iRetILP: An efficient incremental algorithm for min-period retiming under general delay model

Debasish Das*, Jia Wang†, and Hai Zhou
*Place and Route Group, Mentor Graphics, San Jose, CA 95131
†ECE, Illinois Institute of Technology, Chicago, IL 60616
EECS, Northwestern University, Evanston, IL 60208

Abstract— Retiming is one of the most powerful sequential transformations that relocates flip-flops in a circuit without changing its functionality. The min-period retiming problem seeks a solution with the minimal clock period. Since most min-period retiming algorithms assume a simple constant delay model that does not take into account many prominent electrical effects in ultra deep sub micron vlsi designs, a general delay model was proposed to improve the accuracy of the retiming optimization. Due to the complexity of the general delay model, the formulation of min-period retiming under such model is based on integer linear programming (ILP). However, because the previous ILP formulation was derived on a dense path graph, it incurred huge storage and running time overhead for the ILP solvers and the application was limited to small circuits. In this paper, we present the iRetILP algorithm to solve the min-period retiming problem efficiently under the general delay model by formulating and solving the ILP problems incrementally. Experimental results show that iRetILP is on average $100\times$ faster than the previous algorithm for small circuits and is highly scalable to large circuits in term of memory consumption and running time.

I. Introduction

Retiming is a powerful sequential circuit optimization technique that relocates flip-flops (FF) within a circuit without altering the functionality. As relocating FFs balances the critical path and reduces the states of the circuit, retiming transformation can be employed to reduce the clock period and FF area of a circuit. In this paper we focus on min-period retiming problem where retiming optimizations are employed to achieve the optimal clock period of a circuit under retiming transformation. Conventionally, architectural level techniques for retiming optimization were applied early in the design process with relatively simple delay models. However, as feature sizes shrink down, interconnect delay, input slew rate at gates, fanout load and clock skews are increasingly becoming the important factors to decide the performance of circuit. As the effects mentioned above cannot be modeled accurately at high abstraction levels with simple delay models, retiming techniques must be used as a post-placement optimization where the estimation of the above factors are available.

Under a simple constant delay model, the min-period retiming problem was formulated and solved in polynomial time by Leiserson and Saxe [1]. Improvements to the original algorithm [2], [3], [4], [5], [6] were proposed later to improve the efficiency and to consider more factors including verifiability and hold conditions. Recently Hurst et al. [16] proposed an efficient SAT based algorithm for initial state feasibility of the retiming. We propose to use such an technique to solve the initial state feasibility problem in our algorithm.

Retiming as a post-placement optimization technique with general delay models were proposed in [7], [8]. Accurate timing information is considered in the general delay model proposed by [7], including interconnect loading, combinational gate loading, sequential register loading and drive capability, and register propagation delay and clock skews. Based on this model, [8] classified the circuits into two classes: begin/end extendible and two-way extendible. Begin/end extendible circuits follow the key property of path delay monotonicity as the retiming problem under the simple constant delay model. It ensures that the propagation delay along any combinational path increases monotonically with the number of logic gates on it. In other words, whenever the delay along a FF-to-FF path exceeds the specified clock period, the timing violation can be fixed by moving the two bounding FFs closer to each other.

However, the monotonicity property might not hold for all the circuits for the general delay model. Under such circumstance, the circuits are called two-way extendible circuits and an integer linear programming (ILP) based algorithm was proposed in [8] to solve min-period retiming. We call this algorithm for two-way extendible circuits as *Retime-General*. Beginning with an initial clock period, *Retime-General* iteratively decreases the clock period linearly and checks the feasibility of the decreased clock period by formulating and solving an ILP program. As the ILP program is formulated on a dense path graph including paths that may become combinational and violate the clock period, the huge number of constraints incur tremendous storage and running time overhead for the ILP solvers so that the algorithm was only experimented on small circuits in [8].

In this paper, we present the iRetILP algorithm to solve the min-period retiming problem efficiently under the general delay model for two-way extendible circuits. Unlike *Retime-General* where all the constraints in the ILP formulation are generated at the same time, iRetILP incrementally generates those constraints in an approach similar to [5] on the fly. Practically, this incremental approach allows to decompose the previous single ILP problem for feasibility checking with the tremendous number of constraints, into multiple ILP problems with far less number of constraints. Though more ILP problems should be solved, the reduced problem sizes translate to significant reductions in both storage and running time of our iRetILP algorithm. Moreover, iRetILP removes the need of a linear search and clock period decrease factor which were needed by *Retime-General*. Experimental results show that iRetILP is on average $100\times$ faster than the previous algorithm for small circuits and is highly scalable to large circuits which cannot be solved by *Retime-General*. For the small circuits where memory consumption data for comparison are available from the experiments, iRetILP uses as little as $\frac{1}{40}$ memory compared to *Retime-General*.

The rest of this paper is organized as follows. We present the min-period retiming problem formulation for the general delay model in Section II. Our algorithmic idea is presented in Section III. The details of our iRetILP algorithm is presented in Section IV. Experimental results are presented in Section V. Finally we draw our conclusions in Section VI.

II. Problem Formulation

In this section we present details of general delay model used in our paper and formulate the minimum period retiming problem for general delay model.

*This work was supported by NSF under CCF-0811270.

