An Efficient Incremental Algorithm for Min-Area Retiming*

Jia Wang and Hai Zhou
Electrical Engineering and Computer Science
Northwestern University, Evanston, IL 60208, USA

ABSTRACT

As one of the most effective sequential optimization techniques, retiming is a structural transformation that relocates flip-flops in a circuit without changing its functionality. The min-area retiming problem seeks a solution with the minimum flip-flop area (or number) under a given clock period. Even though having polynomial runtime, the best existing algorithms for this problem still need to first construct a dense path graph and then find a min-cost network flow on it, thus incur huge storage and time expenses for large circuits. Recently, provable incremental algorithms have been discovered for min-period retiming, and heuristic incremental algorithms have been proposed for min-area retiming. However, given the complexity of the problem, min-area retiming is still resisting an efficient provable incremental algorithm. In this paper, we fill the gap by presenting an efficient algorithm to solve the min-area retiming problem incrementally and optimally. Contrary to existing approaches, no dense path graph is constructed; only the active timing constraints are dynamically generated in the algorithm. Experimental results show that the total runtime of our algorithm for all the benchmarks is at least 60× faster than the best existing approach.

Categories and Subject Descriptors: J.6 [Computer-Aided Engineering]: Computer-Aided Design
General Terms: Algorithms
Keywords: Retiming

1. INTRODUCTION

Retiming [5] is one of the most powerful sequential transformations that relocates the flip-flops (FFs) in a circuit while preserving its functionality. As relocating the FFs could balance the longest combinational paths and reduce the circuit states, the clock period and the FF area (or number) in a circuit can be reduced through retiming optimizations. As the minimum clock period (min-period) retiming minimizes the clock period, and thus might significantly increase the FF area, the minimum area (min-area) retiming minimizes the FF area under a given clock period, thus could be used to minimize the FF area even under the minimum clock period. Therefore, the min-area retiming problem is more important for sequential circuit design, but of higher complexity [10].

All existing provable approaches to min-area retiming follow the basic ideas of Leiserson and Saxe [5]. Given a circuit represented as a graph of n vertices and m edges, the minimum number of FFs between any two vertices and the maximum delay over the paths of the minimum number of FFs are first computed. Then, besides one constraint for each edge requiring that the FF number is nonnegative, for each pair of the vertices whose computed path delay is larger than the given clock period, i.e. the timing critical path, a constraint is generated requiring that there is at least one FF between them. Minimizing the FF area under those constraints formulates a dual of the min-cost network flow problem. Since each constraint forms an arc in the flow network, the number of arcs in the network is usually $\Theta(n^2)$. Even though polynomially solvable, min-cost network flow computation (see [1]) over a dense graph is usually expensive on large problems.

Shenoy et al. [10] were among the first to consider a practical implementation of the min-area retiming algorithm. They found that the storage requirement to compute the timing critical paths and the number of constraints are the bottleneck and proposed techniques to reduce memory usage and to prune some redundant constraints. *Minaret*, proposed by Maheshwari et al. [8], further pruned redundant constraints to reduce the size of the flow network by exploring the equivalence of retiming and clock skew optimization as proposed in ASTRA [9]. However, even with these pruning techniques, as experimental results indicate, the flow networks could still be very dense compared to the original circuit graphs. Our experiments with Minaret showed that a circuit with more than 180K gates had to formulate and solve a min-cost network flow problem with more than 122M arcs, which used up all the 2GB virtual memory on our machine.

Recently, Zhou [13] proposed an efficient incremental algorithm for min-period retiming which iteratively moves FFs to decrease the clock period while guarantees to find the optimal solution in a short time. To overcome the expenses of existing approaches to min-area retiming, Singh et al. [11] also proposed to incrementally move FFs in the circuit. However, since they only allow moves that are better in cost and feasible in timing, their approach is a heuristic that may end up with a suboptimal solution. An efficient incremental algorithm with provably optimal solution is evasive till now.

In this paper, we fill the gap with an efficient algorithm named *iMinArea* that solves the min-area retiming problem incrementally and optimally. Contrary to existing algorithms, iMinArea directly attacks the FF area minimization problem instead of its dual network flow problem. Starting with the circuit satisfying the clock period constraint, iMinArea will iteratively reduce the number of FFs by moving FFs backward over some gates with fanouts larger than fanins. If a fanout edge currently has no FF or is on a timing critical path requesting at least one FF, such a move may require FF moves over other gates. iMinArea dynamically maintains such relations among the gates as a forest. If there is a cluster of gates whose fanouts are larger than its fanins, the number of FFs can be reduced by moving one FF over the cluster; otherwise, the current solution is optimal. An outstanding feature of iMinArea is that a critical timing constraint is dynamically generated only when it is needed. Maintaining a forest on the gates, it requests only linear storage (that is, $O(n)$) on top of the circuit graph. As can be expected, iMinArea is extremely efficient in handling

*This work is supported in part by NSF under CNS–0613967.

DAC 2008, June 8–13, 2008, Anaheim, California, USA.

large circuits. For the same circuit with more than 180K gates that failed Minaret, our iMinArea algorithm solved the min-area retiming problem with 65MB memory in less than 1 minutes, which is at least 100 times faster than Minaret and uses at most $\frac{1}{30}$th of the memory used by Minaret.

