

# Wire Retiming as Fixpoint Computation

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

**Abstract**—In system-on-chips (SOCs), a nonnegligible part of operation time is spent on global wires with long delays. Retiming—that is moving flip-flops in a circuit without changing its functionality—can be explored to pipeline long interconnect wires in SOC designs. The problem of retiming over a netlist of macro-blocks, where the internal structures may not be changed and flip-flops may not be inserted on some wire segments is called the wire retiming problem. In this paper, we formulate the constraints of the wire retiming problem as a fixpoint computation and use an iterative algorithm to solve it. Experimental results show that this approach is multiple orders more efficient than the previous one.

**Index Terms**—Algorithms, circuit modeling, circuit optimization, fixpoint, interconnects, retiming.

## I. INTRODUCTION

WITH a great market drive for high performance and integration scale, operating frequencies and chip sizes of system-on-chip (SOC) are dramatically increasing. Industry data showed that the frequencies of high-performance IC's 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 arbitrary number of them on a net without considering global synchronization will destroy the correctness of a 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]–[15] has the alternative utility of retiming, that is, besides its computational function, a flip-flop can fulfill communication buffering requirements, been explored.

Manuscript received August 12, 2004; revised May 30, 2005. This work was supported by the National Science Foundation under Grant CCR-0238484.

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

Digital Object Identifier 10.1109/TVLSI.2005.862726

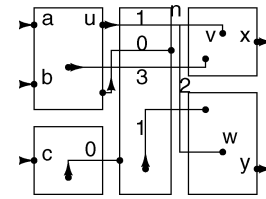


Fig. 1. Global interconnects on an SOC design.

Since dominant wire delays can only happen on global wires, it is meaningful to formulate the problem at the chip level [13], that is, the design we deal with is a netlist of macro-blocks. The wires within a block are relatively shorter and thus do not need multiple clock periods for propagation. In the SOC design, many of the macro-blocks are IP (Intellectual Property) cores. Some blocks may be combinational circuits, and others sequential. Because of the existence of pre-designed blocks such as IP cores or regular-structured blocks such as memories, (combinational) buffers or flip-flops may not be inserted anywhere [16].

For 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 macro-models for 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. To overcome this, we present an algorithm in this paper that uses iterative method to check the feasibility with great efficiency. Theoretical validity of the algorithm is based on the equivalence between the conditions of a retiming solution and a fixpoint computation.

## II. PROBLEM FORMULATION

We consider wire retiming on an SOC design with a given block placement (also known as a floorplan) and a global routing of the global wires. Detailed problem formulation can be found in [13]. We outline it here for completeness.

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 the number of flip-flops on it. Some segments of a wire may not accommodate any buffer or flip-flop because they run over macro-blocks 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. 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).

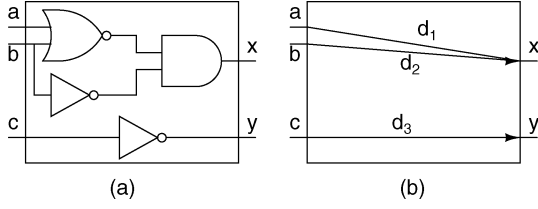


Fig. 2. A combinational block and its timing model.

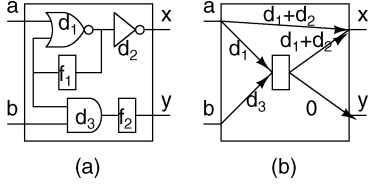


Fig. 3. A sequential block and its timing model.

Since we only care about the setup 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 harder. 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 setup 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, which is further modeled as an edge with delay 0 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 the points where wires are getting into or out of buffer forbidden areas, and at the 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 the edges outside buffer forbidden areas, buffers are generally inserted on appropriate positions to control the delay. According to [17], 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 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

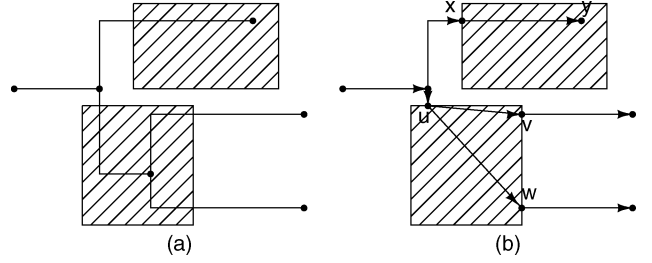
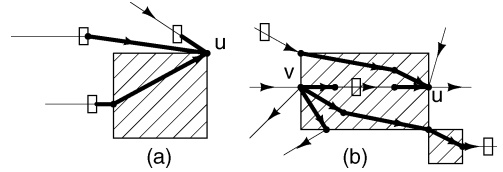


Fig. 4. Route of a net and its timing model.


 Fig. 5. Illustration of  $t$ ,  $fd$  and  $bd$ .

of the wire under the assumption that it is buffered at the region boundaries.

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

**Problem 1 (Minimum Period Wire Retiming):** Given a directed graph  $G = (V, E)$  with two types of edges  $E = E_1 \cup E_2$ , where each edge  $e \in E$  has a delay  $d(e)$  and a weight (number of flip-flops)  $w(e)$ , find a retiming—i.e., a relocation 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; 3. the clock period (i.e., the maximum delay between any two consecutive flip-flops, treating primary inputs (PI's) and primary outputs (PO's) as flip-flops) is minimized.

Problem 1 needs a retiming solution to minimize the clock period. One approach is to have a procedure to check whether a given clock period is feasible by retiming. The minimal period can be obtained by conducting a binary search.

### III. NOTATIONS AND CONSTRAINTS

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

We define  $fd(u)$  and  $bd(u)$  as

$$\begin{aligned} fd(u) &= \max_{\vec{p} \rightarrow u, \forall (x,y) \in p, (x,y) \in E_1, d(x,y) > 0} d(p), \forall u \in V, \\ bd(u) &= \max_{u \vec{p}, \forall (x,y) \in p, (x,y) \in E_1, d(x,y) > 0} d(p), \forall u \in V, \end{aligned}$$

In other words,  $\text{fd}(u)$  is the delay from  $u$ 's farthest preceding flip-flop to  $u$  through forbidden edges, and  $\text{bd}(u)$  is the delay from  $u$  to its farthest succeeding flip-flop through forbidden edges. Their physical meanings are illustrated in Fig. 5(b), where  $\text{fd}(u)$  and  $\text{bd}(v)$  are the maximum of the delays of the bold paths incident to  $u$  and  $v$ , respectively. Intuitively, they specify a range  $[\text{fd}(u), T - \text{bd}(u)]$  for each vertex  $u \in V$  such that the circuit can operate under  $T$  only if the arrival time  $t(u)$  is within this range. In case there are no forbidden edges terminating at (originating from)  $u$ , we set  $\text{fd}(u) = 0$  ( $\text{bd}(u) = 0$ ).

