The Critical Path Problem in Timing Verification

Luís Miguel Silveira

Research Laboratory of Electronics
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

December 15, 1994

Abstract

This paper is an introduction to the problem of Critical Path determination in the context of Timing Verification. After correct logic verification, timing is the next most important concern to be addressed by designers. The speed of a circuit is related to the length of the longest path through which a signal can travel, also known as the longest true path or critical path of the circuit.

This paper will focus on the floating mode of operation and will address two main problems, namely the sensitization conditions for a path to be true and the methods that lead to efficient computation of the true delay of a circuit. Included are descriptions of various sensitization conditions in use, explaining its strengths and shortcomings. The relation between these conditions is also established. Finally a method that efficiently computes the circuit delay is presented.

A summary of three very relevant papers on the subject is included and several other references are also discussed. Since these three papers are not comprehensive of the field, we chose to pursue an historic view of the single-vector condition and then explain the contribution of each of the papers.

1 Introduction

This paper is an introduction to the problem of Critical Path determination in the context of Timing Verification. After correct logic verification, timing is the next most important concern to be addressed by designers. Timing verification is an integral part of any circuit optimizer. The verification of the timing characteristics of a circuit is based on estimations of its timing performance. The accuracy of these estimations is therefore critical in the quality of the optimization and timing analysis performed. The importance of accurate timing analysis cannot be overstated. For combinational logic circuits, an accurate estimation of the circuit delay is the best indication of its performance level. For sequential circuits it is an invaluable piece of information since it determines how fast the circuit can be operated.

There are two main components involved in the timing analysis of a circuit: component delay and circuit delay. Component delay deals with the delays of individual components, such as logic gates or blocks and interconnect delays. In general this is performed a-priori with extensive use of detailed device or transistor level simulations of the component and experimental data gathered from actual measurements of the fabricated elements. Circuit delay deals with the delay of the overall circuit given a topology of interconnection of the various individual components.

The computation of component delays involves extensive simulation of the circuit components. Due to variations in the fabrication process as well as operational parameters, the delay of each particular component will likely vary somewhat. In general one cannot do better than determine an interval such that with some likelihood the delay of all members of a family of components will be no smaller than the lower bound and no larger than the upper bound. The data gathered from these measurements is used to validate and parameterize a delay model that must take into account the delay inherent to the component as well as the delay due to the load that is connected to that component's input and output pins. To include these loading effects into the component delay one needs to know the topology of interconnection of the component. We will not be be interested in modeling component delay and will later assume that this data is readily available.

Computing the delay of a circuit is the next step after the computation of the delay of each individual component. In recent years, increasing emphasis has been placed on the problem of fast verification of the timing constraints of circuits. This need was made even more important with the appearance of many timing optimization tools.

For a sequential circuit, the primary concern with respect to the temporal behavior of the circuit is to determine at what speed we may clock the circuit and still be certain that the logic behavior of the circuit is correct. Often this clock speed is a system-wide requirement that must be met in the design. Obviously the most accurate way of verifying this requirement is to build the circuit and test it in its target environment. This option is almost never used due to the time and expense it involves. One therefore resorts to methods that can determine as accurately as possible before fabrication, whether the speed requirements are satisfied. An options that comes to mind immediately is to simulate the behavior of the circuit using an accurate transistor level simulator such as SPICE [1]. Unfortunately such accuracy is not without a cost and in this case it becomes computationally very expensive to perform such simulation. Furthermore the accuracy of the results thus obtained is dependent upon the set of input chosen. To address the computational problem researchers have come up with less accurate but more efficient algorithms. These simulators use simplified device models and equation formulation techniques. They have been available for many years and are widely used by the design community [2, 3, 4, 5, 6, 7, 8, 9, 10]