The rest of this paper is organized as follows. The retiming problems are introduced in Section 2. Our algorithmic idea is presented in Section 3. The iMinArea algorithm is proposed in Section 4. After experimental results are given in Section 5, Section 6 concludes the paper.

2. PROBLEM FORMULATION

As in Leiserson and Saxe [5], a synchronous sequential circuit is modeled by a directed graph $G = (V, E)$ whose vertices V represent combinational gates and whose edges E represent signals between vertices. Nonnegative gate delays are given as vertex weights $d : V \to \mathbb{R}^*$ and the nonnegative numbers of FFs on the signals are given as edge weights $w : E \to \mathbb{N}$. A special host vertex, the edges from host to the primary inputs, and the edges from the primary outputs to host, are introduced into the graph to represent interfaces with the external environment. Given such a graph, the *min-area retiming problem* asks for an FF relocation $w' : E \to \mathbb{N}$ such that the total FF area in the circuit is minimum while it works under a given clock period ϕ.

Conventionally, to guarantee that w' is a relocation of w, a retiming is given by a vertex labeling $r : V \to \mathbb{Z}$ representing the number of FFs moved backward over each gate from fanouts to fanins. Given r, the FF number on the edge (u, v) after retiming is $w_r(u, v) = w(u, v)+r(v)-r(u)$. A retiming r is *valid* iff the FF number of every edge is nonnegative,

$$\mathrm{P0}(r): \quad (\forall (u,v) \in E : w_r(u,v) \geq 0).$$

For a circuit to work under a given clock period ϕ, the maximum combinational path delay in the circuit can be at most ϕ. To compute the maximum path delay, we introduce a vertex labeling $t : V \to \mathbb{R}$ to represent the arrival time at the output of each gate. A valid retiming r is *feasible* for ϕ iff the following condition holds for some arrival times t,

$$\mathrm{P1}(r,\phi): \quad \big(\forall (u,v) \in E : (w_r(u,v) > 0) \vee (t(v) \geq d(v) + t(u))\big) \\ \wedge \big(\forall v \in V : d(v) \leq t(v) \leq \phi\big).$$

Note that the host vertex and the edges entering and leaving host should be ignored in the above condition.

The total FF number is $\sum_{e \in E} w_r(e)$. For any vertex $v \in V$, let $\mathrm{FI}(v)$ and $\mathrm{FO}(v)$ be the sets of the fanins and the fanouts of v respectively. To minimize the total FF number is equivalent to maximize the quantity $\sum_{v \in V}(|\mathrm{FO}(v)| - |\mathrm{FI}(v)|)r(v)$. More generally, let $b : V \to \mathbb{R}$ represent the reduction in FF area if one FF is moved from the fanouts of the given vertex to its fanins. The FF area reduction for the retiming r is $\sum_{v \in V} b(v)r(v)$. With these notations, the min-area retiming problem can be formally stated as follows.

Problem 1 (Min-Area Retiming).

$$\textit{Maximize} \sum_{v \in V} b(v)r(v), \ \textit{s.t.}\ \mathrm{P0}(r) \wedge \mathrm{P1}(r, \phi).$$

For ease of presentation, we extend b to any graph $X = (V_X, E_X)$ with $V_X \subseteq V$ and any $I \subseteq V$ by defining $b(X) \triangleq \sum_{v \in V_X} b(v)$, $b(I) \triangleq \sum_{v \in I} b(v)$, and $b(\emptyset) \triangleq 0$. We assume that $b(G) = 0$ without loss of generality and that the min-area retiming problem is bounded.

More complicated retiming problems can be solved in the same formulation of Problem 1. One example is to consider the sharing of the FFs at the fanouts of a gate. As proposed by Leiserson et al. [5], this scenario is handled by including additional constraints in $\mathrm{P0}(r)$ and setting the labeling b accordingly. Let $\mathrm{wmax}(u) = \max_{(u,v) \in E} w(u, v)$ and assume that all the fanout edges of u have the same breadth $\beta(u)$, which is the costs of adding a FF along each edge. For each vertex u where the FFs at the fanouts of u should be shared, a dummy vertex u_m is introduced. For each fanout v of u, the breadth of the edge (u, v) is changed to $\frac{\beta(u)}{|\mathrm{FO}(u)|}$ and one constraint is added to $\mathrm{P0}(r)$ by adding the edge (v, u_m) to G with $w(v, u_m)$=wmax(u)−$w(u,v)$ and the breadth $\frac{\beta(u)}{|\mathrm{FO}(u)|}$.

3. ALGORITHM OVERVIEW

In this section, we present the general idea behind our incremental algorithm without delving into technical details, which will be presented in the next section.

As a motivation and comparison to our algorithm, we first discuss Leiserson and Saxe's approach to the min-area retiming [5]. Two $n \times n$ matrices W and D are first computed to capture the critical timing constraints, and based on them, a dual of the min-cost network flow problem is formulated and solved. For any vertex pair (u, v), $W(u, v)$ is the minimum number of FFs along any path from u to v, and $D(u, v)$ is the maximum delay of the paths from u to v with $W(u, v)$ FFs. If $D(u, v) > \phi$, then there is a timing critical path from u and v and a critical timing constraint requiring at least 1 FF on the path should be generated. The dual of the min-cost network flow problem is formulated to maximize the FF area reduction subject to the nonnegative FF number requirement and all the critical timing constraints. As W and D would usually be much denser than the circuit graph, the flow network would be dense when the given clock period is tight. Despite the many efforts [10, 8] to reduce the storage requirement for computing the critical timing constraints and to prune the redundant constraints, the large number of constraints is still the bottleneck for solving the min-area retiming problems.

