Retiming for Wire Pipelining in System-On-Chip*

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

Abstract

At the integration scale of System-On-Chips (SOCs), the conflicts between communication and computation will become prominent even on a chip. A big fraction of system time will shift from computation to communication. In synchronous systems, a large amount of communication time is spent on multiple-clock period wires. In this paper, we explore retiming to pipeline long interconnect wires in SOC designs. Behaviorally, it means that both computation and communication are rescheduled for parallelism. The retiming is applied to a netlist of macro-blocks, where the internal structures may not be changed and flip-flops may not be able to be inserted on some wire segments. This problem is different from that on a gate level netlist and is formulated as a wire retiming problem. Theoretical treatment and a polynomial time algorithm are presented in the paper. Experimental results showed the benefits and effectiveness of our approach.

1 Introduction

With a great market drive for high performance and integration, operating frequencies and chip sizes of 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 [2] and IBM [11] 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 [14] is a traditional sequential optimization technique that moves flip-flops within a circuit without destroying its functionality. In traditional settings, retiming was used only on gate level netlists and in most cases delays were dominated by gate delays–that is, wire delays were ignored. With increasing communication delays as mentioned above, 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. In our problem formulation, we will use timing macro-models to model the timing behavior of the blocks. 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 [20]. We will incorporate buffering position restrictions in our solution.

The contributions of this paper are in multiple aspects. First, timing macro-models of both combinational and sequential blocks are established to facilitate retiming on them. Second, flip-flop location restrictions within blocks and on the wires through blocks are handled uniformly. Finally, a polynomial time algorithm is designed for the wire retiming problem that has both block and wire delays.

2 Problem formulation

We consider wire pipelining via retiming on a SOC design with a given block placement (also known as floorplan) and a global routing of the global wires. Such a design is depicted in Figure 1. Here, we have five blocks with the floorplan and the global routing of global wires. Each wire has an arrow to indicate the signal direction, and a weight to specify how many flip-flops are on the wire. Those with weight 0 have the weight omitted. For example, the wire from u to n has 1 flip-flop, while the wire from n to v has 0 and that from n to w has 2. 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.

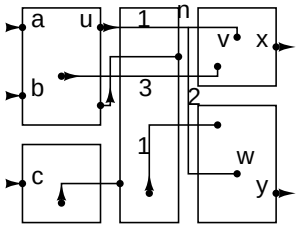

Figure 1: Global interconnections on a SOC design

In order to take the delays within each block into consideration, timing models are used for specifying the timing behavior of each block. Due to the increasing popularity of SOC design and IP-core based design, recently there are increasing research activities on timing models for macro-blocks [7, 16, 10]. If a block is a pure combinational circuit such as an ALU or a multiplier, then a minimum and a maximum delay from each input pin to every output pin can be used to characterize the timing behavior of the block. This is a traditional approach. Recent researches are mainly focused on sequential blocks [7, 16, 10]. Generally speaking, if there are combinational paths from an input pin to an output pin, a minimum and maximum delay pair will be used to characterize the delay on each path. If an input pin has a path to a flip-flop in the block, the arrival time of the input pin must be constrained to satisfy the set-up condition. Finally, if an output pin has paths from flip-flops in the block, then the arrival time of the pin is given by the path delays.

Traditional retiming is applied to logic level netlists that are composed of simple gates. In our application, the netlist is composed of macro-blocks. Since a combinational block can be viewed as a complex gate, moving a flip-flop over it

*This work is supported by the NSF under CCR-0238484.

is simply justified. Now we will show that retiming can be generalized to sequential blocks.

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

However, in order for a retiming procedure to move flip-flops over sequential blocks, the timing model for a sequential block must be a graph that connects its inputs to outputs. The traditional approach of treating sequential inputs as primary outputs and sequential outputs as primary inputs will cut off the connections through the block hence make it impossible to move flip-flops over the sequential block.