The requirements for a retiming solution can be stated as follows.

$$0 \leq \text{fd}(u) \leq t(u) \leq T - \text{bd}(u) \leq T, \quad \forall u \in V, \quad (1)$$

$$r(u) = r(v), \quad \forall (u, v) \in E_1, \quad (2)$$

$$t(v) \geq t(u) + d(u, v) - (w(u, v) + r(v) - r(u))T, \quad \forall (u, v) \in E. \quad (3)$$

Formula (3) actually implies the legality of a retiming, i.e., non-negative edge weights on  $(u, v) \in E_2$  after retiming.

$$\begin{aligned} & (3) \\ \xrightarrow{T \geq 0} & w(u, v) + r(v) - r(u) \geq \frac{t(u) + d(u, v) - t(v)}{T} \\ \xrightarrow{(1)} & w(u, v) + r(v) - r(u) > \frac{(\text{fd}(u) + 0 - T)}{T} \geq -1 \\ \implies & w(u, v) + r(v) - r(u) \geq 0. \end{aligned}$$

Furthermore, we can establish the following key theorem.

*Theorem 1:* Equations (1)–(3) have a solution if and only if the following (4) has a solution for all  $v \in V$ ,

$$(r(v), t(v)) = \begin{cases} (r_1(v), t_1(v)) & \text{if } t_1(v) \leq T - \text{bd}(v) \\ (r_1(v) + 1, \text{fd}(v)) & \text{otherwise} \end{cases}. \quad (4)$$

where See equations (5) and (6) at the bottom of the page.

*Proof:* ( $\leftarrow$ ): We want to show that if (4) has a solution, the solution also satisfies (1)–(3).

First of all,  $t(v)$  equals either  $t_1(v)$  or  $\text{fd}(v)$  by (4),  $\forall v \in V$ . The latter one establishes (1). The former one happens only when  $t_1(v) \leq T - \text{bd}(v)$ , which, together with  $t_1(v) \geq \text{fd}(v)$  by (6), also establishes (1). Therefore the solution satisfies (1). In fact, (2) is also satisfied, otherwise there exists an edge  $(x, y) \in E_1$  such that  $r(x) \neq r(y)$ . For the case of  $r(x) > r(y)$ , we have  $r_1(y) \geq r(x)$  by (5), thus  $r_1(y) > r(y)$ , which contradicts the fact that  $r(y) \geq r_1(y)$  by (4). A contradiction can be similarly derived for the case of  $r(x) < r(y)$ . What remains is to show that the solution also satisfies (3).

To this aim, we first rewrite (6) as

$$t_1(v) = \max \left( \text{fd}(v), \max_{(u, v) \in E} s(u, v) \right),$$

where  $s(u, v) = t(u) + d(u, v) - (w(u, v) + r_1(v) - r(u))T$ . For  $(u, v) \in E_1$ , given the facts that  $w(u, v) \geq 0$ ,  $r_1(v) \geq r(u)$  by (5),  $d(u, v) \leq \text{bd}(u)$  by the definition of  $\text{bd}(u)$ , and  $t(u) + \text{bd}(u) \leq T$  by (1), we have

$$\begin{aligned} s(u, v) & \leq t(u) + d(u, v) - (0 + r_1(v) - r(u))T \\ & \leq t(u) + d(u, v) \leq t(u) + \text{bd}(u) \leq T. \end{aligned}$$

For  $(u, v) \in E_2$ , by (5),

$$\begin{aligned} r_1(v) & \geq \left\lceil \frac{t(u) + d(u, v)}{T} - w(u, v) + r(u) \right\rceil - 1 \\ \implies r_1(v) & \geq \frac{t(u) + d(u, v)}{T} - w(u, v) + r(u) - 1 \\ \xrightarrow{T \geq 0} & t(u) + d(u, v) - (w(u, v) + r_1(v) - r(u))T \leq T \\ \implies & s(u, v) \leq T. \end{aligned}$$

Therefore,  $s(u, v) \leq T, \forall (u, v) \in E$ , thus

$$t_1(v) \leq T, \quad \forall v \in V. \quad (7)$$

Then we assume that there exists an edge  $(x, y) \in E$  on which (3) is violated, that is,

$$t(y) < t(x) + d(x, y) - (w(x, y) + r(y) - r(x))T.$$

Since it satisfies (4),  $(r(y), t(y))$  is either  $(r_1(y), t_1(y))$  or  $(r_1(y) + 1, \text{fd}(y))$ , that is, one of the following inequalities is true.

$$\begin{cases} t_1(y) < t(x) + d(x, y) - (w(x, y) + r_1(y) - r(x))T \\ \text{fd}(y) < t(x) + d(x, y) - (w(x, y) + r_1(y) + 1 - r(x))T \end{cases}.$$

By (7),  $t_1(y) \leq T$ , which implies that the second one is true only if the first is true. However, the first inequality contradicts (6). Therefore, such  $(x, y)$  does not exist and (3) is satisfied.

( $\rightarrow$ ): We assume that (1)–(3) are solvable. Since (1) and (3) are inequalities, it is not the absolute values of the variables but their differences that make sense. Among all possible solutions, we are interested in the ones that satisfy the timing validity, that is, for every PI  $u$ ,  $t(u) = r(u) = 0$ , and for all vertex  $v \in V$  other than the PI's,

$$t(v) = \max_{(u, v) \in E} t(u) + d(u, v) - (w(u, v) + r(v) - r(u))T. \quad (8)$$

Intuitively, timing validity rules out the solutions that have no physical meanings.

In addition, if  $v$  has no incident forbidden edges, it may have two equivalent satisfying assignments:  $(T, r(v))$  and  $(0, r(v) + 1)$ . Physically, it means that if a flip-flop is placed right after  $v$ , it can also be moved ahead to be immediately before  $v$ . Without loss of generality, we choose the solution that assigns  $(T, r(v))$  to  $v$ . In other words,  $0 < t(v) \leq T$  for such  $v$ . We will show in the following that the solution we quantified above is also a solution to (4). More specifically,  $r(v) = r_1(v)$  and  $t(v) = t_1(v) \leq T - \text{bd}(v), \forall v \in V$ .

$$r_1(v) = \max \left( \max_{(u, v) \text{ or } (v, u) \in E_1, r(v) \leq r(u)} r(u), \max_{(u, v) \in E_2} \left\lceil \frac{t(u) + d(u, v)}{T} - w(u, v) + r(u) \right\rceil - 1 \right), \quad (5)$$

$$t_1(v) = \max \left( \text{fd}(v), \max_{(u, v) \in E} t(u) + d(u, v) - (w(u, v) + r_1(v) - r(u))T \right). \quad (6)$$

To show  $r(v) = r_1(v)$ ,  $\forall v \in V$ , we first rewrite (5) as

$$r_1(v) = \max(r_2(v), r_3(v)),$$

where  $r_2(v)$  and  $r_3(v)$  are defined as

$$r_2(v) = \max_{\forall (u,v) \text{ or } (v,u) \in E_1, r(v) \leq r(u)} r(u),$$

$$r_3(v) = \max_{\forall (u,v) \in E_2} \left\lceil \frac{t(u) + d(u,v)}{T} - w(u,v) + r(u) \right\rceil - 1.$$