In fact for the most part, early timing analyzers followed the approach of estimating or verifying the delay of a circuit through simulation. These analyzers, also known as timing simulators, computed circuit delays by simulating the response of the circuit to a given restricted set of input patterns. This approach, although quite accurate, is unfortunately still not very efficient. Even tough the computational burden has been reduced, the task of choosing a set of input patterns is very problematic. The input patterns chosen for simulation must be such that it exercises the circuit in a manner that allows one to accurately determine the delay. If the set of inputs does not exercise the circuit components along the right paths, the true delay may go unnoticed and an incorrect value may be computed. Since the number of input vectors grows exponentially with the number of inputs, it is infeasible to simulate all possible input combinations and therefore vector-independent timing analysis

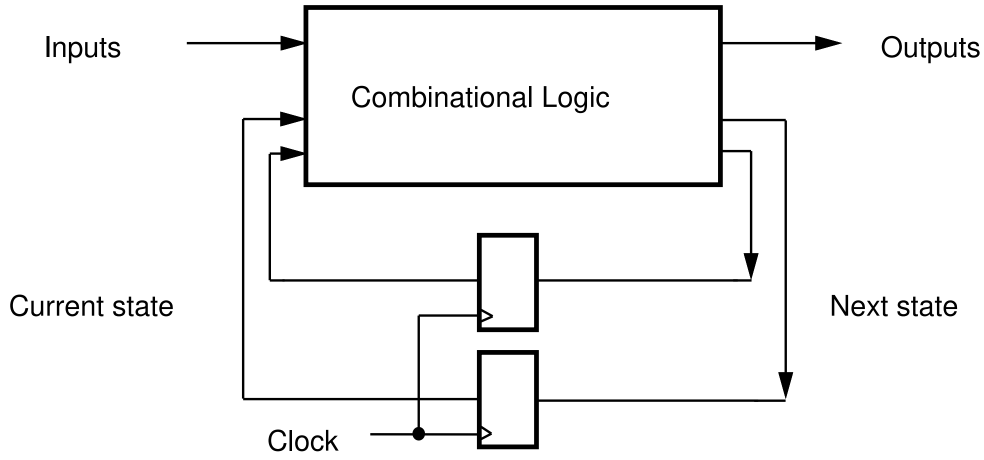


Figure 1: Sequential circuit model.

must be pursued.

For these reasons interest in timing simulation is decreasing and instead timing verification is being used for computing circuit delays. Their principal advantage over simulators is that they find the longest propagation path without requiring input excitation. In early timing verifiers, the logic behavior of the circuit was disregarded and the timing behavior is approximated by simple models [11, 12, 13, 14, 15]. These verifiers, also known as static or topologic timing verifiers compute the delay as being the length of the longest path of the circuit. As a result of these abstractions the problem of computing the delay of a circuit is reduced to a graph problem where the weights in the edges represent the delays of the gates. This problem is known to be easily solved in linear time. Unfortunately in some circuits, there may not be any input pattern that activates the longest path and the estimate of the delay computed may be too pessimistic. Such paths are usually called *false* [16] and it is clear that techniques had to be developed to account for them when determining tighter estimates of the delay of a circuit. Due to the ease of generating false paths in high-level synthesis systems [17], there is a growing need for correctly identifying false paths to guide performance optimization, in a computationally efficient manner.

In this paper we will describe some of the techniques that were attempted to deal with the problem of false paths. The methods described are aimed at determining the length of the longest path through which a signal can travel, also known as the longest true path or critical path of the circuit. Although the problem of timing verification is usually associated with that of verifying that a sequential circuit can be clocked at some given speed, we will restrict ourselves to combinational logic. Even though this may seems restrictive at first, it is much less so than one can envision. Consider for example a simple implementation model of a sequential circuit, as shown in Figure 1. The most important constraint for correct operation of such a circuit is that the clock period, neglecting setup and propagation times, be longer than the maximum propagation delay through the combinational logic. Therefore to a reasonable approximation, the delay calculation of the sequential circuit is solved by computing the propagation delay of a multilevel combinational circuit.