We will now consider how flip-flops can be moved over timing models of macro-blocks. When the block is a combinational circuit, as shown in Figure 2(a), we can use edges between inputs and outputs to represent the path delays between them, as shown in Figure 2(b). If we place these edges in traditional retiming formulation, flip-flops can be moved over them. However, there are two caveats. First, in traditional retiming, flip-flops may be placed on any edge. In order to avoid flip-flops being placed on the edges we introduced in timing models, we require that the retiming tags of the input and the output connected by such an edge be the same. This means that the number of flip-flops moved over the input is the same as the output. Therefore, no flip-flop will be left in between. Second, depending on the structure of the circuit within the block, an input may not have a path to every output. For example, in Figure 2(a), inputs a and b do not connect to output y, and input c does not connect to output x. If the macro-block is substituted by the edges in its timing model, then, the number of flip-flops moved over x may be different from that over y. On the other hand, if a block is treated as a super-gate, it is required that the retiming tags of the inputs and the outputs are all the same, and this may give sub-optimal solutions. Since the delay edges represent the path topology of the circuit, our timing model for macro-blocks gives us more flexibility.

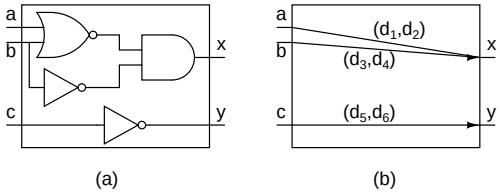

Figure 2: A combinational block and its timing model.

The case for sequential blocks is more tricky. To understand how to build a timing model for a sequential block that can be used for retiming, we use the sequential block shown in Figure 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. Delays from wires can be similarly included. 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 $T - d_1$. To enforce this set-up condition, we introduce a *virtual flip-flop* in the timing model, as shown in Figure 3(b), and add an edge with delay d_1 from a to this virtual flip-flop. Similarly, since the input b has a combinational path to the flip-flop f_2, 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. For example, the output x has a combinational path of delay $d_1 + d_2$ from the output of the flip-flop f_1, thus an edge with delay $d_1 + d_2$ is added from a virtual flip-flop to x. Similar model is used for the output y. In Figure 3(b), all the virtual flip-flops are combined into one flip-flop. It is done to facilitate retiming on the block, and also to make the model simpler. To make the model even more general, we replace the virtual flip-flop by a directed forbidden edge from input to output with delay $-T$ and weight zero. By doing so, all forbidden edges will have zero weight. We will show that as long as the set-up conditions are satisfied, our model for sequential blocks correctly characterizes the timing behavior of the blocks.

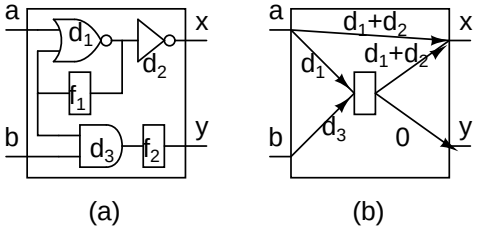

Figure 3: A sequential block and its timing model.

We also need to consider the modeling of a net. Since it is assumed that the (global) routing of nets is given, each net is represented as a Steiner tree. Based on routing positions of the wires, some wire segments of the tree may not accommodate buffers or flip-flops. For example, Figure 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 Figure 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 example, the edge (u, v) in Figure 4(b) represents the wire delay from point u to point v. Applying the timing model, each net becomes a set of edges. Some of the edges, such as (x, y), (u, v), and (u, w) in Figure 4(b), do not allow buffers or flip-flops inserted. But others allow such insertions.

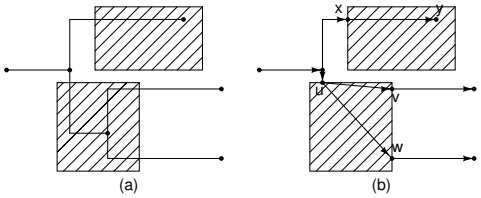

Figure 4: The route of a net and its timing model.

Since the timing model of a macro-block (whether it is combinational or sequential) is composed of a set of edges on which no flip-flop can be placed, and the timing model of a net is composed of a set of edges some of which accommodate buffers and flip-flops but others forbid them, after applying these models, our problem is represented as a directed graph with two types of edges: one allows buffer and flip-flop insertions but the other forbids them. Since the nets we are considering are global interconnections among macro-blocks, they have relatively long lengths. On those wire segments where buffers and flip-flops are allowed, we can use optimal buffering to make their delays to be linear in terms of their lengths [17]. Therefore, we assume that the delays on buffer-allowable edges to be linear.