To totally avoid the bottleneck, our algorithm does not compute the matrices W or D at all. The feasibility of clock period ϕ is checked by dynamically updating the gate arrival times and comparing them with ϕ, as in [5, 13]. The objective in Problem 1 indicates that, in order to improve a given solution, some vertices with $b > 0$ must have their r increased. However, a vertex may not be independent: if $w_r(u, v) = 0$, increasing $r(u)$ requests increasing $r(v)$ at the same time. It is not hard to maintain such a relation. However, a more involved case happens when the increase of r over a path extends it to be longer than ϕ. Incremental arrival time updating can identify such a situation, and we will keep a relation between the source u and sink v of the violating path. It is revealing to note that we have $D(u, v) > \phi$ and $r(v) + W(u, v) - r(u) = 1$ for such u and v. In other words, *our algorithm dynamically identifies timing arcs in Leiserson and Saxe's flow network and only identifies the currently tight ones that "lie on the road to improvement"*. The relations thus identified on normal circuit edges and on tight timing arcs are called *active constraints*. They will force vertices with $b > 0$ to be bundled with vertices with $b \leq 0$. When there is still a bundle I with $b(I) > 0$, the objective can be improved by increasing r on I; otherwise, the current retiming is already optimal.

The flow of our algorithm is shown in Fig. 1. A feasible retiming r for the clock period ϕ and a set of active constraints A are maintained throughout the algorithm. An initial feasible retiming r on line 1 may be obtained by any efficient fixed period retiming algorithms [5, 13]. We call a vertex set I *closed* under active constraints A if $\forall (u, v) \in A$,

	Algorithmic Idea Incremental Min-Area Retiming
1	Find a feasible retiming r under clock period ϕ.
2	$A \leftarrow \emptyset$.
3	**Loop**:
4	Find a positive vertex set I closed under A.
5	**If** no such I exists: Stop, r is optimal.
6	**Else if** $w_r(u,v) = 0$ for an edge (u,v) leaving I:
7	Add (u,v) to A.
8	**Else if** $t(v) > \phi$ in r_I for some vertex v:
9	Add $(q(v), v)$ to A.
10	**Else**: $r \leftarrow r_I$; update A.

Figure 1: Incremental min-area retiming.

$u \in I \Rightarrow v \in I$. A vertex set I is *positive* if $b(I) > 0$, meaning that the increment of r on I will reduce the FF area. We use r_I to denote the new retiming after the increment. However, such an increase may violate the nonnegative FF number requirement on an edge leaving I – an active constraint is added in this case on line 7. Such an increase may also violate the timing constraint if a path longer than ϕ is created. We use $q(v)$ to record the source of the critical path to v. If $t(v) > \phi$ in r_I, an active constraint $(q(v), v)$ is added on line 9. If a positive I is found that will not generate more active constraints, the FF area can be reduced by increasing r on I, as on line 10. If, with the growth of active constraints, there is no positive I closed under A, then r is claimed optimal.

The above idea seems natural but the difficulty to realize it lies in how to effectively and efficiently maintain the active constraints A. Keeping every identified active constraint in A is not efficient since it might make $|A|$ very large. On the other hand, if not careful, removing some active constraints from A may not lead to algorithm convergence, since it is possible to have active constraints cycling in and out of A. We successfully overcome the difficulty by maintaining A as a *regular forest*, which is a forest with special properties. The details are presented in the next section; we only note here that $|A|$ is at most $n-1$ while the termination of the algorithm is guaranteed. Similar ideas of maintaining constraints as a forest have been used in other works [6, 3]. However, a major contribution by our algorithm is to incrementally handle dynamically generated constraints in a regular forest, which can not be done by any existing algorithm. Therefore, our algorithm is much more efficient when it is expensive to generate all the constraints.

We use an example as illustrated in Fig. 2 to further clarify our idea of incremental min-area retiming. Detailed executing information is listed in the following table.

	A, I	comments
1	$\emptyset$, $\{g, f\}$	$w(f,e) + r(e) - r(f) = 0$.
2	$\{(f,e)\}$, $\{g, f, e\}$	$t(c) = 7 > 6$, $q(c) = g$.
3	$\{(f,e),(g,c)\}$, $\{f, e\}$	$r \leftarrow r_I$.
4	$\{(f,e),(g,c)\}$, $\{f, e\}$	$w(f,c) + r(c) - r(f) = 0$.
5	$\{(f,e),(g,c),(f,c)\}$, $\{f, e, g, c\}$	$w(e,d) + r(d) - r(e) = 0$.
6	$\{(f,e),(g,c),(f,c),(e,d)\}$, $\{f, e, g, c, d\}$	$t(b) = 7 > 6$, $q(b) = f$.
7	$\{(g,c),(f,b)\}$, No positive I	r is optimal.

