Optimal Wire Retiming Without Binary Search

Chuan Lin, *Student Member, IEEE*, and Hai Zhou, *Senior Member, IEEE*

***Abstract*—The problem of retiming over a netlist of macroblocks to achieve minimal clock period, where block internal structures may not be changed and flip-flops may not be inserted on some wire segments, is called the optimal wire retiming problem. This paper presents a new algorithm that solves the optimal wire retiming problem with polynomial-time worst case complexity. Since the new algorithm avoids binary search and is essentially incremental, it has the potential of being combined with other optimization techniques. Experimental results show that the new algorithm is very efficient in practice.**

***Index Terms*—Algorithms, circuit modeling, circuit optimization, design methodology, interconnects, pipelining, polynomial complexity, retiming.**

I. Introduction

With a great market drive for high performance and integration, operating frequencies and chip sizes of system-on-a-chip (SOC) are increasing dramatically. Industry data showed that the frequencies of high-performance ICs approximately doubled every process generation and the die size also increased by about 25% per generation. With such short clock periods, the communication among different blocks on an SOC circuit of ever increasing complexity is becoming a bottleneck: even with interconnect optimization techniques such as buffer insertion, the delay from one block to another may still be longer than one clock period, and multiple clock cycles are generally required to communicate such a global signal.

This trend has motivated recent research within Intel [1], [2] and IBM [3] on how to insert flip-flops on a given net if the communication between the pins requires multiple clock cycles. However, inserting flip-flops within a circuit will change its functionality, and inserting an arbitrary number of them on a net without considering global consistency will destroy the correctness of the circuit.

Retiming [4] is a traditional sequential optimization technique that moves flip-flops within a circuit while keeping its functionality. In traditional settings, retiming was used mainly on block level netlists [5]–[10]. Although some research incorporated wire delays in retiming [8], [11], [12], they did not consider the situation where multiple flip-flops may be on a global interconnect. Not until recently [13]–[16] has the alternative utility of retiming—that is, besides its computational function, a flip-flop can be used to fulfill communication buffering requirements—been explored.

Since dominant wire delays can only happen on global wires, it is more meaningful to formulate the problem at the chip level as in [13] and [15], that is, the design we deal with is a netlist of macroblocks. The wires within a block are relatively much shorter and thus do not need multiple clock periods for propagation. In SOC designs, many of the macroblocks are intellectual property (IP) cores. Some of these blocks may be combinational circuits, and others sequential. Because of the existence of predesigned blocks such as IP cores or regular-structured blocks such as memories, (combinational) buffers or flip-flops may not be inserted everywhere [17].

In this application, the approaches in [14] cannot be used because they considered only gates and cannot be extended to handle complex blocks. Our previous work [13], on the other hand, solved the problem with complex blocks by proposing timing macromodels to model the timing behavior of the blocks, based on which a set of integer difference inequalities was shown to be both necessary and sufficient, thus quantifying a feasible solution. Although it gave a polynomial-time algorithm for feasibility checking, the complexity was high, making it inhibitive even for checking circuits with about 1000 vertices. Furthermore, it only gave a fully polynomial-time approximation scheme (FPTAS) for clock period minimization, that is, the overall complexity was dependent on a given precision. The same thing was also true in our improved work [15] and in [14].

In this paper, a polynomial-time algorithm is proposed to compute the minimal clock period without using binary search. To the best of our knowledge, it is the first work that shows the optimal wire retiming problem can be solved incrementally in polynomial time.

II. Problem Formulation

We consider wire retiming on an SOC design with a given block placement (also known as floorplan) and a global routing of the global wires. We can also handle the pipelining by allowing flip-flops to be inserted at primary inputs/outputs and retimed into the circuit. This problem may come from different design methodologies and different design stages. For example, it may come from an interconnect planning stage where the floorplan and global routing are done for estimation [18], [19], or it may come from a physical design stage where the floorplan and global routing are given. The wire retiming problem in these two situations is the same except that we may allow flip-flop insertions in soft blocks during interconnect planning but only allow such insertions in preallocated buffer regions during physical design. A detailed problem formulation is given in [13]. We outline it here for the purpose of completeness.

Manuscript received August 20, 2004; revised January 6, 2005, April 6, 2005, and August 10, 2005. This work was supported by the National Science Foundation under Grant CCR-0238484. This paper was recommended by Associate Editor S. Sapatnekar.

The authors are with the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208 USA (e-mail: clin@ece.northwestern.edu).

Digital Object Identifier 10.1109/TCAD.2005.858268

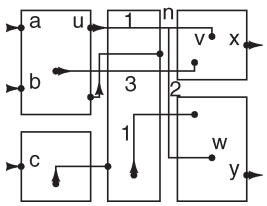


Fig. 1. Global interconnects on an SOC design.

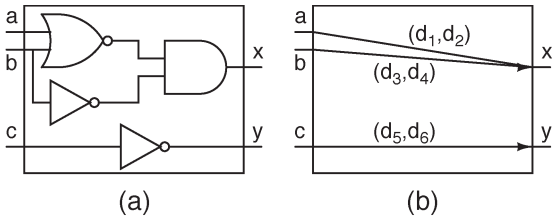


Fig. 2. Combinational block and its timing model.

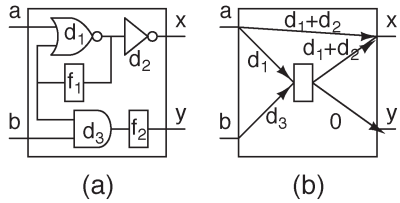


Fig. 3. Sequential block and its timing model.

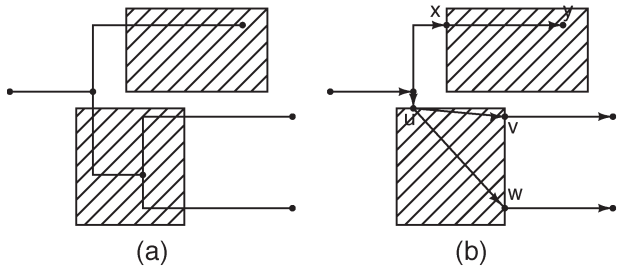


Fig. 4. Route of a net and its timing model.

As an illustration, an SOC design with a floorplan and a global routing is depicted in Fig. 1. Each wire has an arrow to indicate the signal direction and a weight to specify how many flip-flops are on the wire. Those with weight zero have the weights omitted. Some segments of a wire may not accommodate any buffer or flip-flop because they run over macroblocks that do not allow transistors to be added.

In order to take the delays within each block into consideration, timing models are used for specifying the timing behavior of each block. Due to the increasing popularity of SOC designs and IP-core-based designs, recently there are increasing research activities on timing models for macroblocks [20]–[22].

Traditional retiming is applied to logic level netlists that are composed of simple gates. In our application, the netlist is composed of macroblocks. Since a combinational block can be viewed as a complex gate, moving a flip-flop over it is simply justified. The following lemma [13] shows that retiming can be generalized to sequential blocks.

Lemma 1: In an SOC design composed of macroblocks, a flip-flop can be moved from every input to every output of a block or vice versa without changing the function of the design.

Proof: The function of the design only depends on the synchronization of the data flows. When a flip-flop is moved from every input to every output of a sequential block or vice versa, the movement only affects the specific clock cycle during which the correct results are generated. In other words, the data flows are always synchronized under such a movement of flip-flops. Therefore, the function of the design is always kept. ■

When the block is a combinational circuit, as shown in Fig. 2(a), we can use edges between inputs and outputs to represent the path delays between them, as shown in Fig. 2(b). In order to avoid flip-flops being placed on the edges we introduced in timing models, we require that the retiming tags of the input and the output connected by such an edge be the same. In addition, since we only care about the set-up conditions of flip-flops, if minimum and maximum delay pairs are given for a combinational block, only the maximum delays are taken.

The case for sequential blocks is trickier. We use the sequential block shown in Fig. 3(a) as an example. To simplify the presentation, we assume that the delays come only from gates, shown in the figure as d_1, d_2, and d_3. Since there is a combinational path of delay $d_1 + d_2$ from a to x, we introduce an edge (a, x) with delay $d_1 + d_2$ in the model. The input a has a combinational path of delay d_1 to the flip-flop f_1. This implies that the arrival time at a should not be larger than $T - d_1$, where T is the clock period. To enforce this set-up condition, we introduce a virtual flip-flop in the timing model, as shown in Fig. 3(b), and add an edge with delay d_1 from a to this virtual flip-flop. Similarly, an edge with delay d_3 is added from b to another virtual flip-flop. The concept of virtual flip-flops is also used to specify the arrival times at the block outputs that are dependent on interior flip-flops. In Fig. 3(b), all the virtual flip-flops are combined into one flip-flop. Later, the virtual flip-flop is further modeled as an edge with delay zero and weight one.

