The Practical Application of Retiming to the Design of High-Performance Systems[1]

Brian Lockyear and Carl Ebeling

Department of Computer Science and Engineering
University of Washington
Seattle, Washington 98195

Technical Report 93-5-03
May, 1993

[1]This research was funded in part by the Defense Advanced Research Projects Agency under Contract N00014-J-91-4041. Carl Ebeling is supported in part by an NSF Presidential Young Investigator Award with matching funds provided by IBM Corporation and Sun Microsystems.

Abstract

Many advances have been made recently in the theory of circuit retiming, especially for circuits that use level-sensitive latches. In spite of this, automatic retiming tools have seen relatively little use in practice. One reason for this is the lack of good speedup results when retiming has been applied to real circuits. Another reason is that retiming has used a rather simple circuit model which reduces its utility in practice. This paper addresses both of these issues. We suggest that the reason for the poor results reported for retiming is that retiming has been applied too late in the design process when there is little flexibility for performance improvement. We give an example of using retiming early in the design process to achieve better performance while at the same time simplifying the design process itself. We then describe an extension to the retiming circuit model that includes clock skew as well as latch propagation delay, setup and hold parameters. Including these parameters allows retiming to generate the fastest circuit subject to a given amount of clock skew, or generate the most robust circuit with respect to skew for a given clock frequency. This gives level-clocked circuits yet another advantage over edge-clocked circuits since edge-clocked circuits require margin for clock skew while level-clocked circuits can be retimed to be inherently skew-tolerant. We illustrate these techniques using a serial-parallel multiplier circuit.

1 Introduction

Recently there has been a resurgence of interest in the technique of circuit retiming, largely because of the development of efficient retiming algorithms for latch-based circuits and the emphasis placed by modern high-performance systems such as the Alpha processor [2] on this type of circuit. Retiming is the process of moving the synchronizers within a circuit to reduce the clock period. In essence, retiming repositions synchronizers to utilize the clock period as fully as possible. The most familiar application of retiming is pipelining, where synchronizers added to the inputs or outputs of a circuit are spread throughout the circuit to reduce the amount of computation performed during a single clock cycle. Note that retiming is different from timing verification, which is concerned only with demonstrating that a given circuit is correctly timed and does not attempt to modify it.

An efficient algorithm for retiming circuits using edge-triggered registers, known as edge-clocked circuits, was first described by Leiserson, Rose and Saxe in 1983 [5]. Circuits that use level-sensitive latches, known as level-clocked circuits, allow more flexibility in the scheduling of a computation in the circuit. For this reason, they are generally faster but more difficult to analyze in terms of timing behavior. Only recently has retiming been extended to circuits that use level-clocked circuits [7, 4]. These new algorithms make it possible to design circuits as if they used registers, convert them to level-clocked circuits and then optimize them using retiming.

In spite of these advances in the theory of retiming, there has been relatively little application of retiming in practice, for either edge-clocked or level-clocked circuits. We address two of the reasons for this in this paper. The first reason is that the few results that have been reported on retiming benchmark circuits have not been encouraging [8, 10]. One reason for this has been the choice of circuits such as FSMs for which retiming has little or no benefit. Even more important, we believe, is the use of retiming at the very end of the design process. That is, by the time retiming is applied, most decisions have been made and little flexibility remains in the implementation. We address this in Section 2 where we argue that retiming should be used earlier in the design process as part of a system design methodology that transforms a high-level architecture specification into a high-performance implementation. We use the example of a RISC processor to illustrate this methodology.

The second criticism of retiming is that the circuit model used is overly simplistic in that it ignores important practical parameters like clock skew and latch propagation delay, and setup and hold times. In Section 4 we show how these parameters can be included in the circuit model without increasing the complexity of the retiming algorithms and we describe the modifications needed to do this. The most important parameter is clock skew which can be divided into two components: *fixed* skew, which is built into the delivery of the clock, and *variable* skew, which is caused by variations in process parameters, temperature, power supply voltage and other operating conditions. Fixed skew is that which the designer controls or at least can measure in the clock distribution tree, while variable skew is the unpredictable variation in delay that can occur on the clock signal. By including these parameters, retiming can generate the fastest circuit given a fixed skew, which can be faster or slower than the fastest possible circuit with zero clock skew. Retiming also ensures that the circuit generated will operate correctly in the presence of variable skew, which always slows down the circuit.

In Section 6, we describe how level-clocked circuits can be retimed to maximize tolerance to variation in clock skew in level-clocked circuits. Although minimizing the clock period is equivalent to maximizing clock skew tolerance in edge-clocked circuits, this is not always true for level-clocked circuits. This means that the skew margin for edge-clocked circuits is just the difference between the operating clock period and the minimum clock period while level-clocked circuits can have skew margin even when operating at the minimum clock period.

In the final section, we present the results of using retiming with clock skew on a parallel-serial multiplier circuit. We show that including fixed clock skew when retiming can generate faster circuits. We also show how this circuit can be made more robust to variable skew.

2 The Role of Retiming in Designing High Performance Circuits

Retiming is generally thought of as an optimization method to be used as the final step of the design process. However, retiming a circuit that has been carefully designed for performance is not likely to result in much improvement. This is not the fault of retiming itself but of the inappropriate use of retiming. By the time retiming is applied, too many decisions have been made and retiming can do little to improve the circuit. Figure 1 describes our view of the design of a high-performance system. The starting point is some architectural specification of the system which should be easy for the designer to describe and understand. Unfortunately, it may be very difficult or even impossible to synthesize a high-performance design directly from a straightforward architectural specification.

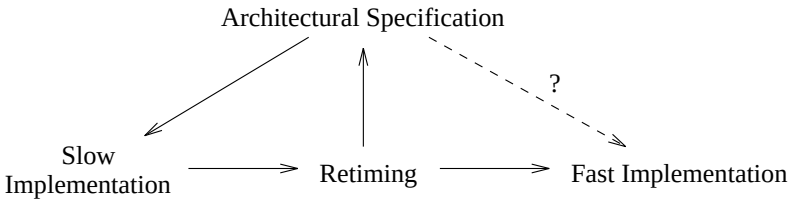

Figure 1: *Retiming as a part of the iterative design process of a high-performance system.*

Retiming offers the possibility of an alternate design methodology. First an implementation is derived directly from the specification which is likely to be too slow to meet the performance goal. Next, this implementation is optimized via retiming to yield a faster implementation. The