In summary, in a graph model of the problem, a vertex is used to represent a source or sink of a net, the input or

output of a virtual flip-flop, 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 put on it. On the other hand, flip-flops can be inserted on an edge representing a wire, and the delay of a segment is positive and proportional to its length. This graph model is very general, since it can also be used to represent flip-flop blockages over wires. In that case, wire segments over blockages can be represented as edges without flip-flop insertion. Based on this unified model, 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) 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 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 maximum delay between any two consecutive flip-flops is smaller than T.*

If we can solve this problem, Problem 1 can be solved by using a binary search. Furthermore, this problem is applicable if a designer only wants to target a fixed clock period.

3 Theoretical results

3.1 Notations and constraints

Before discussing the solutions to the two problems, we will first select some essential notations to help us 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 [14] 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. In [14], these variables were sufficient to specify a retiming solution since there was no wire delay and a solution only needed to tell whether a flip-flop was on a wire. In fact, it is not the absolute values of these variables but their differences that are important, since the number of flip-flops on a given edge (u, v) after retiming is given by

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

However, in our problem, a retiming solution must include the positions of flip-flops on each wire. Because retiming can change the number of flip-flops in the system, it is not even known how many flip-flops there will be after retiming. Fortunately, we can overcome the problem by only specifying the arrival time of every vertex with respect to a clock period. For each vertex $u \in V$, we use $t(u)$ to represent its arrival time with respect to the nearest flip-flop on its incoming paths. Given $t(u)$, the positions of flip-flops directly fanning into u can be found, and the positions of other flip-flops can also be computed.

Using these notations, the requirements for a retiming solution can be stated as follows. First, to ensure that no flip-flop is inserted on forbidden edges, we need

$$r(x) = r(y), \quad \forall (x, y) \in E_1. \tag{1}$$

Then, to make sure the availability of flip-flops, it must be true that

$$w(u, v) + r(v) - r(u) \geq 0, \quad \forall (u, v) \in E_2. \tag{2}$$

The following inequalities will guarantee that the arrival times are all achievable.

$$t(y) \geq t(x) + d(x, y), \quad \forall (x, y) \in E_1, \tag{3}$$

$$t(v) \geq t(u) + d(u, v) - (w(u, v) + r(v) - r(u))T, \quad \forall (u, v) \in E_2. \tag{4}$$

Finally, the set-up conditions at the flip-flop inputs are equivalently stated in the following inequalities.

$$0 \leq t(u) \leq T, \quad \forall u \in V. \tag{5}$$

As we can see, the fixed period wire retiming problem is actually a mixed integer linear programming given by (1)-(5). But we need also to set $t(u) = 0$ for every primary input u, and $r(u) = 0$ for every primary input or output u. A general mixed integer linear programming problem is NP-hard. However, this problem can be solved in polynomial time.

3.2 Lower and upper bounds of clock period

From conditions (1)-(5), it is easy to get some lower bounds and upper bounds of the feasible clock periods. First, from the inequalities (3) and (5), it is easy to see that any feasible clock period T must satisfy

$$T \geq T_1 \stackrel{\text{def}}{=} \max_{(x,y) \in E_1} d(x, y).$$

We define the *path delay* $d(p)$ to be the sum of delays of the edges on the path p. If a path actually forms a cycle c, we use $d(c)$ to denote the cycle delay. Similarly, we can define $w(p)$ and $w(c)$. By applying (1), (3), and (4) on any cycle and PI-PO path, we have the following lemma.

Lemma 2 *A feasible clock period T must satisfy*

$$T \geq T_2 \stackrel{\text{def}}{=} \max_{c \in \text{cycle}} \frac{d(c)}{w(c)}, \tag{6}$$

$$T \geq \max_{p \in \text{PI} \rightsquigarrow \text{PO}} \frac{d(p)}{w(p) + 1}. \tag{7}$$

To simplify the notation, we introduce a virtual vertex M as well as directed edges from each PO to it with d=0, w=0 and from it to each PI with d=0, w=1. Thus the meaning of cycle is extended to cover every PI $\rightsquigarrow$ PO path. Therefore, (7) can be incorporated into (6).

It has been shown in the literature [18, 15, 4] that when the forbidden edges are introduced only by gates, the above lower bounds are tight. In fact, we can generalize the result to the following lemma.

Lemma 3 *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 $T_1 + T_2$.*

Since a gate (with only one output) only gives a directed tree with forbidden edges, Lemma 3 subsumes previous results [18, 15, 4]. However, our result also shows that it can be extended to circuits with complex blocks such as multipliers and adders where each output depends on all inputs. Fortunately, the forbidden edges introduced by a net (as shown in Figure 4) are always a directed forest, thus will not give any trouble.