Since it is assumed that the (global) routing of nets is given, each net is represented as a Steiner tree. For example, Fig. 4(a) shows the route of a net with one source and three sinks. The shaded regions represent the areas where no buffer or flip-flop can be inserted. The timing model used for this net is shown in Fig. 4(b). Besides the sources and sinks of the net, vertices are created at points where wires are getting into or out of buffer-forbidden areas and at Steiner points not within buffer-forbidden areas. Similar to combinational paths within a block, a delay edge will be used to represent the wire delay from an entering point to an exiting point through a buffer-forbidden area.

For edges outside buffer-forbidden areas, buffers are generally assigned on appropriate positions to control the delay. According to [23], a buffer-allowable wire can be optimally buffered such that its delay becomes linear in terms of its length. Thus, in our model, the delays on buffer-allowable edges are assumed to be linear. A buffer-forbidden edge may represent a combinational path within a block or a wire over a buffer-forbidden region. In the former case, its delay is given by the pin-to-pin path delay; in the latter case, its delay can be computed as the Elmore delay of the wire under the assumption that it is buffered at region boundaries.

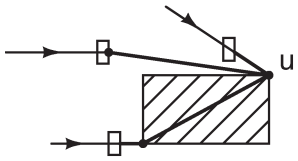


Fig. 5. Illustration of t.

Based on the above models, the problem we want to solve can be formulated as follows.

Problem (Optimal Wire Retiming): Given a directed graph $G = (V, E)$ with two types of edges, buffer-forbidden edges E_1 and buffer-allowable edges E_2 ($E = E_1 \cup E_2$), where each edge $e \in E$ has a delay $d(e)$ and a weight $w(e)$ (representing the number of flip-flops on it), find a retiming—i.e., a reposition of flip-flops in the graph—such that: 1) there is no flip-flop change on any edge $e \in E_1$; 2) the delay between two flip-flops on an edge $e \in E_2$ is linear in terms of their distance; and 3) the clock period (i.e., the maximum delay between any two consecutive flip-flops, treating primary inputs (PIs) and primary outputs (POs) as flip-flops) is minimized.

III. Notations and Constraints

From the formulation of the problem, we already have a delay label $d : E \to R^+$ and a weight label $w : E \to Z^+$. We will adopt the tradition to use a label $r : V \to Z$ to denote the number of flip-flops moved over a vertex and a label $t : V \to R$ to denote the arrival time of the vertex. For example, in Fig. 5, $t(u)$ is the maximum of the delays of the bold paths incident to u.

For any path $p \in G$, we use $d(p)$ to denote the delay along p, which is the sum of the delays of p's constituent edges. Similarly, $w(p)$ denotes the number of flip-flops on p before retiming, which is the sum of the weights of p's constituent edges. To ease the representation, we use $w_r(u, v)$ to denote the number of flip-flops on $(u, v) \in E$ after retiming, i.e.,

$$w_r(u, v) = w(u, v) + r(v) - r(u).$$

We also use $w_r(p)$ to denote the number of flip-flops on path $p = u \to v$ after retiming: $w_r(p) = w(p) + r(v) - r(u)$. When a path actually forms a cycle c, $d(c)$ $(w(c))$ includes the delay (weight) of each edge in the cycle only once. Since retiming will not change the number of flip-flops in a PI→PO path or a cycle, $w(c)$ is independent on retiming. To avoid disturbing issues regarding cycles with weight zero, we assume in this paper that $w(c) > 0$ for all $c \in G$. The cycle ratio of c is defined as $\rho(c) = d(c)/w(c)$.

A valid retiming must satisfy the constraints

$$r(u) = r(v) \qquad \forall (u, v) \in E_1 \tag{1}$$

$$w_r(u, v) \geq 0 \qquad \forall (u, v) \in E_2. \tag{2}$$

Let T denote a given clock period. Timing constraints refer to

$$t(v) = \max\left(0, \max_{\forall (u,v) \in E} t(u) + d(u, v) - w_r(u, v)T\right) \quad \forall v \in V \tag{3}$$

$$t(v) \leq T \qquad \forall v \in V. \tag{4}$$

Note that the arrival times are computed based on the retimed graph. Edge (u, v) is called a critical edge if $t(v) = t(u) + d(u, v) - w_r(u, v)T$. Likewise, critical paths and critical cycles can be similarly defined.

A solution (r, t, T) that satisfies (1)–(4) is called a feasible solution; T is called a feasible clock period. Our task is to find a feasible solution with minimal T.

Similar to [13], a vertex M is introduced into G, along with directed forbidden edges from each PO to it with zero delays and weights, and from it to each PI with delay zero but weight one. These forbidden edges ensure that if flip-flops are moved outside the circuit through PIs (POs), they will be moved back into the circuit through POs (PIs). As a result, if $r(v)$ $\forall v \in V$ is a valid retiming, then $r(v) + K$ $\forall v \in V$ is also a valid retiming for any $K \in Z$. On the other hand, due to the weight of one assigned on the forbidden edges to PIs, all PI→PO paths are transformed into cycles with positive weights, and thus will not contradict the assumption on positive cycle weights. Moreover, given that the arrival time $t(M)$ of vertex M can also be quantified by (3) and (4), we shall not differentiate M and the forbidden edges incident to it from other vertices and edges in the remainder of this paper.

We shall also clarify that the introduction of the vertex M is not necessarily required in our algorithm. In cases where flip-flops cannot be moved through PIs/POs due to restrictions on the initial state [6], [24], [25], M will not be introduced. Our proposed algorithm is guaranteed to work as long as an optimal retiming can be reached from a given flip-flop configuration (which may or may not have been modified from the original circuit) by moving flip-flops in only one direction toward either the POs or the PIs.

IV. Overview

The optimal wire retiming problem asks for a feasible solution with minimal T. To this aim, our algorithm starts with a feasible solution and iteratively reduces T to the optimal while keeping (1)–(4) satisfied.

First of all, we notice that (1) and (2) are trivially satisfied with $r(v) = 0$ $\forall v \in V$. Based on this, we can easily find a feasible solution that also satisfies (3) and (4) if T is chosen large enough.

Since T is only involved in (3) and (4), one intuitive way to reduce it is to tighten (3) and (4) under the same r for the purpose of which we use Burns' algorithm [26]. We shall always try this to reduce T. On the other hand, there must be a lower bound of T that is determined by the particular r. Once T is reduced to the lower bound by tightening (3) and (4), we will compare it with an imagined optimal solution and adjust r to approach the optimal solution. These two ways of reducing T are iteratively applied until we reach the optimal T.

V. Algorithm

A. Initialization

We want an initialization that returns a feasible solution. It is apparent that the original flip-flop configuration without any retiming must satisfy (1) and (2). A natural thought is to

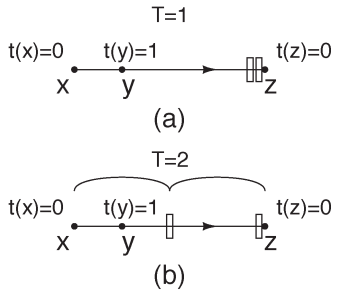


Fig. 6. Flip-flop local distribution.

INIT(G)

$t(v), f(v) \leftarrow 0, v, \forall v \in V$; $T \leftarrow 0$; $Q \leftarrow V$;
While $(Q \neq \emptyset)$ do
  $u \leftarrow$dequeue(Q);
  For each $(u,v) \in E$ with $w(u,v) = 0$ do
    If $\big(t(v) < t(u) + d(u,v)\big)$
      $t(v) \leftarrow t(u) + d(u,v)$;
      $f(v) \leftarrow f(u)$;
      If $\big(T < t(v)\big)$
        $T \leftarrow t(v)$;
      $Q \leftarrow Q \cup \{v\}$ if $v \notin Q$;
For each $(u,v) \in E$ with $w(u,v) > 0$ do
  If $\left(T < \frac{t(u)+d(u,v)-t(v)}{w(u,v)}\right)$
    $T \leftarrow \frac{t(u)+d(u,v)-t(v)}{w(u,v)}$

Fig. 7. Pseudocode of initialization.

initialize it to satisfy the timing constraints (3) and (4) as well, which is easy to achieve when T is large.

To satisfy (3) and (4), we first compute t as if all flip-flops on each edge $(u,v) \in E$ are lined up immediately before v so that (4) is satisfied with $T = \max_{v \in V} t(v)$ and (3) is satisfied on edges with weight zero. But (3) might be violated on edges with positive weights. Consider the example in Fig. 6(a), where $d(x,y) = 1$, $d(y,z) = 3$, and $w(y,z) = 2$. When the two flip-flops are lined up immediately before z, the computed $t(x)$, $t(y)$, and $t(z)$ are 0, 1, and 0, respectively. However, (3) is violated on (y,z) under $T = \max(t(x), t(y), t(z)) = 1$ since $t(z) = 0 < t(y) + d(y,z) - w(y,z)T = 2$.

