Critical Paths in Circuits with Level-Sensitive Latches

Timothy M. Burks, *Member, IEEE,* Karem A. Sakallah, *Senior Member, IEEE,* and Trevor N. Mudge, *Senior Member, IEEE*

Abstract— **This paper 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 cyclic constraints on the data signals at the inputs of each latch and may extend through one or more latches. Two approaches are presented for identifying these critical paths and verifying their timing. The first implicitly checks all paths using a relaxation-based solution 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 the critical paths in a circuit and then verifies that their timing constraints are satisfied. Algorithms are evaluated and compared using circuits from the ISCAS89 sequential benchmark suite and the Michigan High Performance Microprocessor Project.**

I. Introduction

THE TIMING VERIFICATION of latch-controlled circuits has been a popular subject for recent study [1]–[4]. The sequential 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.

Few designers, however, 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 [2], [5], [6], which find the minimum cycle time for a circuit; but these still provide no information about why a circuit fails to meet its desired timing. To determine this, it is necessary to know the *critical paths* in a circuit, those sections of the circuit that place the tightest constraints on its timing behavior. The concept of a critical path has been used for many years to analyze both combinational and sequential circuits [9], [10] and underlies a large number of timing-driven optimization techniques. However, these approaches assume that only edge-triggered storage elements are used, leading to a very simple set of timing constraints. If a circuit instead uses level-sensitive latches, the corresponding timing constraints are complicated by the transparent behavior of enabled latches. Designers are forced to make restricting assumptions to allow use of existing critical path-based optimization techniques; most commonly, the transparent latch properties are ignored and latches are conservatively treated as edge-triggered devices.

This paper extends the classical definition of a critical path to fully account for the timing properties of level-sensitive latches. We show that three distinct types of critical paths can arise and constrast these with the single type of critical path commonly observed in edge-triggered circuits. Each path type is described in detail, and models are presented to relate them back to classical critical path methods. We then discuss procedures for extracting these paths and analyze the factors which affect the feasibility of extraction and enumeration. We present two approaches, one based on existing relaxation timing verification methods and one which is a new algorithm that iteratively constructs possible critical paths.

Several other researchers have independently developed similar critical path definitions. Ishii and Leiserson [1] describe critical paths in the context of the *computational expansion* of a latch-controlled circuit; this is essentially an unrolled version of a cyclic circuit. A more closely related definition of critical long paths was developed by Lockyear and Ebeling [15] (and Ishii *et al.* [16]) for use in retiming applications, and more recently, Shenoy *et al.* [17] used a definition similar to ours to allocate slack among stages of pipelined circuits.

Our approach is unique in that we relate these paths directly to classical critical path-based approaches, suggesting the extension of a wide variety of existing timing-driven design approaches to circuits with level-sensitive latches. We also consider the effects of clock skew in our formulation and consider practical issues in the reporting and application of critical path information. Section II presents the necessary definitions, including a review of the critical path method (CPM) and the timing models we use. The three types of critical paths that can arise are presented in Section III. Section IV presents the relaxation-based approach and discusses the extraction of critical paths from relaxation results. Section V describes the algorithm for iteratively constructing critical paths and describes its asymptotic complexity. Section VI concludes with the results of experiments performed using the various verification and path identification procedures.

Manuscript received June 1, 1993; revised June 17, 1994 and November 16, 1994. This work was supported in part by NSF under Grant MIP-9014058 and by DARPA under Grant DAAL03-90-C-0028. The work of T. Burk was supported by a DoD NDSEG fellowship.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA.

IEEE Log Number 9410839.

II. PRELIMINARIES

A. Critical Path Analysis

The concept of a critical path was originally introduced in the field of project management in two separate but related methods, PERT (program evaluation and review technique) and CPM (critical path method) [7], [8]. Both methods were developed in the 1950's as tools 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 single worst-case values; in PERT, job times are described by a 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 arrows, respectively. The minimum project completion time is determined by the path through the project network from source to sink that requires the longest total time. Unless otherwise noted, we refer to the source and sink nodes as nodes S and F, respectively.

The minimum project completion time can be obtained by making a single pass through the network, processing each node exactly once. Beginning with the source node and proceeding in topological order, 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 and is defined by the following equation

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

$e(u)$ 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. Setting the event time for the source node to zero, (1) defines a unique event time for the remaining nodes. The minimum project completion time is then simply the event time of the sink node.

A *critical path* in a 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 by

$$r(v) = \min_{w \in S(v)} [r(w) - t_{v,w}]. \tag{2}$$

The required time for the sink node is set to T_r, the *required time* for project completion. Required times can be calculated in a single pass that visits nodes in reverse topological order. In (2), $r(v)$ and $r(w)$ are the required times for nodes v and w and $S(v)$ is the set of successors of node v.

Together, the actual and required event times, $e(v)$ and $r(v)$, 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. The required time $r(v)$ is the latest time that event v can occur without causing the project completion time to exceed T_r. The difference between the required and actual event times[1] at a node is defined as the *slack* of the node

$$s(v) = r(v) - e(v). \tag{3}$$

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

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

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 slack (or float) value in the network. A *critical path* is 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 $T_r - e(F)$ is positive, negative, or zero. Also note that when the slacks and floats of critical events and jobs are negative, the desired project completion time is infeasible unless job times are reduced until all floats are zero or positive.

To simplify later discussions, we also define a job (or arrow) $u \rightarrow v$ as *controlling* if $e(v) = e(u) + t_{u,v}$. The set of controlling jobs is a superset of the set of critical jobs; a critical path is thus a sequence of controlling jobs that connect the source and sink nodes.

When large project networks are analyzed, it is often possible to find multiple *parallel* critical paths. When parallel critical paths exist, each path is sufficient to prevent a 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.

Thus far we have described the activity-on-arrow (AoA) formulation for project networks. Although this is the most common formulation, another, called the activity-on-node (AoN) is also used sometimes. In the AoN formulation [7], 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. Analogous definitions are made for actual and required event times, slack, and float.

B. Critical Paths in Digital Circuits

The application of critical path analysis to digital circuits was first described by Kirkpatrick and Clark [9], and later

[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 II-C.

elaborated by Hitchcock, Smith, and Cheng [10]. Both groups used the AoN formulation to analyze the timing of combinational circuits. Each node in the project network represented a combinational block. Kirkpatrick's work was based on PERT, where delays were represented with probability distributions. Hitchcock used scalar delay values and CPM, although these could be modified to represent probability distributions. As in the general CPM, critical paths were determined by a forward pass to calculate signal event times at each gate output and a subsequent backward pass to directly compute slacks (required times were calculated implicitly).

The timing of combinational circuits can also be modeled using an AoA network. 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.

We can also use CPM techniques to find the *shortest* path through a circuit, as described by Hitchcock *et al.* [10]. In an AoA network, actual and required event times are calculated from

$$e(v) = \min_{u \in P(v)} [e(u) + t_{u,v}] \tag{5}$$

$$r(v) = \max_{u \in S(v)} [r(u) - t_{u,v}]. \tag{6}$$

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 (5) can be viewed as a replacement of the nodes in the original project network, which computed *max* functions, with new nodes that 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

$$s(v) = e(v) - r(v) \tag{7}$$

$$f(u \rightarrow v) = t_{u,v} + e(u) - r(v). \tag{8}$$

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.

These 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. The goal in this case is to determine, for the given circuit, the clock schedule with the smallest possible cycle time, $T_{c,\min}$. Thus, the required times for each of the feedback-free regions are set to 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.

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 (5)–(8) are used to identify critical short 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 corrected by adding delay to the short paths or using devices with smaller hold times.

C. Timing Model 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.

Our timing model for latch-controlled circuits is an extension of the one described by Sakallah, Mudge, and Olukotun [2]. For reference, the model equations and constraints are listed in Fig. 1. Variables in the clock model denote the clock cycle time, T_c, and the times R_p and F_p of the rise and fall events for each clock phase p. This allows the modeling of both positive and negative level-sensitive devices, as we only need to specify which clock events control 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, and as done in [2], we restrict ourselves to systems where all clock phases share a common period and do not consider gated clocks. A sample clock schedule is illustrated in Fig. 1(a). Delays in the clock distribution network are represented with a pair of parameters for each latch, q_i and Q_i, which correspond to the minimum and maximum delays between the clock generator and latch l_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 the 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

Clock Constraints	
$(R_p - F_p) \bmod T_c \geq w$	(1)
$(F_p - R_p) \bmod T_c \geq w$	(2)
Combinational Propagation Constraints	
$t_i = \min_{j \in P(i)}(d_j + \delta_j + \delta_{ji} - \varnothing_{ji}) = \min_{j \in P(i)}(d_j + \hat{\delta}_{ji})$	(3)
$T_i = \max_{j \in P(i)}(D_j + \Delta_j + \Delta_{ji} - \varnothing_{ji}) = \max_{j \in P(i)}(D_j + \hat{\Delta}_{ji})$	(4)
Latch Macromodel	
$A_i \leq T_c - S_i + q_i$	(5)
$a_i \geq H_i + Q_i$	(6)
$d_i = \max(a_i, E'_i + q_i)$	(7)
$D_i = \max(A_i, E'_i + Q_i)$	(8)
Phase Shift Operator	
$\varnothing_{ji} = T_c - (L_j - L_i) \bmod T_c$	(9)

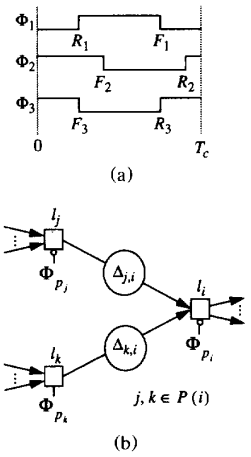


Fig. 1. Latch timing model summary; (a) sample clock schedule, (b) circuit section.

