Efficient Static Timing Analysis Using a Unified Framework for False Paths and Multi-Cycle Paths

Shuo Zhou, Bo Yao, Hongyu Chen, Yi Zhu
and Chung-Kuan Cheng
University of California at San Diego
La Jolla, CA, USA
szhou@cs.ucsd.edu

Mike Hutton
Altera Corp., San Jose, CA, USA
mhutton@altera.com

Abstract - We propose a framework to unify the process of false paths and multi-cycle paths in static timing analysis (STA). We use subgraphs attached with timing constraints to represent false paths and multi-cycle paths. The complexity of the subgraph representation is reduced to improve efficiency. Finally, we present theorems to show that the unified framework produces correct timings. The experimental results demonstrate that the minimization is effective for both artificial and industry test cases.

Keywords: static timing analysis, false subgraphs, multi-cycle subgraphs, time shifting, biclique covering.

1. Introduction

Static timing analysis (STA) is widely used in performance driven optimization programs. The overall procedure of STA performs a forward propagation to derive the *arrival time*s, and a backward propagation to derive the *required arrival time*s and *slack*s at all the points in the circuits [1]. These slacks are useful information for optimization programs.

To derive accurate slacks, STA has to deal with false paths and multi-cycle paths. A *false path* is a path not logically realizable [2]. False path timings must be eliminated from timing analysis. A *multi-cycle path* is a path that signals propagate longer than one clock cycle [8]. The accurate slacks of multi-cycle paths should be computed using multi-cycle arrival times. False paths and multi-cycle paths have to be dealt with efficiently because timing analysis is invoked heavily in the inner loop of optimization programs.

Previous published works in STA focused on false paths [2] [3] [4]. K. P Belkhale et *al*. [2] used *tag*s to distinguish and remove false path timings. Each tag contains a set of false subgraph labels. E. Goldberg et *al*. [3] proposed to reduce the number of timings with *tag*s according to the timing values. Because the timing values may change during the circuit optimization, the reduction is performed together with timing analysis, which induces runtime penalty in the optimization process. D. Blaauw et *al*. [4] removed the specified false subgraphs and produced a new timing graph using node splitting and edge removal. The optimal set of nodes for splitting is identified. We are unaware of reports that can unify the process of false paths and multi-cycle paths efficiently.

In this paper, we propose a framework to unify the process of false paths and multi-cycle paths with *exceptional rule*s, and minimize the number of *rule set*s to improve the efficiency. The contributions of the paper are as follows.

- We represent each *exceptional rule* with a *subgraph* attached with the *setup time* and *hold time*. The rules cover false paths and multi-cycle paths. For the false subgraph, the setup time and hold time are unbounded, i.e., $+\infty$ and $-\infty$, respectively.
- We use *rule set*s to group a set of rules when the timing information can be shared at a particular vertex. We allow *priority* for the rules. When there is a conflict among the rules in a rule set, the rule with the highest priority dominates.
- We devise time shifting to align the hold and setup times of the rules. By doing so, we can merge the distinguished timings, and collect different rule sets into one *rule collection*. For example, when a 2-cycle path and a 3-cycle path converge at a vertex, we can shift the arrival time of the 2-cycle path by one cycle, and merge the timing information.
- We adopt the *biclique covering* approach in [9] to minimize the number of rule collections at every vertex. We present theorems to guarantee that based on the minimized tags timing analysis produces correct slacks.
- The cost of the rule collection minimization is only incurred at the initializing stage, thus not contributing to the CPU time of the optimization program.
- We test our approach on both artificial and industry test cases. For four industry test cases, the number of rule collections is reduced by 84.6% and the runtime of STA decreases by 38.13% on the average. The runtime of minimization is 383 seconds for the test case containing 533,224 nets.

The remainder of this paper is organized as follows. Section 2 defines terminologies. In Sections 3 and 4, we propose the unified framework and the minimization algorithm. The experimental results are presented in Section 5. In the last section, we give the conclusions.

2. Terminology

Timing analysis is performed on a timing graph, which is a directed acyclic *graph* $G = \{V, E\}$, where V is a set of vertices and E is a set of edges. Each *edge* (u, v) is an ordered pair from vertex u to vertex v. The input degree of vertex v, $d^-(v)$, is the number of edges ending at v. The output degree of v, $d^+(v)$, is the number of edges starting at v. The begin set $B = \{v \mid v \in V, d^-(v) = 0\}$ is the set of primary input vertices. The destination set $D = \{v \mid v \in V, d^+(v) = 0\}$ is the set of primary output vertices.

A *path* p in graph G is a sequence of vertices and edges. We can represent path p by only the edges in the path [5]. Each vertex v separates path p into a *head* and a *tail*. Since graph G is directed acyclic, all the paths in G are simple. If a path starts from a vertex in the begin set B, it is a *prefix path* p^-. If a path ends at a vertex in the destination set D, it is a *suffix path* p^+. A *complete path* is both a prefix and a suffix path, which starts from a vertex in the begin set B, and ends at a vertex in the destination set D. The *prefix cone* $P^-(v)$ of a vertex v contains all the prefix paths ending at v. The *suffix cone* $P^+(v)$ of v contains all the suffix paths starting at v.

