Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms

Srinivas Devadas	Kurt Keutzer	Sharad Malik
MIT	Synopsys	Princeton University
Cambridge, MA	Mountain View, CA	Princeton, NJ

Abstract

This paper addresses the problem of accurately computing the delay of a combinational logic circuit in the floating mode of operation. (In this mode the state of the circuit is considered to be unknown when a vector is applied at the inputs.) It is well known that using the length of the topologically longest path as an estimate of circuit delay may be pessimistic since this path may be false, *i.e.* it cannot propagate an event. Thus, the true delay corresponds to the length of the longest true path. This forces us to examine the conditions under which a path is true. We introduce the notion of static co-sensitization of paths which leads us to necessary and sufficient conditions for determining the truth or falsity of a single path, or a set of paths. We apply these results to develop a delay computation algorithm that has the unique feature that it is able to determine the truth or falsity of entire sets of paths simultaneously. This algorithm uses conventional stuck-at-fault testing techniques to arrive at a delay computation method that is both correct and computationally practical, even for particularly difficult circuits.

1 Introduction

Given input stimuli, a combinational circuit produces outputs. These outputs depend on the sensitization of particular paths in the circuit. Certain paths in a combinational circuit may never transmit an event because the logical functions computed by the gates and their propagation delays preclude these paths from being sensitized. These paths are said to be *false* and must be excluded in analyzing the delay of the circuit. This paper examines the problem of computing the *true* delay of a combinational logic circuit that ignores the contribution of these false paths.

We consider the floating mode of operation [5]. In the floating mode of operation when a vector is applied to the primary inputs, the states of the nodes in the circuit are assumed to be unknown. Pessimistic assumptions are made about the states so as to consider any possible event propagation based on previous history. In order to accomplish this, if the input vector results in a controlling value at a gate input, the unknown side-inputs are assumed to have non-controlling values on them so as to facilitate the propagation of the event at this gate.

We use a circuit representation developed by Armstrong, called the Equivalent Normal Form [1], to examine the interactions between the functional and temporal aspects of the behavior of a circuit. We introduce the notion of static co-sensitization of paths that leads us to the necessary and sufficient conditions for a path to be true, or for some path within a specified set of paths to be true. [1] We then use our understanding of these conditions to develop a delay computation algorithm that has the unique feature that it is able to determine the (truth or) falsity of entire *sets of paths*, simultaneously. Because the circuits that are most troublesome for false-path-eliminating static timing analyzers are those with literally millions of paths, and in particular millions of longest paths, the ability to handle entire sets of paths simultaneously results in a very efficient delay computation procedure.

We begin by a taking a brief look at the previous work done in this area in Section 2 and highlight the main limitation of the existing work; thus setting the stage for the research presented in this paper. Next, we formally define important terms in Section 3 and take a closer look at the delay models considered in Section 4. In Section 5 we describe the Equivalent Normal Form (ENF) of a combinational circuit. We introduce the notion of static co-sensitization in terms of the ENF in Section 6 while developing the necessary and sufficient conditions for a path to be true. We then use these conditions to derive a delay analysis algorithm in Section 7. Preliminary results on applying

[1] Chen and Du [5] were the first to determine the the necessary and sufficient conditions for a path to be true, However, our analysis technique is unique and lends itself to easy extension in dealing with sets of paths.

this algorithm to particularly troublesome circuits are given in Section 8.

2 Previous Work

There has been significant research done in the area of timing analysis that includes consideration of false paths during the analysis stage. The important landmarks in this will be pointed out here. A more detailed history may be found in [12].

The earliest reference to this problem is made by Hrapcenko [11] who demonstrated the existence of false paths on a parametric circuit that he constructed. Initial attempts to deal with false paths in timing analysis depended on the designer providing some input to the timing analysis routine. This was done by explicitly listing the paths that were known to be false and had to be ignored [10, 6]. This had two drawbacks. First, the number of false paths may be very large, making their explicit listing very difficult. Second, the procedure to determine the truth or falsity of paths is non-trivial, and except for the most obvious cases this was beyond what can be expected from a designer. This task should be performed by the analysis algorithm itself.

Perhaps the first attempt at using the functional behavior of circuit elements during timing analysis was made by Brand and Iyengar [3]. However, as pointed out by the authors themselves, the conditions established for a path to be false were only necessary, not sufficient, and hence the computed delay was a correct but possibly pessimistic estimate. McGeer and Brayton introduced the notion of *viability* of paths, a path had to be viable for it to be true. Hence, viability too may pessimistically result in a path being classified as true, while it may actually be false. Chen and Du [5] were the first to propose necessary and sufficient conditions for a path in the circuit to be true. A common limitation of all of these previous techniques is that they focus on one path in the circuit at a time. This limits their practical utility to circuits that have only a few long false paths which may be individually eliminated in the analysis. For circuits where this is not true, this path-at-a-time approach breaks down. It is precisely this limitation that the work presented in this paper addresses.

3 Definitions and Notation

As this paper will span the areas of synthesis, testing and timing analysis, we need to provide a minimal amount of terminology from these fields.

3.1 Logic Synthesis

A **Boolean function** F of N variables is a mapping from $B^N = \{0,1\}^N \rightarrow \{0,1\}$. We model B^N as a Binary N-cube. A vertex (or input vector) $v \in B^N$ for which $F(v) = 1$ is a member of the ON-set. If $F(v) = 0$ then v is a member of the OFF-set.

A **literal** is a Boolean variable or its complement. A **cube** is a set of literals and is interpreted as a product of literals. For example $\{a, b, c\}$ is a cube, interpreted as $a \cdot b \cdot c$ which may be abbreviated to abc. A **minterm** is a cube in which every variable in the Boolean function appears. We may interpret the minterm as a **vertex** in the N-cube.

Minterms and cubes may be used to represent the values of a set of input variables: *e.g.* $x\overline{y}z$ is shorthand for $x = 1,\ y = 0$, and $z = 1$. In this way there is a natural correspondence between an input vector or input stimulus, a minterm and a vertex in the N-cube. This correspondence may be extended to cubes where unspecified values in the function are assumed to be undefined values. Thus, if a circuit C has inputs $v,\ w,\ x,\ y,\ z$ then applying the cube $x\overline{y}z$ to C is shorthand for applying $v = X,\ w = X,\ x = 1,\ y = 0$, and $z = 1$. (Here X denotes an unknown value.)

Following the historical usage we define a **cover** as a set of cubes and we interpret the cover as a sum-of-products expression. For example $\{\{a, b, c\}, \{d, e, f\}\}$ is a cover, interpreted as $abc + def$. We say that a cube q **covers** a cube (or vertex) r if $q \subseteq r$. If a cube q covers only ON-set vertices of a Boolean function F then q is called an **implicant** of F. A **relatively essential vertex** of a cube q in a cover C is a vertex that is covered by q and is not contained in any other cube in C.

3.2 Testing

A Combinational Logic Circuit is represented as a labeled, directed, acyclic graph (**dag**) $G = (V, E)$ with each vertex v labeled with the name of a primitive gate such as **and**, **or** or **not**, or with the name of a primary input or output. A combinational logic circuit can be created from gates of arbitrary complexity, but to simplify the discussion we assume that the circuit is expressed in terms of primitive gates. (In Section 7.3 we will address the issue of cell libraries with complex gates in them.) There is an edge $< u,\ v >$ in V between two vertices if the output of the gate associated with u is an input to gate v. If the output of a gate, g_1, is connected to an input of gate, g_2, g_1 is a **fanin** of g_2. Gate g_2 is a **fanout** of gate g_1.

The members of V that have no fanin are the only vertices that may be labeled with the name of a primary input. The members of V that have no fanout are the only vertices that may be labeled with the name of a primary output.

A single-output combinational logic circuit computes a Boolean function in the obvious way. An N input M output combinational logic circuit computes a pseudo-Boolean function from $B^N \rightarrow B^M$. As we will be dealing primarily with combinational logic circuits in this paper, we will use circuit to mean combinational logic circuit.