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.

Primed versions of the enable and latch event variables, E'_i and L'_i, denote these events in the *local frame-of-reference* of latch l_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 frame-of-reference to a variable t' in the frame-of-reference of latch l_i. For the enable event, this gives $E'_i = T_c - (L_i - E_i) \bmod T_c$ and for the latch event, $L'_i = T_c - (L_i - L_i) \bmod T_c = T_c$.

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 ϕ_{ji} is used to convert signal times from the frame-of-reference of latch l_j to that of latch l_i. The definition of ϕ_{ji} insures that $0 < \phi_{ji} \leq T_c$ 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. Other phase-shift operators could be used to model different definitions of correct operation. For example, the timing constraints for a two-cycle path could be obtained by adding T_c to the right-hand side of (17).

We can describe latches as *transparent* or *nontransparent* depending on 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.[2]

[2]Note that clock skew at a latch does not affect the flow of data through the latch when it is transparent.

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

D. Timing Constraint Graphs

The constraints in Fig. 1 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 Fig. 2. 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 times for the latch. A zero-weight arrow connects the arrival and departure time nodes and reflects the arrival time terms in (15) and (16). For each term in the arrival time (11) and (12), an arrow labeled $\hat{\Delta}_{j,i}$ or $\hat{\delta}_{j,i}$ connects the departure time vertex of latch l_j to the arrival time vertex of latch l_i. Note that except for splitting each latch into two vertices, the constraint graphs as defined thus far can be mapped directly to the circuit being modeled, as each $\hat{\Delta}_{j,i}$ or $\hat{\delta}_{j,i}$ connection corresponds to a path from latch l_j to latch l_i. The clock system and its associated constraints are incorporated into the constraint graphs by adding two special vertices and sets of associated arrows. Clock distribution is modeled 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 arrow 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 arrow models the occurrence of the enabling clock event at latch l_i. Arrival time constraints are modeled by connecting arrival time vertices to a sink vertex labeled F or f in the late and early signal graphs, respectively. In the late signal graph, these arrows 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

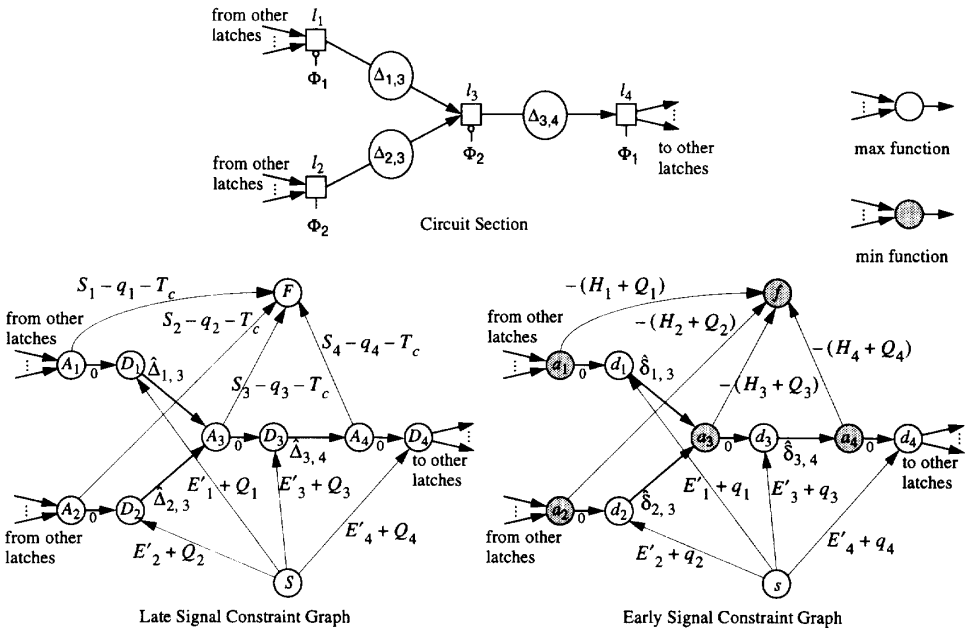


Fig. 2. Graph model of latch timing constraints.

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 described in the constraints of Fig. 1. 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 slack associated with the tightest (most constraining) setup constraint 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 most negative slack associated with circuit hold constraints. For both graphs, the effective project required time, T_r, is zero.

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

Although the nodes and arrows have different labelings, 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 through level-sensitive latches.

III. Critical Path Classification

When circuits are clocked with level-sensitive latches instead of edge-triggered devices, the concept of a critical path

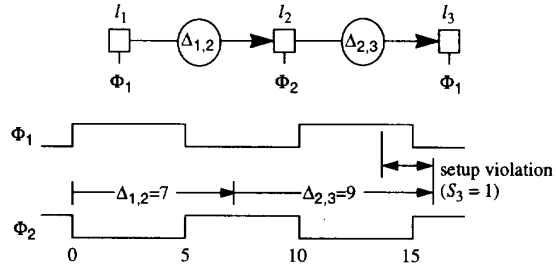


Fig. 3. Critical path through a level-sensitive latch.

becomes more complicated, 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 Fig. 3, 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 latch l_1 to latch l_3. The definition of a path must, therefore, be extended to include multiple combinational segments joined by transparent latches.

This extension complicates the analysis in two ways. The first complication appears when we analyze short paths in latched circuits. For short paths, the constraint graph 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 late and early signal constraint graphs may contain cycles. If a circuit contains feedback, then the project networks will contain cycles, and some of these cycles can constrain circuit

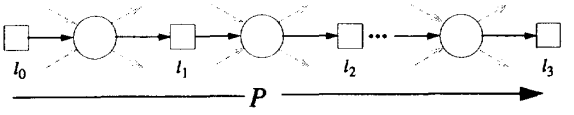


Fig. 4. Sample path.

timing. Classical CPM approaches are unable to deal with cyclic project networks.

In this section we formally define three types of critical paths: critical long paths, critical short paths, and critical loops. All of these types can be observed in the late and early signal constraint graphs and are presented in increasing order of complexity. We also discuss these critical paths directly in terms of latches and combinational logic blocks. In this context, a path P is a sequence of latches $P = l_0 \rightarrow l_1 \rightarrow \cdots \rightarrow l_m$ where each latch is directly connected to its predecessor through a combinational logic segment (Fig. 4). In the following sections we assume, without loss of generality, that the latches in the path are numbered from 0 to m. The length of P is defined as the number of combinational segments in the path so that $|P| = m$. 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 defined to be the number of clock cycles available for signals to propagate the length of the path. For path P, latency is related to phase shifts, cycle time, and enabling events by the following equation:

$$\Lambda(P) = \frac{1}{T_c}\left(\sum_{i=0}^{m-1} \phi_{i,i+1} + T_c - E'_0\right). \tag{18}$$

For multiphase circuits or circuits using level-sensitive latches $\Lambda(P)$ may be fractional. Note that since we are considering the verification problem only, $\Lambda(P)$ is constant throughout the analysis. $\Lambda(P)$ is also constant when all clock event times scale uniformly with the cycle time, T_c.

A. 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 Fig. 5(a). Actual and required times are defined as in Section II-A, although their calculation is complicated by the possible presence of cycles in the late signal constraint graph. Procedures for calculating these times are discussed in Section IV, allowing us to present the formal definition of a critical long path 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 arrows which connect the source and sink nodes.

Critical arrows are defined as in Section II-A as the arrows in the project network having the smallest or most negative slack. Relating this definition to the late signal graph shown in Fig. 5(a), we see that the following set of constraints must hold for any critical long path

$$D_0 = E'_0 + Q_0 \tag{19}$$

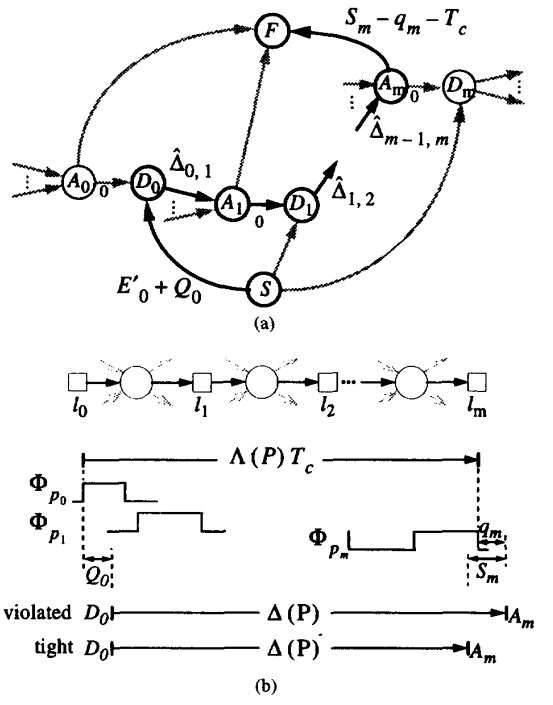


Fig. 5. Sample critical long path; (a) graph model, (b) path timing.

$$\forall\, i \in \{1,\ldots,m\},\ A_i = D_{i-1} + \Delta_{i-1} + \Delta_{i-1,i} - \phi_{L_{i-1}L_i} = D_{i-1} + \hat{\Delta}_{i-1,i} \tag{20}$$

$$\forall\, i \in \{1,\ldots,m-1\},\ D_i = A_i \tag{21}$$

$$\forall\, i \in \{1,\ldots,|L|\},\ A_m - (T_c + q_m - S_m) \geq A_i - (T_c + q_i - S_i) \tag{22}$$

$|L|$ is the number of latches in the circuit. 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 III-C, and in Section V it is shown that we can safely decouple the analysis of loops in cyclic constraint graphs.

