Timing-Safe False Path Removal for Combinational Modules*

Yuji Kukimoto
Monterey Design Systems, Inc.
Sunnyvale, CA 94089
kukimoto@montereydesign.com

Robert K. Brayton
University of California
Berkeley, CA 94720
brayton@eecs.berkeley.edu

Abstract

A combinational module is a combinational circuit that can be used under any arrival time condition at the primary inputs. An intellectual property (IP) module, if combinational, is one such example. The false-path-aware delay characterization of a combinational module without disclosing its internal structural detail is crucial for accurate timing analysis of IP-based designs.

This paper addresses three related issues on delay characterization of combinational modules. We first introduce a new notion called *timing-safe replaceability* as a way of comparing the timing characteristics of two combinational modules formally. This notion allows us to determine whether a new module is a safe replacement of an original module under any surrounding environment with respect to timing. Second, we consider false path detection of combinational modules. Although false path detection is essential in accurate delay modeling, we argue that the conventional definition of false paths such as floating mode analysis is not appropriate for defining the falsity of a path for a combinational module since the falsity is relative to an arrival time condition. A new definition of false paths, termed *strongly false paths*, is introduced to resolve this issue. Strongly false paths are those paths that are guaranteed to be false under any arrival time condition, and thus uniquely defined independent of arrival time conditions. Finally, we propose a new algorithm that removes strongly false paths from a combinational module by a circuit transformation. We prove that the resulting circuit is a timing-safe replacement of the original.

1 Introduction

A delay abstraction of a combinational module is a compact representation of the delay information of the module, which carries effective pin-to-pin delay for each primary-input/primary-output pair. Constructing an accurate delay abstraction of a combinational module is crucial in capturing its delay characteristics precisely. Whether the delay abstraction maintains enough accuracy or not has a direct impact on the accuracy of timing analysis using the delay abstraction. An important requirement is that the delay abstraction needs to be valid under any surrounding environment since we do not know a priori how this module is to be used.

In the past, delay characterization of combinational modules has been done mainly by performing topological analysis, i.e. given a combinational module, the delay between each input/output pair of the module is characterized by the longest topological path between the two terminals. Although this analysis is computationally efficient, the resulting delay abstraction can be too conservative since the analysis completely ignores false paths inside the module. Recently we have proposed a novel delay characterization technique for combinational modules, in which false paths inside the module are correctly identified [6, 7]. This provides a delay abstraction more accurate than the abstraction computed by topological delay, yet still valid under any operating condition of the module[1].

In this paper we address three problems of delay characterization for combinational modules. We first develop a new framework for comparing the timing characteristics of two combinational modules. A new notion called *timing-safe replaceability* is introduced to determine when a module can be safely replaced with another without slowing down the original timing behavior under any surrounding environment. We also define when a path is categorized as false for a combinational module. We show by an example that the conventional definition of false paths such as floating mode analysis [3] is relative to given arrival times at primary inputs, and that it is possible for the same path to be false under a certain arrival time condition, but not under another. In our new definition of false paths, a path is said to be strongly false if and only if it is false under *any* arrival time condition at primary inputs. Finally, supported by the two preceding results, we propose a circuit transformation technique to remove strongly false paths without deteriorating the timing characteristic of the original module. This is a generalization of the KMS algorithm [4] in the sense that strongly false paths are removed without slowing down the module under *any* arrival time condition at the primary inputs. We prove that a new module constructed by the generalized KMS algorithm is a timing-safe replacement of the original. It is also shown that the original KMS algorithm does not satisfy this notion of timing-safe replaceability and thus potentially has a negative impact on circuit delay under some environment.

The paper is organized as follows. Section 2 summarizes the previous result on delay characterization of combinational modules using functional required time analysis [6, 7]. Section 3 discusses timing-safe replaceability of combinational modules. A new definition of false paths, called strongly false paths, is introduced in Section 4. We then discuss how strongly false paths can be removed safely under the notion of timing-safe replaceability in Section 5. Experimental results are presented in Section 6. Section 7 concludes the paper.

2 Preliminaries

Let $\mathcal{M}$ be a single-output combinational module under analysis. Let $X = (x_1, \ldots, x_n)$ and z be the primary inputs and the primary output of $\mathcal{M}$ respectively. Consider delay from the primary inputs X to the primary output z. The standard way to define the delay of a module is by computing the earliest stable time of each output given arrival times at all the primary inputs. The difference between the output stable time and the arrival time of each input gives the delay from the input to the output. This approach, however, is not applicable to our setting since we do not know when the primary inputs arrive. Our goal is to capture the timing characteristics of a given module valid and accurate under *any* surrounding environment. To achieve this the delay of a module is defined in a different way. We first set a required time, say $t = 0$, to the output and analyze the given circuit to see when the primary inputs are required so that the output becomes stable by the required time. The delay from an input to the output is then defined as the difference between the required time at the output ($t = 0$) and that of the input. This is

*This work was supported by SRC-98-DC-324. The first author was with University of California, Berkeley.

