

Identification of Critical Paths in Circuits with Level-Sensitive Latches

Timothy M. Burks, Karem A. Sakallah, and Trevor N. Mudge

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109-2122

April 22, 1993

Abstract

This report extends the classical notion of critical paths in combinational circuits to the case of synchronous circuits that use level-sensitive latches. Critical paths in such circuits arise from setup, hold, and loop constraints on the data signals at the inputs of each latch and may extend through one or more latches. Two approaches are presented for finding and verifying these critical paths. The first implicitly checks all paths using a relaxation-based verification procedure. Results of this procedure are used to calculate slack values, which in turn identify satisfied and violated critical paths. The second approach is based on a constructive algorithm which generates all of the critical paths in a circuit and then verifies that their timing constraints are satisfied. Algorithms are evaluated and compared using the ISCAS89 sequential benchmark suite.

1 Motivation

The timing verification of latch-controlled circuits has been a popular subject for recent study. The timing verification problem is to determine whether a sequential circuit with a specified structure will operate correctly under a desired clock schedule. To do this successfully, it is necessary to have accurate models for the timing behavior of each component in the circuit. To verify that a circuit will run correctly under a given clock schedule, all possible paths through the circuit must be analyzed to guarantee that at no point will the interacting clock and data signals produce undesired behavior.

However, few designers are satisfied with a simple determination of whether or not their circuit will meet its timing constraints. If the design fails to meet its requirements, it may be useful to know how close it came to its desired performance. This falls into the domain of *optimal clocking* methods [12, 13], which find the minimum cycle time for a circuit subject to a set of possible constraints on the clock schedule. However, this too is often not enough as it still provides no information about why a circuit works or fails to work. To determine this, it is necessary to know the *critical paths* in a circuit, those sections of the circuit which place the strongest constraints on its timing behavior. The concept of a critical path has been used for many years to analyze both combinational and sequential circuits. However, for synchronous sequential circuits, classical approaches rely on the assumption that storage elements in a circuit are edge-triggered, which produces very simple timing constraints. If instead level-sensitive latches are used, the timing constraints become more complicated, since the latches allow signals to flow directly through them when they are enabled, and essentially act like sections of combinational logic. As more and more designs use level-sensitive latches, designers are forced to make simplifying assumptions that overconstrain circuit behavior in order to use existing critical-path based timing techniques.

This paper extends the classical definition of a critical path in a way that correctly captures the full range of timing behaviors which can appear in circuits clocked with level-sensitive latches. We show that three distinct types of critical paths can arise and contrast this with the single type of critical path commonly observed in edge-triggered circuits. Each type of path is described in detail, and models are presented that relate them back to classical critical path methods. We then discuss procedures for extracting these paths from verified circuits and discuss factors which affect the feasibility of extracting and enumerating these paths. We present two alternate approaches, one based on existing relaxation-based verification methods and the other a new approach which iteratively constructs possible critical paths, and include an analysis of their algorithmic complexities and a comprehensive strategy for their use.

An overview of critical path methods and their application to digital circuits is presented in Section 2. Section 3 contains the timing models that describe circuits with level sensitive latches. Each of the three types of critical paths is presented and analyzed in Sections 4 through 6. Section 7 presents the relaxation-based approach and describes the extraction of critical paths from the relaxation results. Section 8 describes the algorithm for constructing the set of critical paths, describes its asymptotic complexity, and discusses extensions which verify startup behavior and qualified clocks. It also contains a comparison of the proposed path extension algorithm with existing relaxation techniques. Section 9 describes the results of experiments performed using the various verification and path identification procedures.

2 Critical Path Analysis

The concept of a critical path was originally introduced in the field of project management [17]. Its application to digital circuits was first described by Kirkpatrick and Clark in 1966 [1] and was later elaborated by Hitchcock, Smith, and Cheng in 1982 [2]. In this section, we present the classical project management definition of a critical path and discuss its application to combinational logic circuits. We then discuss the extension of critical path analysis to synchronous sequential circuits and highlight the complexities that arise when these circuits are controlled with level-sensitive latches.

2.1 Critical Paths in Project Networks

Traditional approaches to critical path analysis fall into two separate but related methods, PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method). Both methods were developed in the 1950's as tools

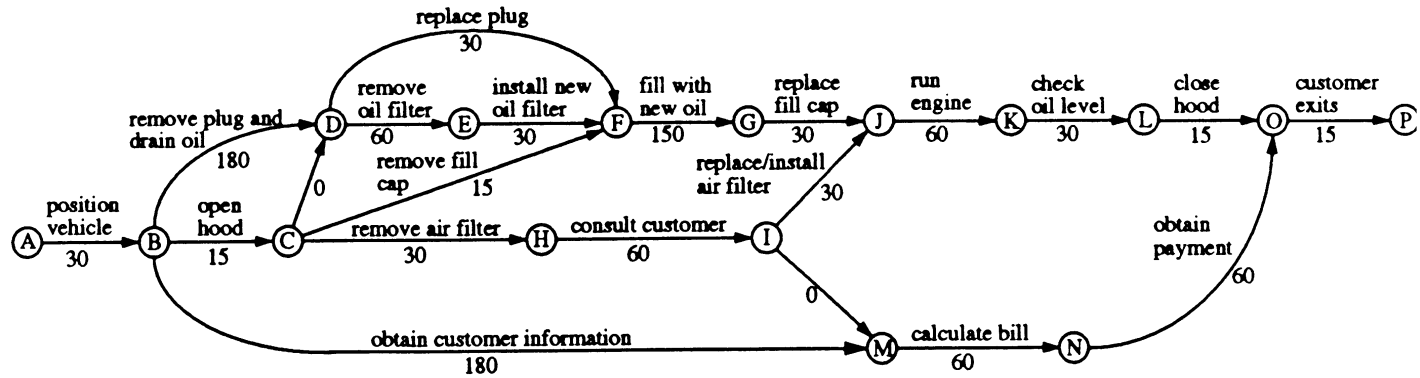


Figure 1: Sample CPM Project Network (Arrow Diagram)

for the planning of complicated design, construction, and maintenance projects. Projects are described with acyclic networks called *arrow diagrams*, where each arrow represents a job or task that must be performed to successfully complete the project. Each arrow is labeled with the expected time needed to complete the corresponding task. In CPM, these times are represented with scalar worst-case values; in PERT, job times are described by a simple probability distribution. Vertices, or nodes, in the diagrams represent states in the project and, when joined with arrows, model the dependencies among project tasks. Two special nodes, called the *source* and *sink*, are commonly used to represent the beginning and end of a project. Without loss of generality, the source and sink are required to be the only nodes in the network with no incoming and no outgoing nodes, respectively.

A sample project network is shown in Figure 1. The network corresponds to a set of tasks that might be performed at a 10-minute oil change shop. The time, in seconds, associated with each task is marked on the corresponding arrow. In Figure 1, node A is the source; node P is the sink. The two arrows marked with zero times ($C \rightarrow D$ and $I \rightarrow M$) are *dummy tasks* [17, p. 17] that are necessary to avoid creating artificial dependencies in the project network.¹

The minimum time to complete a project is determined by the path through the project network (from source to sink) that requires the longest total time. The longest path in the network of Figure 1 is:

$$A \rightarrow B \rightarrow D \rightarrow E \rightarrow F \rightarrow G \rightarrow J \rightarrow K \rightarrow L \rightarrow O \rightarrow P$$

Its corresponding total time is:

$$30 + 180 + 60 + 30 + 150 + 30 + 60 + 30 + 15 + 15 = 600 \text{ seconds} = 10 \text{ minutes}$$

2.1.1 Actual and Required Event Times

This minimum project completion time can be obtained by making a single pass through the network, processing each node exactly once. The nodes are sorted topologically starting with the source node, and then an *actual event time* $e(v)$ for each node v is calculated. This is the earliest possible completion time for the entire set of jobs with arrows terminating at node v . It is computed by taking the maximum over all immediately preceding nodes of the sum of the node event time and the job time marked on the connecting arrow:

$$e(v) = \max_{u \in P(v)} [e(u) + t_{u,v}] \quad (1)$$

In this equation, $e(u)$ and $e(v)$ are the event times for nodes u and v , $t_{u,v}$ is the time of the job $u \rightarrow v$, and $P(v)$ is the set of node predecessors to node v (see Figure 2). Setting the event time for the source node to zero, (1) defines a unique event time for the remaining nodes. The minimum time needed to complete the entire project is then simply the event time of the sink node.

1. For example, if arrow $C \rightarrow D$ were not present, nodes C and D would be a single node, and it would appear necessary to drain the oil before the air filter could be removed! Fortunately, such complications do not arise in the circuit networks addressed in this paper, and will not be discussed further.

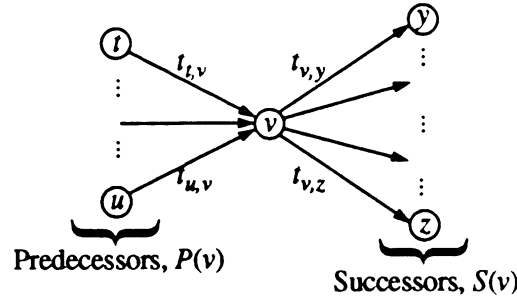


Figure 2: Arrow Diagram Structure

A *critical path* in the project network is a sequence of jobs that connect the source and sink nodes and whose job times determine the project completion time. Critical paths can be identified by making another pass through the network to compute *required event times*, $r(v)$. Required times are defined as:

$$r(v) = \min_{u \in S(v)} [r(u) - t_{u,v}] \quad (2)$$

The required time for the sink node is set to T_r , the *required time* for project completion. Required times can be easily calculated by visiting the nodes in reverse topological order. In (2), $r(u)$ and $r(v)$ are the required times for nodes u and v and $S(v)$ is the set of node successors to node v (see Figure 2).

The actual and required event times, $e(v)$ and $r(v)$, together define a range of times for each event. The actual event time $e(v)$ is the earliest time at which event v can occur if no job times are reduced. Given a required project completion time, T_r , the required time $r(v)$ is the latest time for event v that will not cause the project completion time to exceed T_r . The difference between the required and actual event times¹ at a node is defined as the *slack* of the node:

$$s(v) = r(v) - e(v) \quad (3)$$

A similar quantity, *float*, is defined for each job in the network, and is given by:

$$f(u \rightarrow v) = r(v) - e(u) - t_{u,v} \quad (4)$$

where $u \rightarrow v$ denotes the job between nodes u and v . The float represents the amount by which the time of job $u \rightarrow v$ can be increased without causing the project completion time to exceed the required project completion time T_r .

Critical events and *critical jobs* can be identified as the events and jobs having the smallest slacks and floats, respectively, in the network. A *critical path* is then a sequence of critical events and jobs that connect the source and sink nodes. Note that the slacks and floats of critical events and jobs can be positive, negative, or zero, according to whether $T_r - e(v_{\text{sink}})$ is positive, negative, or zero. Also note that when the critical slacks and floats are negative, the desired project completion time is infeasible unless job times are reduced until all floats are zero or positive.

Actual and required event times and event slacks for the oil change project are shown in Table 1 for two desired completion times: $T_r = 10$ minutes (600 seconds) and $T_r = 8$ minutes (480 seconds).

2.1.2 Maximum Delay from Source and Maximum Delay to Sink

The actual and required event times can also be expressed in terms of the maximum-delay-from-source and maximum-delay-to-sink. For each node v , the maximum-delay-from-source is defined as:

$$\Delta_S(v) = \max_{u \in P(v)} (\Delta_S(u) + t_{u,v}) \quad (5)$$

1. In some texts, the actual and required event times are referred to as *early* and *late* event times, respectively. The terms *actual* and *required* are preferred here to avoid confusion with the early and late *signal* times introduced in Section 3.1 and to simplify the CPM extensions presented in Section 3.2.

vertex v	$T_r = 10$ minutes (600 seconds)			$T_r = 8$ minutes (480 seconds)		$\Delta_S(v)$	$\Delta_F(v)$
	actual time $e(v)$	required time $r(v)$	slack $s(v)$	required time $r(v)$	slack $s(v)$		
A	0	0	0	-120	-120	0	600
B	30	30	0	-90	-120	30	570
C	45	210	165	90	45	45	390
D	210	210	0	90	-120	210	390
E	270	270	0	150	-120	270	330
F	300	300	0	180	-120	300	300
G	450	450	0	330	-120	450	150
H	75	390	315	270	195	75	210
I	135	450	315	330	195	135	150
J	480	480	0	360	-120	480	120
K	540	540	0	420	-120	540	60
L	570	570	0	450	-120	570	30
M	210	465	255	345	135	210	135
N	270	525	255	405	135	270	75
O	585	585	0	465	-120	585	15
P	600	600	0	480	-120	600	0

Table 1: Early and Late Time Calculations for Figure 1

For the source node, $\Delta_S(v) = 0$. The maximum-delay-to-sink is similarly defined as:

$$\Delta_F(v) = \max_{u \in S(v)} (\Delta_F(u) + t_{u,v}) \quad (6)$$

For the sink node, $\Delta_F(v) = 0$. The actual event times are simply $e(v) = \Delta_S(v)$ and the required event times $r(v) = T_r - \Delta_F(v)$. Values for $\Delta_S(v)$ and $\Delta_F(v)$ for the network of Figure 1 are shown in Table 1.

2.1.3 Controlling and Non-Controlling Jobs

Another method for identifying critical paths is based on the classification of arrows in the project network as *controlling* or *non-controlling*. Using the actual event times $e(v)$ computed in the forward pass through the network, an arrow from node u to node v is defined as *controlling* if $e(v) = t_{u,v} + e(u)$. The arrow is *non-controlling* if $e(v) > t_{u,v} + e(u)$. A critical path can thus be identified as a sequence of controlling arrows which connects the source node to the sink node. The slack for the critical path is the difference between the required and actual arrival times at the sink node. Note that this approach does not directly provide slack information for the other events in the project network, and so for most project graphs, the original method for identifying critical paths should be preferred. However, this method provides a consistent definition for critical paths in the more complex network structures which will be encountered in Section 3.2.

We can also make an observation about the controlling nature of the jobs in the critical path. Any increase in the job time for a critical job will cause a corresponding increase in the project completion time. We can express this as

$$\frac{\partial e(F)}{\partial t_{u,v}} = 1 \text{ where } F \text{ is the sink node and } t_{u,v} \text{ is the time of a critical job } u \rightarrow v.$$

2.1.4 Parallel Critical Paths

When large project networks are analyzed, it is often possible to find multiple *parallel* critical paths. When critical paths exist together in parallel, each path is sufficient to prevent any reduction of the event time on the sink node F . In order to reduce the final event time, all parallel critical paths must be shortened. If any one is unmodified, then the unchanged path will remain critical and will hold the project completion time to its original value.

2.1.5 Activity-On-Node Formulation

The discussion has thus far described the activity-on-arrow (AoA) formulation for project networks. Although this is the most common formulation for project networks, another formulation, called activity-on-node (AoN) is also popular. In the AoN formulation [16], each node in the diagram represents a task to be performed. The time required to perform each task is marked on its corresponding node. Arrows still represent dependencies between tasks; an arrow from task A to task B indicates that task A must complete before task B can begin. An AoN diagram for the oil change project is shown in Figure 3. Note that the dummy arrows in Figure 1 are no longer needed as false dependencies cannot arise in AoN diagrams. [17, p. 27]

Analogous definitions can be made for actual and required event times, slack, maximum-delay-from-source, and maximum-delay-to-sink. Event times now refer to the completion of the task corresponding to node v . The required time and maximum-delay-to-source for a node do not include that node's delay, but actual time and maximum-delay-from-source do include it.

2.2 Critical Paths in Combinational Circuits

As mentioned, the application of critical path analysis to digital circuits was pioneered by Kirkpatrick and Clark [1] and was later elaborated by Hitchcock, Smith, and Cheng [2]. Both groups used the AoN formulation to analyze the timing of combinational circuits. Each node in the network represented a combinational block (often a single gate). In Kirkpatrick's work, these delays were represented as probability distributions. Hitchcock used simpler scalar values which could optionally be modified to represent probability distributions. As in the general CPM, critical paths were determined by a forward pass to calculate signal arrival times at each gate output and a subsequent backward pass to directly compute slacks (required times were not calculated). Figure 4 shows the time calculations and critical paths identified by Hitchcock's Timing Analysis program for a simple circuit. In the figure, circles represent combinational logic blocks and squares represent inputs and outputs of the combinational circuit. The delay of each block is marked inside the corresponding circle.

The timing of combinational circuits can also be modeled using an AoA network, as shown in Figure 5. In this formulation, each node corresponds to a net and arrows correspond to paths through gates. Each gate is replaced with multiple arrows instead of a single node, which provides the added flexibility to model multiple input-output delays through combinational logic blocks. Figure 5 contrasts the AoN and AoA formulations for a small circuit.