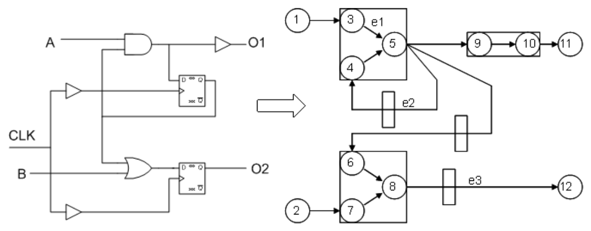

Fig. 1. Graph Model Generation

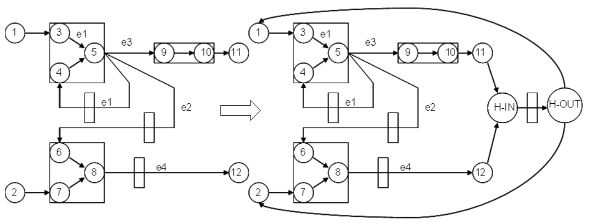

Fig. 2. Extended Graph

Our delay model is similar to the model used by [8] where clock skews, FF setup times, propagation delays and interconnect delays are considered. We are assuming that the sequential circuit for retiming optimization is obtained from physical design stage where the placement of cells and global routing are given [9]. We model the sequential circuit as a directed acyclic graph $G = (V, E)$. A vertex $v \in V$ represents either the input or output port of a combinational cell, or the primary input ot output port. An edge $(i, j) \in E$ belongs to either the set C of the combinational cell edges representing the computational dependency from the input port i of a combinational cell to the output port j, or the set I of the interconnect edges representing the interconnect connecting i to j. For each combinational cell edge e, we associate labels $\delta(e)$ and $\Delta(e)$ to represent minimum and maximum load dependent gate delay. These are feasible lower and upper bounds on the gate delays during retiming. For each interconnect edge e, we associate 4 labels: $d(e)$ representing the interconnect delay without FF on the interconnect, $\alpha(e)$ and $\beta(e)$ representing the delays of the wire segments if there is at least 1 FF on the interconnect, and $w(e)$ representing the number of the FFs on the interconnect.

An example of a sequential circuit and our model transformation is presented in Figure 1. Primary inputs of the circuit are refered as A and B while the two primary outputs are $O1$ and $O2$. We show some of the interesting edges from the graph. Edge $e1$ between vertices 3 and 5 show an example of combinational edge in AND gate. Edges $e2$ and $e3$ have FFs on them and $w(e2)$ and $w(e3)$ are each equal to 1.

A brief description of electrical characteristics of the timing model is essential. First consider combinational cell edge $e1$ from Figure 1. It has three interconnect edges associated with it. Two labels δ and Δ are associated with edge $e1$. Based on first moments of an AWE like technique, labels δ and Δ are computed using the minimum and maximum loading capacitance seen by port 5 during the whole retiming process based on the input slew at port 3. For interconnect edges such as $e2$ we compute $d(e2)$ as the delay due to interconnect resistance if there are no register on the edge. α and β are the delays when there is an associated register with the interconnect edge $e2$.

We propose an extension to G for consistent treatment of all timing constraints as shown in Figure 2. We add two vertices H-IN (Host input) and H-OUT (Host output). We add a virtual FF on the H-IN to H-OUT edge. Delay parameters on this edge are 0. Logically we can see adding two vertices in the graph as the splitting of the host node from the original retiming paper [1]. The virtual FF allows every timing path in the circuit to terminate at H-IN and it helps in consistent generation of the constraints in iRetILP algorithm. For each primary output po, an edge $(\text{po}, \text{H-IN})$ is added to E with $d_{\text{I,po}} = 0$. Similarly for each primary input pi, an edge $(\text{H-OUT}, \text{pi})$ is added to E with $d_{\text{I,pi}} = 0$.

To guarantee that the FFs after retiming are actually a relocation of old ones, a label $r : V \rightarrow \mathcal{Z}$ is used to represent how many FFs are moved from the outgoing edges to the incoming edges of each vertex. The number of FFs on an edge (i, j) after retiming can be computed as $w_r(i, j) \triangleq w(i, j) + r(j) - r(i)$. It is straight-forward that the following constraints must be satisfied:

$$\begin{aligned} &P0(r): \\ &w_r(i,j) \geq 0, \forall (i,j) \in E \\ &w_r(i,j) = 0, \forall (i,j) \in C \\ &w_r(\text{H-OUT}, \text{H-IN}) = 1 \end{aligned}$$

For the ease of presentation, we focus on the setup constraints in this paper but our algorithm can be easily extended for hold constraints similar to [8]. After introducing the label $T : V \rightarrow \mathcal{R}^+$ to denote the latest arrival times at each vertex, the setup timing constraints of the sequential circuit with FF as memory elements can be modeled by the following inequalities via block-based timing analysis.

$$\begin{aligned} &P1(r, \phi) : \exists T \text{ such that:} \\ &T(i) + \Delta(i,j) \leq T(j), \forall (i,j) \in C \\ &T(i) + d(i,j) \leq T(j), \forall (i,j) \in I \wedge w_r(i,j) = 0 \\ &T(i) + \alpha(i,j) \leq \phi \wedge T(j) \leq \beta(i,j), \forall (i,j) \in I \wedge w_r(i,j) \geq 1 \end{aligned}$$

The latest arrival times T can be computed through block based static timing analysis techniques similar to timing analysis methodology proposed by [10], [11]. The following theorem computes the clock period of the sequential circuit represented by G.

