Multiphase Retiming Using $minT_c$ [1]

Timothy M. Burks [2] **Karem A. Sakallah** **Trevor N. Mudge**

Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109

Abstract

This paper describes a general framework for the retiming of sequential circuits using either single or multi-phase clocks and both edge-triggered flip-flops and level-sensitive latches. This framework is compared with previously published work on retiming. The potential for finding optimal retimings within this framework is discussed, and a set of heuristic retiming methods is presented. Finally, examples of both single and multi-phase retimings of circuits with level-sensitive latches are presented.

1 Introduction

Retiming, first described by Leiserson [1], is a powerful method for the timing optimization of synchronous circuits. Retiming algorithms allow storage elements to be moved back and forth across logic in a circuit to minimize the cycle time or number of storage elements in the circuit. In [1], Leiserson described the original retiming model which he developed for optimizing single-phase, edge-triggered circuits and presented a set of efficient, well-characterized algorithms for finding optimal retimings with respect to both cycle time and register count. In Leiserson's model, circuits are partitioned into logic blocks between which registers may optionally be placed (see Figure 1). Logic blocks are represented by nodes in a directed graph and the arcs connecting these nodes are labeled with the number of registers (w_{ij}) between the corresponding logic blocks. A retiming variable $r(v_i)$ is defined which relates changes in the latency of the calculation performed at node v_i to the number of registers on the arcs coming into and out of the node. These retiming variables are then used as the independent variables in an optimization which seeks to minimize the circuit cycle time or the number of registers in the circuit. A key characteristic of these retiming variables is that their use in optimization guarantees that the number of registers around each loop in a circuit is constant, a condition which Leiserson shows to guarantee that circuit functionality is unchanged.

However, a number of restrictions were made in Leiserson's retiming methods to make the algorithms more tractable. These restrictions fall into two broad categories:

1. Structural restrictions: In Leiserson's methods, the gate-level structure of a circuit is fixed and storage elements are simply moved back and forth across gates in the circuit. As a result, the traditional retiming algorithms can miss many optimizations which can be performed when logic transformations are considered.
2. Clocking restrictions: Leiserson assumes that a single-phase clock is used and that the storage elements are edge-triggered flip-flops. As such, his algorithms do not directly apply to multiphase circuits or circuits using level-sensitive latches.

---

1. This work was supported in part by NSF grant MIP-9014058.
2. T. Burks is supported by a DoD NDSEG fellowship.

Clock Constraints	
pulse width constraints	$T_p \geq w$
	$T_c - T_p \geq w$
Combinational Propagation Constraints	
signal propagation into synchronizer i	$a_i = \min_{j=1,n}(d_j + \delta_j + \delta_{ji} - E_{p_i p_j})$
	$A_i = \max_{j=1,n}(D_j + \Delta_j + \Delta_{ji} - E_{p_i p_j})$
Synchronizer Macromodels	
setup and hold constraints	$A_i \leq T_c - S_i$
	$a_i \geq H_i$
flip-flop synchronization equations	$d_i = T_c$
	$D_i = T_c$
latch synchronization equations	$d_i = \max(a_i, T_c - T_{p_i})$
	$D_i = \max(A_i, T_c - T_{p_i})$
wire synchronization equations	$d_i = a_i$
	$D_i = A_i$
Phase Shift Operator	
E[i][j]	$e_j - e_i$, if $e_j > e_i$ $T_c - (e_j - e_i)$, if $e_j \leq e_i$

Table 1: Timing Model Summary

$\delta_{w_i 0}$ is the Kronecker delta function which is equal to 1 whenever $w_i = 0$ and zero otherwise. Combining these with the combinational propagation equations gives

$$a_i = \min_{j=1,n}(\delta_{w_j 0} a_j + \delta_j + \delta_{ji})$$

$$A_i = \max_{j=1,n}(\delta_{w_j 0} A_j + \Delta_j + \Delta_{ji})$$

To ensure correct circuit operation, we must have

$$A_i \leq T_c - S_i$$

$$a_i \geq H_i$$

whenever $w_i > 0$. In Leiserson's model, the hold time constraints were neglected and synchronizer delays and setup times were assumed zero, leaving the following set of constraints to be satisfied:

$$\forall i, A_i = \max_{j=1,n}(\delta_{w_j 0} A_j + \Delta_{ji})$$

$$\forall i, A_i(1 - \delta_{w_i 0}) \leq T_c$$

To determine whether a retimed circuit can be run at a specified cycle time T_c, it is necessary only to compute the values of A_i using the first constraint above and then verify that the second constraint is satisfied. If the solution fails to satisfy the second set of constraints, then the critical paths that caused the setup violation can be shortened by retiming the last node in each path by +1. More aggressively, we can retime each node in the circuit for which $A_i > T_c$. This is equivalent to one pass through Algorithm FEAS as described in [1]. The A_i variables correspond to the values $\Delta(i)$ computed by Leiserson's Algorithm CP. If after iterating this procedure $|V| - 1$ times ($|V|$ is the number of combinational nodes in the circuit), no feasible solution can be found, then no retiming exists and the FEAS algorithm terminates. Otherwise, the feasible retiming will be found.

3 Multiphase Retiming Issues

It is important to show that any timing optimization procedure does not alter the behavior of the circuit being optimized. To guarantee that a retiming would not change the function of a circuit, Leiserson defined the retiming variables with the following set of equations:

$$w_r(i,j) = w_i(i,j) + r(j) - r(j)$$

where $w_i(i,j)$ is the initial number of storage elements between nodes i and j and $w_r(i,j)$ is the number of storage elements between them in the retimed circuit. Leiserson showed that as long as $w_r(i,j) \geq 0$, changes to the r variables would not affect a circuit's operation. The essence of the argument was that the total number of registers around every loop in a circuit must remain constant. With the addition of an extra vertex to represent the environment, enforcement of this condition guarantees equivalent circuit function.

For multiphase retiming, we have a slightly more difficult problem. Instead of simply requiring the number of storage elements around a loop to be constant, we need to require that the latency around loops be constant, where we define latency as the number of clock cycles by which signals moving around the loops are delayed. Although the latency could be preserved under many less-restrictive constraints, for now we preserve it by requiring that the number of latches around each loop be kept constant and that the ordering of phase assignments around each loop is also preserved. Note that this is subject to the assumptions about latch operation embodied in Table 1.

Figure 2 illustrates a possible latency-related problem in multiphase retiming. When the node shown is retimed, there is no way to assign a phase to the new synchronizer without changing the latency through at least one of the branches going into the node. To retime this node, we must determine the appropriate clock phase to control the new synchronizer. Fortunately, for a significant class of retiming applications, it can be shown that this problem will not arise. A potentially powerful application of our multiphase retiming procedure is the conversion of single-phase edge-triggered circuit design into implementations clocked with two-phase level-sensitive latches. Due to their simplicity, single-phase edge-triggered circuits are much easier to design and be understood by human designers. However, level-sensitive latches require significantly less area when implemented, and in fact, edge-triggered synchronizers are often implemented with a pair of level-sensitive latches connected in a master-slave configuration. Also, when level-sensitive latches are

