13

Retiming Synchronous Circuitry

by Charles E. Leiserson and James B. Saxe

August 20, 1986

digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center and two other corporate research laboratories are committed to filling that need.

SRC opened its door in 1984. We are still making plans and building foundations for our long-term mission, which is to design, build, and use new digital systems five to ten years before they become commonplace. We aim to advance both the state of knowledge and the state of the art.

SRC will create and use real systems in order to investigate their properties. Interesting systems are too complex to be evaluated purely in the abstract. Our strategy is to build prototypes, use them as daily tools, and feed the experience back into the design of better tools and the development of more relevant theories. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

During the next several years SRC will explore applications of high-performance personal computing, distributed computing, communications, databases, programming environments, system-building tools, design automation, specification technology, and tightly coupled multiprocessors.

SRC will also do work of a more formal and mathematical flavor; some of us will be constructing theories, algorithms, and proving theorems as well as designing systems and writing programs. Some of our work will be in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. We also expect to explore new ground motivated by problems that arise in our systems research.

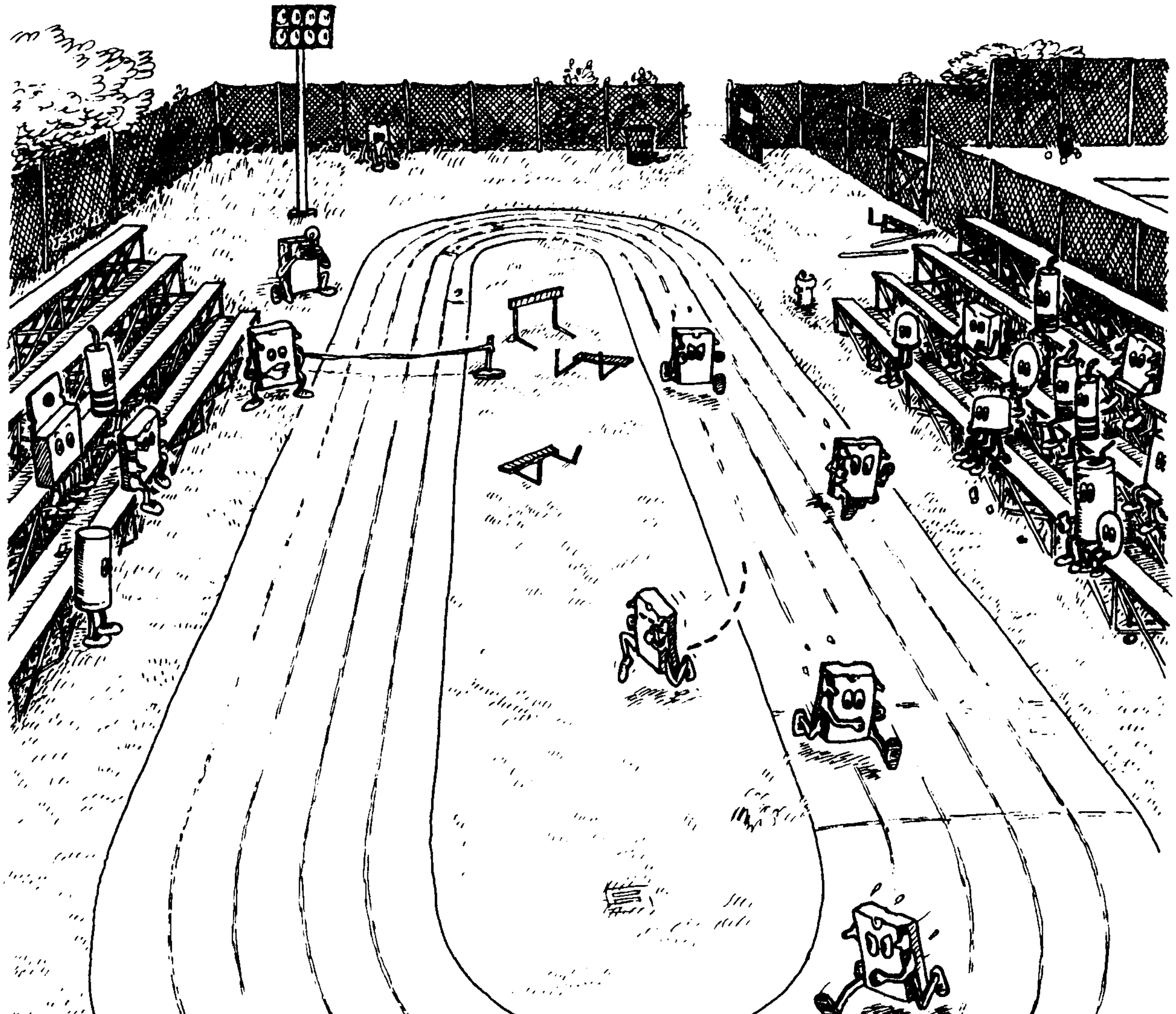
DEC is committed to open research. The Company values the improved understanding that comes with widespread exposure and testing of new ideas within the research community. SRC will therefore freely report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

Retiming Synchronous Circuitry

Charles E. Leiserson and James B. Saxe

August 20, 1986.



Acknowledgments

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-80-C-0622 and by the Office of Naval Research under Contract N00014-76-C-0370. Charles Leiserson was supported in part by an NSF Presidential Young Investigator Award. Most of the results reported here were obtained while James Saxe was in the Computer Science Department at Carnegie-Mellon University. During part of this period, he was supported by an IBM graduate fellowship. Charles Leiserson is at the MIT Laboratory for Computer Science, Cambridge, Massachusetts 02139.

Publication History

An earlier version of this paper appeared under the title "Optimizing synchronous circuitry by retiming" in the proceedings of the *Third Caltech Conference on Very Large Scale Integration*, Randal Bryant, ed., published in 1983 by Computer Science Press, Rockville, MD. That version does not include the efficient algorithm described in Section 5 of this report.

This paper has appeared as MIT Laboratory for Computer Science technical memo MIT/LCS/TM-309 and will appear in *Algorithmica*.

Authors' Abstract

This paper describes a circuit transformation called *retiming* in which registers are added at some points in a circuit and removed from others in such a way that the functional behavior of the circuit as a whole is preserved. We show that retiming can be used to transform a given synchronous circuit into a more efficient circuit under a variety of different cost criteria. We model a circuit as a graph in which the vertex set V is a collection of combinational logic elements and the edge set E is the set of interconnections, each of which may pass through zero or more registers. We give an $O(|V|\,|E|\lg|V|)$ algorithm for determining an equivalent retimed circuit with the smallest possible clock period. We show that the problem of determining an equivalent retimed circuit with minimum state (total number of registers) is polynomial-time solvable. This result yields a polynomial-time optimal solution to the problem of pipelining combinational circuitry with minimum register cost. We also give a characterization of optimal retiming based on an efficiently solvable mixed-integer linear programming problem.

Charles E. Leiserson and James B. Saxe

Capsule Review

This report describes *retiming*, a method of transforming a digital circuit to minimize its clock period or the number of registers required to implement it.

Circuits for which these techniques are applicable consist of combinational logic connected by clocked registers. Unlike many earlier techniques which modify the combinational logic to decrease its size or reduce its propagation time, retiming changes the placement of the registers to distribute the logic delays more uniformly over the circuit. The resulting circuit exhibits the same interface behavior and uses the same functional elements as the original, but it is smaller, faster, or both.

The report is divided into ten sections. The first three introduce the concept of retiming, demonstrate the technique with an example, and present the graph model for circuits used in the balance of the report. The next three sections present several algorithms for performing retiming with the goal of minimizing the clock period of a circuit. The final sections discuss the use of retiming to minimize the amount of state required by a circuit, and the relationship between retimed synchronous circuits and systolic circuits.

Although the report emphasizes the theoretical basis for retiming, a number of the ideas it contains should be of interest to practicing circuit designers.

Chuck Thacker

Contents

1 Introduction

The goal of VLSI design automation is to speed the design of a system without sacrificing the quality of implementation. A common means of achieving this goal is through the use of optimization tools that improve the quality of a quickly designed circuit. In this paper we show how to optimize clocked circuits by relocating registers so as to reduce combinational rippling. Unlike pipelining, this technique, which we call *retiming*, does not increase circuit latency.

In order to illustrate retiming, consider the problem of designing a digital correlator. The correlator takes a stream of bits $x_0, x_1, x_2, \ldots$ as input and compares it with a fixed-length pattern $a_0, a_1, \ldots, a_k$. After receiving each input x_i $(i \geq k)$, the correlator produces as output the number of matches

$$y_i = \sum_{j=0}^{k} \delta\left(x_{i-j}, a_j\right) \ , \tag{1}$$

where δ is the comparison function

$$\delta(x, y) = \begin{cases} 1, & \text{if } x = y; \\ 0, & \text{otherwise.} \end{cases}$$

Figure 1 shows a design of a simple correlator for the case when $k = 3$. Correlator 1 consists of two kinds of functional elements, adders and comparators, whose I/O characteristics are shown in the figure. The boxes between the comparators are registers which act to shift the x_i to the right down the length of the correlator. On each tick of the global clock, each x_i is compared with a character of the pattern, and the adders sum up the number of matches.

This design, though easy to understand, has poor performance. Between ticks of the clock, the partial sums of the matches ripple up the length of the correlator. Suppose, for instance, that each adder has a propagation delay of 7 esec,[1] and each comparator has a propagation delay of 3 esec. Then the clock period must be at least 24 esec—the time for a signal to propagate from the register on the connection labeled A through one comparator and three adders.

[1] Recall that one eptosecond (esec) equals one one-sillionth of a second.

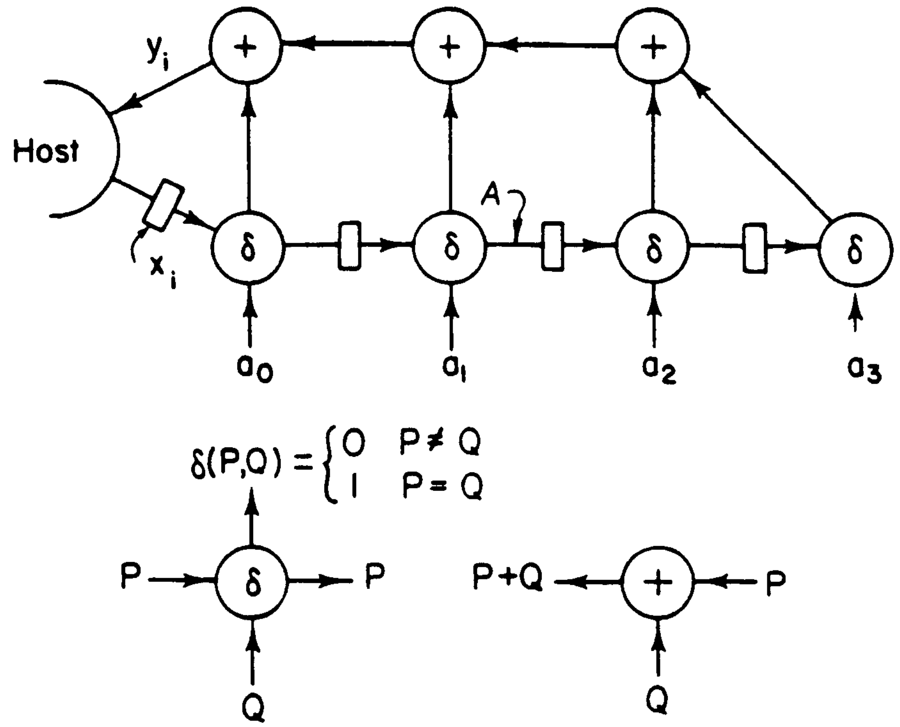


Figure 1. Correlator 1: A simple circuit made of two kinds of functional elements. Each comparator δ has a propagation delay of 3 esec, and each adder + has a propagation delay of 7 esec. A longest path of combinational rippling starts at the register on the connection labeled A, and thus the clock period of the circuit is 24 esec.

