q)$.

By substitution, $D(f_{Q-1})$ can be represented as an extended region from the position of the sink t_1, $D(f_{Q-1}) = R^{+((a(t_1)-d_{t_1})+T)}(D(t_1))$. The algorithm repeats the preceding steps until it reaches the first internal node f_1, where $D(f_1) = R^{+((a(t_1)-d_{t_1})+(Q-1)\times T)}(D(t_1))$. Since the retiming solution is valid, the interconnect delay between s and t_1 will not exceed $(T - a(s)) + ((Q-1) \times T) + (a(t_1) - d_{t_1})$. Therefore, the algorithm can find the candidate region for every register and return the best possible topology when it terminates.

Case 2: $i = 3$—Given a three-pin net, let s be the source and t_1 and t_2 be the two sinks. Let $\hat{w}_r(s, t_1)$ and $\hat{w}_r(s, t_2)$ be p and q, respectively, where $1 \leq p \leq q$. Suppose that there exists a topology tree of maximum register sharing for the three-pin net such that the first k registers, where $1 \leq k \leq p$, are shared (notice that if the kth register can be shared, the hth register can also be shared, where $1 \leq h \leq k$) but that the algorithm cannot find such a topology.

Since the algorithm cannot find that optimal topology, it must fail to find an overlapping region for the kth register to be shared. At the point of failure, the algorithm should find that the regions $R^{+((a(t_1)-d_{t_1})+T\times(p-k-1))}(t_1)$ and $R^{+((a(t_2)-d_{t_2})+T\times(q-k-1))}(t_2)$ do not overlap. However, these two regions encompass *all* the possible positions for the kth register from t_1 and t_2, respectively, such that clock period T will not be violated. Therefore, should the kth register be able to be shared as assumed, it must lie within these two regions, and the algorithm must be able to find it. Contradiction occurs.

Case 3: $i = 4$—Given a four-pin net, let s be the source and t_1, t_2, and t_3 be the three sinks. Let $\hat{w}_r(s, t_1)$, $\hat{w}_r(s, t_2)$, and $\hat{w}_r(s, t_3)$ be p, q, and r respectively, where $1 \leq p \leq q \leq r$. Suppose that the algorithm is attempting to share the kth register, where $1 \leq k \leq p$, i.e., it is trying to find a minimal subset of the overlapping regions such that it covers all the extend regions $R^{+((a(t_1)-d_{t_1})+T\times(p-k-1))}(t_1)$, $R^{+((a(t_2)-d_{t_2})+T\times(q-k-1))}(t_2)$, and $R^{+((a(t_3)-d_{t_3})+T\times(r-k-1))}(t_3)$, which are denoted by A, B, and C, respectively. Notice that we only consider when $k \leq p$ and assume that the three paths from s to t_1, t_2, and t_3 are not merged yet (i.e., no sharing of registers from $k + 1$ to r). Otherwise, the situation will fall into case 1 or case 2, as discussed previously.

There are four distinct subcases. First, A, B, and C are disjoint. It means that the kth register cannot be shared and the algorithm will introduce three new internal nodes to represent the registers and continues with the next internal node f_{k-1}. Second, A, B, and C overlap with each other. It means that the kth register can be shared among t_1, t_2, and t_3. The algorithm will introduce a single internal node to represent the register and continues. The correctness of the algorithm in these two cases is trivial and will not be elaborated.

The third subcase is, without loss of generality, that $A \cap B \neq \emptyset$ and $B \cap C \neq \emptyset$, but $A \cap C = \emptyset$. Denote the region $A \cap B$ as R_{AB} and the region $B \cap C$ as R_{BC}. There are three possible options that the algorithm can choose from when evaluating the kth register: 1) It does not share the kth register and introduces three different registers for the sinks. 2) It shares the kth register between t_1 and t_2 but a separate one for t_3. 3) It shares the kth register between t_2 and t_3 but a separate one for t_1. Our algorithm will choose arbitrarily between options 2) and 3)

(since R_{AB} and R_{BC} have the same size and their order in $Y_{\min}$ is arbitrary), but it will never choose option 1). We assume that the algorithm chooses option 2) in the following analysis.

