Wire Retiming for System-On-Chip by Fixpoint Computation[*]

Chuan Lin and Hai Zhou
Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208

Abstract

In the current and future System-On-Chips, a non-negligible part of operation time is spent on multiple-clock period wires. 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.

1 Introduction

With a great market drive for high performance and integration, operating frequencies and chip sizes of System-On-Chips (SOCs) are dramatically increasing. 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 a 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 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] and IBM [5] 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 consistency will destroy the correctness of a circuit.

Retiming [8] is a traditional sequential optimization technique that moves flip-flops within a circuit without destroying its functionality. In traditional settings, retiming was used mainly on gate level netlists. Although some recent research incorporated wire delays in retiming [7, 11], they did not consider the situation where multiple flip-flops may be on a global interconnect. This paper explores the alternative utility of retiming–that is, besides its computational function, a flip-flop can be used to fulfill communication buffering requirements.

Since dominant wire delays can only happen on global wires, we solve the problem at the chip level, that is, the design we deal with is a netlist of macro-blocks. The wires within a block are relatively much shorter thus do not need multiple clock periods for propagation. In SOC design, many of these macro-blocks are IP (Intellectual Property) cores. Some of these 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 everywhere [12].

Our previous work [9] used timing macro-models to model the timing behavior of the blocks and a set of integer difference inequalities was shown to be both necessary and sufficient, thus quantify a solution. A polynomial-time algorithm was given for feasibility checking under a given fixed clock period and based on that, a fully polynomial-time approximation scheme for clock period minimization was proposed. For any given relative error $\epsilon > 0$, the running time of the algorithm was proportional to $1/\epsilon$. However, its computational complexity was high, making it inhibitive for large circuits. To overcome this, we present another approach in this paper to solve the wire retiming problem. The main contribution of this paper is an algorithm that uses iterative method to solve the wire retiming with great efficiency. Theoretical validity of the algorithm is based on the equivalence between the conditions of a retiming solution and a fixpoint computation.

2 Problem formulation

In fact, we are solving the same problem as [9], which is the retiming on a SOC design with a given block placement(also known as floorplan) and a global routing of the global wires. 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. We will also use the same timing models to specify the timing behavior of the circuit. To avoid repetition, the detailed description of the timing models is omitted here and we only give a circuit example and its corresponding timing model in Figure 1. Interested readers may refer to [9].

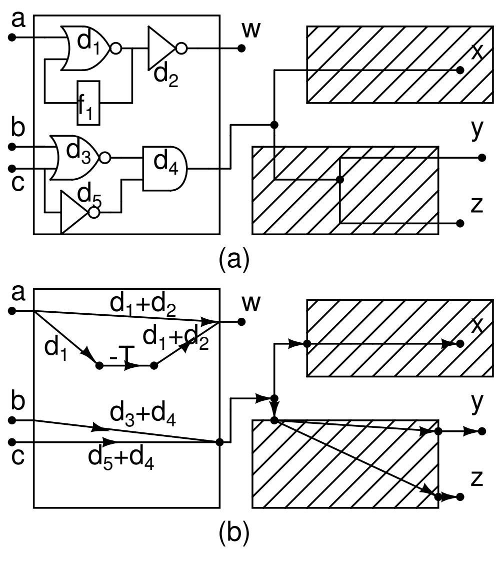

Figure 1: A circuit example and its timing model

With the macro-models, a SOC design becomes a directed graph, where a vertex represents a source or sink of a net, the input or output of the virtual flip-flop introduced by a sequential block, a point where a wire gets into or out of a buffer forbidden area, or a Steiner point outside buffer forbidden areas. A directed edge is used to represent a fan-out relation within a block or a wire connection outside buffer forbidden areas. On an edge representing a fan-out relation, the delay is either a positive constant or $-T$, and no flip-flop can be inserted on it. On the other hand, flip-flops can be inserted on an edge representing a wire outside buffer forbidden areas. Since we assume that buffers can be inserted, the delay of such a wire is positive and proportional to its length. Therefore, the problem we want to solve can be formulated as follows.

[*]This work was supported by the NSF under CCR-0238484.

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 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) is minimized.*

Problem 1 wants to find a retiming solution to minimize the clock period. A reasonable step to solve this problem is to consider whether we can find a retiming to satisfy a given clock period. Such a problem is defined as follows.

Problem 2 (Fixed Period Wire Retiming) *Given a clock period T and 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 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 maximum delay between any two consecutive flip-flops is at most T.*

If we can solve this problem, Problem 1 can be solved by using a binary search.

3 Wire retiming as fixpoint computation

3.1 Notations and constraints

Before discussing the solutions to the two problems, we will first select some notations to clearly state the requirements for a solution.