For both types of networks we can also use CPM techniques to find the *shortest* path through a circuit, as described by Hitchcock et al. [2] Actual and required event times are calculated from:

$$e(v) = \min_{u \in P(v)} (e(u) + t_{u,v}) \tag{7}$$

$$r(v) = \max_{u \in S(v)} (r(u) - t_{u,v}) \tag{8}$$

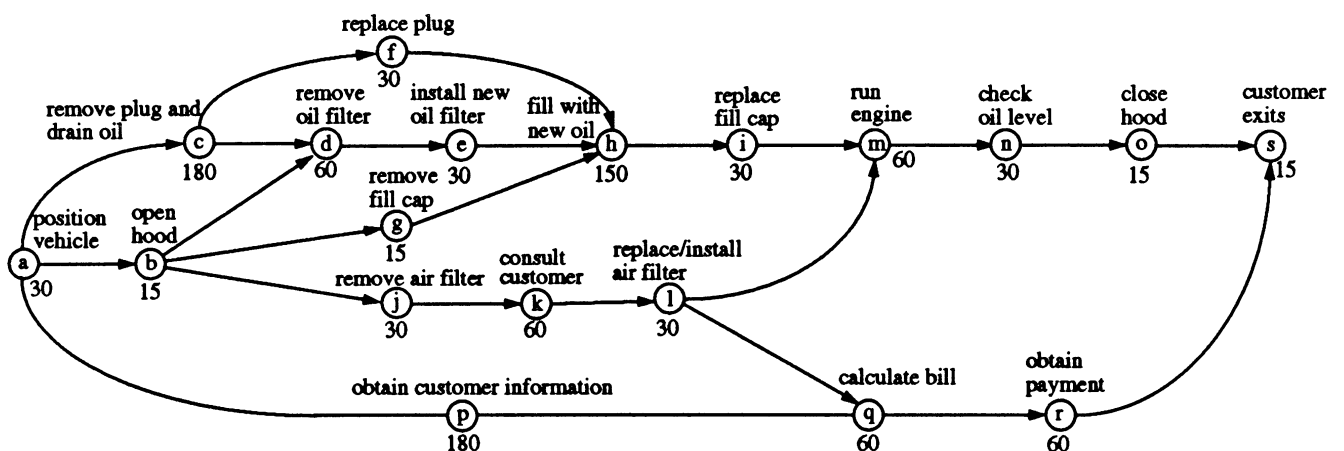


Figure 3: AoN Formulation of the Oil-Change Example

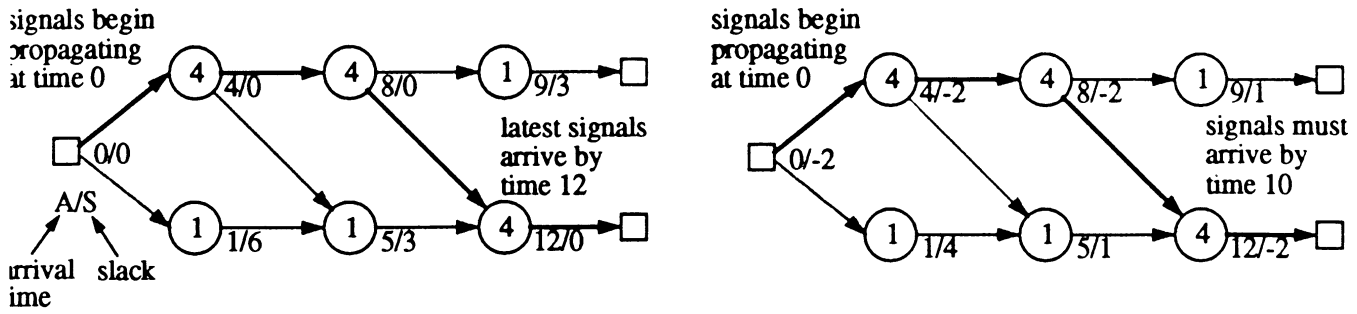


Figure 4: Critical Path Determination in Hitchcock's Timing Analysis Program

Since our concern with shortest paths is that they not be *too short*, the required time T_r becomes the *earliest* time that the project is allowed to complete. The definition of event times in equation (7) can be viewed as a replacement of the nodes in the original project network, which computed *max* functions, with new nodes which propagate the *minimum* event times from their inputs to their outputs.

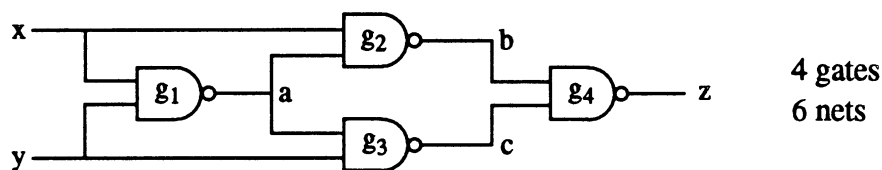
To maintain the notion that negative slacks and floats correspond to errors, we simply exchange terms in the subtraction, so that the slacks and floats are defined by:

$$s(v) = e(v) - r(v) \quad (9)$$

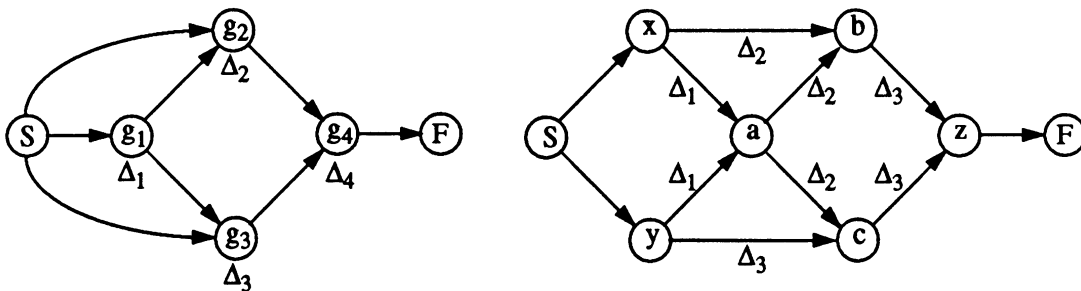
$$f(u \rightarrow v) = t_{u,v} + e(u) - r(v) \quad (10)$$

The float on arrow $u \rightarrow v$ represents the amount by which its delay can be *reduced* without causing the event time at the sink $e(F)$ to fall below T_r . Critical nodes and arrows are again defined as those having the most negative slack and float, respectively. A critical path is again a sequence of critical nodes and arrows that connect the source and sink nodes.

Finally, for the early signals, critical paths again determine the (early) event time at the sink node. Any decrease in the signal delay of a critical arrow will cause a corresponding decrease in the final event time. However, if there are parallel critical paths, increasing the delay of a critical arrow may not affect the event timing at the sink node. When parallel short paths exist, all must be lengthened in order to obtain the benefit of improving any single path.



NAND implementation of EXCLUSIVE-OR



Activity-On-Node Formulation

Activity-On-Arrow Formulation

Figure 5: Alternate Formulations of the Circuit Timing Verification Problem

2.3 Critical Paths in Synchronous Sequential Circuits

The above combinational path analysis techniques are readily adaptable to synchronous sequential circuits that use edge-triggered devices as storage elements. Such circuits can be partitioned into feedback-free regions whose inputs are driven either from edge-triggered flip-flops or primary inputs, and whose outputs are either connected to primary outputs or edge-triggered flip-flops. The flip-flops are usually, but not necessarily, controlled by the same clock signal. The longest and shortest paths in each of these regions can now be determined as discussed earlier. In particular, the required times for the longest path analyses are determined by the specified clock schedule and the setup times for the output flip-flops. Slacks and floats are then calculated and used to identify the long critical paths in each region and to highlight any setup violations. Similarly, the required times for the shortest path analyses are determined by the hold times of the output flip-flops and are used to identify the short critical paths in each region and to highlight any hold violations.

For edge-triggered circuits, we can also solve the *optimal clocking* problem using a slight variation of the above procedure. Here recall that we seek the clock schedule with the smallest possible cycle time, $T_{c, \min}$, for the given circuit. In this case, the required times for each of the feedback-free regions are determined from the maximum path delay so that the slacks on all critical paths are exactly zero. This ensures that the associated clock event occurs as early as possible without causing a timing violation. Adjusting for setup times, these required times determine the spacing of events in the optimum clock schedule, including the minimum clock period.

In general, a critical path in a sequential circuit determines the minimum cycle time of the circuit. An increase to any delay along the critical path will force the minimum cycle time to increase. More formally, if $\Delta(P)$ is the delay of a critical path P , then if P is critical, $\frac{\partial T_{c, \min}}{\partial \Delta(P)} > 0$.

Designers of edge-triggered circuits may also be concerned about the short paths through a network and whether these paths cause the hold constraints on flip-flop inputs to be violated. The network is partitioned as before, and the procedure of Section 2.2 is used to calculate early signal times and identify short critical paths. In edge-triggered circuits, problems with critical short paths are rare, and usually occur only when flip-flops have large hold times and when zero minimum delays are assumed. If problems arise, they are typically fixed by adding delay to the short paths or using devices with smaller hold times.

The definition of critical paths becomes more complicated for circuits clocked with level-sensitive latches, since signal paths may no longer be confined to the individual feedback-free combinational regions. Since signals can flow directly through level-sensitive latches, critical paths can now extend through latches and can include both combinational and sequential circuit elements. For example, in the circuit shown in Figure 6, the setup violation at latch L_3 can be eliminated by reducing either of the delays $\Delta_{1,2}$ or $\Delta_{2,3}$. Thus both of these delays should be considered part of the critical path from L_1 to L_3 . The definition of a path must, therefore, be extended to include multiple combinational segments joined by transparent latches.

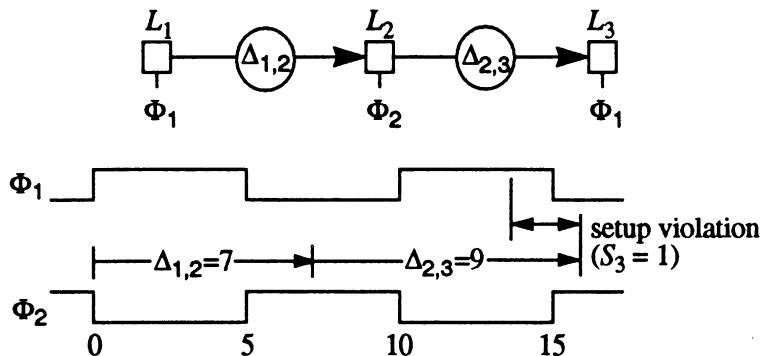


Figure 6: Critical Path through a Level-Sensitive Latch

This extension complicates the analysis in two ways. The first complication appears when we analyze short paths in latched circuits. Section 3.2 presents graph models of the long and short path constraints for level-sensitive latches.

For short paths, the graph is essentially the short-path variant of a CPM network; however, it is complicated by the presence of nodes that correspond to both max and min functions. Existing CPM methods do not provide for such mixed-node graphs.

The second difficulty is that the long- and short-path project networks can no longer be guaranteed to be acyclic. If a circuit contains feedback, then the project networks will contain cycles, and some of these cycles can constrain circuit timing. Classical CPM approaches are unable to deal with cyclic project networks.

These two complications will be analyzed in detail in subsequent sections, and procedures for dealing with them will be presented. Next, however, we describe the timing model for latch-controlled circuits and its associated graph representation.

3 Timing Model and Constraint Graphs for Latch-Controlled Circuits

This section reviews our timing models for circuits with level-sensitive latches and introduces a graph representation of these models. This graph model will serve to define and illustrate each of the three types of critical paths which can arise in such circuits.

3.1 Timing model

Our timing model for latch-controlled circuits is an extension of the one described by Sakallah, Mudge, and Olokutun [12]. For reference, the model equations and constraints are listed in Table 3. Variables in the clock model denote cycle time, T_c , and the times R_p and F_p of the rise and fall events for each clock phase p . This simplifies the modeling of negative level-sensitive devices, as we now only need to specify which clock event controls the opening (enabling) and closing (latching) of each latch. These event times are with respect to a common system frame of reference that is *modulo* T_c . A sample clock schedule is illustrated in Figure 7. Delays in the clock distribution network are represented with a pair of terms for each latch, q_i and Q_i , which correspond to the minimum and maximum delays between the clock generator and latch i . Clock skew between latches is calculated as a difference of clock delay variables.

Parameters in the circuit model include latch setup and hold times, S_i and H_i , the minimum and maximum delays through each latch, δ_i and Δ_i , and minimum and maximum combinational delays between each connected pair of latches, δ_{ij} and Δ_{ij} . Latch operation is described by specifying the enabling and latching event times, E_i and L_i , in terms of the events in the clock schedule. The enable event allows data to move from the input to the output of the latch and the latch event causes the data on the output to be held. By allowing the enabling and latching events to be associated with either the rising or falling clock edges, we can specify both positive and negative latch types, as shown in Table 2.

The primed versions of the enable and latch event variables, E'_i and L'_i , have been shifted into the *local frame-of-reference* of latch i , which represents a single cycle of operation and which ends on the latch event L_i . Their values are obtained using the general phase-shift function $t' = T_c - (L_i - t) \bmod T_c$ which converts a variable t in the global

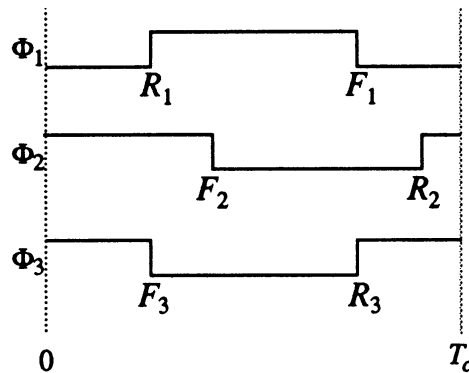


Figure 7: Sample Clock Schedule

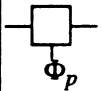
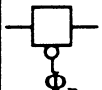
Type	Symbol	E_i	L_i
Positive Level-Sensitive Latch		R_p	F_p
Negative Level-Sensitive Latch		F_p	R_p

Table 2: Single-Phase Latch Types

frame of reference to a variable t' in the frame of reference of latch i . For the enable event, this gives $E'_i = T_c - (L_i - E_i) \bmod T_c$ and for the latch event, $L'_i = T_c - (L_i - L_i) \bmod T_c = T_c$.

Clock Constraints	
$(R_i - F_i) \bmod T_c \geq w$	(11)
$(F_i - R_i) \bmod T_c \geq w$	(12)
Combinational Propagation Constraints	
$a_i = \min_{j=1,n}(d_j + \delta_j + \delta_{ji} - E_{L_j L_i}) = \min_{j=1,n}(d_j + \hat{\delta}_{ji})$	(13)
$A_i = \max_{j=1,n}(D_j + \Delta_j + \Delta_{ji} - E_{L_j L_i}) = \max_{j=1,n}(D_j + \hat{\Delta}_{ji})$	(14)
Synchronizer Macromodels	
$A_i \leq T_c - S_i + q_i$	(15)
$a_i \geq H_i + Q_i$	(16)
$d_i = \max(a_i, E'_i + q_i)$	(17)
$D_i = \max(A_i, E'_i + Q_i)$	(18)
Phase Shift Operator	
$E_{L_j L_i} = T_c - (L_j - L_i) \bmod T_c$	(19)

Table 3: Timing Model Summary

The data input of each latch is modeled by the range of possible times, a_i to A_i , within which a new signal can arrive. Similarly, latch outputs are modeled by the earliest and latest times, d_i and D_i , at which new signals depart. All arrival and departure times are defined in the local frame-of-reference of the corresponding latch. The phase-shift operator $E_{L_j L_i}$ is used to convert signal times from the frame-of-reference defined by latching event L_j to that of event L_i . The definition of $E_{L_j L_i}$ insures that $0 < E_{L_j L_i} \leq T_c$ which implies that signals departing from devices with latching event L_j will arrive at devices with latching event L_i before the next occurrence of L_i .

We can describe latches as *transparent* or *nontransparent* depending upon whether the data departure times are determined by data arrivals or the latch enable event. For the late signals, a latch is transparent if $D_i = A_i$ and nontransparent if $D_i = E'_i + Q_i$. Similar definitions also apply for the early signal times.

The combinational propagation equations can be simplified by introducing the phase-shifted delays $\hat{\Delta}_{ij} \equiv \Delta_i + \Delta_{ij} - E_{L_i L_j}$ and $\hat{\delta}_{ij} \equiv \delta_i + \delta_{ij} - E_{L_i L_j}$. For timing verification, these phase-shifted delays are constants since the clock schedule and phase shifts are specified.

3.2 Graph Model of Timing Constraints

The constraints in Table 3 can be separated into two independent sets: one for early signals and one for late signals. Each of these constraint sets can be represented by a graph, as shown in Figure 8. In both graphs, each latch is represented by a pair of nodes labeled A_i and D_i , or a_i and d_i , which correspond to the arrival and departure time equations for the latch. A zero-weight arc connects the arrival and departure time nodes and reflects the arrival time terms in equations (17) and (18). For each term in the arrival time equations (13) and (14), an arc labeled $\hat{\Delta}_{i,j}$ or $\hat{\delta}_{i,j}$ connects the departure time vertex of latch j to the arrival time vertex of latch i . Note that except for splitting each latch into two vertices, the constraint graphs as defined thus far are structurally isomorphic to the circuit being modelled. The clock system and its associated constraints are incorporated into the constraint graphs by adding two special vertices and sets of associated arcs. Clock distribution is modelled by connecting a source vertex to each latch departure time vertex. The source vertex is denoted by S in the late signal graph and s in the early signal graph and the arc weights are $E'_i + Q_i$ in the late signal graph and $E'_i + q_i$ in the early signal graph. In both cases, the arc models the occurrence of the enabling clock event at latch i . Arrival time constraints are modelled by connecting arrival time vertices to a sink vertex labelled F or f in the late and early signal graphs, respectively. In the late signal graph, these arcs represent setup time constraints and have weight $S_i - q_i - T_c$. In the early signal graph, they represent hold time constraints and have weight $-(H_i + Q_i)$.