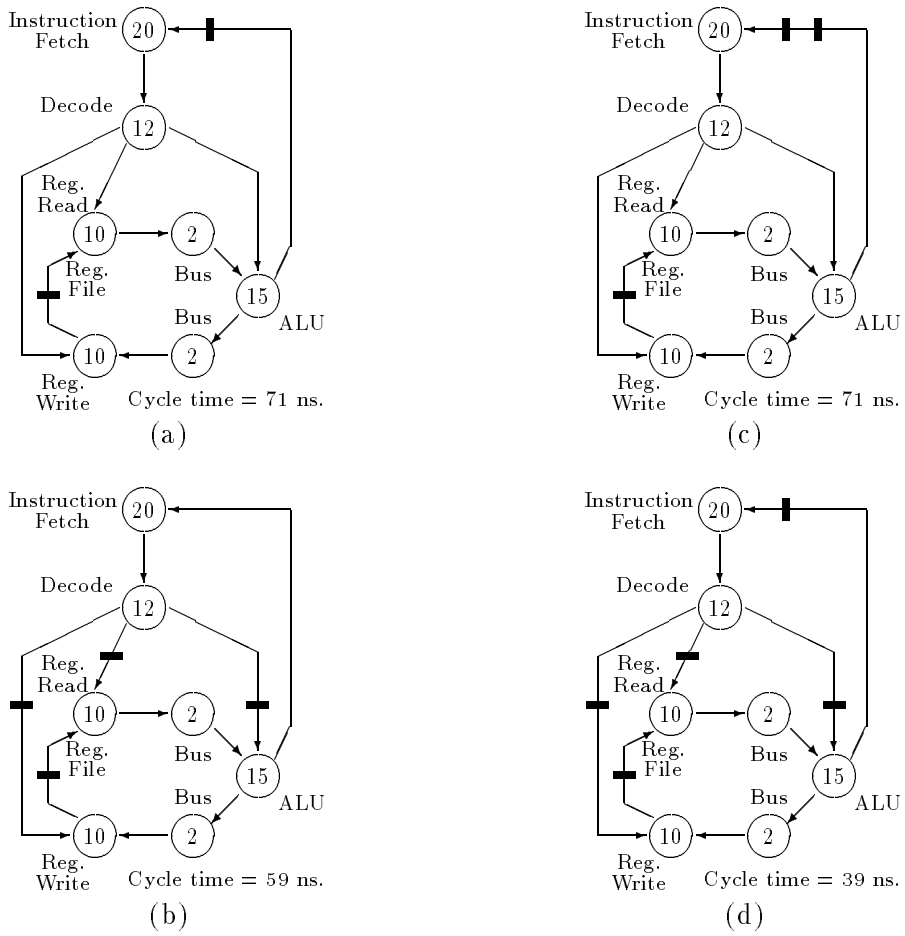

Figure 2: *A simple RISC processor example. Numbers inside vertices are combinational logic delays. a) The initial circuit. b) An optimal edge-triggered retiming of the initial circuit. c) An improved architecture with a single delayed branch register added. d) An optimal retiming of the architecture with a single delayed branch.*

retiming process may not achieve the performance goal either, but it will determine the critical paths in the circuit which limit its performance. The designer can use this information to improve the architectural description to allow a better result to be produced by retiming. Note that the designer does not improve the final implementation, whose functionality is probably obscured by the retiming, but instead improves the underlying architecture to remove the bottlenecks uncovered during retiming.

Figure 2 gives an example of this design process for a simple RISC processor design. This processor is a simple Von Neuman computer which performs an instruction fetch, decode, operand read, ALU operation and operand write to execute each instruction. The figure shows the straightforward implementation of this processor. The register file is modeled as a simple register for the purpose of the timing analysis and but is constrained to be fixed in place. That is, moving the register file is not an option that makes architectural sense. The register leading to the IF unit stores information from the ALU used to determine the address of the next instruction in the case of a branch.

The cycle time for this naive implementation, shown in Figure 2a, is 71 ns. and retiming can

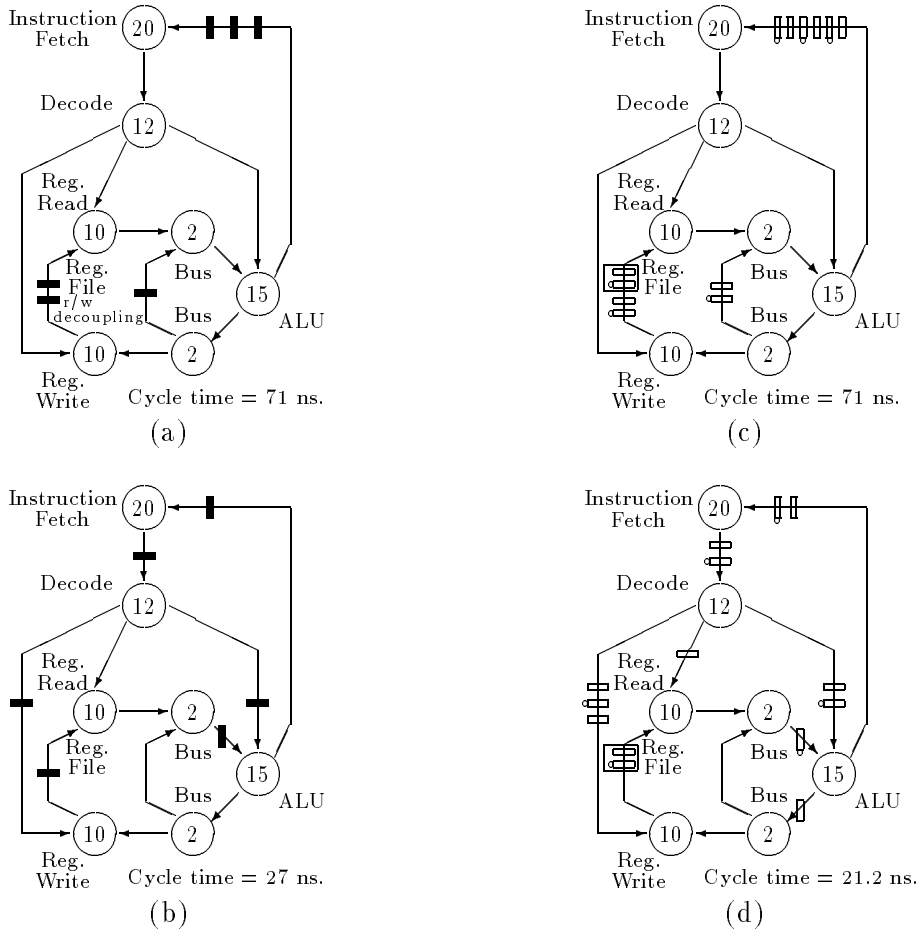

Figure 3: a) The RISC processor modified to have two delayed branch registers, decoupled register file read/write and a register bypass. b) An optimal retiming of the edge-triggered architecture. c) The conversion of circuit (a) to a level-clocked circuit. d) An optimal retiming of the level-clocked circuit.

reduce this only to 59 ns. (Figure 2b). However, the designer can now use the information from the retiming to improve the architecture by introducing a delayed branch. This can be specified simply as an extra delay between the ALU and the IF unit as shown in Figure 2c. The designer does not need to figure out exactly the best way to implement this modification, but instead can rely on retiming to produce the fastest implementation. Figure 2d shows the improvement achieved by this change. The final architectural description, shown in Figure 3a, includes two branch delays, decouples the operand read from the write back of the result and includes a register bypass to avoid ALU stalls. The key point is that the designer is able to make the changes to the architecture in a straightforward way, relying on retiming to generate a high-performance design. This retiming, shown in Figure 3b yields a clock cycle of 27 ns.

To illustrate the performance improvement, and increase in complexity, that results from using level-sensitive latches, Figure 3c shows the same architecture as Figure 3a where each register has been replaced by a pair of latches. Retiming, assuming equal clock phases with no underlap, yields the circuit of Figure 3d, which reduces the clock period to 21.2 ns. (If we use a clock with 10% underlap to avoid problems with clock skew, then retiming yields a clock period of 21.6

ns.) It is even more important here that the designer is shielded from the timing complexities of the level-clocked circuit. In fact, the signals in the implementation can be mapped in time to the corresponding signals in the specification so that when simulating the resulting implementation, the designer interacts with the original description and is thus freed from understanding the complex timing relationship of all the control and data signals in the implementation.