In section 2 we will introduce a series of notations and definitions that are used throughout the paper. We will give some examples of how false paths show up in topological-based verification. We will also characterize more precisely the notion of delay of a circuit, and will illustrate how that definition is dependent upon assumptions relating to the delay models chosen and the mode of operation of the circuit. In section 3 we will very briefly introduce the transition mode of operation and discuss some recent ideas in the field. In section 4 we will devote our attention exclusively to the floating-mode of operation and introduce several sensitization conditions explaining the relations between them. In section 5 we will refer to algorithms that are directed to efficient computation of the delay in floating mode. We will conclude in section 6 with a presentation and discussion of three relevant papers each with its own ideas about how the problem should be solved. Finally some conclusions will be presented in section 7.

2 Timing Analysis and Verification

The actual delay of a combinational circuit can be defined as the longest delay it takes from the application of some input vector until all outputs become stable with their final values. Clearly the delay will never be larger than the length of the longest topological path in the circuit. If this length is smaller than the desired clock period or other timing constraint, then the circuit will meet its timing requirement. However, even if that is not the case, the circuit can still satisfy those requirements. In fact it has long been realized that the actual delay of a circuit may be much smaller than the longest topological path would indicate. In this case any algorithm based on topological estimates may actually overestimate the delay leading to either conservative clocking regimes or even to discarding an otherwise properly functional circuit.

In section 2.1 we will introduce some formal definitions and notation for the terms we have been using and will be referring to in the rest of the paper. In section 2.2 we will illustrate with an example how false paths appear in logic design. Finally in section 2.3 we will discuss how the choice of delay models chosen affects the computation of the delay and will introduce the different modes of operation.

2.1 Notation and Definitions

This section contains some definitions terminology we have and will be using. We start with simple straightforward concepts and proceed from there:

Definition 2.1 (Combinational Circuit)
A combinational circuit is a directed acyclic graph composed of gates, represented by nodes in the graph, and connections, represented by edges between them. Each gate g has a delay $d(g)$ and each connection c has a delay $d(c)$ associated with it.

Definition 2.2 (Path)
A path $P = \{c_0, g_0, c_1, \cdots, c_n, g_n, c_{n+1}\}$ *in a combinational circuit is a alternating sequence of connections and gates (also referred to as vertices and edges in graph terminology). Connection* c_i*,* $1 \leq i \leq n$ *connects the output of gate* g_{i-1} *to an input of gate* g_i*. The* g_i*'s are referred to as the gates along the path.* c_0 *are the primary inputs and* c_{n+1} *are the primary outputs.*

Definition 2.3 (Depth)
The depth of a circuit is the maximum number of gates along any path in the circuit.

Definition 2.4 (Fan in, Fan out, Transitive fan in)
If the output of gate g_i *is connect to the input of gate* g_j*, then* g_i *is a fan in of* g_j *and consequently* g_j *is a fan out of* g_i*. if there is a path from* g_i *to* g_j*, then* g_i *is a transitive fan in of* g_j*.*

Definition 2.5 (On-Input, Side-Input, Side-Path)
If $P = \{c_0, g_0, c_1, \cdots, c_n, g_n, c_{n+1}\}$*, then* c_i *is referred to as the on-input to path* P *at* g_i*. All other inputs to* g_i *are referred to as side-inputs. The set of all side-inputs in all gates in a path are the path side-inputs. Any other path that start at a primary input and ends at a side-input of* P *is called a side-path of* P*.*

Definition 2.6 (Literal)
A literal refers to a boolean variable that appears either in its complemented or uncomplemented form in some boolean expression.

Definition 2.7 (Cube, Minterm)
A cube is a boolean product of literals. A minterm is a cube in which every variable in the boolean function appears.