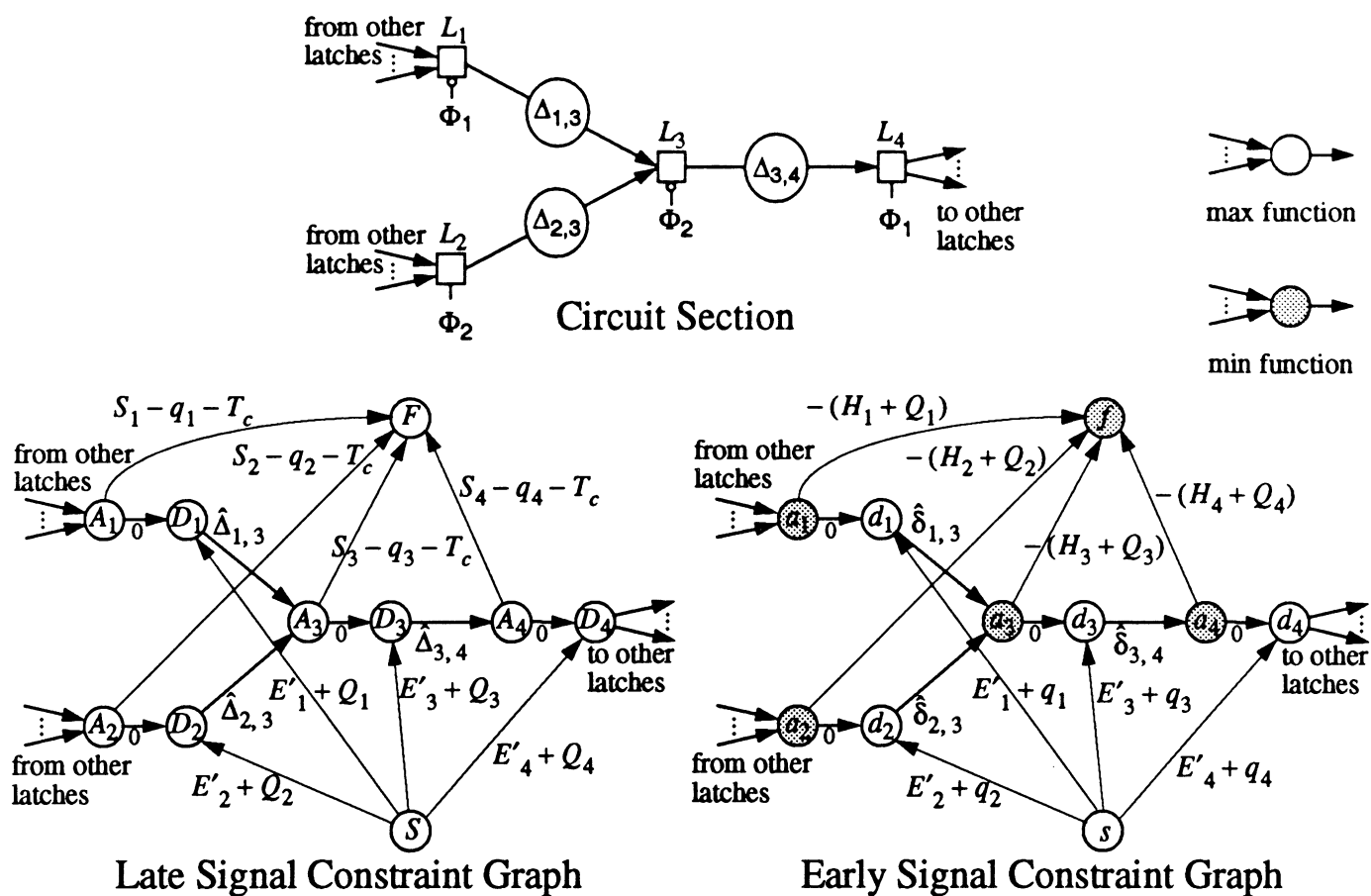
Shaded nodes in the graphs represent min functions; unshaded nodes correspond to max functions. Using the CPM notation, we define an event time for each node such that $e(S) = e(s) = 0$ and the event times for the arrival and departure time nodes are equal to the corresponding times (i.e. $e(A_i) = A_i$, $e(D_i) = D_i$, $e(a_i) = a_i$, and $e(d_i) = d_i$). The constraint graphs are defined so that in the late signal graph, $e(F) > 0$ if and only if a setup violation exists in the circuit, and so that $e(F)$ is the amount of the largest setup violation in the circuit. Similarly, in the early signal graph, $e(f) < 0$ if and only if a hold violation exists in the circuit, and $-e(f)$ is the amount of the largest hold violation. For both graphs, the effective project required time, T_r , is zero.

Primary inputs are modeled with arcs from the source node to some synchronizer arrival node A_i . The arc weights correspond to the early and late arrival times of the input signals shifted into the frame of reference of synchronizer i . Primary outputs are modeled with arcs that connect latch departure times to the sink node. The weights of these arcs are determined by the delay to the output pin and the required output time. Their values are again defined such that $e(F) = 0$ when the late signals arrive at their latest allowable times and $e(f) = 0$ when early signals arrive at their earliest allowable times.

Note that although the nodes and arrows have different labellings, the topological structures of the late and early signal constraint graphs are identical. Also, both graphs contain cycles only when the circuit being analyzed contains synchronous feedback loops. Finally, if we know that $E'_i + Q_i \leq T_c + q_i - S_i$ for all i , then the late-signal graphs can be simplified by eliminating the arrival time nodes and expressing the setup constraints in terms of the synchronizer departure times, D_i .

3.2.1 Critical Paths in Acyclic Late-Signal Graphs

In the late-signal graph, all of the nodes correspond to max functions. If the graph is acyclic, the critical path techniques described in Section 2.1 can be used to obtain critical paths that begin on the start node S and extend to the sink node F . This is a significant result, in that we can now directly apply the large body of existing CPM techniques to the analysis of acyclic circuits with level-sensitive latches. This includes pipelines, many systolic array structures, and FIR


Figure 8: Graph Model of SMO Constraints

filters. If feedback loops are present, we can still apply CPM, but with modifications to be discussed in subsequent sections.

Note, however, that unlike in Section 2.3, the minimum cycle time $T_{c, \min}$ cannot be directly obtained from the event time at the sink node $e(F)$. Instead, $e(F)$ corresponds to the largest setup violation in the circuit. The relationship between $e(F)$ and the minimum cycle time $T_{c, \min}$ can be seen by summing the weights along a critical path P in an acyclic late-signal graph as follows:

$$e(F) = E'_0 + Q_0 + \hat{\Delta}_{0,1} + \hat{\Delta}_{1,2} + \dots + \hat{\Delta}_{m-1,m} + S_m - T_c - q_m \quad (20)$$

where without loss of generality, we assume that the path consists of latches consecutively labeled 0 to m . Substituting the definition of the phase-shifted delays gives:

$$e(F) = \left(\sum_{i=0}^{m-1} (\Delta_i + \Delta_{i,i+1}) + S_m \right) + (Q_0 - q_m) - \left(\sum_{i=0}^{m-1} (E_{L_i L_{i+1}}) + T_c - E'_0 \right) \quad (21)$$

where the first term in the sum corresponds to delays and setup time in the signal path, the second term represents clock skew, and the third term is the amount of time that the clock schedule provides for signal propagation along the path. If we assume that the clock schedule is *scaled*, that is, that all clock parameters scale linearly with the clock schedule, then we can factor the cycle time T_c out of the last term in equation (21) to obtain:

$$e(F) = \left(\sum_{i=0}^{m-1} (\Delta_i + \Delta_{i,i+1}) + S_m \right) + (Q_0 - q_m) - \Lambda(P) T_c \quad (22)$$

Here $\Lambda(P)$ is the latency of path P . $\Lambda(P)$ will be defined more fully in Section 3.3; it is sufficient to note here that its value can be different for each path in the circuit and that for circuits containing level-sensitive latches it is seldom equal to one. As a result, $e(F)$ is not related to the cycle time by a single constant parameter; instead changes to T_c affect the final event time differently depending on the latency of each path. Note, however, that there is a one-to-one correspondence between reductions to critical setup times and propagation delays and reductions to the amount of violation, $e(F)$.

3.2.2 Critical Paths in Acyclic Early-Signal Graphs

The early-signal constraint graphs contain nodes that correspond to min functions, which do not arise in classical CPM. If the graphs contained min nodes only, we could make analogous definitions to the actual and required times and define the required time as the *earliest* allowable time for the project to complete, as described in Section 2.2. However, the early signal constraint graphs contain *both* min and max nodes. This section presents simple extensions that allow analysis of these more complex graphs.

When both node types are present, the actual event times for each node can still be computed in a forward pass through the topologically-sorted network [3]. Event times are uniquely defined by the property that min nodes propagate the minimum of their input times and max nodes propagate the maximum of their input times. As before, required times are calculated with a backward pass, although in mixed networks, some of the required times can be undefined. To see this, consider the graphs shown in Figure 9. Actual and required event times are shown next to each node.

Figure 9-a shows a graph containing predominantly max nodes, and the sink node is also a max node. Actual event times can be easily calculated in a forward pass, but the required time calculation is complicated by the presence of the min node B . The required times are the latest times that each event can occur without increasing the project completion time. The required time for node F is 6, and using equation (2), the required time for B is $6 - 5 = 1$. This is reasonable, since the actual time for event B is also 1 and any increase to this value will cause event F to occur later. However, when we consider event A , we see that the arrow from event A does not control the value of the min function at node B , and as a result, the time of event A could be made arbitrarily late without increasing the completion time at the sink node. As a result, the required time for event A is undefined, and may be considered infinite. A similar analysis for Figure 9-b produces a required time for event a of $-\infty$.

Thus we define the following general rules for performing the backward pass in mixed min/max CPM graphs:

1. The type of the graph (min or max) is determined by the type of the sink node. If the sink node computes a max function, the graph is called a *max graph*. If the sink node computes a min function, the graph is called a *min graph*. Max nodes in min graphs and min nodes in max graphs are termed *minority* nodes.
2. The required times for all nodes are computed as before, using equations (2) and (8), but with the following modification: if an output of a node u is a non-controlling input of a minority node v , then a *modified* required time, $r'(v)$ is used instead of $r(v)$ to calculate $r(u)$. Recall that the definitions of controlling and noncontrolling arrows are given in Section 2.1.3.
3. For each minority node v , the modified required time $r'(v)$ is the required time of v seen by all non-controlling

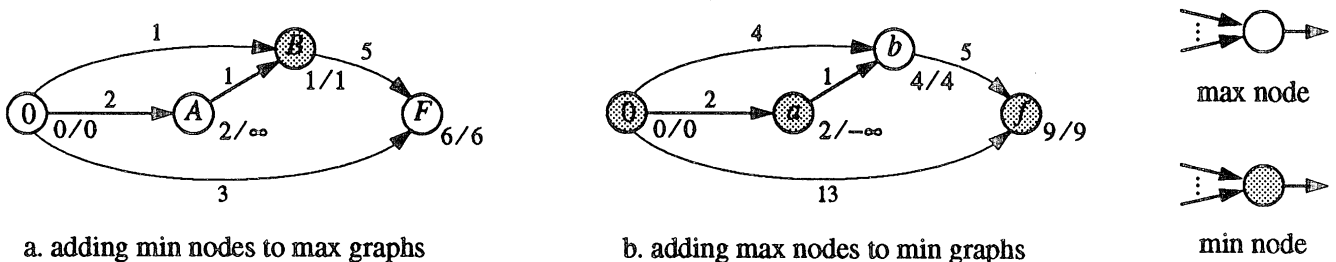


Figure 9: Mixing min and max nodes in CPM graphs

inputs of v . When a max node is a minority node, its modified required time is $r'(v) = -\infty$, since any non-controlling inputs to the max node can occur arbitrarily early without affecting the early completion time of the project graph. For min nodes that are minority nodes, the modified required time is $r'(v) = \infty$, since non-controlling inputs to the min node can occur arbitrarily late without affecting the late completion time of the project.

Using this procedure, actual and required event times can be defined for each node in a mixed min/max CPM graph. Slacks and floats are defined as before, and a critical path can again be identified as a path from the source to the sink consisting of arrows with the most negative float. We can also identify critical paths as sequences of *controlling* arrows (as defined in Section 2.1) that connect the source and sink nodes.

This modification allows us to again apply CPM to latch-controlled circuits, this time in the analysis of the short paths and early signal (hold time) constraints. Again, however, we assume that circuits are feedback-free.

3.2.3 Critical Paths in Cyclic Graphs

If a circuit contains feedback, cycles¹ will be present in the late and early signal constraint graphs. If cycles are present in these graphs, the above two types of paths may still exist, but we must also consider the possibility of a positive-weight cycle existing in the graph. Such a cycle would make it impossible to find a stable set of values for the arrival and departure times and would again preclude the direct use of CPM techniques.

If a positive-weight cycle were present in the late signal graph, it would only contain arrows connecting arrival and departure time nodes. The weight of such a cycle C would be

$$\sum_{i \in C} \hat{\Delta}_i = \sum_{i \in C} (\Delta_i + \Delta_{i, \text{SUCC}(i)} - E_{L_i L_{\text{SUCC}(i)}}) = \sum_{i \in C} (\Delta_i + \Delta_{i, \text{SUCC}(i)}) - \sum_{i \in C} E_{L_i L_{\text{SUCC}(i)}} \quad (23)$$

where $\text{SUCC}(i)$ identifies the next latch in the loop after latch i . This implies that

$$\sum_{i \in C} (\Delta_i + \Delta_{i, \text{SUCC}(i)}) > \sum_{i \in C} E_{L_i L_{\text{SUCC}(i)}}, \text{ or that the total delay around the loop is greater than the amount of time}$$

available for signal propagation. Similar cycles can exist in the early signal graph, but it is easy to show that if there is a positive-weight cycle among the early constraints, the corresponding set of late signal constraints must also contain a positive-weight cycle [9]. This makes it unnecessary to check the early signal graph for positive-weight cycles. Positive-weight cycles significantly affect the verification procedures discussed in Sections 7 and 8; methods for dealing with them will be presented as the need arises.

3.3 Critical Paths in Latch-Controlled Circuits

In the next three sections we formally define three types of critical paths: critical long paths, critical short paths, and critical loops. All three of these types can be observed in constraint graphs of the SMO model constraints. They are presented in increasing order of complexity. We also discuss these critical paths directly in terms of latches and logic blocks. In this context, a path P is a sequence of latches $P = L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_m$ where each latch is directly connected to its predecessor through a combinational logic segment (Figure 10). The length of P is defined as the number of combinational segments in the path, $|P| = m$. The path delays $\delta(P)$ and $\Delta(P)$ are defined as the sum of the minimum and maximum delays along the path, respectively, and the path latency $\Lambda(P)$ is the number of clock cycles available for signals to propagate the length of the path. For multiphase circuits or circuits using level-sensitive

1. Here the term *cycle* is used in the graph theoretic sense, and should not be confused with its usage in the phrase *cycle time* which relates to the operating speed of a circuit.

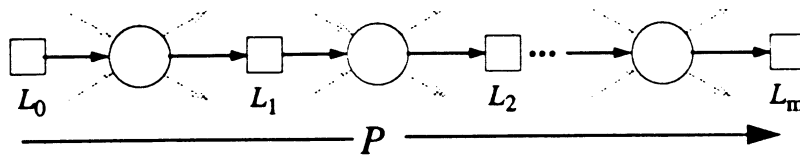


Figure 10: Sample Path

latches $\Lambda(P)$ may be fractional. Note that since we are now considering the verification problem only, $\Lambda(P)$ is constant throughout the analysis.

In our analysis of critical paths, we require that all paths begin with a clock event edge. This is due to an assumption we make about synchronous circuit operation: in a closed synchronous system (free of unsynchronized inputs or outputs), no asynchronous feedback loops will be present, and the only stimuli that the system will receive will be from events in the clock schedule. This assumption is reflected in the graphs of Figure 8, where the only connections from the source node correspond to or are derived from clock events.

These definitions allow for the existence of parallel critical paths as described in Section 2.1.4. When there are multiple critical paths fanning into a latch L , they can be viewed as a tree of paths rooted at latch L . Such a tree can be stored and manipulated as if it were a single path. The *width* of a path is then defined to be the number of leaf nodes in the path tree.

Also, it has been assumed that we are only considering a single critical combinational path through each combinational segment between latches. If instead multiple parallel combinational paths are present, the total number of paths grows combinatorially. In general, the number of unlumped paths corresponding to a single lumped path is the product of the number of parallel critical paths in each combinational segment of the lumped path. Fortunately, in most cases these paths only need to be represented implicitly.

In many applications, it is important to be aware of parallel critical paths when they exist. If we are attempting to optimize circuit timing by reducing critical path lengths, we must reduce the length of all parallel critical paths. If not, the unreduced paths will remain critical and the range of valid clock speeds will be unchanged. It is also important to know how much delay can be removed from (or added to) a critical path before some other path also becomes critical. Once this point is reached, all parallel critical paths must be optimized simultaneously.

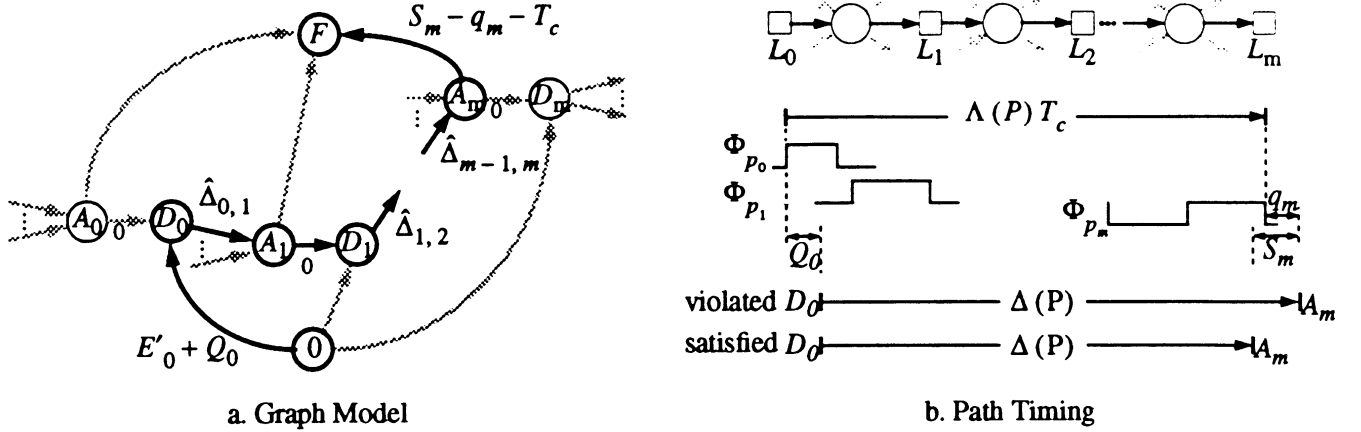
It is a simple extension to extract paths through individual gates in combinational logic blocks. Once we have obtained a set of arrival and departure times for each clocked component, we can quickly calculate the signal arrival and departure times at each combinational gate in the intermediate logic. This can be done in linear time if the gates are sorted topologically. CPM-based procedures can then be used to trace critical paths through the combinational blocks.

4 Critical Long Paths

The first type of critical path we describe is called a *critical long path* and results from the setup constraint at the input of each latch. Such a path corresponds to a critical path in the late signal graph that connects the source and sink nodes, as shown in Figure 11-a. The formal definition of a critical long path is as follows:

Definition 1 A critical long path is a path in a late-signal constraint graph consisting of a cycle-free sequence of critical arcs which connect the source and sink nodes.