In Figure 1, the begin set of graph G contains vertices 1 and 2, and the end set contains vertices 8 and 9. Prefix cone of vertex 5 has two prefix paths {(1, 3), (3, 5)} and {(2, 4), (4,

0-7803-9451-8/06/$20.00 ©2006 IEEE.

5)}. Vertex 5 separates complete path {(1, 3), (3, 5), (5, 7), (7, 9)} into head {(1, 3), (3, 5)} and tail {(5, 7), (7, 9)}.

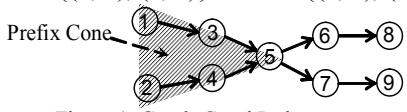


Figure 1. Graph *G* and Paths.

3. Unified Framework Processing False Paths and Multi-cycle Paths

We represent false paths and multi-cycle paths as *exceptional rule*s, and create *rule set*s to unify the process of false paths and multi-cycle paths. We follow the procedure in [2] to compute the rule sets, except that both false and multi-cycle path rules are included in the rule sets. By doing so, we can remove false path arrival times and compute correct slacks for multi-cycle paths according to multi-cycle arrival times.

3.1 General Rule and Exceptional Rules

The *general rule* on graph G is that the complete path p in G satisfies the hold and setup time $[h, s]$, i.e., $h \leq \text{delay}(p) \leq s$.

An *exceptional rule* r describes a false or multi-cycle *subgraph*, $G_r = \{V_r, E_r\}$, a pair of hold and setup time $[h_r, s_r]$, and a priority p_r.

- For the multi-cycle subgraph, $[h_r, s_r]$ is the multi-cycle arrival time, and for the false subgraph, the hold time and setup time are unbounded, i.e., $-\infty$ and $+\infty$, respectively.
- Subgraph G_r describes a set of false paths or multi-cycle paths governed by rule r. The begin set of G_r, denoted as B_r, is a set of vertices which have no input edges in E_r.[1] The destination set of G_r, denoted as D_r, is a set of vertices which have no output edges in E_r. A prefix path p_r^- in G_r starts from a vertex in B_r, and a suffix path p_r^+ in G_r ends at a vertex in D_r. The complete path p_r in G_r is both a prefix and a suffix path in G_r. A path p is a false path or multi-cycle path governed by rule r if the intersection of path p and E_r is a complete path in subgraph G_r. All the paths governed by rule r are constrained by an inequality $h_r \leq \text{delay}(p) \leq s_r$.
- If a path is governed by various rules, the rule with the highest priority p_r supersedes others.

Figure 2 contains two rules, false subgraph rule 0 and multi-cycle subgraph rule 1. Complete path {(1, 3), (3, 5), (5, 6), (6, 8)} is a false path because it contains complete path {(3, 5), (5, 6)} in subgraph G_0. Another complete path {(2, 4), (4, 5), (5, 7), (7, 9)} is a 2-cycle path because it belongs to the subgraph G_1. Complete path {(1, 3), (3, 5), (5, 7), (7, 9)} belongs to both G_0 and G_1. Because the priority of rule 0, i.e., $p_0 = 2$, is higher than the priority of rule 1, i.e., $p_1 = 1$, the complete path is a false path.

When multiple rules are specified, we map the rules on graph G to formulate rule sets at every vertex v and edge (u, v):

- Rule set $F(v) = \{r \mid v \in B_r\}$ contains rules starting from v;
- Rule set $T(v) = \{r \mid v \in D_r\}$ contains rules ending at v;
- Rule set $I(u, v) = \{r \mid (u, v) \in E_r\}$ contains rules covering edge (u, v).

For example rule set $F(3)$ of vertex 3 contains rule 0. Rule set $T(9)$ of vertex 9 contains both rule 0 and rule 1. Rule set $I(5, 7)$ of edge (5, 7) contains rule 0 and rule 1.

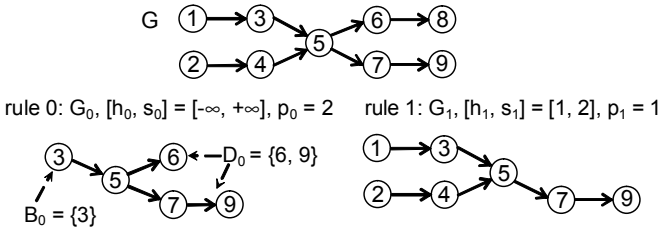


Figure 2. Rules.

3.2 Rule Set Computation

We compute *rule set*s for prefix paths and use prefix *rule set*s to distinguished arrival times.

Definition 3.1: Given a prefix path p^-, the rule set of the prefix path p^- is $R(p^-) = \{r \mid p^- \cap E_r$ is a prefix path in G_r and the tail of $p^-\}$. Given a suffix path p^+, the rule set of the suffix path p^+ is $R(p^+) = \{r \mid p^+ \cap E_r$ is a suffix path in G_r) and the head of $p^+\}$.