A gate has an input/output **stuck-at-1** (**stuck-at-0**) fault if the logical value associated with the input/output is 1 (0) independent of the value presented at the input. A circuit has a **single-stuck-at-fault** if there is one stuck-at-fault in the circuit. A circuit has a **multiple-stuck-at-fault** (multifault) if there are one or more stuck-at-faults in the circuit. A multifault is denoted as a set of its single fault components.

For a fuller treatment of testing terminology, see [4, 8].

3.3 Timing/Delay Testing

A **path** in a combinational circuit is an alternating sequence of vertices and edges, $\{v_0, e_0, ..., v_n, e_n, v_{n+1}\}$, where edge e_i, $1 \leq i \leq n$, connects the output of vertex v_i to an input of vertex v_{i+1}. For $1 \leq i \leq n$, v_i is a gate; v_0 is a primary input and v_{n+1} is a primary output. Each e_i is a net. With each vertex v we associate a delay $d(v)$.

The **length** of a path $P = \{v_0, e_0, ..., v_n, e_n, v_{n+1}\}$, is defined as $length(P) = \sum_{i=0}^{n+1} d(v_i)$.

An **event** is a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ at a gate. Consider a sequence of events, $\{r_0,\ r_1,\ ...,\ r_n\}$ occurring at gates $\{g_0,\ g_1,\ ...,\ g_n\}$ along a path, such that r_i occurs as a result of event r_{i-1}. The event r_0 is said to propagate along the path.

A **controlling value** at a gate input is the value that determines the value at the output of the gate independent of the other inputs. For example, 0 is a controlling value for an **and** gate. A **non-controlling value** at a gate input is the value which is not a controlling value for the gate. For example, 1 is a non-controlling value for an **and** gate. We say a gate g has a **controlled value** if one of its inputs has a controlling value; otherwise, we say g has a **non-controlled value**.

Let $\pi = \{v_0, e_0, ..., v_n, e_n, v_{n+1}\}$ be a path. The inputs of v_i other than e_{i-1} are referred to as the **side-inputs** to π.

We say that an input vector w **statically sensitizes to a 1(0)** path π in C iff the value of v_{n+1} is 1(0) and for each v_i, $1 \leq i \leq n+1$, if v_i has a controlled value then the edge e_{i-1} is the only input of v_i that presents a controlling value. Thus for π to be **statically sensitizable** there must exist an input vector such that all the side-inputs along π settle to non-controlling values for that vector.

For notational convenience, we denote the value of a circuit C on a vector v by $C(v)$. For example for the circuit $abc + def$, $C(abcdef) = 1$. We denote the Boolean value of a cube q on a vector v by $q(v)$. For example given the cube $q = abc$, $q(abcdef) = 1$. Similarly, we denote the Boolean value of a literal l in a cube q on a vector v by $q_l(v)$. For example given the cube $q = abc$, $q_b(abcedef) = 1$.

A path is said to be true if it is can propagate an event. The exact conditions for a path to be true will be the subject of investigation of the rest of this paper. The **critical path** is the longest true path in the circuit. The delay of a circuit is the length of the critical path.

4 Circuit and Gate Delay Models

Most timing analyzers make some assumptions about the electrical behavior of the circuit components and possible variations in the delay. This section first examines some of the models used and then specifies the domain of this paper in terms of these models. We first consider the models based on differences in the electrical behavior [2].

Consider the operation of a circuit over the period of application of two consecutive input vectors v_1 and v_2. In the **transition** mode of operation, the circuit nodes are assumed to be ideal capacitors and retain their value set by v_1 till v_2 forces the voltage to change. Thus the timing response for v_2 is also a function of v_1 (and possibly other previously applied vectors). In the **floating** mode of operation, the nodes are not assumed to be ideal capacitors and hence their state is unknown till it is set by v_2. Thus, the timing behavior for v_2 is independent of v_1.

The most common delay model for a circuit component is one in which the delay is assumed to be a fixed number d. This is referred to as the **fixed delay model**. However, in reality this number is typically an upper bound on the expected delay, so in fact the actual delay may be any number bounded above by d. This potential speedup is incorporated in the **monotone speedup** model [12], which assumes that the delay for each component lies in the range $[0, d]$. The **bounded** delay model attempts to be more realistic about how much each component can in fact speed up. It specifies the delays as a pair of numbers, $[d_l, d_u]$, specifying the lower and upper bounds on the actual delays.

For the purpose of developing the ideas in this paper it is sufficient that we restrict the delays in a circuit to gates. This is general enough to accommodate other delay quantities such as wire delays and pin to pin delays by introducing buffers with appropriate delays in the circuit. While more sophisticated delay parameters such as slope delays and separate rise and fall delays are not

[2] This is the same as that introduced in [5].

directly accommodated into the "delay lumped at a gate" paradigm, it will be subsequently shown in Section 7.3 that the results in this paper hold for even those models.

In [5] the following results were shown:

- For the floating mode of operation, circuit delay under the fixed, monotone speedup and bounded delay model are the same (for the same upper bound on each delay value).
- The circuit delay in the floating mode is an upper bound (hence correct though possibly not tight estimate) of the delay in the transition mode.

This paper focuses on the floating mode and considers the monotone speedup delay model. From the previous results we see that it is sufficient to consider the fixed delay model for this purpose.

5 ENF Analysis

5.1 Introduction

Analyzing the Boolean and temporal behavior of multilevel logic circuits using topological arguments can be confusing. Our approach is to use an alternative representation of a multilevel circuit developed by Armstrong [1] called the *equivalent normal form* or ENF. The ENF of a circuit is a two-level representation that represents the logic function computed by the multilevel circuit while retaining the path information.

We illustrate the derivation of the ENF of a circuit with the help of an example taken from [1]. First, the circuit of Figure 1 is made internal fanout free by unfolding it. This is illustrated in Figure 2. The numbers inside the gates are unique identifiers for those gates. This involves duplicating gates if needed so that each copy of a gate has a single fanout connection. The duplicated gates retain the same integer identifier as the original gate. Next, the inverters are pushed backwards towards the primary inputs using DeMorgan's laws of complementation to change the gates encountered in the process. (See Figure 3.)

This resulting circuit is referred to as the **leaf-DAG** circuit. The ENF of the output is constructed bottom up starting at the primary inputs of this circuit. Each variable in an ENF expression is simply a primary input of C, such as i, and has a label (or tag) consisting of a set of gates denoting the path this input has taken to reach the output. The ENF for a uncomplemented (complemented) primary input i is simply $i_{\{\}}$ ($\bar{i}_{\{\}}$). The ENF expression corresponding to the output of an **and** (**or**)