Similar definitions of critical long paths were presented previously by Ishii, Leiserson, and Papaefthymiou [1], [16] and Lockyear and Ebeling [15]; both groups of researchers were primarily interested in retiming optimization problems; their definitions of a critical long path included all paths which could constrain the level-sensitive retiming problem. Both definitions implied a set of $|V|^2$ critical paths, where $|V|$ is the number of combinational blocks (gates) in a circuit. In contrast, we focus on the path (or paths) which correspond to the largest setup time violation in the circuit, as these paths reflect the largest delay reduction necessary to allow the circuit to operate at the desired speed. Subcritical paths can then be identified using slack and float information from the 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: (L1) any increase in a delay along the path will worsen (reduce the slack of) the most severe setup time constraint in the circuit, and (L2) a decrease to any delay in the path will ease this constraint, as long as no parallel critical paths are present.

Proof: (only if part) It is simple to show that if P is a critical long path, conditions (L1) and (L2) 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 (19)–(22) guarantee that P determines the value of $e(F)$, which can be calculated by simply summing the arrow 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 (L1) and (L2) 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 tightest setup constraint 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 arrow weights along P, a condition which can only be true if (19)–(22) are satisfied for P. These equations imply that the slacks on each arrow of P equal the smallest slack in the network; therefore, P must be a critical long path. □

The timing of a typical critical long path is illustrated in Fig. 5(b). Examining this, we make the following remarks

- Combining conditions (19)–(22), we see that a critical long path constraint with zero slack is of the form

$$\sum_{i=1}^{m}(\Delta_{i-1}+\Delta_{i-1,i})+S_m+Q_0-q_m = \sum_{i=1}^{m}(\phi_{i-1,i})+(T_c-E'_0)=\Lambda(P)T_c. \quad (23)$$

  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 setup constraint were violated, the leftmost "=" would be replaced by ">".
- A subpath of a critical long path may also be critical if an equally severe setup constraint exists on an internal latch in the path.

B. 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 and corresponds to critical paths in early signal graphs as shown in Fig. 6(a). The formal definition of a critical short path is as follows.

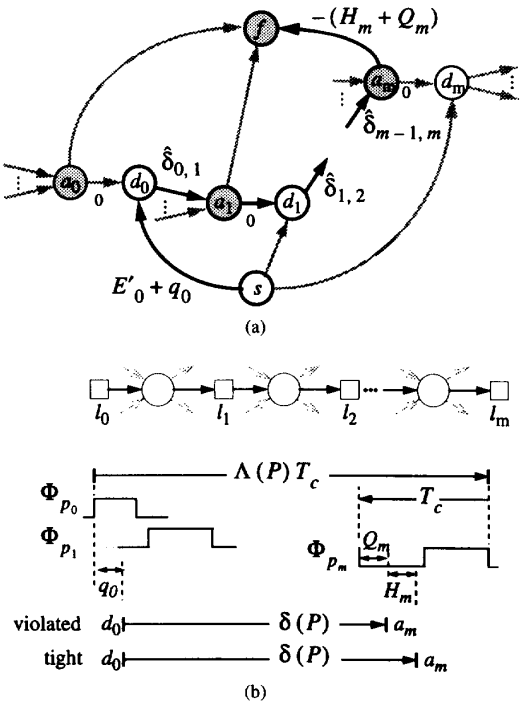


Fig. 6. Sample critical short path; (a) graph model, (b) path timing.

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

Critical arrows are those arrows having the smallest or most negative slack, where slack is now defined for the early signals, as in (7) and (8). However, since the early signal constraint graphs contain both min and max nodes, we must extend the definitions of early actual and required event times given in Section II-A. We can easily extend the event time definition using (1) to specify event times of max nodes and (5) for event times of min nodes [11]. The required time definition is more complex; we first illustrate it for the general mixed min/max graphs shown in Fig. 7; it can then be easily applied to the early signal constraint graphs.

In Fig. 7 actual and required event times are shown next to each node. Fig. 7(a) shows a graph containing predominantly max nodes where the sink node is also a max node. Actual event times are easily determined, but the required time calculation is complicated by the presence of the min node B. Since the sink node is a max node, required times in Fig. 7(a) are the latest times that events can occur without increasing the project completion time. The required time for node F is 6, and using (2), the required time for B is $6-5=1$. However, when we consider event A, we see that the time of event A could be made arbitrarily late without increasing the time of event B or the completion time at the sink node. The required time for event A is effectively infinite. A similar analysis for Fig. 7(b) determines that the required time for event a is $-\infty$.

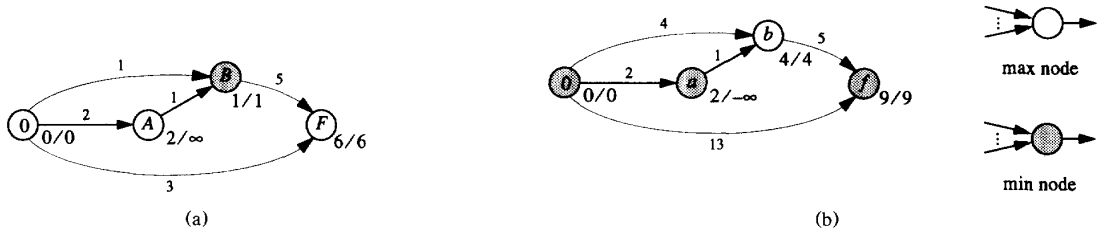


Fig. 7. Mixing min and max nodes in CPM graphs; (a) adding min nodes to max graphs, (b) adding max nodes to min graphs.

Thus we define the following general rules for determining required times in mixed min/max CPM graphs.

1) The type of the graph 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*.
2) Max nodes in min graphs and min nodes in max graphs are termed *minority* nodes.
3) The required times for all nodes are determined using (2) and (6), but with the following modification: if $e(u) + t_{u,v} \neq e(v)$, then a *minority* required time, $r'(v)$ is used to calculate $r(u)$.
4) In max graphs, the minority required times are $r'(v) = \infty$. In min graphs, the minority required times are $r'(v) = -\infty$.

These rules allow required event times to 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. In the degenerate case where more than one node determines the value of a minority node, all such nodes are assigned noninfinite slacks and can be potentially part of a critical path.

Examining the graph of Fig. 6(a), we see that the following constraints must hold for any critical short path:

$$d_0 = E'_0 + q_0 \tag{24}$$

$$\forall\, i \in \{1, \ldots, m\},\ a_i = d_{i-1} + \delta_{i-1} + \delta_{i-1,i} - \phi_{L_{i-1}L_i} = d_{i-1} + \hat{\delta}_{i-1,i} \tag{25}$$

$$\forall\, i \in \{1, \ldots, m-1\},\ d_i = a_i \tag{26}$$

$$\forall\, i \in \{1, \ldots, |L|\},\ (H_m - Q_m) - a_m \geq (H_i - Q_i) - a_i. \tag{27}$$

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: (S1) any decrease in a delay along the path will worsen (reduce the slack of) the most severe hold time constraint if no parallel critical paths are present, and ($S2$) an increase in any path delay will ease this constraint, 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 (S1) and (S2) 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 (24)–(27) guarantee that P determines the value of $e(f)$, which can be calculated by simply summing the arrow weights along P. Increases to any delay in P will allow $e(f)$ to increase, and reductions to any delay in P will 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 (S1) and (S2) 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 arrow weights along P, a condition which can only be true if (24)–(27) are satisfied for P. These equations imply that each arrow of P be a controlling arrow; therefore, P must be a critical short path. □

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 timing of a typical critical short path is illustrated in Fig. 6(b). Examining this, we add the following remarks

- Combining conditions (24)–(27), we see that a zero-slack critical short path constraint is of the form

$$\begin{aligned}\sum_{i=1}^{m}(\delta_{i-1} + \delta_{i-1,i}) + q_0 - Q_m &= \sum_{i=1}^{m}(\phi_{i-1,i}) + (T_c - E'_0) - T_c + H_m \\ &= (\Lambda(P) - 1)T_c + H_m. \end{aligned} \tag{28}$$

  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 over which these propagations must take place. If the path were violated, the leftmost "=" would be replaced by "<".
- 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

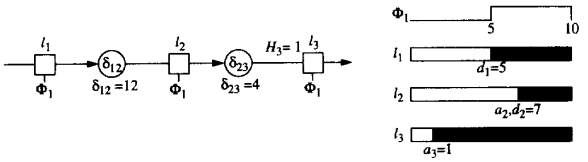


Fig. 8. Circuit that exhibits transient but no steady-state errors.

of length 2 is shown in Fig. 8. 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. This property was first pointed out by Szymanski and Shenoy [4] and implies that for a circuit to be restartable with no hold time violations, critical short paths should be no longer than one combinational 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 $d_i = E_i' + q_i$, as suggested by Szymanski [5] and Sakallah *et al.* [2]. This has the significant additional benefit of simplifying the early signal constraint graph by eliminating all max nodes from the graph.

C. 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 Fig. 9(a). The formal definition of a critical loop is as follows.

Definition 3: A critical loop is a circular path which corresponds to a maximum-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 II-A. Classical CPM theory makes no provision for cycles in project networks; cycles are usually considered errors in planning or analysis. However, the cycles in these constraint graphs are not due to design errors, but are a natural consequence of the use of feedback in sequential circuits. The presence of critical loops was also noted in the retiming work of Ishii *et al.* [16] and Lockyear and Ebeling [15]. Both groups observed that loops of transparent latches place a fundamental limit on the cycle time attainable by retiming algorithms, which relocate latches to eliminate setup time violations. Similarly, Szymanski used critical loop information to identify the minimum cycle time attainable using optimal clocking methods [5], [2], [6], which adjust clock event times to eliminate setup and hold time violations.