When the topology of forbidden edges does not satisfy the condition in Lemma 3, the optimal clock period may not be upper bounded by $T_1 + T_2$. As an example, consider a circuit shown in Figure 5(a), where the forbidden edges within the macro-block do not form a set of complete bipartite graphs. Suppose the delay and weight of each edge are given as the (delay, weight) label shown in the figure, we have $T_1 = 1$ and $T_2 = 34$. However, $T_1 + T_2 = 35$ is no longer an upper bound of the optimal clock period, since a period of 34 requires a retiming as shown in Figure 5(b), from which a feasible solution cannot be derived by moving only the flip-flops within the block.

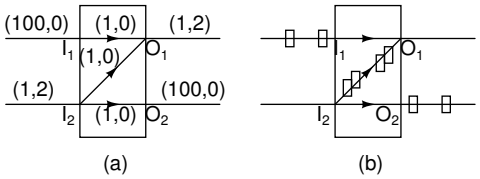

Figure 5: A macro-block with multiple outputs.

Cong and Yuan [5], in a study of multilevel placement and retiming, had the same problem with multiple-output clusters. To find an upper bound, they treated each cluster as a vertex which is equivalent to connecting all outputs of a cluster to a super output with edges of zero delay. Used on the circuit shown in Figure 5(a), their approach will add edges from O_1 and O_2 to a super output and thus the upper bound will be 202. As we can see from this example, even though their approach gives a correct upper bound, the bound is loose. Without any retiming, Figure 5(a) gives a much better upper bound of 101.

Our approach to find a tighter upper 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 form 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. In the example in Figure 5(a), an optimal retiming without considering forbidden edges is shown in Figure 5(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 $3 \times 34 = 102$.

3.3 Fixed period wire retiming

From now on, we will consider how to check whether a clock period T that satisfies the above lower bounds can be realized by retiming. We will use an approach similar to Leiserson and Saxe [14] to solve this problem in polynomial time.

For any path p in the graph, we define the sequential delay $\mathrm{sd}(p)$ as follows

$$\mathrm{sd}(p) = d(p) - w(p)T$$

where T is the clock period. And we define the sequential delay $\mathrm{sd}(u, v)$ of any two vertices u, v to be

$$\mathrm{sd}(u, v) = \max_{p \in u \rightsquigarrow v} \mathrm{sd}(p).$$

Then, consider the following set of inequalities

$$r(v) - r(u) \geq \lceil \mathrm{sd}(u, v)/T \rceil - 1 \quad \forall u \neq v \in V \tag{8}$$

Using the definition of the sequential delay, for any edge $(u, v) \in E_2$, (8) means that

$$w(u, v) + r(v) - r(u) \geq \lceil d(p)/T \rceil - 1 \geq 0.$$

Thus, (8) implies (2). However, the importance of (8) is more than that. Under the lower bound condition of (6), formula (1) and (8) give a solution to the fixed period wire retiming problem.

Theorem 1 *The fixed period wire retiming problem is feasible if and only if (1), (6) and (8) have a solution.*

Proof: Suppose T is a feasible clock period, then considering any two vertices u, v, any path p between them would have $w(p) + r(v) - r(u)$ flip-flops. To make sure that the delay between any two consecutive flip-flops is not larger than T, we must have

$$d(p) \leq (w(p) + r(v) - r(u) + 1)T$$

which is $r(v) - r(u) \geq (d(p) - w(p)T)/T - 1$, for any path p, or $r(v) - r(u) \geq \lceil \mathrm{sd}(u, v)/T \rceil - 1$. The above inequality also justifies the forbidden edges with delay $-T$.

The other direction is more difficult, since besides the number of flip-flops on each edge we also need to find the positions for those flip-flops such that the delay between any two consecutive ones is upper bounded by the clock period. In other words, we need to prove that there exists a solution for (3) -(5) determined by the solution of (1), (6) and (8).

Since (6) is true, once (1) and (8) have a solution, we can always compute a solution for all $t(u)$ satisfying (3), (4) and $t(u) \geq 0$ by applying Bellman-Ford's algorithm [6], because it will not report a positive cycle under (6). In the meanwhile, for any $v \in V$ whose $t(v) > 0$, there must exists at least one u such that $(u, v) \in E$ and $t(v) - t(u) = \mathrm{sd}(u, v) - (r(v) - r(u))T$. Since we have set $t(u) = 0$ for every primary input, we now prove that such solution also satisfies the requirement $t(u) \leq T$ for all $u \in V$.