First, we compare the choices of options 1) and 2). Notice that option 1) can be better than option 2) only when the three separate paths can be merged together at a subsequent step when register h is being processed where $1 \leq h < k$, while the combined path of t_1 and t_2, and the path of t_3 cannot be merged at the hth register. We are going to show that this will not happen.

If we choose option 1), suppose that there exists a point x on the plane such that $x \in A_j \cap B_j \cap C_j$, where A_j, B_j, and C_j represent the j times T-extended regions of A, B, and C, respectively, during a subsequent step when register h is being processed where $1 \leq h < k$. By Lemma 4, it is shown that $x \in R^{+T}((R_{AB})_{j-1})$, where $(R_{AB})_{j-1}$ is the $(j-1)$ times T-extended region from R_{AB}. This means that, if it is possible to share the hth register among the three edges without sharing the kth register at the first place, by choosing option 2), i.e., to share the kth register between t_1 and t_2, the algorithm will also be able to share the hth register among the edges. Therefore, option 2) is better than option 1) by sharing more registers.

Next, we compare the choices of options 2) and 3) similarly. Consider, at a subsequent step, when register h is being processed where $1 \leq h < k$. Suppose that we choose option 3) and there exists a point x on the plane such that $x \in A_j \cap (R_{BC})_j$, where A_j and $(R_{BC})_j$ represent the j times T-extended regions of A and R_{BC}, respectively. Obviously, there exists a point y that is covered by $A_j \cap B_j \cap C_j$, i.e., $y \in A_j \cap B_j \cap C_j$. By Lemma 4, $y \in R^{+T}((R_{AB})_{j-1})$, i.e., $y \in R^{+T}((R_{AB})_{j-1}) \cap C_j$, so the hth register can also be shared among the three edges by choosing option 2). Therefore, option 2) is no worse than option 3). As a result, the algorithm will find the optimal solution by choosing arbitrarily either option 2) or option 3).

Finally, if two pairs of the regions overlap while the other is disjoint, i.e., $A \cap B \neq \emptyset$ but $A \cap C = \emptyset$ and $B \cap C = \emptyset$, the analysis is similar to the third subcase. ■

B. Register Placement

In this section, we discuss how registers are actually placed using the topology tree that is yielded from the algorithm $TopTree()$. Since some parts of the chip are occupied, we need to know where on the chip a register can be placed. To tackle this problem, we divide the chip into a mesh of $m \times n$ grids. For each grid g_{ij}, we keep track of its center coordinates $(x_{g_{ij}}, y_{g_{ij}})$ and the size of the free space in the grid $F(g_{ij})$.

Given a topology tree Υ_N, choose arbitrarily an internal node f to be the root of Υ_N, and direct the edges of Υ_N away from f. Starting from root f, we choose a grid whose center is contained in $D(f)$, i.e., the candidate region for placing register f, and it has the largest free space available. We denote this grid as $g(f)$. If $F(g(f)) \geq z$, where z denotes the size of a register, we take the center of $g(f)$ as the position of register f. Otherwise, we allow a controlled degree of inaccuracy by extending $D(f)$ one grid width further, i.e., $R^{+g_w}(D(f))$, where g_w represents the width of a grid, by repeating the same process with $R^{+g_w}(D(f))$ instead of $D(f)$ in searching for a feasible grid for placing register f. If no such grid is found, the placement of this register is reported as unsuccessful. This could happen because the register counts may increase greatly after retiming.

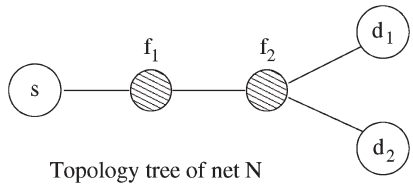


Fig. 12. Topology tree Υ_N of a three-pin net where f_1 and f_2 are two shared registers.

