Circuit Enhancement by Eliminating Long False Paths

Hsi-chuan Chen* David H.C. Du* Siu Wing Cheng†
Department of Computer Science
University of Minnesota

Abstract

One way to improve circuit performance is to introduce redundancies into the circuit. However, this results in higher number of long false paths in the circuit and the testability of the circuit is reduced. Keutzer et al. [4] developed a long false path elimination algorithm to remove the redundancies without sacrificing its performance. We propose a more efficient algorithm and two very effective heuristics to remove long false paths. Our experimental results show that the proposed algorithm and heuristics are feasible for large designs and the increase of circuit area is significantly reduced.

1 Introduction

Due to the fast increasing demands for high-speed computation and highly reliable designs, the quality of a circuit is now evaluated by three criteria: area, performance, and testability. Therefore, in addition to minimizing circuit area, current synthesis systems also attempt to speed up circuit performance and to enhance circuit testability. However, these three objectives are usually conflicting to one another. The performance of the circuit is defined to be the delay of the longest sensitizable paths in the circuit [2]. Therefore, one way to enhance the performance of a circuit is to add stuck-at redundancies to make the current longest paths false [4, 6]. However, the existence of stuck-at redundancies is detrimental to circuit testability, because it makes test pattern generation difficult. Furthermore, a side-effect of this performance optimization approach is that many long false paths are created. As indicated in [7], the existence of long false paths in a circuit usually results in low delay-fault detection coverage.

Keutzer, Malik, and Saldanha [4] are the first to propose an algorithm (denoted as KMS algorithm) which not only enhances circuit testability but also retains circuit performance. KMS algorithm is used to eliminate all long false paths in a circuit [4]. When there is no long false path, the delay-fault detection coverage of the circuit increases. Also, most stuck-at redundancies are removed because of the elimination of long false paths. So, the circuit testability is improved.

KMS algorithm is based on the following observation on a long false path in a circuit: if each gate of the false path has single fanout, then the first lead of the false path is both stuck-at-0 and stuck-at-1 redundant. Therefore, it can be replaced by either logic 0 or logic 1. By replacing the first lead of the path by the controlling value to its succeeding gate, the value at the output of the succeeding gate is fixed. Therefore, the succeeding gate as well as all its input leads can be removed. Since the value at the output of the succeeding gate is fixed, all leads originating from the gate need to be replaced by the fixed value. These induced replacements can further remove more gates and leads in the circuit. As a result, the long false path is considered to be eliminated if it becomes not longer than the longest sensitizable paths. Figure 1 is an example of the logic value replacement. A long false path may contain gates with multiple fanouts. For such a long false path, the KMS algorithm first duplicates gates and leads (path isolation) so that each gate of the path in the modified circuit has single fanout before shortening it. The KMS algorithm repeats the long false path elimination process until the longest path in the final circuit is sensitizable.

The KMS algorithm has been shown to successfully eliminate all long false paths in small circuits. However, it may not be able to handle large circuits due to excessive gate/lead duplications in a path isolation and limited gate/lead removals in a logic value replacement. The reasons are as follows. First, each long false path in a large circuit may contain many gates. Very likely, most gates have multiple fanouts. Therefore, in order to isolate a complete long false path, the KMS algorithm may need to duplicate quite a lot of gates and leads, and consequently...
the area of the circuit increases dramatically.

In this paper, we propose to identify and perform logic replacement for a portion, called segment, of a long path instead of the path itself. This results in less gate and lead duplication and hence reduction in the increase of circuit area. We give sufficient conditions for the choice of segments that can retain the performance of the circuit. If we relax the restriction on the choice of segments, then we obtain two very effective heuristics for removing long false paths. Our experimental results show that these two heuristics are much more superior in reducing the increase in circuit area. Moreover, the performances of all the circuits tested are retained.

2 Definitions

Definition 1 [1] A path is said to be statically sensitizable if there exists at least one primary input vector that stabilizes all side-inputs of P at their non-controlling values. A path is considered to be a statically false path if it is not statically sensitizable.

According to [1, 2], a long false path must be statically false, but the critical path may or may not be statically sensitizable.

Definition 2 (SFS) Partial path Q is called a Statically False Segment if there exists no primary input vector that stabilizes all side-inputs of Q at their non-controlling values.

It is important to note that any partial path including an SFS segment is also an SFS segment. A statically false path itself is the largest SFS segment of the path. Thus, the KMS algorithm only deals with SFS segments that are paths by themselves.