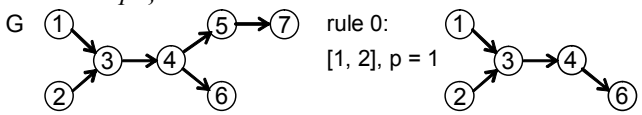


Figure 3. Rule sets.

Conceptually, the prefix or suffix rule set indicates whether the prefix or suffix path belongs to the paths governed by a rule. In Figure 3, the rule set of prefix path $p^-_1 = \{(1, 3), (3, 4)\}$ contains rule 0 because p^-_1 is the prefix path of {(1, 3), (3, 4), (4, 6)}, which is governed by rule 0. Another prefix path $p^-_2 = \{(1, 3), (3, 4), (4, 5)\}$ exits from G_0 at vertex 4, which is not an ending vertex of rule 0. Therefore, rule set $R(p^-_2)$ does not contain rule 0, which means prefix path p^-_2 does not belong to the path of rule 0.

The prefix *rule set*s and the arrival times are computed as follows. [2]

Rule Set Computation (v)

I. If vertex v is a primary input
  If F(v)∩T(v) does not contain false subgraph rule, produce a rule set R = F(v);

II. else
  For each edge (u, v)
    For each rule set R at vertex u
      1. *R′ = (R∩I(u,v)) ∪F(v);*
      2. *If R′∩T(v) does not contain false subgraph rule,*
         arrival_t(v,R′)=max(arrival_t(v,R′),arrival_t(u,R)
         +delay(u,v));

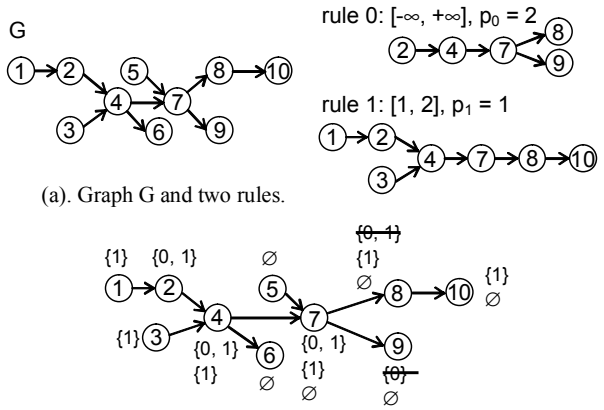
In step *I*, if $F(v) \cap T(v)$ contains false subgraph rules, all the paths through vertex v are false paths. Therefore, we eliminate the arrival times of these false paths by producing no rule set. In step *II.1*, the intersection $R \cap I(u,v)$ means that only rules containing edge (u, v) remain in the rule set. The union with rule set $F(v)$ means that we include rules starting from v in the rule set. In step *II.2*, if the intersection $R' \cap T(v)$ contains false path rules, we eliminate false path arrival times by producing no rule set. Figure 4 illustrates the rule set computation, we compute rule set {0, 1} at vertex 2 by equation $(\{1\} \cap I(1, 2)) \cup F(2)$, where {1} is the rule set at vertex 1, $I(1, 2) = \{1\}$ is the rule set of edge (1, 2), and $F(2) = \{0\}$ is the starting rule set of vertex 2. At vertex 8, rule set {0, 1} is deleted because the intersection of rule set {0, 1} and ending rule set $T(8) = \{0\}$ contains false subgraph rule 0.

We calculate the required arrival time and slack for each rule set R by a backward sweeping. If the arrival time of R is

[1] Since a multi-cycle path is between a pair of flip flops, the vertices in the begin set and end set of a multi-cycle path rule are primary input and output vertices, respectively.

[2] We only compute the maximum arrival times. The minimum arrival times can be computed similarly using min in stead of max operation.

forward propagated to rule set R', the required arrival time of R' is backward propagated to R. For example in Figure 4, the required arrival time for rule set {0, 1} at vertex 4, *req*(4, {0, 1}), is backward propagated from required arrival times *req*(7, {0, 1}) at vertex 7 and *req*(6, ∅) at vertex 6.



(a). Graph G and two rules.

(b). Prefix rule sets at every vertex.

Figure 4. The rule set computation in a forward propagation.

We show that the unified framework based on rule sets produces correct slacks.

Definition 3.2: A prefix rule set R covers a prefix path p^-, if the arrival time labeled by R is the maximum arrival times of a set of prefix paths including p^-.

For example at vertex 7 in Figure 4, prefix paths {(1, 2), (2, 4), (4, 7)}, {(3, 4), (4, 7)} and {(5, 7)} are covered by rule sets {0, 1}, {1}, and ∅, respectively.

Lemma 3.1: At each vertex v, the prefix rule sets produced by forward *Rule Set Computation* cover all the prefix paths in prefix cone $P^-(v)$, and the suffix rule sets produced by backward *Rule Set Computation* cover all the suffix paths in suffix cone $P^+(v)$.

Based on Lemma 3.1, we have Theorem 3.1 as follows.

Theorem 3.1: The slack computation based on prefix rule sets produces correct slacks at every vertex.