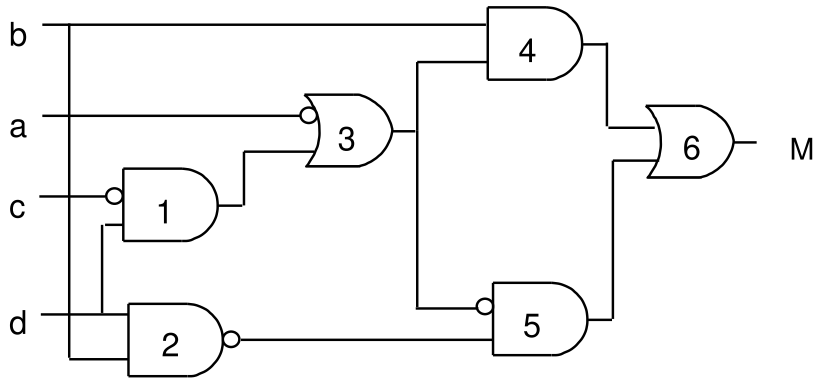

Figure 1: Multilevel Circuit

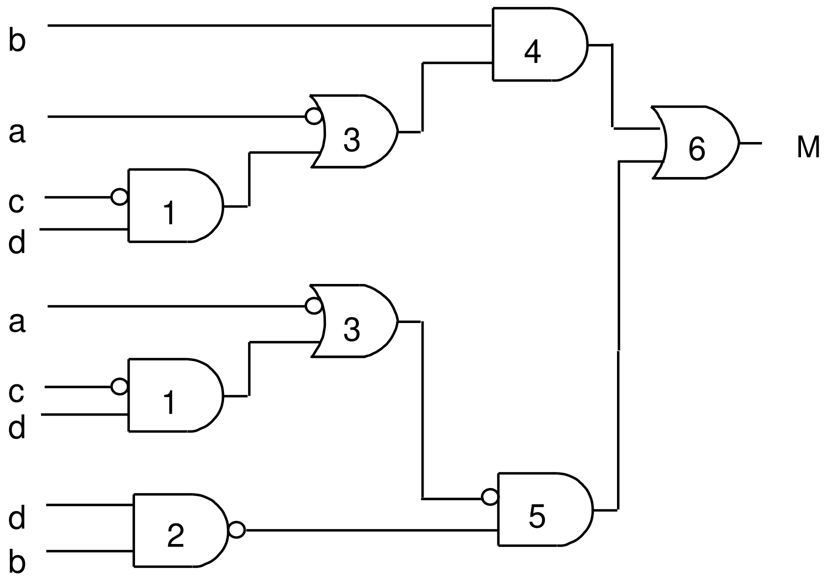

Figure 2: Making a Circuit Fanout Free

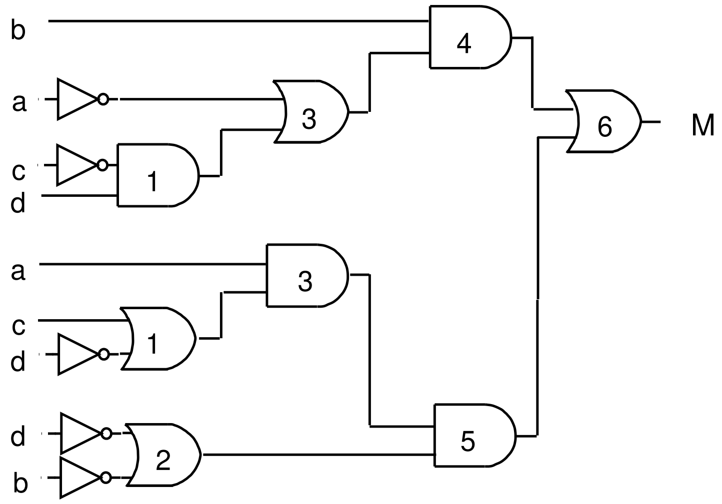

Figure 3: Pushing Inverters to the Primary Inputs

gate g with inputs a_α, b_β is a sum-of-products expression over tagged literals, $E_g = a_{\alpha\cup\{g\}} \cdot b_{\beta\cup\{g\}}$ ($E_g = a_{\alpha\cup\{g\}} + b_{\beta\cup\{g\}}$). No Boolean reductions are made in reducing E_g to sum-of-products form.

The ENF for primary output M is:

$$\begin{aligned} E \;=\; & b_{\{4,6\}}\,\overline{a}_{\{3,4,6\}} + b_{\{4,6\}}\,\overline{c}_{\{1,3,4,6\}}\,d_{\{1,3,4,6\}} + \\ & a_{\{3,5,6\}}\,c_{\{1,3,5,6\}}\,\overline{d}_{\{2,5,6\}} + a_{\{3,5,6\}}\,c_{\{1,3,5,6\}}\,\overline{b}_{\{2,5,,6\}} + \\ & a_{\{3,5,6\}}\,\overline{d}_{\{1,3,5,6\}}\,\overline{d}_{\{2,5,6\}} + a_{\{3,5,6\}}\,\overline{d}_{\{1,3,5,6\}}\,\overline{b}_{\{2,5,6\}} \end{aligned}$$

There is a one to one correspondence between paths in the multilevel circuit, paths in the leaf-DAG, and distinct tagged literals in the ENF. Corresponding to a path π is a tagged literal l_π. l is a literal of the input variable of π and its tag is the set of gates along π. For example, consider the path from b through gates 4 and 6 to M. The tagged literal $b_{\{4,6\}}$ represents this path in the ENF. Thus, the ENF simultaneously captures information about the function and paths of the original multilevel circuit.

There is a two-level circuit that corresponds to the ENF and which can be directly obtained by a direct translation of the ENF to an **andor** circuit. The ENF-two-level circuit for the example in Figures 2 and 3 is shown in Figure 4. While it is obvious that for the ENF-two-level circuit is logically equivalent to the original multilevel circuit, it is also temporally equivalent for purposes of delay computation. For this we need to specify the delay values in the two-level circuit. The delays of paths in the original multilevel circuit can be reflected by adding all the delay to the first net in the two-level circuit and assuming all subsequent circuit elements have zero delay.

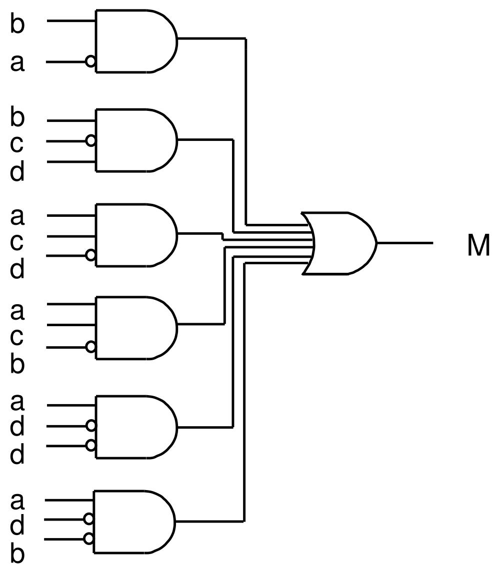

Figure 4: A Two-Level Circuit Representing the ENF

Consider a vector v, such that $C(v) = 1$, applied to the primary inputs of a circuit. In the floating mode, all nodes in the circuit are assumed to be in an unknown state when v is applied. Correspondingly in the ENF, all the literals have unknown values when v is applied. v will cause certain events to propagate in the circuit and finally the output will stabilize to a 1. Correspondingly, in the ENF, different cubes will settle to their final values at different times, and the output settles to a 1 as soon as the first cube settles to a 1. Thus, the delay for the rising transition is determined by the time taken for the first cube to settle to a 1.

6 Delay Analysis using the ENF

6.1 Static Co-sensitization of Paths

The delay of a circuit is the maximum time taken by the last possible transition at a primary output of the circuit. We consider the rising delay (delay of a 0 to 1 transition) and falling delay separately. The rising delay is considered first. The falling delay is analyzed by considering the rising delay of the complemented circuit.

In Section 5, we saw that the rising delay is determined by the time taken for the first cube to settle to a 1. This simple observation leads us to the necessary conditions for a path to be true.

Let π be a path that we wish to check to be true or false. For π to be true, there must be some cube q containing the tagged literal l_π that evaluates to 1 for some vector v. If this were not true then it would not be possible to transmit a 1 along π. This is formally stated in the following theorem.

Theorem 6.1 *Let C be a combinational circuit with ENF expression E. If π is true for the rising transition then there exists a vector v and there exists a cube q such that $l_\pi \in q$ and $l_\pi(v) = q(v) = C(v) = 1$.*

Proof: Suppose there does not exist a cube q in E such that $l_\pi \in q$ and $l_\pi(v) = q(v) = C(v) = 1$. Then, for each v and cube q, s.t. $l_\pi(v) = 1$ and $l_\pi \in q$, we have $q(v) = 0$. Since we are considering the output rising to a 1, we need concern ourselves with only $C(v) = 1$. Since, $q(v) = 0$, $C(v) = 1$ due to cubes, q_i, $(q_i(v) = 1)$, that do not contain l_π. Thus, l_π is not true for the rising delay of C. ■

If v satisfies the condition in this theorem for path π then v is said to **statically co-sensitize** π to a 1. Topologically static co-sensitization is explained as follows.

Let $\pi = \{v_0, e_0, ..., v_n, e_n, v_{n+1}\}$ be a path. An input vector v **statically co-sensitizes to a 1(0)** path π in C iff the value of v_{n+1} is 1(0) and for each v_i, $0 \leq i \leq n+1$, if v_i has a controlled value then the edge e_{i-1} presents a controlling value. In comparison, the more common condition of **static sensitization** requires that e_{i-1} present the only controlling value.

Thus we see that static co-sensitization is a necessary condition for a path to be true. Note that static co-sensitization is a purely logical condition and does not depend on the delay values of the various circuit components.