4. ALGORITHM DESCRIPTION

4.1 Regular Forests

Consider a forest F with vertices V consisting of rooted trees. For any vertex $v \in V$, let T_v be the subtree rooted at v. For any non-root vertex $v \in V$, let p_v be its parent. A labeling $B : V \to \mathbb{R}$ is maintained such that $B(v) = b(T_v)$. For any non-root vertex $v \in V$, a direction is assigned to the edge $\{p_v, v\}$ such that an active constraint can be derived from the edge. A labeling $\mathrm{U}(v)$ is used to maintain the direction: if $\mathrm{U}(v) =$ true, then (v, p_v) is the active constraint; if $\mathrm{U}(v) =$ false, then (p_v, v) is the active constraint. Let $A(F)$

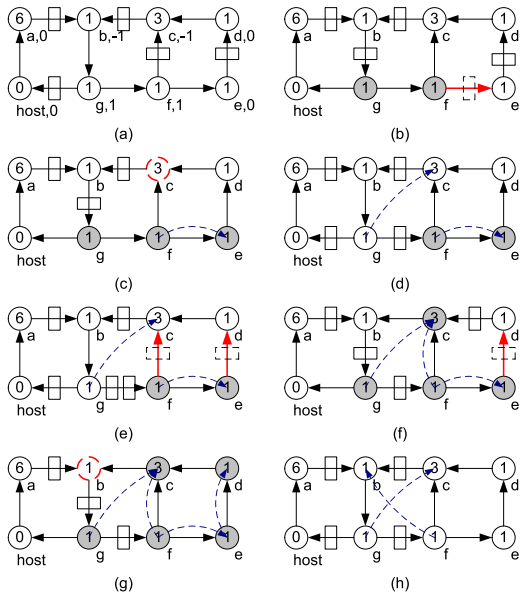

Figure 2: An example of our incremental min-area retiming algorithm. The labeling b is shown in (a) after gate names. Gate delays are inside each gate. The clock period is 6. Dotted arcs are active constraints. Exactly one FF is moved from the fanouts of the gray gates to their fanins.

be the set of the active constraints derived from the edges of F. We define a tree T to be *regular* iff for any vertex v of T that is not the root of T, the following conditions hold, which are illustrated in Fig. 3,

1. if $b(T)>0$, then $\big(\mathrm{U}(v) \wedge (B(v)>0)\big) \vee \big(\neg\mathrm{U}(v) \wedge (B(v) \leq 0)\big)$;
2. if $b(T)=0$, then $\big(\mathrm{U}(v) \wedge (B(v)>0)\big) \vee \big(\neg\mathrm{U}(v) \wedge (B(v)<0)\big)$;
3. if $b(T)<0$, then $\big(\mathrm{U}(v) \wedge (B(v) \geq 0)\big) \vee \big(\neg\mathrm{U}(v) \wedge (B(v)<0)\big)$.

We define a tree to be *almost* regular if the inequalities $B(v) < 0$ and $B(v) > 0$ in the above conditions are substituted with $B(v) \leq 0$ and $B(v) \geq 0$ respectively. We further define the forest to be *regular* if all the trees in the forest are regular.

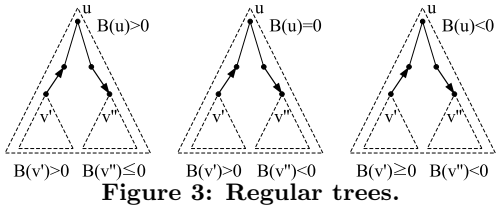

Figure 3: Regular trees.

We call a tree T *positive* (respectively *zero* and *negative*) iff $b(T) > 0$ (respectively $b(T) = 0$ and $b(T) < 0$). Let $P(F)$ (respectively $Z(F)$ and $N(F)$) be the set of all the positive trees (respectively zero trees and negative trees) in F. Let the vertices in $P(F)$ be $V_{P(F)}$. If $P(F) \neq \emptyset$, then $I = V_{P(F)}$ is positive and closed under $A(F)$. Actually, the following lemma states that such I is the one with the maximum $b(I)$.

LEMMA 1. *Let I' be a vertex set that is closed under $A(F)$. Then $b(I') \leq b(I)$ for $I = V_{P(F)}$.*

On the other hand, if $P(F) = \emptyset$, we can claim optimality as in the following lemma,

LEMMA 2. *Suppose that $A(F)$ is a set of the active constraints of a feasible retiming r for the clock period ϕ, i.e., $\forall (u,v) \in A(F)$, either $w_r(u,v) = 0$, or $(D(u,v) > \phi) \wedge (r(v) + W(u,v) - r(u) = 1)$. If $P(F) = \emptyset$, then r is the optimal solution of the min-area retiming problem.*

Subroutine ChangeRoot

Inputs

F: a regular forest. v: a vertex.

Outputs

Updated F where v is the root of a tree.

1 Let $(v_0, v_1, \ldots, v_l)$ with $v_l = v$ be the path from the root of the tree containing v to v in F.
2 **For** $i = 1$ to l:
3 $b_T \leftarrow B(v_{i-1})$. $B(v_{i-1}) \leftarrow B(v_{i-1}) - B(v_i)$.
4 CreateTree(F, v_i).
5 **If** $(b_T > 0) \wedge \mathrm{U}(v_i) \wedge (B(v_{i-1}) > 0)$ or $(b_T < 0) \wedge \neg\mathrm{U}(v_i) \wedge (B(v_{i-1}) < 0)$:
6 Continue.
7 MergeTree(F, v_i, v_{i-1}), $B(v_i) \leftarrow b_T$, $\mathrm{U}(v_{i-1}) \leftarrow \neg\mathrm{U}(v_i)$.