Critical arcs are defined as in Section 2.1.1 as the arcs in the project network having the most negative slack. These arcs are all *controlling* (Section 2.1.3), and Definition 1 could be easily rephrased to describe a critical long path as a sequence of controlling arcs which connect the source and sink nodes of a late signal graph. Comparing this definition


Figure 11: Sample Critical Long Path

with the late signal graph shown in Figure 11-a, we see that the following set of constraints must hold for any critical long path:

$$D_0 = E'_0 + Q_0 \quad (24)$$

$$\forall i \in \{1, \dots, m\}, A_i = D_{i-1} + \Delta_{i-1} + \Delta_{i-1, i} - E_{L_{i-1}, L_i} = D_{i-1} + \hat{\Delta}_{i-1, i} \quad (25)$$

$$\forall i \in \{1, \dots, m-1\}, D_i = A_i \quad (26)$$

$$\forall i \in \{1, \dots, |L|\}, A_m - (T_c + q_m - S_m) \geq A_i - (T_c + q_i - S_i) \quad (27)$$

In equations (24)-(27) we assume, without loss of generality, that the latches in the path are numbered from 0 to m . Critical long paths are required to be acyclic, although they will exist in both cyclic and acyclic circuits. Cyclic critical paths are described in Section 6, where it is shown that we can safely decouple the analysis of loops in cyclic constraint graphs.

The following theorem describes the relationship between critical long paths and the setup time constraints in a circuit:

Theorem 1 A path P is a critical long path if and only if the following two conditions are satisfied: (1) any increase in a delay along the path will tighten or worsen the most severe setup time constraint in the circuit, and (2) a decrease to any delay in the path will reduce the severity of this constraint, as long as no parallel critical paths are present.

Proof: (only if part) It simple to show that if P is a critical long path, conditions (1) and (2) will be satisfied. Recall that in the late signal graphs, the event time at the sink node, $e(F)$, is the amount of the largest setup violation in the circuit. If $e(F)$ is negative, then it represents the amount of slack in the tightest setup constraint. If P is a critical long path, then equations (24)-(27) guarantee that P determines the value of $e(F)$, which can be calculated by simply summing the arc weights along P . An increase in any delay along P will cause $e(F)$ to increase. Reducing any delay in P will allow $e(F)$ to be reduced, but $e(F)$ will only be reduced if there are no other critical paths in parallel with P . This is because the event time at each node is the maximum of its input event times: increasing the maximum input time will always cause a change on the output; reducing the maximum input time will only cause a change when the reduced time remains the maximum input time.

(if part) We also can show that if conditions (1) and (2) are satisfied, P is a critical long path. Except for the qualification for parallel paths made in condition (2), both conditions require that the sensitivity of the largest (or nearest) setup violation to changes in P be nonzero. This implies that the event time at the sink node, $e(F)$, be equal to the sum of the arc weights along P , a condition which can only be true if equations (24)-(27) are satisfied for P . These equations imply that each arc of P be a controlling arc; therefore, P must be a critical long path. If this were not so, then some other path P' would be critical and increases in delays along P would not effect $e(F)$ unless they were

large enough to cause the total arc weight of P to exceed that of P' . Again, this contradiction requires that P be a critical long path. \square

If the clock schedule is *scaled* (as in Section 3.2.1), then the minimum cycle time, $T_{c, \min}$, is directly sensitive to increases in path delays. $T_{c, \min}$ may also be sensitive to reductions in critical path delays, but only if there are no parallel critical paths present. These parallel paths may contain different numbers of latches, but all begin at the source node.

An interesting special case of parallel critical paths occurs when two or more paths meet at a latch where $A_i = E'_i + Q_i$. We define these latches to be *semitransparent*, since their timing satisfies the models for transparent ($D_i = A_i$) and nontransparent ($D_i = E'_i + Q_i$) latches. An illustration of a semitransparent latch is shown in Figure 12. In the circuit shown, latch L_2 is semitransparent. For simplicity, all relevant clock delays are zero. Although signals depart from the latch as soon as they arrive, there is a discontinuous relationship between the departure time and the delays leading up to the latch (which determine the arrival time A_2). As shown, increases in $\Delta_{1,2}$ cause corresponding increases in the departure time from L_2 , but decreasing $\Delta_{1,2}$ has no effect on the timing at latch L_2 .

If conditions (24)-(26) are satisfied for a path P , then we call P a *candidate* long path. If condition (27) is also satisfied, P is a critical long path. If P is a critical long path and $A_m = T_c + q_i - S_m$, then P is a *satisfied* critical long path. If P is a critical long path and $A_m > T_c + q_i - S_m$, then P is a *violated* critical long path.

Combining conditions (24)-(27), we see that a satisfied critical long path constraint is of the form:

$$\sum_{i=1}^m (\Delta_{i-1} + \Delta_{i-1,i}) + S_m + Q_0 - q_m = \sum_{i=1}^m (E_{L_{i-1}, L_i}) + (T_c - E'_0) = \Lambda(P) T_c \quad (28)$$

Note that the terms on the leftmost side of the equation represent times required for data to propagate through logic and get set up at the input of the final storage device. The terms in the center and on the right represent the time available for these operations to take place. If the path were violated, the leftmost "=" would be replaced by ">".

The timing of a typical critical long path is illustrated in Figure 11-b. Note that if a path is critical, modifying S_m or the clock schedule terms will also affect the critical setup constraint. Also note that a subpath of a critical long path may also be critical if the same amount of setup violation is present on another latch in the path.

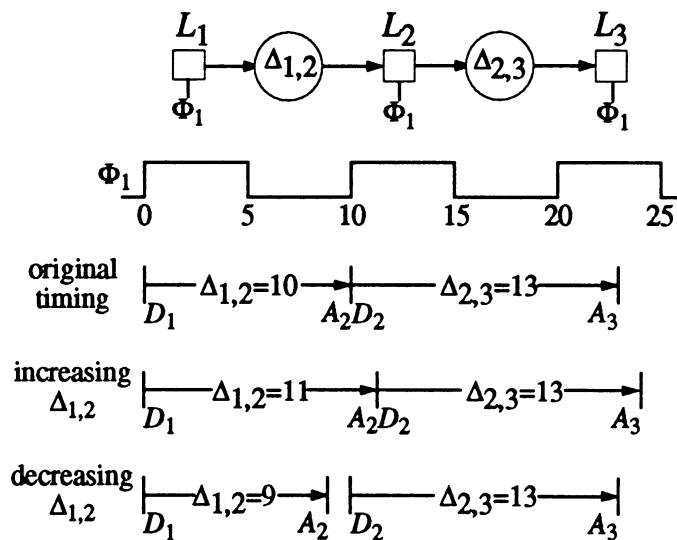


Figure 12: Semitransparent Latch Properties

5 Critical Short Paths

The second type of critical path is called a *critical short path*. It results from the hold constraint at each latch input. Critical short paths correspond to critical paths in the early signal graphs illustrated in Figure 8. These paths begin at the source node, extend through a clock edge, then extend through an arbitrary number of latch arrival and departure time nodes before connecting to the sink node. This is illustrated in Figure 13-a. The formal definition of a critical short path is as follows:

Definition 2 A critical short path is a path in a late-signal constraint graph consisting of a cycle-free sequence of critical arcs which connect the source and sink nodes.

Again, critical arcs are those arcs having the most negative slack, where slack is now defined for the early signals, as in equations (9) and (10) with required times defined as in Section 3.2.2. A critical short path can also be defined as a sequence of controlling arcs which connect the source and sink nodes of an early signal graph. Relating this to the graph of Figure 13-a, we see that the following constraints must hold for any critical short path:

$$d_0 = E'_0 + q_0 \quad (29)$$

$$\forall i \in \{1, \dots, m\}, a_i = d_{i-1} + \delta_{i-1} + \delta_{i-1,i} - E_{L_{i-1}L_i} = d_{i-1} + \hat{\delta}_{i-1,i} \quad (30)$$

$$\forall i \in \{1, \dots, m-1\}, d_i = a_i \quad (31)$$

$$\forall i \in \{1, \dots, |L|\}, (H_m - Q_m) - a_m \geq (H_i - Q_i) - a_i \quad (32)$$

As before, we assume that the latches in the path are numbered from 0 to m . Critical short paths are also required to be acyclic, but they will exist in both cyclic and acyclic circuits.

The following theorem describes the relationship between critical short paths and the hold time constraints in a circuit:

Theorem 2 A path P is a critical short path if and only if the following two conditions are satisfied: (1) any decrease in a delay along the path will tighten or worsen the most severe hold time constraint if no parallel critical path is present, and (2) an increase in any path delay will reduce the severity of this constraint, again as long as there are no parallel critical short paths.

Proof: (only if part) As it was for critical long paths, it is simple to show that if P is a critical short path, conditions (1) and (2) will be satisfied. Recall that in the early signal graphs, $-e(f)$ is the amount of the largest hold violation. If $-e(f)$ is negative, then it represents the amount of slack in the tightest hold constraint. If P is a critical short path, then equations (29)-(32) guarantee that it determines the value of $e(f)$, which can be calculated by simply summing the arc weights along P . Increases to any delay in P will allow $e(f)$ to increase, and reductions to any delay in P will

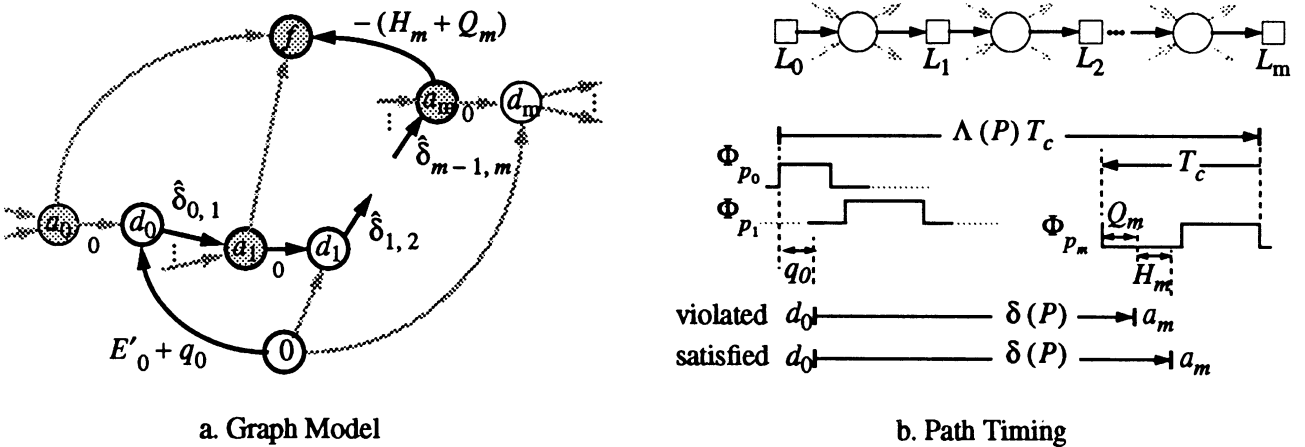


Figure 13: Sample Critical Short Path

allow $e(f)$ to be reduced. Note that both increases and decreases to $e(f)$ can only occur in the absence of parallel critical paths; this is due to the presence of both min and max nodes in early signal constraint graphs. Max nodes only propagate increases if they correspond to the maximum node input; min nodes only propagate decreases in the minimum input.

(if part) We also can show that if conditions (1) and (2) are satisfied, P is a critical short path. Except for the qualifications for parallel paths, both conditions require that the sensitivity of the largest (or nearest) hold violation to changes in P be nonzero. This implies that the event time at the sink node, $e(f)$, be determined by the sum of the arc weights along P , a condition which can only be true if equations (24)-(27) are satisfied for P . These equations imply that each arc of P be a controlling arc; therefore, P must be a critical short path. \square

Observe that unlike the case for critical long paths, both increases and decreases in delays along a critical short path may be blocked by a parallel path. This is due to the presence of both min and max nodes in the early signal constraint graph. The interaction of clock and data signals at max nodes was discussed in Section 4, which described how max nodes can mask reductions in delays and event times. A similar effect can occur at min nodes, where an increase to an event time can be masked by a smaller time value on a min node input.

If conditions (29)-(31) are satisfied for a path P , then P is called a *candidate* short path. If condition (32) is also satisfied, P is a critical short path. If P is a critical short path and $a_m = H_m$, then P is a *satisfied* critical short path. If P is a critical short path and $a_m < H_m$, then P is a *violated* critical short path.

Combining conditions (29)-(32), we see that a satisfied critical short path constraint is of the form:

$$\sum_{i=1}^m (\delta_{i-1} + \delta_{i-1,i}) + q_0 - Q_m = \sum_{i=1}^m (E_{L_{i-1}L_i}) - (T_c - E'_0) - T_c + H_m = (\Lambda(P) - 1) T_c + H_m \quad (33)$$

Note that the terms on the leftmost side of the equation represent times required for data to propagate through logic and reach the input of the final storage device. The terms in the center and on the right represent the time available for these propagations to take place. If the path were violated, the leftmost "=" would be replaced by "<".

The timing of a typical critical short path is illustrated in Figure 13. Critical short paths usually do not contain critical subpaths, since $E'_i + q_i$ is almost always greater than H_i , causing a latch to be nontransparent whenever a hold constraint is tight or violated.

Critical short paths can in some cases place an upper bound on the cycle time of a circuit, especially when the clock schedule is scaled (see Section 3.2.1). We call this upper bound $T_{c,max}$, the maximum cycle time for the circuit. If the clock schedule is scaled, then the latency $\Lambda(P)$ for each path is constant, and $T_{c,max}$ can be calculated from equation (33).

In our definition, critical short paths may be of arbitrary length. However, for most practical circuit designs, critical short paths should be no longer than one combinational segment. A circuit with a critical short path of length 2 is shown in Figure 14. This circuit will work correctly for the timing shown, although the minimum delay from L_2 to L_3 is by itself too short to prevent a hold violation at L_3 . Physically, this means that whenever the circuit is started, a hold violation will occur on L_3 in the first cycle of operation, but then the path from L_1 to L_2 will cause the departure time from L_2 to increase, causing the hold violation to disappear. In some systems, these transient errors may be acceptable, but in others (such as restarting a stopped CPU pipeline), they are unacceptable. As a result, in such systems, all critical short paths should be no longer than one segment. If we wish to verify under this condition, we simply replace the original early departure time equation $d_i = \max(a_i, E'_i + q_i)$ with $d_i = E'_i + q_i$. This has the added benefit of simplifying the early signal constraint graphs by eliminating all max nodes from the early signal constraint graph.

6 Critical Loops

Finally, a third type of critical path can limit circuit operating speeds. We call these paths *critical loops*, and they appear as zero- and positive-weight cycles in the late and early signal constraint graphs. A critical loop in a late signal constraint graph is shown in Figure 15-a. The formal definition of a critical loop is as follows:

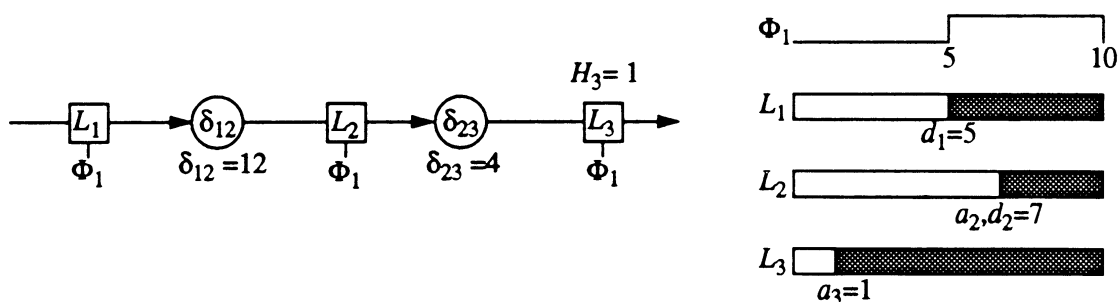


Figure 14: Circuit that exhibits transient but no steady-state errors

Definition 3 A critical loop is defined to be a circular path containing m latches, numbered 0 to m , with $L_m = L_0$, which corresponds to a zero or positive-weight cycle in the late or early signal constraint graphs.

Note that this definition differs from the previous two in that it does not directly correspond to the classical definitions of Section 2. This is due to the fact that classical CPM theory makes no provision for cycles in the project network; in fact, cycles are usually considered errors in planning or job dependency analysis. However, the cycles in our graphs are not due to design errors, they are a natural consequence of the use of feedback in sequential circuits. The majority of sequential circuits use some form of feedback, usually to retain state information. Therefore, we must consider the possibility of cycles in our constraint graphs.

Unfortunately, when a positive-weight cycle exists in a constraint graph, all of the event times in the loop and which depend on the loop are undefined. Examining the constraint graph in Figure 15-a and recalling the discussion of Section 3.2.3, a positive-weight cycle implies the following relationship:

$$\sum_{i=1}^m (\Delta_{i-1} + \Delta_{i-1,i}) > \sum_{i=1}^m (E_{L_{i-1}L_i}) \quad (34)$$

which states that the sum of delays in the loop is greater than the total time available for propagation around the loop. If the total delay of some loop exactly equals the available propagation time, then the loop corresponds to a zero-weight cycle, and all of its arrival and departure times are well-defined as long as they are not disrupted by any violated loop. Again referring to the constraint graph of Figure 15-a, we see that the timing associated with a zero-weight critical loop is as follows:

$$\forall i \in \{1, \dots, m\}, A_i = D_{i-1} + \Delta_{i-1} + \Delta_{i-1,i} - E_{L_{i-1}L_i} \quad (35)$$

$$\forall i \in \{1, \dots, m\}, D_i = A_i \quad (36)$$

Note that all latches in the loop are transparent or semitransparent. Also observe that there is no definition for critical loops in terms of required times, slacks, and floats, as these require a well-defined set of event times and the ability to

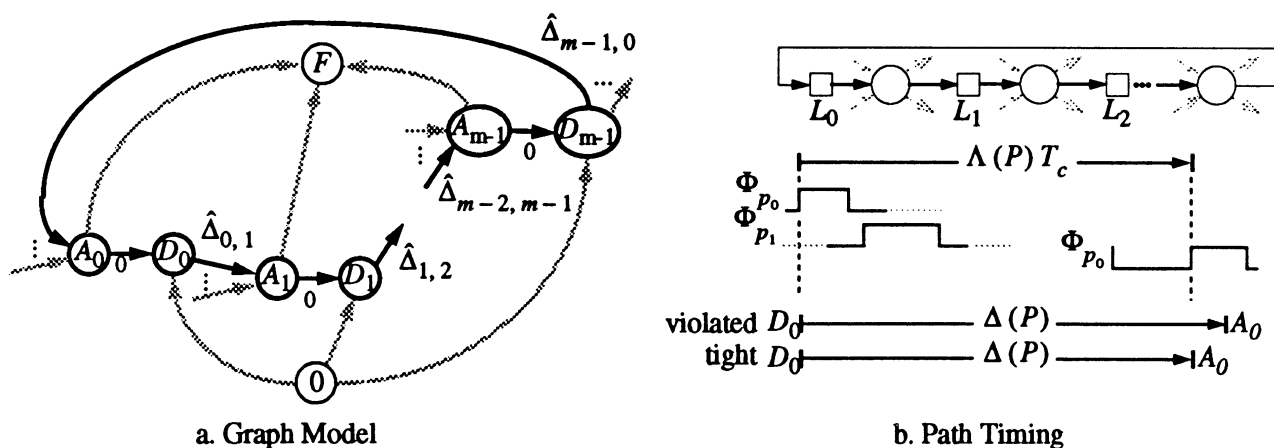


Figure 15: Sample Critical Loop

compare these times against a local constraint (i.e. the setup or hold constraint). In contrast, each loop constraint is not associated with any single latch, but with a group of latches arranged in a topological loop.