While this example is necessarily simplified to demonstrate the key points of the proposed design methodology, the underlying concepts of this methodology extend to larger and more complex designs where the benefits of retiming should yield even more advantages.

3 Background

We next turn our attention to extending the circuit model used by retiming to include clock skew as well as latch propagation delay and setup and hold times. Before describing this extension, we review the circuit and clock models used in [5, 7] for level-clocked circuits. The reader is encouraged to read these earlier papers for full details.

3.1 Circuit Graph Model

A circuit is represented as a graph with a vertex, v, for each functional element and an edge, $u \xrightarrow{e} v$, for each interconnecting wire. Each vertex has delay, $d(v)$, the maximum delay of the corresponding functional element. A unique host vertex v_h, with $d(v_h) \equiv 0$, represents the environment external to the circuit. Registers and latches are placed on the edges connecting vertices and each edge has a weight, $w(e)$, indicating the number of registers or latches on the connection.

A path $u \xrightarrow{p} v$ is a sequence of vertices and edges from u to v. A *simple* path contains no vertex twice. The weight $w(p)$ of a path $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{n-1}} v_n$ is the number of registers or latches placed along it, that is, the sum of the edge weights: $w(p) = \sum_{i=0}^{n-1} w(e_i)$. We say that registers or latches are *adjacent* if the path connecting them has zero weight. The weight of a cycle is the weight of the same sequence of edges and vertices treated as a path. Similarly, the delay of a path $d(p)$ is the sum of the delays of the vertices along the path: $d(p) = \sum_{i=0}^{n} d(v_i)$. The delay of a cycle $d(c)$, $c = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots v_{n-1} \xrightarrow{e_{n-1}} v_0$, includes the delay of vertex v_0 only once; hence $d(c) = \sum_{i=0}^{n-1} d(v_i)$.

A circuit G is transformed into a corresponding retimed circuit G_r through assignment of a *retiming* (or lag) value $r(v)$ to each of the vertices in G. This retiming value represents the number of registers (latches) removed from the output edges of vertex v and added to the input edges. The resulting weight of an edge $u \xrightarrow{e} v$ in the retimed graph is: $w_r(e) = w(e) + r(v) - r(u)$.

3.2 Clock Model

For our retiming work we have adopted the clock model of Sakallah, Mudge & Olukotun [9] which provides a convenient way to describe the resulting timing constraints. A *k-phase clock* is a set of k periodic signals, $\Phi = \{\phi_1 \ldots \phi_k\}$, where ϕ_i is *phase i* of the clock Φ. All ϕ_i have a common cycle time T_Φ. An edge-triggered circuit has a single clock phase and values are passed through the registers once per cycle on the "clocking" edge. For level-clocked circuits, each phase divides the clock cycle into two intervals as shown in Figure 4: an *active* interval of duration T_{ϕ_i} and a *passive* interval of duration $(T_\Phi - T_{\phi_i})$. The latches controlled by a clock phase are *enabled* during its active interval and *disabled* during its passive interval. The clock transitions *into* and *out of* the

active interval are called the *enabling* and *latching* edges respectively. We refer to the clock phase controlling latch l as $P(l)$.

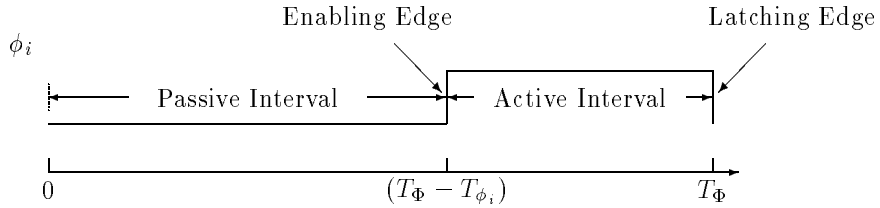

Figure 4: *Diagram from Sakallah et al. showing a clock phase ϕ_i and its local time zone.*

Relative to the beginning of its passive interval at time $t = 0$, the enabling edge of a phase occurs at $t = T_\Phi - T_{\phi_i}$, and its latching edge at T_Φ (Figure 4). Sakallah et al. additionally introduce an arbitrary *global time reference* and values e_i denoting the time relative to the global time reference at which phase ϕ_i *ends* for some specified cycle. Phases are ordered relative to the global time reference so that $e_1 \leq e_2 \leq \cdots \leq e_{k-1} \leq e_k$ and $e_k \equiv T_\Phi$. The phase following ϕ_i in the clock set is referred to as ϕ_{i+1} with phase $\phi_{k+1} \equiv \phi_1$ and $\phi_{1-1} \equiv \phi_k$.

Finally, a *phase shift* operator, $E_{i,j}$, is defined as:

$$E_{i,j} \equiv \begin{cases} (e_j - e_i), & \text{for } i < j \\ (T_\Phi + e_j - e_i), & \text{for } i \geq j \end{cases}$$

$E_{i,j}$ takes on positive values in the range $(0, T_\Phi]$. When subtracted from a time point in the *current* period of ϕ_i, it changes the frame of reference to the *next* period of ϕ_j, taking into account a possible cycle boundary crossing (Figure 5). Because the period of each phase is identical and $e_i \geq e_{i-1}$, the sum of the shifts between k successive phases is T_Φ:

$$\sum_{i=1}^{k} E_{i,i+1} = T_\Phi. \tag{1}$$

A symmetric level-clocked schedule is one in which all active phase periods T_{ϕ_i} are equal and all phase shifts $E_{i,i+1} = \frac{T_\Phi}{k}$.

The latest arrival time of a signal at latch l is denoted by A_l and the latest departure time by D_l, both in terms of the local time zone.

Note that this clock model does not provide for clock phases with differing periods nor for gated clock signals. For simplicity, we assume that the delay characteristics of latches do not vary as they are moved across combinational logic.

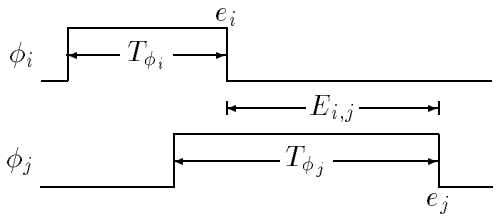

Figure 5: *The phase shift operator provides the relative difference between times in the local time zones of different phases.*

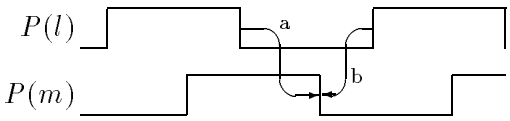

Figure 6: *Constraints on clock phases required for valid schedules. Latches l and m are any pair connected by a zero-weight path $l \twoheadrightarrow m$.*

3.3 Correct Operation, Valid Schedules and Well-formed Circuits

We define a circuit to be correctly timed if for any pair of adjacent latches, the signal leaving the first arrives at the second during the *next* clock period in edge-triggered circuits or the next clock phase in level-clocked ones. Thus the definition of correctness for a level-clocked circuit is a straightforward extension of the definition of correct operation commonly used for edge-clocked circuits. Timing correctness can be summarized as a pair of timing constraints for each case.