To fix the violation of (3), we propose to locally distribute the flip-flops on each edge and increase T accordingly. The idea is illustrated in Fig. 6(b), where one flip-flop is moved to the middle of the path from x to z and (3) is satisfied by setting $T = 2$. In general, if (3) is violated by some edge (u,v) with $w(u,v) > 0$, i.e., $t(v) < t(u) + d(u,v) - w(u,v)T$, we will increase T to $(t(u) + d(u,v) - t(v))/w(u,v)$, under which (3) is merely satisfied on (u,v). Since T is always increased while t remains unchanged, (4) will be kept. When completed, we have a feasible clock period T and the corresponding t values that satisfy (3) and (4).

In addition, we keep a label $f : V \rightarrow V$ such that if $t(v) > 0$, then $f(v)$ is the starting vertex of a critical path terminating at v with $t(f(v)) = 0$ $\forall v \in V$. The initialization procedure is summarized in Fig. 7.

B. Base Algorithm

Starting with the initialization, there are two possible ways by which T can be reduced. Since T is only involved in (3) and (4), one intuitive way to reduce it, referred to as "t-ADJUST," is to tighten (3) and (4) under a given r, i.e., without changing the number of flip-flops on each edge. The other that allows such changes is referred to as "r-ADJUST." If the current r happens to be an optimal retiming, then we can reach the optimal T by "t-ADJUST" only. Otherwise, "r-ADJUST" is needed to adjust r toward an optimal retiming and the following lemma provides a direction of the adjustment.

Lemma 2: Provided that (1)–(3) are satisfied, if $t(v) \geq T$ for some $v \in V$, then any retiming satisfying (1)–(4) but with a smaller clock period must have more than $w_r(p)$ number of flip-flops on p, where p is any critical path from $f(v)$ to v.

Proof: Consider any critical path p from $f(v)$ to v. Since $t(v) \geq T > 0$, we have $t(f(v)) = 0$ by the definition of $f(v)$. In addition, since p is critical, it means that the delay between any two consecutive flip-flops on p is T, which implies that the path delay $d(p)$ is equal to $w_r(p)T + t(v)$.

For the sake of contradiction, we assume that there exists another retiming $(\bar{r}, \bar{t}, \bar{T})$ satisfying (1)–(4) with $\bar{T} < T$ but $w_{\bar{r}}(p) \leq w_r(p)$, i.e., $\bar{r}(v) - \bar{r}(f(v)) \leq r(v) - r(f(v))$. Since $(\bar{r}, \bar{t}, \bar{T})$ satisfies (3), it must be true that

$$\bar{t}(v) \geq \bar{t}\,(f(v)) + d(p) - w_{\bar{r}}(p)\bar{T} \geq d(p) - w_{\bar{r}}(p)\bar{T}.$$

Together with the facts that $d(p) = w_r(p)T + t(v)$, $t(v) \geq T$, $\bar{T} < T$, and $w_{\bar{r}}(p) \leq w_r(p)$, the above inequality can be reduced to

$$\begin{aligned} \bar{t}(v) &\geq w_r(p)T + t(v) - w_{\bar{r}}(p)\bar{T} \\ &\geq (w_r(p) + 1)\,T - w_{\bar{r}}(p)\bar{T} \\ &> (w_r(p) + 1 - w_{\bar{r}}(p))\,T \\ &\geq T. \end{aligned}$$

Given that $(\bar{r}, \bar{t}, \bar{T})$ satisfies (4), we have $\bar{T} \geq \bar{t}(v) > T$, which contradicts $\bar{T} < T$. Therefore, $w_{\bar{r}}(p) > w_r(p)$ must be true. ■

Since the initialization returns a feasible solution satisfying (1)–(4), we can check if $(\exists v \in V : t(v) = T)$ is true. If it is true, then, by Lemma 2, we know that T is the optimal under the current r. In order to reach a possible better solution, "r-ADJUST" needs to be carried out. Otherwise, $(\forall v \in V : t(v) < T)$ is true and "t-ADJUST" is employed to tighten (3) and (4). We now describe "t-ADJUST" and "r-ADJUST" in Section V-B1 and V-B2, respectively.

1) Clock Period Reduction With r Unchanged: Since r is kept unchanged, (1) and (2) will not be violated. Our objective is to find a smaller T such that (3) and (4) are satisfied under the same r.

First of all, we identify the set of critical edges E_c, that is, $(u,v) \in E$ with $t(u) + d(u,v) - w_r(u,v)T = t(v)$. If E_c contains a critical cycle, then the current T coincides with the cycle ratio of the critical cycle, which, by the following lemma [13], implies that the current T is the solution to the optimal wire retiming problem.

Lemma 3: A feasible clock period T must satisfy

$$T \geq \rho^* = \max_{\text{cycle } c \in G} \rho(c).$$

Proof: Since T is feasible, the delay between any two consecutive flip-flops must be no greater than T in order to satisfy the set-up conditions. In other words, $w(c)T \geq d(c)$ for any cycle $c \in G$, i.e., $T \geq \max_{\text{cycle } c \in G} \rho(c)$. ■

Otherwise, $G_c = (V, E_c)$ forms a directed acyclic graph. Suppose there exists a better solution $(r, \bar{t}, \bar{T})$ under the same r satisfying (1)–(4) with $\bar{T} < T$. Let θ denote the difference, i.e., $\theta = T - \bar{T}$. Consider critical edge (u, v) with $t(u) + d(u, v) - w_r(u, v)T = t(v)$. If $\bar{T}(u) = t(u)$, then $\bar{t}(v)$ is at least $w_r(u, v)\theta$ larger than $t(v)$; otherwise, the difference between $\bar{t}(u)$ and $t(u)$ also needs to be counted into $\bar{t}(v)$. To characterize $\bar{t}$, we define $\Delta : V \rightarrow Z^+$ as

$$\Delta(v) = \begin{cases} 0, & v \text{ is a root in } G_c \\ \max\left(\Delta(v), \Delta(u) + w_r(u, v)\right), & (u, v) \in E_c. \end{cases}$$

More specifically, $\Delta(v)$ is the maximum number of flip-flops on critical paths from roots in G_c to $v\ \forall v \in V$. Thus, for a small θ, the following is true, i.e.,

$$\bar{t}(v) = t(v) + \theta\Delta(v) \qquad \forall v \in V.$$

The above relation collapses only when θ exceeds a threshold such that some of (3) and (4) is violated. The threshold is characterized below.

Note that (3) is violated only if the following inequality is true on some edge $(u, v) \in E$, i.e.,

$$\bar{t}(v) < \bar{t}(u) + d(u, v) - w_r(u, v)\bar{T}.$$

Given that $\bar{t}(v) = t(v) + \theta\Delta(v)$, $\bar{t}(u) = t(u) + \theta\Delta(u)$, and $\bar{T} = T - \theta$, the above inequality can be rewritten as

$$t(v) - (t(u) + d(u, v) - w_r(u, v)T) < \theta\left(\Delta(u) + w_r(u, v) - \Delta(v)\right).$$

Since (r, t, T) satisfies (3), the left-hand side of the above inequality is no less than 0. It follows that $\Delta(u) + w_r(u, v) - \Delta(v) > 0$ and $\theta > (t(v) - (t(u) + d(u, v) - w_r(u, v)T))/(\Delta(u) + w_r(u, v) - \Delta(v))$. Therefore, as long as $\theta \leq (t(v) - (t(u) + d(u, v) - w_r(u, v)T))/(\Delta(u) + w_r(u, v) - \Delta(v))$ on edges with $\Delta(u) + w_r(u, v) - \Delta(v) > 0$, (3) is guaranteed to hold. We then have an upper bound for θ. This process of characterization for keeping (3) is exactly the same as for Burns' algorithm [26].

To maintain (4), we need to make sure that for all $v \in V$

$$\bar{t}(v) \leq \bar{T} \Rightarrow t(v) + \theta\Delta(v) \leq T - \theta \Rightarrow \theta \leq \frac{T - t(v)}{\Delta(v) + 1}.$$

It gives another upper bound. The threshold is then the smaller of the two bounds. Given that the first bound is evaluated on noncritical edges and that $(\forall v \in V : t(v) < T)$, the value of the threshold is always positive. By reducing T by the threshold and adjusting t accordingly, we obtain a better solution satisfying (1)–(4).

```
t–ADJUST(G, r, t, T)
Do
  ▷Identify critical edges in E
  E_c ← {(u,v) ∈ E| t(v) = t(u) + d(u,v) − w_r(u,v)T};
  If E_c contains no cycle
    Topological sort G_c = (V, E_c);
    ▷Compute max #FF on critical paths from roots
    For v ∈ V in topological order of G_c do
      If v is a root in G_c
        Δ(v) ← 0;
      Else for each (u,v) ∈ E_c do
        Δ(v) ← max{Δ(v), Δ(u) + w_r(u,v)};
    ▷Compute the threshold θ for T reduction
    θ ← ∞
    For each (u,v) ∈ E do
      If (Δ(u) + w_r(u,v) > Δ(v))
        θ ← min{θ, (t(v) − (t(u) + d(u,v) − w_r(u,v)T)) / (Δ(u) + w_r(u,v) − Δ(v))};
    For each v ∈ V do
      θ ← min{θ, (T − t(v)) / (Δ(v) + 1)};
    ▷Reduce T by θ and update t accordingly
    T ← T − θ;
    For each v ∈ V do
      t(v) ← t(v) + θ · Δ(v);
While (¬(critical cycle) ∧ (∀v ∈ V : t(v) < T));
```

Fig. 8. Pseudocode of adjusting t with r unchanged.