Figure 4: The ChangeRoot subroutine.

We store the forest F in an adjacency list data structure using $O(n)$ storage. We assume that there are two operations that can be completed with $O(n)$ time and space: the first one is CreateTree(F, v), which either removes the edge $\{p_v, v\}$ from the forest if v is not a root, or keep F unchanged if v is a root; the second one is MergeTree(F, u, v), which assumes that v is the root of a tree not containing u and makes u the parent of v. We design the subroutine ChangeRoot(F, v) as shown in Fig. 4 to update the regular forest F in order to make v the root of a tree without introducing additional active constraints into $A(F)$. In each iteration of the **For** loop on line 2, v_{i-1} is the root of the tree containing v, v_i is a child of v_{i-1}, and v is in the subtree rooted at v_i. The subtree rooted at v_i is cut off from the tree rooted at v_{i-1} on line 4. In order to keep the vertices of $P(F)$, $Z(F)$, and $N(F)$ unchanged, we assign v_{i-1} to be a child of v_i on line 7 if necessary. The correctness of the ChangeRoot subroutine is stated in the following lemma.

LEMMA 3. *The invariants of the* **For** *loop on line 2 are that, first, the regular forest F is regular; second, $A(F)$ contains no new active constraint; third, the vertices of $P(F)$, $Z(F)$, and $N(F)$ are not changed. When the subroutine terminates, v is the root of a tree in F.*

Subroutine UpdateForest

Inputs

F: a regular forest. u: a vertex belongs to $V_{P(F)}$. v: a vertex does not belong to $V_{P(F)}$.

Outputs

Updated regular forest F with (u,v) added to $A(F)$.

1 ChangeRoot(F, v).
2 **If** $B(v) = 0$:
3 MergeTree(F, u, v), $\mathrm{U}(v) \leftarrow$ false.
4 **Else**:// must have $B(v) < 0$
5 ChangeRoot(F, u).
6 $b_T \leftarrow B(u) + B(v)$.
7 MergeTree(F, u, v), $B(u) \leftarrow b_T$, $\mathrm{U}(v) \leftarrow$ false.
8 ZeroCut(F, u, b_T).

Figure 5: The UpdateForest subroutine.

Let F_0 be the forest with no edge, i.e., every vertex is a tree in F_0. Then F_0 is regular and $A(F_0) = \emptyset$. Our algorithmic idea in Section 3 suggests to start with the forest F being F_0 and to satisfy $P(F) = \emptyset$ eventually with additional active constraints. Note that $b(P(F)) \geq 0$ always holds and $P(F) = \emptyset$ is equivalent to $b(P(F)) = 0$. Intuitively, either we combine a positive tree with a negative tree to reduce $b(P(F))$, or we combine a positive tree with a zero tree to expand $P(F)$ in order to reduce $b(P(F))$ later. We propose to capture such progress by a potential tuple,

$$\Psi(F) \triangleq (b(P(F)), n - |V_{P(F)}|),$$

with the lexicographic ordering, i.e., for $\Psi(F) = (x, y)$ and $\Psi(F') = (x', y')$, $\Psi(F) \leq \Psi(F')$ iff $x < x'$ or $(x = x') \wedge (y \leq y')$. Assuming that the additional active constraint is (u, v) satisfying $u \in V_{P(F)}$ and $v \notin V_{P(F)}$, we design the UpdateForest subroutine as shown in Fig. 5 that will decrease $\Psi(F)$ by adding (u, v) to $A(F)$ and removing active constraints from $A(F)$ if necessary. Note that such active constraint must exist eventually as we move FFs from the fanouts of $V_{P(F)}$ to their fanins; otherwise the min-area retiming problem is unbounded. In this subroutine, we first simplify the problem by the ChangeRoot subroutine on line 1 in order to make v a root in the regular forest. If v is the root of a zero tree, we assign v to be a child of u on line 3. Otherwise, we further simplify the problem by ChangeRoot on line 5 in order to make u a root. Then we assign v to be a child of u on line 7. Since after line 7, the tree rooted at u will not always be regular but will always be almost regular, we call the ZeroCut subroutine on line 8 to recover F as a regular forest without increasing the potential tuple. The ZeroCut subroutine is shown in Fig. 6, which recursively cuts off the subtrees that do not satisfy the conditions of a regular tree. The correctness of the ZeroCut and the UpdateForest subroutines are stated in the following lemmas.

LEMMA 4. *Assume that every tree in F is regular expect T which is almost regular. Let u be the root of T. Then after we apply ZeroCut$(F, u, b(T))$, F becomes regular and $b(P(F))$ remains unchanged.*

LEMMA 5. *Assume that F is a regular forest, $u \in V_{P(F)}$, and $v \notin V_{P(F)}$. Then after we apply UpdateForest(F, u, v), F remains a regular forest, $\Psi(F)$ is strictly decreased, and (u, v) is the only active constraint added to $A(F)$.*

Subroutine ZeroCut

Inputs

F :a forest.
u :a vertex in an almost regular tree T.
b_T:equals to $b(T)$.

Outputs

Updated forest F where the subtree of T rooted at u becomes a set of regular trees.