Since (1) and (3) are satisfied, for all  $(u, v) \in E$ ,

$$(3)$$

$$\xrightarrow{T \geq 0} r(v) \geq \frac{t(u) + d(u,v) - t(v)}{T} - w(u,v) + r(u)$$

$$\xrightarrow{t(v) \leq T} r(v) \geq \frac{t(u) + d(u,v)}{T} - w(u,v) + r(u) - 1$$

$$\implies r(v) \geq \left\lceil \frac{t(u) + d(u,v)}{T} - w(u,v) + r(u) \right\rceil - 1.$$

That is,

$$r(v) \geq r_3(v), \forall v \in V. \quad (9)$$

We then divide the vertices into two categories  $V_1$  and  $V_2$  depending on whether they have incident forbidden edges or not, respectively. For vertex  $v \in V_1$ , we know  $r_2(v) = r(v)$  otherwise (2) is violated. Therefore,  $r_2(v) = r(v) \geq r_3(v)$  by (9), which leads to  $r(v) = r_1(v)$  since  $r_1(v) = \max(r_2(v), r_3(v))$ . On the other hand, if  $v \in V_2$ , then  $r_2(v) = 0$ , we have  $r_1(v) = r_3(v)$ . What remains is to show that  $r(v) = r_3(v)$ . Suppose otherwise  $r(v) \neq r_3(v)$ , then  $r(v) > r_3(v)$  by (9). By the definition of  $r_3(v)$ , we know that for all  $(u, v) \in E_2$ ,

$$r(v) > \left\lceil \frac{t(u) + d(u,v)}{T} - w(u,v) + r(u) \right\rceil - 1$$

$$\implies r(v) \geq \frac{t(u) + d(u,v)}{T} - w(u,v) + r(u)$$

$$\xrightarrow{T \geq 0} t(u) + d(u,v) - (w(u,v) + r(v) - r(u))T \leq 0.$$

Thus  $t(v) = \max_{\forall (u,v) \in E} t(u) + d(u,v) - (w(u,v) + r(v) - r(u))T \leq 0$ , which contradicts the fact that  $0 < t(v) \leq T$  for such  $v$ . Therefore,  $r_1(v) = r(v)$ ,  $\forall v \in V (= V_1 \cup V_2)$ .

By replacing  $r(v)$  with  $r_1(v)$  in the timing validity (8), (6) becomes  $t_1(v) = \max(\text{fd}(v), t(v))$ . Given that  $\text{fd}(v) \leq t(v) \leq T - \text{bd}(v)$  by (1), we have  $t_1(v) = t(v) \leq T - \text{bd}(v)$ , which concludes our proof.  $\blacksquare$

#### IV. LOWER AND UPPER BOUNDS OF CLOCK PERIOD

Applying (3) on any cycle, we have the following lemma [13].  
*Lemma 1:* A feasible clock period  $T$  must satisfy

$$T \geq \max_{c \in \text{cycle}} \frac{d(c)}{w(c)}.$$

We briefly review the approach in [13] of finding a tighter upper bound of a feasible period. First of all, we find an optimal retiming solution without considering forbidden edges. This can

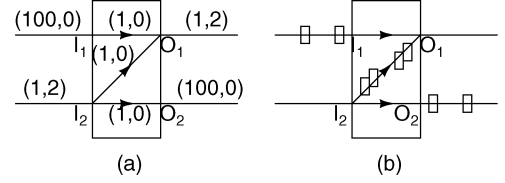


Fig. 6. A macro-block with multiple outputs.

be done based on the computation of  $\max_{c \in \text{cycle}} (d(c)/w(c))$ . Then a local adjustment to move flip-flops out of forbidden edges is done to get a feasible solution. The objective in this step is to keep the increase of the clock period as small as possible. In Fig. 6(a), an optimal retiming without considering forbidden edges is shown in Fig. 6(b) and has a period of 34. The local adjustment will move two flip-flops out from  $O_1$  and two from  $I_2$ . Therefore, our upper bound will be  $3 \times 34 = 102$ .

#### V. WIRE RETIMING UNDER A GIVEN PERIOD AS FIXPOINT COMPUTATION

In this section, we will formulate the wire retiming problem under a given clock period  $T$  as a fixpoint computation.

According to Theorem 1, (1)–(3) are equivalent to (4). For the brevity of presentation, we use a variable  $x_v$  to denote the assignment on vertex  $v \in V$ , that is,  $x_v = (r(v), t(v))$ . Then (4) can be viewed as the following equation for each vertex  $v$ :

$$x_v = f_v(x_{v_1}, x_{v_2}, \dots, x_{v_k}), \quad (10)$$

where  $v_1, v_2, \dots, v_k$  are vertices in  $V$  that have edges incident to  $v$ . A *partial order* ( $\leq$ ) can be defined between two retiming values  $x_u$  and  $x'_u$  as follows.

$$x_u \leq x'_u \stackrel{\text{def}}{=} (r(u) < r'(u)) \vee (r(u) = r'(u) \wedge t(u) \leq t'(u)).$$

In fact, a lexicographic order is defined. We use  $X$  to represent the vector  $(x_1, x_2, \dots, x_{|V|})$ , that is, the retiming information for the whole circuit. The partial order on each vertex can be extended vertex-wisely to get a partial order on the vectors: two vectors  $X = (x_1, x_2, \dots, x_{|V|})$  and  $Y = (y_1, y_2, \dots, y_{|V|})$  satisfy  $X \leq Y$  if and only if  $x_i \leq y_i$  for all  $1 \leq i \leq |V|$ . Put all  $f_v, v \in V$  together, we get a transformation for the whole system which can be written as

$$X = F(X). \quad (11)$$

There is a natural initialization to set  $t(u) = r(u) = 0$  for each PI  $u$  and  $t(v) = r(v) = -\infty$  for each vertex  $v \in V$  other than the PI's. If we regard it as the bottom element ( $\perp$ ), and another extreme assignment with  $t(v) = r(v) = \infty, \forall v \in V$  as the top element ( $\top$ ), according to the lattice theory [18], the solution space  $P$  becomes a *complete partially ordered set*. Fig. 7 shows an illustration of the solution space. For all  $X \in P$ , we have  $\perp \leq X \leq \top$ .

We know from Theorem 1 that the given period  $T$  is feasible if and only if (11) has a solution. To find a solution to (11), iterative method can be used. It starts with  $X_0 = \perp$  as the initial vector, then iteratively computes new vectors from previous ones  $X_1 =$

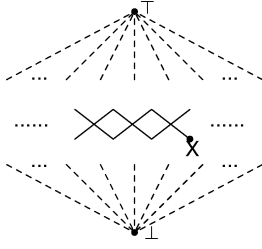


Fig. 7. Illustration of solution space.

$F(X_0), X_2 = F(X_1), \dots$  until we find a  $X_m$  such that  $X_m = X_{m-1}$ . Then  $X_m$  is a solution to (11), also called a *fixpoint* of  $F$ . We will show in the following section that (11) has a solution if and only if  $F$  is finitely convergent, or there exists a finite  $m$  such that  $F^{m-1}(X_0) = F^m(X_0)$ .