The above process can be iteratively applied as long as no critical cycle is formed and $(\forall v \in V : t(v) < T)$ is true. The pseudocode is given in Fig. 8.

Due to the similarity with Burns' algorithm [26], procedure "t-ADJUST" has a provable complexity of $O(|V|^2|E|)$ before either the presence of a critical cycle or an evidence of $(\exists v \in V : t(v) = T)$, the former of which certifies the optimality of T while the latter necessitates "r-ADJUST."

2) Clock Period Reduction by Changing r: Suppose that the current T is not the optimal. Let (r^*, t^*, T^*) denote an optimal solution. Given that $t(v) = T$ for some $v \in V$, we know by Lemma 2 that the optimal solution requires more flip-flops on the critical path from $f(v)$ to v, that is,

$$r^*(v) - r^*\left(f(v)\right) > r(v) - r\left(f(v)\right). \tag{5}$$

We can add more flip-flops on the critical path by either increasing $r(v)$ or decreasing $r(f(v))$. No matter which one is chosen, the amount of change should only be 1 since we do not want to overadjust r. Without loss of generality, we choose to increase $r(v)$ by 1 in our implementation, that is, $r'(v) = r(v) + 1$.

However, the increase of $r(v)$ might violate some of (1)–(3). For example, Fig. 9(a) shows the flip-flop configuration around v before $r(v)$ is increased.[1] Note that the flip-flop on edge (v, x_5) actually stands right after v. We separate v and the flip-flop for the purpose of better visibility. After $r(v)$ is increased, (1) is violated on edges (x_1, v) and (v, x_2), (2) is violated on edge (v, x_3), and (3) is violated on edge (v, x_5).

[1]Cases of (v, x_2) and (v, x_3) may occur when $d(v, x_2) = d(v, x_3) = 0$, e.g., the edges incident to M, if introduced.

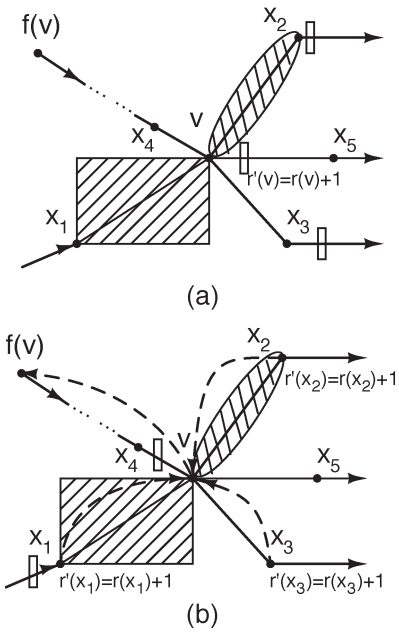


Fig. 9. Restore (1) and (2) and define m labeling.

Our idea to restore (1) and (2) is illustrated in Fig. 9(b). We first examine each edge incident to v. If (1) is violated on $(v,x) \in E_1$ or $(x,v) \in E_1$, it is actually $r'(v) = r(x) + 1$ due to the previous increase of $r(v)$. We then increase $r(x)$ by 1 to restore (1), i.e., $r'(x) = r'(v)$ with $r'(x) = r(x) + 1$. For the example in Fig. 9(b), both $r(x_1)$ and $r(x_2)$ will be increased. If (2) is violated on $(v,x) \in E_2$, it is actually $w(v,x) + r(x) - r'(v) = -1$ due to the previous increase of $r(v)$. We then increase $r(x)$ by 1 to restore (2), i.e., $w(v,x) + r'(x) - r'(v) = 0$ with $r'(x) = r(x) + 1$. For the example in Fig. 9(b), $r(x_3)$ will be increased. We do not need to consider the $(x,v) \in E_2$ case because the previous increase of $r(v)$ ensures $w(x,v) + r'(v) - r(x) \geq 1$, which will not violate (2). Vertex x_4 in Fig. 9(b) is this case. Likewise, we need to check the impact of the increase of $r(x)$, if any, on the edges incident to x, and so on. This process of checking will not stop until (1) and (2) are restored.

In our implementation, we employ a first-in-first-out (FIFO) queue rQ for the bookkeeping of the vertices whose r values are increased during the above process of restoring (1) and (2). For the example in Fig. 9(b), v, x_1, x_2, and x_3 will be queued in rQ. We claim that no vertex will be queued more than once in rQ. Otherwise, let y be the first vertex that is queued twice ($r'(y) = r(y) + 2$) in order to restore (1) and (2) on some edge incident to y. If it is a violation of (1) on $(x,y) \in E_1$ or $(y,x) \in E_1$, then the violation will be fixed after the increase of $r(y)$, that is, $r'(x) = r'(y) = r(y) + 2$. Given that (r,t,T) satisfies (1), we have $r(x) = r(y)$; hence, $r'(x) = r(x) + 2$, which contradicts the assumption that y is the first vertex whose r value is increased by 2 during the process. A contradiction can be similarly derived for the case of a violation of (2). Therefore, the claim is true and we have $r'(u) \leq r(u) + 1 \; \forall u \in V$ after (1) and (2) are restored.

A key observation on Fig. 9 is that if $r(f(v))$ is increased before the next increase of $r(v)$, then the increase of $r(v)$ is necessitated because the increase of $r(f(v))$ cancels the previous increase of $r(v)$ and makes (5) true again. The relation between $r(f(v))$ and $r(v)$ is similar to that between $t(f(v))$ and $t(v)$, where any increase of $t(f(v))$ will be propagated to $t(v)$ unless the path between them ceases to be critical. The same relation exists between $r(v)$ and $r(x_1)$, $r(v)$ and $r(x_2)$, $r(v)$ and $r(x_3)$, and $r(v)$ and $r(x_5)$.

Therefore, we introduce another label $m : V \rightarrow V \cup \{\varnothing\}$, where $\varnothing$ is the default assignment, and define it as follows. For all $u \in V$, if $r(u)$ is increased due to (5), then $m(u)$ is set to $f(u)$; if $r(u)$ is increased due to the violation of (1) and (2) on some edge between u and x, then $m(u)$ is set to x. For the example in Fig. 9(b), we will have $m(v) = f(v)$ and $m(x_1) = m(x_2) = m(x_3) = v$, represented by dashed pointers.

Based on the definition of the m labeling, we can formulate the relation between $r(m(u))$ and $r(u)$ in the following lemma.

Lemma 4: It is true before we reach an optimal retiming r^* that

$$(\forall u \in V, m(u) \in V : r^*(m(u)) - r(m(u)) \leq r^*(u) - r(u)).$$

Proof: By the definition of the m labeling, $m(u) \in V$ only if $r(u)$ has ever been increased since the initialization $\forall u \in V$. For a particular vertex u, we will show that the inequality is kept after the first increase of $r(u)$ and continues to hold before an optimal r^* is reached.

Consider the first time that $r(u)$ is increased. Vertex u is queued in rQ either because (5) is satisfied on u or for the purpose of restoring (1) and (2). For the former case, $m(u)$ will be set to $f(u)$ and (5) becomes $r^*(u) - r^*(m(u)) > r(u) - r(m(u))$ or $r^*(m(u)) - r(m(u)) < r^*(u) - r(u)$. After $r(u)$ is increased by 1, we have $r^*(m(u)) - r(m(u)) \leq r^*(u) - r(u)$. For the latter case, the increase of $r(u)$ is due to the violation of (1) and (2) on the edge between u and $m(u)$. If it is a violation of (1), then $r(u) = r(m(u))$ after (1) is restored. It follows that $r^*(m(u)) - r(m(u)) = r^*(u) - r(u)$ since $r^*(u) = r^*(m(u))$. If it is a violation of (2), then $w(m(u),u) + r(u) - r(m(u)) = 0$ after (2) is restored. Since $w(m(u),u) + r^*(u) - r^*(m(u)) \geq 0$, it follows that $r^*(m(u)) - r(m(u)) \leq r^*(u) - r(u)$.