[1] The use of arrival-time independent path sensitization conditions such as the Brand-Iyengar condition [2] can compute a false-path-aware delay abstraction valid under any arrival time condition. However, delay overestimation can occur.

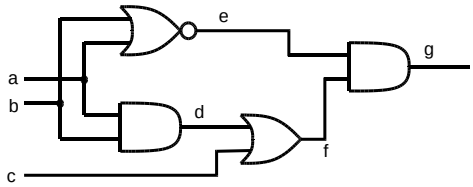

Figure 1: A Combinational Module $\mathcal{M}$

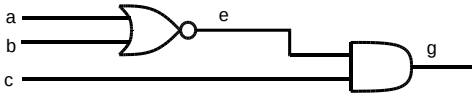

Figure 2: A Timing Safe-Replaceable Module $\mathcal{M}_0$

exactly the same problem as functional required time analysis in [6].

Functional required time analysis gives $T_z \subseteq B^n \times R^n$, where a set of timing tuples is given for each input vector. (R denotes the set of real values.) Given an input vector $\mathbf{x} \in B^n$, each n-tuple $t = (t_1, \ldots, t_n)$ such that $(\mathbf{x}, t) \in T_z$ represents valid required times at the inputs. The interpretation of a tuple t is that the output is guaranteed to be stable at $t = 0$ if the primary input vector $\mathbf{x} = (x_1, \ldots, x_n)$ arrives at or before $t = (t_1, \ldots, t_n)$ respectively. T_z may contain more than one timing tuple for a given input vector, in which case each of the timing tuples captures a different permissible signal arriving behavior at the primary inputs. The *delay abstraction* D_z of $\mathcal{M}$ is then defined as $D_z = \{(\mathbf{x}, (-t_1, \ldots, -t_n)) \mid (\mathbf{x}, (t_1, \ldots, t_n)) \in T_z\}$. To compute the delay abstraction of a multiple-output module, one can apply the same analysis above to the transitive fanin cone of each output independently.

Given an input vector $\mathbf{x}$ and arrival times $(arr(x_1), \ldots, arr(x_n)) = (a_1, \ldots, a_n)$, the signal stable time at the output z can be determined by using the delay abstraction D_z. Suppose that D_z has multiple timing tuples $T_1, \ldots, T_m$ for $\mathbf{x}$. For each timing tuple $T_i = (t_{i,1}, \ldots, t_{i,n})(i = 1, \ldots, m)$ the signal stable time s_i at z under T_i is computed as

$$s_i = \max_j (a_j + t_{i,j}).$$

Since all timing tuples are valid, the signal stable time s at z is determined by taking the earliest time among s_i's.

$$s = \min_i s_i = \min_i \max_j (a_j + t_{i,j}).$$

3 Timing-Safe Replaceability for Combinational Modules

Suppose that we have a combinational module $\mathcal{M}_{org}$ and another combinational module $\mathcal{M}_{new}$ that is claimed to be a sped-up version of $\mathcal{M}_{org}$. Assume that they are single-output modules and functionally equivalent. We are interested in verifying whether $\mathcal{M}_{new}$ is indeed no slower than $\mathcal{M}_{org}$. In other words, we need to verify whether using $\mathcal{M}_{new}$ instead of $\mathcal{M}_{org}$ worsens the delay through this module under some input vector and arrival times at the inputs of the module. If there exist such a vector and arrival times, $\mathcal{M}_{new}$ is not a safe replacement of $\mathcal{M}_{org}$ in terms of timing since under that particular situation one can observe that $\mathcal{M}_{new}$ in fact deteriorates the performance of $\mathcal{M}_{org}$. Throughout this paper we assume that the only timing property we need to preserve is when the output of the module is stabilized. It is acceptable for the output to be available earlier than in the original module, but it should never become stable later.

Definition 1 *$\mathcal{M}_{new}$ is said to be a* timing-safe replacement *of $\mathcal{M}_{org}$, $\mathcal{M}_{new} \preceq \mathcal{M}_{org}$, if there exists no (input vector, arrival times)-pair at the inputs such that the output becomes stable later in $\mathcal{M}_{new}$ than in $\mathcal{M}_{org}$.*

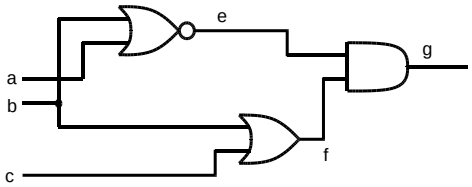

Figure 3: A Timing Non-Safe-Replaceable Module $\mathcal{M}_1$

Definition 2 *Given two timing tuples $(t_1, \ldots, t_n)$ and $(t'_1, \ldots, t'_n)$, $(t_1, \ldots, t_n) \trianglelefteq (t'_1, \ldots, t'_n)$ if and only if $\forall i. t_i \leq t'_i (i = 1, \ldots, n)$.*

The following theorem states that, given two combinational modules, whether one is a timing-safe replacement of the other can be determined by checking if a certain property holds between their delay abstractions.

