Efficient Implementation of Retiming

Narendra Shenoy Richard Rudell

Synopsys Inc., 700 E. Middlefield Road,
Mountain View CA 94043

Abstract

Retiming is a technique for optimizing sequential circuits. It repositions the registers in a circuit leaving the combinational cells untouched. The objective of retiming is to find a circuit with the minimum number of registers for a specified clock period. More than ten years have elapsed since Leiserson and Saxe first presented a theoretical formulation to solve this problem for single-clock edge-triggered sequential circuits. Their proposed algorithms have polynomial complexity; however naive implementations of these algorithms exhibit $O(n^3)$ time complexity and $O(n^2)$ space complexity when applied to digital circuits with n combinational cells. This renders retiming ineffective for circuits with more than 500 combinational cells. This paper addresses the implementation issues required to exploit the sparsity of circuit graphs to allow min-period retiming and constrained min-area retiming to be applied to circuits with as many as 10,000 combinational cells. We believe this is the first paper to address these issues and the first to report retiming results for large circuits.

1 Introduction

Retiming is a sequential logic optimization technique. Leiserson and Saxe provided the first formulation and theoretical solution to this problem in 1983 [4] although their later paper [5] has the most complete overview of this work. Retiming uses the flexibility provided by repositioning memory elements to optimize a circuit to optimize one of several objective functions:

1. **min-period**: minimize the clock period of the circuit
2. **min-area**: minimize the number of registers in the circuit
3. **constrained min-area**: minimize the number of registers in the circuit subject to a maximum constraint on the clock period, or indicate failure if the target period cannot be achieved.

As a means of motivating and introducing the concept of retiming, we present a simple example in Figure 1. Assume that each gate has the delay shown inside it. The solid rectangles represent edge-triggered registers. A single clock is used to drive the clock pins of registers. The best clock period for such a circuit

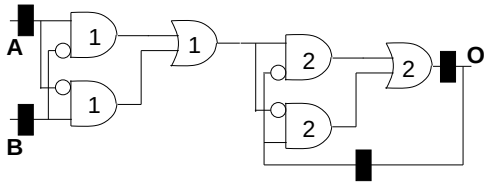

Figure 1: A simple circuit

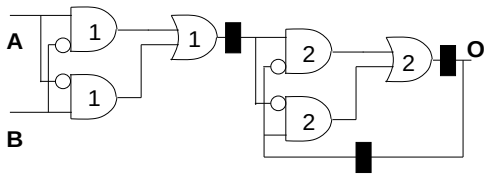

Figure 2: Retiming for minimum registers

(neglecting clock skew and set-up time of registers) is given by the maximum delay of a path consisting of gates. The clock period for the circuit shown in Figure 1 is 6 units. An equivalent circuit with 3 registers and a clock period of 4 units can be obtained by repositioning registers as shown in Figure 2. This circuit has the minimum number of registers. On the other hand, the minimum period achievable by moving registers is 2 units at a cost of 4 registers as shown in Figure 3. Thus a simple re-configuration of memory elements yields designs with differing area costs (number of registers) and performance (clock period). It is this trade-off that we are interested in investigating.

For digital circuit design, the only interesting objective function is constrained minimum area retiming. However, the minimum period retiming problem remains important as a step in solving the constrained min-area problem. This is because the min-period problem is computationally less intensive and it provides a lower-bound for the best delay achievable by the constrained min-area problem. For these reasons, we address both the min-period and constrained min-area optimization problems in this paper.

2 Previous Work

For the case of circuits with edge-triggered memory elements (registers) clocked by the same signal, solutions to all three problems are described by Leiserson and Saxe [5]. Without taking anything away from their significant contribution, we mention sim-

© 1994 ACM 0-89791-690-5/94/0011/0226 $3.50

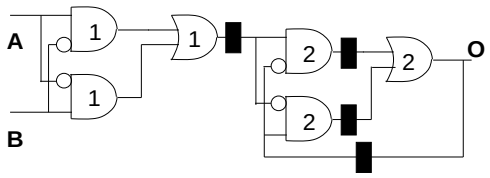

Figure 3: Retiming for minimum period

ply that the implementation details necessary for large sparse circuits were not reported as part of their work. Ishii, Leiserson, and Papaefthymiou [2], extend the concepts to handle level-sensitive memory elements. Lockyear and Ebeling [6] present an alternative approach to retiming circuits with level-sensitive memory elements. However, both of these papers address the theoretical issues involved and not implementation or efficiency details. Papaefthymiou and Randall report experimental results for level-sensitive memory elements [8], but the largest circuit they handle has 379 gates. Munzner and Hemme [7] propose a heuristic algorithm for constrained min-area retiming to convert a combinational circuit into a pipeline. Even though retiming can be used for the same effect, they justify the use of a heuristic algorithm by stating that the retiming algorithms cannot handle circuits with more than 400 gates.