#### A. Finite Convergence

We first show that  $F$  is a *monotonic* (or *order-preserving*) transformation.

**Lemma 2:** For any  $X$  and  $X'$  satisfying  $t(u) \leq T$  and  $t'(u) \leq T$  for all  $u \in V$ , if  $X \leq X'$  then  $F(X) \leq F(X')$ .

*Proof:* We first prove that the member transformations  $f_1, f_2, \dots, f_n$  are monotonic.

Since  $X \leq X'$ , then  $x_u \leq x'_u$  for all  $u \in V$ , that is, either  $r(u) < r'(u)$  or  $r(u) = r'(u) \wedge t(u) \leq t'(u)$ . Together with  $t(u) \leq T$  and  $t'(u) \leq T$ , we have

$$\frac{t(u)}{T} + r(u) \leq \frac{t'(u)}{T} + r'(u). \quad (12)$$

Therefore  $r_1(v) \leq r'_1(v), \forall v \in V$ , by (5).

For  $r_1(v) = r'_1(v)$ , we have  $t_1(v) \leq t'_1(v)$  by (6) and (12). Consider  $f_v(X)$  and  $f_v(X')$  by (4), there are three cases. If  $t_1(v) \leq t'_1(v) \leq T - \text{bd}(v)$ , then  $f_v(X) = (r_1(v), t_1(v)) \leq (r'_1(v), t'_1(v)) = f_v(X')$ . It can be similarly shown that  $f_v(X) \leq f_v(X')$  for the other two cases of  $t_1(v) \leq T - \text{bd}(v) \leq t'_1(v)$  and  $T - \text{bd}(v) \leq t_1(v) \leq t'_1(v)$ .

For  $r_1(v) < r'_1(v)$ , we have two cases. If  $t_1(v) \leq T - \text{bd}(v)$ , then  $f_v(X) = (r_1(v), t_1(v))$ , thus  $f_v(X) \leq (r'_1(v), t'_1(v)) \leq f_v(X')$ . If  $t_1(v) > T - \text{bd}(v)$ , then  $f_v(X) = (r_1(v) + 1, \text{fd}(v))$ . Since  $r_1(v) < r'_1(v)$ , we have  $r_1(v) + 1 \leq r'_1(v)$ . Together with  $t'_1(v) \geq \text{fd}(v)$  from (6), we have  $f_v(X) \leq (r'_1(v), t'_1(v)) \leq f_v(X')$ .

Therefore,  $f_v(X) \leq f_v(X')$ , for all  $v \in V$ . By the definition of partial order on vectors,  $F(X) \leq F(X')$ . ■

The next result is a corollary of the above lemma.

**Corollary 2.1:** For any vector  $X$  and  $X'$  satisfying  $t(u) \leq T$  and  $t'(u) \leq T$  for all  $u \in V$ , if  $X \leq X'$  then  $F^m(X) \leq F^m(X')$ ,  $m \in \mathbb{Z}^+$ .

The next lemma shows that if  $F$  is not finitely convergent but  $T$  is feasible, then  $r$  will keep increasing.

**Lemma 3:** Let  $X$  denote any finitely reachable vector by the iterative method and  $X_{|V|}$  denote the vector such that  $X_{|V|} = F^{|V|}(X)$ . If  $F$  is not finitely convergent but  $T$  is feasible, then there must exist a vertex  $v \in V$  such that  $r_{|V|}(v) > r(v)$ .

*Proof:* Suppose otherwise  $r_{|V|}(v) = r(v)$  for all  $v \in V$ . Then (2) is always kept otherwise  $r(v)$  will be increased for some  $v$  by (4) and (5). We divide the vertices into two categories

$V_1$  and  $V_2$  depending on whether they have incident forbidden edges or not, respectively. Consider one of the  $|V|$  iterations.

For vertex  $v \in V_1$ , since (2) is satisfied, we have

$$r(v) = \max_{\forall (u,v) \text{ or } (v,u) \in E_1, r(v) \leq r(u)} r(u).$$

Thus  $r_1(v) \geq r(v)$  by (5). On the other hand,  $r(v) \geq r_1(v)$  by (4). Therefore  $r(v) = r_1(v)$ .

For vertex  $v \in V_2$  and any  $(u, v) \in E$ ,

$$(5) \\ \implies r_1(v) \geq \left\lceil \frac{t(u) + d(u, v)}{T} - w(u, v) + r(u) \right\rceil - 1 \\ \implies r_1(v) \geq \frac{t(u) + d(u, v)}{T} - w(u, v) + r(u) - 1 \\ \stackrel{T \geq 0}{\implies} t(u) + d(u, v) - (w(u, v) + r_1(v) - r(u))T \leq T.$$

Together with (6), we have  $t_1(v) \leq T = T - \text{fd}(v)$ , which also leads to  $r(v) = r_1(v)$  by (4).

Therefore,  $(r(v), t(v)) = (r_1(v), t_1(v))$  is always chosen for all  $v \in V$  during the  $|V|$  iterations between  $X$  and  $X_{|V|}$ . In other words,  $F$  is reduced to only updating  $t$  by (6), which is exactly what Bellman-Ford's algorithm [19] does to solve a set of inequalities. Since  $F$  is not finitely convergent, after  $|V|$  iterations, there must exist a positive cycle  $c$  in  $G$  such that  $d(c) - w(c)T > 0$  (otherwise,  $F$  converges and becomes finitely convergent since  $X$  is finitely reachable), that is,  $T < d(c)/w(c)$ . By Lemma 1,  $T$  is infeasible, which contradicts that the given  $T$  is feasible.

Therefore, there exists a vertex  $v \in V$  such that  $r_{|V|}(v) \neq r(v)$ . Since  $F$  cannot reduce  $r$ , we have  $r_{|V|}(v) > r(v)$ . ■

Based on Corollary 2.1 and Lemma 3, we can establish the finite convergence in the following theorem.

**Theorem 2:**  $F$  is finitely convergent if and only if  $T$  is feasible.

*Proof:* If  $F$  is finitely convergent, we must have found a vector  $\hat{X}$  such that  $\hat{X} = F(\hat{X})$ . By Theorem 1,  $\hat{X}$  satisfies (1)–(3), thus  $T$  is feasible.