For any two adjacent latches l and m in a level-clocked circuit:

L1. Maximum Delay: $A_m = D_l + d(p) - E_{P(l),P(m)} \leq T_\Phi$

L2. Non-interference: $A_m = D_l + d(p) - E_{P(l),P(m)} > t_{hold}$

These constraints assume that clock skew, latch propagation delay and setup time are all zero. These parameters will be added later in this paper.

The retiming techniques in [7] restrict the type of clocks and circuits that can be retimed. Clock schedules must be "valid" so that only maximum delay constraints need to be satisfied for correct operation. Valid clock schedules do not allow races to occur even if all circuit delays are zero. This is guaranteed if for any two latches l and m connected by a zero-weight path $l \twoheadrightarrow m$, $E_{P(l),P(m)} > 0$ (true by definition and illustrated by constraint **a** in Figure 6) and $E_{P(m),P(l)} > T_{\phi_l}$ (constraint **b** in Figure 6). In general, the class of valid clocks includes schedules with phase overlap, underlap, or both; however, two-phase clocks are required to be non-overlapping and no single phase schedules are allowed.

In addition, level-clocked circuits must be "well-formed," that is, latches must occur in phase order along every path.[2] Retiming is always free to move latches across vertices in well-formed circuit graphs and well-formed circuits remain well-formed following retiming. Furthermore, the minimum number of registers required on a path can be computed solely from the path delay and the phase of the first latch.

Since this work extends our previous work, these restrictions on clock schedules and level-clocked circuits also apply to this paper. Although the designer is constrained to work within this model, it applies to most circuits commonly used in practice.

4 Clock Skew Parameters

Clock skew can be divided into two parts, the fixed clock skew and the variable clock skew. Fixed clock skew refers to the skew which is known to be in the path to a latch. Fixed skew results from delays in the clock distribution network which is controlled, or at least can be measured, by the

[2] A simple exception to this rule allows input and output signals at the host node to occur on different phases as long as timing constraints across the host are not imposed.

designer. We use two parameters to model the fixed clock skew: $\sigma_e(l)$ and $\sigma_l(l)$, the fixed skew of the enabling and latching edge respectively of the clock at latch l. If skew is positive the clock arrives late relative to the reference clock and, if negative, it arrives early. For an edge e, $\sigma_e(e)$ and $\sigma_l(e)$ are the fixed skew of a clock at the physical circuit location of the wire represented by the edge. Adding to the clock skew effectively moves an equivalent amount of delay from the fanins of the latches to the fanouts.

Variable clock skew is the remaining, unpredictable clock skew which results from variations in fabrication parameters and operating conditions. The maximum amount by which these variations may shift clock edges in either direction away from the fixed skew is modeled by the parameter σ_ϵ. Thus an enabling edge may arrive as late as $\sigma_e(l) + \sigma_\epsilon$ and as early as $\sigma_e(l) - \sigma_\epsilon$. In this work we assume that variable skew is the same across the circuit, although our techniques can easily be extended to make the variable skew depend on physical location.

In order for us to be able to use the efficient algorithms we have already developed for retiming circuits, we must make the following restriction on clock skew. The relative clock skew between the input and output edges of a vertex, that is the difference in the clock skew between the latches on these edges, must be no greater than the maximum delay of the vertex. We call this the *skew monotonicity* constraint and it ensures that we need not consider the case where increasing the length of a path reduces the number of latches required. It is easy to see that this constraint is met by practical circuits. Note that this does not impose a minimum on the combinational delay. If the actual delay is less than the skew, a more conservative circuit than necessary will be generated by the algorithms. Formally, the requirement is stated as:

- For every vertex v and pair of edges $e \in \text{fanin}(u)$ and $\acute{e} \in \text{fanout}(u)$:

  SR: $\sigma_e(\acute{e}) - \sigma_e(e) \leq d(u)$
  SF: $\sigma_l(\acute{e}) - \sigma_l(e) \leq d(u)$

In addition to parameters for clock skew, it is straightforward to add register or latch propagation delay, P, and required setup time, S, into our model. We make the simplifying assumption that P and S are the same for all latches in the circuit. Algorithms for asymmetric level-clocked circuits in [7, 4] could be extended to solve problems in which latch setup time varies with the physical placement of latches. However, since retiming constraints are stated in terms of the retiming values $r(u)$ and $r(v)$ of the vertices at the beginning and end of a path, they can refer only to the weight of the path, not the location of latches on the path. Thus it is not possible to vary the latch propagation delay based on latch location.

5 Level-Clocked Circuits and Synchronization Parameters

The timing constraints in level-clocked designs are derived from the time span between two clock edges, from the edge enabling the first latch on a path to the edge latching the final latch. Latch propagation delay and setup time directly reduce the time available for computation along circuit paths, while clock skew changes the time span between the relevant edges (see Figure 7).

Unlike edge-triggered timing constraints, level-clocked timing constraints extend across paths with multiple latches. Signals are required to arrive at the final latch of a path S time prior to the arrival of the falling edge. The constraints must account for the propagation delay of all intermediate latches, however, as well as the initial one (see Figure 7).

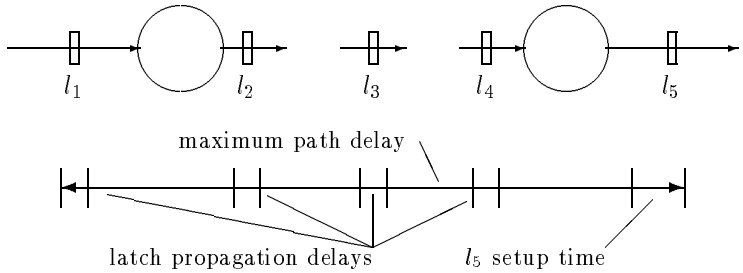

Figure 7: *The propagation delay of each latch must be counted against the maximum path delay; however, only the final latch's setup time is counted.*

With skew, the departure time of a signal from a latch l is given by:

$$D_l = \max\left\{A_l, T_\Phi - T_{P(l)} + \sigma_e(l) + \sigma_\epsilon\right\} + P, \tag{2}$$

and the arrival time at a subsequent latch m connected by a zero-weight path is:

$$A_m = D_l + d(p) - E_{P(l),P(m)}. \tag{3}$$

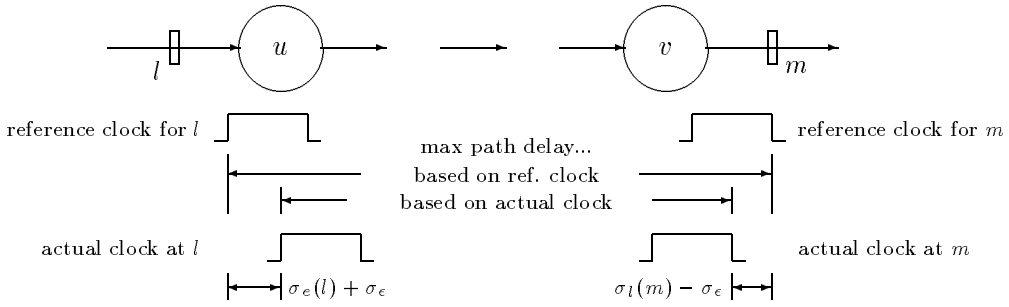

Figure 8: *Late arrival of a signal at latch l or early arrival at m reduces the maximum allowable delay of $u \rightarrow\!\!\!\!\rightarrow v$ and may require a latch to be placed on the path.*