A design that gives better performance can be derived by removing the register on connection A from Correlator 1 and inserting a new register on connection B, as shown in Figure 2. To show that these two correlators are indeed functionally equivalent, consider the portion of the circuit surrounded by the dashed box in the figure. It communicates with the rest of the circuit only through connections A and B. When the register on A is removed, all input signals to this portion of the circuit arrive one clock tick earlier, and thus the boxed portion of Correlator 2 performs the same sequence of computations as in Correlator 1, but one clock tick earlier. Since the output from the boxed portion of Correlator 2 is delayed one clock tick by the new register on connection B, the remainder of the circuit sees the same behavior as in Correlator 1. We say that the three functional elements in the boxed portion of Correlator 2 *lead* by one clock tick the corresponding functional elements in Correlator 1. Alternatively, we say that the three elements in Correlator 1 *lag* by one clock tick the corresponding elements in Correlator 2.

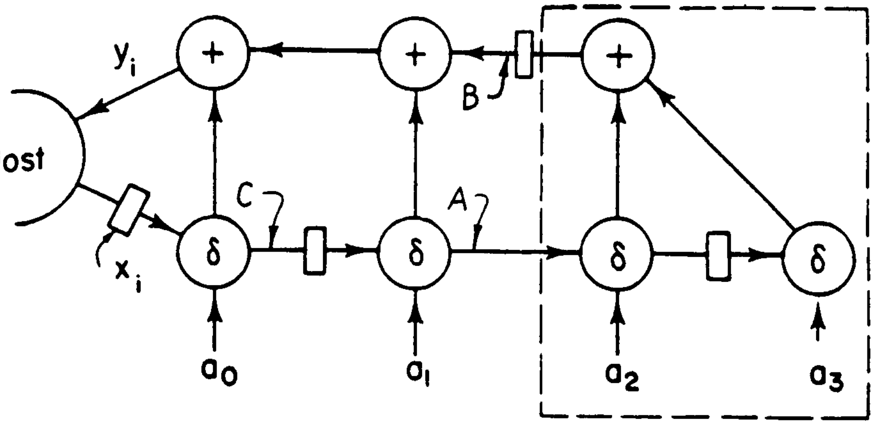


Figure 2. Correlator 2: A *retimed* circuit functionally equivalent to, but more efficient than, Correlator 1. The longest path of combinational rippling begins at the register on connection C, and the clock period of this circuit is 17 esec.

Correlator 1 and Correlator 2 are functionally equivalent, but the performance of the *retimed* circuit Correlator 2 is better than that of Correlator 1. The clock period of Correlator 2 is 17 esec—the time for a signal to propagate from the register on connection C through one comparator and two adders. Notice that the two designs use the same functional elements connected in the same manner and differ only in the locations of registers. Correlator 2 has the I/O characteristic specified by Equation (1), but it should be apparent that a direct verification requires considerably more effort than the verification of Correlator 1.

Retiming, the technique of inserting and deleting registers in such a way as to preserve function, can be used to produce an even faster circuit than Correlator 2. Section 4 gives an implementation of the correlator that achieves a clock period of 13. Remarkably, if the pattern of comparators and adders is extended arbitrarily to the right, a clock period of 14 can always be achieved by retiming. In this paper we exhibit a polynomial-time algorithm for determining a retiming of a circuit that minimizes clock period.

The remainder of this paper is organized as follows. Section 2 presents the graph-theoretic model of synchronous circuits used in this paper. In Section 3 we formally describe the operation of *retiming* [12], in which registers are deleted from some connections of a circuit and added to others so that the circuit function is preserved. Section 4 gives a simple polynomial-time algorithm for minimizing the clock period of a circuit. Section 5 gives an asymptotically more efficient algorithm to solve the same problem. In Section 6 we show that the problem of finding a retiming of a circuit that minimizes clock period can be reduced to an efficiently solvable mixed-integer linear programming problem, thus providing a framework for retiming based on mathematical programming.

Sections 7, 8, and 9 discuss extensions of these results. Section 7 considers the special case where all functional elements have identical propagation delays and shows that optimal retimings can be found more efficiently in this case. The section also discusses the relationship of this work to *systolic* computation and shows how to improve the performance of many systolic circuits in the literature. While earlier sections are concerned with finding retimings that are optimal in the sense of minimizing clock period, Section 8 examines a different optimization criterion, namely minimizing the total amount of state (number of registers) in the retimed circuit. In particular, we show that the problem of retiming a circuit to minimize its state subject to an upper bound on the clock period can be reduced to the linear-programming dual of a minimum-cost flow problem, and hence can be solved optimally in polynomial time. Section 9 extends our methods to a more general circuit model in which individual functional elements may have nonuniform propagation delays—*e.g.,* the low-order output bit of an adder may be available earlier than the high-order bit.

In Section 10, we briefly mention further extensions, including the application of our algorithms to optimal pipelining of combinational circuitry.

2 Preliminaries

In this section we define the notations and terminology needed in the paper and present our graph-theoretic model of digital circuits. We conclude by giving a simple algorithm for determining the minimum feasible clock period of a circuit from its graph.

We can view a circuit abstractly as a network of *functional elements* and globally clocked *registers*. The registers are assumed to have the following characteristics: each has a single input and a single output; all are clocked by the same periodic waveform; and at each clock tick, each storage element samples its input and the sampled value is made available at the output until the next tick. We also assume that changes in the output of one storage element do not interfere with the input to another at the same clock tick. An example of such a storage element is an edge-triggered, master-slave, D-type flip-flop [18].

The functional elements provide the computational power of the circuit. Our model is unconcerned with the level of complexity of the functional elements—they might be NAND gates, multiplexors, or ALU's, for example. Each functional element has an associated ***propagation delay***. The outputs of a functional element at any time are defined as a specified function of its inputs, provided that all the inputs have been stable for a time at least equal to the element's propagation delay. We make the conservative assumption that when an input to a functional element changes, the outputs may behave arbitrarily until they settle to their final values.

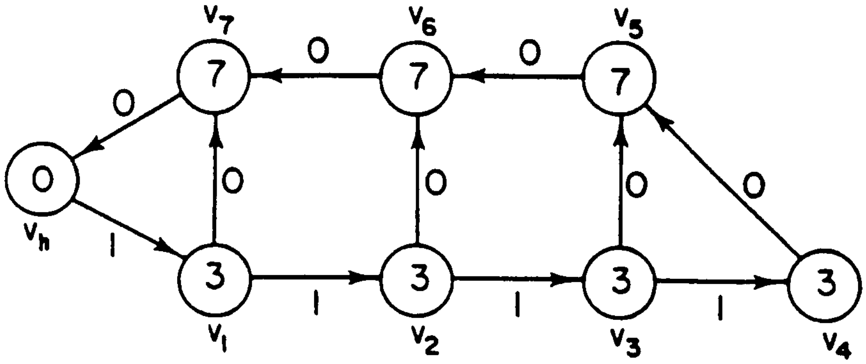


Figure 3. The graph model of Correlator 1 from Figure 1.

To be precise, we model a circuit as a finite, vertex-weighted, edge-weighted, directed multigraph $G = \langle V, E, d, w \rangle$ (henceforth, we shall simply say "graph" or, more frequently, "circuit"). Figure 3 shows the graph of Correlator 1 from Figure 1. The vertices V of the graph model the functional elements of the circuit. Each vertex $v \in V$ is weighted with its numerical propagation delay $d(v)$. The directed edges E of the graph model interconnections between functional elements. Each edge $e \in E$ connects an output of some functional element to an input of some functional element and is weighted with a ***register count*** $w(e)$. The register count is the number of registers along the connection.[2] Between two vertices, there may be multiple edges with different register counts.

Vertices can be designated to represent interfaces with the external world, and each such vertex is given zero propagation delay, as is shown for vertex v_h in Figure 3. (We elaborate on this technicality in Section 10.) If the relative times of events at multiple external interfaces must be preserved, we treat them as a single interface and represent them as a single vertex with multiple incident edges. Otherwise, we assume that multiple external interfaces are independent of each other and cannot communicate with each other externally. For most of our theory, external interfaces can be handled as ordinary vertices, and thus our formalism omits them.

We shall use the following terminology extensively. To avoid confusion between vertex-weight functions such as the propagation delay d and edge-weight functions such as the register count w, we shall use the term ***weight*** for edge-weight functions only. In fact, the only vertex-weight functions we use are the propagation delays $d(v)$, and in general we shall refer to the particular edge weights $w(e)$ of a circuit as register counts. If e is an edge in a graph that goes from vertex u to vertex v, we shall use the notation $u \xrightarrow{e} v$. In the event that the identity of either the head or the tail of an edge is unimportant, we shall use the symbol ?, as in $u \xrightarrow{e} ?$.

For a graph G, we shall view a path p in G as a sequence of vertices and edges. If a path p starts at a vertex u and ends at a vertex v, we use the notation $u \overset{p}{\twoheadrightarrow} v$. A ***simple path*** contains no vertex twice, and therefore the number of vertices exceeds the number of edges by exactly one.

We extend the register count function w in a natural way from single edges to arbitrary paths. For any path $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$, we define the ***path weight*** as the sum of the weights of the edges of the path:

$$w(p) = \sum_{i=0}^{k-1} w(e_i) \; .$$

Similarly, we extend the propagation delay function d to simple paths. For any simple path $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$, we define the ***path delay*** as the

[2] If an output of a functional element fans out to more than one other functional element, the single interconnection can be treated, without loss of generality, as several edges, each with an appropriate weight. Any optimization can be translated from the model back to a circuit with fanout. Section 8 examines fanout more closely.

sum of the delays of the vertices of the path:

$$d(p) = \sum_{i=0}^{k} d(v_i) \ .$$

In order that a graph $G = \langle V, E, d, w \rangle$ have well-defined physical meaning as a circuit, we place nonnegativity restrictions on the propagation delays $d(v)$ and the register counts $w(e)$:

D1. *The propagation delay $d(v)$ is nonnegative for each vertex $v \in V$.*

W1. *The register count $w(e)$ is a nonnegative integer for each edge $e \in E$.*

We also impose the restriction that there be no directed cycles of zero weight:

W2. *In any directed cycle of G, there is some edge with (strictly) positive register count.*

We define a *synchronous circuit* as a circuit that satisfies Conditions D1, W1, and W2. The reason for including Condition W2 is that whenever an edge e between two vertices u and v has zero weight, a signal entering vertex u can ripple unhindered through vertex u and subsequently through vertex v. If the rippling can feed back upon itself, problems of asynchronous latching, oscillation, and race conditions can arise. By prohibiting zero-weight cycles, Condition W2 prevents these problems from occurring, provided that the system clock runs slowly enough to allow the outputs of all the functional elements to settle between each two consecutive ticks.

For any synchronous circuit G, we define the (minimum feasible) *clock period* $\Phi(G)$ as the maximum amount of propagation delay through which any signal must ripple between clock ticks. Condition W2 guarantees that the clock period is well defined by the equation

$$\Phi(G) = \max \{d(p) : w(p) = 0\} \ .$$

For the circuit graph in Figure 3 the clock period is 24, which corresponds to the sum of the propagation delays along the path $v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7$.

Determination of the clock period $\Phi(G)$ is relatively simple. The algorithm we present here is similar to an algorithm that forms a part of a design tool developed at American Microsystems, Inc. [15].

Algorithm CP (*Compute the clock period of a circuit*). This algorithm computes the clock period $\Phi(G)$ for a synchronous circuit $G = \langle V, E, d, w \rangle$.

1. Let G_0 be the subgraph of G that contains precisely those edges e with register count $w(e) = 0$.

2. By Condition W2, G_0 is acyclic. Perform a topological sort on G_0, totally ordering its vertices so that if there is an edge from vertex u to vertex v in G_0, then u precedes v in the total order.
3. Go through the vertices in the order defined by the topological sort. On visiting each vertex v, compute the quantity $\Delta(v)$ as follows:
   a. If there is no incoming edge to v, set $\Delta(v) \leftarrow d(v)$.
   b. Otherwise, set $\Delta(v) \leftarrow d(v) + \max\left\{\Delta(u) : u \xrightarrow{e} v \text{ and } w(e) = 0\right\}$.
4. The clock period $\Phi(G)$ is $\max_{v \in V} \Delta(v)$. ▌

The algorithm works because for each vertex v, the quantity $\Delta(v)$ equals the maximum sum $d(p)$ of vertex delays along any zero-weight directed path p in G such that $? \overset{p}{\rightsquigarrow} v$. The running time is $O(|E|)$.

3 Retiming

Retiming transformations alter the clock period of a circuit by inserting and deleting registers, but without otherwise affecting the circuit's structure. This section formally defines retiming and proves some simple properties of the transformation.