From the formulations of the problems, we already have a delay $d(u, v)$ and a weight $w(u, v)$, for each edge $(u, v) \in E$. We will follow the convention of Leiserson and Saxe [8] to use an integer variable $r(u)$ to represent the number of flip-flops moved from the outgoing edges of a vertex u to its incoming edges. Because of the existence of wire delays, the positions of flip-flops must also be included in a retiming solution. Hence, for each $u \in V$, we associate it with another variable $t(u)$ to represent its arrival time with respect to the farthest preceding flip-flop on its incoming paths,.

Using these notations, the necessary and sufficient conditions of a retiming solution under a given fixed clock period T can be stated as follows.

$$\begin{gathered} r(u) = r(v), \quad \forall (u,v) \in E_1, \\ w(u,v) + r(v) - r(u) \geq 0, \quad \forall (u,v) \in E_2, \\ t(v) \geq t(u) + d(u,v), \quad \forall (u,v) \in E_1, \\ t(v) \geq t(u) + d(u,v) - (w(u,v) + r(v) - r(u))T, \forall (u,v) \in E_2, \\ 0 \leq t(u) \leq T, \quad \forall u \in V. \end{gathered}$$

In fact, we can define $\mathrm{fd}(u)$ and $\mathrm{bd}(u)$ as

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

where the path delay $d(p)$ is defined as the sum of edge delays on the path p. If there are no forbidden edges with positive delay coming into u, $\mathrm{fd}(u) = 0$. Similarly, $\mathrm{bd}(u) = 0$ if there are no forbidden edges with positive delay going from u. Then the above requirements can be equivalently stated as the following set of conditions.

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

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

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

Formula (3) actually implies the legality of a retiming, since for each edge $(u, v) \in E_2$,

$$\begin{aligned} & (3) \\ \Rightarrow\ & w(u,v) + r(v) - r(u) \geq (t(u) + d(u,v) - t(v))/T \\ \Rightarrow\ & w(u,v) + r(v) - r(u) > (\mathrm{fd}(u) + 0 - T)/T > -1 \\ \Rightarrow\ & w(u,v) + r(v) - r(u) \geq 0. \end{aligned}$$

Furthermore, we can establish the following theorem.

Theorem 1 *(1)-(3) have a solution if and only if the following equation has a solution for any $v \in V$,*

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

where

$$\begin{aligned} r_1(v) = \max\Bigg(& \max_{\forall (u,v),(v,u) \in E_1, r(v) < r(u)} r(u), \\ & \max_{\forall (u,v) \in E_2} \left\lceil \frac{t(u) + d(u,v)}{T} - w(u,v) + r(u) \right\rceil - 1 \Bigg), \end{aligned} \tag{5}$$

$$\begin{aligned} t_1(v) = \max\Big(& fd(v), \\ & \max_{\forall (u,v) \in E} t(u) + d(u,v) - (w(u,v) + r_1(v) - r(u))T \Big). \end{aligned} \tag{6}$$

Proof: If (4) has a solution, (1) is directly implied by (4) and (6). And (2) can be implied by (5) otherwise there must exist an edge $(u, v) \in E_1$ such that $r(u) \neq r(v)$. If $r(u) > r(v)$, we have $r(v) \geq r_1(v) \geq r(u) > r(v)$. If $r(u) < r(v)$, we have $r(u) > r_1(u) \geq r(v) > r(u)$. Either of them leads to a contradiction.

For any $v \in V$, $(u, v) \in E_1$, due to (5), we have $r_1(v) \geq r(u)$. Then due to (1),

$$t(u) + d(u,v) - (0 + r_1(v) - r(u))T \leq t(u) + d(u,v) \leq T,$$

for any $(u, v) \in E_2$, due to (5),

$$t(u) + d(u,v) - (w(u,v) + r_1(v) - r(u))T \leq T.$$

Thus, $t_1(v) \leq T$. Based on this, (3) can be implied by (4) otherwise there must exist an edge $(u, v) \in E$ such that

$$t(v) < t(u) + d(u,v) - (w(u,v) + r(v) - r(u))T.$$

Due to (4), it implies that

$$\begin{aligned} & t_1(v) < t(u) + d(u,v) - (w(u,v) + r_1(v) - r(u))T \\ \text{or} \quad & \mathrm{fd}(v) < t(u) + d(u,v) - (w(u,v) + r_1(v) + 1 - r(u))T \end{aligned}$$

Since $t_1(v) \leq T$, it is always true that

$$t_1(v) < t(u) + d(u,v) - (w(u,v) + r_1(v) - r(u))T,$$

which contradicts (6). Therefore, the solution to (4) is also a solution to (1)-(3).

