A Timing Analysis Algorithm for Circuits with Level-Sensitive Latches

Jin-fuw Lee, Donald T. Tang, and C. K. Wong

IBM Thomas J. Watson Research Center,
Yorktown Heights, NY 10598

Abstract

For a logic design with level-sensitive latches, we need to validate timing signal paths which may flush through several latches. We developed efficient algorithms based on the modified shortest and longest path method. The computational complexity of our algorithm is generally better than that of known algorithms in the literature. The implementation (CYCLOPSS) has been applied to an industrial chip to verify the clock schedules.

1. Introduction

For the logic design of a digital computer using edge-triggered flip-flops, signal delays through all combinational logic paths must be less than the clock period. (See Figure 1(b)). For the timing verification of such a design, one needs only to perform a path analysis of combinational logic network within one clock cycle [3]. For designs using level-sensitive latches, signals may flush through latches. This permits the use of a combinational logic path with delay longer than one clock cycle, so long as it is compensated by shorter path delays in the subsequent cycles. This technique is called cycle stealing or slack sharing. An example is shown in Figure 1(c). Since a flush path may extend to many cycles, the path analysis for timing verification becomes more complex.

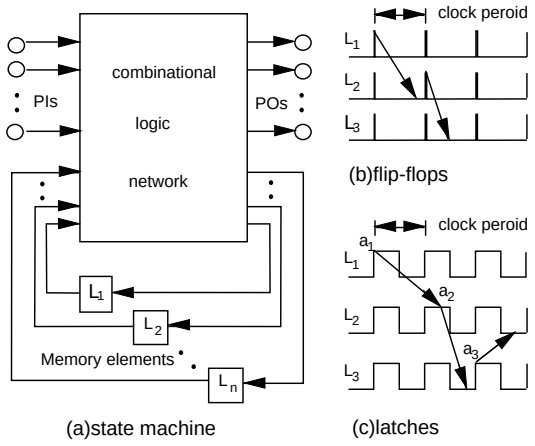


Figure 1. A finite-state machine. (a)Block diagram. (b)Flip-flops. (c)Latches. The delay from L_1 to L_2 is longer than one clock cycle.

A general formulation of clock schedule verification and optimization problem in terms of timing constraints was presented by Sakallah, Mudge, and Olukotun in [1]. These SMO timing constraints have been applied to many timing problems, such as timing verification [1,2,4], clock optimization [5,6], and retiming, etc. Pattern-independent timing analysis programs were developed in [1,4] to calculate the earliest and latest arrival times satisfying SMO timing constraints, with a worst case bound O(1/) where is the smallest loop gain. Szymanski and Shenoy in [2] made an elegant analysis of SMO timing constraints, and developed a timing verification algorithm with an improved worst case bound $O(N \times |G|)$, where N is the number of latches and G is the set of timing constraints. In these algorithms, NP-hard false path problem is not considered.

In this paper, we shall show that above timing verification problem can be transformed into a modified longest path and shortest path problem on a latch delay graph. We then use a modified Bellman-Ford method to find the earliest and latest arrival times. This algorithm has a worst case bound $O(b \times |G|)$ where b is the number of latches which have fan-in feedback edges. This bound is better than the best worst-case bound O($N \times |G|$) in the literature. Experimental results also show that the average run times of this algorithm are significantly better.

2. Timing Model

A finite-state machine consists of a combinational logic network, memory elements(latches or flip-flops), and primary inputs (PIs) and outputs (POs) as shown in Figure 1. Even though the combinational logic network is acyclic, the feedback through memory elements introduces loops in such a design. The presence of loops makes the timing analysis more difficult. For example, the traditional timing verification method [3] based on a good topological ordering of circuit elements does not apply. Another consequence of feedback loops is a big increase in the number of signal paths. To analyze timing properties, it is convenient to abstract such a network with a latch graph, as shown in Figure 2(a). Here a node represents either a memory element of the network, a PI, or a PO, while an edge represents a pair of worst-case early and late path delays. Flip-flops may be regarded as special cases of latches in which the region of active phase shrinks to a sharp edge as shown in Figure 1(b). Each latch is associated with a clock waveform, which is characterized by a clock period, , a setup time, S_i, a hold time, H_i, a latch opening time B_i, and a latch closing time F_i. Each edge is associated with a ***short-path weight*** $_{i,j} = _{i,j} - E_{i,j}$ and a ***long-path weight*** $\Lambda_{i,j} = \Delta_{i,j} - E_{i,j}$, where $_{i,j}$ and $\Delta_{i,j}$ represent respectively the minimum and maximum combinational logic delays from latch i to latch j, and $E_{i,j}$ represents the time-zone adjustment [1,2] from latch i to latch j.