If conditions (35) and (36) are satisfied for a path P , then P is a *candidate* critical loop. If P is also a loop, then P is a *satisfied* critical loop. If P corresponds to a positive-weight cycle in the constraint graph, then P is a *violated* critical path. Combining conditions (35)-(36), we see that a satisfied critical loop constraint is of the form:

$$\sum_{i=1}^m (\Delta_{i-1} + \Delta_{i-1,i}) = \sum_{i=1}^m (E_{L_{i-1},L_i}) = \Lambda(P) T_c \quad (37)$$

As before, the terms on the left-hand side of the equation represent times required for data to propagate through logic and the terms on the right-hand side represent the time available for these propagations to take place. If the constraint were violated, the “=” would be replaced by “>”. The timing of a typical critical loop is illustrated in Figure 15-b.

These critical loop constraints can be shown to place a lower bound on the cycle time. We define $T_{c, \text{loop}}$ for a circuit as the minimum cycle time which satisfies all critical loops. $T_{c, \text{loop}}$ can be written as:

$$T_{c, \text{loop}} = \max_{P \in L} \left(\frac{\sum_{i=1}^m (\Delta_{i-1} + \Delta_{i-1,i})}{\Lambda(P)} \right) \quad (38)$$

In the above expression, L is the set of topological loops in the circuit. For loops, the latency, $\Lambda(P_{\text{loop}})$, is a constant integer whether the clock schedule is scaled or not and corresponds to the number of clock cycles allocated to signals traversing the loop.

It is not necessary to enumerate the possibly-exponential number of topological loops to determine $T_{c, \text{loop}}$. $T_{c, \text{loop}}$ can be calculated in polynomial time using a maximum ratio cycle algorithm [7, 13]. We currently are using a variant of Lawler’s maximum mean-weight cycle algorithm [4]. The algorithm is $O(b|V||E|)$ in the worst case, where b is the number of bits of precision in $T_{c, \text{loop}}$, $|V|$ is the number of vertices in the graph, and $|E|$ the number of edges. The algorithm performs a binary search over a range of possible cycle times, and each step in the search provides an additional bit of precision. Lawler’s original algorithm used a Bellman-Ford style iteration in each search step to determine whether a positive weight cycle existed, causing the worst case complexity of the inner search loop to be $O(|V||E|)$. However, with a heuristic suggested by Szymanski [13], the presence of a positive-weight cycle can be determined in as little as $O(|V_c||E|)$, where V_c is the number of vertices in the shortest positive-weight cycle in the network. For most circuits, $|V_c| \ll |V|$. The modification is quite simple, and involves periodically checking a subgraph for cycles; if a cycle is found, its weight is calculated to see if it has positive weight. The subgraph consists of exactly one controlling arc per node; as a result, it can be checked in $O(|V|)$ time. This in theory raises the worst-case complexity of computing $T_{c, \text{loop}}$ to $O(b|V|^2|E|)$; in practice it runs much faster than the original $O(b|V||E|)$. Another observation which accelerates the procedure is that we can use the delay and latency of each positive-weight cycle we detect to calculate a possibly better lower bound on $T_{c, \text{loop}}$ than that used in the binary search.

In Section 3.2, we observed that positive weight cycles could also exist in the early signal constraint graph. However, the following theorem, similar to one by Szymanski [9], shows that we need not check for critical loops in the early signal graph.

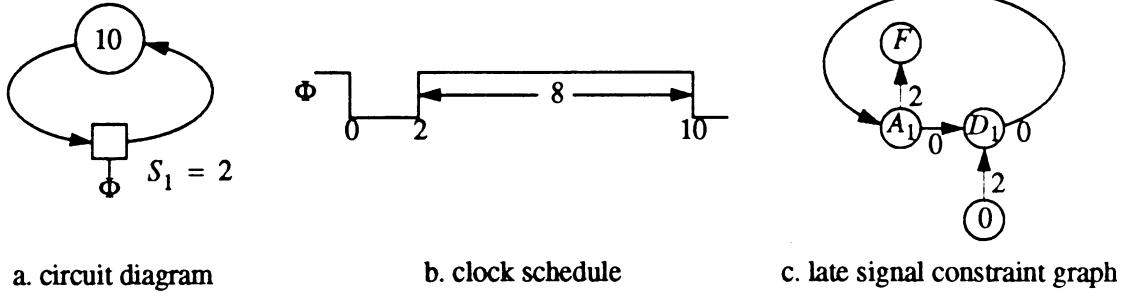
Theorem 3 If a critical loop constraint is satisfied for the late signal variables, it is also satisfied for the early signal variables.

Proof: The critical loop constraints for the early signal variables have the same form as constraints (35)-(36), but with a_i , d_i , δ_i , and δ_{ij} substituted for their late signal counterparts. A satisfied loop constraint has the form:

$$\sum_{i=1}^m (\delta_{i-1} + \delta_{i-1,i}) \leq \sum_{i=1}^m (E_{L_{i-1},L_i})$$

Since $\delta_i \leq \Delta_i$ and $\delta_{i,i-1} \leq \Delta_{i,i-1}$,

$$\sum_{i=1}^m (\delta_{i-1} + \delta_{i-1,i}) \leq \sum_{i=1}^m (\Delta_{i-1} + \Delta_{i-1,i}),$$


Figure 16: Degenerate Loop Example

guaranteeing that the early signal loop constraints will be satisfied whenever the late signal loop constraints are satisfied. \square

An interesting property of satisfied critical loops is that they can have multiple feasible solutions. Szymanski observed that loops consisting entirely of transparent latches (with each latch controlling the timing of its successor) could operate correctly for a range of arrival and departure time values [9]. As an example, he presented the trivial loop shown in Figure 16. The latch shown can operate correctly with any arrival time in the range $2 \leq A_1 \leq 8$.

Since we have assumed purely synchronous operation, all signals can be traced back to an event in the clock schedule. As a result, only one of the range of solutions will correspond to the true physical timing, and this solution will be the one which can be obtained by considering the interaction of the loop with the clock events and external latches. If no other latches feed into the loop, the loop timing will be determined by the enable event of one of its latches. Examining the constraint graph in Figure 16, we see that for the example shown, the actual arrival time $A_1 = 2$. In general, this timing will be the minimum feasible loop solution. If other latches fan into the loop, they can push the loop timing to a higher solution, but if the timing is pushed too high, a setup violation will result at one of the latches in the loop.

The range of feasible timings for an isolated loop can be calculated by observing that for each latch in the loop, $A_i + \hat{\Delta}_{i,i+1} = A_{i+1}$, where the index additions are implicitly *modulo m*. Here we again assume that the loop contains m latches, numbered from 0 to $m-1$ with latch i fanning into latch $i+1$ and latch $m-1$ fanning into latch 0. For each latch, we must have $A_i \leq T_c - S_i + q_i$ to prevent a setup error from occurring on the latch. Also, to maximize the time available for signal propagation, $A_i \geq E'_i + Q_i$. Therefore, for latch i , we must have:

$A_i + \hat{\Delta}_{i,i+1} \geq E'_{i+1} + Q_{i+1}$ $A_i + \hat{\Delta}_{i,i+1} + \hat{\Delta}_{i+1,i+2} \geq E'_{i+2} + Q_{i+2}$ <p style="text-align: center;">...</p> $A_i + \sum_{j=i}^{i+m-1} \hat{\Delta}_{j,j+1} \geq E'_{i+m} + Q_{i+m}$	$A_i + \hat{\Delta}_{i,i+1} \leq T_c - S_{i+1} + q_{i+1}$ $A_i + \hat{\Delta}_{i,i+1} + \hat{\Delta}_{i+1,i+2} \leq T_c - S_{i+2} + q_{i+2}$ <p style="text-align: center;">...</p> $A_i + \sum_{j=i}^{i+m-1} \hat{\Delta}_{j,j+1} \leq T_c - S_{i+m} + q_{i+m}$
$\max_{k \in (1,m)} \left(E'_{i+k} + Q_{i+k} - \sum_{j=i}^{i+k-1} \hat{\Delta}_{j,j+1} \right) \leq A_i \leq T_c - \max_{k \in (1,m)} \left(S_{i+k} - q_{i+k} + \sum_{j=i}^{i+k-1} \hat{\Delta}_{j,j+1} \right)$	

Applying this result to the circuit of Figure 16 verifies that the range of arrival times for the latch is in fact $2 \leq A_1 \leq 8$.¹

1. Note that the hold time constraints for each latch were excluded from the analysis. Their inclusion is discussed in [14].

Another property related to critical loops is that in verification, we need only to verify simple long paths, short paths, and loops. A long path or short path is simple if it contains no cycles, and a loop is simple if it is free other internal cycles. If these paths are error-free, then all other (composite) paths must also be error-free. We also need not verify any loops that contain other internal cycles. This is proven by the following theorem:

Theorem 4 If the timing constraints of all simple long paths, short paths, and loops in a circuit are satisfied, then the timing constraints of all other paths will also be satisfied.

Proof: To prove this, we show that the constraints implied by a path containing an internal cycle are covered by independently verifying the cycle-free path and the separated cycle. Without loss of generality, we consider the path shown in Figure 17, a critical long path with an internal cycle. For this to be a critical long path, the constraints shown in Table 4 must be satisfied. However, the constraints shown will also be satisfied as long as

$$P_1 = L_0 \rightarrow \dots \rightarrow L_i \rightarrow \dots \rightarrow L_j \rightarrow L_l \rightarrow \dots \rightarrow L_m$$

is not a violated critical long path and

$$P_2 = L_i \rightarrow \dots \rightarrow L_j \rightarrow L_{j+1} \rightarrow \dots \rightarrow L_k \rightarrow L_i$$

is not a violated critical loop. Constraints (0)-(j), and (l)-(m) are required in order for P_1 to be a critical long path, and constraint (z) is necessary to ensure that it is satisfied. Constraints (i+1)-(k+1) will be satisfied if P_2 is a satisfied critical loop. The remaining constraints, (k+2)-(k+j-i+1), duplicate constraints (i+2)-(j) and are also satisfied when P_1 is a satisfied critical path. The argument is similar when more than one internal cycle is present and for critical short paths and critical loops that contain internal cycles. One notable difference is that the constraints implied by critical short paths are in terms of the minimum-value variables ($a_i, d_i, \delta_i, \delta_{ij}$), instead of the maximum-value variables used to describe the critical loop constraints. However, we know that the short-path loop constraints are guaranteed to be satisfied whenever the long-path loop constraints are satisfied, which guarantees that short paths containing cycles need not be verified. \square

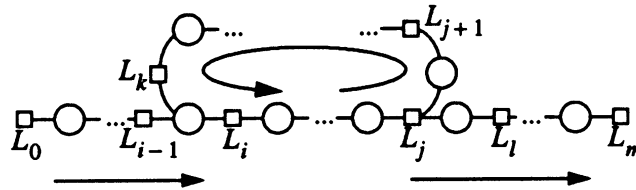


Figure 17: Critical long path with internal cycle

7 Critical Path Verification by Relaxation and Path Extraction

This section discusses relaxation-based approaches to timing verification in light of the Critical Path Method and the graph model of Section 3.2. These approaches can be considered to verify all three types of critical paths *implicitly*, as no path information is carried through the verification. As a result, we also present CPM-based methods for extracting explicit critical path descriptions from the results of these verifications and examine problems that arise when violated loops are present.

7.1 Timing Verification by Constraint Relaxation

The arrival and departure time equations of Table 3 form a system of equations which can be solved by relaxation until a fixed point is obtained. A commonly-used procedure is shown in Figure 18. If the initial arrival times are initialized to $a_i = A_i = -\infty$, then this relaxation algorithm simulates the startup timing of the circuit, with each iteration simulating one clock cycle. If multiple solutions exist for the arrival and departure equations, these initial values ensure that the most physically meaningful solution is found. Once the arrival and departure times for all synchronizers have been determined, these times are checked to determine whether the setup and hold time constraints are satisfied.

If loop constraint violations exist, they cause the departure times around the loop to increase without bound. The iteration count test at the end of each pass is to detect this “runaway” condition. If we modify the algorithm to check arrival times in each iteration, loop constraint violations will be detected as setup violations.

```

for all synchronizers
    initialize arrival times a[i], A[i]
end for
repeat
    for all synchronizers
        D[i] = max (A[i], E' [i] + Q[i])
        d[i] = max (a[i], E' [i] + q[i])
    end for
    for all synchronizers
        A[i] = maxj (D[j] + Delta_Hat[i][j])
        a[i] = minj (d[j] + delta_Hat[i][j])
    end for
until no arrival or departure times have changed or a maximum number of iterations have passed
for all synchronizers
    check setup constraint: A[i] <= Tc - S[i] + q[i]
    check hold constraint: a[i] >= H[i] + Q[i]
end for
    
```

Figure 18: Sample Relaxation Verification Algorithm

$D_0 = E'_0$	(0)
$D_1 = A_1 = D_0 + \hat{\Delta}_{0,1}$	(1)
...	...
$D_i = A_i = D_{i-1} + \hat{\Delta}_{i-1,i}$	(i)
...	...
$D_j = A_j = D_{j-1} + \hat{\Delta}_{j-1,j}$	(j)
$D_{j+1} = A_{j+1} = D_j + \hat{\Delta}_{j,j+1}$	(j+1)
...	...
$D_k = A_k = D_{k-1} + \hat{\Delta}_{k-1,k}$	(k)
$D_i = A_i = D_k + \hat{\Delta}_{k,i}$	(k+1)
...	...
$D_j = A_j = D_{j-1} + \hat{\Delta}_{j-1,j}$	(k+j-i+1)
$D_l = A_l = D_j + \hat{\Delta}_{j,l}$	(l)
...	...
$D_{m-1} = A_{m-1} = D_{m-2} + \hat{\Delta}_{m-2,m-1}$	(m-1)
$A_m = D_{m-1} + \hat{\Delta}_{m-1,m}$	(m)
$A_m \leq T_c - S_m$	(z)

Table 4: Constraints implied by path in Figure 17

Relaxation algorithms have been observed to converge to solutions very rapidly. A performance bound on the algorithm was reported by Szymanski [9], who observed that relaxation of the equations corresponds to a Bellman-Ford solution of a simple linear program. This terminates in at worst $O(|L||E|)$ time, where $|L|$ is the number of latches in the circuit and $|E|$ is the number of edges. If a solution exists, it will be found in at most $|L|$ iterations, so that $|L|$

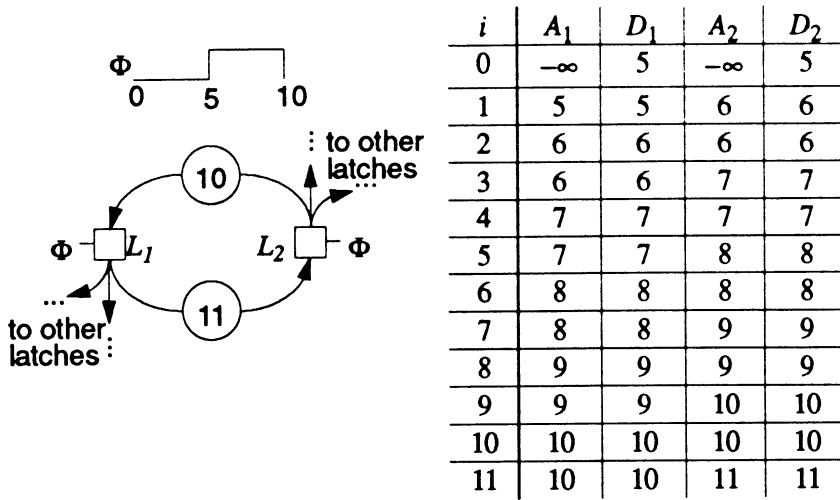


Figure 19: Effects of Violated Loops on Relaxation

should be used as the iteration limit in Figure 18. Each iteration involves looking at up to $|E|$ edges. Since $|E|$ is at most $|L|^2$, the worst-case performance of the relaxation algorithm is cubic in the number of latches.¹

Since the relaxation verification algorithm is essentially a variant of the Bellman-Ford algorithm, the heuristic presented in Section 6 can again be used to accelerate the detection of violated loops. But whether or not we use the acceleration heuristic, when a loop is violated there is no meaningful set of arrival and departure times for the circuit. The next section examines one possible approach for defining an easy to calculate and well-defined set of such times.