Although theoretical solutions to edge-triggered flip-flop retiming and related problems have been presented in the literature, very few papers have reported experimental results using retiming. To the best of our knowledge, experimental results for constrained min-area retiming problem have not been reported. The only reported results for min-period retiming we have found are for small circuits. We believe the primary reason for this is that, although the algorithms are polynomial in the circuit size, naive implementations suffer the worst-case ($O(n^3)$ time and $O(n^2)$ space) for all circuits.

3 Definitions

A sequential circuit is an interconnection of logic gates and memory elements. A sequential circuit can be represented by a directed graph $G(V, E)$, where each vertex v corresponds to a gate v. Each directed edge e_{uv} represents a flow of signal from the output of gate u at its source to the input of gate v at its sink. Each edge has a weight $w(e_{uv})$ which indicates the number of registers that the signal at the output of gate u must propagate through before it is available at the input of gate v. Each vertex v has a constant delay $d(v)$. If there is an edge from vertex u to vertex v, u is called a fanin of v and v is called a fanout of u. The set of fanouts (fanins) of u is denoted by $FO(u)$ ($FI(u)$). A special vertex called the host vertex is introduced in the graph with edges directed from the host vertex to all vertices that represent primary inputs and edges directed from all vertices representing primary outputs to the host vertex.

A retiming is a labelling of the vertices $r : V \rightarrow Z$ where Z is the set of integers. The weight of an edge e_{uv}, after retiming is denoted by $w_r(e_{uv})$ and is given by

$$w_r(e_{uv}) = r(v) + w(e_{uv}) - r(u) \tag{1}$$

The retiming label $r(v)$, for a vertex v, represents the number of registers moved from its output towards its inputs. A path p is defined as a sequence of alternating vertices and edges, such that each successive vertex is a fanout of the previous vertex and the intermediate edge is directed from the former to the later. A path can start and end at vertices only. The existence of a path from vertex u to vertex v is represented as $u \rightsquigarrow v$. The weight of a path $w(p)$ is the sum of the edge weights for the path. The delay of a path $d(p)$ is the sum of the delays of the vertices on the path. A 0-weight path p, is a path with $w(p) = 0$. The clock period c is determined by the following equation:

$$c = \max_{p|w(p)=0} \{d(p)\} \tag{2}$$

We briefly summarize the results obtained by Leiserson and Saxe [5]. An important concept to the retiming algorithms is the definition of the W matrix and the D matrix. They are defined as:

$$W(u, v) = \min_{p:u \rightsquigarrow v} \{w(p)\}, \tag{3}$$

$$D(u, v) = \max_{p:u \rightsquigarrow v \text{ and } w(p)=W(u,v)} \{d(p)\}. \tag{4}$$

The matrices are defined for all pairs of vertices (u, v) such that v is reachable from u by a sequence of edges and the path does not include the host vertex. $W(u, v)$ determines the minimum latency, in clock cycles, for data flowing from u to v and $D(u, v)$ gives the maximum delay from u to v for the minimum latency.

3.1 Minimum period retiming

The objective is to obtain a circuit with the minimum clock period without any consideration to the area penalty due to an increase in the number of registers. The retiming constraints for a target clock cycle c translate into 2 sets of inequalities:

1. Non-negativity of edge-weights after retiming requires $w_r(e_{uv}) \geq 0$, $e_{uv} \in E$, *i.e.*

$$r(v) - r(u) \geq -w(e_{uv}), \quad \forall e_{uv} \in E. \tag{5}$$

2. Correct clocking at a clock period c requires that all paths $u \rightsquigarrow v$ with $D(u, v) > c$, after retiming have at least one register on it, *i.e.*

$$r(v) - r(u) \geq -W(u, v) + 1, \quad \forall u \rightsquigarrow v, D(u, v) > c. \tag{6}$$

The sets of constraints from Equations 5 and 6 can be solved using the Bellman-Ford relaxation technique developed for the "shortest path on a graph" problem [3]. Leiserson and Saxe introduce 3 algorithms to solve the min-period retiming problem; we describe the most efficient one known as the relaxation method.