Definition 3 (SZES (SOES)) Partial path Q is called a Statically Zero-Enforced (One-Enforced) Segment if each primary input vector which stabilizes all side-inputs of Q at their non-controlling values also stabilizes the first lead of Q at logic 0 (1).

We will use segment as a collective name for the SFS segment, the SZES segment, and the SOES segment.

3 Selecting A Segment

The first lead of an SFS segment is replaceable by logic 0 or logic 1 without altering the functionality of the circuit, if each gate of the segment has single fanout. Similarly, the first lead of an SZES (SOES) segment is replaceable by logic 0 (1) if each gate of the segment has single fanout. Figure 2 is an example of performing logic replacement for an SFS segment.

When any gate in a segment has multiple fanouts, we need to isolate the segment. To isolate a segment, all the gates (and so their associated input leads) between the last gate with multiple fanouts in the segment and the first gate of the segment need to be duplicated. We then replace the first lead of the segment by an appropriate logic value.

However, performing logic value replacement for an arbitrary segment may not retain the circuit performance.

Suppose that \( r \) is the circuit delay of the original circuit, then we select the time-constrained segments to guarantee that the delay of the modified circuit will not be greater than \( r \). We first define time-constrained SFS segments as follows.

Definition 4 A Time-Constrained SFS Segment \( S = (I, G_1, \ldots, f_{k-1}, G_k) \) is a SFS segment that satisfies the conditions \( C1 \) or \( C2 \) for all \( s + 2 \leq k \leq t \):

\( C1: \) Min.in\((f_s) + \sum_{i=s+1}^{t} d(G_i) \geq \text{Max.in}(F1(G_k) - \{f_{k-1}\}) \)

\( C2: \) Max.in\((F1(G_k)) - \{f_{k-1}\}) + d(G_k) + \text{Max.out}(G_k) \leq r \)

where Min.in\((f_s)\) is the minimum delay from the primary inputs to \( f_s \), and \( d(G_i) \) is the gate delay of \( G_i \), and Max.in\((F1(G_k)) - \{f_{k-1}\})\) is the maximum delay from the primary inputs to the inputs of \( G_k \) other than \( f_{k-1} \), and Max.out\((G_k)\) is the maximum delay from \( G_k \) to the primary outputs.

The time-constrained SZES and SOES can be defined similarly. However, if logic 0 (resp. logic 1) is non-controlling to the first gate of an SZES (resp. SOES), then \( k \) should vary from \( s + 1 \) to \( t \) instead of from \( s + 2 \) to \( t \) in the above.

Using time-constrained segments, the following result can be established. We omit the proof and the details can be found in [3].

Theorem 1 The circuit delay will not increase after performing isolation and logic value replacement for a time-constrained segment.

The additional time constraint has the effect of pulling the beginning of a segment towards the primary inputs. Thus, a better lower bound for Min.in(\( f_s \)) and better upper bounds for Max.in(\( f_s \)) and Max.out(\( f_s \)) should produce time-constrained segments of shorter lengths.

Remark: If the delay \( r \) of the original circuit is not known, then the alternative condition \( C2 \) can be dropped without affecting the correctness of the algorithm. Then the algorithm just runs until no time-constrained segment can be found along the longest paths of a circuit. This restriction has the advantage of avoiding timing verification as a preliminary step (which is often quite time-consuming) as well as allowing for possible improvement in circuit performance. But this restriction also results in a stricter time constraint which implies that we may work with longer segments.
4 Long False Paths Elimination

Before we describe our algorithm, we first elaborate on how to determine whether a partial path is a segment. Let \( Q = (f_0, G_1, f_1, \ldots, f_{n-1}, G_t) \) be a partial path. The initial condition set for \( Q \) is composed of the conditions requiring all side-inputs of \( Q \) at their non-controlling values. Using the implication and justification procedures in D-algorithm [4], these conditions are propagated both forward and backward and hence new conditions are generated and are added into the condition set. This condition propagation process continues until either a conflict is detected in the final condition set or no more new conditions can be added into the final condition set. For the case when there is no conflict in the final condition set: if \( f_1 \) is assigned a logic 0 (resp. logic 1) in the final condition set, \( Q \) is determined to be an SFS segment. For the case when there is a conflict in the final condition set: if \( f_1 \) is assigned a logic 0 (resp. logic 1) in the final condition set, \( Q \) is determined to be a prime segment.