Theorem 1: The clock period of the circuit G after retiming r is given by

$$\phi(r) = \max\{T(i) + \alpha(i,j) : \forall (i,j) \in I \wedge w_r(i,j) \geq 1\}.$$

Given an assignment of w_r on each edge of the timing graph G which we define by vector F, our timing analysis algorithm TA provides the following three procedures. The get-period procedure computes the clock period using Theorem 1. It is straight-forward that the clock period must be equal to the total delay along a combination path. Let the path starts with the edge e and ends at the edge e'. Then we must have $w_r(e) > 0$ and $w_r(e') > 0$ and the edges e and e' respectively contribute $\beta(e)$ and $\alpha(e')$ to the clock period. Let the get-head-edge procedure returns e' and the get-tail-edge procedure returns e. The complexity of the algorithm TA is stated in the following theorem.

Theorem 2: Given a graph G of n vertices and m edges, the algorithm TA runs in $O(m + n)$ time.

Based on the above discussion, we formulate the following *Generalized Setup Retiming* problem.

Problem 1 (Generalized Setup Retiming): Given a circuit $G = (V, E)$ and the number of FFs w on the edges, find a retiming r such that the constraints $P0(r)$ and $P1(r, \phi)$ can be satisfied with the minimum clock period ϕ.

For the ease of presentation, we use $w(p)$ and $w(c)$ to denote the number of FFs on a simple path $p = i \mapsto j$ and a cycle c before retiming, and $w_r(p)$/$w_r(c)$ to denote the number of FFs on *p*/*c* after the retiming r.

III. Algorithm Overview

In this section we review algorithm Retime-General and identify its drawbacks which motivated us to develop our incremental algorithm iRetILP. We present the general ideas behind the design of iRetILP without delving into technical details, which will be presented in the next section.

Lalgudi et al. [8] referred Problem 1 as the minimum period retiming problem for two-way extendible circuits and proposed an integer linear programming algorithm to solve Problem 1. Based on our modeling approach it is evident that any practical VLSI circuit can be transformed into graph G and therefore solving Problem 1 efficiently (both in terms of performance and memory) has practical significance. Most of the recent advances in incremental minimum period retiming [5], [6] focussed on simple delay models which enjoyed the path delay monotonicity property as presented below

Property 1 (Path Delay Monotonicity): Delay of a register-to-register path decreases as the number of combinational elements on the path is decreased.

For a general delay model used in this paper Property 1 does not hold. In literature Lalgudi et al's algorithm is the most efficient approach to solve Problem 1. We refer to this algorithm as *Retime-General*. *Retime-General* extended the classical approach by Leisserson and Saxe to generate matrices W and D. Given a timing graph G with n vertices and m edges, W is a matrix of of size $n \times n$ while D is a matrix of size $m \times m$. W is still generated in the classical sense where each entry $W[u][v]$ here indicates the minimum register count amongst all possible combinational paths between vertex u and v. Formally W can be written as

$$W(u,v) \quad = \quad min\{w(p) : u \rightsquigarrow_p v\} \tag{1}$$

Equation 1 specifies the paths $u \rightsquigarrow_p v$ in G that can become combinational (and possibly critical) in retimed graph G_r. Given a pair of edges $e = (\hat{u}, u)$, $\hat{e} = (v, \hat{v})$, D(e,$\hat{e}$) is computed as follows

$$D(e,\hat{e}) \quad = \quad max\{\Omega(e,p,\hat{e}) : \hat{u} \rightarrow_e u \rightsquigarrow_p v \rightarrow_{\hat{e}} \hat{v}, \tag{2}$$
$$w(p) = W(u,v)\} \tag{3}$$

Ω in Equation 2 represents the longest propagation delay from a register on e to a register on $\hat{e}$ along any path between these two edges that can become combinational after retiming. Therefore Equation 2 is sufficient to compute maximum propagation delay from edge e to edge $\hat{e}$ in any retiming graph G_r derived from G by r transformation. Equation 1 is used to generate all possible clock periods for the sequential circuit. Elements of matrix D can be computed using a all-pairs shortest-paths algorithm. Given a graph G with n vertices and m edges, following lemma presents the complexity of clock period generation algorithm

Lemma 1: Matrix D can be computed in O($m^2 + n^2 \log n$) or O(n^3) time using Johnson or Floyd-Warshall algorithm [12] and requires memory storage of O(m^2).

Computation of matrix D and W is a preprocessing stage for the integer linear programming formulation used by Retime-General. Given graph G for a sequential circuit we obtain circuit G_r using a retiming transformation. The following lemma from Lalgudi et al's work state necessary and sufficient conditions for G_r to satisfy setup constraints for a specified clock period c.

Theorem 3: Let G_r be a graph with retiming transformation $r : V \rightarrow \mathcal{Z}$ and c be a positive real number. c is a feasible clock period for G_r if and only if for every edge $u \rightarrow_e v \in I$, we have

$$w_r(e) \geq 0$$

and for every edge pair e, $\hat{e} \in I$ such that $\hat{u} \rightarrow_e u \rightsquigarrow v \rightarrow_{\hat{e}} \hat{v}$ and D(e,$\hat{e}$) $> c$, we have

$$W_r(u,v) = 0 \Rightarrow (w_r(e) = 0 \vee w_r(\hat{e}) = 0) \tag{4}$$

Given an initial assignment of register on each edge i.e. vector F and the timing graph G, we invoke our timing analysis algorithm TA to generate an upper bound on clock period ϕ_{ub}. After that a linear search is used to genreate the consecutive clock period $c < \phi_{ub}$ and feasibility of this clock period is checked using Theorem 3. *Retime-General* decreases the clock period c linearly until the period becomes infeasible. Then the period larger than the infeasible period is declared as optimal clock period by *Retime-General*.