Let $\Delta(v)$ denote the largest delay seen along any combinational path that terminates at the output of v.

$$\Delta(v) = d(v) + \max_{u \in FI(v) \text{ and } w(e_{uv})=0} \{\Delta(u)\}. \quad (7)$$

It can be shown that the clock period c is given by the expression

$$c = \max_{v \in V} \{\Delta(v)\}. \quad (8)$$

The relaxation algorithm has the following $O(|V||E|)$ subroutine which determines if a retiming exists for a given clock period c. We refer the reader to [9] for a proof of correctness of this algorithm.

```
For each v ∈ V set r(v) = 0.
For |V| − 1 times {
   Compute retimed edge weights (Equation 1).
   Compute Δ(v), for all v ∈ V (Equation 7).
   For all v ∈ V, such that Δ(v) > c, do r(v)++.
}
Compute retimed edge weights (Equation 1).
If max_{v∈V}{Δ(v)} > c, then no feasible retiming;
else the current r yields a legal retiming.
```

3.2 Constrained Min-Area Retiming

Under the assumption that all registers have the same area, the min-area retiming problem reduces to seeking a solution with the minimum number of registers. Constraints for retiming to be valid are the same as in Equations 5 and 6. The formulation for constrained min-area retiming is:

$$\begin{array}{ll} \min \sum_{v \in V} (|FI(v)| - |FO(v)|) r(v) & \\ r(v) - r(u) \geq -w(e_{uv}) & \forall e_{uv} \in E \\ r(v) - r(u) \geq -W(u,v) + 1 & \forall D(u,v) > c \end{array}$$

These are linear constraints and the objective function is also a linear function of the retiming variables, so linear programming techniques can be used to solve this problem. Leiserson *et al.* [5] indicate that the dual of this problem is an instance of minimum cost circulation on a graph for which efficient algorithms exist. They also indicate that an initial feasible solution can be obtained directly from the problem.

We will not review the construction of the minimum cost circulation problem (details may be found in [9]), except to note that the retiming graph is augmented with dummy vertices, dummy edges, and capacity edges to transform it to the graph on which the minimum cost circulation is solved. The edges in this graph that originate due to the edge weights are termed "circuit" edges as there is one edge for every edge in the original circuit. The edges in this graph that come from the clock period constraints are called "period" edges. Note that the number of period edges can be very large and destroy the sparsity of the graph; we will deal with this issue in the next section.

4 Implementation issues

Our goal is to handle circuits with up to 10,000 combinational cells. However, we expect circuit graphs to be sparse, *i.e.* $|E| = k|V|$ for small k. For min-period retiming, the bottleneck is the requirement that we iterate $O(|V|)$ times to prove that there is no feasible retiming. For constrained min-area retiming, the bottleneck is that even when the graphs are sparse, the W and D matrices are dense. Further, the number of clock period edges which are implied by the retiming equations indicate that the retiming graph augmented with clock period edges will be dense. We demonstrate in this section how to solve each of these problems.

4.1 Minimum period retiming

Let us focus on the relaxation algorithm to determine if c is a feasible clock period. It is empirically observed that if c is feasible then the retiming labels converge rapidly before completing $|V| - 1$ iterations. On the other hand, one cannot determine that a clock period c is infeasible until all $|V| - 1$ iterations have been exhausted and the retiming labels have failed to converge. Thus any hope of speeding up min-period retiming must focus on detecting if a clock period is infeasible before completing the requisite $|V| - 1$ iterations if possible. This is the principal motivation for this section.

The Bellman-Ford algorithm solves the following problem. Given a directed graph $G(V, E)$ with arbitrary edge-weights $f : E \rightarrow R$ (R is the set of reals), and vertices with an initial distance marked on them, find the shortest distance (measured by sum of edge weights) to every vertex that respects the initial distance marking. The algorithm can be described as follows (where $r(v)$ now denotes the distance marking to a vertex v):

```
For each v ∈ V, r(v) = { known original distance,
                       { +∞ otherwise.
Loop |V| − 1 times {
   For each edge e_uv {
      r(v) = min_{u∈FI(v)}(r(v), r(u) + f(e_uv))
   }
}
For each edge e_uv {
   If r(v) < r(u) + f(e_uv) then FAIL (negative cycle)
}
```

The graph is permitted to have negative edge weights and hence can have negative cycles. The presence of a negative cycle makes the shortest distance to any vertex on that cycle undefined. In the presence of a negative cycle, Bellman-Ford must report failure to converge.