4. Rule Collection Minimization

The minimization follows the biclique covering approach in [9], which minimized the number of *rule set*s for false paths. In order to include multi-cycle paths, we devise time shifting to align different hold and setup times, and collects rule sets into *rule collection*s. In this section, we first define the rule collection. Then, we use examples to show basic ideas of the biclique covering approach and time shifting. Finally, we propose our minimization algorithm and show that the produced slacks are correct.

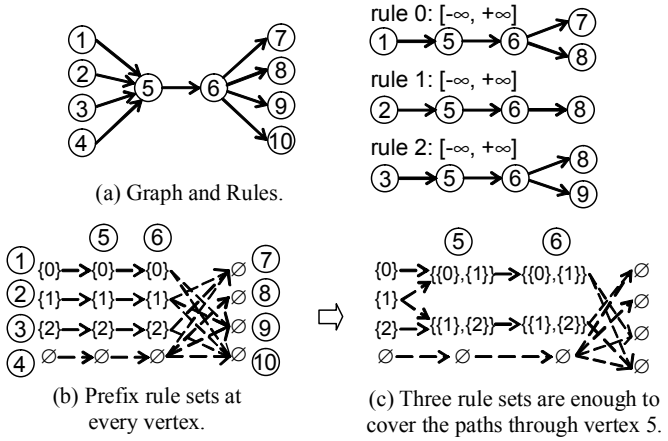
Definition 4.1: A rule collection at vertex v is a set of rule sets of prefix paths which ends at v, i.e. $\Re(v) \subseteq \{R(p^-) | p^- \in P^-(v),\}$, where $P^-(v)$ is the prefix cone at v.

Although the rule sets in a rule collection may contain rules with different hold and setup times, we use *time shifting* to align, which is introduced 4.2.3.

4.1 Motivation

We use two examples to show that the arrival times distinguished by different rule sets can be merged; thus the rule sets can be collected.

Example 1: The paths of three false subgraphs converge at vertex 5, and then diverge at vertex 6. The rule set computation produces four rule sets at vertex 5 and 6. According to the rule set propagation relations in Figure 5 (b), we find that the arrival times $a(5, \{0\})$ and $a(5, \{1\})$ are both propagated through suffix paths {(5, 6), (6, 9)} and {(5, 6), (6, 10)}. Therefore, we can merge the arrival times and collect rule sets {0} and {1} into one rule collection {{0}, {1}}. Similarly, we can merge other arrival times and produce rule collections. As a result, three arrival times distinguished by rule collections are enough to cover all the paths through vertex 5 as shown in Figure 5 (c).



(a) Graph and Rules.

(b) Prefix rule sets at every vertex.

(c) Three rule sets are enough to cover the paths through vertex 5.

Figure 5. Cover complete paths with fewer rule sets.

Example 2: A 2-cycle path {(1, 3), (3, 4)} with hold and setup time [1, 2], and a 3-cycle path {(2, 3), (3, 4)} with hold and setup time [2, 3] converge at vertex 3. Because the hold and setup times are different, the slack of these paths should be computed separately at vertex 4. Therefore, at vertex 3, we have to distinguish the arrival times by rule sets {0} and {1}. However, if we shift the arrival time of prefix path {(1, 3)} forward by 1 cycle, we can merge two arrival times and use hold and setup time [2, 3] to compute the slack for both paths. As a result, we can collect two rule sets into one rule collection.

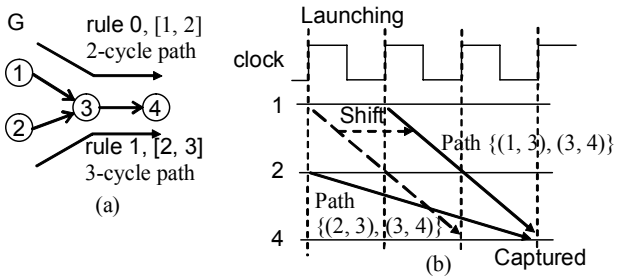


Figure 6. Time shifting: (a) 2-cycle and 3-cycle paths have hold and setup times [1, 2] and [2, 3], respectively; (b) Shift arrival time of path {(1, 3), (3, 4)} to align the hold and setup times.

The examples indicate that: 1) the rule set minimization requires information of the complete paths from both forward and backward sweepings, and 2) we can align the hold and setup times of various multi-cycle paths by time shifting, and merge the timings.

4.2 Rule Collection Minimization Algorithm

The basic idea of the minimization is as follows. 1) Gather prefix and suffix rule sets at each vertex. 2) Obtain and align hold and setup times of complete paths. 3) Collect rule sets into rule collections and cover the complete paths.

4.2.1 Main Flow

We first compute suffix rule sets of all the vertices by backward *Rule Set Computation*. Then, the rule collections are minimized and propagated at every vertex in topological

order. The main flow is as follows.

Main flow
I. Produce suffix rule sets by a backward sweeping;
II. For each vertex v in topological order
1. If vertex v is a primary input
Produce an initial rule collection $\Re(v) = \{F(v)\}$;
2. Rule collection minimization(v);
3. For each edge (v, u) Rule Collection Propagation ($\Re(v)$, v, u);

The rule collection minimization of *II.2* is as follows.