A typical latch has three sets of pins: input and output pins for data signals, and input pins for clock signals. The data paths through a latch can be represented by a set of edges, each of

which is directed from a data input pin to the corresponding data output pin, as shown in Figure 2(b). Each edge is also associated with weights $\delta_{i,j}$ and $\Lambda_{i,j}$ which are respectively the minimum and maximum internal latch delays. With this decomposition of latches into their constituent data pins, the latch graph is transformed into a latch delay graph $G = (V, E)$, in which nodes are either PIs, POs, or latch data pins, as shown in Figure 2(c). On G, latch output nodes are drawn as ***triangles*** and all the other nodes are drawn as ***circles***. We made this distinction because data signals arriving at circular nodes immediately continue their propagation, while signals arriving at the triangular nodes are regulated by the clock in the following way: If a data signal arrives before the latch opens, its propagation from the latch output has to wait until the opening time of the latch. If a signal arrives after the latch opens, it propagates (flushes) through the latch immediately. We shall call it a ***non-flush*** signal in the former case, and a ***flush*** signal in the latter case. For example in Figure 1(c), a_2 is a flush signal, while a_1 and a_3 are non-flush signals.

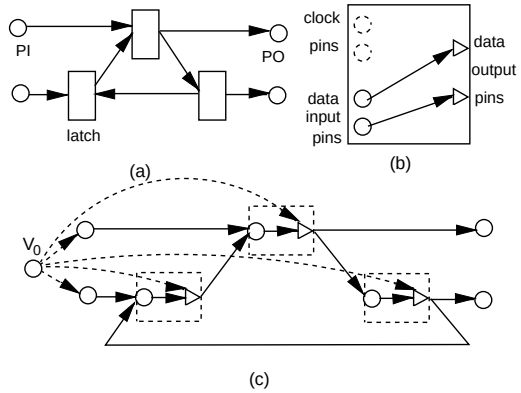


Figure 2. Timing model: (a)A latch graph (b)The latch model. (c)The latch delay graph.

The worst case times satisfy SMO timing constraints [1]:

$$\begin{aligned} d_i &= \max(a_i + \delta_i, B_i); \quad a_i = \min_{j \to i}(d_j + \delta_{j,i}). \\ D_i &= \max(a_i + \Lambda_i, B_i); \quad A_i = \max_{j \to i}(D_j + \Lambda_{j,i}). \end{aligned} \tag{1}$$

where d_i, D_i, a_i, A_i, δ_i and Λ_i are respectively the earliest and latest signal departure times, the earliest and latest signal arrival times, and the internal short and long path delays of latch i. Latch ***hold*** and ***setup*** constraints for correct machine operation are then respectively $a_i \geq F_i - \delta_i + H_i$ and $A_i \leq F_i - S_i$.

On G, we add a ***global source*** node V_0 to represent the time origin and an edge from V_0 to each PI node V_j, as shown in Figure 2(c), with user-asserted early and late arrival times as short and long edge weights. This way, all the signal paths originated from PI nodes may extend to the common source node. To account for the signal paths originated from latches, we add an edge from V_0 to each latch output node V_j, as shown in Figure 2(c), with F_j and B_j as short and long edge weights. The long weight B_j may be regarded as the lower bound of the late arrival times, while F_j may be regarded as the upper bound of early signals. On G, V_0 becomes the single source node from which all signal paths originated. Next, we shall modify the definition of the path length to reflect the signal arrival times through each such path.

Definition 1: Short path length, $l(p)$, of a path p is defined as the sum of weights $\delta_{i,j}$ along its edges. Similarly, long path length, $L(p)$, is defined as the sum of weights $\Lambda_{i,j}$ along its edges.

Definition 2: Let path p be $U_1U_2U_3 \ldots U_n$ ($U_1 = V_0$, and $U_n = V_i$) and its sub-paths p_m be $U_1U_2 \ldots U_m$ with $m \leq n$. The early and late signal arrival times along path p are respectively defined by two new "path length" functions, $a(p)$ and $A(p)$:

$$\begin{aligned} a(p_1) &= 0; \\ a(p_m) &= \begin{cases} \max(a(p_{m-1}) + \delta_m, B_m) & \text{for triangular } U_m; \\ a(p_{m-1}) + \delta_m & \text{for circular } U_m. \end{cases} \\ A(p_1) &= 0; \\ A(p_m) &= \begin{cases} \max(A(p_{m-1}) + \Lambda_m, B_m) & \text{for triangular } U_m; \\ A(p_{m-1}) + \Lambda_m & \text{for circular } U_m. \end{cases} \\ \delta_m &= \delta(U_{m-1}, U_m); \quad \Lambda_m = \Lambda(U_{m-1}, U_m). \end{aligned} \tag{2}$$

where functions $\delta(V_s, V_t)$ and $\Lambda(V_s, V_t)$ are respectively defined as the short-path and long-path weights of edge (V_s, V_t).