7.2 Clipping Departure Times during Constraint Relaxation

If violated loops are present, the simple relaxation algorithm (with no extra heuristics) requires a full $|L|$ iterations to determine that no feasible solution exists to the arrival and departure time equations. However, as shown in Figure 19, we can often identify that a timing problem exists much earlier. Embedded in a larger circuit, the loop shown has a delay slightly larger than the cycle time allows. As a result, relaxing the arrival and departure time equations causes the times to steadily increase without bound. Since no solution exists, relaxation will detect an error after $|L|$ iterations. But we can identify that a problem exists in the 11th iteration, when the arrival time at latch L_2 increases enough to cause a setup violation. How soon this happens in general depends on the size of the loop violation. Observe that the arrival and departure times increase by the amount of violation every $|P_{\text{loop}}|$ iterations, where $|P_{\text{loop}}|$ is the size of the loop. This is the expected worst-case (slowest) rate of increase for the relaxation algorithms when verifying violated loops, as shown by the following analysis:

Each update of an arrival time will have the form: $A_i \leftarrow D_{i-1} + \Delta_{i-1,i} - E_{L_{i-1}, L_i}$. When all of the latches are transparent, for each latch in the loop, $D_i = A_i$, and if the loop is critical, then for all of the latches except the one being updated, $A_i = D_{i-1} + \Delta_{i-1,i} - E_{L_{i-1}, L_i}$. Combining these equations gives $A_i \leftarrow A_i + \Delta(P_{\text{loop}}) - \Lambda(P_{\text{loop}})T_c$, so the amount of increase in each update is $V = \Delta(P_{\text{loop}}) - \Lambda(P_{\text{loop}})T_c$, the amount of the loop violation. In the worst case, only one latch is updated in each iteration, so that each arrival time will increase by V every $|P_{\text{loop}}|$ iterations. \square

1. A similar bound on performance can be obtained by observing that the relaxation procedure performs the same time calculations as the path extension algorithm presented in Section 8, and, for circuits free of violated loops, the same times will be produced in each iteration. If critical loops are violated, the algorithm will not detect them until the relaxation procedure has performed a full $|L|$ iterations to extend the longest possible simple paths in the circuit. Without the path information bookkeeping, each iteration requires $O(|E|)$ time, producing the worst-case complexity $O(|L||E|)$.

We can stop the unbounded upward growth of arrival and departure times by modifying the latch model presented in Table 3. The original model allowed signals to depart at any time after the latch’s enabling event, even after the latch was re-closed. If instead we use the following model: $D_i = \min(\max(A_i, E'_i + Q_i), L'_i + Q_i)$, signals can only leave the latch in the interval between the device’s enabling and latching events. For each latch, this adds a min node to the late signal constraint graphs illustrated in Figure 8. The modified graph structure is shown in Figure 20-a. Late departure times are thus clipped to occur no later than the device’s latching event.

In most cases, this will cause the relaxation to stop after fewer iterations, but still suffers from the worst case performance when V , the amount of violation, is small. The introduction of the min function does not eliminate the possibility of positive-weight cycles existing in the graph, but it does prevent them from causing variables to increase without bound. When a positive-weight cycle exists, the min nodes “short-circuit” it, effectively breaking the cycle when it no longer produces the smallest input to the min node. Figure 21 illustrates how signal arrival and departure times step forward until the clipped solution is found.

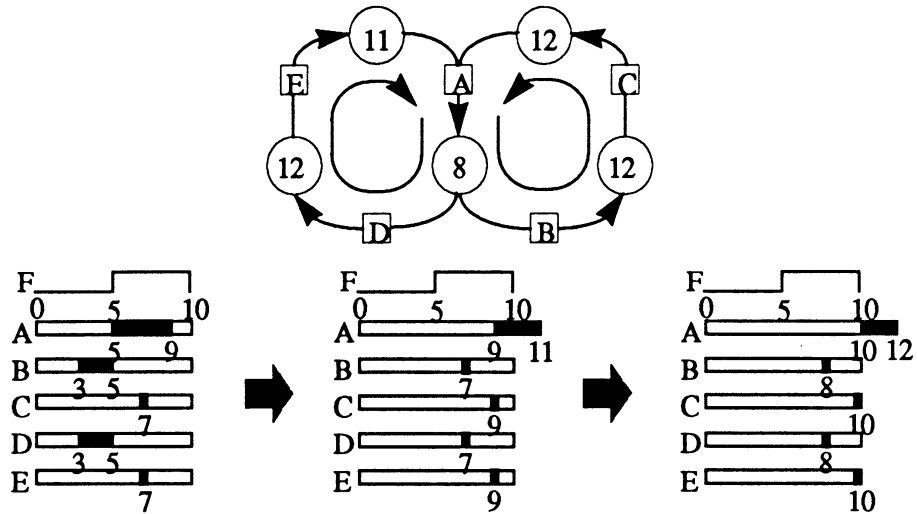


Figure 21: Stable Arrival Times for Violated Loops

A related benefit of the alternate synchronizer model is that it eliminates certain setup violations that might be considered “artificial”. For example, in the circuit shown in Figure 22, the setup violation on latch L_2 falsely causes a violation to appear on latch L_3 . If instead we clip the departure from latch L_2 , the violation on latch L_3 does not appear.

The new model also has implications on the path definitions presented in Sections 4-6. Since the source node is now connected to the graph through latch events as well as enable events, the new model allows paths to begin on both latching and enabling events. As will be seen in Section 7.4, this greatly complicates the analysis, and probably does not correspond to a designer’s intended timing behavior.

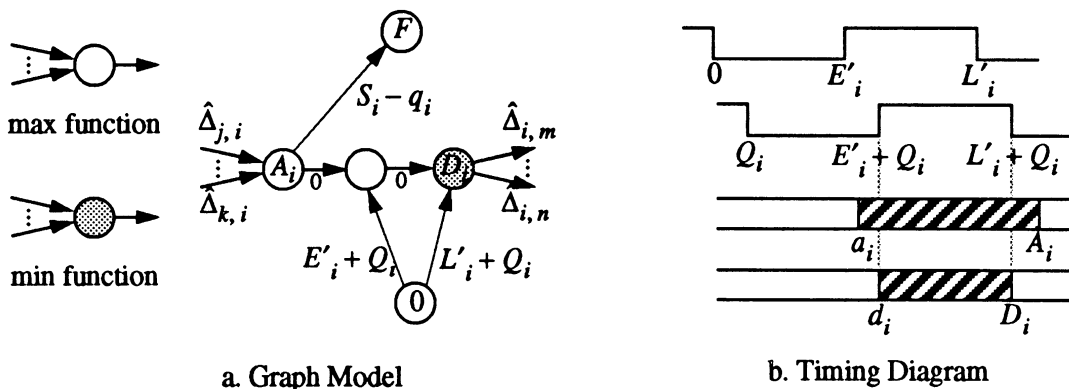


Figure 20: Modified Latch Model

If a loop in a circuit is violated, relaxation of the new model can require a potentially arbitrary number of iterations to stabilize at a fixed point solution. This rarely occurs in practice, however, and only occurs when the amount of the loop violation is made arbitrarily small. If we know in advance which latches lie on violated critical loops (we usually do not), then we can modify the relaxation procedure to quickly obtain a timing solution in a bounded number of iterations. We still use the same relaxation algorithm, but with different initial values. For each latch lying on a violated loop, we set the initial late arrival time to coincide with the latching event time. The late arrival times for all other latches are still initialized to their enable event times. The relaxation will converge in at most $|P_{\max}|$ iterations, where P_{\max} is the longest path in the circuit. Iterations will go at most once around each violated loop, since at least one latch on every violated loop will have its departure time clipped to the latch event, so that during the iterations, its departure time value will remain unchanged.

7.3 Identifying Critical Paths in the Absence of Violated Loops

In most cases, the relaxation verification algorithms are quite fast, primarily due to the typically short path lengths and the simple calculations performed in each iteration. As a result, in most cases we will first perform a quick relaxation-based verification and then, when necessary, extract critical path information from the results. This section and the next present methods for solving this problem. First we consider the case where no loops in the circuit are violated. In the next section, we discuss the more complicated case which arises for violated loops.

If a circuit is free of violated loops, then signals at each latch will have well-defined arrival and departure times using the original timing models of Table 3. As a result, it is possible to calculate slack and float values with respect to these arrival times. The circuit shown in Figure 24 contains a single topological loop, whose timing is satisfied for the clock schedule shown. Figure 24 shows the late signal graph for this circuit, with the corresponding actual and required event times marked by each node. The actual event times were calculated using the simple relaxation algorithm of Figure 18. The setup times for all latches are zero.

Since there is a topological loop in the circuit, the required times must also be calculated by relaxation. An algorithm for doing so is shown in Figure 23. Required time values are computed at each node and are propagated back through the graph until all have stabilized. These values will stabilize in at most $|N|$ iterations, where $|N|$ is the number of nodes in the graph. This relaxation will always stabilize, producing a set of values that satisfy the required time equation given in equation (2). This is guaranteed by the fact that the only thing which can prevent such a solution is

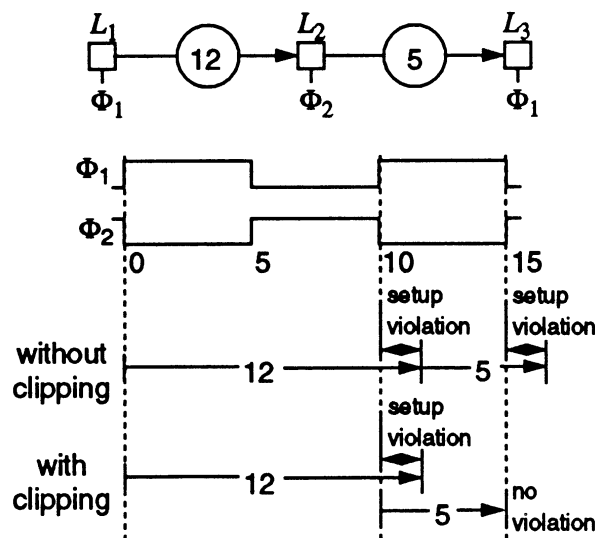


Figure 22: Clipping to Eliminate False Setup Violations

```

for each node v
    set r(v) = ∞
end for
set r(F) = Tr
repeat
    for each node v
        update r(v) ← minu ∈ S(v) (r(u) - wu,v)
    end for
until r(v) values have stabilized (at most |N| iterations)
    
```

Figure 23: Required Time Calculation Algorithm

a positive-weight cycle in the constraint graph, and the fact that such a positive-weight cycle can only exist when a violated loop is present.

Examining the late signal graph, we see that the sink has an actual event time of 11 (due to latch L_5) and a required time of 10 (due to the clock schedule). The resulting slack at the sink is -1, and tracing back along the arrows with -1 float, we see the critical path marked in bold.

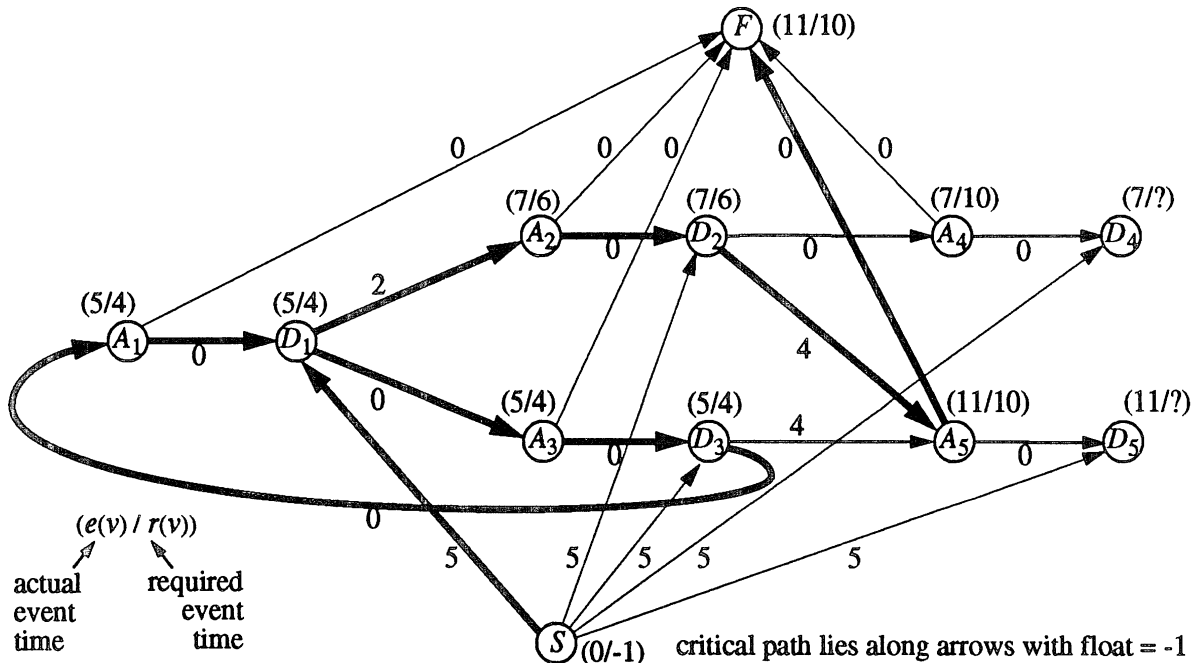


Figure 25: Late Signal Graph Free of Violated Loops

Examining this path, we see that it is in fact a composition of a critical long path (nodes S, D_1, A_2, D_2, A_5, F) and a critical loop (nodes A_1, D_1, A_3, D_3). The two paths touch at node D_1 , causing the nodes on the critical loop to have the same slack as the nodes on the critical long path.

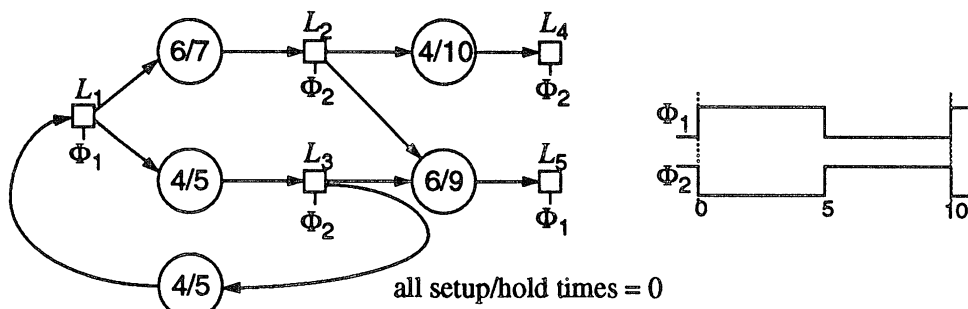


Figure 24: Sample Circuit for Path Extraction Example

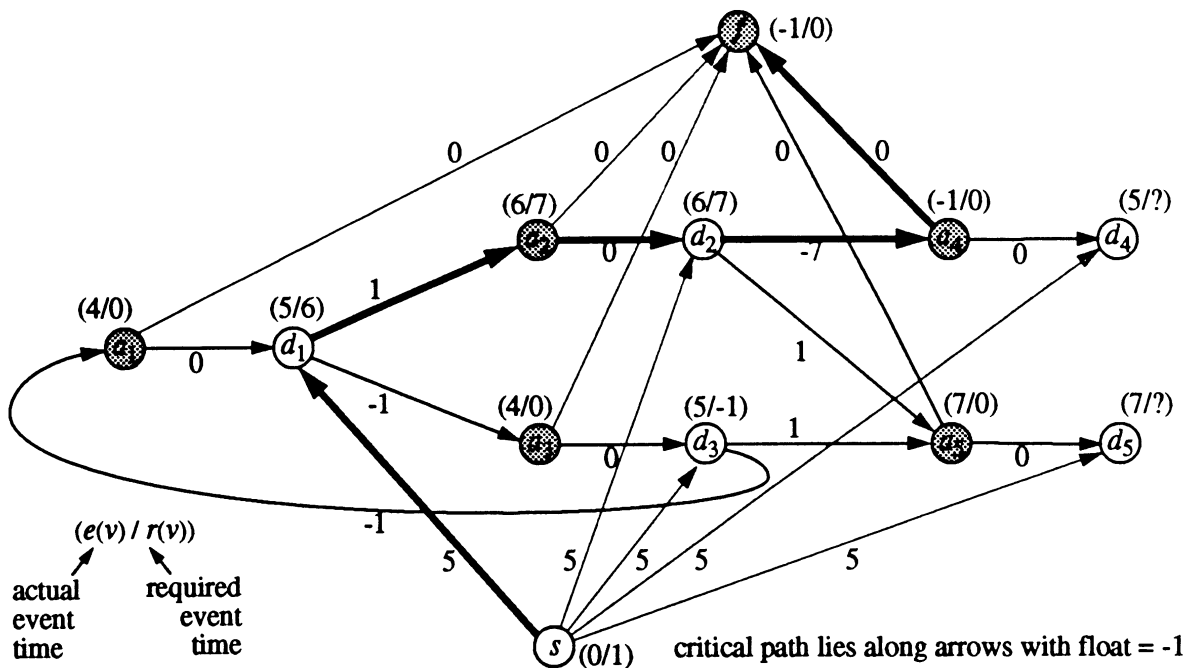


Figure 27: Early Signal Graph

The critical long path can be identified using a depth-first search of the critical arrows, beginning at either the source or sink nodes. Beginning at the sink node, we search back along critical arrows until we reach the source, and the path we obtain is a critical path. If an arrow or node is encountered more than once during the expansion, that subpath is abandoned. This procedure is formally described in Figure 26.

The algorithm in Figure 26 can also be used to identify critical short paths. The short path graph for Figure 24 is shown in Figure 24. The critical path lies along the darkened arrows.

Note that neither of these slack calculations detected the presence of the critical loop containing L_1 and L_3 . This loop prohibits us from increasing the total delay $\Delta_{1,3} + \Delta_{3,1}$. We observe that the critical loop in Figure 24 consists of nodes and arcs with slack and float equal to -1, the most negative slack in the circuit. However, this may not always be the case. We do know that a critical loop will always consist of a set of nodes connected by controlling arrows, and that a result of this is that all of the nodes and arrows in a critical loop will have the same slack and float values. We

```

begin
  initialize stack ST;    // ST will hold the identified path
  trace_long (F);        // F is the sink node
end

procedure trace_long (vertex v)
  if v is marked
    abort search
  else if v is the source node
    critical path found, dump stack ST
  else
    mark v
    push v onto stack ST
    for all critical arrows (u'v) fanning into v
      trace_long (u)
    pop v from stack ST
    unmark v
  end if
end procedure trace_long

```

Figure 26: Long Path Extraction Algorithm

can search over all sets of “equi-float” errors to find critical loops, but a more practical approach may be to simply search the subgraph consisting only of controlling arrows and the nodes they connect. Either case will provide a polynomial-time procedure for identifying each loop in a circuit, although the total number of such loops may be exponential.