For the other direction, since  $T$  is feasible, there exists a solution  $\hat{X} = ((\hat{r}(1), \hat{t}(1)), \dots, (\hat{r}(|V|), \hat{t}(|V|)))$  such that  $F(\hat{X}) = \hat{X}$ . Starting from  $X_0$ , at most  $|E|$  iterations are needed for each vertex  $v \in V$  to have a finite  $(r_{|E|}(v), t_{|E|}(v))$ . After that, if  $F$  is not finitely convergent, then, by Lemma 3,  $r(v)$  for some  $v \in V$  will be increased every  $|V|$  iterations. Let

$$k = |E| + |V| \sum_{i=1}^{|V|} (\hat{r}(i) - r_{|E|}(i))$$

and  $X_k$  be the vector such that  $X_k = F^k(X_0)$ . Lemma 3 implies that there exists a vertex  $v \in V$  whose  $r_k(v)$  exceeds  $\hat{r}(v)$ . However, since  $X_0 \leq \hat{X}$ , we have  $F^k(X_0) \leq F^k(\hat{X}) = \hat{X}$  by Corollary 2.1, i.e.,  $r_k(u) \leq \hat{r}(u), \forall u \in V$ , which is a contradiction. Therefore,  $F$  is finitely convergent. ■

In fact, according to the lattice theory [18], if  $F$ , defined on a complete partially ordered set, has a fixpoint, then  $F$  has a least fixpoint  $X_{lf}$ , defined as

$$(X_{lf} = F(X_{lf})) \wedge (\forall X : \perp \leq X \wedge X = F(X) : X_{lf} \leq X).$$

The next theorem shows that the iterative method will reach the least fixpoint.

**Theorem 3:** The fixpoint found by iterative method from the bottom element ( $\perp$ ) is the least fixpoint.

*Proof:* Let  $X_{lf}$  and  $X_m$  be the least fixpoint and the fixpoint found by iterative method, respectively. Since  $X_{lf}$  is the least, it must be true that  $X_{lf} \leq X_m$ . On the other hand, since  $X_0 = \perp \leq X_{lf}$ , Corollary 2.1 implies  $X_m = F^m(X_0) \leq F^m(X_{lf}) = X_{lf}$ . Therefore, we have  $X_m = X_{lf}$ . ■

## B. Chaotic Iteration

In this section we will establish that no matter what evaluation order is used, the iterative method will always converge to the same fixpoint, that is, the least fixpoint. This is called the scheme of *chaotic iteration* [20]. Here, a transformation  $F$  is composed of a set of partial transformations on each vertex. In each step, one or more partial transformations are applied to update retiming information on one or more vertices. All retiming information on other vertices is kept unchanged. We will use  $F_S$  to represent such a partial transformation done in one step, where  $S$  represents the set of vertices whose retiming information is updated. We have an immediate result.

**Lemma 4:** Given a subset of vertices  $S \subseteq V$ , if  $X \leq F(X)$ , then  $X \leq F_S(X) \leq F(X)$ .

The following lemma states that no matter what evaluation order is used, the generated sequence is monotonic in the same direction.

**Lemma 5:** Let  $X'_0(= X_0), X'_1, \dots, X'_{m'}, X'_{m'+1}(= X'_{m'})$  be the update sequence generated by a sequence of arbitrary partial transformations  $F_{S_0}, F_{S_1}, \dots, F_{S_{m'}}$ . Then  $X'_i \leq F_{S_i}(X'_i) \leq F(X'_i)$ , for all  $0 \leq i \leq m'$ .

*Proof:* We prove it by applying induction on  $i$ . Consider the base for  $i = 0$ , it is obvious that  $X'_0 \leq F(X'_0)$  since  $X'_0 = \perp$ . With “ $X$ ” replaced by “ $X'_0$ ” in Lemma 4, we have  $X'_0 \leq F_{S_0}(X'_0) = X'_1 \leq F(X'_0)$ .

Suppose that the statement is true for  $0 \leq i \leq k$ , that is,  $X'_k \leq F_{S_k}(X'_k) = X'_{k+1} \leq F(X'_k)$ . Since  $X'_k \leq X'_{k+1}$ , we have  $F(X'_k) \leq F(X'_{k+1})$  by Lemma 2. Together with the inductive hypothesis ( $X'_{k+1} \leq F(X'_k)$ ), we get  $X'_{k+1} \leq F(X'_{k+1})$ , which, by Lemma 4, leads to  $X'_{k+1} \leq F_{S_{k+1}}(X'_{k+1}) \leq F(X'_{k+1})$ . Thus the statement is also true for  $i = k + 1$ . The lemma is true. ■

Furthermore, if the evaluation order is *fair*, that is, a partial transformation will always be applied if its inputs and outputs are not consistent, then the chaotic iteration will always reach the same fixpoint as  $F$ .

**Theorem 4:** Any fair evaluation order will reach the same fixpoint as  $F$ , the least fixpoint.

*Proof:* Let  $X'_0(= X_0), X'_1, \dots, X'_{m'}$  be the update sequence generated by a sequence of arbitrary partial transformations  $F_{S_0}, F_{S_1}, \dots, F_{S_{m'-1}}$  and  $X_0, X_1, \dots, X_m$  be the one generated by  $F$ .  $X_m$  and  $X'_{m'}$  are fixpoints while any vector before them in the sequences is not. Our goal is to show that  $X_m = X'_{m'}$ .

First of all, note that if  $X'_i \leq X_i$ , then  $F(X'_i) \leq F(X_i)$  by Lemma 2, and  $F_{S_i}(X'_i) \leq F(X'_i)$  by Lemma 5, thus  $F_{S_i}(X'_i) \leq F(X_i)$ . Applying this relationship to both sequences with the fact that  $X'_0 = X_0$ , we have  $X'_i \leq X_i$  for  $0 \leq i \leq \min(m', m)$ .

$$\begin{array}{ccccccc} X'_0 \leq & X'_1 \leq & \cdots \leq & X'_m \leq & \cdots \leq & X'_{m'} \\ \parallel & \wedge & & \wedge & & \wedge \\ X_0 \leq & X_1 \leq & \cdots \leq & X_m = & \cdots = & X_{m'} \end{array}$$

Fig. 8. Update sequences by  $F$  and any fair partial transformation.

**Input:**  $x_u$ ; **Output:**  $f_v(x_u)$ .  
 If  $(r(v) < r(u))$  then  
 $r(v) \leftarrow r(u)$ ;  $t(v) \leftarrow \text{fd}(v) = 0$ ;

Fig. 9. Update operations for Case 1.

**Input:**  $x_u$ ; **Output:**  $f_v(x_u)$ .  
 If  $(r(v) < r(u))$  then  
 $r(v) \leftarrow r(u)$ ;  
 If (3) is violated, then  
 $t(v) \leftarrow \max(t(u) + d(u, v), \text{fd}(v))$ ;

Fig. 10. Update operations for Case 2.

If  $m' < m$ , then  $X'_{m'} \leq X_{m'} \leq X_m$ . Given that  $X_m$  is the least fixpoint by Theorem 3, we have  $X_m \leq X'_{m'}$ , thus  $X'_{m'} = X_{m'} = X_m$ , which contradicts the assumption that  $X'_{m'}$  is not a fixpoint. Therefore  $m \leq m'$ , which confirms the intuition that the number of iterations required by partial transformations is always larger than that by  $F$ . We then extend the sequence generated by  $F$  to the length of  $m'$  by assigning  $X_i = X_m$ , for all  $m + 1 \leq i \leq m'$ , as illustrated in Fig. 8.