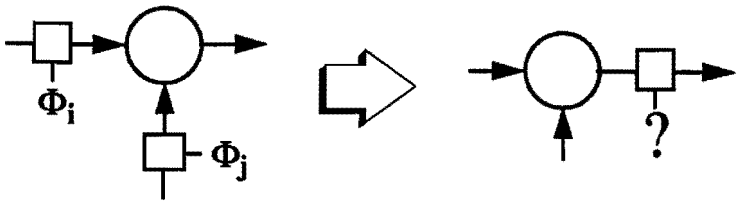

Figure 2 Phase Assignment Conflict in Multiphase Retiming

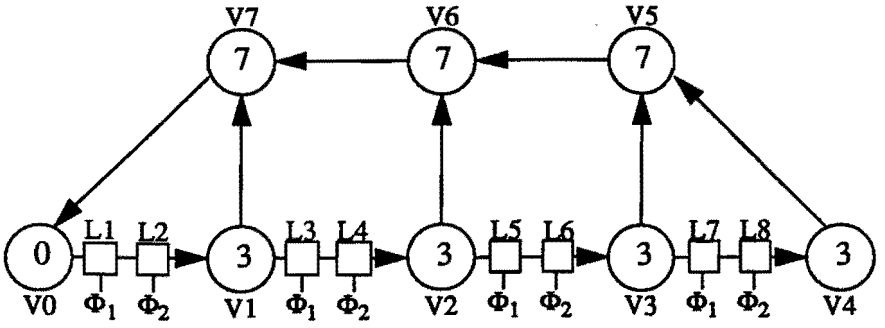

Figure 3 Replacing 1-Phase Flip-Flops with 2-Phase Latch Pairs

used, time can be shared using the flow-through properties of a latch to average delays and reduce the cycle time. For these reasons, most VLSI circuits are implemented with level-sensitive latches as storage elements. These observations suggest a design procedure which begins with a single-phase edge-triggered circuit, replaces all of the edge-triggered devices with pairs of level-sensitive latches (compare Figure 1 and Figure 3), and then retimes these latches to reduce the cycle time below the optimum achievable with edge-triggered devices. An example of such a retiming is presented in Section 6.

Two-phase circuits created in this way are initially free of phase assignment conflicts; and since each retiming step must pull the same number of synchronizers through each input (or output) of the retimed nodes, no phase assignment conflicts can ever be introduced.

4 A Procedure for Multiphase Retiming

We are currently developing a new bottom-up heuristic retiming approach, and can now suggest a method for the retiming of multiphase circuits clocked by either edge-triggered or level-sensitive devices. Any retiming algorithm for timing optimization must have at least two parts:

1.A means for determining the critical circuit delays.

2.A method for modifying these delays by redistributing them throughout the circuit.

For example, in the second of Leiserson's retiming algorithms (algorithm FEAS)[1], the CP algorithm is used to determine the minimum cycle time and identify critical paths in the circuit. To reduce these critical delays, Leiserson retimes nodes at the end of each critical path by +1, effectively pulling a register from the end of the path forward, breaking the long path. In this section, we discuss the approach we take to each part of the retiming problem and in the following sections, present results obtained by applying our methods to two specific examples.

4.1 Identifying Critical Delays

We currently have two methods for determining the critical paths in a circuit. The dual solution of a linear program provides the sensitivity of the objective function (T_c) with respect to each constraint [8]. For those constraints which correspond to combinational propagation delays, these sensitivities are the local partial derivatives of the objective function with respect to the corresponding propagation delays. From this we observe that the constraints having nonzero dual values identify critical paths in a circuit.

The non-zero dual variables identify a single set of constraints that is sufficient to bound the cycle time; however, they will not be able to detect the presence of parallel critical paths, each of which is sufficiently long to hold the cycle time to its current minimum. When such parallel critical paths exist, shortening the one identified by the dual variables is not sufficient to reduce cycle time. Shortening this path will only cause another of these *co-critical* paths to hold the cycle time at its existing optimum. This path, however, can be identified by the new set of dual variables. This

suggests that we can overcome the problem of multiple parallel critical paths by simply shortening each path as it is identified until there are no more paths of that length and the cycle time is reduced.

However, the slack variables produced in the LP solution also contain critical path information. We can identify critical paths by scanning the list of slack variables and marking as critical any path having zero slack. Also, if we do not use linear programming to find the optimum cycle time, we can still look at the slack variables generated by a timing verification procedure such as $checkT_c$ [7].

4.2 Redistributing Critical Delays

Once we have identified which paths are critical, we need some procedure for modifying them to make them less critical. Delays should be retimed out of critical long paths to more easily allow setup constraints to be satisfied. Critical short paths can be lengthened to reduce the effect of hold constraints. If level-sensitive latches are used, a third type of critical path can exist which cannot be removed by retiming. These *critical loops* are cycles in the circuit structure in which the late-arriving signal flows through every latch in the path. The only way to shorten such a loop is to actually remove logic from the loop (which retiming cannot do) or change the delay of gates in the loop. In our current work, we have focused only on shortening critical long paths. Two simple methods are available for doing so: to pull a storage element forward from the end of the path (retime the last node in the path by +1), or push a storage element backward from the beginning of the path (retime the first node in the path by -1). Both operations act to shorten the critical path and can be seen as inverses of one another.