We can abort the iteration at any point if we discover a negative cycle in the graph. Let us call such a negative cycle a certificate of infeasibility. We present a technique inspired by Szymanski who used a similar approach to compute lower bounds for clock periods [10]. The predecessor heuristic maintains a predecessor vertex pointer denoted by `pred()` with each

vertex. The Bellman-Ford algorithm starts with all `pred()` pointers set to be empty. Every time vertex v has its distance decreased, the fanin node that caused the change is stored as `pred(`v`)`; *i.e.*, in the Bellman-Ford algorithm, if during the relaxation of edge e_{uv}, we discover that $r(v) > r(u) + f(e_{uv})$, then we set `pred(`v`) = `u. Thus at every instant of the iteration, we have a sub-graph of the original graph maintained by the predecessor edges with $|V|$ edges and $|V|$ vertices.

Each vertex v has a predecessor graph associated with it, defined by repeated traversing of the `pred()` pointers, starting at v and ending when either the predecessor is empty or a cycle is found. At every iteration the predecessor graphs of all vertices are examined to see if a negative cycle exists. If v is marked, its predecessor graph has been examined during an earlier traversal. A cycle is detected by checking if a vertex has already been visited during the walk started from the current v. The traversal is stopped whenever a cycle is detected, a marked vertex is visited, or the end of the predecessor chain has been reached. The complexity of traversal algorithm is $O(|V|)$ and is dominated by the $O(|E|)$ relaxation of the edges. The traversal mechanism is outlined below:

```
For each v ∈ V, mark(v) = 0
For each v ∈ V, cycle(v) = 0
cycle_count = 0
For each v ∈ V {
   if (!mark(v)) {
      cycle_count++
      u = v
      while(u != NIL && !mark(u)) {
         if (cycle(u) == cycle_count) declare
                  cycle exists
         mark(u) = 1
         cycle(u) = cycle_count
         u = pred(u)
      }
   }
}
```

Let us now extend this analogy to the min-period retiming problem. When $\Delta(v)$ is computed for each vertex v, we store with v, a vertex u, such that there exists a 0-weight path $u \rightsquigarrow v$ and $\Delta(v) = d(p)$. If $\Delta(v) > c$, then we set `pred(`v`) = `u. Consider a cycle in the predecessor sub-graph that includes vertices $u_1, \ldots, u_k = u_0$, *i.e.* `pred(`u_i`) = `u_{i-1}, $i = 1, \ldots, k$. Let $p_{i-1,i}$, denote the path $u_{i-1} \rightsquigarrow u_i$ which is used in the computation of $\Delta(u_i)$. During the iterations, retiming labels increase only by 1. Recall that before the labels are updated, $w(p_{i-1,i}) = 0$. After update,

$$\begin{aligned} w_r(p_{i-1,i}) &= r(u_i) - r(u_{i-1}) + w(p_{i-1,i}) \\ &= r(u_i) - r(u_{i-1}) \\ &\leq 1. \end{aligned}$$

name	*# gates*	*CPU (in sec.)*		*clock period*	
		original	*new*	*before*	*after*
s1494	386	96.2	2.1	17.4	17.4
s1423	384	101.0	3.9	36.6	31.4
s5378	887	527.6	13.7	11.9	10.4
s9234	1107	159.8	13.8	19.9	12.9
s13207	1854	1973.2	28.8	23.2	21.4
s15850	2240	2856.1	37.4	40.1	22.9
s38584	7882	39025.9	306.2	35.5	34.1

Table 1: Minimum period retiming.