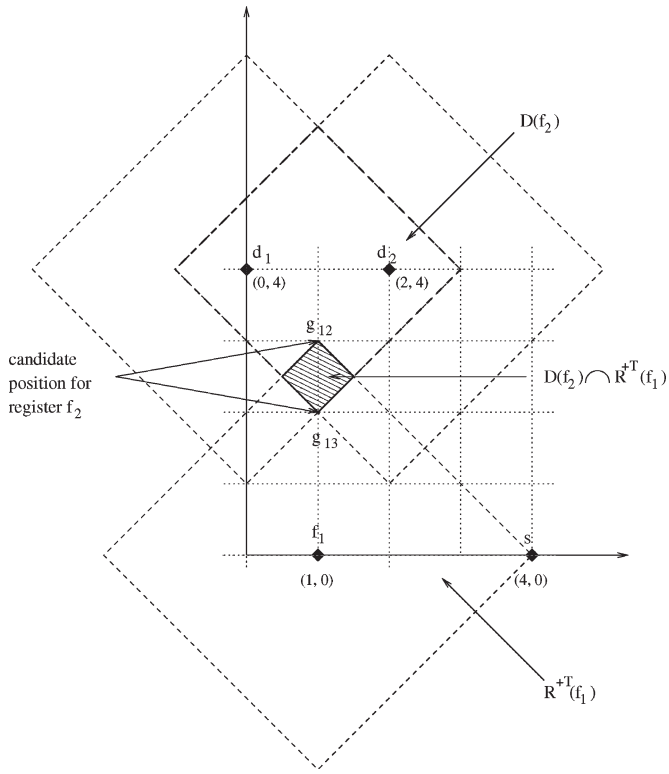


Fig. 13. Illustration of the register placement procedure.

Let $q_1, \ldots, q_m$ be the set of nodes that are the children of f in the topology tree Υ_N. After fixing the position of f, register q_j, for $1 \leq j \leq m$, is placed arbitrarily in its candidate region $D(q_j)$, provided that it is at distance T or less units from f. After visiting all the internal nodes of Υ_N, the position of each register is fixed.

Suppose that we have a three-pin net $N(s, D, L)$ and its topology tree Υ_N is shown in Fig. 12. The topology tree Υ_N shows that the two driven gates d_1 and d_2 will share two registers that are represented by internal nodes f_1 and f_2. In this example, we assume that $T = 3$ units. Consider a 5 $\times$ 5 mesh, as shown in Fig. 13, where driving gate s and two driven gates d_1 and d_2 are assumed to be at the centers of the grids containing them correspondingly, i.e., gate s is located at (4, 0), gate d_1 is located at (0, 4), and gate d_2 is located at (2, 4). Supposing that Υ_N is rooted at node f_1 and the

TABLE I
EXPERIMENTAL RESULTS OF REGISTER PLACEMENT WITH CLOCK PRESERVATION

Circuit	No. of logical regs. after retiming	Min. no. of regs. after sharing	Actual no. of regs. using our method	Reduction in area (%)	Nets with 4 or fewer edges with regs.	Nets with 5 or more edges with regs.	No. of regs. placed in candidate region	No. of regs. placed with controlled error	CPU time (s)
s641 (381)	87 (19)	53	53	8.9X	53	0	53	0	0.01
s713 (395)	94 (19)	55	55	9.9X	54	1	55	0	0.02
s820 (291)	24 (5)	23	23	0.34X	23	0	23	0	0.01
s832 (289)	23 (5)	22	22	0.35X	22	0	22	0	0.01
s1196 (531)	31 (18)	18	18	2.4X	17	1	18	0	0.00
s1238 (510)	32 (18)	18	18	2.7X	17	1	18	0	0.00
s1269 (571)	259 (74)	127	127	23X	113	14	127	0	0.02
s1488 (655)	90 (6)	73	73	2.6X	71	2	73	0	0.02
s1494 (649)	78 (6)	62	62	2.5X	60	2	62	0	0.01
s3271 (1574)	826 (232)	342	438	25X	276	18	438	0	0.18
s4863 (2344)	622 (208)	408	417	8.7X	360	22	417	0	0.25
s15850.1 (9774)	1554 (534)	1264	1264	3.0X	1203	26	1264	0	2.52
s35932 (16067)	5455 (728)	2899	2899	16X	2601	280	2899	0	13.41

algorithm has fixed its position at (1, 0), let us examine how the position of f_2 is determined. The candidate region $D(f_2)$ of f_2 covers the centers of grids g_{03}, g_{04}, g_{12}, g_{13}, g_{14}, g_{23}, and g_{24}. Then, starting from the position of f_1, the algorithm expands a rectangle of distance T from it, which is denoted by $R^{+T}(f_1)$, as shown. Next, the algorithm will find that $D(f_2) \cap R^{+T}(f_1)$ is not empty and covers the center of grid g_{12} and g_{13}–the candidate positions of register f_2. Assuming that the free space of g_{12} is greater than that of g_{13}, i.e., $F(g_{12}) \geq F(g_{13}) \geq z$, the algorithm will then assign the center of g_{12} as the position of f_2.