Definition 2.8 (Length of a path)
The length of a path $P = \{c_0, g_0, c_1, \cdots, c_n, g_n, c_{n+1}\}$ *is defined as* $d(P) = \sum_{i=0}^{n} d(g_i) + \sum_{i=0}^{n+1} d(c_i)$*.*

Definition 2.9 (Event, Event propagation)
And event is a transition in logic value at a node. If a sequence of events $\{e_0, e_1, \ldots, e_n\}$ *occuring at gates* $\{g_0, g_1, \cdots, g_n\}$ *along a path, is such that* e_{i+1} *occurs as a result of event* e_i*, then event* e_0 *is said to propagate along the path.*

Definition 2.10 (Sensitizable Path, True Path)
If an event can propagate along a path, the path is said to be sensitizable. Sensitizable Paths are also known as true paths.

In other words, there must be some input vector that causes a signal to propagate along a sensitizable path.

Definition 2.11 (False Path)
Any path P which is not sensitizable is a false path.

Definition 2.12 (Critical Path, Circuit Delay)
The critical path is the longest sensitizable path in the circuit. The circuit delay is the length of the critical path.

Definition 2.13 (Topological or Graphical Delay)
The topological or graphical delay of a circuit is the length of the longest topological path in the circuit (irrespective of whether it is sensitizable or not).

Definition 2.14 (Path Sensitization problem)
The path sensitization problem is the problem of deciding whether a given path P in a combinational circuit is sensitizable.

Definition 2.15 (Critical Path problem)
The critical path problem is the problem of determining the delay of a circuit or equivalently of determining the critical path of a circuit.

Definition 2.16 (Controlling Value, Non-Controlling value)
A controlling value at a gate input is the value that determines the value at the output of the gate independently of the other inputs. A non-controlling value at a gate input is the value that is not a controlling value for that gate.

For instance a logic **0** is a controlling value for an AND gate. Consequently **1** is a non-controlling value for an AND gate.

Definition 2.17 (Controlling Input, Non-Controlling input)
If an input to a gate presents a controlling value for that gate, that input is referred to as controlling input. Similarly, if an input to a gate presents a non-controlling value for that gate, that input is referred to as non-controlling input.

Armed with this set of definitions one can now take on the problem of determining the delay of a circuit.

2.2 Topological Timing Analysis

Topological timing analyzers describe the circuit as a directed acyclic graph. As such they discard the boolean functionality of the circuit and look for the longest path in the circuit. The problem with this approach is that not all paths are sensitizable by any input

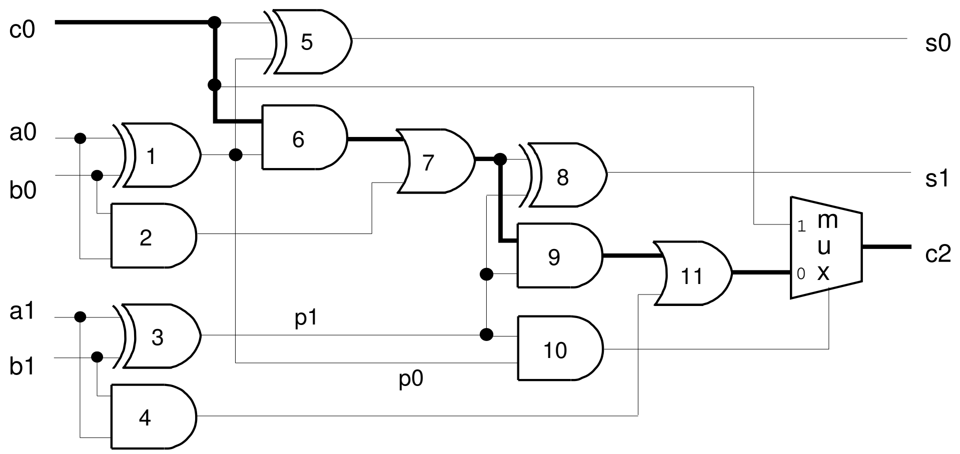


Figure 2: Carry-bypass adder example.