Depending on the nature of the search space, it may be necessary to use both methods to find the truly optimal retimings; but unfortunately, they can also lead to back-and-forth oscillations when used together. It is much simpler to do as Leiserson did in the FEAS algorithm and consistently move latches in the same direction; while this does not eliminate the possibility of oscillations, it does focus the search and allow the algorithm to run in polynomial time. For general circuits, however, it is an open question whether this method will produce polynomial-time solutions.

5 Single-Phase Retiming

To see an example of our retiming techniques applied to level-sensitive latches, we applied them to Leiserson's correlator circuit (Figure 1) with all edge-triggered devices replaced by latches. For this example, we assumed that the minimum delays through each block were equal to the maximum delays, and that the setup and hold times for all latches were zero. We used the dual and slack variables to find critical delay paths and the delay modification heuristic we used was to retime the beginning of each critical path by -1. For the initial circuit, $minT_c$ obtained a minimum cycle time of 21, and the dual solution identified the path from latch 4 to latch 1 as critical. Examining the slack variables showed that the path from latch 3 to latch 1 was also critical. As a result, vertices v_3 and v_4 were retimed to produce the next circuit shown in Figure 4 The remaining steps in the retiming process are also shown in the figure, including the final retimed circuit, which contains four latches and has a cycle time of 10. Note that this is the minimum cycle time obtainable for this circuit by *any* method that does not modify the delays in the logic blocks or change circuit latency: this minimum time can be attributed to any of the three critical loops in the circuit: V_1-V_7-V_0, V_1-V_2-V_6-V_7-V_0, and V_1-V_2-V_3-V_5-V_6-V_7-V_0.

Note that at step 4 in the optimization process we saw a temporary increase in the cycle time. As a result, we cannot rely on simple observations of the cycle time to know when to stop retiming. In [1], Leiserson showed that a similar procedure to this would always find a feasible retiming within a well-defined number of steps, if one existed. However, this proof was for an algorithm for finding a retiming for a predetermined cycle time. Since we do not specify the cycle