V. EXPERIMENTAL RESULT

We performed retiming and topology optimization on the ISCAS89 benchmark suite. The program was implemented in C language and runs on a 1.5-GHz Intel Pentium IV processor with 256 KB cache and 512 MB RAM. In our experiments, we implemented the circuits using a 0.35-μm complementary metal–oxide–semiconductor standard cell library from Austria Micro Systems, and Silicon Ensemble was used to layout the design with a setting of 50% row utilization. Gate delays were referenced from the data book, while wire lengths were estimated using the shortest Manhattan distance between the connected cells. We scaled the wire delay according to [34] in which a 1-mm wire was assumed to have a delay of 150 ps approximately. The size of a grid was set to be twice as large as a D-type flip flop. During the placement of a register, we allowed an error of one-grid width, i.e., the width of a D-type flip flop.

The results are shown in Table I. The first column indicates the names of the circuits, and the numbers shown in brackets are the total numbers of gates in the circuits $cell\_no$. The second column shows the numbers of logical registers ff_{old} in the retiming graph model after retiming, and the numbers shown in brackets are the numbers of registers in the original input circuits before retiming. The number of registers had increased after retiming for most of the circuits because the retiming method that we used did not minimize the number of registers as one of its objectives. In the third column, the minimum possible numbers of registers required after sharing are shown, i.e., assuming that every net could be realized using the best topology. The fourth column shows the numbers of registers ff_{new} that have actually been inserted after the flop topology optimization step. It can be observed that the numbers in the fourth column are the same as those in the third column, except for circuits s3271 and s4863. This observation showed that almost all the nets in our test cases could have their registers inserted using the best topology, showing that our proposed algorithm can very often find a near-optimal solution for register insertion. The fifth column shows the percentage reduction in area due to the topology optimization step. This is calculated as $(ff_{\text{old}} - ff_{\text{new}})X/cell\_no$ where X is the average ratio of the size of a register to the size of a simple gate. We can see from the fifth column that the reduction in area is about $8.1X\%$ on average. Since the size of a register is usually several times larger than that of a simple gate, the reduction in area is significant for most of the circuits. The sixth column shows the statistics of the numbers of nets containing four or fewer edges with registers, whereas the seventh column shows the numbers of nets having five or more edges with registers. The eighth column shows the numbers of registers that are placed within their candidate regions, while the ninth column shows the numbers of registers that are placed outside their candidate regions but with a controlled error range (one grid size). As we can see, all the registers are placed in their candidate regions successfully in all the test cases. Finally, the central processing unit runtime is shown in the last column.

In this set of experiments, the topology optimization step is performed on top of a retiming solution with minimum delay. For a min-area retiming solution, the circuit is retimed to minimize the total number of registers. The benefit of this topology optimization step might be less in that case since the registers will tend to be moved toward the fan-ins or the fan-outs of a gate depending on whichever is smaller in number in a min-area retiming solution, and the number of possible sharings achieved in the topology optimization step might be reduced. However, different from the min-area retiming that it minimizes the number of registers by retiming, the topology optimization step tries to reduce the register count by sharing the registers along the fan-out connections of a gate physically.