When a positive-weight cycle exists in a constraint graph, some or all of the event times are undefined. Examining the

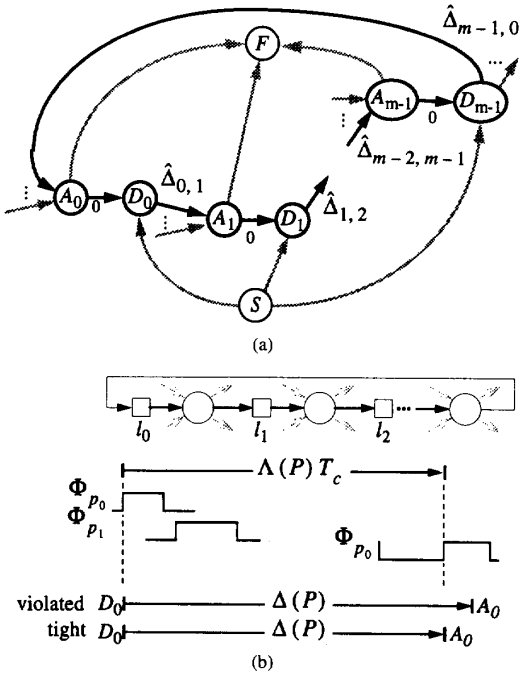


Fig. 9. Sample critical loop; (a) graph model, (b) path timing.

constraint graph in Fig. 9(a), such a cycle implies the following relationship

$$\sum_{i=1}^{m}(\Delta_{i-1} + \Delta_{i-1,i}) > \sum_{i=1}^{m}(\phi_{L_{i-1},L_i}) \tag{29}$$

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 a loop exactly equals the available propagation time, then the corresponding cycle has zero weight, and all of its arrival and departure times are well-defined solutions of the equations of Fig. 1. Referring to the constraint graph of Fig. 9(a), we see that the timing associated with a zero-weight critical loop is as follows

$$\forall i \in \{1,\ldots,m\},\ A_i = D_{i-1} + \Delta_{i-1} + \Delta_{i-1,i} - \phi_{L_{i-1}L_i} \tag{30}$$

$$\forall i \in \{1,\ldots,m\},\ D_i = A_i. \tag{31}$$

There is no definition for critical loops in terms of required times, slacks, and floats, as these require well-defined event times and the ability to compare these times against a local constraint (i.e., the setup or hold constraint). In contrast, loop constraints are not associated with any single latch, but with groups of latches arranged in topological loops.

Positive weight cycles can also exist in the early signal constraint graph. However, the following theorem, equivalent to one by Szymanski and Shenoy [4], shows that we need not check for critical loops in the early signal graph.

Theorem 3: If a critical loop 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 (30)–(31), but with a_i, d_i, δ_i, and δ_{ij} substituted for their late signal counterparts. A satisfied loop constraint has the form

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

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

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

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

The timing of a typical critical loop is illustrated in Fig. 9(b). To this we add the following remarks

- Combining conditions (30)–(31), we see that a zero-weight critical loop constraint is of the form

$$\sum_{i=1}^{m}(\Delta_{i-1} + \Delta_{i-1,i}) = \sum_{i=1}^{m}(\phi_{L_{i-1},L_i}) = \Lambda(P)T_c. \quad (32)$$

  Terms on the left side of the equation represent logic delays and those on the right represent the time available for signal propagation. If the constraint were violated, the "=" would be replaced by ">".
- 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

$$T_{c,\text{loop}} = \max_{P \in \text{LOOPS}} \left(\frac{\sum_{i=1}^{m}(\Delta_{i-1} + \Delta_{i-1,i})}{\Lambda(P)} \right) \quad (33)$$

  where LOOPS is the set of topological loops in the circuit. Note that (33) assumes that the latency of each loop is a constant. In fact, $\Lambda(P)$ will be a constant integer for each loop, as the enabling and latching event times do not appear in the loop constraint (32).

It is not necessary to enumerate the possibly-exponential number of topological loops to determine $T_{c,\text{loop}}$. As a number of researchers have observed [1], [5], [15], $T_{c,\text{loop}}$ can be calculated in polynomial time using a maximum ratio cycle algorithm [14] to find the solution to (33). We currently use the variant of Lawler's maximum mean-weight cycle algorithm [12] which Szymanski used to compute $T_{c,\text{loop}}$ for optimal clocking problems [5]. Lawler's 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 $|E|$ is 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 the Bellman-Ford algorithm 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 the acceleration heuristic developed by Szymanski [5], the presence of a positive-weight cycle can be determined much more quickly. The heuristic involves periodically checking a subgraph for cycles; if a cycle is found, its weight is computed and checked. The subgraph consists of exactly one controlling arrow per node; as a result, it can be checked in $O(|V|)$ time. These extra cycle checks raise the worst-case asymptotic complexity of the algorithm to $O(b|V|^2|E|)$; in practice the enhanced algorithm runs much faster than the unmodified Lawler's algorithm.

```
// initialize the latch arrival times
for i = 1 to |L|
    APrev[i] = aPrev[i] = -∞;

// iterate the evaluation of the departure and arrival time equations
// until convergence or a maximum of |L| iterations
iter = 0;
repeat
    iter = iter + 1;

    // update the latch departure times based on the latch arrival times
    // computed in the previous iteration
    for i = 1 to |L| {
        D[i] = max (APrev[i], E'[i] + Q[i]);
        d[i] = max (aPrev[i], E'[i] + q[i]);
    };

    // update the latch arrival times based on the just-computed
    // latch departure times
    for i = 1 to |L| {
        A[i] = max_j (D[j] + Δ̂[j][i]);

        a[i] = min_j (d[j] + δ̂[j][i]);
    };
until ((A[i] = APrev[i]) && (a[i] = aPrev[i])) || (iter + 1 > |L|);

// check and record setup and hold violations
for i = 1 to |L| {
    SetupVio[i] = A[i] > Tc - S[i] + q[i];
    HoldVio[i] = a[i] < H[i] + Q[i];
};
```

Fig. 10. Algorithm SIMPLE-RELAX: Relaxation verification algorithm.

IV. Critical Path Identification by Slack Calculation

This section describes a procedure for identifying critical paths based on the slack and float definitions presented in Sections II and III. Actual event times are calculated using a common timing verification method for circuits with level-sensitive latches. Required times are calculated and critical paths are identified in a post-processing step. The verification method we consider is that of simply *relaxing*, or iteratively resolving, the timing constraints until a fixedpoint solution is found.

A. Timing Verification by Constraint Relaxation

It was shown by Szymanski and Shenoy that the arrival and departure time equations of Fig. 1 form a system of equations which can be solved by relaxation until a fixed point is obtained [4]. A commonly-used procedure (Algorithm SIMPLE-RELAX) is shown in Fig. 10. If the arrival times are initialized to $a_i = A_i = -\infty$, then this relaxation algorithm simulates the start-up timing of the circuit, with each iteration simulating the events of one clock cycle. They also observed that if multiple solutions exist for the arrival and departure equations, these initial values ensure that the most physically meaningful solution is found, as they result in the solution with the lowest possible arrival time values. Once the arrival and departure times for all latches have been determined, they are checked to determine whether the setup and hold time constraints are satisfied.

If loop constraint violations exist, the arrival and departure times around the loop will increase without bound. The limit on the number of iterations of SIMPLE-RELAX is to prevent this "runaway" condition. Since they cause arrival times to increase arbitrarily, loop constraint violations will be detected as setup errors.

Relaxation algorithms have been observed to converge to solutions very rapidly [2], and the first performance bounds were reported by Ishii *et al.* [1] and later by Szymanski and Shenoy [4]; both groups observed that relaxation of the equations corresponds to a Bellman-Ford solution of a simple set of linear constraints. 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, or latch-to-latch connections. If a solution exists, it will be found in at most $|L|$ iterations, so that $|L|$ should be used as the iteration limit in Fig. 10. Each iteration involves examining 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.

Since the relaxation verification algorithm is essentially a variant of the Bellman-Ford algorithm, the heuristic described in Section III-C can again be used to periodically check for violated loops. A version of algorithm SIMPLE-RELAX enhanced with this heuristic is discussed in Section VI as algorithm ACCELERATED-RELAX. 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, making it impossible to identify critical paths using slack and float values. The next section describes one possible approach for defining an easy to calculate and well-defined set of arrival and departure times.

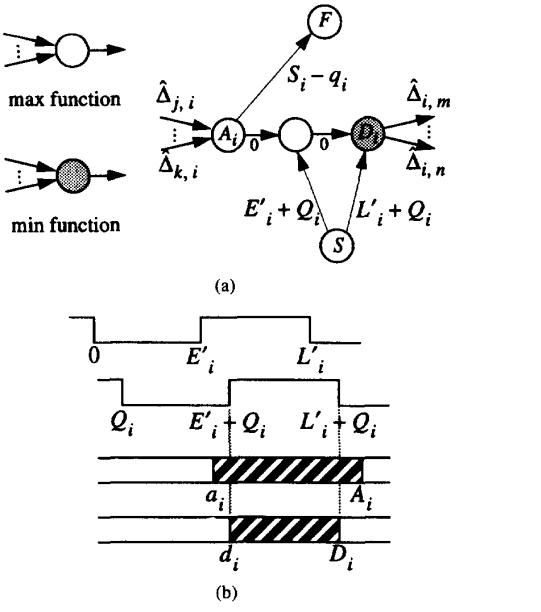


Fig. 11. Modified latch model; (a) graph model, (b) timing diagram.

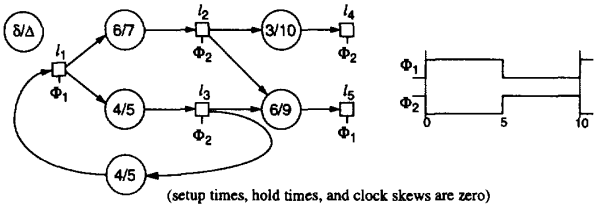


Fig. 12. Example circuit for critical path identification.