A retiming can be viewed as an assignment of a lag to each vertex in a circuit, and this is how we shall define it formally. A *retiming* of a circuit $G = \langle V, E, d, w \rangle$ is an integer-valued vertex-labeling $r : V \rightarrow \mathbf{Z}$. The retiming specifies a transformation of the original circuit in which the registers are added and removed so as to change the graph G into a new graph $G_r = \langle V, E, d, w_r \rangle$, where the edge-weighting w_r is defined for an edge $u \xrightarrow{e} v$ by the equation

$$w_r(e) = w(e) + r(v) - r(u) \ . \tag{2}$$

In the example of Figure 3, the retiming that assigns -1 to functional elements v_3, v_4, and v_5, and assigns 0 to all other vertices, yields the circuit of Figure 2.

Equation (2), which tells how retiming affects the register counts of edges, extends naturally to paths.

Lemma 1. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, and let r be a retiming. Then for any path $u \overset{p}{\rightsquigarrow} v$ in G, we have*

$$w_r(p) = w(p) + r(v) - r(u) \ .$$

Proof. Suppose p is composed of vertices and edges $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$. We have

$$\begin{aligned}
w_r(p) &= \sum_{i=0}^{k-1} w_r(e_i) \\
&= \sum_{i=0}^{k-1} (w(e_i) + r(v_{i+1}) - r(v_i)) \\
&= \sum_{i=0}^{k-1} w(e_i) + \sum_{i=0}^{k-1} (r(v_{i+1}) - r(v_i)) \\
&= w(p) + r(v_k) - r(v_0)
\end{aligned}$$

because the sum on the right telescopes. ▮

Corollary 2. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, and let r be a retiming on the vertices of G. Then for any cycle p in G, we have $w_r(p) = w(p)$.*

Proof. Immediate from Lemma 1. ▮

A retiming r of a circuit G is *legal* if the retimed graph G_r satisfies Conditions W1 and W2. An arbitrary assignment of lags to the vertices of a circuit G may cause the retimed circuit G_r to violate Condition W1, which says that no edge may have a negative register count. This condition must be checked explicitly in order to ensure that a retiming is legal. Interestingly enough, Condition W2 need not also be checked because of the following consequence of Corollary 2.

Corollary 3. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, and let r be a retiming on the vertices of G such that G_r satisfies Condition W1. Then r is a legal retiming.*

Proof. Since the propagation delays d are unaffected by retiming, Condition D1 is satisfied by graph G_r. Since Condition W1 is true by supposition, it remains only to show that G_r satisfies Condition W2. Let p be any cycle in G. We must show that p includes at least one edge e such that $w_r(e) > 0$. Since graph G satisfies Conditions W1 and W2, the register count $w(p)$ of the cycle in G must be positive. But by Corollary 2, the register count $w_r(p)$ of the cycle in G_r is equal to $w(p)$ and is therefore positive. Hence, there must be an edge on the cycle in G_r that has positive register count. ▮

To conclude this section, we comment that it is necessary to prove that when a circuit G is retimed to produce a new graph G_r, the new circuit is functionally ***equivalent,*** as seen by the external world, to the original—provided, of course, that G_r satisfies Conditions W1 and W2. Such a proof can be found in [12], which also contains a technical definition of the term "equivalent."

Moreover, we can show that retiming is, in a sense, the most general possible method for changing the register counts within a circuit without disturbing the circuit's function. Although we do not formally prove it here, we outline the thread of reasoning. (For an example of a similar argument used to prove a weaker result, see the proof of Theorem 3 in [12]). Without loss of generality, assume that any circuit $G = \langle V, E, d, w \rangle$ under discussion has the following two properties. 1. Every vertex $v \in V$ is connected by a path to some external interface. 2. Every vertex $v \in V$ has at least one input. (Otherwise, v computes a constant function.[3]) Given the graph of such a circuit, but no knowledge of what functions are computed by the functional elements, it is impossible, other than by retiming, to alter the register counts on the edges and be assured that the external behavior is unchanged. For any relabeling of the edge weights that is not a retiming, an adversary can specify the functional elements in such a way that the new circuit behaves differently from the original circuit. We omit the details of this argument.

4 An algorithm for clock period minimization

This section presents a polynomial-time algorithm for retiming a circuit so as to maximize performance. Specifically, we solve the following clock period minimization problem: *Given a circuit graph $G = \langle V, E, d, w \rangle$, find a legal retiming r of G such that the clock period $\Phi(G_r)$ of the retimed circuit G_r is as small as possible.* The solution of this problem depends on some basic results from combinatorial optimization and graph theory. In particular, we rely on the fact that the following linear programming problem can be solved efficiently.

Problem LP. *Let S be a set of m linear inequalities of the form*

$$x_j - x_i \le a_{ij} \qquad (3)$$

[3] In the graph (Figure 1) of Correlator 1, for example, we do not use functional elements to input the constants a_i, but have instead incorporated them into the comparators.

on the unknowns $x_1, x_2, \ldots, x_n$, where the a_{ij} are given real constants. Determine feasible values for the unknowns x_i, or determine that no such values exist.

Constraint systems in which each equation has the form of Inequality (3) have been studied extensively. Any such system of linear inequalities can be satisfied—or determined to be inconsistent—in $O(mn)$ time by the Bellman-Ford algorithm [8, p. 74].

The algorithm for minimizing the clock period of a circuit is based on an alternative characterization of clock period in terms of two quantities which we now define:

$$\begin{aligned} W(u,v) &= \min\left\{w(p) : u \overset{p}{\twoheadrightarrow} v\right\} , \\ D(u,v) &= \max\left\{d(p) : u \overset{p}{\twoheadrightarrow} v \text{ and } w(p) = W(u,v)\right\} . \end{aligned} \tag{4}$$

The quantity $W(u,v)$ is the minimum number of registers on any path from vertex u to vertex v. We call a path $u \overset{p}{\twoheadrightarrow} v$ such that $w(p) = W(u,v)$ a *critical path* from u to v. The quantity $D(u,v)$ is the maximum total propagation delay on any critical path from u to v. Both quantities are undefined if there is no path from u to v. Figure 4 shows the values for Correlator 1.

W	v_h	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_h	0	1	2	3	4	3	2	1
v_1	0	0	1	2	3	2	1	0
v_2	0	1	0	1	2	1	0	0
v_3	0	1	2	0	1	0	0	0
v_4	0	1	2	3	0	0	0	0
v_5	0	1	2	3	4	0	0	0
v_6	0	1	2	3	4	3	0	0
v_7	0	1	2	3	4	3	2	0

D	v_h	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_h	0	3	6	9	12	16	13	10
v_1	10	3	6	9	12	⑯	13	10
v_2	17	20	3	6	9	13	10	17
v_3	24	27	30	3	6	10	17	24
v_4	24	27	30	33	3	10	17	24
v_5	21	24	27	30	33	7	⑭	21
v_6	14	17	20	23	26	30	7	⑭
v_7	7	10	13	⑯	19	23	⑳	7

Figure 4. Tables showing the values of the functions W and D for Correlator 1. The quantity $W(u,v)$ is the number of registers on a minimum-weight path from u to v, and $D(u,v)$ is the maximum propagation delay along any such *critical path.* The distinct entries in the table for D include all possible clock periods for any retiming of Correlator 1. A legal retiming r produces a circuit G_r with clock period $\Phi(G_r) \leq c$ if and only if $W_r(u,v) > 0$ wherever $D(u,v) > c$. Circled entries in the table for D are explained in the last paragraph of Section 4.

Lemma 4. *Let $G = \langle V, E, d, w\rangle$ be a synchronous circuit, and let c be any positive real number. The following are equivalent:*

4.1. $\Phi(G) \leq c$.

4.2. For all vertices u and v in V, if $D(u,v) > c$, then $W(u,v) \geq 1$.

Proof. (4.1 ⇒ 4.2): Suppose $\Phi(G) \leq c$, and let u and v be vertices in V such that $D(u,v) > c$. If $W(u,v) = 0$, then there exists a path p from u to v with propagation delay $d(p) = D(u,v)$, which is greater than c, and register count $w(p) = W(u,v) = 0$. Contradiction.

(4.2 ⇒ 4.1): Suppose 4.2 holds, and let $u \overset{p}{\twoheadrightarrow} v$ be any zero-weight path in G. Then we have $W(u,v) = w(p) = 0$, which implies $d(p) \leq D(u,v) \leq c$. ▮

It is not difficult to compute W by solving the all-pairs shortest-paths problem in G. Common ways of solving this problem are the Floyd-Warshall method [8, p. 86], which runs in $O(|V|^3)$ time, and Johnson's algorithm [5], which runs in $O(|V|\,|E| + |V|^2 \lg |V|)$ time using the Fibonacci heap data structure due to Fredman and Tarjan [1]. The basic operations on weights used by these algorithms are addition and comparison. The following algorithm shows that with a suitably chosen weight function, an all-pairs shortest-paths algorithm can be used to compute both W and D.

Algorithm WD (*Compute W and D*). Given a synchronous circuit $G = \langle V, E, d, w\rangle$, this algorithm computes $W(u,v)$ and $D(u,v)$ for all $u, v \in V$ such that u is connected to v in G.

1. Weight each edge $u \xrightarrow{e} ?$ in E with the ordered pair $(w(e), -d(u))$.
2. Using the weighting from Step 1, compute the weight of the shortest path joining each connected pair of vertices by solving an all-pairs shortest-paths algorithm. (In the all-pairs algorithm, add two weights by performing componentwise addition. Compare weights using lexicographic ordering.)
3. For each shortest-path weight (x, y) between two vertices u and v, set $W(u,v) \leftarrow x$ and $D(u,v) \leftarrow d(v) - y$. ▮

The reason that W and D are important is that they behave nicely under retiming.

Lemma 5. *Let $G = \langle V, E, d, w\rangle$ be a synchronous circuit, and let W and D be defined on G by the equations (4). Let r be a legal retiming of G, and let W_r and D_r be defined analogously on G_r. Then*

5.1. a path p is a critical path of G_r if and only if it is a critical path of G,

5.2. $W_r(u,v) = W(u,v) + r(v) - r(u)$ for all connected vertices $u, v \in V$, and

5.3. $D_r(u,v) = D(u,v)$ for all connected vertices $u, v \in V$.

Proof. Condition 5.1 follows from Lemma 1 because retiming changes the weights of all paths from u to v by the same amount, and then 5.2 follows immediately. Condition 5.3 is a consequence of 5.2 together with the fact that retiming does not alter propagation delays. ▮

The next result is a corollary to Lemma 5 which shows that the range of D contains the clock periods of all circuits obtainable by retiming G. In Figure 4 the 20 distinct values in the table for D include all possible clock periods for any retiming of Correlator 1.

Corollary 6. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, and let r be a retiming of G. Then the clock period $\Phi(G_r)$ is equal to $D(u,v)$ for some $u, v \in V$.*

Proof. By the definition of clock period, the circuit G_r contains some zero-weight path $u \overset{p}{\twoheadrightarrow} v$ such that $d(p) = \Phi(G_r)$, and thus we have $W_r(u,v) = w_r(p) = 0$. Moreover, no zero-weight path in G_r has greater propagation delay than p, which implies $D_r(u,v) = d(p)$. Hence, by Lemma 5 we have $\Phi(G_r) = D_r(u,v) = D(u,v)$. ▮

Lemma 4 and Lemma 5 also allow us to characterize the conditions under which a retiming produces a circuit whose clock period is no greater than a given constant.

Theorem 7. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, let c be an arbitrary positive real number, and let r be a function from V to the integers. Then r is a legal retiming of G such that $\Phi(G_r) \le c$ if and only if*

7.1. $r(u) - r(v) \le w(e)$ for every edge $u \xrightarrow{e} v$ of G, and

7.2. $r(u) - r(v) \le W(u,v) - 1$ for all vertices $u, v \in V$ such that $D(u,v) > c$.

Proof. By Corollary 3, the retiming r is legal if and only if Condition 7.1 holds. If r is indeed a legal retiming of G, then by Lemma 4 the retimed circuit G_r has clock period $\Phi(G_r) \le c$ precisely under the condition that

$W_r(u,v) \geq 1$ for all vertices $u,v \in V$ such that $D_r(u,v) > c$. Since by Lemma 5 we have $W_r(u,v) = W(u,v) + r(v) - r(u)$ and $D_r(u,v) = D(u,v)$, this condition is equivalent to Condition 7.2. ▮

Theorem 7 provides the basic tool needed to solve the clock period minimization problem. Notice that the constraints on the unknowns $r(v)$ in the theorem are linear inequalities involving only differences of unknowns, and thus we have an instance of Problem LP.[4] Therefore, using the Bellman-Ford algorithm to test whether a retimed circuit exists with clock period less than some constant c takes $O(|V|^3)$ time since there can be only $O(|V|^2)$ inequalities.