We use a modification to $P0(r)$ based on the hold constraint violation idea suggested by Chuan et al. [6].

$$PO^*(r) \quad \triangleq \quad \forall (i,j) \in E : 0 \leq w_r(i,j) \leq 1 \tag{5}$$

Condition $PO^*(r)$ is a stronger condition than $P0(r)$ as it takes into account the fact that a timing edge cannot accomodate more than one register without causing hold violations among the registers. Equation 5 also help us to simplify the integer linear programming formulation derived from Theorem 3. We give a corollary to Theorem 3 to specify the integer linear programming formulation used in our implementation of *Retime-General*. Note that because of Equation 5 we save the additional overhead of generating the companion graph needed in the integer linear programming formulation in [8].

Corollary 1: Let G_r be a graph with retiming transformation r and c be a positive real number. c is a feasible clock period if and only if for every edge $u \rightarrow_e v \in I$, we have

$$0 \leq w_r(e) \leq 1$$

for every edge $u \rightarrow_e v \in C$, we have

$$w_r(e) = 0$$

and for every edge pair e, $\hat{e} \in I$ such that $\hat{u} \rightarrow_e u \rightsquigarrow v \rightarrow_{\hat{e}} \hat{v}$ and D(e,$\hat{e}$) $> c$, we have

$$w_r(e) + w_r(\hat{e}) <= 1 + W_r(u,v) \tag{6}$$

Here we would like to draw reader's attention to the computational difficulties involved with solving the ILP formulation presented by Corollary 1. For the formulations with simple model [13], [3], [5], [6] and begin/end extendible circuits which enjoy Property 1, Equation 6 reduces to a linear difference constraint with 2 variables, $r(u)$ and $r(v)$ respectively. Other constraints from the formulation are already linear difference constraint with 2 variables. Feasibility checking of integer linear programs having difference constraints can be done quite efficiently using Bellman-Ford algorithm [12] in O($mn + n^2$) time. Due to complexity of our delay model, Equation 6 is no longer a linear difference constraint with 2 variables but essentially a linear constraint with 4 integer variables, $r(u)$, $r(\hat{u})$, $r(v)$ and $r(\hat{v})$. For such problems no efficient algorithms were proposed in literature and therefore *Retime-General* employed general purpose integer linear programming techniques (Branch and Bound, Branch and Cut) implemented in a commercial solver [15] to solve the feasibility checking of clock period c.

Now there were several shortcomings to the ILP formulation proposed in Corollary 1 which resulted in poor efficiency of *Retime-General*. Complexity of feasibility checking of an integer linear program depends on the number of constraints used in the program. Corollary 1 uses dense matrix D to generate the constraints and therefore uses many redundant constraints which are not critical for the feasibility checking of clock period c. One of the novel ideas we used in design of iRetILP is to minimize the number of constraints and generate only the critical constraints used for feasibility checking of a clock period.

Rather than using the idea of pruning the entries of D matrix which has been used by researchers in past ([3] for min period and [14] for minaret) we designed iRetILP to completely remove the dependance of D and W matrices to generate a feasible clock period. W matrix is used in Equation 6 to compute $W(u, v)$. This clearly saves the time needed to run an all pair shortest path algorithm to generate the matrices and the memory needed to save D and W matrices. Generating only the critical constraints to prove the feasibility of a clock period c motivated us to use a timing analysis algorithm TA incrementally. Each iteration of iRetILP generates a critical constraint from the matrix D to prove the feasibility of a given clock period c. Therefore iRetILP ended with only a subset of constraints from matrix D which are critical to minimize the clock period of a circuit. Our experimentals results validate that these constraints are much smaller than the total constraints used in the traditional formulation given by Corollary 1. Due to fewer constraints iRetILP solves the feasibility checking of a clock period c much faster than the tradtional algorithm.

Finally iRetILP also does not need any clock period decrease factor ϵ which is needed to drive the ILP formulation in *Retime-General* algorithm. Once we prove the feasibility of the clock period c, the next step in *Retime-General* algorithm is to find a new clock period by

$$c_n = c - \epsilon$$

Each iteration of algorithm *Retime-General* generates a new clock period c_n and check the feasibility of the period c_n by formulating the ILP using Corollary 1. *Retime-General* terminates when c_n is no longer feasible. In our implementation we improved *Retime-General* by using a timing analysis run after a clock period c_n is checked for feasibility. Once we find that c_n is feasible, we apply the retiming transformation r to the graph G and generate the flip-flop vector F. After that we call our timing analysis algorithm to generate the clock period ϕ for the transformed graph G_r and generate the new clock period for feasibility checking by

$$c_n = \phi - \epsilon$$

Because of incremental timing analysis run, our implementation of *Retime-General* decrease number of iterations of the original algorithm proposed by [8] and make it more efficient but it still has a factor ϵ which is artificial. The factor ϵ is the approximation factor of *Retime-General* algorithm. A more favorable algorithm design idea would be to remove the approximation factor from the algorithm completely and iRetILP achieves that by generating the critical constraints. Each iteration of iRetILP generates a critical constraint and the minimum clock period of all such critical constraints is the optimal clock period that a circuit can achieve.