To establish the converse, we assume that (1)-(3) are solvable. Since (1) and (3) are inequalities, there could exist many different solutions. The ones we are interested in are the ones that satisfy the timing validity, that is, for every primary input u, $t(u) = r(u) = 0$ and for every vertex $v \in V$ other than the primary inputs, it satisfies

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

In addition, for vertex v that has no forbidden edges to or from it, it may have two equivalent satisfying assignments: $(T, r(v))$ or $(0, r(v)+1)$. To avoid the ambiguity, we select such a solution that always chooses the former one. Thus for such vertex, $0 < t(v) \le T$. We will show in the following that the solution we quantified above is also a solution to (4).

For each primary input v, $t_1(v) = 0 = t(v), r_1(v) = 0 = r(v)$. For other vertices, due to (3),

$$\begin{aligned} r(v) &\ge \frac{t(u)+d(u,v)-t(v)}{T} - w(u,v) + r(u) \\ &\ge \left\lceil \frac{t(u)+d(u,v)}{T} - w(u,v) + r(u) \right\rceil - 1. \end{aligned}$$

For simplicity, we use $r_2(v)$ and $r_3(v)$ to denote the first and the second term in (5) respectively, that is, $r_1(v) = \max(r_2(v), r_3(v))$. If there is at least one forbidden edge to or from v, due to (2), $r_2(v) = r(v)$. Due to (4) and (5), $r(v) \le r_1(v) \le r(v)$, thus $r_1(v) = r(v)$. Otherwise, $r_1(v) = r_3(v)$ by (5). If $r_3(v) < r(v)$, due to (5), for any $(u,v) \in E_2$,

$$\begin{aligned} r(v) &\ge \left\lceil \frac{t(u)+d(u,v)}{T} - w(u,v) + r(u) \right\rceil \\ \Rightarrow\ & t(u) + d(u,v) - (w(u,v) + r(v) - r(u))T \le 0. \end{aligned}$$

Due to the timing validity, $t(v) \le 0$, which contradicts that $0 < t(v) \le T$ for such v. Hence, $r_1(v) \ge r(v)$. But due to (4), $r_1(v) \le r(v)$, therefore $r_1(v) = r(v), \forall v \in V$. According to (3), (6), $r_1(v) = r(v)$ and timing validity, we know that $\mathrm{fd}(v) \le t_1(v) = t(v) \le T - \mathrm{bd}(v), \forall v \in V$. Therefore, the solution also satisfies (4). ∎

3.2 Lower and upper bounds of clock period

According to [9], we have the following lemmas, where $w(p)$ is the total number of flip-flops on the path p, $d(c)$ is the cycle delay if a path actually forms a cycle c and $w(c)$ is defined similarly.

Lemma 1 *A feasible clock period T must satisfy*

$$\begin{aligned} T &\ge T_1 \overset{\text{def}}{=} \max_{u \in V}(fd(u) + bd(u)), \\ T &\ge T_2 \overset{\text{def}}{=} \max_{c \in \text{cycle}} \frac{d(c)}{w(c)}, \\ T &\ge \max_{p \in \text{PI} \rightsquigarrow \text{PO}} \frac{d(p)}{w(p)+1}, \end{aligned}$$

If a virtual vertex M is introduced as well as directed forbidden edges from each PO to it with d=0, w=0 and from it to each PI with d=0, w=1, then the meaning of cycle is extended to include every PI $\rightsquigarrow$ PO path. Thus, we extend T_2 to both cycles and PI $\rightsquigarrow$ PO paths.

It has been shown that $T_1 + T_2$ is an upper bound of the optimal clock period for a circuit in the gate level. In fact, we can generalize the result to the following lemma [9].

Lemma 2 *If each connected component in the subgraph $G_1 = (V, E_1)$ is a complete bipartite graph, the optimal clock period can be upper bounded by $T_1 + T_2$.*

If the condition in the above lemma is not satisfied, the approach to find a tighter bound is as follows. First, we find an optimal retiming solution without considering forbidden edges. This can be done based on the computation of T_2. 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. From any set of forbidden edges that forms a complete bipartite graph, any flip-flop can be moved out with at most an increase of T_1 to the period. To move a flip-flop out of a forbidden edge in a non-complete bipartite graph, other flip-flops may be moved over the block, thus the increase of the period could be larger. However, the local adjustment will keep the number of flip-flops moved out of a non-forbidden edge as small as possible.

3.3 Fixpoint formulation

According to Theorem 1, (1)-(3) are transformed into (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:

$$x_v = f_v(x_{v_1}, \ldots, x_{v_k}), \tag{7}$$

where $v_1, v_2, \ldots, v_k$ are vertices in V that have edges to or from v. We use X to represent the vector $(x_1, x_2, \ldots, x_n)$, where $n = |V|$, that is, the retiming information for the whole circuit. Put all f_v, $v \in V$ together, we get a transformation for the whole system which can be written as

$$X = F(X) \tag{8}$$

A partial order ($\le$) can be defined between two retiming values x_u and x'_u as follows.

$$x_u \le x'_u \overset{\text{def}}{=} (r(u) < r'(u)) \vee (r(u) = r'(u) \wedge t(u) \le t'(u))$$

In fact, a lexicographic order is defined.