In addition $d(p_{i-1,i}) > c$. Thus for the cycle

$$\begin{aligned} \sum_{i=1,\ldots k} d(p_{i-1,i}) &> kc \\ \sum_{i=1,\ldots k} d(p_{i-1,i}) &> c \sum_{i=1,\ldots,k} w_r(p_{i-1,i}). \end{aligned}$$

The term on the left hand side represents the total delay encountered as a signal traverses the cycle. The term on the right hand side is the total time available for the signal to propagate under the current clock period. Retiming cannot change the number of registers in a cycle of a circuit (see Lemma 1 in [5]). This implies that for a clock period c, this cycle will prevent any feasible retiming. Retiming can only exist for a clock period given by the inequality,

$$c' \geq \frac{\sum_{i=1,\ldots,k} d(p_{i-1,i})}{\sum_{i=1,\ldots,k} w(p_{i-1,i})}. \tag{9}$$

The implementation of this technique results in dramatic speed-up in execution time. Not only can infeasibility be detected early, but the cycle that caused infeasibility provides a new lower bound for the clock period. This can be used to bias the binary search effectively. The memory overhead consists of a pointer per vertex and an extra field used for detecting the cycle.

4.1.1 Experimental results

All experiments are run on a Solbourne Series 5e. We select some circuits from the ISCAS89 suite chosen so that they reflect the variation in size of this suite.

We compare an implementation of the retiming algorithm without predecessor pointers to an implementation that uses it in Table 1. The two implementations are identical except for the part that traverses the predecessor pointers. We see a substantial reduction in execution time for the predecessor heuristic. Using the cycle obtained as a certificate of infeasibility to update the lower bound on the clock period is useful, as the bias eliminates feasibility checks at clock periods less than the bound and are guaranteed to be infeasible.

4.2 Minimum area retiming

Minimum area retiming poses 2 major hurdles;

1. computing the W and D matrices, and
2. implementing minimum cost circulation.

4.2.1 Computing W and D matrices

We shall not describe the method proposed by Leiserson *et al.* [5] to compute the W and D matrices, except to say that an algorithm with $O(|V|^3)$ time and $O(|V|^2)$ memory is proposed. The W and D matrices are required to add the clock period edges which in turn are required to solve the minimum cost circulation problem using cost scaling techniques. Further, all of the clock period edges must be added prior to solving the circulation problem.

The number of clock period edges greatly increases the density of the original graph. However, only a small subset of the period edges implied by Equation 6 are required for the computation; the rest form redundant constraints. The original algorithm has two drawbacks: the $O(|V|^2)$ memory and the inability to prune the clock period edges. We propose an algorithm which has a worse complexity than the original formulation, but whose memory is $O(|V|)$ and is able to generate a smaller set of constraints for sparse circuit graphs.

Equation 6 describes the conditions under which clock period edges need to be added to the retiming graph. In the original formulation of constrained min-area retiming, clock period edges are required between all vertices u and v such that

$$u \rightsquigarrow v \text{ and } D(u,v) > c. \tag{10}$$

To see why a smaller set of clock period edges is sufficient for constrained min-area retiming, note that if

$$r(v) - r(u) \geq -W(u,v) + 1 \tag{11}$$

is true for a sub-path of a path, then it is also true for the entire path. Hence a period edge need only be added to vertex v, reachable from w, such that:

$$D(w,v) > c \text{ and } D(w,u) \leq c \ \forall u \text{ lying on } w \rightsquigarrow FI(v) \tag{12}$$

Consider a vertex w. We are interested in the period edges that have their source as w. To do this, it is necessary to examine only a single row of the W and D matrices (*i.e.*, the row $W(w,.)$ and the row $D(w,.)$). The set of vertices can be partitioned into disjoint sets depending on the value of $W(w,.)$ as shown in Figure 4. The directed edges in Figure 4 represent paths to other vertices from w. The dashed curve represents the set of vertices v that meet the condition of Equation 12. We can ignore any period edges between w and the fanouts of such vertices. Thus only some of the entries of $W(w,.)$ and $D(w,.)$ need to be computed. The elements of $W(w,.)$ can be computed using Djikstra's algorithm since the edge weights $w(e_{uv}) \geq 0$, $\forall e_{uv} \in E$. The computation of $D(w,.)$ is complicated due to 2 facts; the dependence on $W(w,.)$ and the non-monotonicity of the gate delays (akin to finding the longest path in a graph with positive edge weights).