Definition 3: Let p_1 and p_2 be two paths from V_0 to V_i. We say that p_1 is ***earlier*** (or ***later***) than p_2 if and only if $a(p_1) < a(p_2)$ (or $A(p_1) > A(p_2)$). The ***earliest*** and the ***latest*** arrival times among various paths from V_0 to V_i on G are respectively defined as a_i and A_i:

$$a_i = \min_p(a(p)); \quad A_i = \max_p(A(p)). \tag{3}$$

Lemma 1: The earliest and the latest arrival times, A_i and a_i, satsify Equation (4).

$$\begin{aligned} a_i &= \left\{ \begin{array}{l} \min_{j \to i}(a_j + \delta_{j,i}) \text{ for circular nodes;} \\ \max(B_i, \min_{j \to i}(a_j + \delta_{j,i})) \text{ for triangular nodes.} \end{array} \right\} \\ A_i &= \left\{ \begin{array}{l} \max_{j \to i}(A_j + \Lambda_{j,i}) \text{ for circular nodes;} \\ \max(B_i, \max_{j \to i}(A_j + \Lambda_{j,i})) \text{ for triangular nodes.} \end{array} \right\} \end{aligned} \tag{4}$$

For circular nodes, Equation (4) is exactly the same as Equation (1). In the following, we shall show that for triangular nodes, Equation (4) is also equivalent to Equation (1). Let V_i be a latch output node. Its two fan-in nodes are V_k (the corresponding latch input node) and V_0. For the late mode, we have $A_i = \max(B_i, \max_{j \to i}(A_j + \Lambda_{j,i})) = \max(B_i, A_k + \Lambda_{k,i})$. This is exactly the SMO late mode timing constraint on latch i. For the early mode, we have $a_i = \max(B_i, \min_{j \to i}(A_j + \delta_{j,i})) = \max(B_i, \min(F_i, a_k + \delta_{k,i}))$. This reduces to the SMO early mode timing constraint, for the normal operation in which $a_k + \delta_{k,i} \leq F_i$. When $a_k + \delta_{k,i}$ falls beyond the window bound, F_i, Equation (4) clips it back to F_i. Therefore, the timing verification problem is transformed into the latest and earliest path problem on G with modified "path length" functions $A(p)$ and $a(p)$.

For the case in which G contains no feedback loops, G is a directed acyclic graph which may be topologically sorted. A ***block-oriented*** algorithm in [3] may be used to generate the arrival times by using Equations (2) and (3), instead of the usual

path lengths. In the general case with feedback, the nodes in G may be sorted in the following way:

Algorithm SORT(G)

1. Generate a depth-first spanning tree. A $O(G)$ algorithm for the depth-first tree can be found in [7]. Let E_b be the set of back edges as shown in Figure 3(a).
2. Since the sub-graph (V, $E_f = E - E_b$) is acyclic, we can make a breadth-first traversal on E_f to order the nodes : V_0, V_1, ... This operation is again $O(G)$.

With respect to this sort order, edges (V_i, V_j) in E_f are ***forward-directed*** ($i < j$), while edges in E_b are ***backward-directed*** ($i > j$), as shown in Figure 3(b). Let us call those nodes with fan-in backward-directed edges ***feedback nodes***. For example, V_1 is the only feedback node in Figure 3(b).

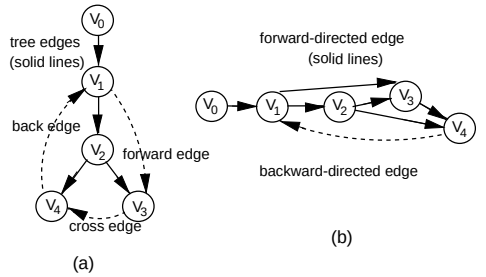


Figure 3. SORT: (a)Depth-first spanning tree. (b)Breadth-first order.

Lemma 2: Algorithm *SORT* yields a sort order with the number of feedback nodes, $b = |\{V_j | (V_i, V_j) \in E_b\}| \le N$ (the number of latches). The complexity of *SORT* is $O(G)$.

Proof: We observe that each feedback node has at least one forwarded-directed fan-in edge, at least one backward-directed fan-in edge, and at least one forward-directed fan-out edge. Then it is evident that V_0, PI, or PO can not be a feedback node. Also the input and output node of a latch can not both be feedback nodes. Hence $b \le N$. **Q.E.D.**

Remark: Algorithm *SORT* does not guarantee that b is a minimum. (Minimum feedback arc set problem is NP hard). However, b is usually much less than N. See Table 1 for numerical examples.

3. Loop analysis

Consider a loop, $c = U_1U_2 \ldots U_{2n}$, which contains n latches with odd-numbered vertices as input nodes, and even-numbered vertices as output nodes. Let p_0 be a path from V_0 to U_1, and path $p_i^{(k)}$ be a path consisting of path p_0, $k-1$ loops around c, and path $U_1 \ldots U_i$, i.e., $p_i^{(k)} = p_0 c^{k-1} U_1 \ldots U_i$. Let $a_i^{(k)} = a(p_i^{(k)})$, and $A_i^{(k)} = A(p_i^{(k)})$. An early(late) signal at the latch output nodes is flush, if $a_{2i}^{(k)} > B_{2i}$ ($A_{2i}^{(k)} > B_{2i}$), and non-flush, if $a_{2i}^{(k)} = B_{2i}$ ($A_{2i}^{(k)} = B_{2i}$). Equation (2) can be rewritten in terms of $a_i^{(k)}$:

$$a_i^{(k)} = \begin{cases} a_{i-1}^{(k)} + (U_{i-1}, U_i) & \text{if } i = \text{odd}; \\ \max(a_{i-1}^{(k)} + (U_{i-1}, U_i), B_i) & \text{if } i = \text{even}. \end{cases}$$

$$A_i^{(k)} = \begin{cases} A_{i-1}^{(k)} + \Lambda(U_{i-1}, U_i) & \text{if } i = \text{odd}; \\ \max(A_{i-1}^{(k)} + \Lambda(U_{i-1}, U_i), B_i) & \text{if } i = \text{even}. \end{cases} \quad (5)$$