The partial order on each vertex can be extended vertex-wisely to get a partial order on the vectors: two vectors $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ satisfy $X \le Y$ if and only if $x_i \le y_i$ for all $1 \le i \le n$. There is a natural initialization to set $t(u) = r(u) = 0$ for each primary input u and $t(v) = r(v) = -\infty$ for each vertex $v \in V$ other than the primary inputs. 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 [4], the solution space P becomes a complete partially ordered set(CPO). For any $X \in P$, we have $\perp \le X \le \top$.

To find a solution to (8), iterative method can be used. It starts with $X_0 = \perp$ as the initial vector, then iteratively computes new vectors from previous ones $X_1 = F(X_0), X_2 = F(X_1), \ldots$ until we find a X_m such that $X_m = X_{m-1}$. Then X_m is a solution to (8), also called a fixpoint of F.

In order to use iterative method, we need to know if F is finitely convergent, that is, if there exists a finite m such that $F^{m-1}(X_0) = F^m(X_0)$.

We first show that F is a *monotonic* (or *order-preserving*) transformation, that is, it satisfies the following lemma.

Lemma 3 *For any vector X and X', if $X \le X'$ then $F(X) \le F(X')$.*

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

Since $X \le X'$, for any vertex $u \in V$, $x_u \le x'_u$, that is, either $r(u) < r'(u)$ or $r(u) = r'(u) \wedge t(u) \le t'(u)$. Therefore, according to (5), $r_1(v) \le r'_1(v)$, for any vertex $v \in V$. If $r_1(v) = r'_1(v)$, due to (6) and $x_u \le x'_u$, we have $t_1(v) \le t'_1(v)$. No matter what the value $T - \mathrm{bd}(v)$ is, we have $f_v(x_v) \le f_v(x'_v)$ by (4). If $r_1(v) < r'_1(v)$, consider the following two cases. If $t_1(v) \le T - \mathrm{bd}(v)$, then $f_v(x_v) \le f_v(x'_v)$ by (4). Otherwise $r_1(v) + 1 \le r'_1(v)$, $t(v) = \mathrm{fd}(v)$, which also leads to $f_v(x_v) \le f_v(x'_v)$ by (4). Thus, f_v is monotonic for any $v \in V$, based on which, the monotonicity of F is established. ∎

Based on the above lemma, we have the following theorem [4].

Theorem 2 *Let P be a CPO and $F : P \to P$ an order-preserving map. Then F has a least fixpoint.*

Based on this, we have the following theorem.

Theorem 3 *The fixpoint found by iterative method 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, we know $X_{lf} \leq X_m$. Since $X_0 \leq X_{lf}$, according to Lemma 3, $X_m = F^m(X_0) \leq F^m(X_{lf}) = X_{lf}$. Therefore, $X_m = X_{lf}$. $\square$

However, if the fixpoint that we could reach is the top element($\top$), it means that there is no finite solution to (8). Actually, such F is not finitely convergent because it requires infinite number of iterations to reach the top element. In order to qualify a finitely convergent F, we establish the following lemma.

Lemma 4 *If F is not finitely convergent but T is feasible, then for any finitely reachable X and its $|V|_{th}$ iteration result $X' = F^{|V|}(X)$, there must exist a vertex $v \in V$ such that $r'(v) > r(v)$.*

Proof: If for any $v \in V$, $r'(v) = r(v)$, then $r(v)$ is kept the same in each iteration. Thus (2) is always true. If vertex v has forbidden edges to or from it, due to (2), (4) and (5), $r(v) \leq r_1(v) \leq r(v)$, thus $r_1(v) = r(v)$. Otherwise, $\mathrm{fd}(v) = \mathrm{bd}(v) = 0$, according to (5) and (6), $t_1(v) \leq T = T - \mathrm{fd}(v)$ resulting in $r_1(v) = r(v)$. Therefore, $(r(v), t(v)) = (r_1(v), t_1(v))$ is always true for any $v \in V$ and $t_1(v) \leq T - \mathrm{bd}(v)$ is always satisfied.

Therefore, F is reduced to only updating $t(v)$ by (6), which is exactly what Bellman-Ford's algorithm [2] 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 through which we have $d(c) - w(c)T > 0$. Otherwise, F converges and becomes finitely convergent since X is finitely reachable. Thus, $T < d(c)/w(c)$. According to Lemma 1, T is infeasible, which is a contradiction. Therefore, there must exist a vertex $v \in V$ such that $r'(v) \neq r(v)$. In fact, $r'(v) > r(v)$ due to $X_0 \leq X_1$ and the monotonicity of F. $\square$

Based on the above lemma, we have the following theorem.