TABLE II
RUNTIME OF THE RETIMING ALGORITHMS AND THE CLOCK PERIODS OBTAINED

Circuit	No. of Nodes in V	No. of Edges in E	No. of Nodes in V_b	No. of Edges in E_H	$\frac{\lvert V\rvert\lvert E\rvert}{\lvert V_b\rvert\lvert E_H\rvert}$	CPU Time: *I-Retiming* (sec)	CPU Time: + LP = *IG-Retiming* (sec)	CPU Time: Optimal (sec)	Clock Period: T^* (ns)	Clock Period: T_{opt} (ns)	Clock Period: $\frac{T^*-T_{opt}}{T_{opt}}$ (%)
s1488	655	1405	27	627	54.36	0.09	0.19	0.28	5.62	18.85	18.82	0.16
s1494	649	1411	30	749	40.75	0.09	0.16	0.25	4.37	20.78	20.78	0.00
s3271	1574	2707	112	3360	11.32	0.38	0.71	1.09	33.70	10.24	10.24	0.00
s4863	2344	4093	154	20413	3.05	2.13	0.99	3.12	87.75	23.58	23.58	0.00
s15850	9774	13794	603	100738	2.22	21.42	2.60	24.02	1545.59	67.82	67.82	0.00
s35932	16067	28590	884	163945	3.17	54.59	6.66	61.25	8644.27	29.59	29.54	0.17

Using a 1.8GHz Intel Xeon PC with 512 KB cache and 512 MB RAM.

Therefore, it will still be beneficial to perform the optimization step on top of a min-area retiming solution.

Another set of experiments was performed to study the optimal and near-optimal retiming algorithms. In these experiments, the circuits were layout by Silicon Ensemble, and wire delays (shortest Manhattan distance) were then extracted. The lower and upper bounds of the binary search were set to 0 and 100 ns, respectively. In the near-optimal approach, we performed the procedure $I\text{-}Retiming()$ with an error bound of 1%. After assigning the registers that were aretimed into a gate to the appropriate wires, a linear program was set up to relocate the registers on the wires to get the smallest possible clock period T^*. In the optimal approach, binary search was performed until an error bound of 0.01% was obtained. We call the resulting clock period $T_{\rm opt}$. Notice that we do not need to obtain a very accurate result from $I\text{-}Retiming()$ because the solution is optimized by the linear program afterward. On average, the number of binary search iterations is 9.6 for the near-optimal approach and 16.5 for the optimal approach.

The results are shown in Table II. The second and third columns give the numbers of nodes and the numbers of edges in graph G, respectively. Notice that all circuits are not strongly connected. The numbers of nodes and edges that are listed are those after the addition of the source node, the target node, and the associated edges. The fourth and fifth columns show the numbers of nodes and the numbers of edges in the reduced graph H, respectively. These two values are dependent on the node that was chosen as the root in the depth-first traversal. In our current implementation, we always pick the additional node s as the root. We notice that using other nodes as the root does not change the result significantly. The speedup of the Bellman–Ford algorithm by the graph reduction approach in Section III-B1 is $(|V||E|)/(|V_b||E_H|)$, which is given in the sixth column. The graph reduction approach is faster in all circuits. On average, it is faster by 19.15 times. However, the speedup is less for larger circuits. The reason is that $|E_H|$ is roughly quadratic in $|V_b|$. For the circuits in Table II, the ratio of $|E_H|$ to $|V_b|^2$ is from 0.21 to 0.86, with an average of 0.55. Therefore, the graph reduction approach may not be useful for large circuits. We can avoid a slowdown of the Bellman–Ford algorithm by determining whether to use G or H based on the ratio $(|V||E|)/(|V_b||E_H|)$. $|V_b|$ and $|E_H|$ can be found in $O(|V_b||E|)$ time. We only need to perform this checking once for each circuit. Hence, the runtime overhead is insignificant compared with the total runtime. The seventh, eighth, and ninth columns show the runtime of the $I\text{-}Retiming()$ procedure, which is the time that is taken to solve the linear program and the total runtime, respectively. The tenth column shows the runtime for the optimal approach. We can see that the near-optimal approach is much more efficient than the optimal approach (particularly for large circuits). The eleventh and twelfth columns show the clock periods T^* and $T_{\rm opt}$ that were obtained by the near-optimal approach and the optimal approach, respectively. The last column is the percentage increase of T^* over $T_{\rm opt}$. The clock period that was produced by the near-optimal approach is only 0.06% more than that by the optimal approach on average.