1 **For** each child v of u:
2 **If** $\neg\mathrm{U}(v) \wedge (b_T \leq 0) \wedge (B(v) = 0)$ or $\mathrm{U}(v) \wedge (b_T \geq 0) \wedge (B(v) = 0)$:
3 CreateTree(F, v).
4 ZeroCut$(F, v, 0)$.
5 **Else**:
6 ZeroCut(F, v, b_T).

Figure 6: The ZeroCut subroutine.

4.2 The iMinArea Algorithm

Combining the algorithmic idea in Section 3 and the regular forest data structure, we design the iMinArea algorithm that solves the min-area retiming problem incrementally and optimally as shown in Fig. 7. The invariants of the loop on line 3 are stated in the following lemma.

LEMMA 6. *At the beginning of each iteration of the loop on line 3, r is a feasible retiming for ϕ, F is a regular forest, $A(F)$ is the set of the active constraints of r.*

The preconditions for line 8 and 9 are established by the following lemma.

LEMMA 7. *If $w_r(u, v) \neq 0$ for any fanout edge (u, v) of I, then r_I is valid. If $t(v) > \phi$ in r_I for some vertex v on line 8, then $q(v) \in V_{P(F)}$ and $v \notin V_{P(F)}$.*

When the optimality is not claimed on line 5, either the FF area of r will be strictly decreased by $b(I) > 0$ for some $I \subset V$ and $\Psi(F)$ remains the same on line 10, or $\Psi(F)$ will be strictly decreased and the FF area of r remains the same on line 7 and 9 according to Lemma 5. Since the problem is bounded, the number of the subsets of V is finite, and the number of the regular forests with vertices V is finite, we can claim the termination of the iMinArea algorithm. Together with Lemma 2 and 6, we have the following theorem,

Algorithm iMinArea

Inputs

ϕ:the clock period.

Outputs

The optimal feasible retiming r for ϕ.

```
1   Initialize a feasible retiming r for φ.
2   Initialize F to be a forest with no edge.
3   Loop:
4     I ← V_{P(F)}.
5     If I = ∅: Stop, r is optimal.
6     Else if w_r(u, v) = 0 for an edge (u, v) leaving I:
7       UpdateForest(F, u, v).
8     Else if t(v) > φ in r_I for some vertex v:
9       UpdateForest(F, q(v), v).
10    Else: r ← r_I.
```

Figure 7: The iMinArea algorithm.

THEOREM 1. *The iMinArea algorithm will terminate and when it terminates, r is an optimal solution of the min-area retiming problem.*

The iMinArea algorithm requires $O(m)$ storage for the circuit graph and $O(n)$ storage for the auxiliary data structures. The time complexity of each iteration of the loop on line 3 is $O(m)$. The number of iterations can be bounded for reasonable practical VLSI circuits as stated in the following theorem,

THEOREM 2. *The space complexity of the iMinArea algorithm is $O(m)$. If we assume that the labeling b is integer-valued, that the FF area in the initial feasible retiming is bounded by $O(m)$, and that $b(P(F_0)) = \sum_{(v \in V) \wedge (b(v)>0)} b(v)$ is bounded by $O(n)$, then the time complexity of the iMinArea algorithm is $O(n^2 m)$.*

Such time complexity is comparable to that of the min-period retiming problem, which is much simpler than the min-area retiming problem – the best theoretical time complexity for min-period retiming is $O(nm \log n)$ [5], while the worse-case runtime for the most efficient practical algorithm is $O(n^2 m)$ [13].

4.3 Implementation Details

Many details of the iMinArea algorithm are not specified. We extend the incremental idea to the implementation of the algorithm. All the techniques introduced in this section will not affect the theoretical complexity but will effectively improve the practical runtime of the algorithm.

First of all, it is not necessary to generate I on line 4 from scratch every time. It can be proved that the UpdateForest subroutine changes $V_{P(F)}$ by either inserting vertices or removing vertices but not both. We modify the UpdateForest subroutine to provide such information so that we can construct I to be $V_{P(F)}$ incrementally. Let the inserted vertices or the removed vertices be Q. They will be used later when the constraints on line 6 and 8 are checked incrementally.

Secondly, it is not efficient to check every fanout edge of I, to compute the labelings t and q in r_I, and to check every vertex every time when the algorithm reaches line 6 and 8. The constraints should be checked incrementally, i.e. the constraints that are known to hold should be excluded from being checked, and the labelings t and q should be updated incrementally. We maintain two vertex queues J and K for such purpose. For any vertex $u \notin J$, if $u \in I$, then for any edge (u, v), either $v \in I$ or $w_r(u, v) > 0$. For any vertex $u \notin K$ and any vertex v in the combinational fanin cone of u (including u) in r_I, $t(v)$ and $q(v)$ are up-to-date, and $t(v) \leq \phi$. On line 6, we repeatedly remove a vertex u from J until a edge (u, v) leaving I satisfying $w_r(u, v) = 0$ is found or J is empty. On line 8, we repeatedly remove a vertex and its combinational fanin cone from K, compute $t(v)$ and $q(v)$ for any vertex v in the cone, until $t(v) > \phi$ for some vertex v or K is empty. The queues J and K are updated incrementally when I is changed. When I is changed by inserting the vertex set Q, it is sufficient to insert every vertex in Q to J and to insert every vertex in the combinational fanout cone of Q in r_I to K. Computing the cone could be time consuming when $|Q|$ is large. In such case we insert every vertex of G to K. When I is changed by removing the vertex set Q, it is sufficient to insert to J the vertices $u \in I$ that fanouts to a vertex $v \in Q$ satisfying $w_r(u, v) = 0$. Identifying such vertices could be inefficient when $|Q|$ is large. In such case we insert every vertex of G to J. For the queue K, we insert every vertex of G to it.