Theorem 4 *F is finitely convergent if and only if T is feasible.*

Proof: If F is finitely convergent, we must have found a vector X' such that $X' = F(X')$. Since X' satisfies (4), according to Theorem 1, it also satisfies (1)-(3), thus T is feasible.

For the other direction, assume T is feasible, but F is not finitely convergent. Since T is feasible, there exists a solution $X^s = ((r^s(1), t^s(1)), \ldots, (r^s(n), t^s(n)))$ such that $F(X^s) = X^s$. Starting from X_0, at most $|E|$ iterations are needed for each vertex $v \in V$ to have a finite $r(v)$ and $t(v)$. Since F is not finitely convergent, according to Lemma 4, we can always find such a finite m and a vertex $v \in V$ that $X^m = ((r^m(1), t^m(1)), \ldots, (r^m(n), t^m(n))) = F^m(X_0)$ and $r^m(v) > r^s(v)$. However, Because $X_0 \leq X^s$, according to the monotonicity of F, $F^m(X_0) \leq F^m(X^s) = X^s$. That is, for any vertex $v \in V$, $r^m(v) \leq r^s(v)$, which is a contradiction. Therefore, F is finitely convergent. $\square$

Based on Theorem 4, the iterative method is guaranteed to find a solution to (8) in a finite number of iterations if the given clock period is feasible. For the infeasible cases of T, we can simply apply some criteria to stop the iterations when the number of iterations exceeds a certain value, which will be elaborated in the next section.

In the following we will establish that no matter what evaluation order is used, the update process will always converge to the same fixpoint, that is, the least fixpoint. This is called the scheme of *chaotic iteration* [3]. Here, a transformation F is composed of a set of partial transformations $f_1, f_2, \ldots, f_n$. 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 the same. 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.

Lemma 5 *Given S a subset of vertices, if $X \leq F(X)$, then $X \leq F_S(X) \leq F(X)$.*

Lemma 6 *Let $X'_0 = X_0, X'_1, \ldots, X'_{m'}, X'_{m'+1} = X'_{m'}$ be the update sequence generated by a sequence of arbitrary partial transformations $F_{S_0}, F_{S_1}, \ldots, F_{S_{m'}}$. Then $X'_i \leq F_{S_i}(X'_i) \leq F(X'_i)$, for any $0 \leq i \leq m'$.*

Proof: We prove it by applying induction on i.

For $i = 0$, $X'_0 = X_0 \leq F(X_0)$. According to Lemma 5, $X'_0 \leq F_{S_0}(X'_0) = X'_1 \leq F(X'_0)$. Assume it is true for $0 \leq i \leq k$, then $X'_k \leq F_{S_k}(X'_k) = X'_{k+1} \leq F(X'_k)$. According to Lemma 3 and the inductive hypothesis, $F(X'_k) \leq F(X'_{k+1})$, thus $X'_{k+1} \leq F(X'_k) \leq F(X'_{k+1})$. By Lemma 5, we have $X'_{k+1} \leq F_{S_{k+1}}(X'_{k+1}) \leq F(X'_{k+1})$. $\square$

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

Theorem 5 *Any fair evaluation order will reach the same fixpoint as F, the least fixpoint.*

Proof: Let $X'_0 = X_0, X'_1, \ldots, X'_{m'}$ be the update sequence generated by a sequence of arbitrary partial transformations $F_{S_0}, F_{S_1}, \ldots, F_{S_{m'-1}}$ and $X_0, X_1, \ldots, X_m$ be the one generated by F. If $X'_i \leq X_i$, due to Lemma 3, we have $F(X'_i) \leq F(X_i)$ and due to Lemma 6, we have $F_{S_i}(X'_i) \leq F(X'_i)$. Thus, $F_{S_i}(X'_i) \leq F(X_i)$. Since $X'_0 = X_0$, we then have $X'_i \leq X_i$ for $0 \leq i \leq \min(m', m)$. Our goal is to prove $X'_{m'} = X_m$.

Since X_m and $X'_{m'}$ are fixpoints, they satisfy $X_m = F(X_m)$ and $X'_{m'} = F(X'_{m'})$. According to Theorem 3, X_m is the least fixpoint, $X_m \leq X'_{m'}$. If $m > m'$, then $X'_{m'} \leq X_{m'} \leq X_m$. Because $X_{m'}$ is not a fixpoint, $X'_{m'}$ becomes the least fixpoint, which is a contradiction. Therefore, $m \leq m'$. We can extend the sequence generated by F to the length of m' by assigning $X_i = X_m$, for all $m+1 \leq i \leq m'$. Then $X_m \leq X'_{m'} \leq X_{m'} = X_m$, that is, $X'_{m'} = X_m$. $\square$

4 Algorithm

4.1 Detailed description

(4) gives us a clear scheme to do the updates. But since the evaluation order will not affect the result, we can propagate the update on a vertex to its adjacent vertices instead of determining the update from them as (4) suggests. Therefore, based on (4), we have the following operations for all four different cases to propagate the update of x_u out.