vector and therefore some paths cannot be responsible for the delay of the circuit. As defined previously, we refer to those paths as being false. The existence of false paths has been known for some time now [18]. Researchers have come out with simple example circuits where false paths can be found. A well-known real-life circuit exhibiting a false path is the modified ripple-carry adder shown in Figure 2. The functionality of the circuit is straightforward: when the propagate signals `p0` and `p1` are high, the multiplexer simply passes the carry-in signal `c0` to the output `c2` thus allowing the carry to skip the ripple-carry chain computation. For the sake of argument let us assume that the delay of all gates is fixed, being 1 for the simple AND and OR gates and 2 for the EXOR and the MUX. It is easy to see that the longest path in the circuit is the one shown in bold from `c0` to `c2` through gates `6`, `7`, `9`, `11` and the MUX. The delay of this path is 11. However, this path is clearly false because for a signal to propagate through the path both propagate signals would have to be high and in that case the MUX chooses the bypass route outputting `c0` directly. Since the path is false, it cannot be responsible for the delay of the circuit, and in fact the path that determines that delay is the path from `a0` to `c2` through gates `1`, `6`, `7`, `9`, `11` and the MUX and the delay of the circuit for the values given is 8.

As the example shows, the longest topological path approach is needlessly pessimistic since the chosen paths can be false and therefore not responsible for the delay of the circuit. Clearly any topology-based approach to the critical path problem is bound to fail since the logic behavior of the circuit is discarded and only its connectivity is accounted for. In early timing verifiers [12], facilities were provided for the designer to indicate, based on his experience and knowledge of the circuit behavior, which paths would never be exercised. This technique became common in the use of timing verifiers, because for an algorithm to be efficient and accurately compute the delay of a circuit it had to eliminate these false paths from consideration. Unfortunately it is widely accepted that the circuits that are most-troublesome for false path elimination are those with literally thousands of paths and in particular thousands of longest paths. The consensus seems to be in fact, that in any circuit false paths either do not occur or are quite numerous, which excludes manual elimination as an overly tedious and error-prone approach. Therefore the automatic solution of the false path problem, and the incorporation of such automated solution in modern timing verifiers has become a matter of great importance and a subject of intense research [16, 19, 20, 21, 22, 23]

The validity of any useful algorithm will be determined by the ability to correctly identify and extract the sensitizable paths of a circuit. It has been proposed [20] that the validity of such an algorithm will force any sensitization condition to meet two criteria. The first condition is a mere correctness condition:

Definition 2.18 (Correctness Criteria)
For each true path in the given circuit, there must be a path P at least as long which is sensitizable under the current sensitization condition. In other words, the sensitization condition must not underestimate the length of the critical path.

It is critical that the length of the critical path not be underestimated. Were this to happen and a value smaller than the true delay could be reported, which could lead to incorrect circuit functionality. On the other hand it is desirable that this length not be overestimated, although this is only due to accuracy concerns and is not a critical problem [1].

The second criteria takes into account the uncertainty due to fabrication and environmental conditions. For any given circuit perfect timing information is not available since the circuit being analyzed by our verifier is an idealized one which does not suffer from any of the variations that the real circuits will undoubtfully present. To understand this situation one can envision the circuit under analysis as being a member of a family of topologically identical circuits with varying timing properties. The previous criteria follows from the reasoning that the component delays usually quoted correspond to worst-case and best-case delays and the fact that one wants to ensure that the delays is not underestimated [2]

Definition 2.19 (Robustness Criteria)
For each true path P in any member of the circuit family, there must be a path Q in the slowest member of the family such that Q is at least as long and also true.

These criteria will be used as guidelines for determining our acceptance of any sensitization condition for delay computation.