Definition 4: Short loop gain, $l(c)$, of a loop c is defined as the sum of weights $_{i,j}$ along its edges. Similarly, long loop gain, $L(c)$, is defined as the sum of weights $\Lambda_{i,j}$ along its edges.

Lemma 3: Signal arrival times in consecutive traversals around loop c satisfy the inequality: $a_i^{(k)} \ge a_i^{(k-1)} + l(c)$, and $A_i^{(k)} \ge A_i^{(k-1)} + L(c)$. Equality holds when the signal flushes through all latches during the path from the $(k-1)$th loop traversal of U_i to the (k)th loop traversal of U_i.

Proof is omitted here.

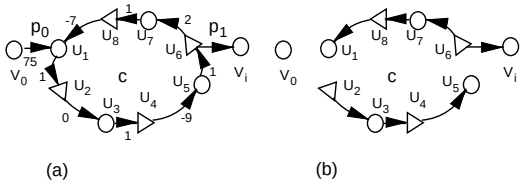
The monotonic relation obtained by Szymanski and Shenoy [2] may be applied to $a_i^{(k)}$ and $A_i^{(k)}$ in the following form: $\forall i, k$

$$\begin{array}{l} a_i^{(k+1)} \le a_i^{(k)} \text{ if } a_1^{(2)} < a_1^{(1)};\ A_i^{(k+1)} \le A_i^{(k)} \text{ if } A_1^{(2)} < A_1^{(1)} \\ a_i^{(k+1)} = a_i^{(k)} \text{ if } a_1^{(2)} = a_1^{(1)};\ A_i^{(k+1)} = A_i^{(k)} \text{ if } A_1^{(2)} = A_1^{(1)} \\ a_i^{(k+1)} \ge a_i^{(k)} \text{ if } a_1^{(2)} > a_1^{(1)};\ A_i^{(k+1)} \ge A_i^{(k)} \text{ if } A_1^{(2)} > A_1^{(1)} \end{array} \quad (6)$$

Hence, both $\{a_i^{(1)}, a_i^{(2)}, \ldots, a_i^{(\infty)}\}$ and $\{A_i^{(1)}, A_i^{(2)}, \ldots, A_i^{(\infty)}\}$ are monotonic sequences.

3.1 The short loop path problem

Problem: Among the family of paths $\{p_i^{(k)} | k = 1..\infty\}$ to node U_i, find the path with the smallest $\bar{a}_i = \min\{a_i^{(k)} | k = 1, 2..\}$, and the first $\bar{k} = \min\{k | a_i^{(k)} = \bar{a}_i\}$, if there are ties.



k	V_0	U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8
1	0	75	76	76	77	68	69	71	72
2	0	65	66	66	67	58	59	61	62
3	0	55	56	56	57	48	50	52	53
4	0	46	50	50	51	42	50	52	53

(c)

Figure 4. A short loop path example. (a)Latch delay graph (b)Flush paths in $\{\bar{a}_i\}$. (c)The (k)th row lists $a_i^{(k)}$. Squares mark the first occurrences of minimum arrival times.

Lemma 4

1. For the case $a_1^{(2)} \ge a_1^{(1)}$, we have $\bar{a}_i = a_i^{(1)}$, and $\bar{k} = 1$.
2. For the case $a_1^{(2)} < a_1^{(1)}$, we have $l(c) < 0$, with $\bar{a}_i$ and $\bar{k}$ given by Equation (7-9).

Proof: Part 1 of Lemma 4 follows from the fact that $a_i^{(k)}$ is monotonically non-decreasing in k.

We next consider the case $a_1^{(2)} < a_1^{(1)}$. Lemma 3 implies $l(c) < 0$. In this case, $a_i^{(k)}$ is monotonically non-increasing in k. This means that a non-flush signal during the (k)th loop, say $a_{2i}^{(k)} = B_{2i}$, will remain non-flush during the $(k+1)$th loop ($a_{2i}^{(k+1)} = B_{2i}$). Therefore once we reach the first non-flush signal, say $a_{2i_0}^{(k_0)}$, the signal arrival times start to converge to minimum values, i.e.,

$$\begin{array}{ll} \bar{a}_i = a_i^{(k_0)}; & \bar{k} = k_0 \quad \text{for } i \ge 2i_0. \\ \bar{a}_i = a_i^{(k_0+1)}; & \bar{k} = k_0 + 1 \quad \text{for } i < 2i_0. \end{array} \tag{7}$$