7.4 Identifying Critical Paths in the Presence of Violated Loops

When violated loops are present, the above procedures for identifying and extracting paths become much more complicated. If even a single loop is violated, the arrival and departure time equations for a circuit will have no solution. The most reasonable way to force a solution is to use the model of Figure 20, which clips departure times to obtain stable values. This approach is illustrated in Figure 24. Figure 24 shows the late signal actual graph for the circuit of Figure 24, but with the delay from L_3 to L_1 ($\Delta_{3,1}$) increased to 6. This causes the critical loop containing L_3 and L_1 to be violated, so that no stable solution will exist for the original model. Using the modified latch model of Section 7.2, the event times stabilize to those shown in the figure. However, when we attempt to calculate required times, the positive-weight cycle is again a problem, as it prevents the required time relaxation from stabilizing.

Some information can still be obtained by identifying controlling and non-controlling arcs; critical paths can then be traced out along these arcs. Controlling arcs are drawn in boldface in Figure 24. A number of paths appear violated, but as we know from Figure 24, most of these paths are satisfied when the critical loop is satisfied. This is a severe problem, since it is important that we distinguish between false paths and those which truly cause timing errors. Also note that there are no loops of controlling arcs in the clipped-departure graph (although there are workarounds for this problem).

The apparent dilemma is this: without clipping, we have no defined solution when loops are violated. With clipping, we reach a well-defined solution quickly, but if loops are violated, the slack information is difficult to use and identifies many false violations. Thus the clipped timing model actually *limits* our ability to find critical paths.

7.5 A Strategy for Extracting Critical Paths

The dilemma of the previous section can be resolved by a simple modification to our verification procedure. We can first attempt to calculate the arrival and departure times using the modified relaxation algorithm of Section 7.1. If the loop-detection heuristics detect a violated critical loop, or if the algorithm fails to converge, then we can use the modified Lawler’s algorithm to calculate $T_{c,loop}$. Rerunning the relaxation at $T_{c,loop}$, we can obtain a list of all critical long paths and loops for this cycle time. This will be a subset of the critical paths at the desired cycle time, but will probably be sufficient for the purpose of incremental design optimization.

In Section 8, we present a procedure for identifying all of the critical loops in a circuit for arbitrary clock schedules, including those with cycle times below $T_{c,loop}$.

8 Critical Path Verification by Path Enumeration

In the previous section, we presented an approach for identifying and verifying critical paths based on CPM-style approaches. Actual event times were calculated in a forward relaxation through the network, and then required times were calculated in a reverse relaxation. This procedure worked well for circuits that were free of loops or for which all

```

for all storage elements in circuit
  mark part
  for each controlling fanin to part
    if part is the desired part, a loop was found.
    else if part has mark, we found subloop-abort search
    else expand controlling fanins
  end for
  unmark part
end for

```

Figure 28: Loop Extraction Algorithm

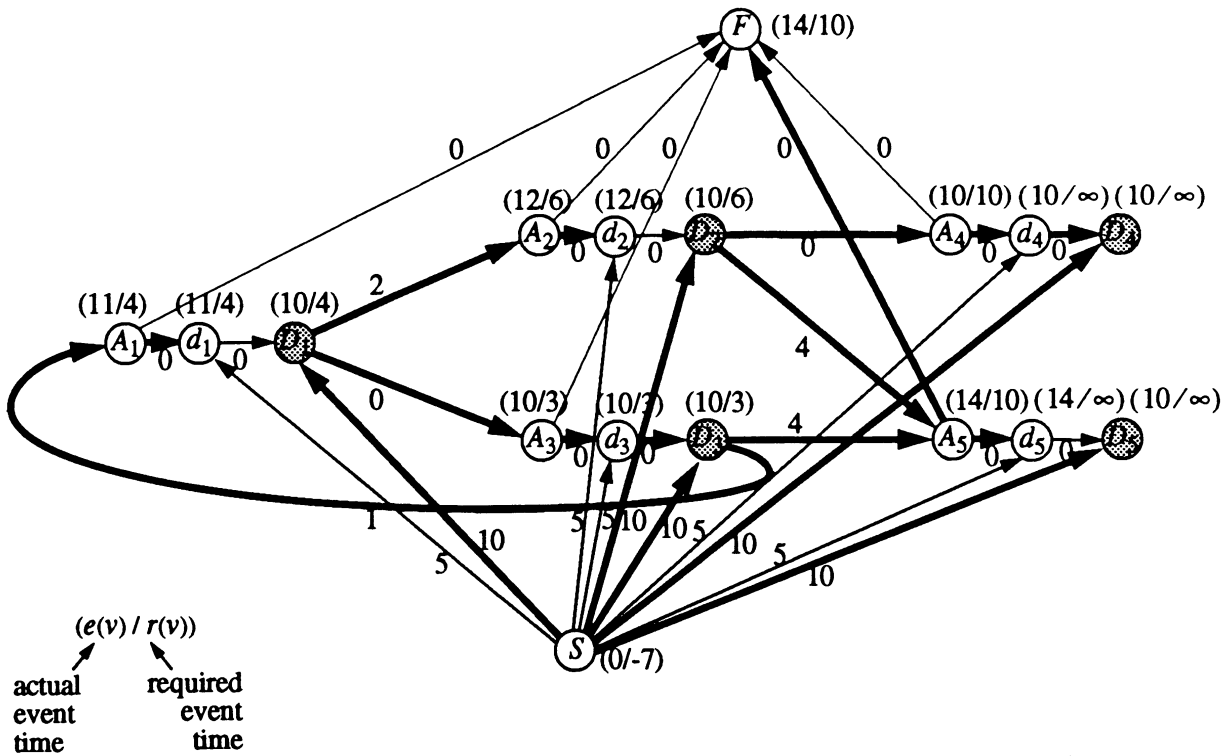


Figure 29: Clipping Departure Times to Block Violated Loops

critical loops could be guaranteed satisfied. However, it was seen to break down when violated loops were present, since either the arrival and departure times were undefined or a large number of paths falsely appeared violated.

This section describes an alternate approach based on enumerating all of the critical paths in a circuit, using heuristic information to reduce the number of paths examined. These paths can then be checked for timing violations. This involves carrying more information forward through the timing verification process, which complicates and slows the verification procedure. However, it will also prove to be a more robust procedure for obtaining critical path information in the presence of violated critical loops.

Since the path enumeration approach also must perform timing verification, it is necessary to show that verifying the critical paths in a circuit is equivalent to verifying the timing constraints of Table 3.

Theorem 5 If all critical paths are satisfied, then the timing constraints of Table 3 are also satisfied.

Proof: This follows trivially from the critical path definitions. Since a critical long path is defined to cause the largest setup violation in a circuit, if the path is satisfied, then the setup constraint must be satisfied for all other paths. Similarly, satisfaction of the critical short paths ensures that all hold constraints will be satisfied. The final constraint is the loop constraint, which requires that the arrival and departure time equations have a stable solution. Such a solution will always exist as long as no positive-weight cycle is present in the late or early signal graphs. (Recall that it is sufficient to show that no such cycle exists in the late signal graph) This is equivalent to the condition for a violated loop constraint presented in Section 6. Thus if all critical loops are satisfied, no positive weight cycle can be present in a circuit and the arrival and departure time equations will have a stable solution. Note that we have assumed that the pulse-width constraints (1,2) are trivially checked and satisfied.

8.1 Path Extension Algorithm

Figure 30 sketches an algorithm that constructs all the critical long paths and loops in a circuit. The algorithm which identifies critical short paths is trivially similar. A path which partially satisfies the criticality constraints of Sections 4, 5, and 6 was called a *candidate* path. Although candidate paths may not be critical, every critical path must also be a candidate path. Each iteration in the algorithm generates successively longer candidate paths, with the i -th iteration generating all the candidate paths of length i . This is done by extending paths from iteration $i-1$. The

```

create an initial null candidate path at the output of each latch
copy paths to the inputs of all fanout latches, marking them active
while there are active paths on latch inputs
  for each latch with an active path on its inputs
    identify the latest arriving signal(s) on the latch inputs
    if (this signal is active and
        the signal is not blocked by the clock and
        the associated path is not a cycle) then
      extend the path to include the current latch
      propagate it on the latch outputs and mark it active
    end if
    deactivate all signals and paths on the latch inputs
  end for
  copy new paths to the inputs of all fanout latches, marking them active
end while
for each latch in the circuit
  check timing constraints paths at latch inputs
end for

```

Figure 30: Long Path/Loop Extension Algorithm

extensions are performed by identifying the most critical input to each latch and if the associated input path is active, the path is extended as long as (1) the resulting extended path is not a cycle, and (2) signal flow through the latch is not blocked by the clock. If parallel candidate paths are present, they are grouped together into a treelike structure and all are extended in successive iterations. Paths are deactivated at the end of each iteration to ensure that only paths of length $i-1$ are extended in iteration i . Any path which is not a candidate is also not critical and need not be extended.

The maximum number of iterations required by the path extension algorithm is bounded by Theorem 4, which shows that it is not necessary to examine simple paths, which are free of internal cycles.

Lemma 6 The longest simple critical path in a circuit can be no longer than $|L|$, where $|L|$ is the number of latches in the circuit.

Proof: Since a simple critical path can contain no internal cycles, each latch in a circuit can occur at most once in such a path, with one exception: a critical loop will contain exactly one duplicated latch (since $L_m = L_0$). Thus the longest possible simple critical path is a critical loop that extends through every cell in the circuit and which includes exactly one duplicated cell. The size of this path is $|L|$. \square

The path extension algorithm verifies that all critical paths up to length $|L|$ are satisfied. The algorithm iteratively looks at all candidate paths of length 1, length 2, ..., length i , length $i+1$, ... up to the paths of length $|L|$. Although the total number of paths is exponential, they can be easily pruned to the one path (or group of parallel paths) per latch. Each iteration then will examine at most $|E|$ such paths, where $|E|$ is the number of edges (latch-to-latch connections) in the circuit. That all candidates are examined is guaranteed by the following theorem:

Theorem 7 If a path $P = L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_m$ is a candidate path of length m that ends at latch L_m , then $P^* = L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{m-1}$ must also be a candidate path.

Proof: This follows directly from the critical path definitions. The constraints that are implied by P being a candidate path are a proper superset of those that imply that P^* be a candidate path. \square

Each iteration of the verification algorithm looks at the $|E|$ edges in the circuit and identifies the candidate paths into each latch in the circuit. These candidate paths can then be checked for possible setup, hold, or cyclical timing errors. The worst case performance of this algorithm is $O(|L||E|C_{loop})$, since it expands paths up to length $|L|$ and each iteration involves looking at as many as the $|E|$ edges which interconnect the latches in the circuit. C_{loop} is the cost of manipulating and extending paths in each iteration and is discussed in Section 8.2. A more practical estimate of performance is obtained by observing that the algorithm needs only to run for $|P_{max}|$ iterations, where P_{max} is the longest candidate path in the circuit. Although the worst case value of $|P_{max}|$ is $|L|$, in practice it is much lower, and can usually be bounded by a small constant.

The algorithm for identifying and verifying short paths has the same form as the long path algorithm, but with the loop constraint checks removed. This algorithm also finds successively later arrival times at latches until they finally stabilize. However, making the early arrival times later can mean that initial hold time violations can be later covered by a longer path. An example of this situation was shown in Figure 14. This circuit will work correctly for the timing shown, although the algorithm will initially find a hold time violation on L_3 when it examines the path $L_2 \rightarrow L_3$. This error will occur during the startup phase of the circuit, but the steady state timing will correspond to the error-free path $L_1 \rightarrow L_2 \rightarrow L_3$.

Each iteration of the algorithm can thus be viewed as a simulation of one clock cycle of the circuit's startup behavior. In some cases, we may choose to ignore these transient hold violations much as we ignore the data flowing through a pipeline before it has reached its steady state. If instead we report them, we will detect all timing errors that will occur during the startup of the circuit.

8.2 Path Storage and Manipulation

To fully analyze the runtime complexity of the path extension algorithm, the complexity of manipulating paths and searching them for loops must be considered. Depending on the representation chosen, this cost can increase with path length, and can greatly extend runtimes where paths are large.

A number of possibilities exist for efficiently storing the paths identified in each iteration. The first is to represent paths with recursive list- and tree-like data structures. This allows paths constructed in previous iterations to be reused to construct paths in current and future iterations. Paths stored as trees can easily represent multiple parallel paths to a latch. Each iteration of the algorithm then involves constructing new paths of length i from existing paths of length $i - 1$. This allows each path to be extended in constant time, and since at most $|L|$ new paths will be constructed in each of at most $|L|$ iterations, the maximum memory utilization will be $O(|L|^2)$. Using $|P_{\max}|$ as a bound on the number of iterations, this reduces to $O(|L||P_{\max}|)$. However, each iteration also involves searching the path structures to determine whether a new path contains a loop. Each examination involves looking into a tree at most $|P_{\max}|$ levels deep, with each level branching to up to $|L|$ subpaths. If there are many parallel critical paths, this can add an exponential cost to each iteration, since the path tree can describe a potentially exponential number of paths. However, if we mark subpaths during the path traversal, we can reduce the lookup cost to at most quadratic (since the data structure has at worst quadratic size). In either case, the lookup cost is low when the trees are relatively short and the number of parallel paths is small. We believe this to generally be the case, and observe it to be true for the examples presented in Section 9.

A second approach which reduces lookup cost is to represent paths as bit vectors, where each bit corresponds to a latch in the circuit and is set to 1 if the latch is in the path and 0 if it is not. Here we have sacrificed some information about the path structure (which can be regained in a post-processing step), but the important task of determining whether or not a new path is a loop can be determined by a constant-time lookup in the bit vector. However, we now must pay more to update each path, since extending each path involves copying $|L|$ bits. For large circuits, this can involve an inordinate amount of storage and, unlike the previous approach, the cost of manipulating bit vectors is constant and is not reduced when the path lengths are small.

Other schemes are conceivable, including representing paths with linked lists or hash tables. However, for our test implementation we chose the tree-based approach for its low overhead per extension and our belief that paths in the circuit have limited length.

8.3 Path Extension Example

A simple illustration of the path extension algorithm for late signals is shown in Figure 31. The circuit being analyzed is the circuit from Figure 24, but with the loop violation that made slack-based path extraction difficult. Each column in the table shows paths produced in the corresponding iteration. Late arrival times associated with each path are also shown. Observe that the algorithm correctly identifies the violated loop involving L_1 and L_3 , and that it also

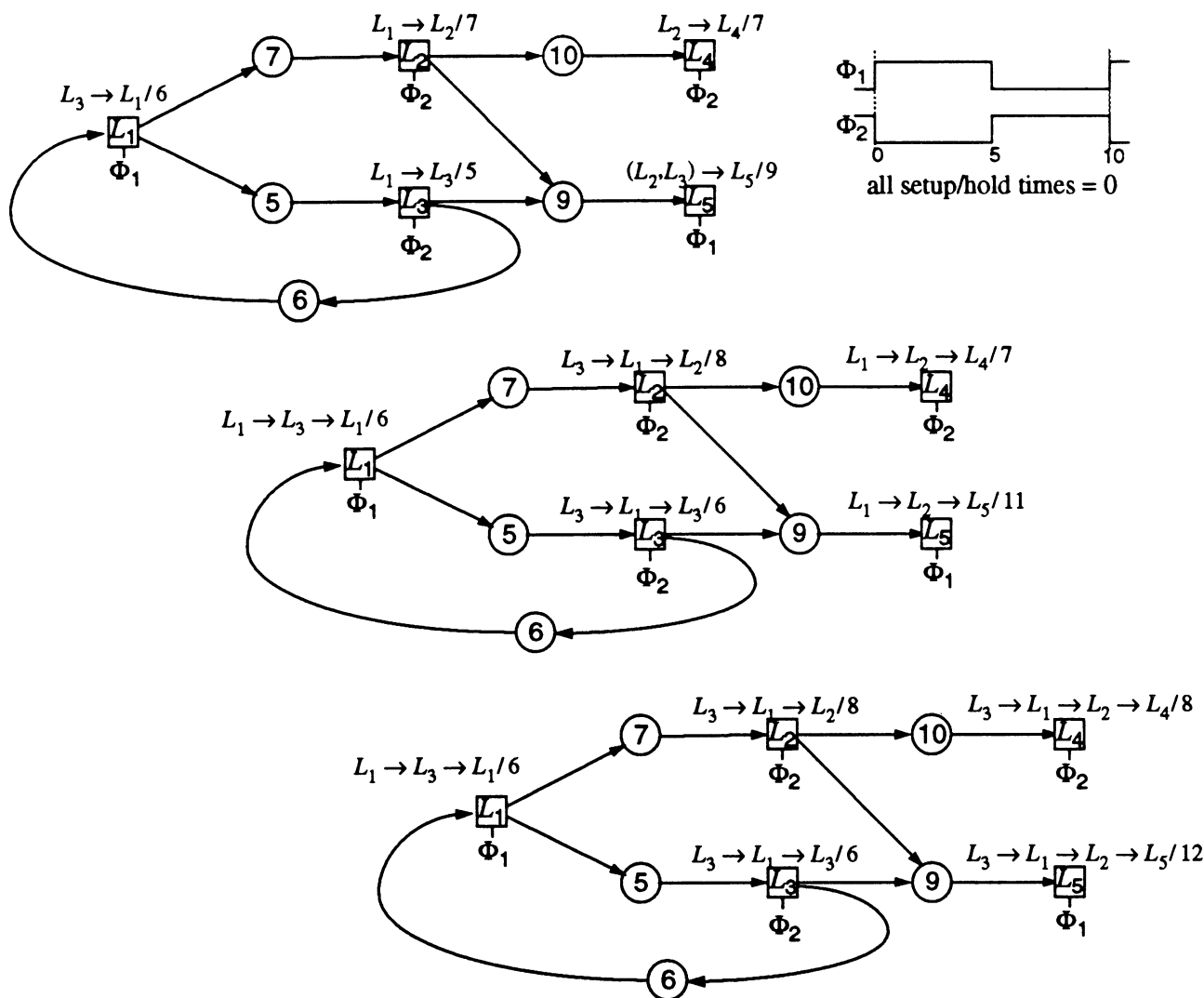


Figure 31: Long Path and Loop Extension Example

identifies the critical long path $L_3 \rightarrow L_1 \rightarrow L_2 \rightarrow L_5$. Note the observed violation on this path correctly includes the increased delay from L_3 to L_1 .