Correct timing requires that signals arrive at latch m S time before the earliest occurrence of its falling edge at $T_\Phi - \sigma_l(m) - \sigma_\epsilon$ (see Figure 8). Thus the delay constraints for each pair of adjacent latches l and m in a level-clocked circuits become:

L1'. Maximum Delay: $A_m = D_l + d(p) - E_{P(l),P(m)} \leq T_\Phi - S - \sigma_l(m) - \sigma_\epsilon$

L2'. Non-interference: $A_m = D_l + d(p) - E_{P(l),P(m)} > t_{hold} + \sigma_l(m) + \sigma_\epsilon$

We first address the L2' constraint which constrains the minimum delay in the circuit. By combining this constraint with the earliest possible departure time of a signal from latch l which is $T_\Phi - T_{P(l)} + P + \sigma_e(l) - \sigma_\epsilon$, we get a minimum delay constraint of:

$$d(p) \quad > \quad t_{hold} - P + \sigma_l(m) - \sigma_e(l) + 2\sigma_\epsilon - T_\Phi + T_{P(l)} + E_{P(l),P(m)} \tag{4}$$

where the term $T_\Phi - T_{P(l)} - E_{P(l),P(m)}$ is the amount of underlap between the two clock phases. In our previous work we have ignored short paths by relying on valid clock schedules to avoid races

even when the minimum delay is zero. If the latch hold, propagation delay, and clock skew are all zero as we assumed in previous work, then this reduces to a constraint that the path delay be greater than the phase overlap (which corresponds to negative underlap). Allowing minimum delays to be zero then leads to the definition of a valid clock schedule, which for the case of 2-phase clocks forbids any phase overlap.

We can weaken the valid clock schedule constraint somewhat by ensuring via a static check that the minimum delay constraint will be satisfied regardless of retiming. We do this by checking that the constraint given in Eqn. 4 is satisfied by each path consisting of zero or one vertex in the circuit graph. Since all longer paths are composed of subpaths with zero or one vertex then it can be shown that any retiming will satisfy L2' if each of these subpaths satisfies Eqn. 4. Note that the underlap value used in the static check must be the smallest over all pairs of successive phases, since the latches assigned to the edges may use any pair.

Theorem 1: *For a level-clocked circuit graph G, if constraint Eqn. 4 is satisfied for every path with zero or one vertex, then constraint E2' is satisfied for any retiming of G.*

Proof Omitted

5.1 Maximum Path Delay Constraints

By combining the earliest possible departure time of a signal from l, $T_\Phi - T_{P(l)} + \sigma_e(l) + \sigma_\epsilon + P)$, and the latest permissible arrival at each successive latch on a path in turn, we can compute the maximum allowable delay along a signal path of any weight. The next theorem uses the definition of correct circuit timing to identify the latching clock edge for each latch. Since latches occur in phase order along paths of well-formed circuits, the available computation time is computed by summing successive phase shifts. An important aspect of this result is that only the skew of the initial enabling and final latching edges appears in the maximum path delay constraint. Although each internal latch along the path may be affected by skew, the skew is added to the time on one side of the latch and subtracted from the time on the other side, thus canceling out. Thus clock skew can vary with latch location since the skew appearing in the maximum delay constraints can be associated with specific edges.

Theorem 2: *A multi-phase, level-clocked graph using a valid clock schedule is correctly timed only if the delay of any simple path $l_0 \overset{p}{\rightsquigarrow} l_{n+1}$ is bounded by:*

$$d(p) \leq T_{\phi_{l_0}} + \sum_{i=0}^{w(p)} \Big(E_{P(l_i),P(l_{i+1})} - P \Big) - \sigma_e(l_0) + \sigma_l(l_{n+1}) - 2\sigma_\epsilon - S.$$

Proof sketch. Through induction on the path weight, and substitution of $D_l = T_\Phi - T_{P(l)} + \sigma_e(l) + \sigma_\epsilon + P$ for the earliest possible departure time of a signal from latch l and $A_{l_{n+1}} = T_\Phi - \sigma_l(l_{n+1}) - \sigma_\epsilon$ for the maximum permissible arrival time at latch m, Eqn. 3 becomes:

$$\begin{aligned} d(p) \quad \leq \quad & T_{\phi_{l_0}} - \sigma_e(l_0) - \sigma_\epsilon + \sigma_l(l_0) - \sigma_\epsilon - S + \\ & \sum_{i=0}^{w(p)} \Big(E_{P(l_i),P(l_{i+1})} + (\sigma_l(l_{i+1}) + \sigma_\epsilon - \sigma_l(l_i) - \sigma_\epsilon) \Big) - \sum_{i=0}^{w(p)} P. \end{aligned}$$

This equation simplifies to the desired result. ■

5.2 Critical Cycles

Because the latch at the start of the cycle is the same as the one at the end, the maximum delay allowed around a complete cycle in the circuit is unaffected by clock skew or setup time. It is, however, reduced by the total latch propagation delay. This amount is fixed since propagation delay is constant and the number of latches on a cycle is unchanged by retiming. The lower bound on possible clock periods caused by circuit cycles may again be found using the max-ratio-cycle algorithm.

Theorem 3: *A multi-phase, level-clock graph using a valid clock schedule is correctly timed only if the delay of any cycle* $l_0 \xrightarrow{c} l_{n+1}$ *is bounded by:*

$$d(c) \leq \sum_{i=0}^{w(c)-1} \left(E_{P(l_i),P(l_{i+1})} - P \right)$$

Proof omitted.

Corollary 4: *A well-formed graph using a k-phase clock schedule is correctly timed if and only if:*

$$\forall \text{ cycles } c \in G : T_\Phi \geq k \left(\frac{d(c)}{w(c)} + P \right).$$

Proof sketch: In a well formed graph each cycle must contain some multiple of k latches, therefore the value $\sum_{i=0}^{w(c)-1} \left(E_{P(l_i),P(l_{i+1})} - P \right)$ reduces to $\frac{w(c)}{k} T_\Phi - w(c)P$. If cycle constraints are satisfied and the previous departure time of a signal was prior to the setup period (which must be guaranteed independently by path constraints), then the signal's arrival after traversing the cycle will be prior to the setup period as well. Thus required setup time S does not appear as a parameter in cycle constraints. ■

Using a maximum-ratio-cycle algorithm such as the one in [1] we solve for the maximum value of $\frac{d(c)}{w(c)}$ over all circuit cycles. This in turn identifies the critical cycle bound, or the minimum possible value of T_Φ to which the circuit can be retimed. To identify the minimum period possible through retiming, a search is performed at and above this bound for the minimum period at which a retiming satisfying path constraints can be found.

5.3 Critical Paths

Being able to identify the critical paths in the circuit graph instead of enumerating the constraints for all paths is crucial to finding a polynomial time bound on the retiming algorithm. Lemma 5 provides a characteristic of critical paths which allows them to be identified using an all-pairs-shortest-paths algorithm.