It is obviously that  $X'_i \leq X_i$ ,  $m \leq i \leq m'$ . Hence,  $X'_{m'} \leq X_{m'} = X_m$ . Given that  $X_m$  is the least fixpoint, we have  $X'_{m'} = X_m$ . ■

## VI. ALGORITHM

### A. Detailed Description

Equation (4) provides a clear scheme to do the updates. However, each update of a single variable requires a series of checkings on all the adjacent vertices, which could be inefficient. Since the fixpoint is independent on evaluation orders, an alternative is to propagate an update on a vertex to its adjacent vertices, instead of determining the update from them. In this way, updates are triggered only when they are needed, and unnecessary checkings are saved.

To propagate the update of  $x_u$  out, we decompose (4) into four different cases. We list the corresponding operations for each case in Figs. 9–12, respectively. They are further illustrated in Fig. 13.

- Case 1:  $(u, v) \in E_1, d(u, v) = 0, w(u, v) = 1$ ;
- Case 2:  $(u, v) \in E_1, d(u, v) > 0, w(u, v) = 0$ ;
- Case 3:  $(u, v) \in E_2$ ;
- Case 4:  $(v, u) \in E_1$ .

It can be verified that the operations will satisfy (2) and (3) on  $(u, v)$ . Take Case 2 as an example (arguments for other cases are similar), (2) is satisfied after  $r(v)$  is assigned with the value of  $r(u)$ . Since  $w(u, v) = 0$  for  $(u, v) \in E_1$ , (3) is satisfied after the  $t(v)$  assignment. In addition, if  $t(u)$  satisfies (1), then  $t(v)$  will also satisfy (1), otherwise  $T - \text{bd}(u) + d(u, v) \geq t(u) + d(u, v) = t(v) > T - \text{bd}(v)$ , or  $\text{bd}(u) < \text{bd}(v) + d(u, v)$ , which contradicts the definition of  $\text{bd}$ .

With these explicit operations, any fair evaluation order will lead to the least fixpoint if  $F$  is finitely convergent. However,

**Input:**  $x_u$ ; **Output:**  $f_v(x_u)$ .  
 If (3) is violated then  

$$r(v) \leftarrow \left\lceil \frac{t(u)+d(u,v)}{T} + r(u) - w(u,v) \right\rceil - 1;$$

$$t(v) \leftarrow \max\left(\text{fd}(v),\right.$$

$$\left. t(u) + d(u,v) - (w(u,v) + r(v) - r(u))T\right);$$
 If  $t(v) > T - \text{bd}(v)$  then  

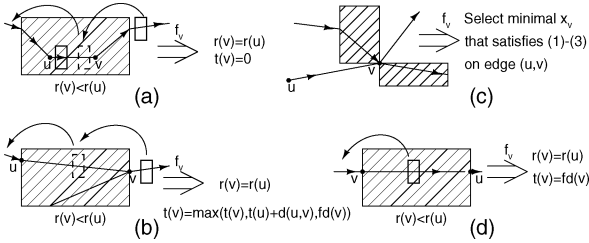
$$r(v) \leftarrow r(v) + 1; t(v) \leftarrow \text{fd}(v);$$

Fig. 11. Update operations for Case 3.

**Input:**  $x_u$ ; **Output:**  $f_v(x_u)$ .  
 If  $r(v) < r(u)$  then  

$$r(v) \leftarrow r(u); t(v) \leftarrow \text{fd}(v);$$

Fig. 12. Update operations for Case 4.

Fig. 13. Four cases to propagate the update of  $x_u$  out.

if  $T$  is infeasible, the update procedure will never stop. Therefore, we need to come up with some criteria to efficiently check the infeasibility and terminate the procedure. The following theorem provides one.

**Theorem 5:** A given  $T$  is infeasible if and only if  $(\exists v \in V : r(v) > N_{\text{ff}})$  is true during the iteration, where  $N_{\text{ff}} = \sum_{(u,v) \in E} w(u,v)$  is the total number of flip-flops in the original circuit.

**Proof:** ( $\rightarrow$ ): Suppose  $T$  is infeasible, then  $F$  has no fixpoint and will never converge. Similar to Lemma 3, we can show that  $r$  will be increased every  $|V|$  iterations. In other words,  $r(v)$  will exceed  $N_{\text{ff}}$  for some  $v \in V$  after a finite number iterations.

( $\leftarrow$ ): Suppose otherwise  $(\exists v \in V : r(v) > N_{\text{ff}})$  and  $T$  is feasible. Let  $\hat{X}$  be the least fixpoint of  $F$ . When it is reached, we have  $\hat{r}(v) \geq r(v) > N_{\text{ff}}$  by Lemma 2. Let  $V'$  be the set of vertices that can reach  $v$  in  $G$  regardless of edge directions, including  $v$ . Let  $u \in V'$  be the vertex whose  $\hat{r}(u) = \min_{i \in V'} \hat{r}(i)$ . There are two cases depending on whether a direct path exists between  $u$  and  $v$ .

If no, 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  $v$ . Suppose they are all from  $v$  to  $x$  (similar arguments apply to the other case), let  $p$  denote one such a path. We have  $\hat{w}(p) = w(p) - \hat{r}(v) + \hat{r}(x)$ . 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,  $w(p) + \hat{r}(x) \leq N_{\text{ff}}$ . Given that  $\hat{r}(v) > N_{\text{ff}}$ , it leads to  $\hat{w}(p_1) < 0$ , which is impossible for a legal retiming with nonnegative edge weights guaranteed by (3).

Therefore, a direct path exists between  $u$  and  $v$ . This implies  $\hat{r}(u) > 0$ , otherwise more than  $N_{\text{ff}}$  number of flip-flops have to be moved into or out of the path between  $u$  and  $v$ , which is impossible by retiming. However we can construct another fixpoint by decreasing all  $\hat{r}(i)$ ,  $i \in V'$ , by  $\hat{r}(u)$ , which contradicts

**Algorithm** Fixed period wire retiming  
**Input:** A graph representation  $G = (V, E)$ ,  $T$   
**Output:** A fixpoint or infeasibility report.

Initialization, add  $PI$  into  $Q$ ;  
 While  $(Q \neq \emptyset)$  do  
 $u \leftarrow \text{extract}(Q)$ ;  
 For each  $(u,v)$  or  $(v,u) \in E$  do  
 If  $(x_v \neq f_v(x_u))$  then  
 $x_v \leftarrow f_v(x_u)$ ;  
 $Q \leftarrow v$  if  $v \notin Q$ ;  
 If  $(r(v) > N_{\text{ff}})$  then  
 Return infeasibility report;  
 Return the fixpoint;

Fig. 14. Pseudocode of algorithm for feasibility checking.

Theorem 4 that the fixpoint we finally reach is the least fixpoint. Therefore,  $T$  is infeasible. ■

In Fig. 14, we present our algorithm for feasibility checking under a given fixed clock period  $T$ . It uses an auxiliary queue  $Q$  to facilitate the update procedure. Initially, we have in  $Q$  all  $PI$ 's. Vertices in  $Q$  are extracted one at a time to trigger possible updates on adjacent vertices. A vertex is inserted into  $Q$  whenever its retiming information is updated. The algorithm terminates when  $Q$  is empty or it finds  $T$  infeasible. In conjunction with binary search, it solves the minimum period wire retiming problem.