6.1.1 Static Co-Sensitization and Viability

In [12] a condition termed **viability** is presented as being sufficient for a path to be true. We now proceed to relate static co-sensitization to the relevant problem of viability of paths in a circuit. Consider a path π passing through a gate g in the network. Let e_g be the edge along π that is an input to g. The inputs of g other than e_g are termed the side inputs of g for π. In analyzing the viability of a path π on a vector v the side-inputs are divided into two groups:

1. The side-inputs that settle to their final value before e_g does. These are referred to as the early side-inputs.

2. The side-inputs that settle to the final values no earlier than e_g. These are referred to as the late side-inputs.

For a path to be viable the following conditions must be true:

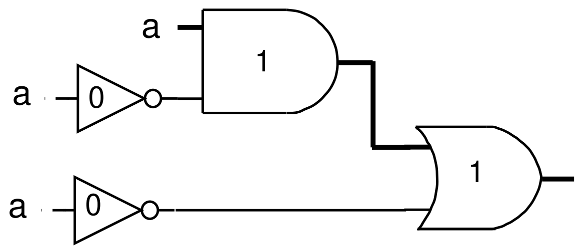

Figure 5: A Circuit with a Path that is Viable but not Statically Co-sensitizable

- All side-inputs that arrive early must present a non-controlling value at g.
- The values on the late side-inputs do not matter. The time taken for the side-inputs to settle to their final values is determined recursively by this definition.

It is now shown that viability does not imply static co-sensitization, so a path may be classified as being viable even though it may not be statically co-sensitizable and hence false [3]. When timing analysis is used in performance optimization to identify paths that must be speeded up, this incorrect analysis may result in resources being wasted to speed up this path. In particular in Figure 5 we give an example of a path that is determined, using the given delays in the figure, to be viable, but which is not statically co-sensitizable. In the figure, integers assigned to each gate represent the delay of the gate. The path in consideration is the one shown in bold from the non-inverted input of the **and** gate to the primary output.

This path is viable for $a = 1$. The side-input to the **and** gate is late, and the side-input to the **or** gate is non-controlling. However, this path is not statically co-sensitizable. This can be verified by using either the ENF or by the topological conditions for static co-sensitization.

First consider the ENF, E. $E = a_1\,\overline{a_2} + \overline{a_3}$. The tags $1, 2, 3$ on the literals a a indicate the three paths in the circuit. Since there is no cube $q \in E$, $a_1 \in q$, for which $a_1(v) = 1 = q(v) = E(v)$, the path corresponding to a_1 is not statically co-sensitizable to a 1. Consider the complemented ENF, $\overline{E} = \overline{a_1}\ a_3 + a_2 a_3$. Again, there is no cube $q \in E$, $\overline{a_1} \in q$, for which $\overline{a_1}(v) = 1 = q(v) = \overline{E}(v)$, the path corresponding to a_1 is not statically co-sensitizable to a 0.

Let us now examine what it means for a path not to be statically co-sensitizability in terms of the topology of the circuit. For $a = 1$, the side input at the **and** gate presents a controlling value, while the value along the path is non-controlling. Similarly for $a = 0$, the side input at the **or** gate

[3] The fact that a path may be classified as being true even when it is false has been previously demonstrated by Chen and Du [5]. The purpose of this section is to show that the specific condition that viability does not consider is nothing but static co-sensitization.

is controlling while the value along the path is non-controlling. Thus, in either case the path is not statically co-sensitizable.

It should be noted that while viability may incorrectly identify some paths as being true the final delay value returned by it is correct.

6.2 Necessary and Sufficient Conditions

Static co-sensitization just ensures that there is some vector v and some cube q containing l_π such that q evaluates to 1 for v. However, this is not enough. Not only must q evaluate to a 1 but it must be the first cube to evaluate to 1. If this were not so, *i.e.* some other cube evaluates to a 1 faster than q for each v satisfying Theorem 6.1 then by the time q evaluates to 1 the output has already settled to its final value and q cannot affect the transition. In addition, for l_π to affect the transition in q it must be the last literal in q to evaluate to a 1. This is formally stated in the following theorem.

Theorem 6.2 *Let C be a circuit with ENF E and let π be a path in C. If there exists a vector v such that*

1. *there exists a cube $q \in E$ that contains l_π such that $q(v) = 1$, $C(v) = 1$ and*
2. *there does not exist any cube $r \in E$ such that $r(v) = 1$ and $\forall m_\rho \in r$, $length(\rho) < length(\pi)$,*
3. *and finally $\forall n_\sigma \in q \neq l_\pi$, $length(\sigma) \leq length(\pi)$*

then π is true for the rising transition.

Proof: Condition 1 has already been shown to be necessary by Theorem 6.1. Thus, we just need to show that Conditions 2 and 3 listed above are necessary.

Suppose Condition 2 is not satisfied, i.e. for each v such that $q(v) = 1$, $l_\pi \in q(v)$, there is some cube r such that $r(v) = 1$ and $\forall m_\rho \in r$, $length(\rho) < length(\pi)$. Then for each literal in r, the corresponding path is shorter than π. For input v, r and thus C, will rise to a 1 before π can propagate the 1 to the output. Thus, Condition 2 is necessary.

Suppose Condition 3 is not satisfied, i.e. for each v such that $q(v) = 1$, $l_\pi \in q$, there is some tagged literal $n_\sigma \in q$ (for each q) such that $length(\sigma) > length(\pi)$. Then this cube will settle to a 1 only when n_σ settles to a 1, which will be after l_π settles to a 1. Thus, l_π will never be responsible for getting the 1 to the output of the circuit and hence is false for the rising delay. Thus Condition 3 is necessary. ∎

The topological interpretation of this theorem is as follows. Let g be a gate along π and let v, the vector satisfying the conditions of Theorem 6.2 be applied to the primary inputs. If π has a controlling value then it must be the first of all the controlling values arriving at g. If π has a non-controlling value not only must all the other side inputs have non-controlling values on them (imposed by static co-sensitization), the non-controlling value along π must be the last to arrive at g. This topological interpretation has been used previously as the starting point for the delay computation algorithm presented in [5]. However, as we will see in the next section, analysis of the ENF enables us to strengthen this observation by extending this result to entire sets of paths.

While Theorem 6.2 states the necessary conditions for π to be true, any v satisfying those conditions actually sensitizes π and thus these conditions are sufficient too. Thus, Theorem 6.2 can be strengthened to an if and only if form.

Theorem 6.3 *Let C be a circuit with ENF E and let π be a path in C. π is true for the rising transition if and only if there exists a vector v such that*

1. *there exists a cube $q \in E$ that contains l_π such that $q(v) = 1$, $C(v) = 1$ and*
2. *there does not exist any cube $r \in E$ such that $r(v) = 1$ and $\forall m_\rho \in r$, $length(\rho) < length(\pi)$,*
3. *and finally $\forall n_\sigma \in q \neq l_\pi$, $length(\sigma) \leq length(\pi)$.*

Proof: The necessity of these conditions has been demonstrated by Theorem 6.2. We just need to show their sufficiency here.

Let v be a vector that satisfies the conditions stated in the theorem. Then by Condition 2, no cube that does not contain l_π will settle to a 1 before q. Further, by Condition 3, q will settle to a 1 only when l_π settles to a 1, since all other literals in q settle to a 1 no later than l_π. Thus, for v, l_π is the last literal to settle to a 1 in the first cube that settles to a 1. Alternatively, v sensitizes π. ■

6.3 Handling Sets of Paths

The analysis presented in the previous section can be easily extended to sets of paths. In particular, consider the set of paths $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ that contain all paths of length at least δ. We wish to determine the necessary and sufficient conditions for there being at least one true path in this set. The results of the previous section can be extended to the following.

Theorem 6.4 *Let C be a circuit with ENF E and let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ be the set of all paths of length at least δ. At least one $\pi_i \in \Pi$ is true for the rising transition if and only if there exists a vector v, such that $C(v) = 1$ and for all q for which $q(v) = 1$, there exists some l_{π_i} satisfying $l_{\pi_i} \in q$, $\pi_i \in \Pi$.*

Proof:

Necessity: Suppose that the condition in the theorem statement is not true, i.e. for each vector v such that $C(v) = 1$, there exists some cube q such that $q(v) = 1$ and q does not contain any tagged literal from Π. Then for each tagged literal $m_\rho \in q$, $length(\rho) < \delta$ or else ρ would have been in Π. Thus, q rises to a 1 before time δ and hence no path in Π is true.