Theorem 1 *Let D_{org} and D_{new} be the delay abstractions of $\mathcal{M}_{org}$ and $\mathcal{M}_{new}$ respectively. $\mathcal{M}_{new} \preceq \mathcal{M}_{org}$ if and only if for every input-vector/timing-tuple pair $(\mathbf{x}, (t_1, \ldots, t_n)) \in D_{org}$, there exists an input-vector/timing-tuple pair $(\mathbf{x}, (t'_1, \ldots, t'_n)) \in D_{new}$ such that $(t'_1, \ldots, t'_n) \trianglelefteq (t_1, \ldots, t_n)$.*

It is easy to see that $\preceq$ is reflexive, transitive and anti-symmetric. Aziz *et al.* [1] proposed a different notion of timing-safe replaceability, where each gate is given a minimum delay and a maximum delay, and a circuit is called a timing-safe replacement if and only if the delay range of the output is completely contained in the delay range of the original circuit. Our definition is more relaxed since the only delay property of interest is maximum delay. Therefore, speeding up a circuit preserves timing-safe replaceability in our definition while it may not in [1].

Based on this theory one can safely determine when a combinational module can be replaced with another without increasing the delay through the module under any environment.

Consider a circuit $\mathcal{M}$ shown in Figure 1 taken from [2]. Assume the unit delay model[2]. The delay abstraction D of $\mathcal{M}$ is:

abc	$d_{a\to g} d_{b\to g} d_{c\to g}$
000	$\{(3, -\infty, 2), (-\infty, 3, 2)\}$
001	$\{(2, 2, 2)\}$
010	$\{(3, -\infty, 2), (-\infty, 2, -\infty)\}$
011	$\{(-\infty, 2, -\infty)\}$
100	$\{(2, -\infty, -\infty), (-\infty, 3, 2)\}$
101	$\{(2, -\infty, -\infty)\}$
110	$\{(2, -\infty, -\infty), (-\infty, 2, -\infty)\}$
111	$\{(2, -\infty, -\infty), (-\infty, 2, -\infty)\}$,

where $-\infty$ denotes that the availability of the corresponding input is irrelevant to the stability of the output. The input-edge of path $P_a = a \to d \to f \to g$ is stuck-at-0 redundant and stuck-at-1 redundant. Figure 2 shows the circuit $\mathcal{M}_0$ obtained from $\mathcal{M}$ by removing the stuck-at-0 redundancy at the input edge of path P_a. The delay abstraction D_0 of $\mathcal{M}_0$ is:

abc	$d_{a\to g} d_{b\to g} d_{c\to g}$
000	$\{(-\infty, -\infty, 1)\}$
001	$\{(2, 2, 1)\}$
010	$\{(-\infty, -\infty, 1), (-\infty, 2, -\infty)\}$
011	$\{(-\infty, 2, -\infty)\}$
100	$\{(2, -\infty, -\infty), (-\infty, -\infty, 1)\}$
101	$\{(2, -\infty, -\infty)\}$
110	$\{(2, -\infty, -\infty), (-\infty, 2, -\infty), (-\infty, -\infty, 1)\}$
111	$\{(2, -\infty, -\infty), (-\infty, 2, -\infty)\}$.

[2] To simplify the exposition of ideas we use the unit delay model all the way through the paper. However, the theory developed here is general. Note that wire delay can also be handled naturally by assuming the existence of a virtual buffer whose delay is set to the wire delay.

It is easy to see that $\mathcal{M}_0 \preceq \mathcal{M}$. For example, under input vector $(0,0,0)$, D has two timing tuples, $(3,-\infty,2)$ and $(-\infty,3,2)$. D_1 has a single tuple $(-\infty,-\infty,1)$ for this vector, which gives $(-\infty,-\infty,1) \trianglelefteq (3,-\infty,2)$ and $(-\infty,-\infty,1) \trianglelefteq (-\infty,3,2)$. All the other input vectors also meet the condition of timing-safe replaceability. Therefore, $\mathcal{M}_0$ can always be used instead of $\mathcal{M}$ under any environment without having a negative impact on delay.

If we remove the stuck-at-1 redundancy of the input edge instead, another circuit $\mathcal{M}_1$ shown in Figure 3 is obtained. The delay abstraction D_1 of $\mathcal{M}_1$ is:

abc	$d_{a\to g} d_{b\to g} d_{c\to g}$
000	$\{(-\infty,2,2)\}$
001	$\{(2,2,2)\}$
010	$\{(-\infty,2,-\infty)\}$
011	$\{(-\infty,2,-\infty)\}$
100	$\{(2,-\infty,-\infty),(-\infty,2,2)\}$
101	$\{(2,-\infty,-\infty)\}$
110	$\{(2,-\infty,-\infty),(-\infty,2,-\infty)\}$
111	$\{(2,-\infty,-\infty),(-\infty,2,-\infty)\}$.