For the seek of a contradiction, we assume there exists such a vertex v that $t(v) > T$. Starting from v we trace back along those critical edges (u, v) such that $t(v) - t(u) = \mathrm{sd}(u, v) - (r(v) - r(u))T$ until we reach a vertex u whose $t(u) = 0$. In the worst case, u is a primary input. Now for $t(u)$ and $t(v)$, we have

$$t(v) - t(u) = t(v) = \mathrm{sd}(u, v) - (r(v) - r(u))T.$$

From (8), we know that

$$\begin{aligned} & \mathrm{sd}(u, v) - (r(v) - r(u))T \\ \leq\ & \mathrm{sd}(u, v) - \lceil (\mathrm{sd}(u, v))/T \rceil T + T \\ \leq\ & T \end{aligned}$$

Hence $t(v) \leq T$, which contradicts our assumption. □

4 Algorithms

4.1 Detailed description

In this section, we give an algorithm (Figure 6) to solve both the fixed and minimum period wire retiming problems. The

operation of our algorithm can be broken down to three main steps.

In the first step, we apply an efficient algorithm to find the lower bound of the feasible clock periods. The formula of T_2 actually reflects a cycle property of the graph also known as *maximum cycle ratio problem* (MCRP). Let $d(c)$, $w(c)$ denote two parameters associated with each cycle c in G. The maximum cycle ratio ρ^* is then defined as

$$\rho^* = \max_c \frac{d(c)}{w(c)}, w(c) > 0.$$

In our application, $d(c)$ is the cycle delay and $w(c)$ is the number of flip-flops in the cycle. Then the value of ρ^* is exactly what we want for T_2. To solve MCRP, a lot of algorithms have been designed and presented such as Burns' [1], Lawler's [13], Howard's [3, 9], etc. In terms of complexity, Burns' takes $O(|V||E|)$, Lawler's takes $O(|V||E|\lg(|V|W))$, where W is the maximum edge delay, and Howard's takes $O(N|E|)$, where N is the product of the out-degrees of all the vertices in G. In [9], some popular algorithms that were widely used in CAD community were systematically compared and their comprehensive experimental results revealed that Howard's algorithm was by far the fastest algorithm though the only known bound of its running time is exponential. In our implementation, we adopt an improved version of Howard's algorithm [3, 9]. After T_2 is obtained, we compute a tight upper bound of the feasible clock periods in the way as described in section 3.2.

In the second step, a binary search is used to find the optimal clock period. Given a particular T, we need to apply Johnson's all-pair shortest path algorithm [6] first to compute all $\mathrm{sd}(u,v) = \max_{p \in u \rightsquigarrow v} \mathrm{sd}(p)$. Based on the results, we create a new graph rG incorporating all the vertices in V and edges (u,v) with weight $\lceil \mathrm{sd}(u,v)/T \rceil - 1$ if there exists a path from u to v in G, for all $u \neq v \in V$. We also need to introduce a virtual vertex M in rG as well as directed edges from each PO to it and from it to each PI with zero weight. Then we apply Bellman-Ford's algorithm to check if there exists a positive cycle in rG. When it terminates, we can decide how to adjust the two bounds accordingly. Note that once rG was created, its structure was kept throughout the rest of the algorithm and its edge weights were recomputed and updated every time T was changed.

In the third and last step, we use the optimal clock period and corresponding $r(u)$ computed in step two to calculate $t(u)$ for all $u \in V$. Due to Theorem 1, a feasible solution for $t(u)$ is guaranteed.

4.2 Pruning and optimization

Further examining (8), we found that some inequalities are actually redundant. For example, if $(u,v) \in E_2$, $(v,y) \in E_1$ and $d(v,y) > 0$, according to (8)

$$\begin{aligned} & r(y) - r(u) \geq \lceil \mathrm{sd}(u,y)/T \rceil - 1 \\ \geq\ & \lceil (\mathrm{sd}(u,v) + d(v,y))/T \rceil - 1 \\ \geq\ & \lceil \mathrm{sd}(u,v)/T \rceil - 1 \end{aligned}$$

Since $r(v) = r(y)$, the inequality above actually implies $r(v) - r(u) \geq \lceil \mathrm{sd}(u,v)/T \rceil - 1$ which then becomes redundant. Generally speaking, we call an inequality in (8) a redundant inequality if it can be implied by other inequalities in (8) that have not yet been proved to be redundant.