In any case, the inequality is true after the first increase of $r(u)$. Even if $r(m(u))$ may be increased thereafter, the inequality will remain true until the next increase of $r(u)$, when $m(u)$ will be assigned to a new value that may or may not differ from the original assignment. By the same case study as above, we can show that the inequality will continue to hold. By induction, the lemma is true. ■

In addition, if there exists a sequence of vertices x_i, $i = 0, 1, \ldots, k-1$, such that $x_i = m(x_{i+1})$ and $x_k = x_0$, we refer to it as an m cycle. For the example in Fig. 9(b), if $m(f(v))$ is already equal to x_1, setting $m(v) = f(v)$ and $m(x_1) = v$ will cause an m cycle among x_1, v, and $f(v)$.

$m(u) = \varnothing$ only if $r(u)$ remains at 0 since the initialization $\forall u \in V$. This fact enables us to conclude the following lemma.

Lemma 5: $(\forall u \in V : r(u) > 0)$ implies an m cycle.

Proof: Since $r(u) > 0$, we know that $m(u) \in V$ by definition of the m labeling $\forall u \in V$. If the m labeling forms no cycle but paths, then the ending vertex u of one of the paths has to

have $m(u) = \varnothing$, which contradicts $m(u) \in V$. Therefore, the m labeling must form an m cycle. ■

The next result is a corollary to Lemma 5, which shows that an m cycle can also appear when $r(v)$ exceeds an upper bound for some vertex $v \in V$.

Corollary 5.1: $(\exists v \in V : r(v) > N_{\text{ff}})$ implies an m cycle, where $N_{\text{ff}} = \sum_{(u,v)\in E} w(u,v)$ is the total number of flip-flops in the original circuit.

Proof: Suppose (1) and (2) are satisfied (if not, we shall wait until the process of restoring (1) and (2) is completed and $r(v) > N_{\text{ff}}$ is also true at that time). For the sake of contradiction, we assume that there is no m cycle. We will show that this assumption contradicts Lemma 5 by conducting a case study. Let V' be the set of vertices that can reach v in G regardless of edge directions, including v. Suppose $u \in V'$ is such a vertex whose $r(u) = \min_{i\in V'} r(i)$. There are two cases depending on whether a direct path exists between u and v or not.

If there is no such path, we can find another vertex $x \in V'$ distinct from u and v such that all acyclic paths between v and x are in one direction, either from v to x or from x to v. If they are all from v to x, let p denote one such path. We therefore have $w_r(p) = w(p) + r(x) - r(v)$. Since all paths between v and x are from v to x, the flip-flops that were moved into p through x are different from those that were originally in p. Thus, $w(p) + r(x) \le N_{\text{ff}}$. It leads to $w_r(p) < 0$, which is impossible when (2) is kept. On the other hand, if all paths between v and x are from x to v, then $w_r(p) = w(p) + r(v) - r(x)$. By the same argument, the flip-flops that were moved into p through v must be different from those $w(p)$ number of flip-flops, that is, $w(p) + r(v) \le N_{\text{ff}}$. Thus, $r(v) \le N_{\text{ff}} - w(p) \le N_{\text{ff}}$, which contradicts $r(v) > N_{\text{ff}}$.

Therefore, there must exist a direct acyclic path[2] p between u and v. It follows that $r(u) > 0$; otherwise, more than N_{ff} number of flip-flops have to be moved into or out of p, which is impossible by retiming. In addition, V' cannot be V; otherwise, it contradicts Lemma 5. However, since the vertices in $V - V'$ have no connection with those in V' at all, the vertices in V' and the edges connecting them actually constitute an independent subcircuit $G' \subset G$. The flip-flop configuration of a subcircuit is also independent on those of others. Thus, Lemma 5 can be applied to G' and $G - G'$ separately. Since we assume no m cycle, Lemma 5 implies that $(\exists \gamma \in V' : r(\gamma) = 0)$ is true, which contradicts the fact that u is a vertex in V' with minimal r value. Therefore, we must have an m cycle. ■

The reason why an m cycle is important is because its appearance certifies the optimality of the current T.

Theorem 1: If an m cycle appears, then the current T is optimal.

Proof: Suppose the last m assignment is $m(x_0) = x_{k-1}$, by setting which the m labeling forms a cycle, that is, a sequence of vertices x_i, $i = 0, 1, \ldots, k-1$ such that $x_i = m(x_{i+1})$ and $x_k = x_0$.

For the sake of contradiction, we assume that the current T is not optimal. Let r denote the retiming before the current call of "r-ADJUST" and r' denote the retiming when the m cycle appears. Recall that "r-ADJUST" is necessitated only when T is reduced to the optimal under r by "t-ADJUST." Since T is not optimal, it follows that r is not an optimal retiming. Then, by Lemma 4 and $m(x_0) = x_{k-1}$, we have $r^*(x_{k-1}) - r'(x_{k-1}) \le r^*(x_0) - r'(x_0)$, which implies that $r^*(x_{k-1}) - r'(x_{k-1}) < r^*(x_0) - r(x_0)$ before the increase of $r'(x_0) = r(x_0) + 1$. On the other hand, Lemma 4 guarantees that $r^*(x_0) - r(x_0) \le r^*(x_1) - r'(x_1)$ and $r^*(m(x_i)) - r'(m(x_i)) \le r^*(x_i) - r'(x_i)$, $2 \le i \le k-1$. It follows that $r^*(x_0) - r(x_0) \le r^*(x_{k-1}) - r'(x_{k-1})$, which is a contradiction. Therefore, the current T is optimal. ■

Once (1) and (2) is restored, restoring (3) is straightforward. First of all, we reset $t(v) = 0 \ \forall v \in V$. After that, for edges $(y,z) \in E$ with $w_r(y,z) = 0$, we update $t(z)$ with $\max(t(z), t(y) + d(y,z))$. For others with $w_r(y,z) > 0$, if $t(y) < T$, then $t(z)$ is updated with $\max(t(z), t(y) + d(y,z) - w_r(y,z)T)$ as if all flip-flops are locally evenly distributed with delay T in between; otherwise, the first flip-flop on (y,z) will be positioned right after y and $t(z)$ is updated with $\max(t(z), d(y,z) - (w_r(y,z) - 1)T)$. The change of $t(z)$, if any, will propagate to the edges incident to z, and so on. For the example in Fig. 9(b), if $t(x_4) < T$, then the flip-flop on (x_4, v) can be moved toward v until it hits v or the delay from x_4 to the flip-flop reaches $T - t(x_4)$; otherwise, the flip-flop is placed right after x_4.

In our implementation, another FIFO queue tQ is employed to facilitate this process. When completed, it produces t values satisfying (3).

However, if the resulting t satisfies $(\exists u \in V : t(u) \ge T)$, then, by Lemma 2, we have

$$r^*(u) - r^*\left(f(u)\right) > r(u) - r\left(f(u)\right).$$

All the above operations [increasing $r(u)$ and then restoring (1)–(3)] need to be carried out again on that particular u. On the other hand, if $(\forall v \in V : t(v) < T)$, (4) is restored and we obtain a better retiming. We then switch to "t-ADJUST."

The pseudocode for adjusting r is given in Fig. 10. It turns out that Zhou's algorithm [10] for traditional retiming without binary search is a special case of Fig. 10 with $d(u,v) = 0 \ \forall (u,v) \in E_2$.

Having provided the pseudocodes of both "t-ADJUST" and "r-ADJUST" and the criterion of switching between them, we then present our algorithm in Fig. 11. The structure of the algorithm is clear; it alternates between "t-ADJUST," which reduces T to the minimum under r, and "r-ADJUST," which adjusts r to approach an optimal retiming, and terminates when either a critical cycle or an m-cycle is found.