We now present an algorithm to determine a retiming for a circuit G such that the clock period of the retimed circuit is minimized.

Algorithm OPT1 (*Clock period minimization*). Given a synchronous circuit $G = \langle V, E, d, w \rangle$, this algorithm determines a retiming r such that $\Phi(G_r)$ is as small as possible.

1. Compute W and D using Algorithm WD.
2. Sort the elements in the range of D.
3. Binary search among the elements $D(u,v)$ for the minimum achievable clock period. To test whether each potential clock period c is feasible, apply the Bellman-Ford algorithm to determine whether the conditions in Theorem 7 can be satisfied.
4. For the minimum achievable clock period found in Step 3, use the values for the $r(v)$ found by the Bellman-Ford algorithm as the optimal retiming. ▮

Algorithm OPT1 runs in $O(|V|^3 \lg |V|)$ time. For some circuits, we can sometimes improve the performance of Algorithm OPT1 by using a smaller set of inequalities. (An algorithm with provably better asymptotic performance is given in the next section.) The key observation is that we may eliminate any inequality $r(u) - r(v) \leq W(u,v) - 1$ from Condition 7.2 if either $D(u,v) - d(v) > c$ or $D(u,v) - d(u) > c$. The intuition behind this optimization is that there is no need to explicitly require a path p to have positive weight $w_r(p)$ if we already require some subpath of p to have positive weight.

[4] Actually, we have the *integer linear programming* version of the problem because the unknowns $r(v)$ are required to be integer. Since the value on the right-hand side of each equation is integer, however, the Bellman-Ford algorithm produces an integer optimal solution if one exists.

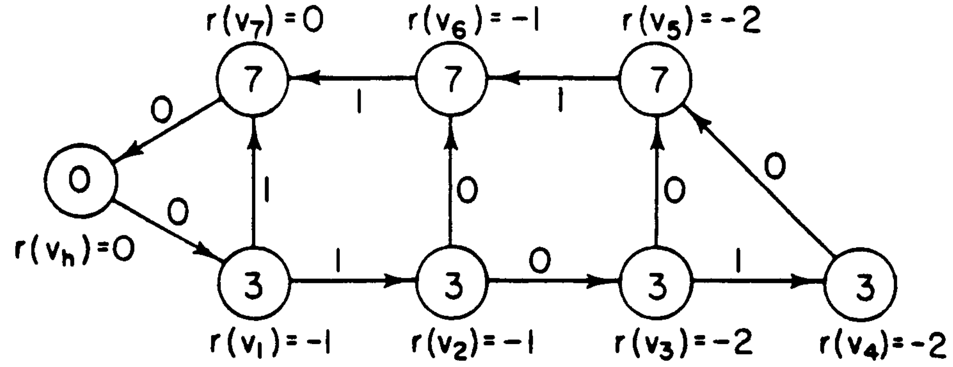


Figure 5. The graph model of an optimal correlator with clock period 13. The circuit is obtained from the graph of Correlator 1 in Figure 3 by applying the optimal retiming, determined by Algorithm OPT1. For each vertex v, the value $r(v)$ is the lag of v with respect to the corresponding vertex in Correlator 1. The retimed weight of an edge $u \xrightarrow{e} v$ is given by $w_r(e) = w(e) + r(v) - r(u)$.

As an example, Figure 5 shows the circuit graph of Correlator 3, which can be obtained from Correlator 1 by applying the Bellman-Ford algorithm to the inequalities from Theorem 7 with clock period $c = 13$. There are 11 inequalities (one for each edge) that must be satisfied to ensure a legal retiming—Condition 7.1 in Theorem 7. Of the potential 34 inequalities arising from cases where $D(u, v) > 13$—Condition 7.2 in the theorem—only five need be included if we eliminate those for which either $D(u, v) - d(v) > 13$ or $D(u, v) - d(u) > 13$. In the table for D in Figure 4, those entries corresponding to the five relevant inequalities are circled.

5 A more efficient algorithm for clock period minimization

In this section we describe an asymptotically more efficient algorithm for the clock period minimization problem. Specifically, we will show that the *feasible clock period test* in Step 3 of Algorithm OPT1, which determines whether there exists a retiming of G with clock period at most c, can be performed in $O(|V||E|)$ time, a significant improvement over $O\left(|V|^3\right)$

for sparse graphs. This result yields an $O(|V|\,|E| \lg |V|)$-time algorithm for determining the optimal retiming.

We begin with the $O(|V|\,|E|)$-time algorithm for determining whether a given clock period is feasible.

Algorithm FEAS *(Feasible clock period test). Given a synchronous circuit $G = \langle V, E, d, w \rangle$ and a desired clock period c, this algorithm produces a retiming r of G such that G_r is a synchronous circuit with clock period $\Phi(G_r) \leq c$, if such a retiming exists.*

```
1. For each vertex $v \in V$, set $r(v) \leftarrow 0$.
2. Repeat the following $|V| - 1$ times:
  2.1. Compute graph $G_r$ with the existing values for $r$.
  2.2. Run Algorithm CP on the graph $G_r$ to determine $\Delta(v)$ for each
       vertex $v \in V$.
  2.3. For each $v$ such that $\Delta(v) > c$, set $r(v) \leftarrow r(v) + 1$.
3. Run Algorithm CP on the circuit $G_r$. If we have $\Phi(G_r) > c$, then no
   feasible retiming exists. Otherwise, $r$ is the desired retiming. ∎
```

Proof of correctness.[5] Algorithm FEAS works by relaxation. Step 1 specifies an initial tentative retiming in which each vertex has zero lag (so that $G_r = G$). Each iteration of Step 2 is equivalent to one pass of a Bellman-Ford algorithm on the constraints in Theorem 7. We assume that the tentative values produced during each pass over the constraint set depend only on the tentative values from the previous pass.

After each iteration of Step 2, the tentative retiming is guaranteed to be legal. Consider an edge $u \xrightarrow{e} v$ in G_r. If the retimed weight $w_r(e)$ is strictly positive at the beginning of the iteration, then it will be nonnegative at the end of the iteration because $r(u)$ can increase by at most 1 and $r(v)$ cannot decrease. If $w_r(e) = 0$ at the beginning of the iteration and if $r(u)$ is incremented, then $r(v)$ will be incremented as well because $\Delta(v) \geq \Delta(u) + d(u) \geq \Delta(u) > c$ in this case.

It remains to show that Step 2 simulates a pass of a Bellman-Ford algorithm on the constraints from Theorem 7. Since the tentative retiming is always legal at the beginning of an iteration, the constraints 7.1 are already satisfied. Thus, the relaxation step in the inner loop of the Bellman-Ford algorithm does not change the value of any $r(v)$ for these constraints.

To see that the effects of the relaxations due to the constraints 7.2 are achieved, consider any two vertices $u, v \in V$. If we have $D(u, v) \leq c$, then no inequality for critical paths from u to v occurs in this constraint set. If

[5] A more detailed proof can be found in [17].

the retimed critical path weight $W_r(u,v) = W(u,v) + r(v) - r(u)$ is positive, then the corresponding inequality in the constraint set is already satisfied. Finally, when $D(u,v) > c$ and $W(u,v) = 0$, there is some path $u \overset{p}{\twoheadrightarrow} v$ such that $w_r(p) = W_r(u,v) = 0$ and $d(p) = D(u,v)$. The existence of this path implies that $\Delta(v) \geq d(p) = D(u,v) > c$, so that $r(v)$ will be given the new value $r(v)+1 = r(u)+W_r(u,v)-W(u,v)+1 = r(u)-W(u,v)+1$, precisely achieving the effect of the desired relaxation of the constraint $r(u) - r(v) \leq W(u,v) - 1$. Conversely, $r(v)$ is incremented only when there exists some path $x \overset{p}{\twoheadrightarrow} v$ such that $w_r(p) = 0$ and $d(p) = \Delta(v) > c$, implying that $D(x,v) \geq d(p) > c$ and $W_r(u,v) = 0$. ▮

Using Algorithm FEAS in Step 3 of Algorithm OPT1 yields the following improved algorithm for clock period minimization.

Algorithm OPT2 (*Clock period minimization*). Given a synchronous circuit $G = \langle V, E, d, w \rangle$, this algorithm determines a retiming r such that $\Phi(G_r)$ is as small as possible.

1. Compute W and D using Algorithm WD.
2. Sort the elements in the range of D.
3. Binary search among the elements $D(u,v)$ for the minimum achievable clock period. To test whether each potential clock period c is feasible, apply Algorithm FEAS.
4. For the minimum achievable clock period found in Step 3, use the values for the $r(v)$ found by Algorithm FEAS as the optimal retiming. ▮

Algorithm OPT2 runs in $O(|V|\,|E| \lg |V|)$ time.

6 A mathematical programming framework for retiming

In this section we describe another algorithm for clock period minimization based on a special case of mixed-integer linear programming. Specifically, we will show that the feasible clock period test can be performed in $O(|V|\,|E| \lg |V|)$ time. Although this bound is not an improvement over the $O(|V|\,|E|)$ bound for Algorithm FEAS, the mathematical programming framework in this section provides further insight into retiming.

The feasible clock period test can be reduced to the following mixed-integer programming problem.

> **Problem MILP.** *Let S be a set of m linear inequalities of the form $x_j - x_i \leq a_{ij}$ on the unknowns $x_1, x_2, \ldots, x_n$, where the a_{ij}*

are given real constants, and let k be given. Determine feasible values for the unknowns x_i subject to the constraint that x_i is integer for $i = 1, 2, \ldots, k$ and real for $i = k+1, k+2, \ldots, n$, or determine that no such values exist.

Although mixed-integer programming is in general NP-complete (because integer programming is [3, p. 245]), this special case can be solved in $O(mn + km \lg n)$ time [13]. The reduction of the feasible clock period test to Problem MILP makes use of the following lemma.

Lemma 8. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, and let c be a positive real number. Then the clock period $\Phi(G)$ is less than or equal to c if and only if there exists a function $s : V \rightarrow [0, c]$ such that $s(v) \geq d(v)$ for every vertex v and such that $s(v) \geq s(u) + d(v)$ for every zero-weight edge $u \xrightarrow{e} v$.*

Proof. For each vertex v, let $\Delta(v)$ be the maximal sum of the combinational delays along any zero-weight path that ends at v. (This Δ is the same as the one in Algorithm CP.) By definition, we have $\Phi(G) \leq c$ if and only if $\Delta(v) \leq c$ for all v. If we have $\Phi(G) \leq c$, the function Δ satisfies the desired properties for s. Conversely, if a function s exists that has the desired properties, then we have $\Delta(v) \leq s(v) \leq c$ for every vertex v. ∎

Lemma 8 and Corollary 3 together give a characterization of when it is possible to retime a circuit so that the retimed circuit has a clock period of c or less.

Lemma 9. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, and let c be a positive real number. Then there exists a retiming r of G such that $\Phi(G_r) \leq c$ if and only if there exists an assignment of a real value $s(v)$ and an integer value $r(v)$ to each vertex $v \in V$ such that the following conditions are satisfied:*

9.1. $-s(v) \leq -d(v)$ *for every vertex* $v \in V$,

9.2. $s(v) \leq c$ *for every vertex* $v \in V$,

9.3. $r(u) - r(v) \leq w(e)$ *wherever* $u \xrightarrow{e} v$, *and*

9.4. $s(u) - s(v) \leq -d(v)$ *wherever* $u \xrightarrow{e} v$ *such that* $r(u) - r(v) = w(e)$.

Proof. Condition 9.3 captures the requirements for r to be a legal retiming, as given in Corollary 3, namely that r be a mapping from V to $\mathbf{Z}$ such that G_r

satisfies Condition W1 (no negative-weight edges). Conditions 9.1, 9.2, and 9.4 capture the requirement for G_r to have a clock period $\Phi(G_r) \le c$ as given in Lemma 8. (Recall that G_r is defined to have $w_r(e) = w(e) + r(v) - r(u)$ for each edge $u \xrightarrow{e} v$.) ▮

Unfortunately, this result does not quite allow us to recast an instance of the feasible clock period test as an instance of Problem MILP because of the qualifying clause "such that $r(u) - r(v) = w(e)$" in Condition 9.4. The next theorem shows that the conditions can be expressed without such a clause.