## B. Speed-Up Techniques

1) **Infeasibility Checking Criteria:** Although Theorem 5 provides a criterion for infeasibility checking, it may not be efficient for all cases. For example, if  $G$  involves a critical cycle  $c$  (a cycle that determines the optimal clock period),  $N_{\text{ff}}$  is a large number and the given  $T$  is slightly smaller than the optimal clock period, then the algorithm will spend a long time to update the critical cycle before the infeasibility is discovered by Theorem 5. In this section, we provide three additional criteria to accelerate the infeasibility checking.

The first one makes use of the history of the updates. The idea is similar to Shenoy and Rudell [5] to maintain a predecessor vertex pointer denoted by  $\pi : V \rightarrow V \cup \{\text{null}\}$  for each vertex. Whenever  $x_v$  is updated through edge  $(u,v)$  or  $(v,u)$ , we set  $\pi(v) = u$ , and reset  $\pi(x)$  to "null" for those  $x$  with  $\pi(x) = v$ . Thus, (2) and (3) are always satisfied on edge  $(\pi(v), v)$  or  $(v, \pi(v))$ ,  $\forall v \in V$ . This gives us another criterion.

**Theorem 6:** If there exists a vertex  $u$  whose consecutive updates  $x_u$  and  $x'_u$  satisfy  $t'(u) = t(u)$ , and before resetting  $\pi(v)$  to "null" for those  $v$  with  $\pi(v) = u$ , we find a cycle by following  $\pi$  from  $u$ , then  $T$  is infeasible.

**Proof:** According to Lemma 5,  $x_u \leq x'_u$ . Given that  $t'(u) = t(u)$ , we have  $r'(u) > r(u)$ , otherwise  $x'_u = x_u$  is not an update.

If the cycle involves only one vertex  $u$ , the only edge  $e$  in the cycle is not in  $E_1$  since none of Case 1 (Fig. 9), Case 2 (Fig. 10) and Case 4 (Fig. 12) will increase  $r(u)$ . For Case 3 (Fig. 11),  $r(u)$  is increased only when (3) is violated on  $e$ , that is,  $t(u) < t(u) + d(e) - w(e)T$ , or  $d(e) > w(e)T$ . By Lemma 1,  $T$  is infeasible.

For the cycle with multiple vertices, there exists a vertex  $v$  other than  $u$  such that  $\pi(v) = u$  before it is reset to "null".

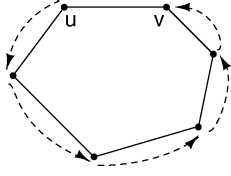


Fig. 15. Valid  $\pi$  assignments after the  $x'_u$  update.

Fig. 15 gives an illustration, where the dashed edges represent the valid  $\pi$  pointers after  $\pi(v)$  is reset, and the solid edges represent the cycle on which updates were carried out.

We now propagate the update on  $u$  to  $v$ . If it is Case 1 or Case 4, we will have  $r'(v) = r'(u) > r(u) = r(v)$  and  $t'(v) = t(v) = \text{fd}(v)$ . For Case 2 and Case 3, we have  $r'(v) = r'(u) > r(u) = r(v)$ , and  $t'(v) = t(v)$  because of  $t'(u) = t(u)$ . Therefore,  $r'(v) - r(v) = r'(u) - r(u)$  and  $t'(v) = t(v)$ .

The above analysis can be applied to other edges along the cycle. As a consequence, the next update on  $u$  can be computed as:  $r''(u) > r'(u)$ ,  $t''(u) = t'(u)$ . Since the evaluation order will not affect the final result, we can stick to this cycle for updates. Thus,  $r(u)$  will keep increasing and eventually exceed  $N_{\text{ff}}$ . Therefore,  $T$  is infeasible. ■

The second is inspired by the proof of Lemma 3.

*Theorem 7:* let  $X'_k$  represent the retiming vector for the whole system when the  $k_{\text{th}}$  vertex is extracted from  $Q$  in Fig. 14. For any  $X'_k$  and its  $|V|_{\text{th}}$  succeeding representation  $X'_{k+|V|^2}$ , if  $r'_k(u) = r'_{k+|V|^2}(u)$ , for all  $u \in V$ , then  $T$  is infeasible.

*Proof:* To characterize the update sequence executed in Fig. 14, we insert dummies into  $Q$ . The dummies are only used as separators and no operation is actually executed when they are extracted from  $Q$ . We insert the first dummy right before we enter the while-loop. The second dummy is inserted when the first one is extracted from  $Q$ , and so on. Let  $\tilde{X}_i$  and  $\tilde{X}_{i+1}$  denote the retiming vector for the whole system when the  $i_{\text{th}}$  and  $(i+1)_{\text{th}}$  dummies are extracted from  $Q$ , respectively. Then, the updates executed between them in the algorithm actually simulate one pass of  $F$ , that is,  $\tilde{X}_{i+1} = F(\tilde{X}_i)$ . Given that a vertex will not be inserted into  $Q$  when it is already in  $Q$ , the number of vertices between any two consecutive dummies is at most  $|V|$ . Therefore,  $F^{|V|}(X'_k) \leq X'_{k+|V|^2}$ .

Since  $r'_k(u) = r'_{k+|V|^2}(u)$ , for all  $u \in V$ , updates on  $t$  can only happen in Case 3. As a result,  $F$  is reduced to updating  $t$  as Bellman-Ford's algorithm. The only reason why it did not converge within  $|V|$  passes is the discovery of a positive cycle  $c$  in  $G$  through which  $d(c) - w(c)T > 0$ . By Lemma 1,  $T$  is infeasible. ■

The third one checks the  $r$  values at primary outputs.

*Theorem 8:* If  $r(v) > 0$  for some primary output  $v$ , then  $T$  is infeasible.

*Proof:* Suppose  $T$  is feasible, then  $F$  is finitely convergent by Theorem 2. Let  $\hat{X}$  denote the least fixpoint of  $F$ . When it is reached, we have  $\hat{r}(v) = h \geq r(v) > 0$  by Lemma 2. It means that  $h$  number of flip-flops need to be moved into the circuit through PO  $v$ . This can only happen when these flip-flops are moved outside the circuit through PI's. As a result,  $\hat{r}(u) = h$ , for all PI/PO  $u$ . However, we can construct another fixpoint by

decreasing all  $\hat{r}(i)$ ,  $i \in V$  by  $h$ , which contradicts the fact that  $\hat{X}$  is the least fixpoint. Therefore,  $T$  is infeasible. ■

2) *Evaluation Order Setting:* Although the evaluation order will not affect the final result, it is a key factor in determining the actual computational time because different orders lead to different update sequences.

Since every  $x_u$  has a dependency set of vertices, an efficient way to do the update is to update  $x_u$  when possibly most of its dependent variables are stable. Therefore, a good strategy to do the update is to first come up with a dependency graph<sup>1</sup>  $G_d = G \cup \{(v, u)\}, \forall (u, v) \in E_1$ . Then in  $G_d$ , we identify each of the strongly connected components, also called a cluster, and do the updates on each cluster until it converges, in a topological order of the clusters. By doing so, the evaluation order of each cluster is determined. But the vertices inside a cluster have no order and the convergent process can be any chaotic iteration.