Sufficiency: Suppose that the condition in the theorem statement is true, i.e. there exists a a vector v, such that $C(v) = 1$ and for all q for which $q(v) = 1$, there exists some l_{π_i} satisfying $l_{\pi_i} \in q$, $\pi_i \in \Pi$. Among all cubes q_j for which $q_j(v) = 1$ let q be the one that is the first to settle to a 1. Let l_{π_i} be the tagged literal satisfying the condition in the theorem statement, i.e. $l_{\pi_i} \in q$, $\pi_i \in \Pi$. Let l_{π_j} be the longest tagged literal in q, i.e. $length(\pi_j) \geq length(\sigma)$, $n_\sigma \in q$, $l_{\pi_j} \neq n_\sigma$. Now π_j is true for the rising delay since on v, l_{π_j} is the last literal to settle to a one in the first cube that settles to a 1. Since $length(\pi_j) \geq length(\pi_i)$, $\pi_j \in \Pi$. ■

In other words, for some input vector v, all the cubes that evaluate to 1 must belong to the path cube complexes of the paths in Π. (The path cube complex of l_π is the set of all cubes containing the literal l_π.) This is equivalent to saying that v is a relatively essential vertex of the path cube complexes of the paths in Π, *i.e.* it is covered by no other cubes. This immediately leads to the following equivalent result expressed in terms of stuck-at-faults on the tagged literals $\{l_{\pi_i}\}$.

Theorem 6.5 *Let C be a circuit with ENF E and let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ be the set of all paths of length at least δ. At least one $\pi_i \in \Pi$ is true for the rising transition if and only if there is a test for the multifault $\{l_{\pi_i}$stuck-at-0$\}$.*

Proof: Let $\tilde{E}$ be the ENF obtained from E by setting l_{π_i} to 0, for all $\pi_i \in \Pi$, i.e. effecting the multifault $\{l_{\pi_i}$stuck-at-0$\}$. Thus, $\tilde{E}$ has no cubes containing any l_{π_i}. A test for this multifault is a vector w such that $E(w) \neq \tilde{E}(w)$. Note that since $\tilde{E}$ has been obtained by strictly deleting cubes from E; for any test w, $E(w) = 1$ and $\tilde{E}(w) = 0$.

Necessity: Suppose no test exists for the multifault. Then for each vector w such that $E(w) = 1$, $\tilde{E}(w) = 1$. Thus, w is covered by some cube not containing any l_π. For w, this cube rises to a 1 before time δ and thus no path in Π is true for the rising delay.

Sufficiency: Suppose w is a test for the multifault. Then, $E(w) = 1$ and $\tilde{E}(w) = 0$. Thus, w is covered by some cube containing an l_{π_i} and by no cube that does not contain an l_{π_i}. By Theorem 6.4, at least one path in Π is true for the rising delay. ■

This theorem forms the basis for a delay analysis algorithm that is presented in the next section.

7 Algorithmic Aspects

The theoretical results derived in the previous section are now used to develop an algorithm for delay analysis. We show how the problem of delay analysis can be mapped to one of stuck-at-fault testing (with a modification). Thus, the large body of work done in stuck-at-fault testing can be applied towards the delay analysis problem.

As before, we first examine the problem of determining the rising delay of the circuit, the falling delay can be determined by a simple change at the first step of the algorithm. The delay of the circuit is then determined as the maximum of the rising and the falling delay.

The algorithm works by answering one or more questions of the form: "Is the delay of the circuit $\geq \delta$?" where δ is some positive number. δ may be chosen by a binary search among the possible values or by examining the distinct path lengths sorted in decreasing order.

We now consider the resolution of the above question for the rising delay first. ¿From Theorem 6.5 we note that this question can be directly answered if we are given the leaf-DAG for the circuit. Now the multifault on the tagged literals l_{π_i} is the multifault with the first edge of each π_i stuck-at-0. As shown in Figure 6 the first edge refers to the edge after any possible inverters at the primary inputs. This fault may then be tested using classic testing strategies. Using the classic *D-calculus* notation [13], the test generation algorithm will try and find an input vector for which the stuck-at-0 edges have either a 0 or a D on them and a D is propagated to a primary output.

7.1 Timed Test Generation

In general the leaf-DAG's are not available and are not easily constructed. Unfolding a circuit to generate the leaf-DAG will often lead to an exponential blow-up since the size of the leaf-DAG is proportional to the number of paths in the circuit which is often exponentially related to the size of the circuit. Fortunately it is possible to determine the test for the multifault by directly working on the original circuit C by considering timing information during test generation. We call this procedure **timed test generation** and the use of this procedure to compute a test for a multifault is the major

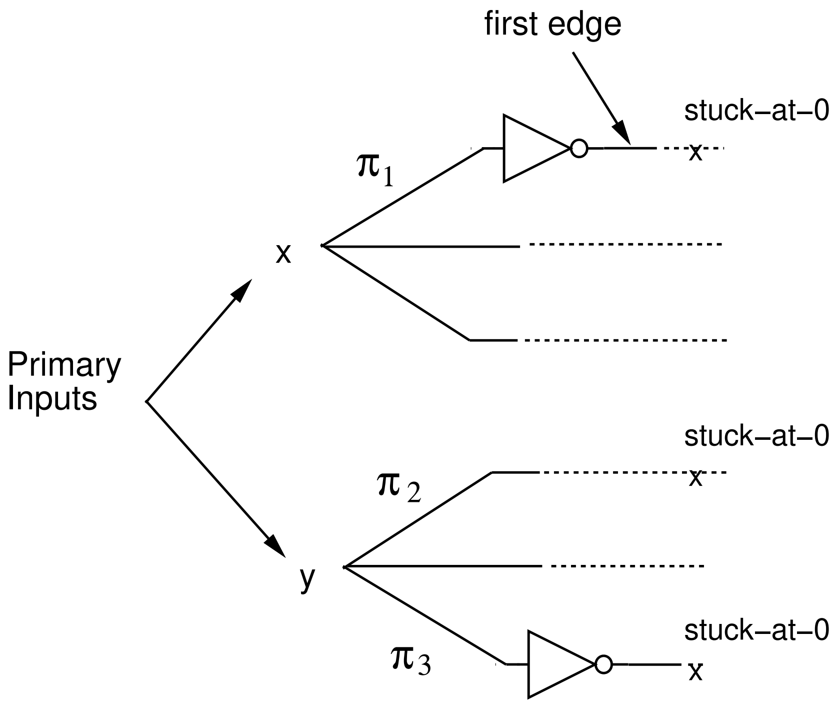

Figure 6: A multifault of s-a-0's in a leaf-DAG

contribution of this section. For ease of exposition we will demonstrate this on a modified circuit C' obtained from the original circuit C as follows: C' is obtained from C by migrating all inverters in C to the primary inputs. The inverter at the output of a gate may be moved to the inputs by using DeMorgan's laws of complementation. Thus, all the inverters may be moved to the primary inputs by starting at the primary outputs and recursively applying this procedure to all gates in the circuit. As shown in Figure 7, a gate that is used in both inverting and non-inverting phase may need to be duplicated. Since each gate is duplicated no more than once, C' is at most twice the size of C. Note that the sensitization conditions on paths in C' are the same as those for the corresponding paths in C by an argument similar to that given for leaf-DAG's. We would like to reiterate that C' is being used only for purpose of exposition and as will be shown later, timed test generation may be done directly on C.

Our objective is to test the multifault directly on C'. We accomplish this by restricting the fault propagation to paths in Π by considering timing information during the fault propagation. Surprisingly very little timing information is needed to do this. For each edge i in C' we just need to determine the following quantity: $max^t(e_i)$, which is the length of the longest path starting at the head of edge i and ending at some sink of the circuit DAG, *i.e.* at some primary output.