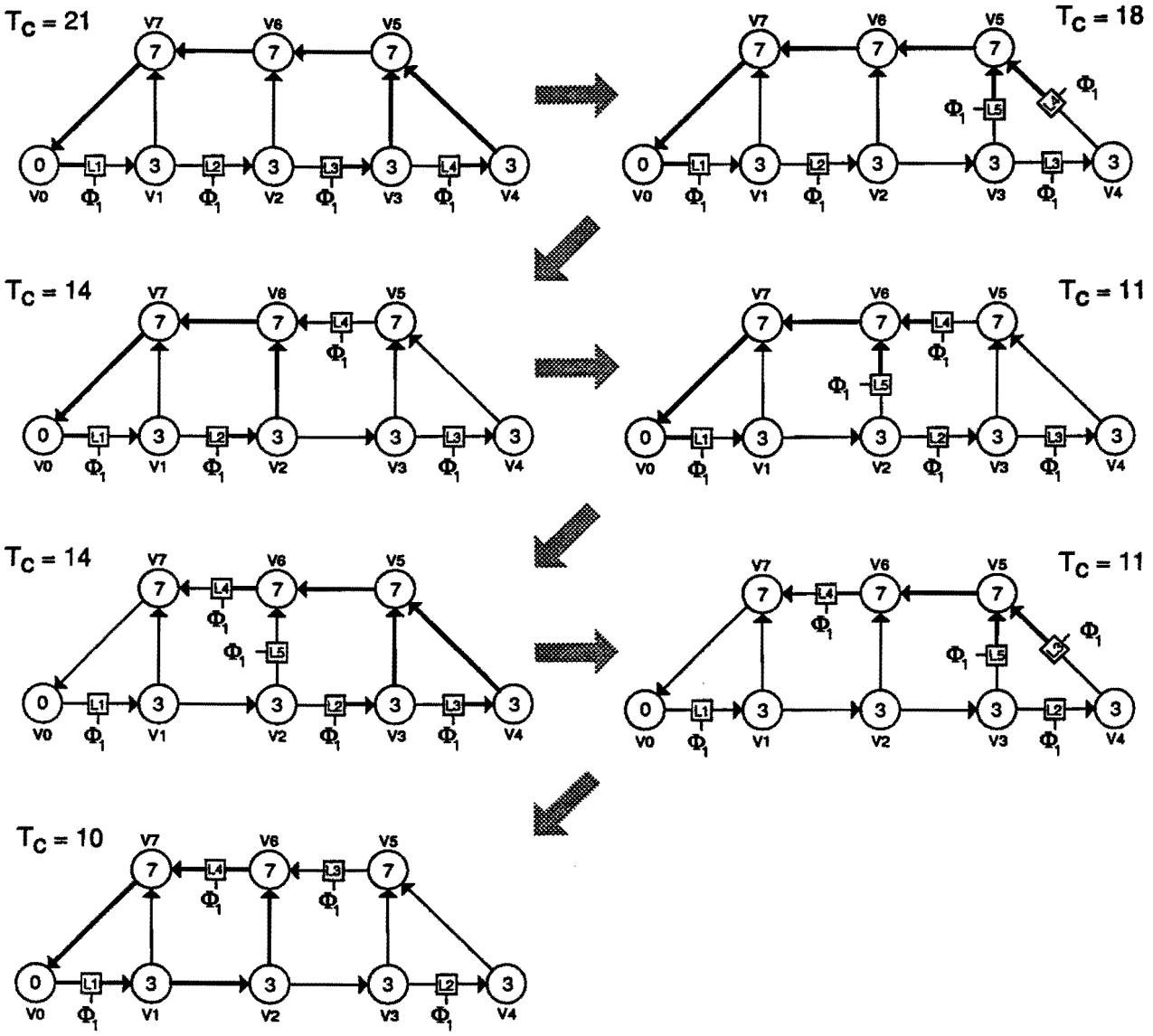

Figure 4 Single-Phase Latch Retiming: Optimization Steps

time for our retimings, we cannot directly use his result. Instead, we conjecture that a similar statement may be provable for our retimings: that the optimal retiming can be achieved after a bounded number of steps. For the current retiming algorithm, we have three possible stopping schemes: to stop retiming when (1) the minimum achievable cycle time (due to a critical loop) has been achieved, (2) the retiming has led to a previously-examined circuit (and further iterations will simply continue in cycles), or (3) the retiming has proceeded for a significant number of iterations without improvement.

6 Two-Phase Retiming

In the previous example, we assumed that the minimum and maximum delays between latches were equal. For level-sensitive latches, this is the most optimistic assumption possible, as it provides the most insulation against hold violations and double clocking. On the other hand, the most conservative assumption would be to assume that the minimum delays between latches were zero.

There are a number of ways to clock circuits with zero (or unpredictable) minimum delays. One simple solution is to use edge-triggered devices controlled by a single clock. Another solution

is to use single-phase level-sensitive latches and attempt to pad all of the minimum delays sufficiently to ensure that no signal can race through and cause hold violations. Yet another approach is to replace the edge-triggered synchronizers with pairs of level-sensitive latches controlled by two different clock phases, as described in Section 3. If we do this, then it should be possible to retime logic into the space between latch pairs and reduce the cycle time below what is achievable using edge-triggered synchronizers. To demonstrate this, we chose to again look at Leiserson's correlator circuit, only this time assuming that the minimum delay between latches is zero. The steps in the retiming are shown in Figure 5. To see an example of a more sophisticated critical path than those which occurred in the one-phase solution, consider the state of the circuit at the next-to-last step in the retiming process. $minT_c$ finds the cycle time of this circuit to be 13.5 and the critical path extends from latch 3 to latch 8 along the path through latches 6 and 7. The critical path extends through latches 6 and 7 because the critical late signal passes directly through both latches without being held up. This path can be shortened by retiming vertex v_2 to produce the final retimed circuit. This circuit has a cycle time of 10, and since we know that this is the minimum cycle time achievable for this circuit, we stop.