VI. CONCLUSION

In this paper, we propose an algorithm to retime a circuit with both gate and interconnect delay and then realize the retimed circuit physically to achieve the optimal clock period. The proposed algorithm can preserve the target clock period that is obtained by retiming with a controlled error using as few registers as possible. In addition, the algorithm is proven to be giving the optimal topology for nets with four or fewer pins. Since this type of nets makes up for about 90% of the nets in a sequential circuit, on average, the algorithm performs very well and effectively under most situations. For the circuit retiming problem, we presented two elegant approaches to perform retiming on sequential circuits with both interconnect and gate delay. Our first approach is extended from the MILP approach in [3] and can solve the problem optimally. Our second approach is an improvement over the first one in terms of practical applicability. The main idea is to transform the problem into a single-source longest path problem in a reduced graph. Experimental results show that the second approach gives solutions that are only 0.06% larger than the optimal on average but in a much shorter runtime. Together with this powerful retiming method, our proposed algorithm can be applied to pipeline-long global interconnects. This is particularly useful in today's designs in which multiple clock cycles are required to propagate a signal across a global wire.

REFERENCES

[1] T. C. Tien, H. P. Su, and Y. W. Tsay, "Integrating logic retiming and register placement," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1998, pp. 136–139.

[2] I. Neumann and W. Knuz, "Layout driven retiming using the coupled edge timing model," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 7, pp. 825–835, Jul. 2003.

[3] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.
[4] C. V. Schimpfle, S. Simon, and J. A. Nossek, "Optimal placement of registers in data paths for low power design," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1997, pp. 2160–2163.
[5] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1993, pp. 398–402.
[6] A. El-Maleh, T. E. Marchok, J. Rajski, and W. Maly, "Behavior and testability preservation under the retiming transformation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 5, pp. 528–542, May 1997.
[7] S. Dey and S. Chakradhar, "Retiming sequential circuits to enhance testability," in *Proc. IEEE VLSI Test Symp.*, 1994, pp. 28–33.
[8] R. K. Ranjan, V. Singhal, F. Somenzi, and R. K. Brayton, "On the optimization power of retiming and resynthesis transformation," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1998, pp. 402–407.
[9] P. Pan, A. K. Karandikar, and C. L. Liu, "Optimal clock period clustering for sequential circuits with retiming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 6, pp. 489–498, Jun. 1998.
[10] J. Cong, H. Li, and C. Wu, "Simultaneous circuit partitioning/clustering with retiming for performance optimization," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1999, pp. 460–465.
[11] J. Cong, S. K. Lim, and C. Wu, "Performance driven multi-level and multiway partitioning with retiming," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2000, pp. 274–279.
[12] J. Cong and S. K. Lim, "Physical planning with retiming," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 2000, pp. 2–7.
[13] N. Shenoy and R. Rudell, "Efficient implementation of retiming," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1994, pp. 226–233.
[14] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming of circuits with single phase transparent latches," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1991, pp. 86–89.
[15] R. B. Deokar and S. S. Sapatnekar, "A fresh look at retiming via clock skew optimization," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1995, pp. 310–315.
[16] M. C. Papaefthymiou, "Asymptotically efficient retiming under setup and hold constraints," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1998, pp. 396–401.
[17] H. J. Touati and R. K. Brayton, "Computing the initial states of retimed circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 1, pp. 157–162, Jan. 1993.
[18] I. Karkowski and R. H. J. M. Otten, "Retiming synchronous circuitry with imprecise delay," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1995, pp. 322–326.
[19] V. Singhal, S. Malik, and R. K. Brayton, "The case for retiming with explicit reset circuitry," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1996, pp. 618–625.
[20] N. Maheshwari and S. S. Sapatnekar, "An improved algorithm for minimum-area retiming," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1997, pp. 2–7.
[21] T. Soyata and E. G. Friedmann, "Retiming with nonzero clock skew, variable register and interconnect delay," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1994, pp. 234–241.
[22] K. N. Lalgudi and M. C. Papaefthymiou, "DELAY: An efficient tool for retiming with realistic delay modeling," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1995, pp. 304–309.
[23] A. Tabbara, R. K. Brayton, and A. R. Newton, "Retiming for DSM with area-delay trade-offs and delay constraints," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1999, pp. 725–730.
[24] C. Lin and H. Zhou, "Retiming for wire pipelining in system-on-chip," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 2003, pp. 215–220.
[25] C. Lin and H. Zhou, "Wire retiming for system-on-chip by fixpoint computation," in *Proc. Des., Autom. and Test Eur. Conf. and Exhib.*, 2004, pp. 1092–1097.
[26] C. Lin and H. Zhou, "Optimal wire retiming without binary search," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 2004, pp. 452–458.
[27] C. E. Leiserson and J. B. Saxe, "A mixed-integer linear programming problem which is efficiently solvable," *J. Algorithms*, vol. 9, no. 1, pp. 114–128, Mar. 1988.
[28] E. L. Lawler, *Combinatorial Optimizations: Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.
[29] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," Ph.D. dissertation, California Inst. Technol., Pasadena, CA, 1991.
[30] M. C. Papaefthymiou, "Understanding retiming through maximum average-delay cycles," *Math. Syst. Theory*, vol. 27, no. 1, pp. 65–84, Jan./Feb. 1994.
[31] A. Dasdan, S. S. Irani, and R. K. Gupta, "Efficient algorithms for optimum cycle mean and optimum cost to time ratio," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1999, pp. 37–42.
[32] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 8th ed. New York: McGraw-Hill, 1992.
[33] Y.-Z. Liao and C. K. Wong, "An algorithm to compact a VLSI symbolic layout with mixed constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-2, no. 2, pp. 62–69, Apr. 1983.
[34] J. Cong, L. He, K. Y. Khoo, C. K. Koh, and Z. Pan, "Interconnect design for deep submicron ICs," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1997, pp. 478–485.