k_0 can be derived as follows: The signal flushes through all the latches on loop c, until $k = k_0$ and $i = 2i_0$. During this flush period, $a_i^{(k)}$ is reduced by the amount $|l(c)|$, each time when a new loop is completed, according to Lemma 3. Let $\Delta_i = (a_{2i}^{(1)} - B_{2i})/|l(c)|$, and let $\lceil\ \rceil$ be the ceiling function. During the $(k_0 - 1)$th loop traversal to node U_{2i}, we have a flush signal $a_{2i}^{(k_0-1)} = a_{2i}^{(1)} + (k_0 - 2) \times l(c) > B_{2i}$, which implies that $k_0 - 1 < 1 + \min \Delta_i$. So $k_0 - 1 \le \min\lceil \Delta_i \rceil$. Also $a_{2i_0}^{(k_0)}$ is non-flush. This implies that $B_{2i_0} = a_{2i_0}^{(k_0)} \ge a_{2i_0}^{(1)} + (k_0 - 1) \times l(c)$ and $\lceil \Delta_{i0} \rceil \le k_0 - 1$. Hence,

$$k_0 = \min_i \lceil \Delta_i \rceil + 1; \qquad \Delta_i = (a_{2i}^{(1)} - B_{2i})/|l(c)|. \tag{8}$$

To find i_0, we observe that during the (k_0)th loop, the signal flushes through latches until it reaches node U_{2i_0}. So for $i < i_0$, the quantity, $r_i = a_{2i}^{(1)} + (k_0 - 1) \times l(c) - B_{2i}$, is positive, since it is equal to $a_{2i}^{(k)} - B_i$. For $i = i_0$, r_i becomes non-positive since $B_{2i_0} = a_{2i_0}^{(k_0)} \ge a_{2i_0}^{(1)} + (k_0 - 1) \times l(c)$. Hence,

$$i_0 = \min\{i \,|\, r_i \le 0\}; \quad r_i = a_{2i}^{(1)} + (k_0 - 1) \times l(c) - B_{2i}. \tag{9}$$

Q.E.D.

Figure 4(a) shows a 4-latch example of the case $a_1^{(2)} < a_1^{(1)}$. Only the short-path weights are shown, and the short loop gain $l(c) = -10$. The clock period is 100, setup times and hold times are 0, and the latch opening times are $B_2 = B_4 = B_6 = B_8 = 50$. The early arrival times for first four loop traversals are shown in Figure 4(c). From $\Delta_1 = 2.6$, $\Delta_2 = 2.7$, $\Delta_3 = 1.9$ and $\Delta_4 = 2.2$, we have $k_0 = 1 + \lceil 1.9 \rceil = 3$. From $r_1 = 6$, $r_2 = 7$, $r_3 = -1$ and $r_4 = 2$, we derive $i_0 = 3$.

In the second case of Lemma 4, if $l(c) = -\varepsilon$ is a very small negative number, then $k_0 \propto 1/\varepsilon$ will become very large. This is why earlier algorithms [1,4] may need many iterations to converge. With Equation (7-9), we can quickly calculate the earliest arrival times after the loop is traversed the first time.

3.2 The long loop path problem

Problem: Among the family of paths $\{p_i^{(k)} \,|\, k = 1..\infty\}$ to node U_i, find the path with the largest $\bar{A}_i = \max\{A_i^{(k)} \,|\, k = 1, 2..\}$, and the first $\bar{k} = \min\{k \,|\, A_i^{(k)} = \bar{A}_i\}$, if there are ties.

Lemma 5

1. For the case $A_1^{(2)} \le A_1^{(1)}$, we have $L(c) \le 0$, $\bar{A}_i = A_i^{(1)}$, and $\bar{k} = 1$.
2. For the case $A_1^{(2)} > A_1^{(1)}$ and $L(c) \le 0$, we have $\bar{A}_i = A_i^{(1)}$ and $\bar{k} = 1$ for $i \ge i_0$; $\bar{A}_i = A_i^{(2)}$ and $\bar{k} = 2$ for $i < i_0$, where $i_0 = \min\{i \,|\, A_{2i}^{(2)} = B_{2i}\}$.
3. For the case $A_1^{(2)} > A_1^{(1)}$ and $L(c) > 0$, we have $A_i^{(k)} = A_i^{(2)} + (k - 2) \times L(c)$ which is unbounded in k, and $\bar{A}_i$ and $\bar{k}$ do not exist.

Proof: The case $A_1^{(2)} \le A_1^{(1)}$. follows from the fact that $A_i^{(k)}$ is monotonically non-increasing in k.