7 A Bounded Retiming Algorithm

We are currently working to develop retiming algorithms that have well-established stopping criteria. Leiserson demonstrated that his FEAS algorithm would stop after a finite number of steps (|V| - 1), by showing that each iteration of the algorithm was equivalent to a pass of the Bellman-Ford algorithm for constraint satisfaction, which was itself guaranteed to find a solution (if one existed) after at most (|V| -1) iterations. Leiserson's optimization procedures worked by first calculating a list of all possible minimum cycle times for a circuit and then performing a binary search through the list in which he tested at each step whether a retiming existed to allow the circuit to run at the specified cycle time. The FEAS algorithm was one such algorithm for determining whether or not such a retiming existed.

For circuits clocked with level-sensitive latches, it is not so straightforward a matter to make an exhaustive list of the possible minimum cycle times of a circuit. Instead, we take an iterative approach that can be converges to the exact solution.

The retiming algorithm which was demonstrated in the previous sections was an iterative one: it repeatedly retimed critical long paths in a circuit in an attempt to reduce the circuit cycle time. The algorithm as demonstrated retimed only the last node in each critical path during each of its iterations. Ignoring the presence of zero-delay nodes and applied to edge-triggered circuits, each such iteration is equivalent to one pass of Leiserson's FEAS algorithm where the target cycle time is just less than $T_c^{(i)}$, the minimum cycle time of the retimed circuit in the i-th iteration.

If we make a small modification to the demonstrated procedure, we can guarantee that our new algorithm can be stopped |V| - 1 iterations after the optimum retiming has been found, where |V| is the number of combinational nodes in the retiming graph. The proposed algorithm is listed in Figure 6. Each iteration of the algorithm is a equivalent to a pass of the FEAS algorithm with a target cycle time just below the best cycle time found thus far. If no reduction in cycle time is seen after |V| - 1 iterations, then we know that we can stop, because we have just completed |V| - 1 iterations of the FEAS algorithm at the current target cycle time. We know that this algorithm will eventually find an optimal retiming because as soon as a feasible retiming is found for the current target cycle time, it starts again with a new (lower) target time. There are two differences between this algorithm and the one used in the previous two sections. The first is that it is continually trying to retime the current circuit to go faster than the best cycle time seen thus far, whereas the original algorithm only tried to beat the cycle time of the circuit in its current state. The second difference is that the previously demonstrated procedure only retimed nodes at the ends of critical paths; the