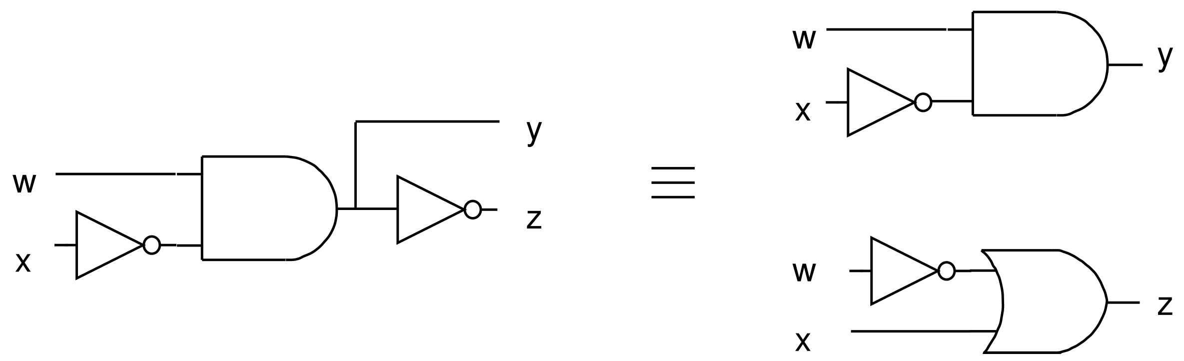

Figure 7: Pushing inverters to the primary inputs

$max^t(e_i)$ is determined as follows. A dummy sink is added with an edge to it from each primary output. $max^t(e_i)$ is determined by a depth-first search from edge i to this sink. Thus this step is $O(|V| + |E|)$.

To test the multifault in question each fanout edge, e_i, of a primary input for which $max^t(e_i) \geq \delta$ may have a D or a 0 on that edge. As in Figure 6 the D is placed after any inverters present at the inputs. For each such e_i we know that there is a path starting at e_i along which the D may propagate to the output. The goal of test generation is to propagate a D to the output of the circuit. For each D that we are trying to propagate forward through the circuit there is an associated value $D.s$, which captures the timing information associated with the error value. The semantics of $D.s$ will be described shortly, at this point it is sufficient to note that for each D placed on the fanout edges of the primary inputs, $D.s = 0$.

There is a difference in the testing of this multifault and classical stuck-at-fault testing in terms of how a fault effect is propagated through a gate. We now examine this difference.

7.1.1 Timed D Calculus

Evaluation of the output of a gate, g, given its inputs, is done in a manner similar to the standard D-calculus with two differences. The first is that the evaluation for each fanout edge, e_i, of g is done separately. Thus, the different fanout edges may evaluate to different values. The second is that the timing information of each error value E (D or $\overline{D}$), *i.e.* $E.s$ is taken into account while evaluating the value for e_i.

The following cases describe the evaluation of the value on e_i, given the values on the inputs to g for some input vector v.

1. There is a controlling value at the input of g: This forces e_i to the corresponding controlled

value.

2. Each input of g has a non-controlling value on it: e_i takes on the non-controlled value.

3. Some set of inputs have an error value on them, some (possibly none) have non-controlling values: Note that both D and $\overline{D}$ cannot be present on the inputs of g since we only propagate either D's or $\overline{D}$'s at a time, but not both. Let E_k be the error values on the inputs and $d(g)$ be the delay of gate g. Let s be as given by the following table:

E	gate	s
D	**or**	$min(E_k.s)$
D	**and**	$max(E_k.s)$
$\overline{D}$	**or**	$max(E_k.s)$
$\overline{D}$	**and**	$min(E_k.s)$

   (a) If $s + d(g) + max^t(e_i) \geq \delta$, then e_i evaluates to the error value E with $E.s = s$.

   (b) If $s + d(g) + max^t(e_i) < \delta$, then e_i evaluates to the error free value, *i.e.*, 1 for a D and 0 for a $\overline{D}$.

The following results provide the reassurance that the timing information included in the evaluation is sufficient to ensure that the timed D calculus applied to C' is equivalent to the standard D calculus applied to the leaf-DAG.

Lemma 7.1 *Let e_i be an edge in C' and $e_{i,1}, e_{i,2}, \ldots, e_{i,k}$ be edges corresponding to e_i in the leaf-DAG. For input vector v, the timed-D calculus results in an error value, E, with time value $E.s$, on an edge e_i in C' if and only if the standard D calculus results in an error value on each edge, $e_{i,l}$ in the leaf-DAG, for which $E.s + max^t(e_{i,l}) \geq \delta$.*

Proof: The proof is by induction on the depth of the edges. The depth of an edge is the maximum number of gates along any path from this edge to a primary input. A few comments on the notation used are in order. A single subscript used with an edge (e.g. e_i) indicates an edge in C'. A double subscript (e.g. $e_{i,k}$) indicates an edge in the leaf-DAG. The time associated with an error value on edge e_i is denoted by $s(e_i)$.

Induction Basis: Consider an edge, e_i, of depth 0. e_i is the fanout of a primary input. If e_i has an error value on it then $E.s = 0$. We know that e_i has an error value on it if and only if each edge, $e_{i,l}$ such that $max^t(e_{i,l}) \geq \delta$ has an error value on it.

Induction Step: Assume the statement in the theorem to be true for depth less than n. We need to prove it for depth n. Let e_i be an edge of depth n. e_i is a fanout of gate g_i. Let e_j and e_m be fanins of g_i.

We consider four separate cases depending on the gate type and the error value.

Gate Type: and; Error Value: D : Note that an error value on e_i in C' represents an error value on some edges $e_{i,1}, e_{i,2}, \ldots, e_{i,l}$ and an error free value on edges $e_{i,l+1}, \ldots, e_{i,p}$. In the leaf-DAG let $e_{j,k}$ be the edge corresponding to e_j in C', that connects through g_i to $e_{i,k}$.

If part: Let $e_{i,k}$ have a D on it. Then, some $e_{j,k}$ must have a D on it and the other inputs do not have controlling values. Then by the induction hypothesis there is some edge e_j with a D on it and time value $s(e_j) \geq \delta - max^t(e_{j,k})$. Thus, e_i has a D on it with time value $s(e_i)$ such that $s(e_i) \geq s(e_j) + d(g_i)$. Rewriting this we get $s(e_i) \geq \delta - max^t(e_{j,k}) + d(g_i)$. Since $max^t(e_{j,k}) - d(g_i) = max^t(e_{i,k})$, we get $s(e_i) + max^t(e_{i,k}) \geq \delta$.

Only If part: Let e_i have a D with $s(e_i)$ denoting the time value of this D. Let e_j be the input of g_i such that the D on it had the maximum time value among all the D's at the inputs of gate g_i. Let this time value be $s(e_j)$. Then we know that $s(e_i) = s(e_j) + d(g_i)$. Let $e_{i,k}$ be an edge in the leaf DAG corresponding to e_i such that $s(e_i) + max^t(e_{i,k}) \geq \delta$. Rewriting this we get $s(e_j) + d(g_i) + max^t(e_{i,k}) \geq \delta$ or equivalently $s(e_j) + max^t(e_{j,k}) \geq \delta$. Thus, by the induction hypothesis there must be a D on edge $e_{j,k}$ in the leaf-DAG. The other inputs to gate i in the leaf-DAG cannot have controlling values on them since the error free value for a D is 1 which is non-controlling for an **and** gate. Thus, with a D on $e_{j,k}$ and no controlling values on any other inputs of the gate, a D is propagated to $e_{i,k}$ in the leaf-DAG.

Gate Type: or; Error Value: D :

If part: Let $e_{i,k}$ have a D on it. Then, some $e_{j,k}$ must have a D on it and the other inputs do not have controlling values. Then by the induction hypothesis there is some edge e_j with a D on it and time value $s(e_j) \geq \delta - max^t(e_{j,k})$. If there is any other input, e_m, of g_i in C' that has a D on it then $s(e_m) \geq \delta - max^t(e_{m,k})$. If this were not true then $e_{m,k}$ would have an error free value of 1 on it which for an **or** gate would be controlling and then $e_{i,k}$ would not have a D on it. Since $max^t(e_{m,k}) = max^t(e_{j,k})$, we see that $s(e_i) \geq \delta - max^t(e_{j,k}) + d(g_i)$ which gives us $s(e_i) + max^t(e_{i,k}) \geq \delta$.