To distinguish from the graph G, we call rG *retiming* graph and its edges *retiming* edges. We now present two pruning techniques which could help to reduce the redundancy of (8).

First, if there exist three distinct vertices u, v, y such that

$$\lceil \mathrm{sd}(u,y)/T \rceil - 1 + \lceil \mathrm{sd}(y,v)/T \rceil - 1 = \lceil \mathrm{sd}(u,v)/T \rceil - 1,$$

and either of $r(y) - r(u) \geq \lceil \mathrm{sd}(u,y)/T \rceil - 1$ and $r(v) - r(y) \geq \lceil \mathrm{sd}(y,v)/T \rceil - 1$ has not yet been proved to be redundant, then the inequality $r(v) - r(u) \geq \lceil \mathrm{sd}(u,v)/T \rceil - 1$ is redundant by definition and can be safely deleted.

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

`Find the upper bound` T_u `and lower bound` T_l;
`Create` rG;
do
  $T = \frac{T_u + T_l}{2}$;
  `Find` $\mathrm{sd}(u,v) = \max_{p \in u \overset{p}{\rightsquigarrow} v} \mathrm{sd}(p)$ `by Johnson's;`
  `For each` $u \overset{p}{\rightsquigarrow} v$ `do`
    `Update` (u,v) `in` rG `with weight` $\left\lceil \frac{\mathrm{sd}(u,v)}{T} \right\rceil - 1$;
  `Apply Bellman-Ford's to` rG;
  `If` T `is feasible then`
    `Record` $r(u)$ `for all` $u \in V$;
    $T_u = T$;
  `Else`
    $T_l = T$;
while $T_u - T_l > \epsilon$;
`Use` T `and` $r(u)$ `to compute` $t(u), \forall u \in V$.

Figure 6: Pseudocode of the retiming algorithm

Second, for the forbidden edges with positive delay, all retiming edges from the ending points are redundant since their effects are implied by the retiming edges from the starting points. For the same reason, all retiming edges to the starting points of those forbidden edges are redundant too. By deleting the redundant inequalities, we can reduce the number of retiming edges dramatically. Although we did not improve the asymptotic complexity, we did get great benefit in reducing the running time in reality.

Using the two techniques above, the size of rG may be much smaller but we still have to run Bellman-Ford's algorithm during each test of the binary search. Since the outcome of Bellman-Ford's algorithm gives not only feasibility answer but all corresponding $r(u)$ values which are more than what we need during most of the tests. Instead, we can take advantage of the high efficiency of Howard's algorithm to calculate the maximum cycle ratio of rG. If during the calculation, it discovers a cycle with positive weight, current test is immediately finished. Hence, a second scheme to substitute the Bellman-Ford's algorithm is to use Howard's algorithm during the process of optimal period searching. After that, only one pass of Bellman-Ford's algorithm is needed to obtain those corresponding $r(u)$ values.

4.3 Computational complexity

In [8], the running time of Howard's algorithm is bounded by $O(|V||E|\alpha)$ and $O(|V|^2|E|(\omega_{max} - \omega_{min})/\epsilon)$, where α is the number of simple cycles in G, ω_{max} and ω_{min} are the maximum and minimum edge delays and ϵ is the precision. While in practice, using Howard's to get the lower bound takes much less time than doing the binary search.

Johnson's all-pair shortest path algorithm uses the Bellman-Ford's algorithm and Dijkstra's algorithm [6] as subroutines. Since we want to find the maximum sequential delay between any (u,v) pair if v is reachable through some path from u, we actually run Johnson's algorithm to solve a longest path problem. Its running time is $O(|V|^2 \lg |V| + |V||E|)$ if the priority queue in Dijkstra's algorithm is implemented by a Fibonacci heap [6].