B. Clipping Departure Times During Constraint Relaxation

With a simple modification to the latch model of Fig. 1, we can guarantee that all arrival and departure times will have well-defined values, even in the presence of violated loops. The original model allowed signals to depart at any time after the latch's enabling event, even after the latch was closed. A more accurate model instead uses the following departure time equation: $D_i = \min(\max(A_i, E'_i + Q_i), L'_i + Q_i)$. With this equation, signals are only expected to depart from the latch in the interval between the latch's enabling and latching events. For each latch, this adds a min node to the late signal constraint graphs of Fig. 2. The modified graph structure is shown in Fig. 11(a). Late departure times are thus *clipped* [2] to occur no later than the device's latching event.

The introduction of the min function does not eliminate the possibility of positive-weight cycles, but it does prevent them from causing variables to increase without bound. When a positive-weight cycle exists, the min nodes effectively break the cycle when it no longer produces the smallest input to the min node. In most cases, this will cause the relaxation to stop after fewer iterations, but if the amount of violation, V, is small, a full $|L|$ iterations may still be required to discover that a loop constraint is violated. This is a result of the fact that we cannot clip a departure time until it exceeds $L'_i + Q_i$ and that in the worst case, the departure time at each node in a critical loop will increase by the amount of violation every $|P_{\text{loop}}|$ iterations, where $|P_{\text{loop}}|$ is the size of the loop [13]. This effect is illustrated in Section VI using algorithm CLIP-RELAX, a version of algorithm SIMPLE-RELAX which has been augmented to clip nonphysical departure times.

C. Identifying Critical Paths in the Absence of Violated Loops

This section and the next (Section IV-D) present methods for extracting critical path information from verification results. First we consider the case where no loops in the circuit are violated. In Section IV-D, 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 model of Fig. 1. With an appropriate definition of the required time for each event, we can calculate slack and float values and identify critical paths. The circuit shown in Fig. 12 contains a single topological loop, whose timing is satisfied for the clock schedule shown. Fig. 13 shows the late signal constraint 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 Fig. 10 (algorithm SIMPLE-RELAX). The setup times for all latches are zero.

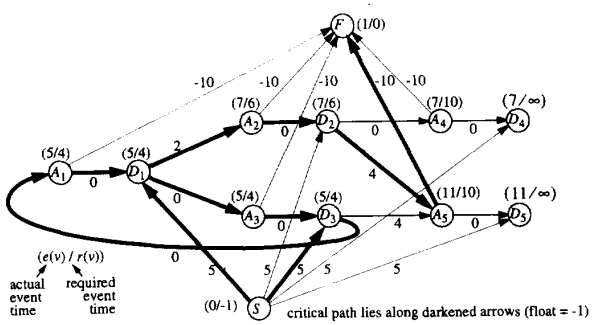


Fig. 13. Late signal graph with no violated loops.

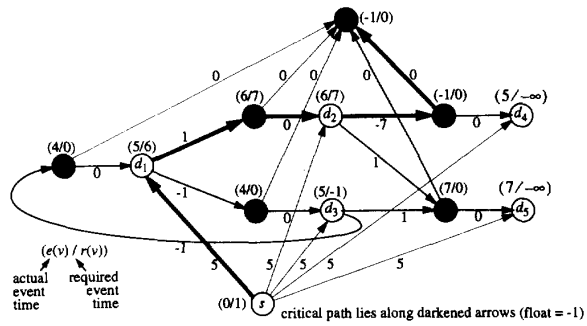


Fig. 14. Early signal graph.

Required times are obtained by recalling (2), but since the circuit contains cycles, the required times must also be calculated by relaxation. Using a procedure similar to SIMPLE-RELAX, required time values are computed at each node and are propagated back through the graph until all have stabilized. Since this is again an application of the Bellman-Ford algorithm, we know that if there are no violated loops, these required times will stabilize in at most $|N|$ iterations, where $|N|$ is the number of nodes in the graph. The only thing which can prevent a solution is a positive-weight cycle in the constraint graph, and such a positive-weight cycle will only exist when a violated loop is present.

Examining the late signal graph in Fig. 13, we see that the sink has an actual event time of 1 and a required time of 0. The resulting slack at the sink is -1, and tracing back along the arrows with -1 float, we see the critical arrows marked in bold.

Examining the set of critical arrows, we see that there are actually two overlapping critical long paths, $(l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_5)$ and $(l_1 \rightarrow l_2 \rightarrow l_5)$, and a critical loop $(l_1 \rightarrow l_3 \rightarrow l_1)$. The loop overlaps the long paths at nodes A_1, D_1, A_3, and D_3, and the nodes on the critical loop have the same slack as those on the critical long paths.

Only those arrows connecting nodes D_i and A_j correspond to actual circuit delays; all other arrows are fixed by the clock schedule and timing model. Keeping this in mind, increasing the time on any critical arrow will worsen the setup violation at latch l_5; reducing the time on arrows between nodes D_1 and F will likewise reduce the amount of violation. Reductions to other delays will be masked by a parallel critical path; e.g., reducing the delay on the subpath $(l_3 \rightarrow l_1)$ will not eliminate the violation due to the parallel path $(l_1 \rightarrow l_2 \rightarrow l_5)$.

Critical long paths can be enumerated 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 of a path P, then P contains a cycle and the search for long paths along P can be ended.

These same methods can also be used to identify critical short paths. The early signal constraint graph for Fig. 12 is shown in Fig. 14. Critical arrows are drawn in bold, and the critical short path is $(l_1 \rightarrow l_2 \rightarrow l_4)$.

Note that neither of these slack calculations directly detected the presence of the critical loop containing l_1 and l_3; it was instead discovered as a side effect of the long path identification. This loop prevents us from increasing the total delay $\Delta_{1,3} + \Delta_{3,1}$ or lowering the cycle time below 10. We observe that the critical loop in Fig. 13 consists of nodes and arrows with slack and float equal to -1, the most negative slack in the circuit.

It is easiest to identify critical loops when the corresponding cycle in the constraint graph has zero weight. When this is the case, all slacks and floats around the loop will have the same value, as shown by the following theorem.

Theorem 4: If a critical loop in a circuit has zero weight, then all of the slacks and floats around the loop will have equal values.

Proof: From Definition 3, if a critical loop has zero weight, then the maximum-weight cycle in the constraint graph will have zero weight, and thus well-defined values will exist for all the actual and required event times in the graph. Let P_{loop} be the path corresponding to the critical loop. At each node in the loop, $e(v) = \max_{u \in P(v)}[e(u) + t_{u,v}]$ and $r(v) = \min_{w \in S(v)}[r(w) - t_{v,w}]$. Since P_{loop} is a zero-weight cycle, any increase de in a value of $e(v)$ will cause all the actual event times in the loop to increase by de. Similarly, a reduction dr to $r(v)$ will cause all required event times in the loop to be reduced by dr. This implies that $e(v) = e(u) + t_{u,v}, u \in P(v), u \in P_{\text{loop}}$ and $r(v) = r(w) - t_{v,w}, w \in S(v), w \in P_{\text{loop}}$. Substituting variables and eliminating $t_{u,v}$ gives $r(u) - e(u) = r(v) - e(v)$. Comparing this with (3), we conclude that the slack of each node v in the loop is equal to that of its predecessor u and that all slacks in the loop are equal. Float is defined by (4) as $f(u \rightarrow v) = r(v) - e(u) - t_{u,v}$. Substituting for $e(u)$ gives $f(u \rightarrow v) = r(v) - e(v)$, and thus all floats in the loop also share the same value. □

Theorem 4 does not imply that zero-weight critical loops will have the most negative slack in a circuit, only that the slacks of the nodes in each zero-weight loop will have equal values. To find zero-weight loops we can thus perform a depth-first search over all sets of such arrows and nodes with the same float and slack to find critical loops, but a more practical approach may be to simply search the subgraph consisting only of controlling arrows. Either case will provide a linear-time procedure for identifying one of the critical loops of a

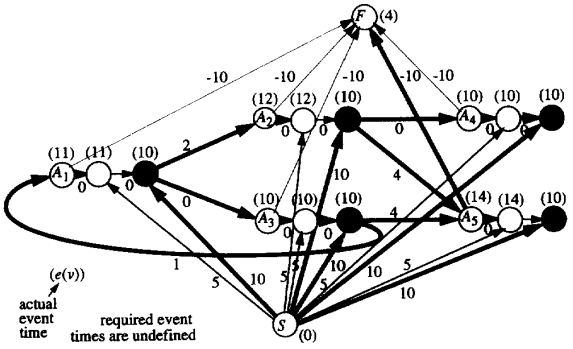


Fig. 15. Clipping departure times during relaxation.

circuit (if one exists), although the total number of critical loops may be exponential.

D. Identifying Critical Paths in the Presence of Violated Loops

When the critical loop(s) in a circuit are violated, the above procedures for identifying and extracting paths break down. Since a violated loop corresponds to a positive weight cycle in the constraint graph, the arrival and departure time equations will have no solution. One way to force a solution is to use the model of Fig. 11, which clips departure times to obtain stable values.

However, this approach also fails to provide adequate critical path information, as illustrated below. Fig. 15 shows the late signal constraint graph for the circuit of Fig. 12 where the delay from l_3 to $l_1(\Delta_{3,1})$ has been increased to 6. This causes the critical loop containing l_3 and l_1 to be violated for the original cycle time of 10. Using the modified latch model of Section IV-B, the event times stabilize to the values shown. However, the positive-weight cycle is again a problem when we attempt to calculate required times, as it prevents the required time relaxation from stabilizing. If we attempt to trace critical paths along the set of controlling arrows (drawn in boldface in Fig. 15), a number of paths appear violated, but as we know from Fig. 13, most of these apparent critical paths are unrelated to the critical loop $l_1 \rightarrow l_3 \rightarrow l_1$ and the critical long path $l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_5$.