Only If part: Let e_i have a D with $s(e_i)$ denoting the time value of this D. Let e_j be the input of g_i such that the D on it had the minimum time value among all the D's at the inputs of gate g_i. Let this time value be $s(e_j)$. Then we know that $s(e_i) = s(e_j) + d(g_i)$. Let $e_{i,k}$ be an edge in the leaf DAG corresponding to e_i such that $s(e_i) + max^t(e_{i,k}) \geq \delta$. Rewriting this we get $s(e_j) + d(g_i) + max^t(e_{i,k}) \geq \delta$ or equivalently $s(e_j) + max^t(e_{j,k}) \geq \delta$. Thus, by the induction hypothesis there must be a D on edge $e_{j,k}$ in the leaf-DAG. The other inputs to gate i in the leaf-DAG cannot have controlling values on them since any other input e_m to g_i in C' that had a D on it will also force a D on $e_{m,k}$ since $s(e_j) \leq s(e_m)$. Thus, with a D on $e_{j,k}$ and no controlling values on any other inputs of the gate, a D is propagated to $e_{i,k}$ in the leaf-DAG.

Gate Type: and; Error Value: $\overline{D}$: Similar to gate type **or** and error value D.

Gate Type: or; Error Value: $\overline{D}$: Similar to gate type **and** and error value D.

■

Theorem 7.1 *For an input vector v an error value propagates to the primary output in C' under the timed-D calculus if an only if it propagates the same error value to the primary output in the leaf-DAG under the standard D calculus.*

Proof: The proof follows directly by applying Lemma 7.1 to the fanout edge of the primary output. Let e_i be the fanout edge of the primary output in C'. Then if there is an error value on e_i, $s(e_i) \geq \delta$. This ensures that the fanout edge of the primary output in the leaf-DAG has an error value on it. Conversely, let there be an error value on the primary output in the leaf-DAG. Then by Lemma 7.1 there must be an error value on e_i with $s(e_i) \geq \delta$. ■

7.2 Test Pattern Generation

The previous section described the simulation semantics of timed test generation, i.e. it specified how the outputs of a gate are determined given the inputs. If we look at the historical evolution of traditional test pattern generation for stuck-at-faults, we observe that first the simulation semantics we described for the D-calculus, then this was used to search the space of input vectors to find a vector that would generate an error value in the circuit and propagate it to a primary output of the circuit. Exactly the same paradigm is followed here for timed test generation. No details are provided here. These are reported in a companion paper [7].

7.2.1 Bounded Justification

It is interesting to note that the timing information also helps in pruning the search space during justification in the process of test pattern generation. Analogous to $max^t(e_i)$, we define $max^s(e_i)$ to be the length of the longest path from edge e_i to a source, i.e. a primary input of the circuit. Like max^t, the computation for determining max^s is done in a single traversal of the circuit graph. For an error value, E, to propagate across a gate g, the other side-inputs to g must either have non-controlling values on them, or the same error value E. Let k vary over the inputs of g, and i vary over the outputs of g. E will only be propagated to the output of g if $min_k(E.s) + d(g) + max_i(max^t(e_i)) \geq \delta$ or equivalently $min_k(E.s) \geq \delta - max_i(max^t(e_i)) - d(g)$. This puts a lower bound on the time value, $E.s$, of any E on the input of the gate. Let e_k be an input edge of gate g. If $max^s(e_k) < \delta - (max_i(max^t(e_i))) - d(g)$, then an appropriate error value can never be obtained at input edge e_k. If this is not the case then it might be possible to propagate the error from some first edge to e_k. However, we still need to justify E on edge e_k. To prune the backward search for E during justification, the lower bound on the timing value of E is stored along with E as $E.b$. Thus, for e_k, $E.b = \delta - (max_i(max^t(e_i))) - d(g)$. Let g_k be the input gate of edge e_k. Then, for an input edge e_j of g_k to provide E on e_k the following condition must be true: $E.b - d(g_k) \leq max^s(e_j)$. Thus we see that these bounds prune the search for error values during justification.

It should be noted that the exact procedure used for justification is different for different testing algorithms. For example, in PODEM [9] justification is only done for fault excitation. Since the multifault in question is at the primary input fanout edges, justification is trivial for that case. For such an algorithm where justification of error values is not used during fault propagation, $max^s(e_i)$ need not be computed.

7.2.2 Avoiding Duplication of Gates

We now look at timed test generation directly on C without needing to generate C'. With each edge e_i in C we need to store two numbers instead of one. We store $max^t_+(e_i)$ and $max^t_-(e_i)$ which are the length of the longest path with an even and an odd parity of inversions from the head of e_i to a primary output respectively. If there is no path with a given inversion of parities then the corresponding quantity is $-\infty$. Again the computation of these quantities is done using a simple depth-first search. These quantities capture both the path length information as well as the parity of inversions seen along the paths. To start with, if we are testing a multifault of s-a-0's on the first edges, we may place a D on the primary output fanout edge e_i if $max^t_+(e_i) \geq \delta$ and a $\overline{D}$ if $max^t_-(e_i) \geq \delta$. Note that both a D and a $\overline{D}$ may be placed on the same edge. Now the fault propagation proceeds as discussed above with $max^t_+(e_i)$ being used for propagating a D and $max^t_-(e_i)$ for propagating a $\overline{D}$. Similarly, $max^s_+(e_i)$ is used for justifying a D and $max^s_-(e_i)$ for justifying a $\overline{D}$.

7.2.3 Computing the Falling Delay

The falling delay can be analyzed by considering the rising delay of the circuit obtained by adding a zero-delay inverter at the primary output of the original inverter. In terms of the multifault this is equivalent to testing a multifault of s-a-1 faults on the first edges of the paths in Π in the original circuit. The circuit delay is the maximum of the rising and falling delays.

7.3 Circuit and Delay Models: Practical Considerations

We now address some practical concerns that arise in the modeling of delays in the circuits. Thus far in this paper we made the following simplifying assumptions: we considered circuits of only simple gates and we restricted the delays to one number per gate. We are now in a position to state that these restrictions are in fact not necessary.

The delay computation algorithm presented in this section was developed in two stages. First the simulation semantics for the timed-D calculus was developed. These semantics were a combination of timing simulation and fault simulation semantics. The former determined the delay of an error value as it progressed through the circuit, the latter controlled the propagation of the error values. Next these simulation semantics were used to search for an input vector that would result in an appropriate error value at the primary outputs. In order for the timed-D calculus to directly handle gates and delay models of arbitrary complexity, all we need to do is define the timing and fault simulation semantics of the gate. This is not a problem, since these are typically available with any library gate [4]

Not only do the library simulation models help solve this problem, they are absolutely necessary to determine the behavior of complex gates as is illustrated by the following example. Consider the circuit in Figure 8. Here the delay for the buffer after input x is 100 and all other gate delays are 0.

[4] McGeer and Brayton [12] discuss the issue of complex gates in the context of viability. However only a conservative handling of the situation, using what they term as a *symmetric macro-expansion*, is provided. This section demonstrates the need for and the sufficiency of a library simulation model for a complex gate.