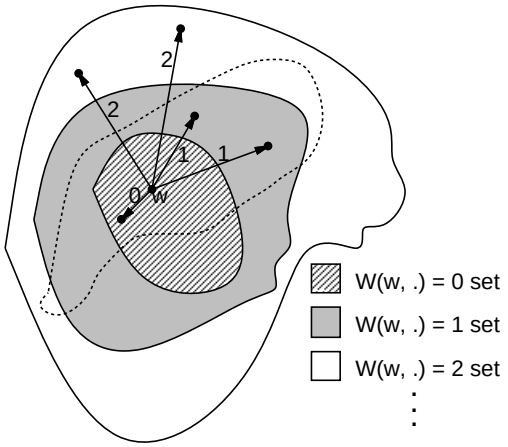

Figure 4: Disjoint partitions of the vertices

We now describe how to generate a single row of the W and D matrices for a single vertex w, and how to find only the set of vertices which satisfy Equation 12. An ordered pair $(w(e_{uv}), -d(v))$ is associated with each edge e_{uv} and is used to compute the shortest distance from w. Comparisons are done in lexicographical order. Thus for a path $w \rightsquigarrow v$, we obtain the distance as an ordered pair denoted by (a_v, b_v). Upon termination of the algorithm this pair is the same as $(W(w,v), D(w,v))$. The algorithm to compute period edges consists of applying the following mix of Dijkstra's algorithm and Bellman-Ford algorithm. The algorithm maintains a heap for each distinct value of a_v (the heap is indexed by this value since there could be several such heaps). Since $w(e_{uv}) \geq 0$, we are guaranteed that the first component of the distance measure cannot decrease for all vertices in the heap with the lowest index. To compute b_v for all v in the smallest indexed heap, the Bellman-Ford algorithm is used.

```
c = target clock period
S_k = the k^th heap
∀w ∈ V {
    w = current vertex
    ∀v ∈ V, a_v = 0 and b_v = +∞
    S_0 = {w}, a_w = d(w) and b_w = 0
    k = current register weight
    do {
        k = min{p|S_p ≠ ∅}
        u = pop_min (S_k)
        if (b_u > c) {
            add period edge from w to u
```

```
            with weight a_u − 1
        } else {
            ∀v ∈ FO(u) {
                if ((a_v, b_v) > (a_u + w(e_uv), b_u + d(v))) {
                    heap_insert(S_{a_u + w(e_uv)}, v)
                }
            }
        }
    } while (∃p | S_p ≠ ∅)
}
```

Dijkstra's algorithm for shortest paths on a graph requires that the distance measure be monotonic. The distance measure (as defined by the lexicographical order) of a vertex v is a function of the edge-weights and the vertex delays. Edge weights are monotonic, since $w(e) \geq 0$, $\forall e \in E$ and we are interested in distances with minimum number of edge-weights. However, the vertex delays are non-monotonic; *i.e.* after popping u using `pop_min()`, we cannot conclude that b_u has attained the value of $D(w,u)$. To handle this, a Bellman-Ford relaxation is carried out for each value of k (the minimum index amongst the set of heaps).

Consider the analysis for a given w. Let the index k increase from 0 to K. Let V_k be the set of vertices for which $W(w,v) = k$. Let E_k be the set of edges with 0 edge weights with either sources or sinks in V_k. Due to the non-monotonicity of vertex delays, we are forced to execute a Bellman-Ford set of relaxations for each k. Hence for a given k, there will be at most $|V_k||E_k|$ heap queries, each of which requires $\log|V_k|$ time; yielding a complexity of $|V_k||E_k|\log|V_k|$. Since we are guaranteed that every cycle has at least one register, distances cannot be increased arbitrarily by traversing cycles. Thus the algorithm requires $O(\sum_{k:V_k \neq \emptyset} |V_k||E_k|\log|V_k|)$. Note that the vertex sets V_k (and edge sets E_k) are disjoint and hence we can bound this by $O(|V||\overline{E}|\log|\overline{V}|)$, where $\overline{V}$ and $\overline{E}$ are the maximum sized sets amongst V_k and E_k (among possible values of k). Since this is repeated for each vertex the worst case bound is $O(|V|^2|E|\log|V|)$ — considerably worse than a Floyd-Warshall ($O(|V|^3)$). There are two benefits to using this algorithm. First the memory overhead is a set of heaps whose total size can be kept to $O(|V|)$ with some book-keeping. Secondly, the execution time rarely displays the behavior predicted by worst case analysis. On sparse graphs, the term $\sum_{k:V_k \neq \emptyset} |V_k|$, rarely reaches its upper bound of $|V|$ because, not all vertices are reachable from w, and the pruning of distance propagation once the delay of a path becomes larger than c effectively restricts the set of vertices examined. This pruning strategy speeds up the convergence of the algorithm.