$\mathcal{M}_1 \not\preceq \mathcal{M}$ since, for example, under input vector $(0,0,0)$ D has a timing tuple $(3,-\infty,2)$, but the only timing tuple $(-\infty,2,2)$ in D_1 does not meet the safe replaceability condition, i.e. $(-\infty,2,2) \not\trianglelefteq (3,-\infty,2)$. This implies that if $(arr(a),arr(b),arr(c)) = (-3,0,-2)$, the output becomes stabilized at $t=0$ in $\mathcal{M}$ while under the same condition $\mathcal{M}_1$ only becomes stable at $t=2$.

4 False Paths in Combinational Modules

We will show by an example that the conventional definition of false paths such as floating mode analysis [3] is not appropriate for defining the falsity of paths in combinational modules. Specifically we show that the same input-output path of a module can be true under some arrival time condition at its primary inputs, while false under another. Since a combinational module can be used under any surrounding environment, we introduce a more stringent notion of false paths, where a path is said to be *strongly false* if it is false under *any* arrival time condition. We then illustrate how one can systematically determine the strong falsity of a path.

Consider a circuit $\mathcal{M}$ in Figure 1. Assume the unit delay model. If the primary inputs a, b and c arrive at $t = 1, 0$ and 1 respectively, functional timing analysis guarantees that the output g is stabilized at $t = 3$. Note that the topological delay of this circuit under the arrival time condition is $t = 4 (> 3)$ because of the path $P_a = a \to d \to f \to g$. Since g is available at $t = 3$, P_a is false. Consider another path $P_b = b \to d \to f \to g$. Given an input vector $(a,b,c) = (0,0,0)$, P_b is true under this arrival time condition.

Now, let us analyze the same circuit under a different condition where a, b and c arrive at $t = 0, 1$ and 1 respectively. The output g is again available at $t = 3$. Therefore P_b is false since otherwise the output would become stable at $t = 4$. Note that P_b was true under the previous condition. P_a, which was false before, is true this time, for example, under input vector $(a,b,c) = (0,0,0)$.

These two cases clearly demonstrate that the falsity of a path is relative to a given arrival time condition, and that the same path can be false in one condition and true in another under an arrival-time dependent path sensitization condition such as floating mode analysis [3].

One can examine this circuit more systematically by examining the delay abstraction of the circuit. The delay abstraction D of this circuit is already shown in Section 3. In this circuit P_a (P_b) is the only path of length 3 from a (b) to g. Therefore, the fact that the delay abstraction has a timing tuple whose first (second) element is 3 means that, given the corresponding input vector, it is possible to make P_a (P_b) responsible for delay.

For example under $(a,b,c) = (0,0,0)$, when the arrival times of a, b and c are $(arr(a),arr(b),arr(c)) = (1,0,1)$, the second timing tuple $(-\infty,3,2)$ gives an earlier signal stable time of $t = 3$ at the output than the first timing tuple $(3,-\infty,2)$ giving the stable time of $t = 4$. As described in Section 2, the timing tuple that gives the earliest stable time can be used in determining the timing behavior of the output. Since the delay from b to the output in this second timing tuple is 3, the corresponding path P_b is true. P_a is false since the timing tuple indicates that input a is irrelevant. If $(arr(a),arr(b),arr(c)) = (0,1,1)$, however, the first timing tuple gives an earlier stable time than the second, showing that P_a is true and P_b is false. This analysis illustrates that once the delay abstraction of a module is computed, false path analysis can be performed under any arrival time condition only using the delay abstraction.

We are now ready to introduce a new definition of false paths for combinational modules.

Definition 3 *Let $\mathcal{M}$ be a single-output combinational module whose primary inputs are $x_1,\ldots,x_n$. Let z be the primary output of the module.* The path set of length $\geq L$ from x_i, $\Pi(x_i, L)$, *is the set of all input-output paths that start from x_i and end at z and whose topological delays are $\geq L$.*

Definition 4 $\Pi(x_i, L)$ *is said to be strongly false if and only if the delay abstraction D of the module $\mathcal{M}$ contains no timing tuple where the delay corresponding to x_i is greater than or equal to L under any input vector. Otherwise, $\Pi(x_i, L)$ is said to be not strongly false.*

The idea behind this new definition is that a path set is said to be strongly false if all the paths in the set are false under *any* arrival time condition. In this definition neither $\Pi(a,3) = \{P_a\}$ nor $\Pi(b,3) = \{P_b\}$ in the example is strongly false since it is possible to sensitize these paths as we saw.

We also define a new class of false paths where the strong falsity of a path set is claimed only under some value at the input.

Definition 5 $\Pi(x_i, L)$ *is said to be* strongly false for value 0 (1) at x_i *if the delay abstraction D of the module $\mathcal{M}$ contains no timing tuple where the delay corresponding to x_i is greater than or equal to L under any input vector that has value 0(1) for x_i.*

Note: If $\Pi(x_i, L)$ is strongly false, it is strongly false for both values 0 and 1 at x_i.