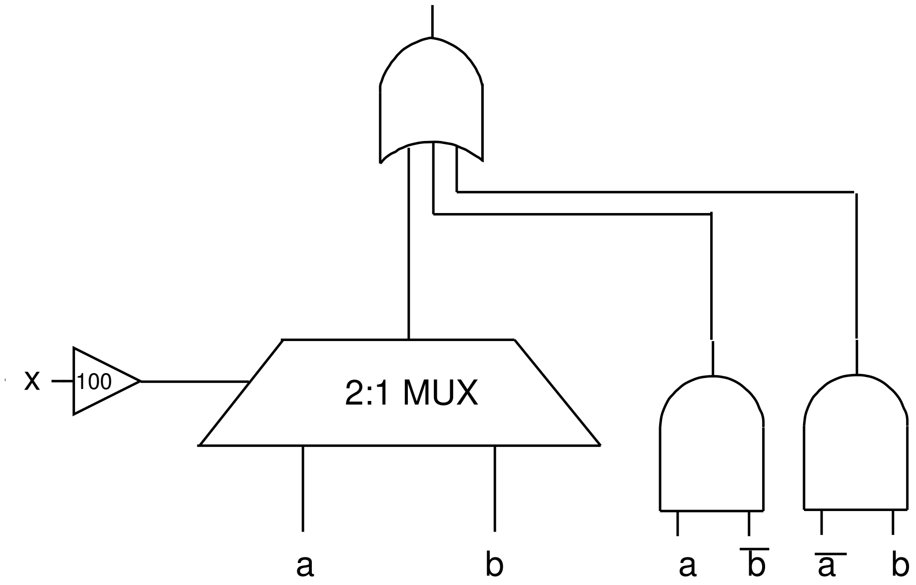

Figure 8: An Example with Complex Gates

It is not possible to accurately determine the delay of the circuit without knowing the behavior of the multiplexor. Any arbitrary expansion of the multiplexor in terms of simple gates is not sufficient. For example, consider the two expansions of the multiplexor shown in Figure 9. a and b are the two data inputs and x is the control input to the mux. In the former expansion, the rising delay is dependent on the time at which x is ready, even when a and b are equal. In the latter, when a and b are equal, the rising delay does not depend on the time x is ready. With the former expansion the rising delay of the circuit in Figure 8 is 100 while with the latter it is 0!

7.4 Comparison with Previous Work

The techniques previously described in the literature for delay analysis such as those presented in [12, 5] determine the condition for a path to be true. This reduces to a satisfiability problem where a satisfying input assignment needs to be determined that will make that condition true. It may be argued that the timed test generation technique described here is also solving a satisfiability problem, so it is not clear if this is not a restatement of the previously presented solutions.

This brings us to the fundamental difference between timed test generation and the path based techniques. These techniques operate on *one path at a time*, while timed test generation works on all paths of length $\geq \delta$ at the same time. Working on a path at a time is a critical deficiency for circuits such as multipliers which have a very large number of long paths that need to be examined

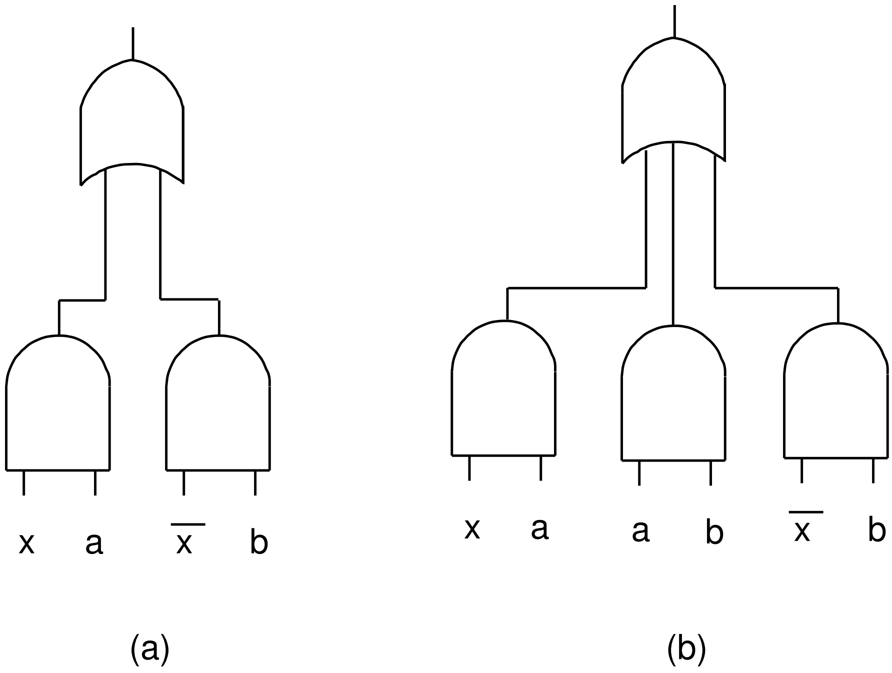

Figure 9: Alternate Expansions of the Multiplexor

before one of them can be ratified as being responsible for the delay.

Another point to be noted here is that no assumption is made about the number of distinct path lengths in the circuit. All the paths of length greater than or equal to δ are implicitly considered, regardless of their number or individual delays.

8 Preliminary Results

In this section we show the results from applying the delay computation algorithm to some particularly troublesome examples. The examples adder16x2 and adder16x4 are 16-bit carry bypass adders with 2 and 4 bits in the bypass chain, respectively. The examples mult8x8 and mult16x16 are carry-propagate parallel 8×8 and 16×16 multipliers, respectively. C6288 is an optimized version of the 16×16 multiplier from the ISCAS-85 combinational logic benchmark suite. The remaining two examples are random logic benchmarks from the MCNC suite.

We are able to run these troublesome examples within reasonable CPU times. The multiplier examples take over 20 hours of CPU time when run on a path by path basis.

The CPU times reported for our implementation were on a SUN-4 320 workstation. More comprehensive experimental results are reported in [7].

Name	Delay Estimate		CPU Time
	Longest	True	
adder16x2	25.0	12.0	96s
adder16x4	21.0	12.0	132s
mult8x8	44.0	44.0	6s
mult16x16	88.0	88.0	23s
C6288	94.0	94.0	18.9m
5xp1	11.0	9.0	3s
bw	29.0	25.0	10s

Table 1: Timing analysis using Timed-Test Generation

9 Concluding Remarks

Due to the ease of generating false paths in high-level synthesis systems [2] there is a growing need for correctly identifying false paths to guide performance optimization, in a computationally efficient manner. In this paper we provided necessary and sufficient conditions for a path to be true in the floating mode of operation. In particular static co-sensitization has been introduced as a necessary condition. Our results are then extended to determine the truth or falsity of entire sets of paths simultaneously by expressing them in terms of the testability of a multifault in an ENF expression. The second part of the paper is devoted to applying this result directly to an *unmodified multilevel circuit.* Because the circuits that are most troublesome for false-path-eliminating static timing analyzers are those with literally millions of paths, and in particular millions of longest paths, the ability to handle entire sets of paths simultaneously results in a very efficient delay computation procedure. This is demonstrated by the results from a preliminary implementation of the algorithm. Based on these results we are confident that we can meet the growing need for a computationally efficient correct delay-computation procedure.

10 Acknowledgments

Srinivas Devadas is supported in part by the Defense Advanced Research Projects Agency under contract N00014-91-J-1698. Sharad Malik is supported in part by an IBM Faculty Development

Award.

References

[1] D. B. Armstrong. On Finding a Nearly Minimal Set of Fault Detection Tests for Combination Logic Nets. *IEEE Transactions on Computers*, EC-15(2):66–73, February 1966.

[2] R. Bergamaschi. The Effects of False Paths in High-Level Synthesis . In *The Proceedings of the International Conference on Computer-Aided Design*, November 1991.

[3] D. Brand and V. Iyengar. Timing analysis using functional analysis. *IEEE Transactions on Computers*, 37(10), October 1988.

[4] M. A. Breuer and A. D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science Press, 1976.

[5] H. C. Chen and D. H. Du. Path Sensitization in Critical Path Problem. In *Proceedings, Tau 90: 1990 ACM Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, August 1990.

[6] J. J. Cherry. PEARL: A CMOS timing analyzer. In *The Proceedings of the Design Automation Conference*, 1988.

[7] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Computation of Floating Mode Delay in Logic Circuits: Practice and Implementation. In *MIT Technical Report, Massachusetts Institute of Technology*, October 1991.

[8] H. Fujiwara. *Logic Testing and Design for Testability.* MIT Press, Cambridge MA, 1985.

[9] P. Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, C30(3):215–222, March 1981.

[10] R. B. Hitchcock. Timing verification and the timing analysis program. In *The Proceedings of the Design Automation Conference*, 1982.

[11] V. Hrapcenko. Depth and delay in a network. *Soviet Math. Dokl.*, 19(4), 1978.

[12] P. C. McGeer and R. K. Brayton. *Integrating Functional and Temporal Domains in Logic Design.* Kluwer Academic Publishers, 1991.

[13] J. P. Roth. Diagnosis of Automata Failures: A Calculus and a Method. *IBM Journal of Research and Development*, 10:278–291, July 1966.