Lemma 5: (modified from Lemma 5.5 [7]) *A path* $u \xrightarrow{p} v$ *in a well-formed circuit is a critical path iff:*

$$\{w(p) \left(\frac{T_\Phi}{k} - P \right) - d(p)\} \leq \{w(q) \left(\frac{T_\Phi}{k} - P \right) - d(q)\} \text{ for all } u \xrightarrow{q} v.$$

Proof sketch. Clock skew and setup time reduce the slack equally along all paths between two vertices, and thus do not affect which path is critical. ■

The weight and delay of critical paths are again defined as $W(u,v)$ and $D(u,v)$ respectively. The critical fanin and fanout edges are again those for which:

$$\begin{aligned} \sigma_{\text{rmax}}(u) &= \max\{\sigma_e(e) : e \in \text{fanin}(u)\} \\ \sigma_{\text{fmin}}(v) &= \min\{\sigma_l(e) : e \in \text{fanout}(v)\}. \end{aligned}$$

Substitution of these values into Theorem 2 and combining that result with Theorem 3 leads to Corollary 6, which defines all timing requirements which must be satisfied for correct timing of a symmetric-phase, level-clocked circuit:

Corollary 6: *A well-formed, level-clocked circuit graph G, using a symmetric, k-phase clock schedule is correctly timed if and only if the weights $W(u,v)$ of all critical path are bounded by:*

$$W(u,v) \geq \left(\frac{D(u,v) - T_\phi + \sigma_{\text{rmax}}(u) - \sigma_{\text{fmin}}(v) + 2\sigma_\epsilon + S}{\frac{T_\Phi}{k} - P} \right) - 1.$$

and, for all cycles c:

$$T_\Phi \geq k \left(\frac{d(c)}{w(c)} + P \right).$$

Proof omitted.

For simplicity, several parameters from Corollary 6 can be combined into a single value which represents the effective delay of a path:

$$\gamma(u,v) = D(u,v) + \sigma_{\text{rmax}}(u) - \sigma_{\text{fmin}}(v) + S. \tag{5}$$

Using γ, the path constraints of Corollary 6 can be written as:

$$W(u,v) \geq \left(\frac{\gamma(u,v) - T_\phi}{\frac{T_\Phi}{k} - P} \right) - 1. \tag{6}$$

The ceiling of this value is the minimum number of latches required along the critical path between vertices u and v and is referred to as $L(u,v)$. ILP constraints are now formed as

$$r(u) - r(v) \leq W(u,v) - L(u,v)$$

which may be solved using the Bellman-Ford algorithm. Mixed-ILP constraints may also be formed and the more efficient solution technique ($O(|V|^2 \log |V|)$) for them used [4]. The formulation may also be extended for unequal phase schedules as shown in [7] and solved using extensions to the Bellman-Ford algorithm in $O(k \cdot |V|^3)$ time. In summary, incorporating these additional parameters into the retiming model does not change the form of the constraints and thus the same algorithms described in the previous papers [7, 4] can still be used.

6 Making Circuits Robust to Parameter Variations

Retiming is usually cast as a way to find the minimum clock period for a circuit, but often the problem is to meet some goal clock period rather than find the minimum period. In this section,

we show that we can use extra freedom in the clock period to make a circuit robust with respect to clock skew variations.

Tolerance to parameter variation is defined as the maximum amount by which the *actual* parameter values can vary without a resulting constraint violation. Operating a particular circuit at a faster clock period naturally implies less tolerance to parameter variation. Determining a circuit's tolerance to variation in a particular parameter involves enumerating the path and cycle maximum delay constraints that depend on that parameter. The tolerance is the amount of "slack" in the most constraining of these constraints.

The advantages of identifying the most robust circuit retiming are clear; each of the parameters used in the circuit model, including critical path delay $D(u,v)$, clock skew values $\sigma_e(u)$ and $\sigma_l(v)$, latch setup S, and propagation delay P, may vary from the expected value either due to poor estimates or variations in the implemented circuit. If a timing constraint is violated in the actual circuit because a delay exceeds the modeled values used in its retiming, the circuit will fail to operate correctly.

Retiming cannot increase the tolerance in parameters involved in cycle constraints in level-clocked circuits because it cannot change the number of latches on a cycle. Thus tolerance of a cycle constraint to parameter variation is simply $\frac{w(c)T_\Phi}{k} - d(p)$. Similarly an edge-clocked circuit most tolerant of parameter error is found by retiming to the minimum cycle period and then operating at a higher speed. By increasing the number of latches on the path, however, retiming of level-clocked circuits can further increase the slack in path constraints. Moreover, as shown in Theorem 3 and Corollary 4, clock skew is not involved in cycle constraints but is included in path constraints. Thus, tolerance to variation in clock skew can be improved by retiming paths even if the tolerance to delay variation is limited by cycle delays.

6.1 An Algorithm to Generate Robust Circuits

To improve circuit tolerance to clock skew variations we add a tolerance, τ, to the effective path delay defined in Eqn. 5. Using τ, the required path weight in Eqn. 6 is written:

$$W(u,v) \quad \geq \quad \left(\frac{\gamma(u,v) + \tau - T_\phi}{\frac{T_\Phi}{k} - P} \right) - 1.$$

An increase in τ increases effective path delay as would be caused by an increase in variable clock skew, an increase in the relative skew between enabling and latching edges, or an increase in the setup time. For a given clock period, if a circuit is retimed with a value of τ, then that circuit can tolerate variations in these parameters summing to τ.

A circuit retimed with some value of τ can also tolerate increases in path and latch propagation delays. Changing either of these delay parameters, however, may also change which paths are critical. Thus a retiming using τ does not necessarily guarantee a circuit that can tolerate this much variation in element delays.

To find a circuit retiming that tolerates the most clock skew variation, we fix T_Φ to the desired clock period and search over increasing values of τ for the maximum one at which a retiming can be found. Finding the maximum value of τ is equivalent to finding the minimum clock period unless a critical cycle bound limits the minimum clock period. Further decreases of T_Φ are prevented because negative weight cycles appear in the slack graph on which a shortest path algorithm is used to identify critical paths. Thus it may be possible to increase τ further but not to further

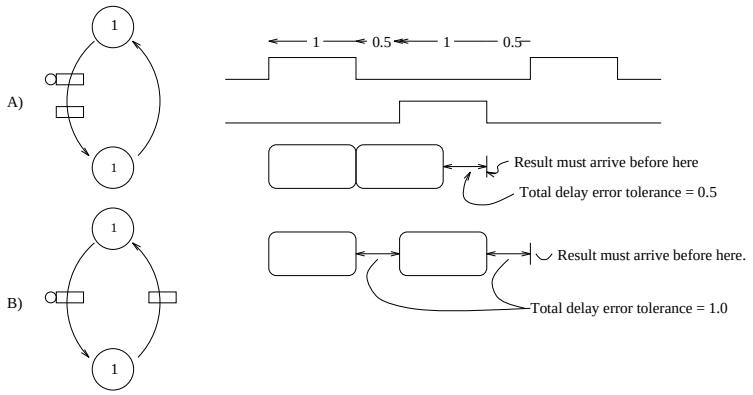

Figure 9: *A simple example showing the tolerance gain from optimizing circuit paths. Each circuit operates correctly under the given schedule. The circuit in A), however, can only tolerate a total of 0.5 units of delay estimation error in the two nodes while the one in B) can tolerate 0.5 units error in each of the nodes. (The clock period shown here is determined by some other cycle in the circuit.)*

decrease T_Φ. In other words, retiming to the fastest clock possible and then running with a slower clock does not give the most robust circuit when the cycle constraints determine the optimal clock period.