1. $(u, v) \in E_1, d(u, v) = -T$
   f_v : If $r(v) < r(u)$, then $r(v) = r(u), t(v) = \mathrm{fd}(v) = 0$.

2. $(u, v) \in E_1, d(u, v) > 0$
   f_v : If $r(v) < r(u)$, then $r(v) = r(u)$.
   If (3) is violated, then $t(v) = \max(t(u) + d(u, v), \mathrm{fd}(v))$.

3. $(u, v) \in E_2$
   f_v : If $t(v), r(v)$ violate (3), then select $r(v)$ for $t(v) = t(u) + d(u, v) - (w(u, v) + r(v) - r(u))T$ to be within $(0, T]$. If $t(v) < \mathrm{fd}(v)$, then set $t(v) = \mathrm{fd}(v)$. If $t(v) > T - \mathrm{bd}(v)$, then increase $r(v)$ by 1 and set $t(v) = \mathrm{fd}(v)$.

4. $(v, u) \in E_1$
   f_v : If $r(v) < r(u)$, then $r(v) = r(u), t(v) = \mathrm{fd}(v)$.

With these explicit operations, any fair evaluation order will finally lead to the least fixpoint if F is finitely convergent. However, 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 use them to terminate the procedure. The following theorem provides such a criterion.

Theorem 6 *A given T is infeasible if and only if the iteration gives $r(v) > maxFF$ for some vertex $v \in V$, where $maxFF$ is the total number of flip-flops in the circuit.*

Proof: Since we introduced a virtual vertex M and additional forbidden edges, all PI$\rightsquigarrow$PO paths become cycles. If T is infeasible, there must either exist a cycle c through which the retiming values of the constituent vertices never become stable, i.e., a fixpoint is never reached. Thus, for any $i \in c$, $r(i)$ will monotonically increase and eventually exceed the $maxFF$ bound.

To prove the other direction, we assume that T is feasible. Since $r(v)$ increases monotonically, when a fixpoint is reached, it must also be true that $r(v) > maxFF$. Let V' be the set of vertices that can reach or be reached from v in the circuit, including v. Let u be such a vertex whose $r(u)$ is the minimum among all the vertices in V', i.e., $r(u) = \min_{i \in V'} r(i)$. If $r(u) > 0$, we can construct another fixpoint by decreasing all $r(i)$, $i \in V'$ by $r(u)$, which contradicts Theorem 5 that the fixpoint we finally reach is the least fixpoint. Therefore, $r(u) \leq 0$.

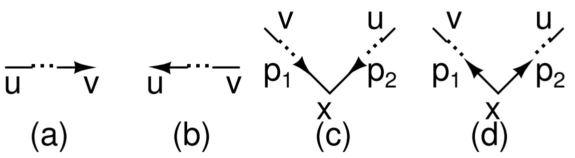

Figure 2: Four possible relations between u and v.

All possible relations between u and v are shown in Figure 2. For case (a) and (b), more than $maxFF$ number of flip-flops must be moved into or out of the path between u and v, which is impossible. For case (c), we have $w'(p_1) = w(p_1) - r(v) + r(x)$. Because there is no direct path between u and v, all paths between v and x must be from v to x. Therefore, the flip-flops that were moved into p_1 are different from those that were originally in p_1. Thus, $w(p_1) + r(x) \leq maxFF$, which leads to $w'(p_1) < 0$, which is impossible for a valid fixpoint. For case (d), $w'(p_1) = w(p_1) + r(v) - r(x)$. By the same argument, the flip-flops that were moved into p_1 through x must be different from those $w(p_1)$ number of flip-flops. Thus, $w(p_1) + r(v) \leq maxFF$, which leads to $r(v) \leq maxFF - w(p_1) \leq maxFF$, which contradicts $r(v) > maxFF$. Therefore, our assumption is incorrect, i.e., the given T is infeasible. ∎

In Figure 3, we present our algorithm for feasibility checking under a given fixed clock period. It uses an auxiliary queue Q to facilitate the update procedure. Initially, we have in Q all primary inputs. During the procedure, a vertex is inserted into Q whenever its retiming information is updated. The algorithm terminates when Q becomes empty or it discovers the infeasibility.