Dennis K. Y. Tong received the B.Eng. (Hons.) and M.Phil. degrees in computer science and engineering from Chinese University of Hong Kong, Shatin, Hong Kong, in 2002 and 2004, respectively.

He is currently with the Department of Computer Science and Engineering, Chinese University of Hong Kong. His research interests include retiming and floorplanning.

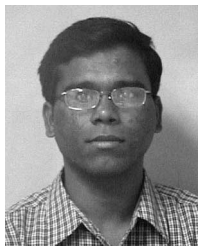
Evangeline F. Y. Young received the B.Sc. degree and M.Phil. degree in computer science from Chinese University of Hong Kong (CUHK), Shatin, Hong Kong, and the Ph.D. degree from the University of Texas, Austin, in 1999.

She is currently an Associate Professor in the Department of Computer Science and Engineering, CUHK. She is now working actively on floorplan design optimization, placement, shuttle mask floorplanning, and circuit retiming. Her research interests include algorithms and CAD of VLSI circuits.

Chris Chu received the B.S. degree in computer science from the University of Hong Kong, Hong Kong, in 1993 and the M.S. degree and the Ph.D. degree in computer science from the University of Texas, Austin, in 1994 and 1999, respectively.

He is currently an Associate Professor in the Department of Electrical and Computer Engineering, Iowa State University, Ames. His area of expertise includes CAD of VLSI physical design and design and analysis of algorithms. His recent research interests are performance-driven interconnect optimization and fast circuit floorplanning, placement, and routing algorithms.

Dr. Chu has served on the technical program committees of several major conferences including ISPD, ISCAS, DATE, ASP-DAC, and SLIP. He has also served as an organizer for the ACM SIGDA Ph.D. Forum. He was the recipient of the IEEE TCAD Best Paper Award in 1999 for his work in performance-driven interconnect optimization, the Bert Kay Best Dissertation Award for 1998–1999 from the Department of Computer Sciences in the University of Texas at Austin, and the ISPD Best Paper Award in 2004 for his work in efficient placement algorithm.



Sampath Dechu received the M.S. degree in computer engineering from Iowa State University, Ames, in 2003.

He is currently a Senior Software Engineer at Blaze DFM, Inc., Sunnyvale, CA, developing algorithms and tools for statistical timing analysis, sensitivities of various circuit parameters, such as delay and leakage power, to inter- and intradie variations. Prior to joining Blaze DFM, he was a Senior Software Engineer at Zenasis Technologies from 2004 to 2006. His research interests include statistical timing analysis and optimization, and design for manufacturing.