If the sharing of the FFs at the fanouts of gates is considered, we introduce redundant constraints to P0. Let u be any vertex with the dummy vertex u_m and let v be a fanout of u. In P0(r), we should have $w(u, v) + r(v) - r(u) \geq 0$ and $\text{wmax}(u) - w(u, v) + r(u_m) - r(v) \geq 0$. Thus, $\text{wmax}(u) + r(u_m) - r(u) \geq 0$. This redundant constraint is inserted to P0 and is checked first on line 6 after we remove u from the vertex queue J. The effect is that when both (u, v) and (v, u_m) are active constraints, we directly identify (u, u_m) as an active constraint and thus include u and u_m in one regular tree without requiring a detour to v. As $b(u) > 0$ and $b(u) + b(u_m) = 0$ for most u, $b(P(F))$ is reduced more frequently without the necessity to expand $P(F)$ first and the algorithm runs faster.

5. EXPERIMENTAL RESULTS

We implement the iMinArea algorithm in C++. We obtain the code of Minaret [8] for comparison and obtain the code of the incremental min-period retiming algorithm [13] to compute the minimum clock period and the initial feasible retiming in the iMinArea algorithm. All the codes are compiled by GCC version 3.4 and run on a Linux workstation with dual 2.4GHz Intel Xeon processors and 2GB memory.

We perform experiments with three benchmark suites: the first one are the conventional ISCAS89 sequential circuits; the second one are the large circuits (myex1 through myex5) created by the authors of Minaret from combining ISCAS89 circuits; the third one are the ITC'99 sequential circuits [15], among which there are a few even larger circuits. We follow Minaret [8] to assume unit FF area and unit gate delay. Note that this is only for the purpose of comparison and our iMinArea algorithm can handle arbitrary FF areas and gate delays. For each circuit, we first apply Zhou's min-period retiming algorithm [13] to obtain the minimum clock period. Then we use both the iMinArea algorithm and Minaret to minimize the number of FFs, subject to the minimum clock period and considering the sharing of the FFs at the fanouts of gates.

The experimental results of the largest 26 circuits among all the benchmark suites are reported in Table 1. The iMinArea algorithm solved the problem for all the other circuits in less than 0.1 second and thus the results of them are excluded from being reported here. The statistics of the circuits are reported in the columns "$|V|$" and "$|E|$". For each circuit and each algorithm, we report the number of FFs in the columns "# FFs" and the runtime in seconds in the columns "t_{mp}(s)", "t_{mr}(s)", and "t_{ima}(s)" respectively. The number of FFs obtained by Minaret and that obtained by iMinArea are the same as expected, which is significantly less than that obtained by min-period retiming ignoring the FF area. The minimum clock period is reported in the column "ϕ". The size of the flow network is tremendous in Minaret, as indicated by the number of arcs reported in the column "# arcs". The iMinArea algorithm can find

Table 1: Results comparison between Minaret and iMinArea.

	statistics		Zhou's Min-Period [13]			Minaret [8]			iMinArea			$\frac{t_{\mathrm{mr}}}{t_{\mathrm{ima}}}$
name	$\|V\|$	$\|E\|$	# FFs	ϕ	t_{mp}(s)	# FFs	# arcs	t_{mr}(s)	# FFs	# R	t_{ima}(s)	
s13207.1	8005	11295	629	51	0.01	446	38630	1.37	446	6	0.54	2.5
s15850.1	9844	13792	565	63	0.03	525	38318	6.10	525	4	0.96	6.4
s35932	16421	28944	1729	27	0.01	1729	53087	1.67	1729	1	0.58	2.9
s38584.1	19596	33402	1428	48	0.08	1427	97268	13.79	1427	2	2.32	5.9
s38417	21505	31261	1619	32	0.02	1370	1507162	21.52	1370	5	1.57	13.7
myex1	26114	42576	2578	42	0.11	2293	812872	25.76	2293	4	3.72	6.9
myex2	29135	46890	4545	45	0.17	2022	398697	38.91	2022	5	5.74	6.8
myex3	35712	62018	3912	35	0.10	3279	5693689	88.20	3279	2	12.32	7.2
myex4	40994	64527	3831	35	0.11	2803	2635127	83.62	2803	6	4.61	18.2
myex5	50533	77939	6292	47	0.27	3369	3563555	151.43	3369	7	13.38	11.3
b14(opt)	5331	11523	1021	27	0.02	402	896045	7.65	402	3	0.63	12.2
b14-1(opt)	4042	8750	810	24	0.01	423	511627	3.48	423	2	0.64	5.4
b15(opt)	5945	11599	736	35	0.03	segmentation fault			417	2	0.33	-
b15-1(opt)	6514	13303	1006	23	0.01	579	1468197	10.23	579	3	1.45	7.1
b17(opt)	17228	32503	1982	35	0.09	1249	10981826	110.89	1249	4	2.17	51.0
b17-1(opt)	18813	36819	3016	24	0.06	1640	6712930	62.59	1640	4	8.01	7.8
b18(opt)	58141	115232	5319	59	0.65	2800	12189334	514.82	2800	4	33.36	15.4
b18-1(opt)	57985	116401	6295	50	0.80	3055	8875292	331.63	3055	3	55.76	5.9
b19(std)	183558	328866	9537	107	1.80	-	122286738	>7200	5113	4	57.72	>120
b19-1(std)	173823	310840	9428	107	1.72	-	112595516	>7200	5108	4	49.36	>120
b20(opt)	9952	19257	1360	40	0.04	453	179307	18.75	453	2	0.91	20.6
b20-1(opt)	8392	15854	1103	39	0.03	444	108695	12.19	444	2	1.05	11.6
b21(opt)	9861	18148	1146	43	0.04	368	879762	28.78	368	2	0.56	51.2
b21-1(opt)	8007	14961	1066	38	0.03	424	121706	12.04	424	2	0.88	13.6
b22(opt)	14493	28009	1753	39	0.15	702	236192	23.26	702	3	3.13	7.4
b22-1(opt)	12955	25346	1711	42	0.12	707	612839	30.62	707	2	1.90	16.1
total			74417		6.48			>16K	43147		263.6	>60