Rule collection minimization (v)
I. For each rule collection $\Re(v)$
For each suffix rule sets $R(p^+)$, where p^+ is the suffix path in the suffix path cone $P^+(v)$
Intersect rule collection $\Re(v)$ with suffix rule sets $R(p^+)$;
II. Construct bipartite graph at v based on the intersections;
III. Shift the hold and setup times of the intersections, and perform biclique covering on bipartite_graph(v);

The rule collection propagation in *II.3* propagates rule collection $\Re(v)$ to vertex *u*, which is introduced in Section 4.2.4.

4.2.2 Intersections of the Prefix and Suffix Rule Sets

At each vertex, the intersections of the prefix and suffix rule sets provide the hold and setup times of the complete paths. If rule *r* belongs to intersection $R(p^-) \cap R(p^+)$, the complete path of prefix path p^- plus suffix path p^+ is governed by rule *r*. If there are various rules in intersection $R(p^-_i) \cap R(p^+)$, the rule with the highest priority is dominant.

Definition 4.2: The intersection between a prefix rule collection $\Re(v)$ and a suffix rule set $R(p^+)$ intersects each prefix rule set in $\Re(v)$ with $R(p^+)$, i.e., $Intersect(\Re(v), R(p^+)) = \{R(p^-_i) \cap R(p^+) \mid R(p^-_i) \in \Re(v)\}$.

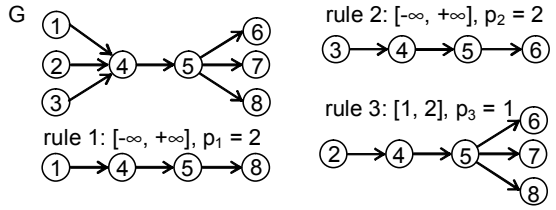


Figure 7. Example for rule collection minimization.

The intersections correspond to the complete paths. For example at vertex 4 in Figure 7, there are three prefix paths and three suffix paths. The first column in Table 4.1 contains the rule collections for the prefix paths, and the first row shows the suffix rule sets. Rule collection {{3}} corresponds to prefix path {(2, 4)}, and suffix rule set {1, 3} corresponds to suffix path {(4, 5), (5, 8)}. *Intersect*({{3}}, {1, 3}) = {{3}} corresponds to complete path {(2, 4), (4, 5), (5, 8)}, which is in rule 3 with hold and setup time [1, 2].

We construct a bipartite graph based on the intersections to cover the complete paths through the vertex. For every intersection, we produce an edge from the rule collection to the suffix rule set and attach the hold and setup time of the intersection on the edge. We eliminate the edge with hold and setup time [-∞, +∞], thus removing false paths.

Table 4.1 Intersections of the prefix rule collections and the suffix rule sets at vertex 4

$\Re(v)$ \ $R(p^+)$	{3}	{1, 3}	{2, 3}
$\{\{1^{[-\infty, +\infty]}\}\}$	$\{\varnothing^{[0, 1]}\}$	$\{\{1^{[-\infty, +\infty]}\}\}$	$\{\varnothing^{[0, 1]}\}$
$\{\{2^{[-\infty, +\infty]}\}\}$	$\{\varnothing^{[0, 1]}\}$	$\{\varnothing^{[0, 1]}\}$	$\{\{2^{[-\infty, +\infty]}\}\}$
$\{\{3^{[1, 2]}\}\}$	$\{\{3^{[1, 2]}\}\}$	$\{\{3^{[1, 2]}\}\}$	$\{\{3^{[1, 2]}\}\}$

Figure 8 illustrates the bipartite graph at vertex 4. The bipartite graph does not contain an edge from rule collection {{1}} to suffix rule set {1, 3} because the hold and setup time of *Intersect* ({{1}}, {1, 3}) = {{1}} is [-∞, +∞].

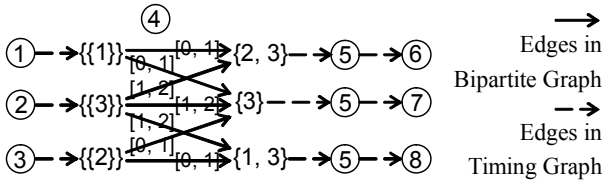


Figure 8. Bipartite graph at vertex 4.

4.2.3 Time Shifting and Biclique Covering

We cover the edges in the bipartite graph by a set of bicliques, i.e. complete bipartite graphs, and collect the prefix rule sets in each biclique into one rule collection. By doing so, we cover the complete paths represented by the edges by rule collections.

All the complete paths covered by one biclique should have aligned hold and setup times. Therefore, we devise time shifting to align different hold and setup times.

Definition 4.3: *Time shifting* on rule collection $\Re(v)$ plus ΔT cycles on the hold and setup time of every rule in $\Re(v)$. The rule collection after time shifting is denoted as $\Re(v)^{+\Delta T}$.

Definition 4.4: *Biclique covering* on a bipartite graph covers the bipartite graph by a set of complete bipartite subgraphs.