In practice, the term $|\overline{E}|\log|\overline{V}|$ is much smaller than $|E|\log|V|$. The final proof of this algorithm is in the implementation. For some of the large circuits used in the experiments, it is almost impossible to finish computing the W and D matrices in a reasonable amount of time using a Floyd-Warshall implementation. With the above algorithm, the execution time is comparable to minimum cost circulation computation.

4.2.2 Minimum cost circulation using cost-scaling

A disadvantage of cost-scaling techniques is that the graph cannot change during the computations; consequently all the period edges must be determined before starting the cost-scaling algorithm. Our implementation is based on the generalized cost-scaling framework of Goldberg and Tarjan [1] and has a complexity of $O(|V||E|\log|V|\log(|V|C))$ where C is the largest cost in the graph *i.e.* one more than the number of registers in the circuit. If $|V|$ is 10,000, and we restrict ourselves to 32-bit integer arithmetic, C must be less than $200,000$; since C is the number of registers in the circuit, this is a reasonable assumption. The algorithm operates by maintaining an error from optimality and successively halving it. At the start this error is $|V|C$. However if an initial flow is known, the value of C can be reduced to be the minimum value that has to be added to the cost of each edge so that the graph does not have a negative cost cycle. This fact reduces the value of C by an order of magnitude. An initial flow for the circulation graph can be constructed easily. For most circuits C turns out to be less than 10, effectively making the algorithm independent of C.

As an aside, one has to be careful with the edge capacities. In general these are real numbers and can cause convergence problems. For sake of stability, the edge capacities are scaled to integers using a factor which is a function of the least common multiple of the number of fanouts of vertices in the graph.

4.2.3 Experimental results

The experimental setting for min-area retiming consists of the following steps. A min-period retiming is carried out to determine a bound on the best achievable clock period. The number of registers in the min-period solution is examined. A min-area retiming is performed on this circuit with the clock period set to the best achievable clock period. This yields an area optimal solution without any sacrifice in performance.

The results for constrained min-area retiming are compared to the results for min-period retiming in Table 2. The first set of circuits consists of the ISCAS circuits used for min-period retiming. The last 3 circuits (the second set) are multipliers pipelined by placing three resisters in series at the outputs in each case (see Section 5 for details). As can be seen, the min-period solution can be very far away from the area optimal solution at the same clock period.

5 Tracing area-delay curves

One motivation for constrained min-area retiming algorithms is to examine the area-delay trade-off. This section presents a set of experiments on two multipliers: an 8x8 multiplier with 16 outputs and a 16x16 multiplier with 32 outputs. We choose multipliers because they common circuits found in many data-paths and signal processing designs. Each multiplier has 3 registers in series at each output. The area and delay

name	*# gates*	*# registers*		*CPU for min-area*
		min period	*min-area at min-period*	
s1494	386	7	7	58
s1423	384	81	78	84
s5378	887	296	169	300
s9234	1107	328	241	641
s13207	1854	548	481	2467
s15850	2240	655	529	4668
s38584	7882	1433	1429	139328
8x8	542	179	102	38
16x16	3030	710	183	2459
32x32	≈13k	2763	403	67155

Table 2: Minimum area retiming.

name	*# gates*	*regi-sters*	*clock period*	
			original	min-period
8x8	542	48	27.85	7.53
16x16	3030	96	58.82	15.92

Table 3: Properties of multipliers

characteristics for the initial circuits are summarized in Table 3 (columns 2-4). The clock period is the maximum delay from an input to an output, since the registers are all placed at the outputs. As the latency of each output is 3, we expect that min-period retiming will partition the circuit into 4 regions that have almost the same delay. Thus the value of the clock period at a min-period solution is expected to be roughly a fourth of the original clock period (column 5).