> **Theorem 10.** *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, and let c be a positive real number. Then there is a retiming r of G such that $\Phi(G_r) \le c$ if and only if there exists an assignment of a real value $R(v)$ and an integer value $r(v)$ to each vertex $v \in V$ such that the following conditions are satisfied:*
>
> *10.1.* $r(v) - R(v) \le -d(v)/c$ *for every vertex* $v \in V$,
>
> *10.2.* $R(v) - r(v) \le 1$ *for every vertex* $v \in V$,
>
> *10.3.* $r(u) - r(v) \le w(e)$ *wherever* $u \xrightarrow{e} v$, *and*
>
> *10.4.* $R(u) - R(v) \le w(e) - d(v)/c$ *wherever* $u \xrightarrow{e} v$.

Proof. Any solution to the conditions in Lemma 9 can be converted to a solution to the conditions above by using the same values for the $r(v)$ and taking $R(v) = r(v) + s(v)/c$ for each vertex v. Conversely, any solution to the conditions above yields a solution to the conditions in Lemma 9 using the substitution $s(v) = c\ (R(v) - r(v))$. ▮

Theorem 10 is the basis for the following improvement on Algorithm OPT1.

Algorithm OPT3 (*Clock period minimization*). Given a synchronous circuit $G = \langle V, E, d, w \rangle$, this algorithm determines a retiming r such that $\Phi(G_r)$ is as small as possible.

1. Compute W and D using Algorithm WD.
2. Sort the elements in the range of D.
3. Binary search among the elements $D(u, v)$ for the minimum achievable clock period. To test whether each potential clock period c is feasible, solve Problem MILP to determine whether the conditions in Theorem 10 can be satisfied.
4. For the minimum achievable clock period found in Step 3, use the values for r found by the algorithm that solves Problem MILP as the optimal retiming. ▮

This algorithm can be made to run in $O(|V|\,|E|\lg|V| + |V|^2\lg^2|V|)$ time by choosing efficient algorithms for each of the steps. If Johnson's all-pairs shortest-paths algorithm [5] using the Fibonacci heap data structure due to Fredman and Tarjan [1] is used in Algorithm WD, Step 1 runs in $O(|V|\,|E| + |V|^2\lg|V|)$ time. Since there are only $O(|V|^2)$ elements in the range of D, Step 2 runs in $O(|V|^2\lg|V|)$ time. Each iteration of the binary search in Step 3 requires solving an instance of Problem MILP with $|V|$ integer variables, $|V|$ real variables, and $2\,|V| + 2\,|E| = O(E)$ inequalities. Thus the total time for Step 3 is $O(|V|\,|E|\lg|V| + |V|^2\lg^2|V|)$ if the algorithm from [13] is used. The optimal retiming from Step 4 is produced as a side effect of Step 3.

7 Unit propagation delay, systolic circuits, and slowdown

This section examines circuits in which the propagation delays of all functional elements are equal. For such circuits, the clock period minimization problem can be solved more simply than for arbitrary circuits. In this section, we explore the relation of this class of circuits to *systolic computation* [6], [7], [9], [12]. We observe that many systolic circuits in the literature can support several independent, interleaved computations. In [12], we introduced a transformation called *slowdown* which, when coupled with retiming, can be used to produce a systolic circuit from an arbitrary synchronous circuit. In this section, we give an efficient algorithm for determining whether any given circuit G can be produced from another circuit (called a *reduced* form of G) by slowdown and retiming. If such a reduced circuit exists, then our algorithm finds one.

We define a circuit $G = \langle V, E, d, w\rangle$ to be a *unit-delay* circuit if each vertex $v \in V$ has propagation delay $d(v) = 1$. The next theorem gives a characterization of when a unit-delay circuit has clock period less than or equal to c. The theorem is phrased in terms of the graph $G - 1/c$, which is defined as $G - 1/c = \langle V, E, d, w'\rangle$ where $w'(e) = w(e) - 1/c$ for every edge $e \in E$. Thus $G - 1/c$ is the graph obtained from G by subtracting $1/c$ from the weight of each edge in G.

> **Theorem 11.** *Let $G = \langle V, E, d, w\rangle$ be a unit-delay synchronous circuit, and let c be any positive integer. Then there is a retiming r of G such that $\Phi(G_r) \le c$ if and only if $G - 1/c$ contains no cycles having negative edge weight.*

Proof. First, suppose $G - 1/c$ has no negative-weight cycles. We shall produce a retiming r of G such that $\Phi(G_r) \leq c$. Assume without loss of generality that there is a path from each vertex v of G to some vertex v_0 (if not, add edges of the form $v \rightarrow v_0$ with sufficiently large weight so that no negative-weight cycles are introduced into $G - 1/c$), and let $g(v)$ be the weight of the shortest path from v to v_0 in $G - 1/c$. For each vertex v, let $r(v) = \lceil g(v) \rceil$.

We now prove that the function r so defined is a legal retiming and that $\Phi(G_r) \leq c$. First we show legality by showing $w_r(e) = w(e) + r(v) - r(u) \geq 0$ for every edge $u \xrightarrow{e} v$. The shortest path in $G - 1/c$ from u to v_0 is at least as short as the path $u \xrightarrow{e} v \overset{p}{\twoheadrightarrow} v_0$, where p is the shortest path (in $G - 1/c$) from v to v_0. Thus, we have $g(u) \leq g(v) + w(e) - 1/c$. Taking ceilings of both sides gives $r(u) \leq \lceil g(v) + w(e) - 1/c \rceil \leq \lceil g(v) \rceil + w(e) \leq r(v) + w(e)$, and thus $w_r(e) = w(e) + r(v) - r(u) \geq 0$, as desired.

Next, we must show that the clock period of the retimed circuit G_r is at most c. That is, we must show that $w_r(p) \geq 1$ for any path $u \overset{p}{\twoheadrightarrow} v$ containing c or more edges. The shortest path from u to v_0 is at least as short as the path $u \overset{p}{\twoheadrightarrow} v \overset{q}{\twoheadrightarrow} v_0$, where q is the shortest path from v to v_0. Furthermore, the total weight along p in $G - 1/c$ is at most $w(p) - 1$ since there are at least c edges in the path. Thus we have $g(u) \leq g(v) + w(p) - 1$, and $w_r(p) = w(p) + \lceil g(v) \rceil - \lceil g(u) \rceil \geq 1$.

On the other hand, suppose $G - 1/c$ contains some cycle p with negative weight. We must prove that G cannot be retimed to have a clock period of c or less. Let n be the number of edges in the cycle p. By the definition of $G - 1/c$, we have $w(p) - n/c = w'(p)$, where w' is the edge-weight function for $G - 1/c$. But by supposition, $w'(p)$ is negative, which means that $w(p) - n/c < 0$, that is, the cycle p contains fewer than n/c registers in G. But retiming leaves the number of registers on any cycle unchanged (Corollary 2). Thus, no matter how the fewer than n/c registers are distributed on the cycle of n vertices, there must be some register-free path with at least c edges and, therefore, with at least $c + 1$ vertices. Consequently, G cannot be retimed to have a clock period of c or less. ▮

To test whether there is a retiming r of a unit-delay circuit G such that $\Phi(G_r) \leq c$, we can use the Bellman-Ford algorithm to find the weight $g(v)$ of the shortest path in $G - 1/c$ from each vertex v to an arbitrary vertex v_0. If the shortest-path weights are not well defined, the Bellman-Ford algorithm detects a negative-weight cycle, which means that no retiming exists. Thus, the feasible clock period test can be performed in $O(|V|\,|E|)$ time for unit-delay circuits using the Bellman-Ford algorithm directly.

A *systolic* circuit is a unit-delay circuit in which there is at least one register along every interconnection between two functional elements. Thus the clock period of a systolic circuit is the minimum possible—the propagation delay through a single functional element. Systolic circuits have been studied extensively ([6], [7], [9], [12]), and they have many applications including signal processing, matrix manipulation, machine vision, and raster graphics.

Interpreted in the context of systolic circuits, Theorem 11 is a generalization of the Systolic Conversion Theorem from [12], which says that G can be retimed to be systolic if the constraint graph $G - 1$ has no cycles of negative weight. (Simply restrict Theorem 11 to the case where $c = 1$.) The Systolic Conversion Theorem is generalized in a different way in [12], however, through the idea of *slowdown*.

For any circuit $G = \langle V, E, d, w\rangle$ and any positive integer c, the circuit cG is the circuit obtained by multiplying all the register counts in G by c. That is, the circuit cG is defined as $cG = \langle V, E, d, w'\rangle$ where $w'(e) = cw(e)$ for every edge $e \in E$. All the data flow in cG is slowed down by a factor of c, so that cG performs the same computations as G, but takes c times as many clock ticks and communicates with the external interfaces only on every cth clock tick. In fact, cG acts as a set of c independent, interleaved instances of G.

If a circuit G can be obtained by retiming a circuit of the form $cG\prime$, then we say that G is a *c-slow* circuit, and more specifically that G is a *c-slow* form of $G\prime$. In this situation $G\prime$ is said to be a *reduced* form of G. The main advantage of a c-slow circuit is that it can often be retimed to have a shorter clock period than any of its reduced forms. For some applications, throughput is the issue, and multiple, interleaved streams of computation can be effectively utilized. A c-slow circuit that is systolic offers maximum throughput.

The following corollary to Theorem 11 tells when a circuit has a c-slow form which is systolic.

> Corollary 12. *Let $G = \langle V, E, d, w\rangle$ be a unit-delay synchronous circuit, and let c be an arbitrary positive integer. Then the following are equivalent:*
>
> *12.1. The graph $G - 1/c$ has no negative-weight cycles.*
>
> *12.2. The circuit G can be retimed to have clock period less than or equal to c.*

12.3. The circuit cG can be retimed to be systolic.

Proof. That 12.1 and 12.2 are equivalent is exactly Theorem 11. The equivalence of 12.1 and 12.3 follows by applying Theorem 11 to cG with clock period 1, and observing that $cG - 1$ has a negative-weight cycle if and only if $G - 1/c$ has a negative-weight cycle. ∎

The registers of a circuit of the form cG are naturally divided into c *equivalence classes.* Given any two registers A and B in cG, the number of registers on any two paths from register A to register B are congruent modulo c. Moreover, if we consider *undirected paths,* in which edges can be traversed in the reverse direction, and if we generalize the notion of path weight by adding 1 for each register on a forward edge and subtracting 1 for each register on a reverse edge, the register counts of two undirected paths from register A to register B are also congruent modulo c. Consequently, the registers are naturally divided into equivalence classes according to their undirected path weight (modulo c) from an arbitrary vertex.

At any given time step, any two registers in different equivalence classes contain data from independent streams of computation—data that can never arrive at inputs of the same functional element at the same time. Although retiming destroys the individual identities of the registers, Lemma 1 guarantees that the registers of any c-slow circuit can still be partitioned into c such equivalence classes.

Using the notion of equivalence classes of registers, the following scenario illustrates the relationships given in Corollary 12. Let $G = \langle V, E, d, w \rangle$ be a unit-delay synchronous circuit. Find an integer c such that $G - 1/c$ has no negative-weight cycles, and consider the circuit cG. There is a retiming r of cG such that $(cG)_r$ is systolic. If we remove all the registers in the c-slow circuit $(cG)_r$ except for those in one equivalence class, the resulting circuit is a retimed form of the original circuit G whose clock period is less than or equal to c.

Many systolic circuits appearing in the literature are 2-slow or 3-slow—even if the ideas of slowdown and retiming were not explicitly used in their design. For example, the systolic algorithms for band-matrix multiplication and LU-decomposition from [7] are 3-slow and can support three independent, interleaved streams of computation. If all independent streams of computation cannot be utilized in a c-slow circuit, it may be desirable to remove all registers except for those in one equivalence class. The following algorithm determines if a circuit is actually a c-slow form of another, and if so, produces a reduced circuit.

Algorithm R (*Remove all but one equivalence class of registers in a circuit*). Given a synchronous circuit $G = \langle V, E, d, w \rangle$ this algorithm determines the largest c such that G is c-slow and produces a reduced circuit G' such that G is a c-slow form of G'.