E. Strategies for Critical Path Extraction

The example of the previous section illustrates the difficulty of identifying critical paths when loops are violated. No solution exists to the original timing model; and although a solution can be found for the clipped model, we are unable to calculate required times or to unambiguously trace critical paths.

Fortunately, in most situations it is sufficient to identify only those critical paths at $T_{c,\min}$, the minimum cycle time of the circuit or at $T_{c,\text{loop}}$, the critical loop frequency. In such an approach, $T_{c,\text{loop}}$ can be calculated using Lawler's algorithm, and $T_{c,\min}$ can be obtained using an optimal clocking algorithm [2], [5]. Choosing one of these as the target cycle time, we can rerun the relaxation and obtain the critical long path (or paths) and any zero-weight loops. Using slack information, we can also identify near-critical long and short paths, but no such information is available for loops without enumerating and checking all the topological loops in the circuit. The difference is that slack and float are only defined with respect to the sink node, which does not appear in critical loops. A maximum ratio cycle algorithm can find the critical loop, but will not provide any information about the second or third (etc.) most constraining loops in the circuit.

However, timing optimization is typically done iteratively, in which analysis and optimization steps are interleaved. For any such approach, the above procedure will be adequate as long as a circuit does not contain a large number of violated loops with differing weights. Such a circuit could require an impractical number of iterations, stalling the optimization process.

An alternative approach would be to treat latches in a circuit as edge-triggered devices, much has been done in classical applications of CPM to latched circuits. However, in this case we only select a sufficient subset of latches to make the constraint graph acyclic. For each such latch, we pick a required arrival time for the latch and compute the latch departure time based on that value. This decouples the actual latch arrival time A_i from the latch departure time D_i, eliminating the $A_i \rightarrow D_i$ arrow from the constraint graph. For each modified latch, the $S \rightarrow D_i$ and $A_i \rightarrow F$ arrows in the constraint graph are modified to enforce the new constraints. This is a conservative approach, as it does not allow the departure times of the modified latches to change during the optimization. It has the advantage that slacks and floats also reflect the loop constraints. However, the selection of latches to "fix" is arbitrary, as is the selection of the new required arrival time for each modified latch.

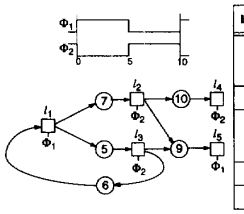
V. Critical Path Verification by Construction

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

This section describes an alternate approach based on the enumeration of possible critical paths, called *candidate* paths, using heuristic information to reduce the number of paths enumerated. The approach is similar to the relaxation methods of Section IV and can provide insight into their operation. It is significantly more complicated, as it carries more information forward through the timing verification. However, it is also a more robust procedure for obtaining critical path information in the presence of violated critical loops. No slacks or floats are computed, so that we have less information about sub-critical paths, but when the algorithm terminates, it has identified the most constraining acyclic path(s) to each latch; i.e., the path(s)

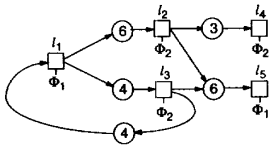
```
// enumeration phase
create an initial zero-length candidate path at the output of each latch
set i = 1
while there are paths of length i-1
    for each latch l
        identify the latest arriving signal(s) on the input of l
        if (the latest arriving signal is due to a path of length i-1
            and the signal is not blocked by the clock) then
            if (the path contains latch l) then
                report the possible critical loop
            else
                extend the path to include latch l
            end if
        end if
    end for
    place each extended path on the output of the corresponding latch l
    set i = i+1
end while
// verification of extended paths
for each latch in the circuit
    check timing constraints on paths at latch inputs
end for
```

Fig. 16. Algorithm EXTEND-LATE: Long path/loop construction.



latch	iteration 1	iteration 2	iteration 3
l_1	$l_3 \rightarrow l_1/6$	no change, but identify loop $l_1 \rightarrow l_2 \rightarrow l_1$	no change
l_2	$l_1 \rightarrow l_2/7$	$l_3 \rightarrow l_1 \rightarrow l_2/8$	no change
l_3	$l_1 \rightarrow l_3/5$	no change, but identify loop $l_3 \rightarrow l_1 \rightarrow l_3$	no change
l_4	$l_2 \rightarrow l_4/5$	$l_1 \rightarrow l_2 \rightarrow l_4/7$	$l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4/8$
l_5	$(l_2,l_3) \rightarrow l_5/9$	$l_1 \rightarrow l_2 \rightarrow l_5/11$	$l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_5/12$

Fig. 17. EXTEND-LATE example.



latch	iteration 1	iteration 2
l_1	$l_3 \rightarrow l_1/4$	no change
l_2	$l_1 \rightarrow l_2/6$	no change
l_3	$l_1 \rightarrow l_3/4$	no change
l_4	$l_2 \rightarrow l_4/-2$	$l_1 \rightarrow l_2 \rightarrow l_4/-1$
l_5	$(l_2,l_3) \rightarrow l_5/6$	$l_1 \rightarrow l_2 \rightarrow l_5/7$

Fig. 18. EXTEND-EARLY example.

which produces the latest (or earliest) signal arrival at the latch input. Zero- and positive-weight loops are identified as a side effect of the algorithm.

A. Preliminaries

A path which partially satisfies the criticality conditions of Sections III-A or III-B is called a *candidate* path. Although candidate paths may not be critical, every critical path must also be a candidate path and any path which is *not* a candidate path *cannot* be critical. If conditions (19)–(21) are satisfied for a path P, then we call P a *candidate long path.* If condition (22) is also satisfied, P is a critical long path. If conditions (24)–(26) are satisfied for a path P, then P is called a *candidate short path.* If condition (27) is also satisfied, P is a critical short path.

B. Path Extension Algorithm

Fig. 16 sketches an algorithm that constructs the candidate long paths in a circuit. The algorithm for candidate short paths is similar. Each iteration in the algorithm generates successively longer candidate paths, with the ith iteration generating all the candidates of length i. The paths generated in iteration i are constructed by extending the paths constructed in iteration $i-1$. The extensions are performed by identifying the latest arriving input to each latch. If the corresponding path is of length $i-1$, then it is extended as long as (1) the resulting extended path does not contain a cycle, and (2) the latch at the end of the path is transparent. Since they produce the latest arrival times and flow through transparent latches, the set of paths extended in algorithm EXTEND-LATE is identical to the set of candidate long paths. If parallel candidate paths produce the same arrival time at a latch, they are grouped together and are extended as a single path in successive iterations.

All candidate paths of length i can be generated from those of length $i-1$. This is guaranteed by the following theorem, similar to Lemma 5.2 in [5].

Theorem 5: If a path $P = l_0 \rightarrow l_1 \rightarrow \cdots \rightarrow l_m$ is a candidate path of length m that ends at latch l_m, then $P^* = l_0 \rightarrow l_1 \rightarrow \cdots \rightarrow l_{m-1}$ 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. □

A simple illustration of the algorithm EXTEND-LATE is shown in Fig. 17. The circuit being analyzed is the circuit of Fig. 12, but with the loop violation that complicated slack-based path identification. Each column in the table shows paths produced in the corresponding iteration. Late arrival times associated with each path are also shown, and are in the local frame-of-reference of each latch. The algorithm correctly identifies the critical long path $l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_5$. In iteration 2, the violated loop involving l_1 and l_3 is detected twice, on the extension of paths to each latch.

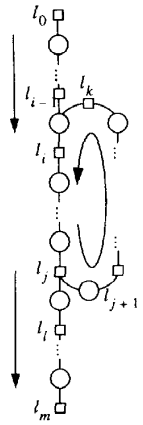
Fig. 18 illustrates the short path extension algorithm (Algorithm EXTEND-EARLY) for the same circuit. Early arrival times associated with each path are shown. In the first iteration, a hold violation on l_4 appears due to the shortness of the path $l_2 \rightarrow l_4$; however, the violation is reduced in the next iteration due to the extended path $l_1 \rightarrow l_2 \rightarrow l_4$.

C. Performance Analysis

The maximum number of iterations required by the algorithms EXTEND-LATE and EXTEND-EARLY is bounded by Theorem 6, which shows that for timing verification purposes, it is only necessary to consider *simple* paths. A long path or short path is simple if it contains no cycles, and a loop is simple if it contains no cycles other than the loop itself. If the simple paths are error-free, then the timing constraints of all other (composite) paths must also be satisfied.

Theorem 6: 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 Fig. 19, a critical long path with an internal cycle. For this to be a critical long path, the constraints shown in the associated table must be satisfied. However, the constraints shown will



$D_0 = E'_0$	(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)

Fig. 19. Critical long path with internal cycle.

also be satisfied as long as

$$P_1 = l_0 \rightarrow \cdots \rightarrow l_i \rightarrow \cdots \rightarrow l_j \rightarrow l_l \rightarrow \cdots \rightarrow l_m$$

is not a violated critical long path and

$$P_2 = l_i \rightarrow \cdots \rightarrow l_j \rightarrow l_{j+1} \rightarrow \cdots \rightarrow l_k \rightarrow l_i$$

is not a violated critical loop. Constraints (0)–(j), and (l)–(m) are required in order for P1 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. □

Theorem 6 implies that the longest simple candidate path will have at most $|L|-1$ segments, where $|L|$ is the number of latches in the circuit. Since a simple candidate path can contain no internal cycles, each latch in a circuit can occur at most once in such a path, making the longest possible simple candidate path be one that extends through every latch in the circuit. The length of such a path is $|L|-1$.