IV. Algorithm Description

In this section we discuss the theoretical details of an algorithm to solve Problem 1 for a special class of circuits called one-way extendible circuits. Experimental results confirming such algorithm already appear elsewhere in literature [5] and therefore we do not repeat them in this paper. Followed by that we present details of algorithm iRetILP for solving the generic case defined by Problem 1.

One-way extendible circuits are a special class of two-way extendible circuits where Property 1 holds if we increase (delay increases) or decrease (delay decreases) the number of combinational nodes in a path. We give the following theorem to solve the special case

Theorem 4 (Monotonic Retiming): If the circuit is one-way extendible, then Problem 1 can be solved using an extension of Zhou's algorithm [5] in O(n^2m) time and O(1) memory.
Algorithm proposed by Lalgudi et al. [8] to solve this problem has a time complexity of O($n^3|F|\log(n)$) where F denotes the maximum number of registers on a cycle in the circuit. Also Lalgudi et al.'s algorithm has a memory complexity of O(m^2). Algorithm presented by Theorem 4 has the advantage of having provable better bounds than Lalgudi's algorithm. Next we present the technical details of algorithm iRetILP.

A. Algorithm iRetILP

Our proposed algorithm iRetILP has 3 main steps:Initialization, Iterations and Termination. The detailed algorithm is presented in Figure 3. The initialization step begins with a timing graph G and a initial assignment of register on each edge of G which satisfy Equation 5. Once we generate the vector F from the register assignment, algorithm TA is invoked to get the initial clock period ϕ. Let the optimal clock period for Problem 1 be ϕ^* and the register vector corresponding to the optimal clock period be F^* . We initialize ϕ^* and F^* in Line 5 of the algorithm. Our algorithm maintains a list of critical constraints called CV (constraint vector) shown in Line 2. CV is initialized to empty set.

Beginning from an initial clock period ϕ, iterations in iRetILP achieves the optimal clock period ϕ^* using retiming transformations or provide a proof that clock period ϕ is optimal. Each iteration of our algorithm generates a graph with retiming transformation G_r and identifies the critical clock period generating edge pair from G_r as shown in Line 11. The constraint is added in CV in Line 12. Due to the incremental nature of iRetILP it always maintains an upper bound on the optimal feasible clock period. Line 13 shows the period update step where ϕ^* is updated if the clock period ϕ improves the upper bound of the optimal period. Line 15 generates a ILP formulation to generate a critical constraint which is not in CV. Now if ϕ^* is updated then Line 15 generates an integer linear program to prove if the new optimal clock period lower bound is feasible. If ϕ^* is not updated then the integer linear program proves that the present clock period ϕ^* is still feasible with the new critical constraint. Our integer linear program is feasible if an assignment of retiming labels within the bounds is obtained at Line 21.

Next we describe the ILP formulation generation shown in Line 15 in more details. Let r be the retiming transformation associated with the ILP formulation. We want to generate an ILP where the critical constraints from vector CV do not appear as the clock period of the graph G_r. Each entry of CV has a set of edges $e = (\hat{u}, u)$, $\hat{e} = (v, \hat{v})$ and an integer constant. Let's consider one such entry c from CV . c is 2-tuple with an ordered pair of edges and an integer. Let the ordered pair of edges be (e,$\hat{e}$). e and $\hat{e}$ are obtained from the procedures get-tail-edge() and get-head-edge() respectively. Let's call the retiming transformation which generated constraint c as r_c. We have $w_{r_c}(e)$ and $w_{r_c}(\hat{e})$ respectively 1 (Based on our timing analyis routine and our graph extension). For the retiming transformation r, the critical path $e \rightsquigarrow \hat{e}$ can be transformed to the following formal condition

$$w_r(\hat{u}, u) = 1 \wedge w_r(v, \hat{v}) = 1 \Rightarrow W_r(u, v) = 0$$

The following theorem gives the condition for formation of G_r such that the critical path $e \rightsquigarrow \hat{e}$ does not exist in G_r.

Theorem 5 (G_r formation): The necessary and sufficient conditions for G_r formation is given by following three conditions on edge pair $(e,\,\hat{e})$

$$w_r(\hat{u},u)=1 \wedge w_{r_c}(v,\hat{v})=1 \Rightarrow W_r(u,v)\geq 1 \quad (7)$$
$$w_r(\hat{u},u)=1 \wedge w_r(v,\hat{v})=0 \Rightarrow W_r(u,v)\geq 0 \quad (8)$$
$$w_r(\hat{u},u)=0 \wedge w_r(v,\hat{v})=1 \Rightarrow W_r(u,v)\geq 0 \quad (9)$$

for every edge $e\in I$, we have

$$0\leq w_r(e)\leq 1$$

for every edge $u\ e\in C$, we have

$$w_r(e)=0$$

Interestingly several constraints can be generated which satisfies Equation 7-9 i.e $w_r(e)\wedge w_r(\hat{e})\leq W_r(u,v)$ (which suffers from non-linearity) but it turns out that the integer linear constraint used in Corollary 1 is the simplest constraint that can be used in our integer linear programming formulation for G_r formation