1. For each vertex $v \in V$, set $dist(v)$ to the weight of some undirected path from v to an arbitrary vertex $v_0 \in V$.
2. Compute $c = \gcd\left\{w(e) + dist(v) - dist(u) : u \xrightarrow{e} v\right\}$.
3. For each vertex $v \in V$, set $r(v) = dist(v) \bmod c$.
4. Produce $G' = \langle V, E, d, w' \rangle$, where $w'(e) = (w(e) + r(v) - r(u))/c$ for each edge $u \xrightarrow{e} v$.

Proof of correctness. We first show that for each edge $u \xrightarrow{e} v$, the value $w'(e)$ is a legal register count. By construction, c evenly divides $w(e) + r(v) - r(u)$ because c divides $w(e) + dist(v) - dist(u)$.[6] Thus, for any edge, the register count $w'(e)$ produced in Step 4 is guaranteed to be an integer. In addition, $w'(e)$ is guaranteed to be strictly greater than -1 because $r(u)$ must be less than c and $w(e) + r(v)$ is at least 0. Since we have just shown that $w'(e)$ is an integer, it must be nonnegative.

The construction in Step 4 directly provides the identity $G' = G_r/c$, and thus G is a c-slow form of G'. We now show that the c computed in Step 2 is the largest possible. Suppose there is a c' such that G is a c'-slow form of another circuit G'. We wish to show that c' divides $w(e) + dist(v) - dist(u)$ for each edge $u \xrightarrow{e} v$, and thus that c' divides c. For every vertex v, the weights of all undirected paths in $c'G'$ from v to v_0 are congruent modulo c'. Since retiming changes all path weights between two vertices by the same amount (which is provable for undirected paths by generalizing Lemma 1), it must be the case that in G, the weights of all undirected paths from v to v_0 are congruent modulo c'. In particular, the weight of the path $u \twoheadrightarrow v_0$ that determines $dist(u)$ and the weight $w(e) + dist(v)$ of the path $u \xrightarrow{e} v \twoheadrightarrow v_0$ must be congruent modulo c'. Hence c' divides $w(e) + dist(v) - dist(u)$. ∎

Step 1 of Algorithm R can be performed in time $O(|V| + |E|) = O(|E|)$ by depth-first search. Step 3 runs in $O(|V|)$ time, and Step 4 takes $O(|E|)$ time. Step 2 takes more work, but not much more. The computation of the greatest common divisor of $|E|$ integers can be performed in $O(|E| + \lg x)$ time, where x is the least nonzero absolute value of any of the numbers. Just start with this value x—which can be found in $O(|E|)$ time—as a

[6] If $w(e) + dist(v) - dist(u) = 0$ for every edge $u \xrightarrow{e} v$, then in Step 2 we get $c = \infty$. Using the standard convention that $x \bmod \infty = x$ and $x/\infty = 0$, Algorithm R yields a reduced graph G' in which each edge $u \xrightarrow{e} v$ has weight $w'(e) = 0$.

tentative gcd, and gcd in each of the other numbers in any order. Each mod operation in Euclid's algorithm either uses up one of the $|E|$ numbers, or else divides the current tentative gcd by the golden ratio $(1+\sqrt{5})/2$.[7] (As a practical matter, starting with any of the $|E|$ numbers as the initial tentative gcd would give reasonable performance, since the number of registers in a typical circuit is much less than exponential in the number of edges.) Thus the total running time of Algorithm R is $O(|E| + \lg x)$.

Observe that Algorithm R works not only for unit-delay circuits, but for any synchronous circuit. Furthermore, when the extra equivalence classes of registers from a c-slow circuit are removed, the clock period of the reduced circuit is not unduly lengthened. The definition of w' in Step 4 provides that for any path p of weight $w(p) \geq c$ in G, we have $w'(p) > 0$, which implies that $\Phi(G') \leq c\Phi(G)$. To guarantee the minimum clock period, however, the reduced circuit must generally be retimed.

A systolic circuit that is naturally c-slow can be converted by Algorithm R into a circuit that performs an operation on every clock tick and whose clock period is bounded by c. This conversion can result in a performance advantage because, in practice, there are time penalties associated with the loading of registers. Because of this overhead, c clock ticks of a circuit with nominal period 1 typically use more time than one clock tick of a circuit with nominal period c. Also, a reduction in registers may save chip area, which can lead to further performance improvements since the wires will in general be shorter. A possible disadvantage of reducing the number of equivalence classes of registers is that throughput is also reduced in cases where the independent streams of computation might be effectively utilized.

8 Register minimization and fanout

Thus far, we have concentrated on clock period as the objective function for determining a retiming. In Section 7, however, we showed that the number of registers in a circuit could sometimes be reduced by a method other than retiming. This section shows that the problem of retiming a circuit to minimize the total *state* of a circuit is polynomial-time solvable by reducing that problem to a minimum-cost flow [8, p. 129] problem. We also show that the total state of a circuit can be minimized subject to a bound on the clock period. These results can be extended to reflect the widths of the

[7] That is, if n mod operations are performed to compute a new tentative gcd, then it will be smaller than the old tentative gcd by at least a factor of $((1+\sqrt{5})/2)^{n-1}$.

interconnections and ways by which *fanout* is modeled.

For a given circuit $G = \langle V, E, d, w\rangle$, the ***state minimization problem*** is to determine a retiming r such that the total state $S(G_r) = \sum_{e\in E} w_r(e)$ of the retimed circuit is minimized. By the definition of w_r, we have

$$\begin{aligned} S(G_r) &= \sum_{e\in E} w_r(e) \\ &= \sum_{u\xrightarrow{e}v} (w(e) + r(v) - r(u)) \\ &= S(G) + \sum_{v\in V} r(v)\,(\text{indegree}(v) - \text{outdegree}(v)) \ . \end{aligned}$$

Since $S(G)$ is constant, minimizing $S_r(G)$ is equivalent to minimizing the quantity

$$\sum_{v\in V} r(v)\,(\text{indegree}(v) - \text{outdegree}(v)) \ , \qquad (5)$$

which is a linear combination of the $r(v)$ since $(\text{indegree}(v) - \text{outdegree}(v))$ is constant for each v. The minimization is subject to the constraint that for each edge $u \xrightarrow{e} v$, the register count $w_r(e)$ is nonnegative—that is,

$$r(u) - r(v) \le w(e) \ .$$

The state minimization problem, cast in this manner, is the linear-programming dual of a minimum-cost flow problem and can be solved in polynomial time by an algorithm due to Edmonds and Karp [8, p. 131]. (As a practical matter, this problem can be efficiently solved by the primal simplex technique [4, p. 186].) Furthermore, since the constants $w(e)$ in the problem are all integers, there is an integer optimal solution to the $r(v)$.

More complicated problems can be solved within the same framework. For example, the total state of a circuit can be minimized subject to a bound on the clock period. Given a maximum allowable clock period c, we wish to find a retiming r that minimizes the state $S(G_r)$ of the retimed circuit subject to the condition that $\Phi(G_r) \le c$. In this case, we must minimize the quantity (5) subject to the constraints from Theorem 7, which require that $r(u) - r(v) \le w(e)$ wherever $u \xrightarrow{e} v$, and that $r(u) - r(v) \le W(u,v) - 1$ wherever $D(u,v) > c$. The state minimization problem remains the dual of a minimum-cost flow problem, but the flow graph is augmented with some extra edges.[8]

[8] For a more explicit description of the duality between state minimization and minimum-cost flow, see [17, section VII.2].

The state minimization problem can be generalized by allowing registers on different edges to have different costs. For example, it may be cheaper to add a register along a one-bit wide control path than along a 32-bit wide data path. We may model such situations by assigning to each edge e a *breadth* $\beta(e)$ proportional to the cost of adding a register along e. The objective function which we must minimize is then given by

$$\sum_{v \in V} r(v)\Bigl(\sum_{? \xrightarrow{e} v} \beta(e) - \sum_{v \xrightarrow{e} ?} \beta(e)\Bigr) , \tag{6}$$

and the constraints on the $r(v)$ are unchanged. This problem is still the dual of a minimum-cost flow problem since the quantity in the large parentheses is a constant for each v. Although the $\beta(e)$ need not be integers, if there is a solution to the state minimization problem, there is an integer optimal solution because the linear programming tableau for the problem is unimodular and the right-hand side is an integer vector. If the $\beta(e)$ are arbitrary numbers, the minimum-cost flow algorithm of Edmonds and Karp does not necessarily run in polynomial time, but the algorithm of Galil and Tardos [2] does.

In a physical circuit, a signal from a register or functional element may *fan out* to several functional elements. As was mentioned in footnote 2 in Section 2, we model this situation with several different edges in the circuit graph. For the clock period minimization problem, there was no harm in modeling fanout in this manner, but for the state minimization problem, there can be. The difficulty that arises in the state minimization problem is that registers can be shared along the physical interconnection. The objective functions (5) and (6) do not take sharing into account.

Fanout can be incorporated into the model in several ways that allow the sharing of registers to be accounted for exactly. We begin by looking at the situation in Figure 6(a) where one vertex u has an output that fans out to two vertices v_1 and v_2. To deal properly with this situation in the state minimization problem, it is sufficient to introduce a dummy vertex $\hat{u}$ with zero propagation delay which models the fork of the interconnection, as is shown in Figure 6(b). When the circuit is retimed to minimize the number of registers, either the edge from $\hat{u}$ to v_1 or the edge from $\hat{u}$ to v_2 will have zero register count, and the edge from u to $\hat{u}$ will have the shared registers. In Figure 6(c) the edge to v_1 ends up with zero weight after retiming so as to minimize the total number of registers.

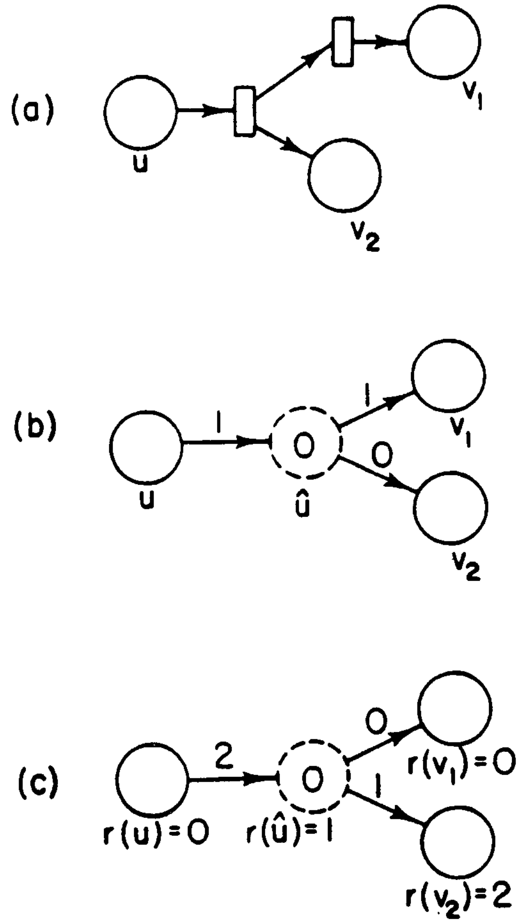


Figure 6. Modeling two-way fanout with an extra vertex having delay zero.

Large multiway forks present some modeling alternatives not encountered in the two-way case. If a physical interconnection is to be modeled, the fork can be decomposed into several two-way forks. (In fact, our concern for modeling the physical interconnection prompted us to design Correlator 1 with the x_i running through the comparators rather than with multiway fanout directly from the external interface.)

For logical design, however, it may be undesirable to model the physical interconnection. In the case of a three-way fork, for instance, we might wish to share the largest possible number of registers between the two edges with greatest register counts, regardless of which two edges these end up being. Modeling a k-way fork for $k \geq 3$ by decomposing the interconnection into two-way forks will not work.