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

Initialization, add $PI$ into $Q$ and $flag \leftarrow 1$;
Introduce $M$, $E' = E \cup \{(PO, M)\} \cup \{(M, PI)\}$;
While $Q \neq \emptyset$ && $flag$ do
   $u \leftarrow$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 $r(v) > maxFF$ then
         $flag \leftarrow 0$;
If $flag$ then
   Report the fixpoint;
Else
   Report infeasibility;
```

Figure 3: Pseudocode of algorithm for feasibility checking

4.2 Speed-up techniques

4.2.1 Infeasibility checking criteria

Although Theorem 6 provides a criterion for infeasibility checking, it may be inefficient. Consider a special case when G involves a critical cycle, that is, a cycle that determines the optimal clock period. When $maxFF$ is large and the given T is slightly smaller than the optimal clock period, the algorithm will spend a long time on updating the critical cycle before the infeasibility is discovered.

To develop a more efficient criterion, we need more information, that is, the history of the updates. The idea is similar to Shenoy and Rudell [10] to maintain a predecessor vertex pointer denoted by $\pi()$ for each vertex. Whenever x_v is updated through edge (u, v) or (v, u), we set $\pi(v) = u$. It can be seen that (2) and (3) are satisfied after the update operations for any of the four possible cases. In fact, $\pi()$ can serve as an indicator if we do the following: whenever x_u is updated, for any $v \in V$ whose $\pi(v)$ equals u, we set $\pi(v) =$ null. Thus, (2) and (3) are always satisfied on edge $(\pi(v), v)$, for any $v \in V$. And we have the following theorem.

Theorem 7 *If there exists a vertex u whose consecutive updates x_u and x'_u satisfy $t'(u) = t(u)$, and before resetting $\pi(v) =$ null for those v with $\pi(v)$ equal to u, we find a cycle by following $\pi()$ from u, then T is infeasible.*

Proof: According to the Lemma 6, $x_u \leq x'_u$. Given $t'(u) = t(u)$, we know that $r'(u) > r(u)$, otherwise $x'_u = x_u$ is not an update.

If the cycle only involves u, the only edge e in the cycle should not be forbidden otherwise any finite T is infeasible. Thus the update is on edge $(u, u) \in E_2$ as in case 3. If $t'(u) = \mathrm{fd}(u)$, according to the operations done for case 3, either $0 < t(u) + d(e) - (w(e) + r'(u) - r(u))T \leq \mathrm{fd}(u)$ or $T - \mathrm{bd}(u) < t(u) + d(e) - (w(e) + r'(u) - 1 - r(u))T \leq T$. For either one, since $r'(u) > r(u)$ and $T \geq \mathrm{fd}(u) + \mathrm{bd}(u)$, we have $d(e) - w(e)T > 0$. According to Lemma 1, T is infeasible. If $t'(u) \neq \mathrm{fd}(u)$, then $t'(u) = t(u) + d(e) - (w(e) + r'(u) - r(u))T$. Also, $d(e) - w(e)T > 0$, T is infeasible.

For the cycle that involves multiple vertices, there must exist a vertex v other than u such that $\pi(v) = u$. Since (2) and (3) are satisfied after the operations done on edge (u, v), we have the following argument. If (u, v) is case 1 or case 4, we have $r'(v) = r'(u) > r(v)$ and $t'(v) = t(v) = \mathrm{fd}(v)$. If (u, v) is case 2, we also have $r'(v) = r'(u) > r(v)$, and $t'(v) = t(v)$ because $t'(u) = t(u)$. If (u, v) is case 3, consider the following two situations. If $0 < t(u) + d(u, v) - (w(u, v) + r(v) - r(u))T \leq T - \mathrm{bd}(v)$, since $r'(u) > r(u)$, it must be true that $r'(v) > r(v)$ and $r'(v) - r(v) = r'(u) - r(u)$. Due to $t'(u) = t(u)$, it is also true that $t'(v) = t(v)$. If $T \geq t(u) + d(u, v) - (w(u, v) + (r(v) - 1) - r(u))T > T - \mathrm{bd}(v)$, by the same reasoning, $r'(v) - r(v) = r'(u) - r(u)$ and $t'(v) = t(v) = \mathrm{fd}(v)$. The argument above can be applied similarly to every $(\pi(x), x)$, where x is a vertex in the cycle, and have $r'(x) > r(x)$, $t'(x) = t(x)$. Since the evaluation order can be chosen arbitrarily, we can keep doing the updates along the

Table 1: Optimal Clock Period

Circuit	$\|V\|$	$\|E\|$	T_2	w/o non-CB blocks		w/ non-CB blocks		
				No.Iter	T_{opt}	No.Part	No.Iter	T_{opt}
s386	519	700	51.0	5	51.1	50	10	55.0
s400	511	665	32.2	6	32.2	50	10	50.6
s444	557	725	35.0	5	35.2	40	10	63.2
s838	1299	1206	76.0	5	76.0	130	11	84.0
s953	1183	1515	60.6	5	60.6	110	10	69.5
s1488	2054	2780	70.1	5	70.6	200	11	73.3
s1494	2054	2792	76.8	5	76.9	160	11	79.9
s5378	7205	8603	111.0	5	111.2	500	13	115.3
s13207	19816	22999	239.5	12	239.5	1000	15	292.8
s35932	46097	58266	148.3	11	148.3	2000	13	163.2
s38584	53473	66964	203.9	12	204.0	2000	14	264.0

cycle. Thus, $r(u)$ will keep increasing and eventually exceed the $maxFF$ bound. Therefore, T is infeasible. □