$\Pi(a,3) = \{P_a\}$ is strongly false for value 1 at a since under $a = 1$ there is no timing tuple where the delay from a is 3. $\Pi(b,3) = \{P_b\}$ is strongly false for value 1 at b similarly.

The definition of strongly false paths is based on the false-path-aware delay abstraction of a given module. However, it is an expensive operation to compute the delay abstraction of a large network. To alleviate this difficulty we have developed an algorithm to determine whether a path set is strongly false or not without computing the delay abstraction of the circuit. The main idea of the algorithm is a reduction of the decision problem to a satisfiability problem, which is then solved by a satisfiability checker. This approach makes it possible to perform strong falsity checks of paths on large networks. The detail of the algorithm is available in [5]. We confirmed the applicability of the algorithm by experiments, which will be reported in Section 6.

5 False Path Removal for Combinational Modules

Strongly false paths are the only paths that can be safely assumed to be false for a combinational module since the actual environment under which a combinational module is to be used is unknown. Since they are never responsible for the stability of an output under any arrival time condition, it is desirable if they can be structurally removed from the module by a circuit transformation. If such a transformation is possible, the resulting module is false-path-free and thus can be analyzed accurately even with topological timing analysis. Although this transformation is attractive, we do not want

to slow down the original circuit by the transformation especially in the context of high-performance designs. Thus, the structural transformation also needs to guarantee that the resulting module $\mathcal{M}'$ is no slower than the original $\mathcal{M}$ under any arrival time condition. In this section we present an algorithm that removes strongly false paths from a combinational module $\mathcal{M}$ without increasing the delay of the module under any arrival time condition, i.e. $\mathcal{M}' \preceq \mathcal{M}$.

5.1 The KMS Algorithm

Keutzer, Malik and Saldanha [4] showed that redundancy is not necessary to reduce delay. The motivating example for the work was a carry-skip adder. This circuit has a single stuck-at redundancy, but the direct removal of the redundancy makes a long false path true thereby slowing down the circuit. The redundancy in the circuit is a by-product of making its longest topological path false to improve the performance. However, such a redundant circuit is problematic since the existence of the fault causes the circuit to slow down, but the fault is not detectable by conventional testing techniques. A natural question is whether redundancy is necessary to reduce delay in general. They resolved this issue negatively by giving a constructive algorithm, commonly known as the KMS algorithm.

The KMS procedure [4] takes 1) a gate-level redundant combinational circuit and 2) arrival time for each primary input, and returns a functionally equivalent irredundant circuit no slower than the original under the given arrival times. The core of the algorithm is 1) the isolation of long false paths by circuit duplication and 2) the removal of the false paths by propagating a constant from the input edges of the paths. After removing all long false paths, the longest topological path is guaranteed to be true. Remaining redundancies are then removed to obtain an irredundant circuit.

Given a combinational circuit whose longest topological paths are false under a given arrival time condition, one can simply apply the KMS procedure to obtain a false-path-free circuit which is no slower than the original. The final circuit can then be used as a replacement of the original without the risk of slowing down the circuit.

This approach, however, has fundamental limitations to be used for false path removal of combinational modules.

First, the KMS procedure takes an arrival time condition at the primary inputs and works under this particular condition. Therefore, it is not directly applicable to a combinational module since the arrival times at the inputs are unknown. If a representative arrival time condition is chosen and the procedure is applied under the condition, the delay of the resulting circuit is not guaranteed once it is used under a different arrival time condition. Second, redundancy removal performed as the final step of the KMS algorithm can increase delay even under the arrival time condition chosen for the analysis if the delay of the circuit is examined for each input vector separately. Keutzer *et al.* [4] argued that straight-forward redundancy removal cannot slow down the circuit since the topological longest path is true after false path removal. In this argument the delay of a circuit is defined as the earliest time when *all* the primary outputs are stabilized for *all* input vectors under a given arrival time condition at the inputs. However, the delay of an output under an input vector can increase as the result of redundancy removal although it never increases so much as to increase the "delay" of the circuit. Under the delay definition of [4] this local delay increase for an input vector does not cause the increase of the "delay". However, since there exists a surrounding environment of the module which can detect this delay increase, it should be thought of as a delay increase in the context of combinational modules.

5.2 Motivating Examples

This subsection shows why a simple-minded application of the KMS algorithm is not appropriate to remove false paths from combinational modules.