The next experiment concerns the area delay trade-off that is possible by changing the target clock period. The lower bound on the clock period is computed using the min-period retiming algorithm. The original clock period is an upper bound. 4 equi-distant points are picked between the upper and lower bound, as a target clock period for each circuit. The minimum number of registers required for each clock period is computed (in Table 4). An effective area-delay trade-off is seen. The CPU variation with the clock period is also shown in Table 4. This variation can be ascribed to the changing number of edges in the graph, which varies significantly as the clock period is varied. This variation is a response to the different target clock periods. For a given circuit, as the target clock is varied, the number of period edges varies considerably (see Table 5). At clock periods close to the lower bound, the period edges dominate all the other edges in E. The number of period edges steadily increases as the target clock period is lowered and then decreases. This can be explained as follows. As the clock period decreases, the number of paths that need to be constrained increases. This is the general trend. Recall the pruning strategy used in the computation of period edges that ignores any path that extends from

name	*# registers at clock* $= T_{min} + kt_{step}$					
	k=0	*k=1*	*k=2*	*k=3*	*k=4*	*k=5*
8x8	102	69	63	54	52	48
16x16	183	139	126	117	104	96
	CPU time at clock $= T_{min} + kt_{step}$					
8x8	33.8	31.0	31.4	37.9	36	36.1
16x16	2459	4264	2325	2860	3397	2840

Table 4: Variation versus clock period.

name	$\lvert V\rvert$	$\lvert E\rvert/\lvert V\rvert$ *at clock* $= T_{min} + kt_{step}$					
		k=0	*k=1*	*k=2*	*k=3*	*k=4*	*k=5*
8x8	799	8.6	9.7	6.7	4.1	3.3	3.1
16x16	3944	25.5	38.5	26.5	7.3	3.7	3.2

Table 5: Retiming graph density.

a sub-path with delay greater than the clock period. This has little effect at large clock periods, since most reachable vertices have delays less than the clock period. But when the clock period decreases, this strategy results in a decrease of period edges. Note that in all the cases, the graph remains sparse even after the period edges are added.

6 Conclusions

The practicality of constrained min-area retiming (min-area retiming subject to a target clock period) for large circuits has been demonstrated. The issues that need to be addressed for a successful implementation require careful analysis and a good understanding of software techniques and computer algorithms. We hope to have high-lighted some of these issues in this technical discussion.

The predecessor technique to detect infeasibility of a clock period has been demonstrated to be very effective for min-period retiming.

Two key contributions are made for the constrained min-area retiming problem. First, we have shown that finding essential period edges can be done efficiently. Secondly, using the information provided by an initial flow is critical for convergence of the cost-scaling algorithm.

The area-delay trade-off demonstrates a big win for the constrained min-area algorithm over the min-period algorithm, although this is hardly surprising. Although both algorithms have been known for several years, the experimental evidence has been lacking, especially for large circuits. The min-area algorithm enables a designer to control the area-delay trade-off of retiming in a precise manner.

Finally, we would like to dispel the notion that the existence of a polynomial-time algorithm implies that the techniques can be applied to large circuits; we must have near-linear complexity to handle entire circuits in one piece.

Acknowledgments

We thank Albert Wang and Don MacMillen for stimulating several discussions. This work was sponsored by the ARPA Application Specific Electronic Modules (ASEM) Program under the Manufacturing Technology Directorate of Wright Laboratory, Air Force Material Command (ASC).

References

[1] A. Goldberg, E. Tardos, and R. E. Tarjan. Network flow Algorithms. Technical report, Department of Computer Science, 1989.

[2] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou. Optimizing Two-Phase Level-Clocked Circuitry. In *Advanced Research in VLSI*, 1992.

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

[4] C. E. Leiserson and J. B. Saxe. Optimizing Synchronous Systems. In *Journal of VLSI and Computer Systems*, pages 41–67, 1983.

[5] C. E. Leiserson and J. B. Saxe. Retiming Synchronous Circuitry. In *Algorithmica*, 1991. 6(1).

[6] B. Lockyear and C. Ebeling. Optimal Retiming of Multi-Phase Level-Clocked Circuits. In *Advanced Research in VLSI*, 1992.

[7] A. Munzner and G. Hemme. Converting Combinational Circuits into Pipelined Data Path. In *Proceedings of the International Conference on Computer-Aided Design*, 1991.

[8] M. C. Papaefthymiou and K. H. Randall. TIM: A Timing Package for Two-phase Level-clocked Circuitry. In *Proceedings of the Design Automation Conference*. IEEE/ACM, 1993.

[9] J. B. Saxe. *Decomposable Searching Problems and Circuit Optimizations by retiming: Two Studies in General Transfromations of Computational Structures*. PhD thesis, Carnegie-Mellon University, 1985.

[10] T. G. Szymanski. Computing Optimal Clock Schedules. In *Proceedings of the Design Automation Conference*, 1992.