Increased circuit tolerance is illustrated in Figure 9. Using the clock schedule shown with increasing values of τ causes the initial circuit in (A) to be retimed to the one in (B). Even though the circuit in (B) cannot run faster because some other constraint defines the clock period, it can tolerate a significantly greater amount of clock skew than the one in (A). Because there are no changes in critical paths, the increased tolerance to error in vertex delay estimates is also clear.

7 Circuit Examples and Results

In this section we illustrate using retiming with clock skew to achieve better performance and improved tolerance. These examples have been generated using a retiming tool we have developed at the University of Washington. The tool is implemented as a library of routines implemented in C and has been interfaced to the SIS sequential synthesis tools as well as the xdp drawing program. The tool is capable of edge-triggered [6] and symmetric or asymmetric level-clocked circuit retiming [7, 4]. It incorporates the techniques of retiming with skew discussed in this paper and can optimize circuits for greatest skew tolerance.

The serial-parallel multiplier circuit of Figure 10 [3] provides a good example where clock skew is a factor. The vertices v_1 through v_4 are summand-adders used to perform the multiplication operation. One operand is fed in parallel across the top and a second operand fed serially on the broadcast path through the bottom vertices v_5 through v_8. The broadcast incurs significant delay which increases the circuit cycle time linearly with the width of the operands. No timing constraints are included across the host vertex v_h, since the inputs to the multiplier do not depend on the outputs.

The circuit as presented in Figure 10 can operate with a clock period of 6. Retiming this edge-clocked circuit produces a new circuit which can operate with a clock period of 3. Transforming

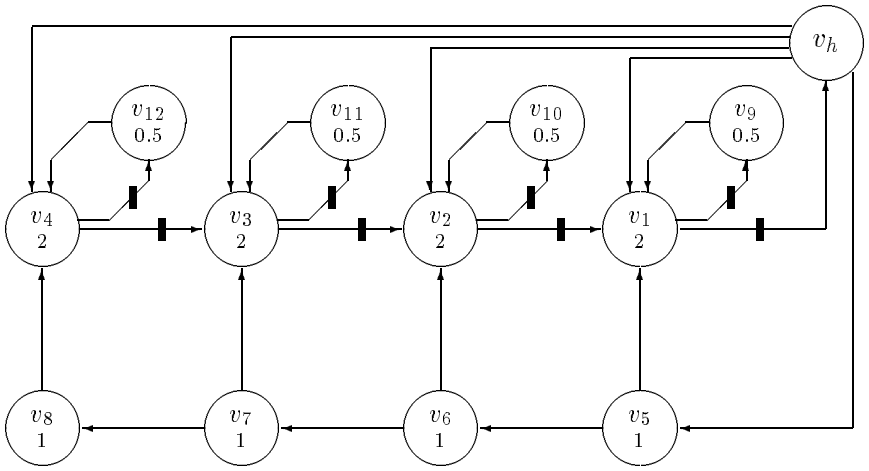

Figure 10: *A simple, edge-triggered serial/parallel multiplier circuit. No timing constraints across the host vertex are used.*

	$T_\phi = 0.5T_\Phi$		$T_\phi = 0.4T_\Phi$	
Skew mult factor	min period w/o skew	min period w/skew	min period w/o skew	min period w/skew
0.0	2.67	2.67	2.86	2.86
1.0	3.0	2.67	3.21	2.86
2.0	3.33	2.8	3.57	2.92
3.0	3.67	2.9	3.93	3.16
4.0	4.0	3.0	4.28	3.33

Table 1: *Circuit speed improvements resulting from accounting for skew when retiming the serial/parallel multiplier in Figure 11. Column 1 is an adjustment factor by which all skew values in the circuit are multiplied. Columns 2 is the clock period resulting if skew is not accounted for in retiming, while column 3 gives the period if skew is accounted for. Similarly, Columns 4 and 5 show results for an underlapped clock where* $T_\phi = 0.4T_\Phi$.

the circuit into a level-clocked circuit by converting each register into a pair of ϕ_1/ϕ_2 latches and then retiming using a symmetric clock schedule with no underlap yields the circuit in Figure 11 with a clock period of $2\frac{2}{3}$.

We next illustrate the effect of clock skew on the circuit. The fixed skew shown in the circuit corresponds to the skew as it might be caused by the clock distribution. To run our fastest level-clocked circuit with this much skew, we must increase the clock period to 3. By including the clock skew in the retiming, however, we can generate the new circuit of Figure 12 which can still run with the clock period of $2\frac{2}{3}$. Table 1 shows that including clock skew when retiming can mean a significant difference in the clock speed, especially when the skew is large with respect to the clock period.

We next show how retiming can be used to make circuits more robust to variable clock skew. Figure 13 gives a slightly modified version of the serial-multiplier circuit whose feedback delay through the nodes v_9-v_{12} has been increased to 2. This changes the most constraining timing

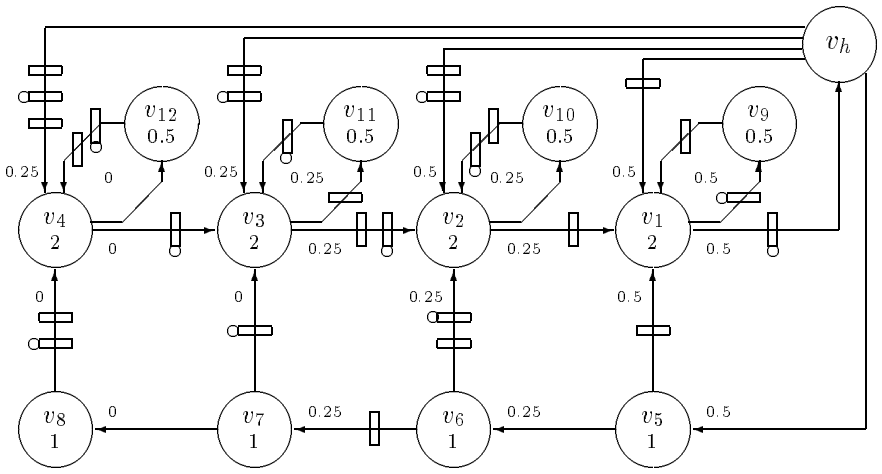

Figure 11: *The multiplier retimed without accounting for clock skew. The numbers on the edges represent the skew of both enabling and latching edges. The single number inside the small loops (i.e. $v_1 \rightarrow v_9 \rightarrow v_1$), is the skew for both edges in the loop.*