We produce aligned bicliques by time shifting and cover the edges of the bipartite graph at vertex *v*. A biclique *b* is *aligned* if for any suffix rule set $R(p^+) \in b$, all the edges to $R(p^+)$ are attached the same hold and setup time. A biclique *b* can be *aligned by time shifting* if there are a set of ΔTs such that after time shifting on each rule collection $\Re(v)_i \in b$ by ΔT_i, and updating the hold and setup times of the edges, biclique *b* becomes aligned.

Biclique Covering (v)
I. Initialize the biclique set as $B = \varnothing$;
II. For every rule collection $\Re(v)$ in minimum degree order
a) For every biclique b in the biclique set B
If b remains a biclique after including a set of edges from $\Re(v)$, and b can be aligned by timing shifting
Add edges from $\Re(v)$ to b, and do time shifting on b;
b) If $\exists$ edges from $\Re(v)$ not covered by bicliques in B
1. For every biclique b in B
If b includes edges from $\Re(v)$, remove the edges from $\Re(v)$ and Shift the hold and setup times back;
2. Produce a new biclique containing all the edges from $\Re(v)$;
3. Add the new biclique to the biclique set B;
III. For each biclique b in the biclique set B
New $\Re(v)' = \cup\Re(v)_i^{\Delta Ti}$, where $\Re(v)_i \in$biclique b, the hold and setup time of $\Re(v)_i$ is shifted by ΔT_i;

Since computing the minimum biclique covering on the general bipartite graph is NP complete [6, 7], we use a minimum degree order approach. Each time, we try to cover the edges from a rule collection $\Re(v)$ by enlarging smaller bicliques. If all the edges from $\Re(v)$ are covered and the enlarged bicliques are aligned under time shifting, we update the enlarged bicliques by adding the edges and shifting the hold and setup times. Otherwise, we produce a new biclique containing the edges from the rule collection $\Re(v)$. Based on each biclique *b*, the new rule collection is the union of all the rule collections in the biclique, i.e., $\Re(v)' = \cup\Re(v)_i$, $\Re(v)_i \in b$.

Figure 9 shows the biclique covering at vertex 4. Rule collection $\Re'_1$ collects rule collections {{1}} and {{3}}. The hold and setup time of {{1}}, [0, 1], is shifted by 1 cycle to align with the hold and setup time of {{3}}, [1, 2]. After minimization, two rule collections cover the complete paths

through vertex 4.

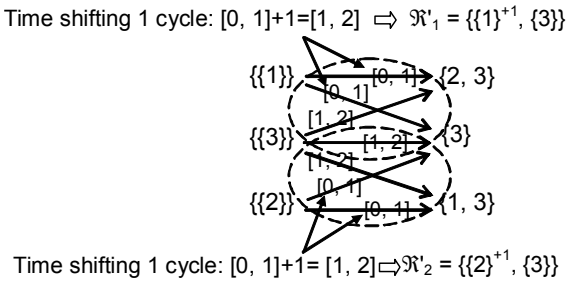


Figure 9. Biclique covering at vertex 4.

4.2.4 Rule Collection Propagation

For edge (v, u), we propagate rule sets in rule collection $\Re(v)$ to vertex u similarly as the *Rule Set Computation* computes the rule sets.

Rule Collection Propagation ($\Re(v)$, v, u)
For each $R \in \Re(v)$
1. *$R' = (R \cap I(v, u)) \cup F(u)$;*
2. *$\Re(u) = \Re(u) \cup \{R'\}$, where $\Re(u)$ is initialized as $\varnothing$;*

4.2.5 Hold and setup Time Conflict

The intersections $R(p^-_i) \cap R(p^+)$ in each *Intersect*($\Re(v)$, $R(p^+)$) should produce the same hold and setup time. Otherwise, there is hold and setup time conflict. For example, Table 4.2 shows the intersections at vertex 5. The rule collections are propagated from vertex 4. The intersection of rule collection $\{\{1\}^{+1}, \{3\}\}$, and suffix rule set $\{1, 3\}$ contains two sets, $\{1\}^{+1}$ and $\{3\}$. The hold and setup time of rule 1, i.e., $[-\infty, +\infty]$, conflicts with the hold and setup time of rule 3, i.e., [1, 2].

Table 4.2 Intersections at vertex 5

$\Re(v)$ \ $R(p^+)$	{3}	{1, 3}	{2, 3}
$\{\{1\}^{+1}, \{3\}\}$	$\{\varnothing^{+1}, \{3\}\}$	~~$\{\{1\}^{+1}, \{3\}\}$~~	$\{\varnothing^{+1}, \{3\}\}$
$\{\{2\}, \{3\}\}$	$\{\varnothing^{+1}, \{3\}\}$	$\{\varnothing^{+1}, \{3\}\}$	~~$\{\{2\}^{+1}, \{3\}\}$~~

The hold and setup time conflict indicates that the complete paths corresponding to the intersection have conflict hold and setup times. For example, *intersect*($\{\{1\}^{+1}, \{3\}\}$, $\{1, 3\}$) at vertex 5 corresponds to two complete paths, $p_1 = \{(1, 4), (4, 5), (5, 8)\}$ which is a false path, and $p_2 = \{(2, 4), (4, 5), (5, 8)\}$ which is multi-cycle path.