### C. Computational Complexity

For the sake of simplicity, we put aside the evaluation order setting. The complexity can be broke down into two categories: processing the updates and criteria checking. According to Theorem 7,  $r$  will increase before the next  $|V|_{\text{th}}$  vertex is extracted from  $Q$ , otherwise the program will terminate immediately. On the other hand,  $r(v)$  is upper bounded by  $N_{\text{ff}}$  for all  $v \in V$  by Theorem 5. Therefore, in the worst case,  $O(|V|^3 N_{\text{ff}})$  number of vertices will be extracted from  $Q$ . Let  $n(v)$  denote the number of incoming and outgoing edges incident to  $v \in V$ , and  $N$  denote the maximum of them, i.e.,

$$N = \max_{v \in V} n(v).$$

Then processing the updates triggered by extracted vertices takes  $O(N|V|^3 N_{\text{ff}})$  time. Checking the criterion in Theorem 6 takes  $O(N|V|^4 N_{\text{ff}})$  in all because every vertex extracted from  $Q$  could induce  $N$  number of  $\pi$  cycle checkings, each of which takes at most  $O(|V|)$  time. It dominates the complexity of checking Theorem 5, 7 and 8. Summing up these two categories, the program will terminate in  $O(N|V|^4 N_{\text{ff}})$  time with either the fixpoint or an infeasibility answer.

*Remark 1:* Caution should be taken on this bound. First of all, the design methodology behind the algorithm is to make it as efficient as possible. In other words, we try to improve the practical efficiency of the algorithm as much as we can. This explains why we incorporate the criterion Theorem 6 into the infeasibility checking process, though it has the dominant worst case bound  $O(N|V|^4 N_{\text{ff}})$ . If we drop it, the worst case bound of the algorithm can be reduced to  $O(N|V|^3 N_{\text{ff}})$ . On the other hand, we believe that the worst case bound we obtained is just an upper bound of the actual running time. Although we analyze the worst case bounds for both processing updates and criteria checking, the overall complexity that combines them is hard to analyze since they interact with each other. For example, if the circuit involves a critical cycle and the given period is slightly smaller than the optimal clock period, then the algorithm will reach the worst case bound assuming that Theorem 6 is not considered. However, with Theorem 6, a  $\pi$ -cycle will be found in  $O(V^2)$  time, which will terminate the algorithm. So

<sup>1</sup>The reason why we include  $\{(v, u)\}, \forall (u, v) \in E_1$  is because an update can follow the inverse direction of a forbidden edge.

TABLE I  
RUNNING TIME COMPARISON (SECONDS).

Circuit	set 1		set 2	
	$t_{pre}$	$t_{new}$	$t_{pre}$	$t_{new}$
s386	1.97	0.01	3.67	0.01
s400	1.64	0.01	3.38	0.01
s444	2.23	0.03	4.31	0.01
s838	8.79	0.03	33.42	0.02
s953	9.76	0.04	17.56	0.07
s1488	9.76	0.04	17.56	0.07
s1494	34.13	0.08	62.86	0.09
s5378	684.6	0.24	1344.74	0.29
s13207	-	1.07	-	206.52
s35932	-	18.63	-	6.19
s38584	-	7.44	-	21992.67

far, we cannot find a concrete example that reaches the worst case bound when all the proposed checking criteria are considered. A tighter bound may exist but its mathematical analysis is so complex that we cannot deduce it so far. The efficiency of our algorithm is confirmed by the experiments.

## VII. EXPERIMENTAL RESULTS

We implemented the algorithm with the proposed speed-up techniques in Theorem 5–8 in a PC with a 2.4 GHz/512 K Xeon CPU and 1 GB RAM. To give a comparison with the results of our previous work [13], 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 and 0.2 to 5.0 to wires).

The test files are classified into two sets. In the first set, gates are treated as blocks. In the second set, hMETIS [21] is applied to partition a circuit into groups. Each group is treated as a block. The computed optimal clock periods for both sets match the results in [13]. In Table I, we highlight the running time differences between the algorithm in [13] and the current one, denoted as  $t_{pre}$  and  $t_{new}$  respectively. They both enjoy a precision of 0.1. The results clearly show that the fixpoint computation approach is much more efficient. It confirms that the bound on computational complexity in Section VI-C is loose.

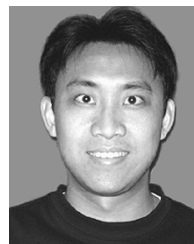
## VIII. CONCLUSIONS

In this paper, we formulate the constraints of the wire retiming problem as a fixpoint computation. An iterative algorithm is proposed with polynomial worst case runtime for feasibility checking. Although the asymptotic complexity is not improved compared with our previous work [13], the practical efficiency of the new algorithm is revealed and corroborated by the experimental results.

## REFERENCES

- [1] P. Cocchini, "Concurrent flip-flop and repeater insertion for high performance integrated circuits," in *Proc. Intl. Conf. on Computer-Aided Design*, 2002, pp. 268–273.
- [2] N. Akkiraju and M. Mohan, "Spec based flip-flop and buffer insertion," in *Proc. Int. Conf. on Computer Design*, 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 in VLSI*, 1983, pp. 86–116.
- [5] N. Shenoy and R. Rudell, "Efficient implementation of retiming," in *Proc. Int. Conf. on Computer-Aided Design*, 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.*, 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.
- [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.*, 2005.
- [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. on Computer-Aided Design*, 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. on Computer-Aided Design*, 2003, pp. 221–226.
- [15] D. K. Y. Tong and E. F. Y. Young, "Performance-driven register insertion in placement," in *Int. Symp. on Physical Design*, 2004, pp. 53–60.
- [16] H. Zhou, D. F. Wong, I.-M. Liu, and A. Aziz, "Simultaneous routing and buffer insertion with restrictions on buffer locations," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 7, pp. 819–824, 2000.
- [17] R. H. J. M. Otten and R. Brayton, "Planning for performance," in *Proc. Design Automation Conf.*, 1998, pp. 122–127.
- [18] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*. Cambridge, U.K.: Cambridge Univ. Press, 1990.
- [19] T. H. Cormen, C. E. Leiserson, and R. H. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1989.
- [20] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *ACM Symp. on Principles of Programming Languages*, Los Angeles, CA, Jan. 1977, pp. 238–252.
- [21] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. Design Automation Conf.*, 1997, pp. 526–529.



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

His research interests are on VLSI computer-aided design, especially deep sub-micron physical design.



**Hai Zhou** (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.

He is an Assistant Professor of Electrical and Computer Engineering at Northwestern University, Evanston, IL. Before he joined the faculty of Northwestern University, he was with the Advanced Technology Group, Synopsys, Inc. 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 is a recipient of the CAREER Award from the National Science Foundation in 2003.