This is an elegant result that confirms our intuition. We know that if all edges can accommodate flip-flops, that is, $E_1 = \emptyset$, then (1) is dropped and we can compute a valid retiming by simply separating every two consecutive flip-flops by a delay value equal to the maximum cycle ratio ρ^*. The corresponding arrival times will satisfy (3) and (4) under $T = \rho^*$. In other words, the optimal period is equal to ρ^*. When the cycle with ratio ρ^* becomes critical under T, we have $T = \rho^*$ and

[2]This is the only case if vertex M is introduced, provided that each vertex is reachable from some PI and can reach some PO.

```
r–ADJUST(G, r, t, T, Q)
 While ((Q ≠ ∅) ∧ ¬(m-cycle)) do
   v ←dequeue(Q);
   If (t(v) ≥ T)
     r(v) ← r(v) + 1; m(u) ← f(v);
     rQ ← {v};
     ▷Operations to restore (1)-(2)
     While (rQ ≠ ∅) do
       u ←dequeue(rQ);
       For each e = (x,u) ∈ E or e = (u,x) ∈ E do
       If (((e ∈ E1) ∧ (r(x) ≠ r(u))) | | (w_r(e) < 0))
         r(x) ← r(x) + 1; m(x) ← u;
         rQ ← rQ ∪ {x} if x ∉ rQ;
     ▷Operations to restore (3)
     t(v), f(v) ← 0, v, ∀v ∈ V; tQ ← V;
     While (tQ ≠ ∅) do
       y ←dequeue(tQ);
       For each (y,z) ∈ E do
         If (w_r(y,z) = 0)
           If (t(z) < t(y) + d(y,z))
             t(z) ← t(y) + d(y,z);
             f(z) ← f(y);
             tQ ← tQ ∪ {z} if z ∉ tQ;
         Elsif (t(y) < T)
           If (t(z) < t(y) + d(y,z) − w_r(y,z)T)
             t(z) ← t(y) + d(y,z) − w_r(y,z)T;
             f(z) ← f(y);
             tQ ← tQ ∪ {z} if z ∉ tQ;
         Elsif (t(z) < d(y,z) − (w_r(y,z) − 1)T)
           t(z) ← d(y,z) − (w_r(y,z) − 1)T;
           f(z) ← f(y);
           tQ ← tQ ∪ {z} if z ∉ tQ;
         ▷Checking (4) for further adjustments
         If (t(z) ≥ T)
           Q ← Q ∪ {z} if z ∉ Q;
```

Fig. 10. Pseudocode of adjusting r.

```
Algorithm  Incremental Optimal Wire Retiming
Input : A graph representation G = (V,E).
Output: A retiming with minimal clock period.

   INIT(G); T* ← T; r*(u) ← 0, ∀u ∈ V;
   While (¬(critical cycle) ∧ ¬(m-cycle)) do
     Q ← {v}, ∀v ∈ V with t(v) = T;
     If (Q = ∅)
        t–ADJUST(G, r, t, T);
        Update T* and r* with T and r;
     Else
        r–ADJUST(G, r, t, T, Q);
   Return T* and r*;
```

Fig. 11. Pseudocode of retiming algorithm.

T is optimal. However, in the presence of forbidden edges, separating every two consecutive flip-flops with delay ρ^* may result in some flip-flop being placed on a forbidden edge, which is prohibited. Consequently, the optimal period T^* will be larger than ρ^*. When T is reduced to T^*, adding more flip-flops on critical paths will not help to further reduce T. In fact, the requirement for more flip-flops on one critical path will be propagated through forbidden edges to other critical paths, which is exactly captured by m labeling. When an m cycle is formed, the total number of flip-flops in the m cycle is a constant, independent of retiming. In other words, the requirement of more flip-flops in an m cycle cannot be fulfilled by retiming. Therefore, the current T cannot be further reduced; it is the optimal.

C. Correctness Proof and Computational Complexity

We prove the correctness of the algorithm by showing that it returns an optimal solution when it terminates.

Theorem 2: The algorithm in Fig. 11 returns a retiming with minimal clock period when it terminates.

Proof: The algorithm terminates only if a critical cycle is found in procedure "t-ADJUST" or an m cycle is found in "r-ADJUST." By Lemma 3 and Theorem 1, T is optimal, and the algorithm will return it along with a corresponding optimal retiming. ■

We finally present the computational complexity of the algorithm to show that it terminates in polynomial time.

Theorem 3: The algorithm in Fig. 11 terminates in $O(|V|^3|E|N_{\text{ff}})$ time, where N_{ff} is the total number of flip-flops in the original circuit.

Proof: First of all, due to the similarity with Burns' algorithm [26], procedure "t-ADJUST" has a provable complexity of $O(|V|^2|E|)$ before either the presence of a critical cycle or an evidence of $(\exists v \in V : t(v) = T)$. The latter actually triggers procedure "r-ADJUST." Since no vertex is queued more than once in rQ, restoring (1) and (2) takes $O(|V||E|)$ time. Restoring (3) also takes $O(|V||E|)$ time. Therefore, the complexity between two consecutive r increases is bounded by $O(|V|^2|E|)$.

On the other hand, by Corollary 5.1, the total number of r increases before the emergence of an m cycle is bounded by $O(|V|N_{\text{ff}})$. Given that the initialization takes $O(|V||E|)$ time, we can conclude that the algorithm terminates in $O(|V|^3|E|N_{\text{ff}})$ time with either a critical cycle or an m cycle. ■

Theorems 2 and 3 together also confirm that checking critical cycle and m cycle is sufficient in order to determine the optimality of T.

Remark 1: The significance of Theorem 3 is not the actual formula of the bound, but showing that the optimal wire retiming problem can be indeed solved in polynomial time without using binary search. Furthermore, caution should be used on this bound. Firstly, a program usually has large running time variations on different problem instances. The worst case running time may only happen on a few rare instances, and thus it may not be a good indication of the efficiency on most other instances. For example, the worst case assumes that the algorithm terminates when all $r(v)$s $\forall v \in V$ are increased to N_{ff}, which is hardly the case in real applications. Second, even if the worst case does occur on most problem instances, a bound may be loose due to the difficulty of carrying out an accurate analysis. For example, reaching the minimal period under a given r typically takes $O(|E|)$ time on real circuits. These assumptions lead to an ideal worst case that is very unlikely to be attainable in reality. In other words, the bound in Theorem 3 is very loose. Since only necessary operations are conducted in the algorithm, it should be efficient on most instances. This is confirmed by our experimental results in Section VI.

TABLE I
MINIMAL CLOCK PERIOD

Circuit	$\|V\|$	$\|E\|$	N_{ff}	ρ^*	w/o non-CB blocks		w/ non-CB blocks		
					No.Step	T^*	No.Part	No.Step	T^*
s386	519	700	6	51.0	37	51.1	50	1	55.0
s400	511	665	21	32.2	125	32.2	50	1	50.6
s444	557	725	21	35.0	312	35.2	40	1	63.2
s838	1299	1206	32	76.0	2	76.0	130	1	84.0
s953	1183	1515	29	60.6	39	60.6	110	2	69.5
s1488	2054	2780	6	70.1	108	70.6	200	1	73.3
s1494	2054	2792	6	76.8	63	76.9	160	1	79.9
s5378	7205	8603	179	111.0	50	111.2	500	1	115.3
s13207	19816	22999	669	239.5	270	239.5	1000	1	292.8
s35932	46097	58266	1728	148.3	125	148.3	2000	1	163.2
s38584	53473	66964	1452	203.9	341	204.0	2000	1	264.0

D. Speed-Up Technique (Better Initialization)

Let T_0 and T_0^* denote the clock period returned by the initialization and the minimal clock period under $r(v) = 0\ \forall v \in V$, respectively. Intuitively, the closer T_0 is to T_0^*, the smaller the number of iterations the algorithm has to take to reach T_0^*. Recall that in the initialization procedure we proposed in Fig. 7, flip-flops are not allowed to be distributed until the arrival times of all vertices are obtained. This may result in too conservative a value of T_0. For instance, a circuit with only one edge (u, v) and $w(u, v) = 1$ will be initialized with $T_0 = d(u, v)$. However, if we distribute flip-flops simultaneously during the computation of t, for this case we will place the flip-flop in the middle of (u, v), then the circuit ends up being initialized with $T_0 = d(u, v)/2$, which is exactly the optimal.

More specifically, the idea to compute t and distribute flip-flops simultaneously is as follows. For each edge $(u, v) \in E$, its flip-flops are first distributed with delay T in between, where T is the current maximum of t. Then, we check if the resulting $t(v) = \max(t(v), t(u) + d(u, v) - w_r(u, v)T)$ exceeds T. If it does exceed T, then both T and $t(v)$ are updated to $(t(u) + d(u, v))/(w_r(u, v) + 1)$, that is, flip-flops on (u, v) are evenly distributed.

When completed, it gives a much better T_0. However, (3) might not be satisfied because the flip-flops that are distributed earlier may have less than T_0 in between. As a remedy, the operations used in Fig. 10 to restore (3) are employed. The resulting t will not violate (4). By doing so, we obtain a better T_0 within the same complexity as for Fig. 7.

VI. EXPERIMENTAL RESULTS

We implemented the algorithm in a PC with a 2.4-GHz Intel Xeon CPU, 512-KB cache, and 1-GB RAM. In order to give a comparison with the results of our previous works [13], [15], we use the same test files, which are modified from ISCAS-89 suite with random delay assignment—1.0 and 2.0 units to gates (treated as macroblocks) and 0.2–5.0 to wires. To reflect the impact of global interconnects on an SOC design, the delay range is intentionally chosen in order for the wire delays to be commensurate or even many times larger than the block delays.

Generally speaking, different blocks can be classified into two categories: complete bipartite and noncomplete bipartite. A block is complete bipartite only if each connected component is a complete bipartite graph in its timing model. For example, the block in Fig. 2 is complete bipartite. It becomes noncomplete bipartite if we add an additional path from c to x.