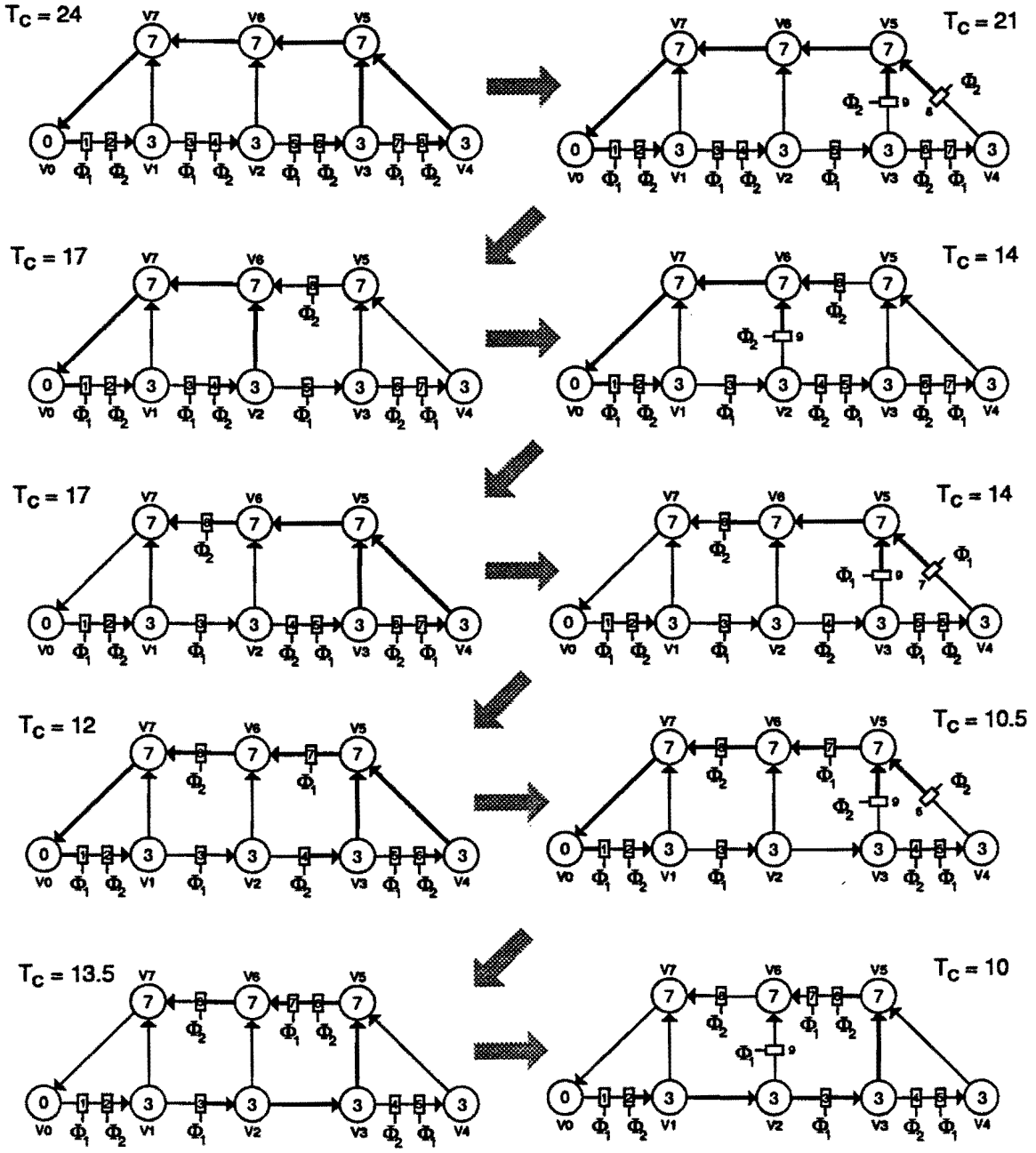

Figure 5 Two-Phase Latch Retiming: Optimization Steps

proposed algorithm retimes all nodes whose outputs would cause a setup violation at the input of a subsequent storage element. The above-derived bound has currently only been proven for circuits clocked with edge-triggered flip-flops; however, we are currently working to prove that it also applies for level-sensitive latches.

8 Conclusions

This paper has presented a general framework for the retiming of sequential circuits controlled by both edge-triggered and level- sensitive devices under complex clocking schemes. We have discussed and demonstrated a method for retiming that is successful for both single- and two-phase circuits with level-sensitive latches. We have also proposed a method with a well-defined

```
bestTc = minTc (current-circuit)
best-circuit = current-circuit
countdown = |V| - 1
while (countdown ≥ 0) do
  for all nodes with arrival times ≥ bestTc - setuptime
    mark node to be retimed
  end for
  move latches forward past all marked nodes (retime the current-circuit)
  if minTc (current-circuit) < bestTc
    bestTc = minTc (current-circuit)
    best-circuit = current-circuit
    countdown = |V| - 1
  else
    countdown = countdown - 1
  end if
end while
```

Figure 6 Proposed Retiming Algorithm

stopping criterion. We are currently working to fully automate these methods within the *minT$_c$* software framework.

References

[1] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.

[2] G. De Micheli, "Synchronous logic synthesis: Algorithms for cycle time minimization," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 1, pp. 63–73, January 1991.

[3] S. Malik, E. Sentovich, and R. Brayton, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 1, pp. 74–84, January 1991.

[4] K. Bartlett, G. Borriello, and S. Raju, "Timing optimization of multiphase sequential logic," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 1, pp. 51–62, January 1991.

[5] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming of circuits with single phase transparent latches," in *Proceedings of the International Workshop on Logic Synthesis*, Research Triangle Park, NC, May 1991.

[6] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits", in *Proceedings of the 27h Design Automation Conference*, 1990.

[7] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "*checkT$_c$* and *minT$_c$*: Timing verification and optimal clocking of synchronous digital circuits", in *ICCAD-90 Digest of Technical Papers*, November 1990.

[8] M. J. Best and K. Ritter, *Linear Programming: Active Set Analysis and Computer Programs*, Prentice-Hall, 1985.