Because each rule collection only labeling one arrival time, we cannot use time shifting to align these different hold and setup times. Therefore, in the bipartite graph we do not produce the edge between the rule collection and the suffix rule set if there is a conflict. Theorem and lemmas in Section 4.4 guarantee that all the paths are still covered.

Bipartite graph Construction (v)
I. Collect rule collections propagated from the previous vertices;
II. For each rule collection $\Re(v)$ and suffix rule set R(p+)
1. *Intersect($\Re(v)$, $R(p^+)$) = $\{R(p^-)_i^{\Delta Ti} \cap R(p^+) \mid R(p^-)_i \in \Re(v)\}$;*
2. *For each $R(p^-)_i^{\Delta Ti} \cap R(p^+) \in$ Intersect($\Re(v)$, $R(p^+)$)*
   The hold and setup time is $[h_r+\Delta T_i, s_r+\Delta T_i]$, where r is the rule in $R(p^-)_i^{\Delta Ti} \cap R(p^+)$ with the highest priority;
3. *If there is no conflict among all $[h_r+\Delta T_i, s_r+\Delta T_i]$s, and $\forall [h_r+\Delta T_i, s_r+\Delta T_i] \neq [-\infty, +\infty]$*
   Add an edge attached with $[h_r+\Delta T_i, s_r+\Delta T_i]$ from $\Re(v)$ to $R(p^+)$;

Figure 10 illustrates the bipartite graph at vertex 5, which does not contain edges from $\{\{1\}^{+1}, \{3\}\}$ to $\{1, 3\}$ and from $\{\{2\}^{+1}, \{3\}\}$ to $\{2, 3\}$ due to hold and setup time conflicts.

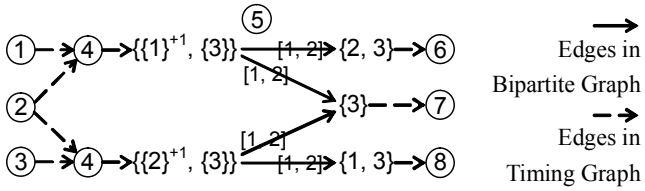


Figure 10. Bipartite graph at vertex 5.

Figure 11 summarizes the rule collections produced by our approach. At vertices 4 and 5, the number of tags is reduced from 3 to 2.

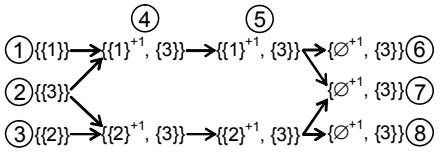


Figure 11. Rule collections at every vertex: $\varnothing^{+1}$s at vertex 6 and 8 are propagated from $\{1\}^{+1}$ and $\{2\}^{+1}$ at vertex 5, respectively; $\varnothing^{+1}$ at vertex 7 is propagated from $\{1\}^{+1}$ and $\{2\}^{+1}$ at vertex 5.

4.3 Timing Analysis with Rule Collections

We compute the arrival time, required arrival and slack for each rule collection by forward and backward sweepings similar as the timing analysis process based on rule sets. The only difference is that for rule collections with time shifting, $\Re(v)^{+\Delta T}$, we forward and backward shift the arrival times and required arrival times by ΔT. In Figure 11, for the rule collection $\Re'_1 = \{\{1\}^{+1}, \{3\}\}$ at vertex 4, the arrival time of prefix path $\{(1, 4)\}$ is shifted by 1 cycle and merged with the arrival time of $\{(2, 4)\}$. When the required arrival time labeled by $\Re'_1$ is backward propagated to vertex 1, we shift the required arrival time back by 1 cycle.

4.4 Correctness

This section presents lemmas and theorems to guarantee the correctness. We show that all the complete paths are covered by rule collections. Thus, timing analysis based on rule collections produces correct slacks of all the paths. We omit formal proofs due to space constraints.

Definition 4.5: A rule collection $\Re(v)$ covers a complete path p through vertex v, where p is the concatenation of prefix path $p^- \in P^-(v)$ and suffix path $p^+ \in P^+(v)$, if 1) p^- is covered by prefix rule set $R(p^-) \in \Re(v)$, 2) p^+ is covered by suffix rule set $R(p^+)$, and 3) the bipartite graph at vertex v contains an edge from $\Re(v)$ to $R(p^+)$.

Lemma 4.1: If rule collection $\Re(u)$ covers a complete path p at vertex u and edge $(u, v) \in p$, after *Rule Collection Propagation*, the produced rule collection $\Re(v)$ covers path p at vertex v.

Theorem 4.1: All the complete paths through a vertex v are covered by the rule collections at v.

Lemma 4.1 and Theorem 4.1 show that rule collections cover all the complete paths. The next lemma and theorem show that timing analysis produces correct slacks.

Lemma 4.2: If rule collection $\Re(v)$ covers path p and path p is governed by rule r, *slack* (v, $\Re(v)$) covers the correct slack of path p, which is computed based on rule r.

Theorem 4.2: Timing analysis with rule collections produces correct slack at each vertex v, which covers the slacks of all the paths through vertex v.