For comparison (with the KMS algorithm), we define our long false paths elimination algorithm to be repeated iterations (runs) of path-tracing, time-constrained segment identification, segment isolation, and logic value replacement. The algorithm continues until the longest path in the final circuit contains no time-constrained segment. A longest path is traced by Best First Search. Then we identify a time-constrained segment along this path as follows.

Let \( P = (I, f_0, G_1, f_1, \ldots, f_{m-1}, O) \) be the path traced. We divide the identification process into two phases. In Phase 1, we first locate gate \( G_1 \) in \( P \) closest to \( I \) such that partial path \( Q = (f_0, G_1, \ldots, f_{n-1}, G_t) \) is a segment. Then, we locate gate \( G_{n+1} \) in \( Q \) closest to \( G_t \) in Phase 2 such that partial path \( S = (f_s, G_{n+1}, f_{n+1}, \ldots, f_{n-1}, G_t) \) is a time-constrained segment. In Phase 1, partial path \( Q \) is initialized to be \((f_0, G_1)\). If \( Q \) is determined to be a segment, Phase 1 stops. Otherwise, we expand \( Q \) forward to include the next pair of lead and gate. Then, the expanded \( Q \) is under determination again. This expansion-and-determination process continues until \( Q \) is determined to be a segment. Let the final \( Q \) of Phase 1 be \((f_0, G_1, f_1, G_2, \ldots, f_{n-1}, G_t)\).

Phase 2 is similar to Phase 1. The partial path \( S \) is initialized to be \((f_{n-1}, G_t)\) and we expand \( S \) backward until \( S \) is determined to be a segment. Then we verify the conditions \( C1 \) or \( C2 \). If the conditions \( C1 \) or \( C2 \) are satisfied, then \( S \) is a time-constrained segment and we can stop. Otherwise, we expand \( S \) backward by one lead and one gate and perform the verification again. In our implementation, we use the topological shortest delay (resp. longest delay) to approximate Min.in(-) (resp. Max.in(-) and Max.out(-)).

After a time-constrained segment is identified, we duplicate gates and leads if necessary to isolate the segment. Then, the first lead of the segment is replaced by an appropriate logic value according to the type of the segment. The effect of the replacement will be fully propagated so as to remove as many gates and leads as possible.

5 Two Very Effective Heuristics

If the segments could be chosen freely without worrying about the additional time constraint, then much less increase in circuit area is expected. The reasons are as follows.

First, much shorter segments may possibly be obtained if the additional time constraint is disregarded. Much fewer gates and leads would be duplicated during isolation. Second, gate/lead removals in a logic value replacement depends on the extent of its propagation. The extent of the forward propagation of a logic value replacement is often limited, because AND gates and OR gates of a path usually interleave in reality. Replacing a lead of a path by the controlling value usually fixes the next lead at the non-controlling value, and consequently the forward propagation stops. Most gate/lead removals corresponding to a logic value replacement are due to the backward propagation in the input cone. When a lead is removed, if it is the only output lead of its originating gate, the gate and all its input leads are useless and can be removed too. Hence, if a segment whose first lead is far from the primary inputs could be chosen, then many gates and leads may possibly be removed in a logic value replacement. Furthermore, all the paths passing through that segment are eliminated in one run.

Unfortunately, such an approach cannot, in theory, retain the performance of a circuit because a long false partial path that merges with the segment at some point may become sensitizable. Thus, theoretically speaking, a long false path may become sensitizable and the circuit delay may then increase (a more detailed explanation is offered in [3]). However, our experimental results for the two heuristics actually show the opposite. Indeed, the performance of all the circuits in our experiments is retained. It is probably due to the fact that in circuits, which are speeded up by introducing redundancies to bypass long paths, short bypasses are likely to meet a long path close to the primary outputs. Therefore, the long partial path that becomes sensitizable will be blocked by some (earlier) controlling inputs later. Hence, the circuit performance will most probably not be affected.

The two heuristics differ from the algorithm in Section 4 only in the identification of segments. Since it is important to identify segments that are as short as possible, we give the definition of a prime segment as follows.

Definition 5 A segment is a prime segment if no proper partial path of the segment is a segment.

It is very time consuming to identify the best prime segment. Instead, we propose two prime segment identification procedures: one is to identify the prime segment closest to the primary input of a path (front segment identification procedure, denoted as FSIP), and the other is to identify the prime segment closest to the primary output of the path (rear segment identification procedure, denoted as RSIP).