We next consider the case $A_1^{(2)} > A_1^{(1)}$. $A_i^{(k)}$ is monotonically non-decreasing in k. If $L(c) > 0$, then according to Lemma 3, $A_{2i}^{(2)} > A_{2i}^{(1)} \ge B_{2i}$ which means that the computed signal arrival times are all flush in the second loop and hence also in the subsequent loops. By the repeated application of Lemma 3, we obtain $A_i^{(k)} = A_i^{(2)} + (k - 2) \times L(c)$, which is unbounded in k. Figure 5(a) shows such an example. On the other hand, if $L(c) \le 0$, then not all signals in the second loop are flush. Let the first non-flush signal be $A_{2i_0}^{(2)}$. Then $A_{2i_0}^{(1)}$ must be also non-flush. So $A_i^{(k)} = A_i^{(1)}$ for $i \ge i_0$, and $A_i^{(k)} = A_i^{(2)}$ otherwise. Figure 5(b) shows an example. **Q.E.D.**

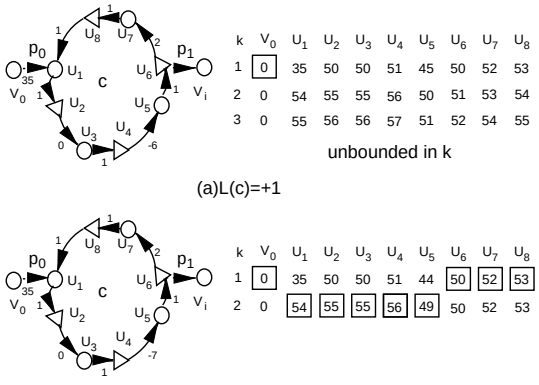


Figure 5. Long loop path examples (a)A positive loop. (b)A zero-gain loop. Squares mark the maximum arrival times.

In the third case of Lemma 5, if $L(c) = -\varepsilon$ is a very small positive number, then after each loop $A_i^{(k)}$ grows by ε, and it will take a large number of loops $k_{\max} = O((\ - S_i - B_i)/\varepsilon)$ for $A_i^{(k)}$ to reach the time which violates the setup constraint. Therefore when there are positive loops, algorithm in [2] will run until k is equal to the number of latches in G, and algorithms in [1,4] will run until k reaches the bound $k_{\max}$.

4. Latest arrival times (latest paths)

Lemma 6: The latest arrival time, A_i, defined by Equation (3,4), is the same as the longest path length from V_0 to V_i on G.

Proof: Let us re-examine Equation (4). Let V_i be a latch output node and V_j its corresponding latch input node. Since the long weight on (V_0, V_i) is B_i, $\max(A_j + \Lambda_{j,i}) \ge B_i$, and hence $A_i = \max(A_j + \Lambda_{j,i})$ also for triangular $\overrightarrow{j\ i}$ nodes. Therefore A_i is indeed equal to the longest path length from V_0 to V_i. **Q.E.D.**

Therefore, the late-mode timing problem is equivalent to the longest path problem, which has been studied extensively in graph theory. In Algorithm *LATE*, Yen's modification [9] to the Bellman-Ford method is adopted to take advantage of the sort order provided by Algorithm *SORT*. The search for the longest paths is done iteratively with alternating passes through E_f and E_b. During the ***forward pass*** through edges in E_f, the nodes are examined in the breadth-first order as produced by algorithm *SORT*. During the ***backward pass*** through edges in E_b, nodes are examined in the reverse order. If an edge (V_j, V_i) examined generates a new longest path, then A_i is updated by the new longest path length. Also the ***dominant predecessor*** t_i of V_i is set

to V_j. The dominant graph T formed from these dominant edges (t_i, V_i) contains $|V| - 1$ edges (V_0 has no predecessor). Such a graph may either be a tree which represents the longest paths, or contain loops all of which have positive gains. (A proof can be found in [8]). We shall illustrate this with two examples: For the example with a positive-gain loop in Figure 6(a), T contains the loop. For the example without a positive loop in Figure 6(b), T is a longest path tree.

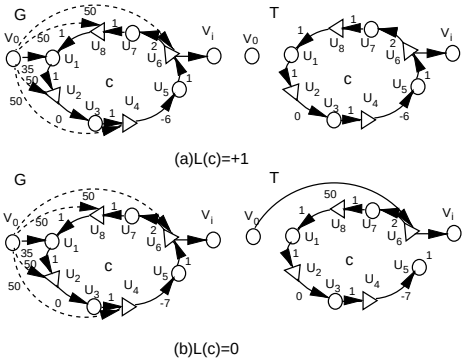


Figure 6. Dominant graphs: (a)T contains a positive loop. (b)T is a tree.

Algorithm LATE(G)

1. Initialize $A_0 = 0$ and other A_i as undefined. Set the iteration counter m=0.
2. For i=1, ...n do For each edge (V_i, V_j) in E_f do LateLabel(i, j, *converge*). /*generate the longest path tree in E_f */
3. Set *converge* = true, and m = m+1.
4. For each edge (V_i, V_j) in E_b do LateLabel(i, j, *converge*).
5. For i=1, ...n do For each edge (V_i, V_j) in E_f do LateLabel(i, j, *converge*).
6. If loops are found in T, report the positive loops and exit.
7. If *converge* = false, go to Step 3.
8. End {LATE}

Procedure LateLabel(i, j, *converge*)

1. If A_j is undefined or $A_j < A_i + \Lambda_{i,j}$, then set $A_j = A_i + \Lambda_{i,j}$, $t_j = V_i$, and *converge* = false.
2. End {LateLabel}

Theorem 1 (Latest path)

1. If there are no positive loops in G, Algorithm *LATE* will converge in a finite number of iterations $m_L \le b$.
2. Otherwise, some positive loops will be found in T in the m_L*th* iteration, with $m_L \le b + 1$.