5. Experimental Results

We test our algorithm on both artificial and industry test cases. The algorithm is implemented in C and tested on a

Pentium 4 Linux machine.

We first follow the experiments in [2] to randomly create false and multi-cycle subgraphs on a 100×100 mesh. The average number of edges in subgraphs is 6000. Each test case contains from 9 to 104 rules including 30 percent false subgraph rules. The hold and setup times of multi-cycle paths are in the range from 2-cycle to 4-cycle. We compare the number of rule collections with the number of prefix rule sets in Table 5.1. The number of prefix rule sets equals to the number of tags produced by the approach in [2] if there are only false paths. The average reduction ratio for five test cases is 31.22%. The CPU time for tag minimization increases when the number of rules in the case increases. The largest case including 104 rules consumes 87 seconds.

Table 5.1. *Tag* minimization on a 100 ×100 mesh

# rules	# prefix rule sets	# rule collections	Reduction ratio	Runtime (sec)
9	9129	8281	9.29%	2
34	77102	49321	36.03%	19
69	137581	89987	34.59%	44
88	176384	97124	44.94%	61
104	209718	145484	30.63%	87
average			31.10%	

--Reduction ratio= (#prefix rule sets - #rule collections)/ (#prefix rule sets)

We also test our algorithm on four industry test cases and show the experimental results in Table 5.2. The largest circuit, i.e., atmlcore, contains 533,224 nets, 2 false path rules and 2262 multi-cycle path rules. We use *Rule Set Computation* to produce prefix rule sets, and minimize rule sets into rule collections. The average reduction ratio of four test cases is 84.60%. For the largest circuit, i.e. atmlcore, the runtime of minimization is only 383 seconds, including the CPU time for loading the test cases, mapping the rules on the graph, and minimizing the rule collections.

Table 5.2 also shows runtimes of STA (STA). If STA uses prefix rule sets to deal with false paths and multi-cycle paths, for the largest circuit, i.e., atmlcore, the runtime is 40.33 seconds. If rule collections are used in STA, the runtime is reduced to 19.5 seconds. Though the reduction is only 20.83 seconds for performing STA once, the reduction ratio is 51.65%. If timing analysis is repeatedly called during performance driven optimization, for example 100 times, the reduction on STA runtime would be 2083 seconds, which is larger than the minimization runtime cost 40.33 seconds.

6. Conclusions

We propose a framework to unify the process of false paths and multi-cycle paths in STA. Furthermore, we improve the efficiency by minimizing the number of distinguished timings created for false paths and multi-cycle paths. Finally, we present theorems to guarantee that our approach produces correct timing information with false paths and multi-cycle paths considered. The experimental results demonstrate that our minimization is effective.

Acknowledgements

This work was supported in part by the California MICRO program and a grant from Altera Corp.

References

[1] R. B. Hitchcock, "Timing Verification and Timing Analysis Program", DAC 1982, pp. 594-604.
[2] K. P. Belkhale, A. J. Suess, "Timing Analysis with known False Sub-Graphs", ICCAD 1995, pp. 736-740.
[3] E. Goldberg, A. Saldanha, "Timing Analysis with Implicitly Specified False Path", Int. Workshop on Timing Issues in the Specification and Synthesis of Digital Designs, T99, 1999.
[4] D. Blaauw, R. Panda, A. Das, "Removing user-specified false paths from timing graphs", DAC 2000, pp. 270-273.
[5] TC Hu, Combinatorial Algorithms, Dover Publication, Second Edition, 2002.
[6] J. Orlin, "Containment in graph theory: Covering graphs with cliques", Nederl. Akad. Wetensch. Indag. Math., 39:211-218, 1977.
[7] H. Muller, "Alternate Cycle-Free Matchings", Order, (7):11-21, 1990.
[8] M. Hutton, D. Karchmer, B. Archell, J. Govig, "Efficient Static Timing Analysis and Applications Using Edge Masks", FPGA 2005, pp. 174-183.
[9] S. Zhou, B. Yao, H. Chen, Y. Zhu, CK Cheng, M. Hutton, et *al.*, "Improving the efficiency of Static Timing Analysis with False Paths", ICCAD 2005, pp. 527-531.

Table 5.2. *Tag* minimization on industry test cases

Cases	# nets	#rules		# prefix rule sets	# rule collections	reduction ratio	Minimization Runtime (sec)	STA runtime		
		False path	Multi-cycle path					Use prefix rule sets (sec)	Use rule collections (sec)	Runtime reduction
tdl	27,555	1	27	158	67	57.59%	1	1.2	1.2	0
cq_mod	38,535	2517	3181	217,456	14,972	93.11%	22	4.2	2	52.38%
pm25c	325,582	7	2574	1,781,400	101,238	94.32%	106	55.33	28.5	48.49%
atmlcore	533,224	2	2262	2,411,892	159,451	93.39%	383	40.33	19.5	51.65%
average						84.60%				38.13%

--Reduction ratio = (#prefix rule sets - #rule collections)/ (#prefix rule sets)

--Reduction of STA runtime = (STA Runtime using prefix rule sets – STA Runtime using rule collections)/ STA Runtime using prefix rule sets