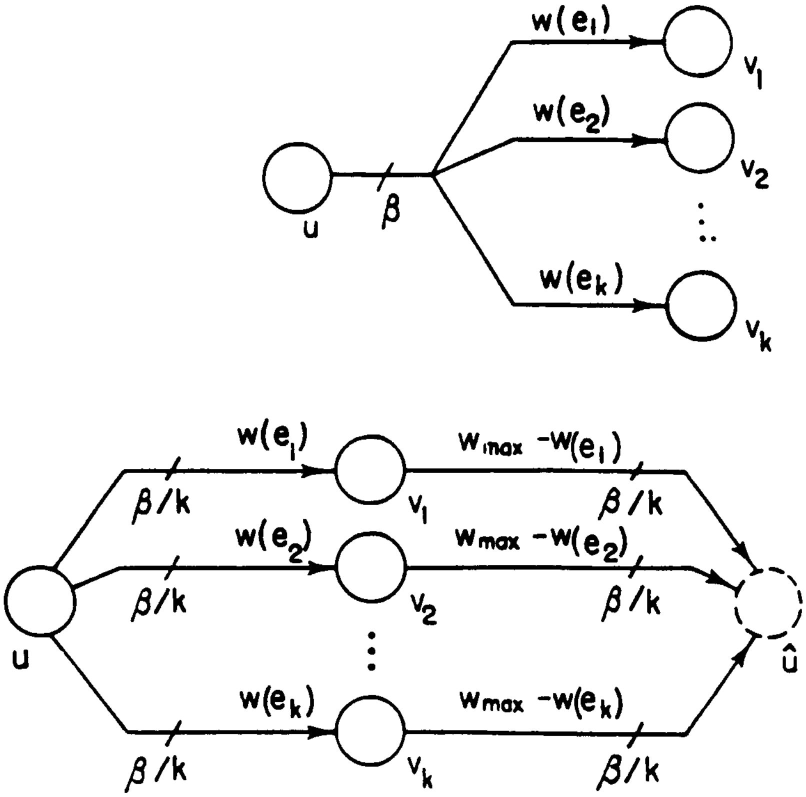


Figure 7. A gadget for modeling the cost of multiway fanout with maximal sharing of registers.

A solution to this problem of modeling k-way fanout with maximum register sharing is depicted in Figure 7. An output of vertex u, having breadth β, fans out to $v_1, \ldots, v_k$ along edges $u \xrightarrow{e_1} v_1, u \xrightarrow{e_2} v_2, \ldots, u \xrightarrow{e_k} v_k$. In the retimed circuit, the cost of this fanout should be β times the maximum of retimed edge weights $w_r(e_i)$. So that the register count cost function $S(G_r)$ will properly model the register sharing, we first add a dummy vertex $\hat{u}$ with zero propagation delay. Letting $w_{\max} = \max_{1 \leq i \leq k} w(e_i)$, we add edges $v_i \xrightarrow{\hat{e}_i} \hat{u}$ with weights $w(\hat{e}_i) = w_{\max} - w(e_i)$. Finally, we give all edges e_i and $\hat{e}_i$ breadths of β/k.

The modified circuit graph accurately models the sharing of registers among the edges e_i involved in the fanout when the state is minimized. For any retiming r, Lemma 1 dictates that the weights $w_r(p_i)$ of all paths $p_i = u \xrightarrow{e_i} v_i \xrightarrow{\hat{e}_i} \hat{u}$ will be identical since they are identical in the unretimed circuit. The retimed register counts $w_r(e_i)$ are constrained by the rest of the circuit, but the weights $w_r(\hat{e}_i)$ will be as small as possible because $\hat{u}$ is a sink in the graph. Thus the register count of one of the $\hat{e}_i$ will be zero, and therefore the weight of each path p_i will be $\max_{1 \leq i \leq k} w_r(e_i)$. Since there are k paths, each with breadth β/k, the total cost of the paths will be $\beta \cdot \max_{1 \leq i \leq k} w_r(e_i)$ as desired.

9 A more general model for propagation delay

In this section we extend the methods of Sections 5 and 6 to deal with functional elements in which the propagation delays through individual functional elements are nonuniform. In an adder, for example, the propagation delay from a low-order input bit to a high-order output bit may be far greater than the propagation delay from a low-order input bit to a low-order output bit or from a high-order input bit to a high-order output bit. Thus, the worst-case propagation delay through two cascaded adders can be much less than twice the worst-case propagation delay through a single adder. This section gives a more general circuit model to handle this commonly occurring situation. We show how the retiming problem in this model can be reduced to simple mixed-integer programming as in Section 6. We also give a more efficient relaxation algorithm similar to that in Section 5.

We may take into account nonuniform propagation delays through functional elements by modifying the model for synchronous circuits given in Section 2, so that from each input to each output of a given functional element, an independent propagation delay may be assigned. Figure 8 shows graphically the "insides" of a functional element in this model. The vertex contains *internal edges* drawn from a set $F \subset E \times E$, and each internal edge belongs to a single vertex. Each normal edge e goes from an internal edge f_a to an internal edge f_b, and we adopt the notation $f_a \xrightarrow{e} f_b$. The propagation delay function d, rather than being a function from V to the nonnegative reals, is a function from F to the nonnegative reals. For an internal edge f in vertex v, the value $d(f)$ denotes the propagation delay through f. A given output of a functional element v need not depend on all the inputs. In an adder, for example, the values of the high-order input bits have no effect on the low-order bits of the output.

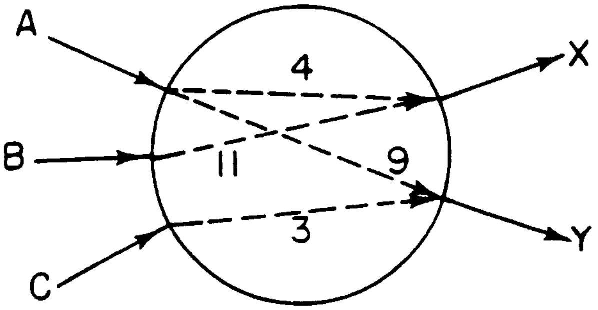


Figure 8. A functional element with nonuniform propagation delays. The time at which output X must settle is either 4 esec after input A settles or 11 esec after input B settles, whichever is later.

Paths in the extended model, rather than going from vertex to vertex, go from an internal edge to an internal edge. For any path $p = f_0 \xrightarrow{e_0} f_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} f_k$, the delay is naturally defined as $d(p) = d(f_0)+d(f_1)+\cdots+d(f_k)$, and the register count of the path is defined as $w(p) = w(e_0)+w(e_1)+\cdots+w(e_{k-1})$. We use the notation $u \overset{p}{\twoheadrightarrow} v$ to denote that path p goes from some internal edge of vertex u to some internal edge of vertex v.

The clock period $\Phi(G)$ of a circuit $G = \langle V, E, d, w\rangle$ is the maximum delay along any path of zero weight. We define retiming, which affects w but not d, exactly as it is defined in the model of Section 2.[9]

The following results show how the clock period minimization problem for a circuit G under the extended model can be reduced to an instance

[9] The possibility that the internal connections between the inputs and outputs of a functional element may not be a complete bipartite graph gives rise to some technical differences between the extended model and the model of Section 2. First, Condition W2 need only be imposed for those cycles in which consecutive edges are actually connected by the internal data paths in the vertices. Second, retiming may not be the only way to adjust register counts so that function is guaranteed to be preserved—if the (undirected) graph of internal connections in some functional element is not connected, then the element can be broken up into two or more independent components which can be given different lags.

of Problem MILP having one integer variable for each functional element of G and one real variable for each edge of G. The results, which parallel Lemma 8, Lemma 9, and Theorem 10, are presented without proof.

Lemma 13. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit in the extended model, and let c be a positive real number. Then the clock period $\Phi(G)$ is less than or equal to c if and only if there exists a function $s : E \to [0, c]$ such that*

13.1. $s(e) \geq d(f)$ *wherever* $f \xrightarrow{e} ?$, *and*

13.2. $s(e_b) \geq s(e_a) + d(f)$ *wherever* $? \xrightarrow{e_a} f \xrightarrow{e_b} ?$ *and* $w(e_a) > 0$.

Lemma 14. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit, and let c be a positive real number. Then there is a retiming r of G such that $\Phi(G_r) \leq c$ if and only if there exists an assignment of a real value $s(e)$ to each edge $e \in E$ and an integer value $r(v)$ to each vertex $v \in V$ such that the following conditions are satisfied:*

14.1. $-s(e) \leq -d(f)$ *wherever* $f \xrightarrow{e} ?$,

14.2. $s(e) \leq c$ *for every edge* $e \in E$,

14.3. $r(u) - r(v) \leq w(e)$ *wherever* $u \xrightarrow{e} v$, *and*

14.4. $s(e_a) - s(e_b) \leq -d(f)$ *wherever* $u \xrightarrow{e_a} f \xrightarrow{e_b} ?$, $r(u) - r(v) = w(e_a)$, *and f is an internal edge of v.*

Theorem 15. *Let $G = \langle V, E, d, w \rangle$ be a synchronous circuit in the extended model, and let c be a positive real number. Then there is a retiming r of G such that $\Phi(G_r) \leq c$ if and only if there exists an assignment of a real value $R(e)$ to each edge $e \in E$ and an integer value $r(v)$ to each vertex $v \in V$ such that the following conditions are satisfied:*

15.1. $r(v) - R(e) \leq -d(f)/c$ *wherever* $v \xrightarrow{e} ?$ *and f is an internal edge of v,*

15.2. $R(e) - r(v) \leq 1$ *wherever* $v \xrightarrow{e} ?$,

15.3. $r(u) - r(v) \leq w(e)$ *wherever* $u \xrightarrow{e} v$, *and*

15.4. $R(e_a) - R(e_b) \leq w(e_a) - d(f)/c$ *wherever* $? \xrightarrow{e_a} f \xrightarrow{e_b} ?$.

Theorem 15 says that the problem of testing whether a given clock period is feasible for a circuit $G = \langle V, E, d, w \rangle$ in the extended model can be efficiently reduced to an instance of Problem MILP having $k = |V|$ integer variables, $n - k = |E|$ real variables, and $m = 2\,|E| + 2\,|d|$ inequalities, where $|d|$ is the number of pairs (e_a, e_b) of edges for which $d(e_a, e_b)$ is defined. Thus the feasible clock period test can be performed in $O\left(|d|\left(|E|+|V|\lg|E|\right)\right)$ time.

We can reduce the cost of the feasibility test to $O(|V|\,|F|)$ by using an algorithm similar to Algorithm FEAS. The dominant cost of this computation is due to $|V|$ executions of an algorithm similar to Algorithm CP, each of which runs in $O(|F|)$ time.

The clock period minimization algorithm for circuits in the extended model is similar to Algorithm OPT2 which performs a binary search over a set of possible values for the minimal clock period. By the same argument used in Corollary 6 for the model of Section 2, the optimal clock period must be equal to $D(u,v)$ for some pair of vertices u and v, where D and W are defined as in Equation (4), but with the semantics of the notation interpreted in the extended model. The values $D(u,v)$ for all connected pairs of vertices u and v can be found in $O\left(|E|\,|F|+|V|\,|E|\lg|E|\right)$ time by an algorithm similar to Algorithm WD. The key step is to apply Johnson's all-pairs shortest-paths algorithm [5] to the edge-weighted graph $H=\langle E,F,wd\rangle$, where the weighting function is defined by $wd(f)=(w(e),-d(f))$ wherever $? \xrightarrow{e} f$ in G (that is, wherever $e \xrightarrow{f} ?$ in H). Using Fibonacci heaps [1] for the priority queue in Johnson's algorithm, a time bound of $O(|E|\,|F|+|E|^2\lg|E|)$ can be achieved. One additional observation is required to prove the claimed time bound of $O\left(|E|\,|F|+|V|\,|E|\lg|E|\right)$. The dominant cost of computing W and D by Johnson's all-pairs shortest paths algorithm is due to $|E|$ applications of Dijkstra's algorithm to find shortest paths from each vertex $e\in H$ to each other vertex. Since W and D are defined on $V\times V$ rather than on $E\times E$, however, we really only need to solve $|V|$ problems of the form: *Given a vertex $v\in V$, find for each $e\in E$ the weight of a shortest path from v to e.*[10] Each such problem can be solved by using Dijkstra's algorithm to find the shortest path weights from a set of vertices in H (namely, from any $x\in E$ such that $? \xrightarrow{x} v$ in G), rather than from a single vertex. Using Fibonacci heaps, the claimed running time for computing W and D is obtained. The total cost of clock period minimization in the extended model is therefore $O(|E|\,|F|+|V|\,|E|\lg|E|)+O(|V|\,|F|\lg|V|)=O(|E|\,|F|+|V|\,|F|\lg|V|)$.