Lemma 2 (G_r ILP formulation): Formation of G_r used the following integer linear constraints
for every edge $e\in I$, we have

$$0\leq w_r(e)\leq 1$$

for every edge $u\ e\in C$, we have

$$w_r(e)=0$$

for every 2-tuple $\{\ (e,\,\hat{e}),\ w_{uv}\ \}\in CV$

$$w_r(e)+w_r(\hat{e})<=w_{uv}+1 \quad (10)$$

Lemma 2 is used by iRetILP at Line 15 and therefore we need to generate Equation 10 for each constraint c from CV. Note that w_{uv} is equivalent to the equation $W(u,v)+r_c(v)-r_c(u)$. Here we do not know the value of $W(u,v)$ unlike the formulation proposed by Corollary 1 where W matrix is used to generate this value. But in our formulation since we know that when the edge pair (e,$\hat{e}$) was critical, $W_r(u,v)$ must be equal to 0. Based on that we use the following equation to generate $W(u,v)$ for Lemma 2 for each constraint c in CV

$$W(u,v)=r_c(u)-r_c(v)$$

We give the following theorem for the correctness of our algorithm.

Theorem 6 (Critical Constraints): Each iteration of our algorithm finds a entry from D matrix which is one of the timing feasibility constraint for the ILP formulation given by Theorem 3 with c as the optimal clock period ϕ^*.
Invariant used in our iterations are as follows

Theorem 7 (Loop Invariant): Loop invariant of iRetILP is that the integer linear program generated by Lemma 2 is either feasible or there are no critical cycle in the graph G_r.
We terminate the algorithm when loop invariant is negated. The update condition in the loop guarantees that upon termination we get the optimal clock period ϕ^* and the register vector F^*. Note that due to the incremental nature of our algorithm we can run our algorithm with specified number of iterations and terminate the algorithm in Line 22 with ϕ^* as a upper bound on optimal clock period.

Next we present complexity analysis of iRetILP. Let the size of the constraint vector CV be k when the algorithm terminates. At each iteration the size of the constraint vector increases by 1. Let s_i be the size of constraint vector at each iteration i. Complexity of SolveILP becomes a function of s_i. For the purpose of analysis let's consider it to be a linear function $\eta\cdot s_i$. Runtime of our algorithm iRetILP is dominated by the total runtime T of integer linear program solver.

```
                    **Algorithm** iRetILP
**Inputs** G, Register count on each edge $w(e)$ : $\forall e\in E$
**Outputs** The optimal period $\phi^*$ and register vector $F^*$
 1  Initialize $\phi^*=0$, $F^*[e]=0:\forall e\in E$
 2  Initialize $CV=\emptyset$, $R[v]=0:\forall v\in V$
 3  Initialize $F[e]=w(e):\forall e\in E$
 4  Invoke TA using G and F
 5    $\phi^*$ = get-period(), $F^*$ = $F$
 6  **Loop**:
 7    **For** e = (u,v) ∈ E
 8      F[e] = w[e] + R[v] - R[u]
 9    Invoke TA using G and F
10      $\phi$ = get-period()
11      $e$ = get-head-edge(), $\hat{e}$ = get-tail-edge()
12    CV ← CV ∪ ($e$,$\hat{e}$,R[u]-R[v])
13    **If** $\phi<\phi*$
14      $\phi^*=\phi$, $F^*$ = $F$
15    Generate ILP using Lemma 2
16    **If** ILP infeasible:
17      Stop, $\phi^*$ is optimal
18    **Else if** Critical cycle:
19      Stop, $\phi^*$ is optimal
20    **Else**:
21      R ← Solve ILP
22      Stop, if reached iteration bound (ITER)
```

Fig. 3. Algorithm iRetILP

$$T=\sum_{i=0}^{k}\eta\cdot s_i \quad (11)$$

T can be computed as O(k^2) from Equation 11 which is a quadratic function of the number of critical constraints identified by iRetILP. Our experiments show that the number of constraints generated by our algorithm are very less as compared to *Retime-General* which results in the improved efficiency of our algorithm. Also because of that the additional memory consumption of our algorithm is O(k) which is quite less compared to *Retime-General*.

V. Experimental Results

We implement the iRetILP algorithm in C++. To do a fair comparison with [8] we generate random model parameters δ, Δ, α, β and d for all edges of our timing graph. We implemented an incremental version of *Retime-General* which is efficient compared to the original algorithm proposed by [8]. Due to space limitations we are not providing the details but a brief discussion is presented in Section III. For D and W matrix generation we use Floyd-Warshall algorithm [12]. For solving the integer linear program we use CPLEX solver [15] using C++ API level integration. All the codes are compiled by GCC version 3.4.6 and run on a Linux server with dual 2.8GHz Intel Xeon processors and 1GB memory.