Algorithms EXTEND-LATE and EXTEND-EARLY thus require at most $|L|$ iterations to generate and verify all candidate paths up to length $|L|-1$. (The additional iteration is to recognize that no more paths can be extended.) Each iteration will examine as many as the $|E|$ edges in the circuit to identify new candidate paths. The worst case performance of this algorithm is thus $O(|L||E|C_{\text{loop}})$, where C_{loop} is the cost of manipulating and extending paths in each iteration. A more realistic performance estimate is obtained by observing that the algorithm needs only to run for $|P_{\max}|+1$ iterations, where $P_{\max}$ is the longest candidate path in the circuit. The worst case value of $|P_{\max}|$ is $|L|-1$; in practice it is much lower, and can usually be bounded by a small constant (less than 10).

We have yet to specify C_{loop}, the cost of manipulating path data in each iteration. Two operations must be performed:

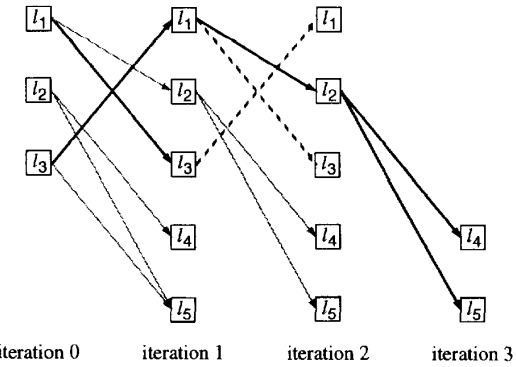


Fig. 20. Candidate path construction.

1) we must extend any candidate paths, and 2) we must determine whether a possible candidate is a loop. We represent paths recursively, reusing subpaths from previous iterations to construct each new candidate path. Fig. 20 illustrates the construction of candidate paths for the example of Fig. 17 and shows the extension of paths performed in each iteration. The dashed arrows represent loops that were identified but not extended, and the boldface arrows mark the final set of candidate paths. Since there can be at most $|E|$ parallel paths extended in each iteration, the cost of each iteration (excluding Step (2) above) remains at worst $O(|E|)$ and the additional storage required in each iteration is also $O(|E|)$. If the number of fanins to each node is bounded, then the storage requirement reduces to $O(|L|)$. Thus the maximum memory utilization is either $O(|L||E|)$ or $O(|L|^2)$. Using $|P_{\max}|+1$ as a bound on the number of iterations, these reduce to $O(|P_{\max}||E|)$ and $O(|P_{\max}||L|)$, respectively.

In each iteration, we must also search the path structures to determine whether a path to be extended contains a specified latch. This adds a significant runtime cost to the algorithm, as each examination involves looking into a tree up to $|P_{\max}|$ levels deep, with each level branching to up to $|L|$ subpaths. Since there are potentially an exponential number of parallel paths, a depth-first search to determine whether they contain a latch l can add an exponential cost to each iteration. However, if we mark already-examined subpaths during the depth-first search, we can reduce the lookup cost to at worst $O(|P_{\max}||E|)$,[3] the maximum size of the path storage. If the number of parallel paths is small, the $|E|$ term reduces to a small constant, keeping the lookup cost low when $|P_{\max}|$ 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 VI. However, the worst-case performance of the algorithm is $O(|L|^2|E|^2)$ as a result of the loop checks.

An alternative to the above approach is to represent paths as bit vectors in which each bit corresponds to a latch and is set to 1 if the latch is in the path and 0 if it is not. This allows lookups to be performed in constant time, but each path extension involves copying $|L|$ bits. This is clearly impractical for even moderately-sized circuits and unlike the previous approach, costs are not reduced when path lengths are small.

[3]Or $O(|L|^2)$ if the number of latch fanins is bounded.

For these reasons, we chose to use the tree-based approach in our implementation and in the experiments presented in Section VI.

D. Comments and Limitations of the Path Extension Algorithms

As illustrated in Fig. 20, the construction of candidate paths is very similar to the unrolled timing constraint graphs of Wallace and Sequin's ATV program [18] and the computational expansions described by Ishii and Leiserson [1]. Both approaches expand sequential circuits into equivalent acyclic circuits to obtain timing constraints. The differences among approaches are in how they relate to critical loops in the constraint graph. In ATV, Wallace and Sequin limit the unrolling with a user-specified parameter and do not describe the possibility of cyclic timing constraints. Ishii and Leiserson handle these constraints implicitly by using the Bellman-Ford algorithm to check for the existence of a violated loop; they do not address the problem of identifying critical paths, although the maximum-weight cycle can be easily identified using the methods of Section IV-C.

In developing the path extension approach, our attempt was to identify as many violated paths as possible when the desired cycle time was below the critical loop frequency $T_{c,\text{loop}}$. However, we cannot guarantee that all violated paths will be identified, since to make the algorithm practical, we only extend *candidate* paths; these are the paths that produce the latest (or earliest) arrival times at each latch. Paths that produce earlier arrival times are eliminated; although they may also contain timing violations, the magnitude of these violations will be smaller than those of the candidate paths.

The check for cycles adds a significant cost to the extension procedure. When there are violated loops in the circuit, these checks identify loops and allow us to directly compute the minimum cycle time for the loop and the delay reduction necessary to make each loop feasible. However, we observed in Fig. 17 that critical loops can be identified more than once; in fact, the algorithm will identify a loop containing n latches n separate times, once at each latch in the loop.

At and above the critical loop frequency $T_{c,\text{loop}}$, the arrival and departure time calculations are equivalent to those of the relaxation algorithm SIMPLE-RELAX. At cycle times above $T_{c,\text{loop}}$, no loops will appear in the analysis; all will be broken by the clock or subsumed by another path. When the cycle time is equal to $T_{c,\text{loop}}$, the path extension approach will not propagate signal times to the next latch l when a path rejoins itself to become a loop; however, since the loop corresponds to a zero-weight cycle, eliminating this propagation will not affect the timing at latch l. As a result, we see that Algorithm SIMPLE-RELAX can be derived from Algorithm EXTEND-LATE by simply removing the path extension and cycle checking functions. The upper bounds on the number of iterations required by both algorithms is the same, and are in both cases due to the maximum path length in the circuit. After $|L|-1$ iterations, there are no more latches for Algorithm EXTEND-LATE to add to an existing acyclic path. Similarly, after $|L|-1$ iterations, Algorithm SIMPLE-RELAX will have computed the timing of the longest possible critical long path, and failure of the arrival and departure times to stabilize will only be due to a violated critical loop. This same line of reasoning was used by Szymanski and Shenoy [4] in their Lemma 2.5 which proved that the arrival and departure times computed by Algorithm SIMPLE-RELAX will converge in at most $|L|-1$, if they converge at all.

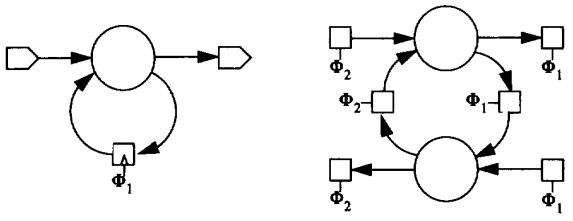


Fig. 21. Transformation to obtain two-phase circuits.

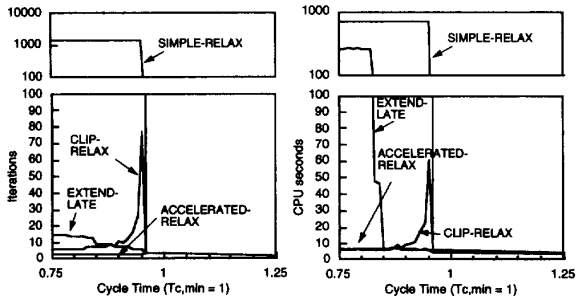


Fig. 22. Iterations and CPU time required to verify s15850.

VI. Timing Verification Experiments

To explore the different procedures for timing verification and critical path identification, we tested our implementations on circuits from the ISCAS89 sequential benchmarks and on circuits obtained from the Michigan High Performance Microprocessor project [19]. The microprocessor circuits included the result register data path (rddpath) and the program counter datapath (pcdpath) and were two-phase level-sensitive designs. They were both of moderate size: the rddpath circuit contained 572 gates and 82 latches and pcdpath contained 4367 gates and 512 latches. Because the ISCAS89 circuits are single-phase edge-triggered circuits, we performed the transformation used by Szymanski to convert them to two-phase level-sensitive circuits [5]. The transformation is illustrated in Fig. 21. Combinational logic blocks were duplicated and placed between alternating Φ_1 and Φ_2 latches and inputs and outputs were also duplicated and replaced with level-sensitive latches. The transformed ISCAS89 circuits ranged in size from 20 gates and 16 latches to over 16000 gates and 4166 latches. All circuits were clocked with a symmetric nonoverlapping 2-phase clock with 50% duty cycle. Assuming zero clock skew and hold times, this eliminated the possibility of hold violations and made the maximum feasible cycle time $T_{c,\max} = \infty$. A unit delay model was used for the ISCAS89 circuits, and the microprocessor circuits were analyzed using delays mapped from a commercial process.