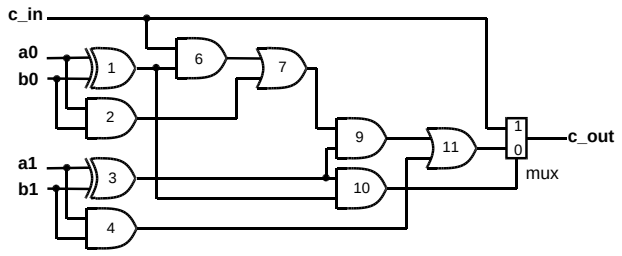

Figure 4: 2-bit Carry-Skip Adder $\mathcal{C}$

An example is a carry-skip adder. This is the circuit that motivated the entire research on the KMS algorithm. Figure 4 shows a 2-bit carry-skip adder described in [4]. Assume a gate delay of 1 for the AND gate and the OR gate, and gate delays of 2 for the XOR gate and the MUX gate. The selector input of the multiplexor is stuck-at-0 redundant since under the existence of the fault, the circuit simply degenerates into a ripple-carry adder, which is functionally equivalent to the original circuit. The performance of the circuit, however, is deteriorated by the fault since the ripple-carry adder is slower than the carry-skip adder.

In [4] this circuit is analyzed under the condition where the carry input arrives at $t = 5$ and all other inputs arrive at $t = 0$. In this particular situation the longest topological delay is 11 by the path of length 6 $(c_{in}, g_6, g_7, g_9, g_{11}, mux, c_{out})$. Since this longest path is false under the given arrival times, the KMS algorithm is invoked to remove the path.

If the resulting circuit is used under the same arrival time condition, it is guaranteed to be no slower than the original. However, under a different arrival time condition it is possible that the performance of the resulting circuit is worse than that of the original.

Let us analyze the same circuit under different arrival times to see the problem. Assume that all the inputs arrive at $t = 0$. The topological longest paths are now the paths of length 8 from a_0 and b_0 to c_{out}[3]. These paths are true under the arrival time condition. Therefore if one simply follows the KMS procedure, any redundancy can be removed arbitrarily without slowing down the circuit, which results in a ripple carry adder. Notice that although the effective delay from the carry input to the carry output has increased in this transformation, the delay of 8 from a_0 and b_0 still determines the circuit performance. Thus the transformation is valid under the given arrival times. Now assume that the resulting circuit is used where the carry input arrives at $t = 5$ and the other inputs arrive at $t = 0$. Obviously we now observe a larger delay of 11 instead of 8. This example clearly shows that the delay non-increasing property of the KMS algorithm is only guaranteed for a given arrival time condition at primary inputs. In order to remove false paths from a combinational module we are interested in a more robust algorithm which never slows down the circuit under *any* arrival time condition.

Finally assume that the carry input arrives at $t = 5$ and all the other inputs arrive at $t = 0$ again. The removal of the long false path from the carry input to the carry output yields the circuit $\mathcal{C}_0$ in Figure 5. Each of the fanin edges of g_2 is stuck-at-1 redundant. If one follows the KMS procedure, any of these redundancies can be removed without slowing down the circuit. If the a_0 edge is replaced with a constant 1, the circuit $\mathcal{C}_{0-RR}$ in Figure 6 is obtained. Now that the b_0 edge is not redundant any more, this is the final result of the KMS procedure.

We are now ready to show that this redundancy removal in fact increases the delay of the circuit even under the arrival time condition analyzed, once the delay is determined for each input vector separately. Consider the input vector $(a_0, a_1, b_0, b_1, c_{in}) =$

[3]The long path from c_{in} considered in the previous case has length 6 and is no longer the longest.

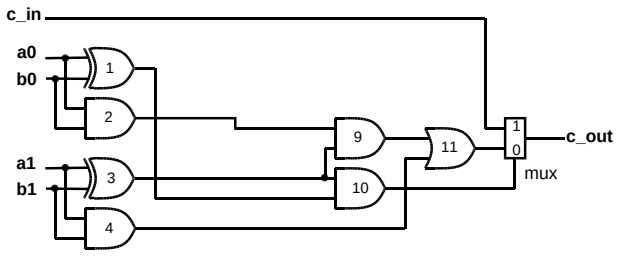

Figure 5: 2-bit Carry-Skip Adder $\mathcal{C}_0$ before Redundancy Removal

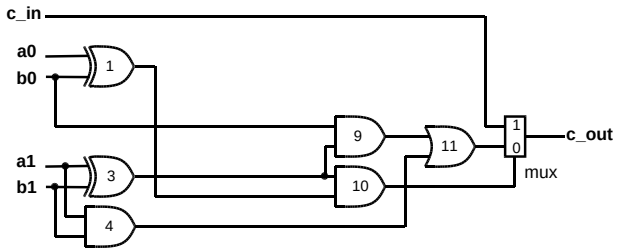

Figure 6: 2-bit Carry-Skip Adder $\mathcal{C}_{0-RR}$ after Redundancy Removal

$(0,0,1,0,0)$. In the circuit before the redundancy removal, the path $(a_0, g_2, g_9, g_{11}, mux, c_{out})$ is the longest true path. Therefore the delay under this vector is 5. On the other hand, in the circuit after the redundancy removal, the paths $(\{a_1, b_1\}, g_3, g_9, g_{11}, mux, c_{out})$, which were false before the redundancy removal, become true and give delay $6(>5)$. Notice that there exists an input vector that sensitizes the longest topological path from c_{in} to c_{out} of length 7 in both of the circuits. Therefore, the redundancy removal is safe under the traditional definition of delay. However, if we need to preserve the performance of the circuit under any surrounding environment, redundancy removal can worsen the delay.