Figure 31 illustrates the short path extension algorithm for the same circuit. Early arrival times associated with each path are shown. In the first iteration, the hold violation on L_4 appears due to the shortness of the path $L_2 \rightarrow L_4$; however, the violation disappears in the next iteration due to the extended path $L_1 \rightarrow L_2 \rightarrow L_4$. This indicates that a transient hold violation is present on L_4 , a condition which may or may not be acceptable to the system designers.

After considering these examples, a few observations can be made:

- When verifying loops, it is necessary to check the final arrival time against the original departure time that started the signal flowing around the loop. Since the original path was extended, the arrival time at the first latch could have increased, possibly masking a loop violation. To deal with this potential problem, our implementation currently keeps track of the delay and latency of each path as it is extended. When a loop is found, its total delay is checked against its latency to ensure that signals can complete the circuit around the loop in the required cycle time.
- The long and short path extensions produce separate sets of paths.
- Although violated loops may still arise in the short path extension, it is not necessary to check for them since any loop violations will show up in the late path extension.

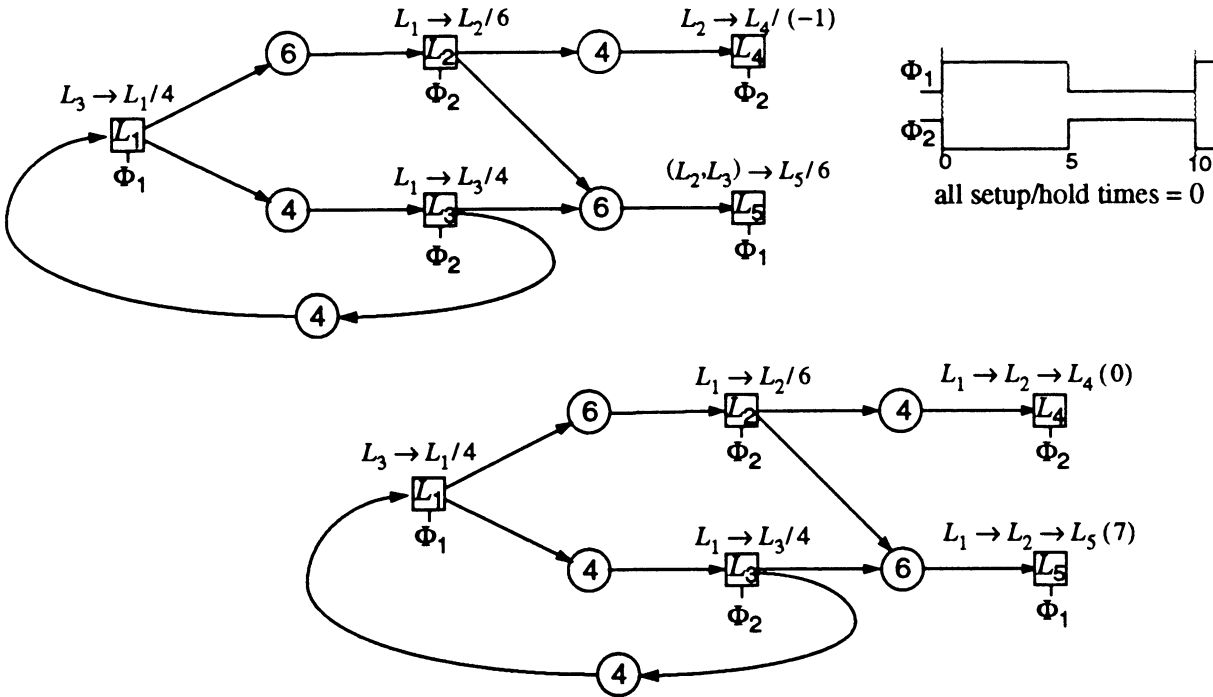


Figure 32: Short Path Extension Example

- At one point in the late signal extension, parallel paths appeared. As a result, we must allow for them in the data structures used to store path information.

9 Timing Verification Experiments

To compare the different procedures for timing verification and critical path identification, we tested our implementations on circuits in the ISCAS89 sequential benchmarks. Because the ISCAS89 circuits were originally single-phase edge-triggered circuits, we performed the transformation used by Szymanski to convert them to two-phase level-sensitive circuits [13]. Circuits were clocked with a symmetric non-overlapping 2-phase clock with 50% duty cycle. This eliminated the possibility of hold violations and made the maximum feasible cycle time $T_{c, \max} = \infty$.

Figure 33 contains a plot of the number of iterations and cpu time required to verify s15850, one of the larger benchmark circuits. As modified, s15850 contains 1396 latches and was verified at different fractions of its minimum cycle time, $T_{c, \min}$. The circuit was verified using four algorithms: the simple relaxation algorithm, the modified relaxation algorithm that clips departure time, the relaxation algorithm that checks for loops, and the path extension algorithm. Note that all algorithms require the same number of iterations until $T_c = 0.95T_{c, \min}$, at which point a critical loop is violated. As a result, the number of iterations required by the simple relaxation approach jumps to its maximum value $|L|$ and the number of iterations required by the relaxation algorithm with clipping also jumps, but quickly descends as the amount of the violation increases. The number of iterations for the path extension algorithm increases steadily as more latches become transparent and path lengths increase and the relaxation algorithm with loop checks requires only a few iterations across the entire range of cycle times shown.

As could be expected, CPU times for the path extension algorithms reflect the overhead required to manipulate the extra path information that they maintain. At large values of T_c , there is very little penalty, due to the shortness of the paths examined (evidenced by the small number of iterations). However, as T_c decreases and path lengths increase, the cost of path extension increases until eventually it becomes impractical.

It was also interesting to compare the runtime performance of the algorithms as a function of circuit size, or the number of latches in each circuit. CPU times required for the algorithms are shown in Figure 34 on a log-log plot. The

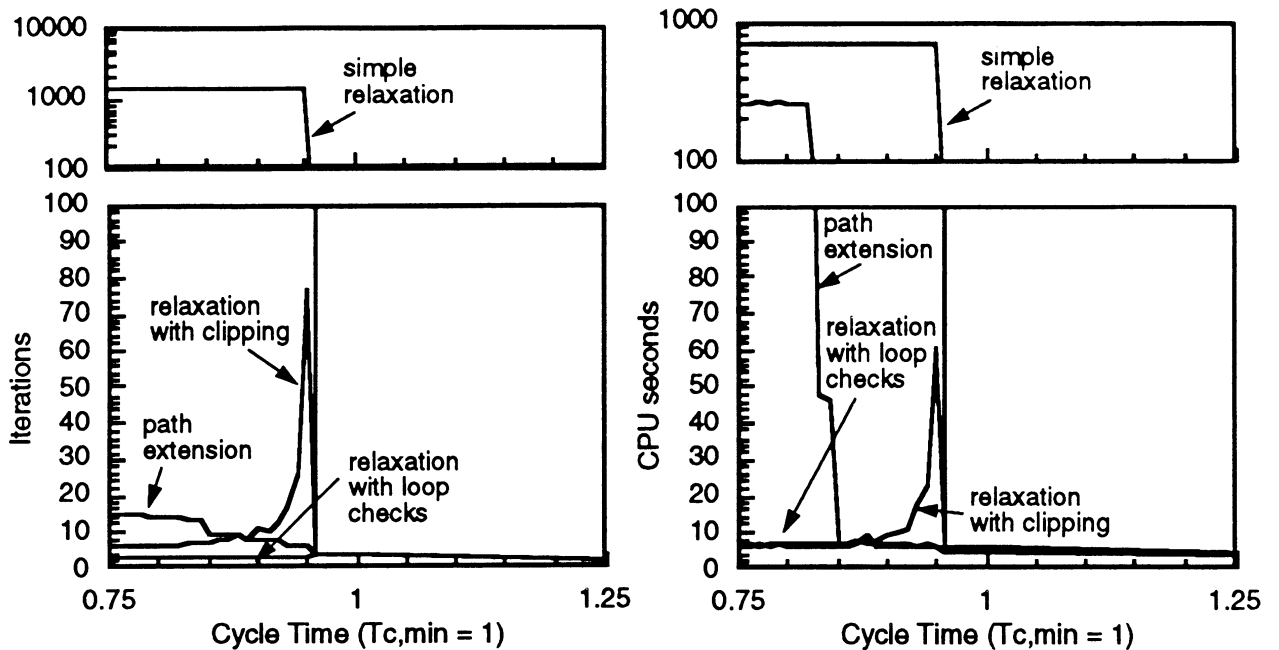


Figure 33: Iterations and CPU Time Required to Verify s15850

plot shows the result of verifying all of the ISCAS benchmarks at $T_c = 0.9T_{c, \min}$. The runtimes for each of the algorithms fell roughly into straight lines, with the slopes of the lines corresponding to the exponent in the complexity function $O(I^m)$. The simple relaxation algorithm exhibited an approximately cubic time complexity, and the other algorithms showed quadratic or subquadratic complexity. This reflected the fact that one of the factors in the cubic worst case complexity is the path length, which is bounded by a small constant in all but the simple relaxation algorithm.

To further observe the length of the longest path in each circuit, we plotted the lengths of the longest paths produced by the path extension algorithm for each of the benchmark circuits at a variety of cycle times. At $T_c = T_{c, \min}$, the longest observed paths were typically 2 to 4 segments long, but in one circuit were as long as 13 segments. As the cycle time decreased, these path lengths increased gradually, until at $T_c = 0.8T_{c, \min}$, the longest observed path was 17. As

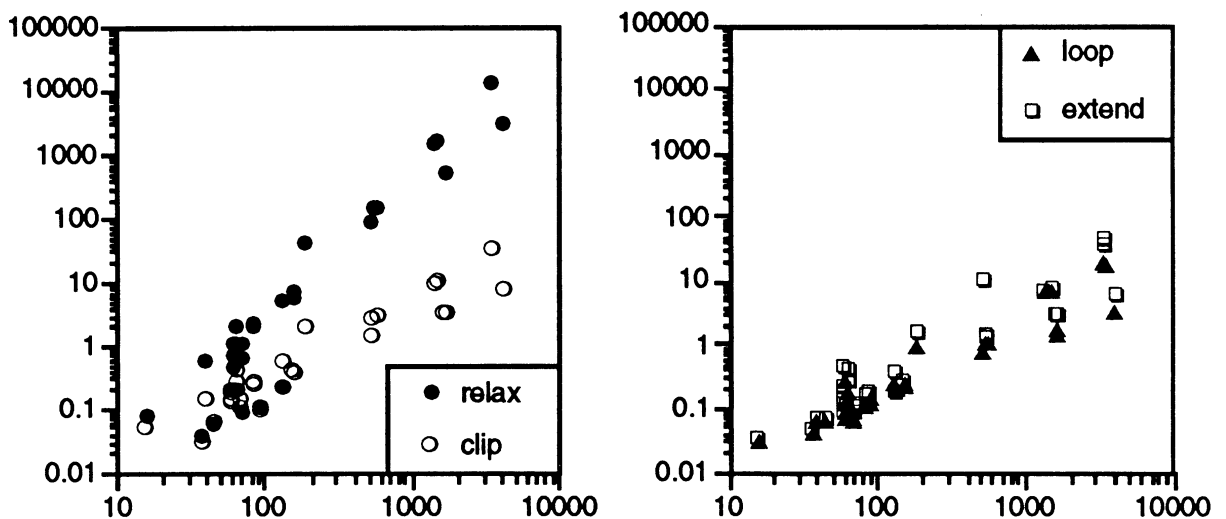


Figure 34: Runtime Performance vs. Circuit Size (Verifying at $0.9T_{c, \min}$)

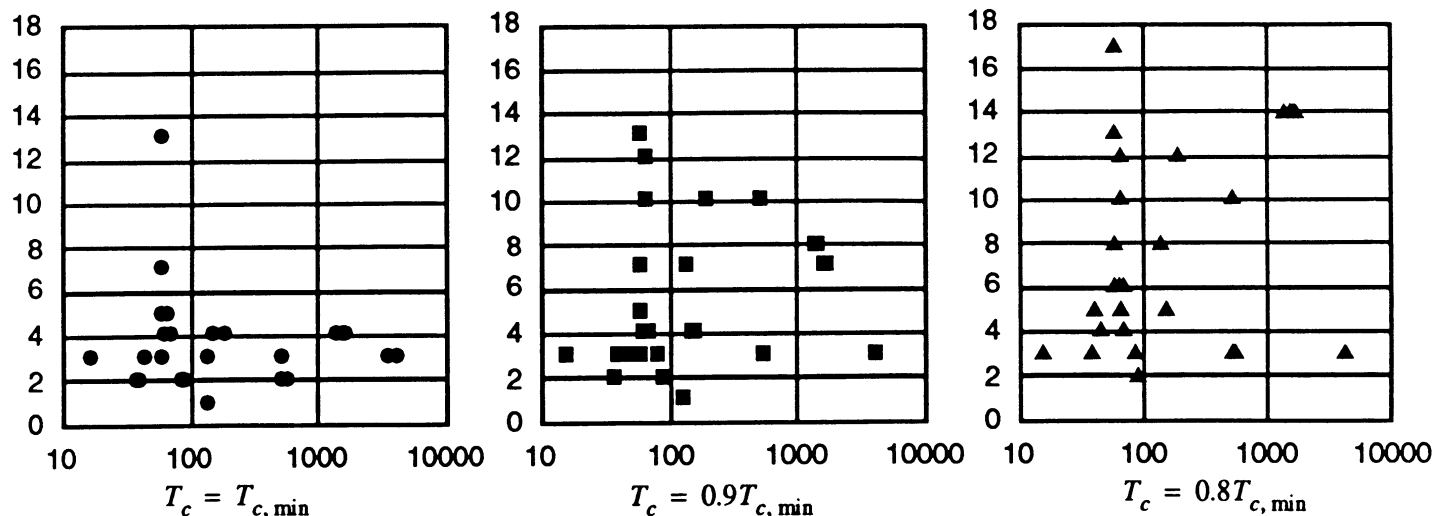


Figure 35: Maximum Path Length vs. Circuit Size (in Latches)

the plot in Figure 34 shows, these path lengths were relatively uncorrelated with circuit size and increased only slightly as the cycle time decreased.

10 Conclusions

We have discussed a pair of approaches for identifying critical paths that can extend through level-sensitive latches. The first, based on relaxation techniques, was simpler, faster, and more closely based on the original CPM-based approach to identifying critical paths in combinational circuits. However, as we saw in Section 7.4, it is difficult for these procedures to produce meaningful results in the presence of violated loops, which cause the original CPM definitions to break down. To deal with this problem, we proposed a second algorithm which constructively generates paths and can find all critical paths even when violated loops are present. This complicates the verification process, as the second algorithm carries the additional overhead of updating paths in each cycle and checking for the presence of loops in the extended paths. However, as demonstrated in Section 9, this added cost is quite moderate when the circuit is verified near its actual minimum feasible clock frequency, as at these frequencies paths are typically quite short and thus the extension overhead is small. Regardless, if the critical loops in a circuit are known to be satisfied or if the circuit is acyclic, the relaxation-based procedure should be used as it is simpler and faster.

We are currently enhancing our software tool to allow it to be used in the timing analysis for the Michigan Gallium Arsenide MIPS Processor project. This involves strengthening and streamlining our current implementation, adding the ability to extract combinational subpaths, adding a parser for Verilog circuit descriptions, and adding support for more complex delay models. We are also studying several simple delay optimization approaches based on the path definitions described in this paper.

References

- [1] T. I. Kirkpatrick and N. R. Clark, "PERT as an Aid to Logic Design", *IBM Journal of Research and Development*, March 1966.
- [2] R. B. Hitchcock, Sr., G. L. Smith, and D. D. Cheng, "Timing Analysis of Computer Hardware", *IBM Journal of Research and Development*, Vol. 26, No. 1, January 1982.
- [3] E. A. Dinic, "The Fastest Algorithm for the PERT Problem with AND- and OR-Nodes (The New Product-New Technology Problem)", *Integer Programming and Combinatorial Optimization: Proceedings of a Conference held at the University of Waterloo, May 28-30 1990, by the Mathematical Programming Society*, University of Waterloo Press, 1990.
- [4] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, 1976.
- [5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.
- [6] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, 6(1):5-35, 1991.

- [7] B. Lockyear and C. Ebeling, "Optimal Retiming of Multi-Phase Level-Clocked Circuits", *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, 1992.
- [8] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou, "Optimizing Two-Phase Level-Clocked Circuitry", *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, 1992.
- [9] T. G. Szymanski, "Verifying Clock Schedules", in *ICCAD-90 Digest of Technical Papers*, November 1992.
- [10] N. Shenoy, R. K. Brayton, A. L. Sangiovanni-Vincentelli, "A Pseudo-Polynomial Algorithm for Verification of Clocking Schemes", in *TAU '92: Proceedings of the 1992 ACM/SIGDA Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, March 18-20, 1992.
- [11] N. Shenoy, R. K. Brayton, A. L. Sangiovanni-Vincentelli, "Graph Algorithms for Clock Schedule Optimization", in *ICCAD-92 Digest of Technical Papers*, November, 1992.
- [12] 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.
- [13] T. G. Szymanski, "Computing Optimal Clock Schedules", in *Proceedings of the Design Automation Conference*, 1992.
- [14] K. A. Sakallah, T. N. Mudge, T. M. Burks, and E. S. Davidson, "Optimal Clocking of Circular Pipelines," *International Conference on Computer Design*, October, 1991.
- [15] J. Horowitz, *Critical Path Scheduling: Management Control through CPM and PERT*, Ronald Press Company, 1967.
- [16] J. D. Wiest and F. K. Levy, *A Management Guide to PERT/CPM*, Second Edition, Prentice-Hall, 1977.
- [17] K. Lockyer and J. Gordon, *Critical Path Analysis and other Project Network Techniques*, Pitman, 1991.