Proof: Let $Q^{(m)}$ be the set of paths which contains m or fewer backward-directed edges. It can be shown by mathematical induction that at the end of the (m)th iteration, $\{A_i\}$ consists of the longest path lengths in $Q^{(m)}$, and the maximum number of backward-directed edges among paths in T is m. Let $p^{(m)}$ be such a path which contains m feedback nodes. If the iteration continues to $m = b + 1$, then $p^{(b+1)}$ must contain duplicated feedback nodes(i.e. a loop). For the case that G does not contain positive loops, this can not be true, and Part 1 of Theorem 1 follows. For Part 2 of Theorem 1, for some $m \le b + 1$, $p^{(m)}$ must contain a loop, and stop the iteration. **Q.E.D.**

The computational complexities for Step 4, 5 and 6 of Algorithm *LATE* are linear in $|V| + |E_b|$, $|V| + |E_f|$, $|T|$ respectively. The overall computational complexity of Algorithm *LATE* is $O(m_L \times |G|)$ with a worst-case bound $O(b \times |G|)$. After detecting a positive loop, we may break an appropriate edge in the loop. This way, Algorithm *LATE* can be continued with all positive loops elliminated. [10]

5. Earliest arrival times (earliest paths)

Lemma 7: The search of earliest paths can be restricted to paths which contain (1)no loop or (2)one simple loop.

The proof is omitted here. It may be constructed by the mathematical induction on the number of distinct loops on an earliest path. An example of a path with one simple loop is shown in Figure 4(a). Let P be the set of paths which satisfy Lemma 7. Let $B(p)$ be the number of distinct backward-directed edges in path p, and $P^{(k)}$ the subset $\{p \mid p \in P \text{ and } B(p) \le k\}$. Then $P^{(b)} = P$, since $B(p) \le b$ for any path p.

Algorithm EARLY(G)

1. Initialize $a_0 = 0$ and other a_i as undefined. Set the iteration counter m=0.
2. For i=1, ...n do For each edge(V_i, V_j) in E_f do EarlyLabel(i, j, *converge*).
3. Set *converge* = true, and m = m+1.
4. For each edge(V_i, V_j) in E_b do EarlyLabel(i, j, *converge*).
5. For i=1, ...n do For each edge(V_i, V_j) in E_f do EarlyLabel(i, j, *converge*).
6. For each loop found in T, apply Equation (7-9) and set t_j to the predecessor, if a_j is flush, and NULL, otherwise.
7. If *converge*= false, then go to Step 3.
8. End {EARLY}

Procedure EarlyLabel(i, j, *converge*)

1. $a' = a_i +$ $_{i,j}$.
2. If V_j is a latch output node, $a' = \max(a', B_j)$.
3. If a_j is undefined or $a_j > a'$, then set $a_j = a'$ and *converge* = false. Set t_j to V_i if a_j is flush, and NULL, otherwise.
4. End {EarlyLabel}

Steps 3-7 in Algorithm EARLY form an iteration loop. It consists of (1) a backward pass (Step 4), in which new early paths are generated by adding a backward-directed edge during the traversal in E_b, (2) a forward pass (Step 5), in which new early paths are generated by adding a segment of forward-directed edges during the breadth-first traversal in E_f, (3) a search of loops in the dominant graph T (Step 6), which consists of dominant edges (t_i, V_i). For a node V_i carrying a flush signal, t_i is set to its dominant predecessor. For a node V_i carrying a non-flush signal, t_i is set to NULL. This way, paths in T are all early flush signal paths, as shown in Figure 4(b).

Theorem 2 (Earliest path)

Algorithm *EARLY* will converge in a finite number of iterations $m_E \le b$, to a solution with the earliest arrival times.

Proof: During Step 2, the dominant graph T generates a spanning forest of early flush paths in the acyclic graph (V, E_f). Next, during the first iteration of backward and forward passes, some edge in T, say (V_i, V_j), may be replaced by a new dominant

edge (V_k, V_j). If a loop c forms in T, then the new path generating the new time a'_j is $p_1 c$ where p_1 is the old path generating the old time a_j. Since $a'_j < a_j$, according to Lemma 4 $l(c) < 0$ and we may use Equation (7-9) to quickly update early times around the loop (Step 6). Since predecessor pointers of non-flush signals are set to NULL, loop c is broken into segments of flush paths, and T is again a spanning forest. See Figure 4(b). Let $\{a_i^{(m)}\}$ and $T^{(m)}$ be respectively the early arrival times and the dominant graph at the end of the (m)th iteration. Then, we can prove by mathematical induction that (1)$T^{(m)}$ is a spanning forest, and (2) $a_i^{(m)}$ is the minimum solution among paths in $P^{(m)}$. The arrival times $a_i^{(b)}$ must be the minimum solution, since $P^{(b)} = P$. Algorithm *EARLY* must converge before the iteration count reaches b (.i.e. $m_E \le b$). **Q.E.D.**

The computational complexities for Step 4, 5 and 6 of Algorithm *EARLY* are respectively linear in $|V| + |E_b|$, $|V| + |E_f|$, and $|T|$, which add up to $O(|V| + |E|)$ for one iteration. The overall computational complexity of Algorithm *EARLY* is $O(m_E \times |G|)$ with a worst-case bound $O(b \times |G|)$.