We perform experiments with ISCAS89 benchmark suites:We divide ISCAS benchmarks into two sets small and big respectively. We run iRetILP till optimality on the small set of benchmarks. Unfortunately *Retime-General* as reported by [8] can only be run on very small benchmarks till optimality (correlator and s27 from our example which had maximum of 34 vertices). Runtime of ILP becomes quite inefficient when an integer linear program is infeasible. [8] reported that observation in their work and they were unable to

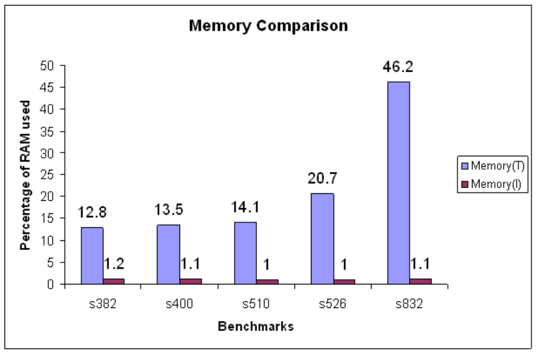

Fig. 4. Memory Comparison

run their algorithm till infeasibility for ISCAS89 benchmarks s298 onwards which had 368 gates. We tried to run *Retime-General* till infeasibility on s298 and the algorithm didn't terminate after even 6 hours. But iRetILP proved the infeasibility in s298 within 91.12 secs. Therefore we let our algorithm iRetILP to run till infeasibility for all the benchmarks from small set and used the optimal clock period to drive Retime-General algorithm. Infeasibility proof of integer linear program is not a viable practical option. Therefore for the fairness of comparison we didn't take into account the runtime of the final infeasibility run for both the algorithms Retime-General and iRetILP for the performance comparisons. Runtime of Retime-General shows the time taken to reduce the upper bound of the clock period to the optimal clock period generated by iRetILP.

Table I reports the experimental results of running *Retime-General* and iRetILP on 16 benchmarks. For 3 of the benchmarks, retiming didn't improve the clock period. We excluded them from our results. $|V|$ and $|E|$ shows the vertices and edges in each benchmark. ϕ_{init} and ϕ^* are the initial and optimized clock periods respectively. For algorithm *Retime-General* M shows the time taken to generate matrices using Floyd-Warshall algorithm while T is the time taken by CPLEX solver [15] to solve the ILP formulation given by Theorem 3. For iRetILP I shows the time taken to get the optimal clock period without the runtime of the last iteration where the G_r ILP formulation given by Lemma 2 was infeasible. I_{inf} shows the total runtime of iRetILP. k refers to the number of critical constraint identified by iRetILP which is required for the complexity analysis of the algorithm. As we mentioned before the number of critical constraints are not of O(E^2) but less than that. The last two columns T/I and $(T+M)/I$ respectively shows the constraint solving speedup and total speedup of iRetILP over *Retime-General*. For benchmarks like s344, s349, s420.1, s526, s838.1 where I_{inf} and I are equal, they terminated by critical cycle criteria from Theorem 7. Note that for some of these benchmarks runtime of *Retime-General* is dominated by M. iRetILP turns out to be more efficient because of removing the bottleneck of matrix generation. On an average our proposed algorithm iRetILP outperforms RetimeGeneral by 100 times even on comparatively small benchmarks.

Since iRetILP generates only critical constraints and do not require the space needed to store D and W, we took some of the benchmarks that generated $>$ 200 critical constraint to do a peak memory analysis. Results are shown in Figure 4. Memory consumption of iRetILP and *Retime-General* are shown respectively by Memory(I) and Memory(T). On s832 iRetILP took $>$ 40 times less memory than *Retime-General*.

For our next set of experiments we run iRetILP in incremental mode on the big set of ISCAS89 benchmarks and use Line 22 for termination. Table II shows the detailed result of our incremental runs. ϕ_{init} represent the initial clock period. We run our algorithm for 100 and 200 iterations respectively. Clock period at the end of the iterations are shown by ϕ_{100} and ϕ_{200} respectively. $R200/R100$ represent the ratio of iRetILP runtime. PD shows the period decrease from running 200 iterations as compared to 100 iterations. Following equation computes period decrease

$$\text{PD} = \frac{(\phi_{200} - \phi_{100})}{\phi_{init}} \cdot 100$$

Based on our complexity analysis with a linear dependance of ILP on k, we expected the ratio of iRetILP runtime to be around 4 but for most testcases ILP solver shows a superlinear dependance on k (varying between 2 and 4). It is interesting to note that for 7 benchmarks PD is 0. For these benchmarks iRetILP terminated based on critical cycle condition and generated optimal results without proving the ILP infeasibility. For rest of the benchmarks the decrease in clock period over iterations is not more than 4.5%. Therefore for bigger benchmarks designer can get an optimized upper bound on clock period quite efficiently using iRetILP with an iteration bound of around 100.

VI. Conclusions

In this paper, we presented an efficient algorithm named iRetILP to solve the minimum period retiming algorithm incrementally and optimally. Instead of generating an integer linear program from a dense timing constraint matrix our algorithm iRetILP generates the critical constraints for proving optimality of a reduced clock period incrementally. Experimental results confirmed that our algorithm is orders of magnitude faster and uses much less memory than the best existing approach. Because of the incremental nature of our algorithm, iRetILP can be stopped any time when a designer is satisfied with the clock period bound. Since the computational core of our algorithm is an integer linear program, generating the infeasibility proof should be avoided for practical usage.

References

[1] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[2] G. Even, I. Y. Spillinger, and L. Stok. Retiming Revisited and Reversed. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 15(3):348–357, March 1996.

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

[4] M. C. Papaefthymiou. Asymptotically efficient retiming under setup and hold constraints. In *ICCAD*, 1998.

[5] H. Zhou. Deriving a new efficient algorithm for min-period retiming. In *Asia and South Pacific Design Automation Conference*, Shanghai, China, January 2005.

[6] C. Lin and H. Zhou. An efficient retiming algorithm under setup and hold constraints. In *DAC*, San Francisco, CA, 2006.