Even though gates are treated as blocks, they can only be complete bipartite as each gate has only one output. To further test the cases with noncomplete bipartite (denoted as "non-CB" in Tables I and II) blocks, we apply hMETIS [27] to partition a circuit into groups. All edges inside a group are then treated as forbidden edges. For simplicity, we did not further apply our timing model to partitions when generating the results. The number of partitions of a circuit, which is denoted as "No.Part" in Table I, plays a key role in determining the percentage of noncomplete bipartite blocks. To better reflect the influence of noncomplete bipartite blocks on the optimal clock period, we intentionally choose these numbers in order for the resulting differences to be significant.

In Table I, we report the computed minimal clock period for each circuit.[3] They match the results in both [13] and [15]. The lower bound ρ^* defined in Lemma 3 is also reported as a comparison. Since we approach the optimal clock period by gradual reduction in the algorithm, we also report the total number of occurrences that T is successfully reduced during "t-ADJUST" in the column of "No.Step." Note that, although we did not change the topology of the circuit after partitioning, we have forced the type of the edges within a group to E_1. The consequent configuration of edges has little to do with that before partitioning. We report them in one table only to share the basic circuit information, such as $|V|$, $|E|$, N_{ff}, and ρ^*, and to be consistent with the report format in [13] and [15] for clear comparisons.

In Table II, we highlight the differences of running time among the algorithms in [13] and [15] (both with precision 0.1), the base algorithm in Fig. 11, and the further improved one with the proposed speed-up technique, denoted as t_{bs1}, t_{bs2}, t_{base}, and t_{new}, respectively. For a particular algorithm, we can compute the ratio of its running time to t_{new} for each test case[4] and obtain the arithmetic (geometric) mean of all the ratios

[3] Although theoretically the algorithm in Fig. 11 will reach the exact solution without being provided a precision, we have to consider the impact of floating point error introduced by practical finite precision arithmetic due to the divisions involved in the computation of θ in Fig. 8. In our experiments, we set the error to be 0.001, which is consistent with the works in [13] and [15].

[4] If the numerator is "-" or both the numerator and the denominator are "0.00," the case is ignored. If only the denominator is "0.00," it is treated as "0.01" for estimation.

TABLE II
RUNNING TIME COMPARISON WITH PREVIOUS WORKS (SECONDS)

Circuit	w/o non-CB blocks				w/ non-CB blocks			
	t_{bs1}	t_{bs2}	t_{new}	t_{base}	t_{bs1}	t_{bs2}	t_{new}	t_{base}
s386	1.97	0.01	0.00	0.00	3.67	0.01	0.00	0.00
s400	1.64	0.01	0.01	0.01	3.38	0.01	0.00	0.00
s444	2.23	0.03	0.05	0.05	4.31	0.01	0.00	0.00
s838	8.79	0.03	0.00	0.00	33.42	0.02	0.00	0.00
s953	9.76	0.04	0.01	0.01	17.56	0.07	0.00	0.00
s1488	35.17	0.08	0.07	0.07	98.88	0.05	0.00	0.00
s1494	34.13	0.08	0.14	0.14	62.86	0.09	0.00	0.00
s5378	684.6	0.24	0.18	0.18	1344.74	0.29	0.00	0.00
s13207	-	1.07	3.94	4.07	-	206.52	0.02	0.06
s35932	-	18.63	4.81	5.02	-	6.19	0.17	0.21
s38584	-	7.44	17.18	117.76	-	21992.67	0.16	0.48
arith	851.3X	1.6X	1	1.7X	19610.3X	13442.9X	1	2.4X
geo	393.6X	1.1X	1	1.3X	2675.1X	23.0X	1	2.2X

in row "arith" ("geo"). The results clearly show that our new algorithm with the speed-up technique achieves multiple-order improvement over [13] and [15], and the speed-up technique contributes almost $2\times$ speed-up by average. It also confirms that the bound in Theorem 3 is loose.

Remark 2: A few sentences are worth mentioning for a better understanding of the data presented in Table II. First of all, though faster by average, t_{new} is a little slower than t_{bs2} for some cases "without non-CB blocks." This can be explained by looking into the nature of binary search. There are a couple of factors that will affect the runtime of a binary search-based algorithm. First, the smaller the precision is chosen, the more the runtime will be. Second, the runtime is highly dependent on the binary search range. As we know, checking an infeasible period usually takes more time than checking a feasible one. Consequently, if the upper bound is chosen to be slightly larger than the optimal, then during most of the binary search iterations the algorithm will be checking infeasible periods, which corresponds to the worst case runtime scenario. On the other hand, the best case scenario corresponds to the situation when the lower bound is slightly smaller than the optimal then most binary search iterations will be on feasible periods. Our experiments on test cases "without non-CB blocks" happened to be the best scenario. That explains why t_{bs2} is more efficient in some cases. Another phenomenon that needs to be explained is that the running times of the new algorithm actually decrease (to almost 0) when non-CB blocks are included, which counters the trend revealed by both t_{bs1} and t_{bs2}. The increasing trend of t_{bs1} and t_{bs2} can be explained by the higher binary search upper bounds caused by non-CB blocks and the fact that for a significant number of iterations the binary search based algorithms are now checking infeasible periods. For the proposed new algorithm, however, since it always works on feasible periods, the running time per iteration is much less. If the initialization happens to generate a feasible solution that is close to the optimal, then the number of iterations to reach the optimal will be very few, as it is the case in our experiments shown in Table I. Besides, the design methodology behind the new algorithm is to make it as efficient as possible. Three FIFO queues employed in the implementation are for this purpose.

Interestingly, Chu *et al.* [14] also considered the retiming problem with interconnect delays. However, they formulated their problem at block level, that is, only gates exist in the circuits. Therefore, they handled one issue (interconnect delay) in our work but not the other (retiming over blocks and buffer-forbidden regions). Furthermore, the approximation approach of first ignoring gate boundaries and then moving flip-flops out of gates does not work for noncomplete bipartite blocks, that is, their near-optimal algorithm is not applicable to our second set of experiments (with non-CB blocks). In this sense, they solved a different problem, even though it looks similar to our problem.

TABLE III
RUNNING TIME COMPARISON WITH THE WORK IN [14] (SECONDS)

Circuit	$\lvert V\rvert$	$\lvert E\rvert$	t_{chu}	t_{new}	No.Step
s1488′	655	1405	0.20	0.01	8
s1494′	649	1411	0.22	0.01	19
s3271′	1574	2707	0.73	0.84	901
s3330′	1791	2890	0.40	0.05	42
s3384′	1687	2782	0.52	0.40	385
s4863′	2344	4093	2.42	0.03	11
s5378′	2781	4261	0.88	0.00	1
s6669′	3082	5399	1.16	10.50	3424
s9234′	5599	8005	3.13	1.47	180
s13207′	7953	11302	6.80	0.02	1
s15850′	9774	13794	19.18	27.84	1551
s35932′	16067	28590	48.40	0.08	1
s38417′	22181	32135	64.63	210.34	5292
s38584′	19255	33010	361.68	0.12	1
arith			298.8X	1	
geo			11.7X	1	

For better comparison with their results within the first set of experiments (without non-CB blocks), we obtained their source code of the near-optimal algorithm and test files used in [14], and ran our new algorithm on their test files. We report the running time differences in Table III, where t_{chu} and t_{new} represent the running time for their near-optimal algorithm and our new algorithm with speed-up technique, respectively. Likewise, row "arith" ("geo") denotes the arithmetic (geometric) mean of running time improvement. Column "No.Step" denotes the total number of occurrences that T is successfully reduced during the execution of our new algorithm. The near-optimal algorithm assumed a 1% error bound with an average 9.6 number of binary search iterations over all benchmark suites.

Note that, although all test files come from the same benchmark suite, they have different delay assignments. In their

test files, wire delays were obtained by first implementing the circuits in a 0.25-μm process, laying out the circuits by silicon ensemble, and then extracting the parasitics from the layout. Besides, since they formulated the problem at block level, each gate was treated as a vertex. Thus, for a given benchmark circuit, the generated problem size in their formulation is approximately three times smaller than ours, where each input/output of a gate is treated as a vertex instead. For example, "s5378" in Table I and "s5378′" in Table III come from the same circuit, but the number of vertices and edges for "s5378" and "s5378′" is 7205 and 8603, and 2781 and 4261, respectively. Even with problem sizes that are three times larger, Table III reveals that our new algorithm is generally more efficient than their near-optimal algorithm. For those cases where $t_{\text{new}} > t_{\text{chu}}$, the long t_{new} is mainly due to the large "No.Step," e.g., 3424 for "s6669′" and 5292 for "s38584′." In addition, since their near-optimal algorithm uses binary search, comments similar to Remark 2 can be used to explain the results in Table III.