5.3 Algorithm for False Path Removal of Combinational Modules

We argued that simple-minded application of the KMS algorithm is not enough for our purpose. The first problem is that the KMS algorithm only removes long false paths under given arrival times. Because of this strategy, false paths not critical under the situation remain in the circuit. To make matters worse, those paths can become true long paths after redundancy removal thereby slowing down the circuit under a different arrival time condition. Moreover, since false paths removed by the KMS algorithm are not necessarily strongly false, even false path removal alone can slow down the circuit.

To alleviate this problem all long strongly false paths are removed from each input by a circuit transformation. As a result, the topological longest path from any input is responsible for delay under some input vector and some arrival time condition.

The second problem is that the final redundancy removal in the KMS algorithm can slow down a circuit if the delay of the circuit is computed for each primary input vector. This is unacceptable for combinational modules since there exists a surrounding environment whose performance is deteriorated by this delay increase. Therefore the redundancy removal is dropped intentionally.

We first illustrate the key idea of the algorithm using an example. Consider again the circuit $\mathcal{M}$ in Figure 1. We have already shown that $\Pi(a,3) = \{P_a\}$ is strongly false for 1 at a in Section 4. This means that if $a = 1$, this path is never responsible for the signal stability at the output under any arrival time condition. Therefore, the input edge of the path can be safely replaced with a constant 0 without slowing down the circuit. Notice that the path is fanout-free and this modification cannot adversely affect the other paths. Propagating this constant through the circuit is also a safe operation. We already saw in Section 3 that the resulting circuit $\mathcal{M}_0$ in Figure 2 is a timing-safe replacement of $\mathcal{M}$.

If the original KMS algorithm is applied to $\mathcal{M}$ in Figure 1 under $(arr(a), arr(b), arr(c)) = (1,0,1)$, P_a is identified as false. One can then choose either constant 0 or 1 to replace the input edge of P_a with. If a constant 1 is chosen, the circuit $\mathcal{M}_1$ in Figure 3 is obtained, which is not a timing-safe replacement of $\mathcal{M}$ as discussed in Section 3. This example shows that strongly false paths are the appropriate paths to be removed for combinational modules. Keutzer *et al.* [4] only suggest that one pick the constant that gives better simplification of the circuit. However, they use this choice only as an optimization.

We are ready to prove the correctness of the transformation formally.

Definition 6 *The set of all paths beginning at an primary input edge e and ending at a primary output is called the* path set of e.

Definition 7 *An* L-path disjoint circuit[4] *with respect to primary input x_i is a circuit where the path set of any primary input edge from x_i consists of either paths of length $\geq L$ or paths of length $< L$.*

Given a combinational module, one can always construct a module that is L-path disjoint with respect to x_i by fully preserving the original functional and timing properties. The detail can be found in [9][5]. Let $\mathcal{M}$ be a single-output combinational module whose primary inputs are $x_1, \ldots, x_n$. Let $\mathcal{M}'$ be an L-path disjoint circuit with respect to x_i that is obtained from $\mathcal{M}$.

Lemma 1 *If path set $\Pi(x_i, L)$ is strongly false for value $v (v = 0,1)$ at x_i in $\mathcal{M}$, the input edge of the path set is stuck-at-$\bar{v}$ redundant in $\mathcal{M}'$.*

Theorem 2 *Suppose path set $\Pi(x_i, L)$ is strongly false for value v at x_i in $\mathcal{M}$. Let $\mathcal{M}'$ be an L-path disjoint circuit with respect to x_i obtained from $\mathcal{M}$. Let $\mathcal{M}''$ be the circuit obtained from $\mathcal{M}'$ by substituting $\bar{v}$ for the input edge of $\Pi(x_i, L)$ in $\mathcal{M}'$. Finally let $\tilde{\mathcal{M}}$ be the circuit obtained from $\mathcal{M}''$ by performing a constant propagation of $\bar{v}$. Then $\tilde{\mathcal{M}} \preceq \mathcal{M}$.*

Based on Theorem 2 one can design a procedure that takes a single-output combinational module and removes strongly false paths to create a timing-safe replacement module where the longest topological path from any input is strongly false neither for 0 nor for 1 at the input. The algorithm examines primary inputs one by one by checking whether the longest topological paths from a primary input are strongly false for either 0 or 1. If either is true, the circuit is modified based on Theorem 2. This process is repeated until no change is observed. To handle a combinational module with multiple outputs the same procedure is applied to the transitive fanin cone of each primary output separately and the resulting circuits are merged into a single circuit by sharing isomorphic subcircuits. This step is guaranteed not to change the timing characteristics of the circuits[6].