6. Results and discussion

We have implemented Algorithm *SORT*, *LATE*, and *EARLY* in C and have integrated them into CYCLOPSS (CYCLe OPtimization with Slack Sharing), a timing analysis tool. We ran CYCLOPSS through the transformed version (as described in [6]) of all 35 ISCAS'89 benchmark circuits, and an IBM processor chip. The experimental results (on a 33 MIP machine of RISC System/6000) are shown in Table 1. For ISCAS circuits, a complementary two-phase clock and a unit-delay model were used. For the example "chip", we used the realistic clock wave forms and delay rules, provided by an IBM CAD tool, EinsTimer, which included the effects of clock skews, signal slews, and capacitance loading.[1] Column "b" shows the number of feedback nodes. Column "read" shows the data read-in time. Column "Te" shows the extraction time of latch delay graphs. Column "$T1$" shows CPU times of our algorithms based on latch delay graph. Column "$T2$" shows CPU times of the algorithm from [2] we implemented. It can be seen that in most cases the number of iterations m_L and m_E are less than 5, and $T1$ is more than 20% better than $T2$.

circuit name	*gate count*	*latch count*	*b*	*read* sec	*Te* sec	*T*1 sec	*T*2 sec	*T*3 sec	*T*4 sec
*s*13207	17240	1338	407	10.4	26.4	1.2	1.5	27.6	3.2
*s*1423	1462	148	93	0.9	5.3	0.4	0.6	5.7	1.1
*s*27	26	6	3	0.09	0.02	0.0	0.0	0.02	0.00
*s*35932	35586	3456	1161	28.4	37.0	2.2	2.8	39.2	18.9
*s*38584	41410	2904	1707	36.7	74.4	5.0	6.0	79.4	6.0
*s*5378	5916	358	80	3.3	7.2	0.5	0.7	7.7	2.3
*s*9273	11650	456	234	6.5	21.9	0.7	1.0	22.6	1.7
chip	51129	5338	22	307	439	5.0	8.5	444	15.5

Table 1 Experimental results

The latch graph extraction is obtained by running longest and shortest path algorithm through combinational logic N times, each with one of N latches as the source node. The overall complexity is therefore $O(N \times |G|)$, which consumes a significant amount of CPU time. We experimented with the following alternative approach by applying algorithms *LATE* and *EARLY* directly to the full delay graph, which consists of a node set, representing pins of both latches and combinational logic gates, and an edge set, representing pin-to-pin delays, (pins of the combinational logic gates are considered as circular nodes). Column "$T3$" showed the sum $T3 = Te + T1$. Column "$T4$" showed the CPU times for solving full delay graph. Depending on the size of circuits, an additional 2 to 20 times reduction in CPU time is achieved by switching from the latch graph approach to the direct approach. This is especially attractive for incremental timing problem for which the overhead of reading data may be ignored.

Acknowledgement

We would like to acknowledge IBM Rochester's support of this project. We would also like to thank Prof. K.A. Sakallah of Univ. of Michigan, Dr. I. Lin of Silicon Graphics Inc., D. Hathaway of IBM Burlington, K. Belkhale of IBM Fishkill and unknown referees for many helpful discussions and suggestions.

References

1. K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Check Tc and min Tc: Timing verification and optimal clocking of synchronous digital circuits," Proc. ICCAD, pp. 552-555, Nov 1990.
2. T. G. Szymanski, and N. Shenoy, "Verifying clock schedules" Proc. ICCAD, pp. 124-131, Nov. 1992
3. R. B. Hitchcock, "Timing verification and the timing analysis program," Proc. 19 th Design Automation conference, pp. 605-615, 1982.
4. R. S. Tsay and Ichiang Lin, "A system timing verifier for multiple-phase level-sensitive clock design," Research Report RC 17272, IBM Yorktown, 1991.
5. N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli, "Graph Algorithms for clock schedule optimization," ICCAD, pp. 132-136, Nov. 1992.
6. T. G. Szymanski, "Computing optimal clock schedules," Proc. 29 th Design Automation conference, pp. 399-404, 1992.
7. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "Data structure and algorithms," Addison-Wesley, 1983, pp. 216-218.
8. R. E. Tarjan, "Data structures and network algorithms," the Society for Industrial and Applied Mathematics, 1983.
9. E. L. Lawler, "Combinational Optimzation: Networks, and Matroids," Holt, Rinehart and Winston, 1976.
10. J. F. Lee, D. T. Tang and C. K. Wong, "Timing Verification Algorithm for Clock Design with Slack Sharing," IBM Technical Disclosure Bulletin, pp. 249-252, 1993.

1 EinsTimer is a Trademark of IBM Corp.