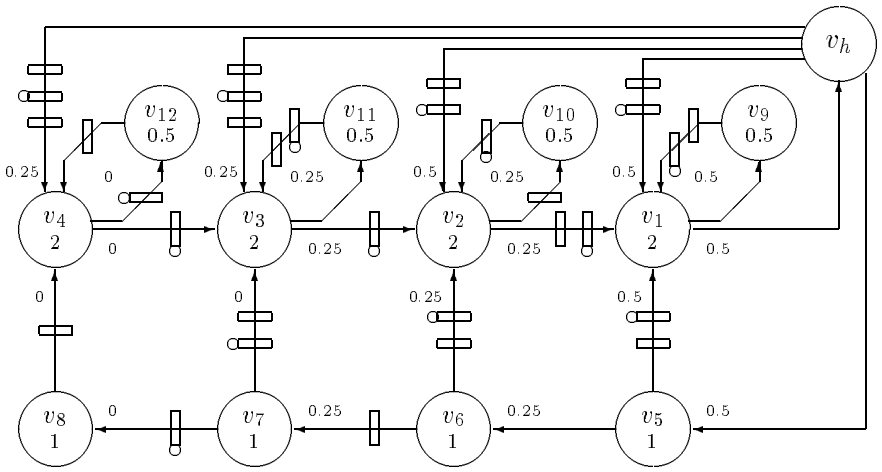

Figure 12: *The multiplier circuit retimed taking into account the clock skew.*

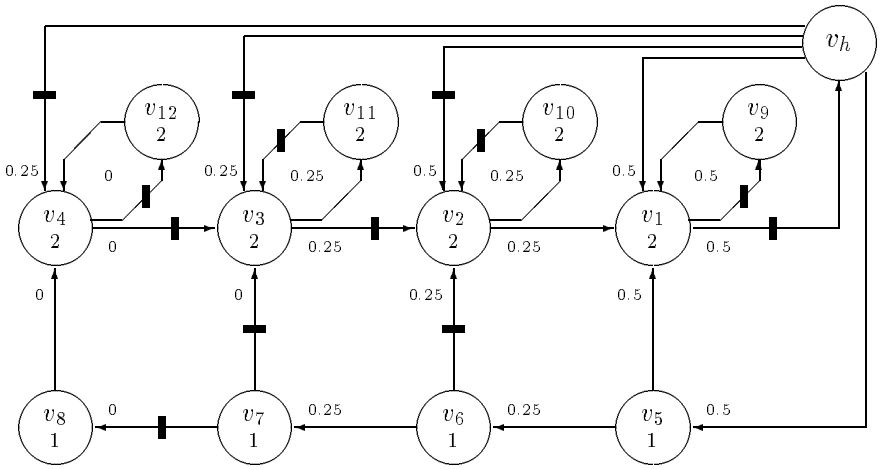

Figure 13: *The modified serial/parallel multiplier with $d(v_9)$ through $d(v_1 2)$ increased to 2. This is the optimal edge-triggered retiming with a period of $T_\Phi = 4$.*

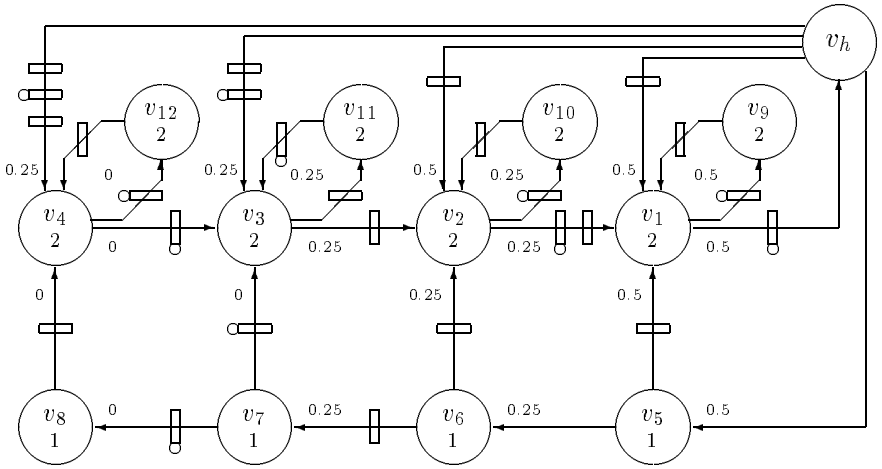

Figure 14: The modified multiplier retimed for tolerance to increased clock skew. Although now the circuit is no faster than the edge-triggered one due to the critical cycle limits, it now tolerates variable skew up to 0.875 (less skew is tolerated at the latches adjacent to vertices v_7 and v_8 due to minimum delay constraints).

constraint from a path constraint to a cycle constraint. Thus the fastest either the edge-triggered or level-clocked circuit can be retimed to is a clock period of 4. This is the kind of circuit often used as an example why edge-clocked circuits can do as well as level-clocked circuits. However, the edge-clocked circuit can tolerate no skew whatsoever and so some margin for skew must be left in the clock. By contrast, the level-clocked circuit in this example, when retimed to maximize tolerance to clock skew, can accommodate variable clock skew up to 0.875! That is, the level-clocked circuit requires no margin for clock skew and is thus substantially faster in practice than the edge-clocked circuit. Note also that the retiming accounts for fixed skew while maximizing variable skew tolerance. The final circuit is shown in Figure 14.

8 Concluding Remarks

We believe that retiming will become an increasingly important tool for the synthesis of high-performance circuits, especially in the area of level-clocked circuits. Retiming can have a large impact on the synthesis process if it is applied sufficiently early in the design cycle as we illustrated in the design of a simple RISC processor.

Retiming will also become more important as a means to control the effect of clock skew, which will become more of a factor in high-performance circuits. Edge-triggered circuits have no built-in tolerance for clock skew and thus the clock must be slowed down to create a margin for skew. Level-clocked circuits on the other hand, can often be retimed to include built-in skew tolerance without slowing down the clock as illustrated in the serial-parallel multiplier example. Level-clocked circuits are thus inherently faster than edge-clocked circuits and retiming can be used to exploit this advantage.

References

[1] S. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1991. Caltech-CS-TR-91-01.

[2] D. W. Dobberpuhl, R. T. Witek, R. Allmon, et al. A 200-MHz 64-b dual-issue CMOS microprocessor. *IEEE Journal of Solid-State Circuits*, 27(11):1555–1567, Nov. 1992.

[3] G. Even. A real-time systolic integer multiplier. Technical Report 763, Technion- Israel Institute of Technology, Jan. 1993.

[4] A. T. Ishii, C. E. Leiserson, and M. C. Papaefthymiou. Optimizing two-phase, level-clocked circuitry. In *Advanced Research in VLSI and Parallel Systems: Proc. of the Brown/MIT Conference*, pages 245–264, 1992.

[5] C. E. Leiserson, F. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. In *Proc. of the 3rd Caltech Conference on VLSI*, Mar. 1983.

[6] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991. Also available as MIT/LCS/TM-372.

[7] B. E. Lockyear and C. Ebeling. Retiming of multi-phase, level-clocked circuits. In *Advanced Research in VLSI and Parallel Systems: Proc. of the Brown/MIT Conference*, pages 265–280, Mar. 1992.

[8] M. C. Papaefthymiou. Edge-triggering vs. two-phase level-clocking. In *Research on Integrated Systems: Proc. of the 1993 Symposium*, pages 201–218, Mar. 1993.

[9] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous circuits. In *Proc. 27th ACM-IEEE Design Automation Conf.*, 1990.

[10] H. Touati, N. Shenoy, and A. Sangiovanni-Vincentelli. Retiming for table-lookup field-programmable gate arrays. In *First Intl. ACM/SIGDA Workshop on FPGAs*, pages 89–94, California, 1992. Berkeley.