As expected, the dominant part of running time is con-

Table 1: Experimental Results

Circuit	$\|V\|$	$\|E\|$	T_2	w/o non-CB blocks			w/ non-CB blocks			
				No.Iter	time(sec)	T_{opt}	No.Part	No.Iter	time(sec)	T_{opt}
myex	21	24	13.0	7	0.00	18.7	2	10	0.01	24.0
s386	519	700	51.0	5	1.97	51.1	50	10	3.67	55.0
s400	511	665	32.2	6	1.64	32.2	50	10	3.38	50.6
s444	557	725	35.0	5	2.23	35.2	40	10	4.31	63.2
s838	1299	1206	76.0	5	8.79	76.0	130	11	33.42	84.0
s953	1183	1515	60.6	5	9.76	60.6	110	10	17.56	69.5
s1238	1581	2100	100.2	5	7.88	100.3	150	11	28.45	100.3
s1488	2054	2780	70.1	5	35.17	70.6	200	11	98.88	73.3
s1494	2054	2792	76.8	5	34.13	76.9	160	11	62.86	80.0
s5378	7205	8603	111.0	5	684.60	111.2	500	13	1344.74	115.3

sumed in applying Bellman-Ford's algorithm to rG. In the worst case, rG is a complete graph and consists of $|V|^2$ number of edges. Solving this will take $O(|V|^3)$ time. Therefore, the total running time inside the binary search loop is $O(|V|^3)$ and the time complexity for the whole algorithm is $O(|V|^3 \lg \frac{T_u - T_l}{\epsilon})$.

5 Experimental results

We implemented the algorithm in a PC with two 2.4GHz/512K Xeon CPUs and 1GB RAM. We performed retiming on the ISCAS-89 benchmark suite. In absence of delay information for ISCAS-89 circuits, we randomly assign delay values between 1.0 and 2.0 units to gates (we treat them as macro-blocks) and 0.2 to 5.0 to wires. In terms of the chip level we are focusing on, 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 [12] 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 an important role in determining the percentage of non-complete bipartite blocks. For simplicity, we did not further apply our timing model to the partitions when generating the results. In addition, the lower bound T_2 of each circuit is reported as a comparison with the optimal clock periods we computed.

In practical implementation, we found that the benefit we got using the first pruning technique presented in section 4.2 did not actually pay off the time penalty. The $\theta(|V|^3)$ complexity for the first technique is tight hence becomes the dominant part of the total running time. Therefore, we only apply the second pruning technique and compare the running time of the two schemes mentioned in section 4.2. Experimental results show that the performances of the two schemes are in the same level. Thus, we only report the running time of the second scheme in Table 1.

6 Conclusions and future work

The wire retiming problem that relocates existing flip-flops to multiple clock period interconnections is formulated with the help of the block timing models. Similar to Leiserson and Saxe [14], a set of integer difference inequalities is shown to be both necessary and sufficient, thus gives a polynomial time algorithm. Our current and on-going work is to investigate a more efficient algorithm similar to [19] for the wire retiming problem. The preliminary results are very encouraging.

References

[1] S.M. Burns. *Performance analysis and optimization of asynchronous circuits*. PhD thesis, California Institute of Technology, 1991.

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

[3] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Proc. IFAC Conf. on Syst. Structure and Control*, 1998.

[4] J. Cong and S. K. Lim. Physical planning with retiming. In *ICCAD*, pages 2–7, November 2000.

[5] J. Cong and X. Yuan. Multilevel global placement with retiming. In *DAC*, pages 208–213, Anaheim, CA, 2003.

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

[7] A. J. Daga, L. Mize, S. Sripada, C. Wolff, and Q. Wu. Automated timing model generation. In *DAC*, pages 146–151, 2002.

[8] A. Dasdan, S. Irani, and R.K. Gupta. An experimental study of minimum mean cycle algorithms. Technical Report 98-32, Univ. of California, Irvine, July 1998.

[9] Ali Dasdan, Sandy S. Irani, and Rajesh K.Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio. In *DAC*, 99.

[10] M. Foltin, B. Foutz, and S. Tyler. Efficient stimulus independent timing abstraction model based on a new concept of circuit block transparency. In *DAC*, pages 158–163, 2002.

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

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

[13] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[14] 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.

[15] N. Maheshwari and S. S. Sapatnekar. Optimizing large multi-phase level-clocked circuits. *IEEE TCAD*, 18(9):1249–1264, September 1999.

[16] C. W. Moon, H. Kriplani, and K. P. Belkhale. Timing model extraction of hierarchical blocks by graph reduction. In *DAC*, pages 152–157, 2002.

[17] R. H. J. M. Otten and R. Brayton. Planning for performance. In *DAC*, pages 122–127, 1998.

[18] P. Pan, A. K. Karandikar, and C. L. Liu. Optimal clock period clustering for sequential circuits with retiming. *IEEE TCAD*, 17(6):489–498, June 1998.

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

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