Refer to the 2-input carry-skip adder in Figure 4 again. The analysis of strongly false paths on this circuit indicates that path set $\Pi(c_{in}, 6)$ is strongly false for value 1 at c_{in}. Therefore, one can safely assert a constant 0 at the input edge of the long path of length 6 from the carry input to the carry output. Note that the circuit is already L-path disjoint with respect to c_{in} for $L = 6$. Figure 5 shows the resulting false-path-free circuit $\mathcal{C}_0$. $\mathcal{C}_0$ is a timing-safe replacement of $\mathcal{C}$. $\Pi(c_{in}, 6)$ is not strongly false for value 0 at c_{in}. Therefore we cannot use 1 for constant propagation.

[4] This definition is a variation of L-path disjoint circuits introduced in [9].

[5] The procedure in [9] is applicable by assuming that x_i arrives at $t = 0$ and all the other inputs arrive at $t = -\infty$. All the paths from the other inputs are ignored effectively this way.

[6] As in [4] we assume that gate delay is independent of loads.

6 Experimental Results

We implemented on top of SIS a procedure to check if a path set is strongly false for a value (0 or 1) at the corresponding primary input, and to remove such a path set structurally. The procedure determines the strong falsity of a path set without constructing the delay abstraction of a given module explicitly, and thus is applicable to large networks. The original problem is reduced into a satisfiability problem, which is then solved by a satisfiability checker. The removal of a false path set is a simple structural transformation, and thus takes negligible time compared with strong falsity checking. We only summarize the result of a representative circuit, the largest primary output cone of C7552. The cone has 194 primary inputs and 1096 gates. Recall that the strong falsity of a path set is defined for a single-output network. For each primary input the strong falsity of the longest paths from the input was tested for both values 0 and 1, and the paths were removed if they are strongly false. We found out that for 8 out of 194 primary inputs the longest paths from each primary input are strongly false for either value 1 or 0 at the input. A strong falsity check for a primary input took 37.5 seconds in the average on DEC AlphaServer 8400 5/625[7]. These strongly false paths were then removed by the proposed procedure. The resulting circuit has 1216 nodes, only 11% area increase from the original.

7 Conclusions

Three issues arising in delay characterization of combinational modules have been discussed in this paper. First we introduced a new notion called timing-safe replaceability as a formal way of comparing the timing characteristics of two combinational modules under an unknown arrival time condition. Second, we argued that the conventional definition of false paths is relative to a given arrival time condition, and thus is not appropriate to define the falsity of a path in a combinational module used in various surrounding environments. This led to a new definition of false paths, called strongly false paths, for combinational modules. Strongly false paths are the paths that are false under any arrival time condition, and can safely be assumed to be false for combinational modules. Finally, we proposed a procedure to remove those false paths from a combinational module with a formal guarantee that the final circuit is no slower than the original under any arrival time condition. We have shown that the final redundancy removal in the KMS algorithm can increase the delay of a circuit if the delay is defined for each primary input vector separately. Therefore the proposed algorithm does not make the circuit irredundant after false path removal. Thus, one of the advantages of the KMS algorithm, i.e. removing redundancies that cannot be detected by conventional testing, but can adversely affect the timing of the circuit, is absent. Whether redundancies can be removed without slowing down the circuit under this strict definition of delay is still open.

References

[1] A. Aziz, R. K. Brayton, F. Balarin, and V. Singhal. Timing-safe replaceability for combinational designs. In *Proceedings of TAU 95: ACM/SIGDA International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 121–128, November 1995.

[2] D. Brand and V. S. Iyengar. Timing analysis using functional analysis. *IEEE Transactions on Computers*, 37(10):1309–1314, October 1988.

[3] H.-C. Chen and D. H.-C. Du. Path sensitization in critical path problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(2):196–207, February 1993.

[4] K. Keutzer, S. Malik, and A. Saldanha. Is redundancy necessary to reduce delay? *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(4):427–435, April 1991.

[5] Y. Kukimoto. *Timing Analysis and Optimization for High-Performance Digital Circuits*. PhD thesis, University of California, Berkeley, October 1998.

[6] Y. Kukimoto and R. K. Brayton. Exact required time analysis via false path detection. In *Proceedings of 34th Design Automation Conference*, pages 220–225, June 1997.

[7] Y. Kukimoto and R. K. Brayton. Hierarchical functional timing analysis. In *Proceedings of the 35th Design Automation Conference*, pages 580–585, June 1998.

[8] P. C. McGeer, A. Saldanha, R. K. Brayton, and A. Sangiovanni-Vincentelli. Delay models and exact timing analysis. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 167–189. Kluwer Academic Publishers, 1993.

[9] A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Circuit structure relations to redundancy and delay. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(7):875–883, July 1994.

[7] We expect that this CPU time can be further reduced by incorporating existing techniques for generating simplified SAT formulas developed for functional timing analysis [8].