VII. Post-Retiming Placement

As stated in Section II, the wire retiming problem is formulated at an abstract enough level that it may be used at different design stages such as interconnect planning and physical design. When flip-flops are repositioned in the retiming process, there is an issue of how they will be accommodated in the given placement. A simple solution may allow the floorplan to be modified after the wire retiming stage if there is not enough space for flip-flops. Interconnect planning renders more flexibility to this approach, but we should in general avoid the iterations between floorplanning and retiming. A better approach will estimate and allocate buffer regions during floorplanning [28], and then place buffers only in the buffer regions. Furthermore, if we assume that the buffers (not the flip-flops) on the long wires are already placed, then when a flip-flop is moved to a wire, we can replace the closest buffer with the flip-flop. This will alleviate the impact of repositioned flip-flops on the area.

A few sentences may be deemed necessary for the linear delay model of buffer-allowable edges and for our above suggestion to substitute the closest buffer by the flip-flop. By assuming a continuous number of buffers and buffer sizes, Otten and Brayton [23] showed that the delay of a wire could be made linear in terms of its length. However, even when the buffer number is an integer and the size is fixed, the delay of a wire can still be bounded by a linear function of its length. The difference between the model and the "real" delay is at most the delay of one buffer. Since only very long global interconnects need to be wire pipelined, the difference of at most one buffer delay is negligible.

VIII. Conclusion

A polynomial-time algorithm for optimal wire retiming is presented in this paper. Contrary to all previous algorithms that used binary search to check the feasibility of a range of clock periods, the new algorithm directly checks the optimality of the current feasible clock period and can thus either push down the period or certify the optimality.

The underlying idea looks into the nature of the binary search approach. At each step, the binary search gives the answer to the question of whether the current clock period is feasible. The optimality of a feasible clock period can only be established indirectly, that is, through the infeasibility of the next smaller clock period. However, in our algorithm, the question being answered at each step is whether a feasible clock period smaller than the current one exists. Since it gives the existence answer, optimality is established directly once we reach such a step that gives the answer: "No."

Besides the difference of program methodology, our algorithm has many other advantages over the binary search approach. First of all, it is polynomial time bounded. No precision is required. Second, the implementation is simpler. No upper and lower bounds are needed. It is even automatically determined by the algorithm itself how far a necessary step can proceed. Third, the algorithm is very efficient in practice, which is confirmed by the experimental results. Last, but not the least, without using binary search, our algorithm is essentially incremental and has the potential of being combined with other optimization techniques, such as gate sizing, budgeting, etc., and thus can be used in incremental design methodologies [29].

Acknowledgment

The authors would like to thank Dr. W. Shi for helpful discussions and Dr. C. Chu and Dr. F. Y. Young for providing their source code and test files for comparison. The authors also wish to acknowledge the anonymous reviewers for their constructive suggestions.

References

[1] P. Cocchini, "Concurrent flip-flop and repeater insertion for high performance integrated circuits," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, 2002, pp. 268–273.

[2] N. Akkiraju and M. Mohan, "Spec based flip-flop and buffer insertion," in *Proc. Int. Conf. Computer Design*, San Jose, CA, 2003, pp. 270–275.

[3] S. Hassoun and C. J. Alpert, "Optimal path routing in single- and multiple-clock domain systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 11, pp. 1580–1588, Nov. 2003.

[4] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. 3rd Caltech Conf.: Advanced Research VLSI*, Rockville, MD, 1983, pp. 86–116.

[5] N. Shenoy and R. Rudell, "Efficient implementation of retiming," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, 1994, pp. 226–233.

[6] G. Even, I. Y. Spillinger, and L. Stok, "Retiming revisited and reversed," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 3, pp. 348–357, Mar. 1996.

[7] R. B. Deokar and S. S. Sapatnekar, "A fresh look at retiming via clock skew optimization," in *Proc. Design Automation Conf.*, San Francisco, CA, 1995, pp. 310–315.

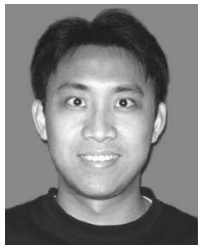
[8] K. N. Lalgudi and M. C. Papaefthymiou, "An efficient tool for retiming with realistic delay modeling," in *Proc. Design Automation Conf.*, San Francisco, CA, Jun. 1995, pp. 304–309.

[9] P. Pan, A. K. Karandikar, and C. L. Liu, "Optimal clock period clustering for sequential circuits with retiming," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 6, pp. 489–498, Jun. 1998.

[10] H. Zhou, "Deriving a new efficient algorithm for min-period retiming," in *Proc. Asian and South Pacific Design Automation Conf.*, Shanghai, China, 2005, pp. 990–993.

[11] T. Soyata, E. G. Friedman, and J. H. Mulligan, "Incorporating interconnect, register, and clock distribution delays into the retiming process,"

IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 16, no. 1, pp. 105–120, Jan. 1997.
[12] J. Cong and S. K. Lim, "Physical planning with retiming," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 2000, pp. 2–7.
[13] H. Zhou and C. Lin, "Retiming for wire pipelining in system-on-chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 9, pp. 1338–1345, Sep. 2004.
[14] C. Chu, E. F. Y. Young, D. K. Y. Tong, and S. Dechu, "Retiming with interconnect and gate delay," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, 2003, pp. 221–226.
[15] C. Lin and H. Zhou, "Wire retiming for system-on-chip by fixpoint computation," in *Proc. DATE*, Paris, France, 2004, pp. 1092–1097.
[16] D. K. Y. Tong and E. F. Y. Young, "Performance-driven register insertion in placement," in *Proc. Int. Symp. Physical Design*, Phoenix, AZ, 2004, pp. 53–60.
[17] H. Zhou, D. F. Wong, I.-M. Liu, and A. Aziz, "Simultaneous routing and buffer insertion with restrictions on buffer locations," in *Proc. Design Automation Conf.*, New Orleans, LA, 1999, pp. 96–99.
[18] R. Lu, G. Zhong, C. K. Koh, and K. Y. Chao, "Flip-flop and repeater insertion for early interconnect planning," in *Proc. DATE*, Paris, France, 2002, pp. 690–695.
[19] R. Lu and C. K. Koh, "Interconnect planning with local area constrained retiming," in *Proc. DATE*, Munich, Germany, 2003, pp. 442–447.
[20] A. J. Daga, L. Mize, S. Sripada, C. Wolff, and Q. Wu, "Automated timing model generation," in *Proc. Design Automation Conf.*, New Orleans, LA, 2002, pp. 146–151.
[21] C. W. Moon, H. Kriplani, and K. P. Belkhale, "Timing model extraction of hierarchical blocks by graph reduction," in *Proc. Design Automation Conf.*, New Orleans, LA, 2002, pp. 152–157.
[22] M. Foltin, B. Foutz, and S. Tyler, "Efficient stimulus independent timing abstraction model based on a new concept of circuit block transparency," in *Proc. Design Automation Conf.*, New Orleans, LA, 2002, pp. 158–163.
[23] R. H. J. M. Otten and R. Brayton, "Planning for performance," in *Proc. Design Automation Conf.*, San Francisco, CA, 1998, pp. 122–127.
[24] S. Dey, M. Potkonjak, and S. G. Rotweiler, "Performance optimization of sequential circuits by eliminating retiming bottlenecks," in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, 1992, pp. 504–509.
[25] V. Singhal, S. Malik, and R. K. Brayton, "The case for retiming with explicit reset circuitry," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 1996, pp. 618–625.
[26] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," Ph.D. dissertation, Comput. Sci. Dept., California Inst. Technol., Pasadena, 1991.
[27] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. Design Automation Conf.*, Anaheim, CA, 1997, pp. 526–529.
[28] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, 1999, pp. 358–363.
[29] J. Cong, O. Coudert, and M. Sarrafzadeh, "Incremental CAD," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, 2000, pp. 236–243.



Chuan Lin (S'05) received the B.S. degree in electrical engineering from Tsinghua University, Beijing, China, in 2002. He is currently working toward the Ph.D. degree in computer engineering at Northwestern University, Evanston, IL.

His research interests are on very large scale integration (VLSI) computer-aided design, especially deep submicron physical design.

Hai Zhou (M'04–SM'04) received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer sciences from the University of Texas, Austin, in 1999.

Before joining the faculty of Northwestern University, Evanston, IL, he was with the Advanced Technology Group, Synopsys, Inc. He is currently an Assistant Professor of Electrical and Computer Engineering at Northwestern University. His research interests include very large scale integrated computer-aided design, algorithm design, and formal methods.

Prof. Zhou served on the technical program committees of the ACM International Symposium on Physical Design and the IEEE International Conference on Computer-Aided Design. He was the recipient of the CAREER Award from the National Science Foundation in 2003.