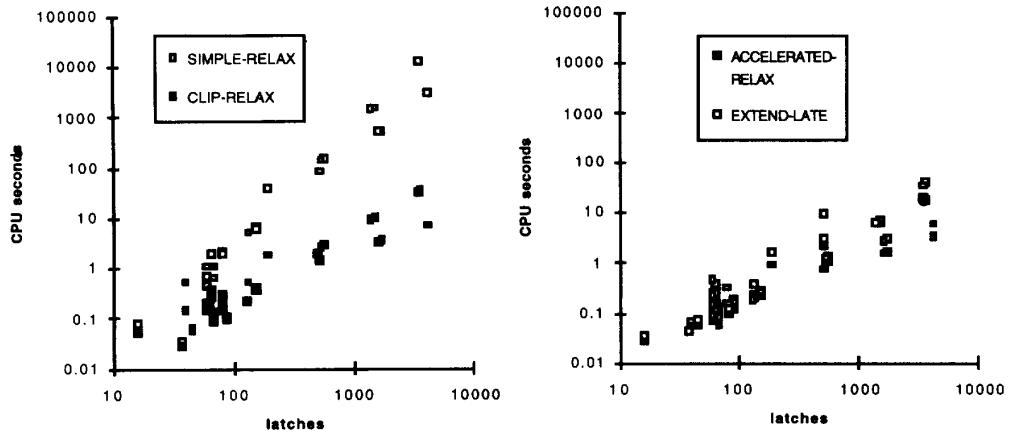


Fig. 23. Runtime performance versus circuit size (verifying at $0.9T_{c,\min}$).

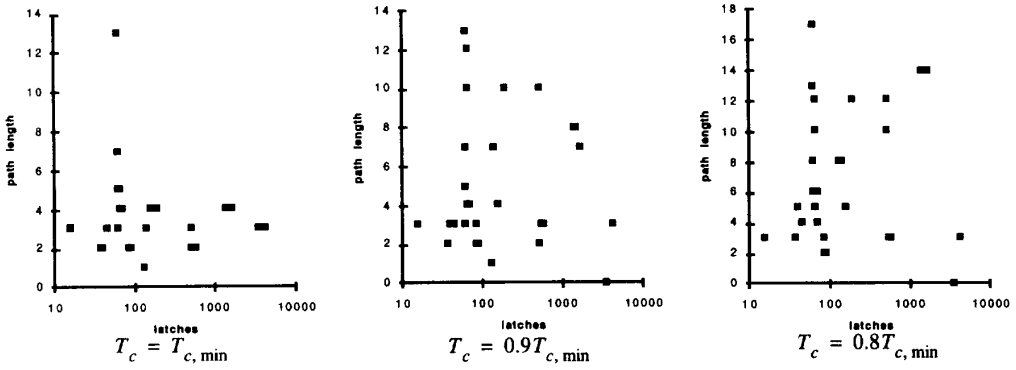


Fig. 24. Maximum path length versus circuit size (in latches).

Fig. 22 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 (SIMPLE-RELAX), the modified relaxation algorithm that clips departure time (CLIP-RELAX), the relaxation algorithm that checks for loops (ACCELERATED-RELAX), and the path extension algorithm (EXTEND-LATE). 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 more latches become transparent, the cost of path extension increases until eventually it becomes impractical.

It was also interesting to compare the run times 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 Fig. 23 on a log-log plot. The plot shows the result of verifying the test circuits 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 $O(l^m)$. 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 the plot in Fig. 24 shows, these path lengths were relatively uncorrelated with circuit size and increased only slightly as the cycle time decreased. Results for the transformed ISCAS89 benchmarks and the microprocessor circuits were similar.

VII. Conclusion

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 the identification of critical paths in combinational circuits. However, as we saw in Section IV-D, 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. As demonstrated in Section VI in some cases this added cost is moderate. However, if the number of transparent latches in a circuit is large, or if a large number of parallel paths have the same delay, the cost of the cycle checks used by the extension algorithm will be prohibitive. Regardless, if the critical loops in a circuit are known to be satisfied or if the circuit is acyclic, the relaxation-based procedure is favored for its simplicity, speed, and the availability of slack and float information that can be used to identify sub-critical paths.

All algorithms described were implemented and tested in a software tool we have developed. Additional work remains to allow our tool to be used in the timing analysis of general designs. This work includes adding the ability to extract combinational subpaths, adding parsers for standard hardware description languages, and adding support for more complex delay models.

References

[1] A. T. Ishii and C. E. Leiserson, "A timing analysis of level-clocked circuitry," in *Proc. 6th MIT Conf.: Adv. Res. VLSI*, 1990, pp. 57–69.

[2] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "checkT_c and min T_c: Timing verification and optimal clocking of synchronous digital circuits," in *ICCAD-90 Digest of Technical Papers*, pp. 552–555, Nov. 1990.

[3] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "A pseudo-polynomial algorithm for verification of clocking schemes," in *TAU '92: Proc. 1992 ACM/SIGDA Workshop, Timing Iss., Spec., Synth. Digit. Syst.*, Mar. 18–20, 1992.

[4] T. G. Szymanski and N. V. Shenoy, "Verifying clock schedules," in *ICCAD-90 Digest of Technical Papers*, pp. 124–131, Nov. 1992.

[5] T. G. Szymanski, "Computing optimal clock schedules," in *Proc. Des. Automa. Conf.*, pp. 399–404, 1992.

[6] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Graph algorithms for clock schedule optimization," in *ICCAD-92 Digest of Technical Papers*, pp. 132–136, Nov. 1992.

[7] J. D. Wiest and F. K. Levy, *A Management Guide to PERT/CPM*. Englewood Cliffs, NJ: Prentice-Hall, 1977, 2nd ed.

[8] K. Lockyer and J. Gordon, *Critical Path Analysis and Other Project Network Techniques*. New York: Pitman, 1991.

[9] T. I. Kirkpatrick and N. R. Clark, "PERT as an aid to logic design," *IBM J. Res. Develop.*, pp. 135–141, Mar. 1966.

[10] R. B. Hitchcock, Sr., G. L. Smith, and D. D. Cheng, "Timing analysis of computer hardware," *IBM J. Res. Develop.*, vol. 26, no. 1, pp. 100–105, Jan. 1982.

[11] E. A. Dinic, "The fastest algorithm for the PERT problem with AND- and OR-nodes (the new product-new technology problem)," in *Integer Programming and Combinatorial Optimization: Proc. Conf., Univ. Waterloo, Math. Programm. Soc.*, Waterloo, Ont., Canada, Univ. Waterloo Press, May 28–30, 1990.

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

[13] T. Burks, K. Sakallah, and T. N. Mudge, "Identification of critical paths in circuits with level-sensitive latches," Univ. Michigan, Ann Arbor, MI, Tech. Rep. CSE-TR-160-93, 1993.

[14] S. Burns, "Performance analysis and optimization of asynchronous circuits," Ph.D. dissertation, California Inst. Technol., Caltech-CS-TR-91-01, 1991.

[15] B. Lockyear and C. Ebeling, "Optimal retiming of multi-phase level-clocked circuits," in *Adv. Res. VLSI Parallel Syst.: Proc. 1992 Brown/MIT Conf.*, 1992, pp. 265–280.

[16] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou, "Optimizing two-phase level-clocked circuitry," in *Adv. Res. VLSI Parallel Syst: Proc. 1992 Brown/MIT Conf.*, pp. 245–264, 1992.

[17] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Resynthesis of multi-phase pipelines," in *Proc. Des. Automa. Conf.*, 1993, pp. 490–496.

[18] D. E. Wallace and C. H. Sequin, "ATV: An abstract timing verifier," in *Proc. Des. Automa. Conf.*, 1988, pp. 154–159.

[19] M. Upton, T. Huff, P. Sherhart, P. Barker, R. Brown, R. Lomax, T. Mudge, and K. Sakallah, "A 160,000 transistor GaAs microprocessor," in *Int. Solid-State Circ. Conf. (ISSCC)*, Feb. 1993, pp. 92–93.

Timothy M. Burks (S'86–M'93) received the B.S. degree in computer engineering from Virginia Polytechnic Institute and State University, Blacksburg, VA, in 1989, the M.S.E. degree in electrical engineering, and the Ph.D. degree in computer sccience and engineering in 1991 and 1994, respectively, both from the University of Michigan, Ann Arbor, MI.

His graduate work was supported by a National Defense Science and Engineering Graduate Fellowship from 1989 to 1993. Since December 1993, he has been with the IBM Corporation at the Somerset Design Center, Austin, TX, developing software tools for microprocessor circuit performance analysis and optimization.

Karem A. Sakallah (S'76–M'81–SM'92) received the B.E. degree (with Distinction) in electrical engineering from the American University of Beirut, Beirut, Lebanon, in 1975, and the M.S.E.E. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1977 and 1981, respectively.

In 1981, he joined the Department of Electrical Engineering at CMU as a Visiting Assistant Professor. From 1982 to 1988 he was with the Semiconductor Engineering Computer-Aided Design Group at Digital Equipment Corporation in Hudson, MA, where he headed the Analysis and Simulation Advanced Development team. Since September 1988, he has been at the University of Michigan, Ann Arbor, MI, as Associate Professor of Electrical Engineering an Computer Science. His research interests are primarily in the area of computer-aided design of integrated circuits and systems, with particular emphasis on numerical analysis, multilevel simulation, timing verification and optimal clocking, modeling, knowledge abstraction, and design environments.

Dr. Sakallah is a member of the ACM .

Trevor N. Mudge (S'74–M'77–SM'84) received the B.S. degree in cybernetics from the University of Reading, England, in 1969, and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana in 1973 and 1977, respectively.

Since 1977, he has been on the Faculty of the University of Michigan, Ann Arbor. He is presently a Professor of Electrical Engineering and Computer Science, and the Director of the Advanced Computer Architecture Laboratory—A group of eight faculty and 80 graduate research assistants. He is author of more than 120 papers on computer architecture, programming languages, VLSI design, and computer vision, and he hods a patent in computer aided design of VLSI circuits.

Dr. Mudge is a member of the ACM, a member of the IEE, and a member of the British Computer Society. He is also Associate Editor for *ACM Computing Surveys*, and a member of the Editorial board of IEEE Parallel and Distributed Technology.