[10] Technically, we should say, "from any internal edge of v to any internal edge of the form $(e,?)$."

10 Concluding remarks

Our goal has been to provide a general framework for the precise understanding of circuit timing. Through the use of a simple graph-theoretic model, we have been able to cast a variety of circuit timing problems in purely combinatorial terms. We believe our approach to be robust. Many other circuit models and many other circuit problems can be handled within the basic framework. We take time here to discuss a few.

Pipelining. An important special case of clock period minimization is the problem of optimally pipelining combinational circuitry. In a *combinational* circuit, all register counts are zero, and thus the circuit graph is acyclic. We can consider the circuit to have one input interface v_I and one output interface v_O. By retiming a combinational circuit G, we can produce a *pipelined* circuit G_r which achieves a shorter clock period at the cost of introducing a *latency* of $r(v_O) - r(v_I)$ clock ticks for signals to propagate from the input interface v_I to the output interface v_O.

In the *optimal pipelining problem*, we are given a combinational circuit G and a nonnegative integer l and asked to produce a retimed circuit G_r with minimum clock period subject to the constraint that the retimed circuit have latency at most l. This problem is just the clock period minimization problem with the additional constraint that $r(u) - r(v) \leq l$. This additional constraint can be modeled by augmenting the circuit G with an edge $v_O \xrightarrow{e} v_I$ having weight $w(e) = l$. Also, the methods of Section 8 can be applied to solve the problem of minimizing the state of pipelined circuitry subject to upper bounds on clock period and latency.

Timing at external interfaces. An external interface may be forced to meet various timing specifications. For instance, if an external interface has a known time delay between the time at which it receives outputs from the circuit and the time at which it presents inputs, the external vertex can be assigned a propagation delay greater than 0. By augmenting the set of inequalities specified in Theorem 10, it is often possible for the optimization algorithms to act subject to other constraints. If data must be available to an interface along some edge $v \xrightarrow{e} v_0$ within some time t after each clock tick, for example, we can express this by the inequality

$$R(v) - r(v_0) \leq \frac{t}{c} + w(e),$$

where v_0 is the vertex representing the interface. This constraint is equivalent to saying that we must have $\Delta(v) \leq t$ if the register count $w_r(e)$ of edge e is zero in the retimed circuit. Similarly, if data from the interface is not available on an edge $v_0 \xrightarrow{e} v$ until some time t after each clock tick, this constraint can be expressed by the inequality

$$r(v_0) - R(v) \leq w(e) - \frac{d(v)}{c} - \frac{t}{c}.$$

Geometric considerations. The optimization methods discussed in this paper can be applied largely independently of geometric considerations because although retiming causes the addition and deletion of registers, it otherwise leaves the functional elements and their pattern of interconnection the same. Thus if a given circuit has an area-efficient layout, chances are that a retimed form of the circuit can be laid out efficiently. In some cases, however, the floorplan of a circuit may limit the number of registers on certain interconnections.

The inequalities that constrain the retimed system can be augmented to express these geometric constraints. For example, to specify an upper bound k on the number of registers that can fit along some edge $u \xrightarrow{e} v$, we can impose the constraint

$$r(v) - r(u) \leq k - w(e).$$

We can also model a situation in which the first k registers on an edge $u \xrightarrow{e} v$ are relatively cheap and additional registers are more expensive. Add an auxiliary vertex $\hat{u}$ in the middle of edge e. Then assign a high cost to registers on the edge $u \rightarrow \hat{u}$ and a low cost to registers on the edge $\hat{u} \rightarrow v$, but constrain $\hat{u} \rightarrow v$ to have at most k registers in the retimed system. Solve the system of constraints as in Section 8. On the other hand, if the first register on a connection is expensive and additional registers are cheap, then it is NP-complete to determine whether a circuit can be retimed to achieve a specified bound on register cost [17, pp. 182–183].

Slowdown. In Section 7 we showed how a c-slow circuit, which supports c independent streams of computation, can be reduced to support a single stream of computation by removing registers. The notion of c-slow circuitry offers new insight into many circuit designs that are not technically c-slow. Consider, for example, a 2-slow circuit in which only one stream of computation is being used. The registers in the circuit fall into two equivalence classes, one of which is idle during each clock period. Using Algorithm R to remove all the registers in one equivalence class is one way to optimize such a circuit.

Another way to save registers is to modify the functional elements to perform slightly different actions on even and odd time steps so that each physical register plays the roles of two logical registers, one in each equivalence class. A cursory examination of the resulting circuit would not reveal that it is 2-slow according to the circuit model, but it would nevertheless communicate with the host only on every other clock tick. Although this method for saving registers may sometimes be acceptable, the overhead of register multiplexing and the complexity of control suggest that Algorithm R is a more reasonable alternative. Moreover, when confronted with a circuit that communicates externally only on every other clock tick or a circuit whose functional elements perform different operations on alternate clock ticks, we may suspect that it is really a 2-slow circuit in disguise, and that penetrating the disguise might lead to improved performance and simplification of the control logic.

Data-dependent propagation delays. A major deficiency of the circuit model is its inability to represent combinational logic elements with data-dependent propagation delays. For example, if a multiplier can produce an answer quickly whenever one of its inputs is zero, its propagation delay is data dependent. One would like to take advantage of the shorter delay whenever possible in order to speed a larger computation.

While we are unable to model data dependence in the general case, we can sometimes use the extended circuit model of Section 9 to partially model the effects. As an example, in nMOS circuits [14], the transition of a Boolean signal from 0 to 1 can take much longer than the transition of the same signal from 1 to 0. We can model this situation somewhat by representing each wire as two edges in the graph, one representing the value 0 on the wire and the other representing 1. We choose propagation delays for internal edges of a functional element depending on how 0 or 1 inputs affect the output. Unfortunately, we cannot model how the delays affect the clock period exactly, but upper bounds can be obtained which will, for example, properly model the propagation delay through two cascaded inverters.

There is much more to be understood about clocked circuits. Do powerful combinatorial optimization techniques apply to other timing models such as those involving multiphase clocking disciplines? Can data-dependent propagation delays be handled in a reasonably general setting? Is it possible to solve the state minimization problem with a polynomial-time algorithm that is simpler than the typical algorithms for solving minimum-cost flow problems? Can hierarchically described circuits be optimally retimed in time proportional to their descriptions? Under what circumstances can optimal retimings of parametrized families of circuits be algorithmically obtained?

Retiming is a transformation that can be used to produce efficient circuits, and we have presented a variety of algorithms for automatically retiming circuits. Of great interest, however, are design methodologies in which retiming is performed by an individual instead of a algorithm, as is done in [9], [10], [12], and [17]. Retiming seems to be a valuable technique which could be incorporated into both circuit compilers and interactive design tools.

Acknowledgments

We would like to offer thanks to many individuals for their contributions to this paper. We wish to give special thanks to Flavio Rose of Digital Equipment Corporation. Flavio was a contributor to the first general algorithm for retiming circuits, which appeared in [11] and [16]. A discussion with Ron Rivest and Rich Zippel of MIT led to our investigation of the state minimization problem. Ravi Boppana of MIT found and corrected several technical errors in an early version of the paper, and Cynthia M. Hibbard suggested many changes that improved the quality of our exposition. Thanks also go to Dan Gusfield of Yale and Jim Orlin of MIT Sloan School for helpful discussions. The ARPAnet was an invaluable resource in the preparation of this paper.

References

[1] Michael L. Fredman and Robert Endre Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Proceedings of the 25th Annual Symposium on Foundations of Computer Science,* IEEE Computer Society, October 1984, pp. 338–346.

[2] Zvi Galil and Éva Tardos, "An $O(n^2 \log n(m + n \log n))$ minimum cost flow algorithm," unpublished manuscript.

[3] Michael R. Garey and David S. Johnson, *Computers and Intractability,* W. H. Freeman and Co., San Francisco, 1979.

[4] Paul A. Jensen and J. Wesley Barnes, *Network Flow Programming,* John Wiley and Sons, Inc., New York, 1980.

[5] Donald B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *Journal of the Association for Computing Machinery,* Vol. 24, No. 1, pp. 1–13, January 1977.

[6] H. T. Kung, "Let's design algorithms for VLSI systems," *Proceedings of the Caltech Conference on Very Large Scale Integration,* Charles L. Seitz, ed., Pasadena, California, January 1979, pp. 55–90.

[7] H. T. Kung and Charles E. Leiserson, "Systolic arrays (for VLSI)," *Sparse Matrix Proceedings 1978,* I. S. Duff and G. W. Stewart, ed., Society for Industrial and Applied Mathematics, 1979, pp. 256–282. (An earlier version appears in Chapter 8 of [14] under the title, "Algorithms for VLSI processor arrays.")

[8] Eugene L. Lawler, *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart and Winston, New York, 1976.

[9] Charles E. Leiserson, *Area-Efficient VLSI Computation*, Ph.D. dissertation, Department of Computer Science, Carnegie-Mellon University, October 1981. Published in book form by the MIT Press, Cambridge, Massachusetts, 1983.

[10] Charles E. Leiserson, "Systolic and semisystolic design," *IEEE International Conference on Computer Design/VLSI in Computers (ICCD '83)*, Rye, New York, October 1983, pp. 627–632.

[11] Charles E. Leiserson, Flavio M. Rose, and James B. Saxe, "Optimizing synchronous circuitry by retiming," *Third Caltech Conference on Very Large Scale Integration,* Randal Bryant, ed., pp. 87–116.

[12] Charles E. Leiserson and James B. Saxe, "Optimizing synchronous systems," *Journal of VLSI and Computer Systems*, Vol. 1, No. 1, Spring 1983, pp. 41–67.

[13] Charles E. Leiserson and James B. Saxe, "A mixed-integer programming problem which is efficiently solvable," *Proceedings of the Twenty-First Annual Allerton Conference on Communication, Control, and Computing*, October 1983, pp. 204–213.

[14] Carver A. Mead and Lynn A. Conway, *Introduction to VLSI Systems,* Addison-Wesley, Reading, Massachusetts, 1980.

[15] Pauline Ng, Wolfram Glauert, and Robert Kirk, "A timing verification system based on extracted MOS/VLSI circuit parameters," *Eighteenth Design Automation Conference Proceedings,* IEEE, 1981.

[16] Flavio M. Rose, *Models for VLSI Circuits,* Masters Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, March 1982. Also available as MIT VLSI Memo No. 82-114.

[17] James B. Saxe, *Decomposable Searching Problems and Circuit Optimization by Retiming: Two Studies in General Transformations of Computational Structures,* Ph.D. dissertation, Department of Computer Science, Carnegie-Mellon University, August 1985.

[18] Texas Instruments Incorporated, *The TTL Data Book for Design Engineers,* Dallas, Texas, 1976.

Index

SRC Reports

"A Kernel Language for Modules and Abstract Data Types."
R. Burstall and B. Lampson.
Report #1, September 1, 1984.

"Optimal Point Location in a Monotone Subdivision."
Herbert Edelsbrunner, Leo J. Guibas, and Jorge Stolfi.
Report #2, October 25, 1984.

"On Extending Modula-2 for Building Large, Integrated Systems."
Paul Rovner, Roy Levin, John Wick.
Report #3, January 11, 1985.

"Eliminating **go to**'s while Preserving Program Structure."
Lyle Ramshaw.
Report #4, July 15, 1985.

"Larch in Five Easy Pieces."
J. V. Guttag, J. J. Horning, and J. M. Wing.
Report #5, July 24, 1985.

"A Caching File System for a Programmer's Workstation."
Michael D. Schroeder, David K. Gifford, and Roger M. Needham.
Report #6, October 19, 1985.

"A Fast Mutual Exclusion Algorithm."
Leslie Lamport.
Report #7, November 14, 1985.

"On Interprocess Communication."
Leslie Lamport.
Report #8, December 25, 1985.

"Topologically Sweeping an Arrangement."
Herbert Edelsbrunner and Leonidas J. Guibas.
Report #9, April 1, 1986.

"A Polymorphic λ-calculus with Type:Type."
Luca Cardelli.
Report #10, May 1st, 1986.

"Control Predicates Are Better Than Dummy Variables For Reasoning About Program Control."
Leslie Lamport.
Report #11, May 5, 1986.

"Fractional Cascading."
Bernard Chazelle and Leonidas J. Guibas.
Report #12, June 23, 1986.

by Charles E. Leiserson and James B. Saxe

digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301