the optimal feasible retiming after only a few incremental changes according to the column "# R", which shows the number of the feasible retimings found during optimization. The speed-up of the iMinArea algorithm in comparison to Minaret is reported in the column "$\frac{t_{\mathrm{ima}}}{t_{\mathrm{mr}}}$". Our iMinArea algorithm is much more efficient than Minaret with a speed-up of up to more than 100× and more than 60× in total for all the circuits. We do not report the detailed memory usage but mention that while Minaret used up all the 2GB virtual memory on our machine for the circuits "b19(std)" and "b19-1(std)", our iMinArea algorithm required at most 65MB memory as for the circuit "b19(std)".

We are also curious about iMinArea's performance for a degenerated special case: the *unconstrained* min-area retiming problem without clock period constraints. This problem is conventionally solved as a dual min-cost network flow problem on the circuit graph. We simply turn off line 1, 8, and 9 in iMinArea. Experimental results show that our algorithm is 4× faster than the conventional approach using the min-cost network flow solver CS2 version 4.3 [16]. We also compare our algorithm to Hurst et al. [4], which is an efficient algorithm specially designed only for the problem with unit FF area and implemented in ABC [14]. Surprisingly, even though iMinArea is designed for a much more general problem, it is only 3× slower than Hurst et al. on the special problem.

6. CONCLUSIONS

In this paper, we presented an efficient algorithm named iMinArea to solve the min-area retiming algorithm incrementally and optimally. Instead of attacking the problem by formulating and solving a min-cost network flow problem in a dense flow network, our iMinArea algorithm generates the critical timing constraints dynamically, maintains active constraints in a forest, and retimes a circuit incrementally. Experimental results confirmed that our algorithm is significantly faster and uses much less memory in comparison to the best existing approach.

Even though only backward retiming is discussed in iMinArea, forward retiming can be symmetrically handled and mixed with backward retiming. Therefore, initial state can be easily enforced in the retimed circuits [12, 2]. Moreover, similar to [7] for min-period retiming, if the hold conditions need to be satisfied, iMinArea can be extended to solve the problem optimally.

We should also note that, since it is incremental, iMinArea can be stopped any time when a designer is satisfied (with the area) or impatient (with the runtime). However, we have not (yet) found it necessary to do so.

7. REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Application*. Prentice Hall, 1993.

[2] G. Even, I. Y. Spillinger, L. Stok. Retiming revisited and reversed. *IEEE TCAD*, 15(3):348–357, Mar. 1996.

[3] D. S. Hochbaum. A new–old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks*, 37(4):171–193, Jul. 2001.

[4] A. Hurst, A. Mishchenko, and R. Brayton. Fast minimum-register retiming via binary maximum-flow. In *FMCAD*, pages 181–187, 2007.

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

[6] H. Lerchs and I. F. Grossmann. Optimum design of open-pit mines. *Trans C.I.M.*, 68:17–24, 1965.

[7] C. Lin and H. Zhou. An efficient retiming algorithm under setup and hold constraints. In *DAC*, pages 945–950, 2006.

[8] N. Maheshwari and S. S. Sapatnekar. Efficient retiming of large circuits. *IEEE TVLSI*, 6(1):74–83, Mar. 1998.

[9] S. S. Sapatnekar and R. B. Deokar. Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. *IEEE TCAD*, 15(10):1237–1248, Oct. 1996.

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

[11] D. P. Singh, V. Manohararajah, and S. D. Brown. Incremental retiming for FPGA physical synthesis. In *DAC*, pages 433–438, 2005.

[12] H. J. Touati, R. K. Brayton. Computing the initial states of retimed circuits. *IEEE TCAD*, 12(1):157–162, Jan. 1993.

[13] H. Zhou. Deriving a new efficient algorithm for min-period retiming. In *ASPDAC*, pages 990–993, 2005.

[14] Berkeley Logic Synthesis and Verification Group. ABC–a system for sequential synthesis and verification. `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[15] CAD Group at Politecnico di Torino. ITC'99 benchmarks (2nd release). `http://www.cad.polito.it/tools/itc99.html`.

[16] CS2 version 4.3. Andrew Goldberg's network optimization library. `http://www.avglab.com/andrew/soft.html`.