In addition, we have the following theorem.

Theorem 8 *If there exists a primary output v such that $r(v) > 0$, then T is infeasible.*

Proof: Since every vertex $x \in V$ other than the primary inputs is initialized with $r(x) = t(x) = -\infty$, its update is essentially induced by the primary inputs. If $r(v) = c > 0$ for some primary output v, due to the virtual vertex M and the additional forbidden edges we introduced, each primary input u will be updated with $r(u) = c$, $t(u) = 0$. Since there is no restriction on the evaluation order, we can repeat the update history and have $r(v) = 2c$ after the second round. If we keep repeating, $r(v)$ will finally exceed the $maxFF$ bound. Therefore, T is infeasible. □

The above theorem shows that the introduction of the virtual vertex M is unnecessary for infeasibility checking. Thus, the algorithm can run directly on E.

4.2.2 Evaluation order setting

Although the evaluation order will not affect the final result, it is the key factor in determining the computational complexity because different order will lead to different update history.

Since every x_u has a dependency set of vertices, a natural 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 $G_d = G \cup \{(v, u)\}, \forall(u, v) \in E_1$. 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. 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.

5 Experimental results

We implemented the algorithm with the proposed speed-up techniques in a PC with a 2.4GHz/512K Xeon CPU and 1GB RAM. In order to give a comparison with the results in [9], we use the same test files, which are modified from ISCAS-89 suite with random delay assignment. To reflect the impact of global interconnects in a SOC design, the delay range is intentionally chosen in order for the wire delay to be commensurate or even many times larger than the block delay. To further test the cases with non-complete bipartite("non-CB" in Table 1) blocks, we apply hMETIS [6] to partition a circuit into groups. All edges inside a group are then treated as forbidden edges. The number of partitions of a circuit, which is denoted as "No.Part" in Table 1, plays a key role in determining the percentage of non-complete bipartite blocks.

In Table 1, we report the computed optimal clock period for each circuit. The lower bound T_2 is also reported as a comparison. In Table 2, we highlight the difference of running time between the algorithm in [9] and the current one for the circuits with non-complete bipartite blocks, denoted as t_{old} and t_{new} respectively. Those for the circuits without non-complete bipartite blocks are similar and omitted due to the space limit. The results clearly show that the fixpoint computation approach is much more efficient.

Table 2: Running Time Comparison

Circuit	t_{old}(sec)	t_{new}(sec)
s386	3.67	0.01
s400	3.38	0.01
s444	4.31	0.01
s838	33.42	0.02
s953	17.56	0.07
s1488	98.88	0.05
s1494	62.86	0.09
s5378	1344.74	0.29
s13207	-	206.52
s35932	-	6.19
s38584	-	21992.67

References

[1] P. Cocchini. Concurrent flip-flop and repeater insertion for high performance integrated circuits. In *ICCAD*, 2002.

[2] T. H. Cormen, C. E. Leiserson, and R. H. Rivest. *Introduction to Algorithms.* MIT Press, 1989.

[3] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, CA, January 1977.

[4] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge, 1990.

[5] S. Hassoun and C. J. Alpert. Optimal path routing in single and multiple clock domain systems. In *ICCAD*, 2002.

[6] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning:application in vlsi domain. In *DAC*, pages 526–529, 1997.

[7] K. N. Lalgudi and M. C. Papaefthymiou. An Efficient Tool for Retiming with Realistic Delay Modeling. In *DAC*, San Francisco, CA, June 1995.

[8] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing Synchronous Circuitry by Retiming. In *Advanced Research in VLSI: Proc. of the Third Caltech Conf.*, pages 86–116. Computer Science Press, 1983.

[9] Chuan Lin and Hai Zhou. Retiming for wire pipelining in system-on-chip. In *ICCAD*, pages 215–220, 2003.

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

[11] T. Soyata, E. G. Friedman, and J. H. Mulligan. Incorporating Interconnect, Register, and Clock Distribution Delays into the Retiming Process. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 16(1):105–120, January 1997.

[12] Hai Zhou, D. F. Wong, I-Min Liu, and Adnan Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. DAC, 1999.