[7] T. Soyata, E. Friedman, and J. Mulligan. Incorporating interconnect, register, and clock distribution delays into the retiming process. *IEEE TCAD*, 16:105–120, January 1997.

[8] K. N. Lalgudi and M. C. Papaefthymiou. Retiming edge-triggered circuits under general delay models. *IEEE TCAD*, 16(12):1393–1408, December 1997.

[9] C. Chu, E. F. Y. Young, D. K. Y. Tong, and S. Dechu. Retiming with interconnect and gate delay. In *ICCAD*, pages 221–226, 2003.

[10] A. Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *ICCAD*, pages 607–614, San Jose,CA, November 2003.

[11] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett. First-order incremental block-based statistical timing analysis. In *IEEE TCAD*, 2006. Accepted for publication.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

TABLE I
RESULTS COMPARISON BETWEEN RETIME-GENERAL AND IRETILP.

	statistics				Retime-General [8]		iRetILP			Speedup	
name	$\|V\|$	$\|E\|$	ϕ_{init}	ϕ^*	M(s)	T(s)	I(s)	I_{inf}(s)	k	T/I	$(T+M)/I$
correlator	16	19	512.77	261.56	0.003	0.127	0.281	0.314	12	0.45	0.46
s27	34	42	724.17	626.18	0.018	0.050	0.035	0.036	1	1.42	1.94
s208.1	297	374	1272.89	1057.78	10.321	3.016	0.478	0.059	8	6.31	27.90
s298	368	498	864.42	645.47	19.247	121.425	18.430	91.120	147	6.59	7.63
s344	440	559	2007.72	1527.20	32.790	35.298	6.500	6.500	74	5.43	10.48
s349	445	567	2269.91	1626.52	34.002	37.547	6.897	6.897	84	5.44	10.37
s382	469	622	1093.17	768.17	39.491	1411.990	58.225	194.442	287	24.25	24.93
s386	515	709	1513.80	1457.67	52.299	38.568	4.450	15.765	33	8.67	20.42
s400	492	655	1339.62	828.75	45.452	1290.740	35.183	166.987	223	36.69	37.98
s420.1	621	786	1359.84	1240.78	91.768	15.836	0.432	0.432	3	36.66	249.08
s444	538	714	1459.49	843.54	59.822	3488.260	260.100	2958.470	688	13.41	13.64
s510	656	875	1331.22	1157.91	107.237	452.885	32.183	98.593	194	14.07	17.40
s526	643	900	1133.48	715.73	101.476	1895.950	34.488	34.488	218	54.97	57.92
s526n	644	900	1133.48	708.69	101.688	1540.900	81.800	10951.200	304	18.84	20.08
s832	1076	1576	1127.20	1104.16	472.100	6821	58.120	274.890	210	117.36	125.48
s838.1	1269	1610	1957.05	1824.97	771.289	53.532	0.852	0.852	3	62.83	968.10
Average										25.84	99.61

TABLE II
RESULTS COMPARISON BETWEEN IRETILP INCREMENTAL RUNS.

	statistics			ITER=100		ITER=200			
name	$\|V\|$	$\|E\|$	ϕ_{init}	ϕ_{100}	$R100$(s)	ϕ_{200}	$R200$(s)	R200/R100	PD
s953	1156	1526	1837.62	1604.31	21.86	1575.38	92.06	4.2	1.57
s1196	1554	2047	2610.60	2610.60	0.02	2610.60	0.02	1	0.0
s1238	1565	2111	2547.15	2547.15	0.02	2547.15	0.02	1	0.0
s1423	1840	2351	6360.50	5808.80	6.02	5808.80	6.02	1	0.0
s1488	2050	2802	1900.21	1900.21	0.03	1900.21	0.03	1	0.0
s1494	2050	2814	1987.04	1987.04	30.16	1895.54	99.65	3.3	4.6
s9234	13589	15984	6818.35	4710.79	107.44	4690.79	254.45	2.4	0.29
s9234.1	13606	16018	6387.14	4646.15	109.23	4633.42	261.89	2.4	0.20
s13207.1	19180	22545	6379.63	5691.10	11.93	5691.10	11.93	1	0.0
s15850	23433	27392	9458.89	7159.57	235.94	7159.57	540.82	2.3	0.0
s15850.1	23496	27518	9240.33	7127.76	172.03	6788.94	351.17	2	3.67
s35392	44371	56984	3845.43	3543.24	76.96	3543.24	76.96	1	0.0
s38417	54237	64191	4887.70	3805.09	420.52	3791.65	1035.85	2.5	0.27
s38584	52023	65803	5979.10	5251.22	1628.92	5125.55	4328.48	2.7	2.1
s38584.1	52049	65855	5984.84	5220.33	35.47	5220.30	35.47	1	0.0

[13] M. C. Papaethymiou. Asymptotically efficient retiming under setup and hold constraints. In *ICCAD*, 1998.
[14] N. Maheshwari and S. S. Sapatnekar. Efficient retiming of large circuits. *IEEE TVLSI*, 6(1):74–83, March 1998.
[15] ILOG, Inc. ILOG CPLEX. http://www.ilog.com/products/cplex.
[16] A. Hurst, A. Mishchenko, and R. Brayton. Scalable min-register retiming under timing and initializability constraints. In *DAC*, pages 534–539, 2008.