We first trace the longest path \( P \). In FSIP, we locate the shortest segment on \( P \) that begins at the primary inputs. Let it be \((f_0, G_1, \ldots, f_{n-1}, G_t)\). Then we initialize a partial path \( S \) to be \((f_{n-1}, G_t)\) and we expand \( S \) backward until \( S \) is determined to be a prime segment. The determination process is implemented as before using the D-algorithm. In RSIP, we first locate the shortest segment on \( P \) that ends at the primary outputs. Let it be \((f_0, G_{n+1}, \ldots, G_m, f_m, O)\). Then starting with \( S \) equal to \((f_0, G_{n+1})\), we expand \( S \) forward until \( S \) is determined to be a prime segment.

251
We denote the heuristics that uses FSIP and RSIP to identify prime segments by FLPEP and RLPEP, respectively.

6 Experimental Results

We have implemented our long false path elimination algorithm (denoted by LPEA) and FLPEP and RLPEP in C on a SUN Sparstation. Due to the concern on computational efficiency, we use a simplified D-algorithm as in [1], which implements logic implication only, in identifying segments. We have experimented all of them on some ISCAS85 circuits. Table 1 shows the characteristics of each ISCAS85 circuit tested.

Table 1. Characteristics of ISCAS85 benchmark circuits

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>No. of Gates</th>
<th>Topological Delay (ns.)</th>
<th>Critical Delay (ns.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1908</td>
<td>670</td>
<td>87.90</td>
<td>87.90</td>
</tr>
<tr>
<td>C2670</td>
<td>1426</td>
<td>88.90</td>
<td>88.10</td>
</tr>
<tr>
<td>C3540</td>
<td>1710</td>
<td>98.70</td>
<td>98.70</td>
</tr>
<tr>
<td>C3552</td>
<td>2488</td>
<td>99.30</td>
<td>97.50</td>
</tr>
<tr>
<td>C5315</td>
<td>2448</td>
<td>319.00</td>
<td>316.30</td>
</tr>
<tr>
<td>C5316</td>
<td>2710</td>
<td>88.30</td>
<td>83.80</td>
</tr>
</tbody>
</table>

Table 2 shows our experimental results. For comparison, we also implemented the KMS algorithm. In Table 2, \( \tau \) represents the circuit delay after eliminating all long false paths. \( \Delta(G) \) denotes the net gate increase and \#(R) the number of runs needed to eliminate all the long paths. Notation "-" inside Table 2 denotes the failure of completion due to \( \Delta(G) > 20,000 \). As expected, LPEA outperforms the KMS algorithm. LPEA results in much smaller increase in circuit area than the KMS algorithm does. For circuits that both LPEA and the KMS algorithms successfully terminate, the average area increase ratio of LPEA to the KMS algorithm is less than 17%. The two heuristics FLPEP and RLPEP show much superior ability in reducing the increase in circuit area. Also, the performances of all the circuits tested are retained. Both FLPEP and RLPEP successfully terminates even for circuits like C1908 and C3540, and RLPEP even terminates for C6288. The average area increase ratio of FLPEP and RLPEP to the KMS algorithm is less than 3%.

According to our experimental results, RLPEP outperforms FLPEP in most circuits (i.e., C2670, C3540, and C6288). Possible reasons are as follows. Since a RSIP prime segment is closer to the primary output, performing logic replacement for a RSIP prime segment will give us a larger reduction in path length. Therefore, RLPEP tends to use fewer runs than FLPEP to eliminate all long false paths. Furthermore, a RSIP prime segment is closer to the primary outputs and hence the input cone of the succeeding gate of its first lead may be large. As a consequence, RLPEP very likely removes more gates and leads than FLPEP in a logic value replacement.

Circuit C6288 is the most surprising example. While FLPEP fails to complete, RLPEP efficiently eliminates all long false paths with a negligible area increase as well as an improvement in performance. This can be explained by the fact that performing logic replacement for segments may also reduce the delay of the longest sensitizable paths. The best known estimation on its critical delay before elimination is 316.30 nano seconds [5].

However, as shown in our experiments there are some circuits (e.g., C3315 and C5762) on which FLPEP outperforms RLPEP. That might be because the number of long false paths sharing an FSIP prime segment is larger than that sharing the corresponding RSIP prime segment. In this case, FLPEP can eliminate more long false paths in one run than RLPEP does.

Acknowledgement

We would like to thank an anonymous referee for the comments that lead to the discovery of an error in an earlier version of this paper.

References