2.3 Delay Models and Modes of Operation

While studying the example circuit in Figure 2 we made two assumptions regarding the temporal behavior of the circuit and how we determine its delay. First we established fixed delays for every gate. Then we also did not detail what type of input patterns would allow us to detect the true delay of the circuit.

The first assumption regards the delay model, shorthand notation for component delay model which was already presented in section 1. Using a fixed delay D is the most commonly used model, properly called the *fixed delay model*. Of course a value of zero can also be

[1] In the worst case one ends up with a design working under its capabilities, which is wasteful but still correct.

[2] This condition has also been called the monotone speedup property in [20]; the idea is to guarantee that speeding up the delay of any set of components will not result in a larger circuit delay, which would be awkward.

used, although that is not of interest in terms of delay computations. In some situations this value is one for every gate, in which case it is called the unit delay model. However it is not necessary to use the same value for every gate. Fixed delays are to some extent unrealistic and can only provide for an abstraction of a fabricated circuit and not a family of circuits. For the most part the delay value used if typically either and upper or lower bound (generally an upper bound) on the delay of the component in the fabricated circuit, or some average value for a family of components. Another model that is commonly used is the *bounded delay model*, where the delay of each component is now given as two numbers, $[d_{min}, d_{max}]$, where d_{min} is a lower bound on the delay of the family of components, and d_{max} is the equivalent upper bound. These values take into account parameter variations in the fabrication or packaging processes and represent best-case, worst-case bounds [3]. In the special case where the lower bound equals zero, the model takes the name of *monotone speedup delay model*. The rationale behind this model is that it allows one to certify oneself that, as expected, the computed circuit delay will not be exceeded if gate delays in the circuit are decreased. The bounded delay is however more realistic about how much each gate can be sped up.

A wealth of other delay models exists that are used whenever more accuracy is desired. These include among others, models with nonzero and different rise and fall times for node transitions, models with different delays in each pin, slope delay models, etc. For the sake of simplicity, we will not consider such models in this discussion and will restrict ourselves to using the simpler fixed and monotone speedup component delay models.

The second assumption relates to the mode of operation of the circuit while computing its delay. Two possibilities have been considered and extensively studied in the literature. In the *transition mode of operation* two input vectors, v_1 and v_2 are successively applied to the circuit. The circuit nodes are assumed to be ideal capacitors, with values set by v_1 and retained until v_2 forces the voltage to change. Immediately one can see that for an n-input circuit, the search space under this mode of operation is of size 2^{2n}. Using a bounded delay model instead of a fixed delay one would further difficult the analysis. In the *floating mode of operation*, a terminology first used in [25], the nodes are not assumed to be ideal capacitors and hence their state is unknown until set by v_1. In other words, this mode of operation implies a single vector condition which makes it therefore significantly easier to compute the delay. Although this simplifies the search space and the algorithms to be used, that is achieved at a cost. In this case, since the timing behavior with respect to the applied vector must be independent from past input patterns, pessimistic conditions must be assumed regarding the previous state of the circuit nodes. In the following sections we will consider the computation of delay under the transition and the floating mode of operation. Since we are really mostly interested in floating mode conditions, our presentation of the transition mode will be very cursory and will basically consist on the presentation of the basic ideas, while indicating some references for further reading. This is done in section 3. The floating mode of operation is discussed in detail in section 4.

[3] In [24] it is noted that there seems to be some ambiguity in the literature about the definition of a bounded delay. This is mostly due as to whether one interprets this delay in terms of a switching or a propagation. If interpreted as a switching delay, then on a transition, the value at the gate output should be considered unknown for the time period between d_{min} and d_{max} which implies the use of ternary algebras. If on the other hand the delay is interpreted as a propagation delay, then the gate switches instantly but at an uncertain time within the interval $[d_{min}, d_{max}]$. In this case the logic value at the gate output is always either a 0 or a 1 and ternary algebras are not needed. In this paper we will adopt the propagation delay interpretation.

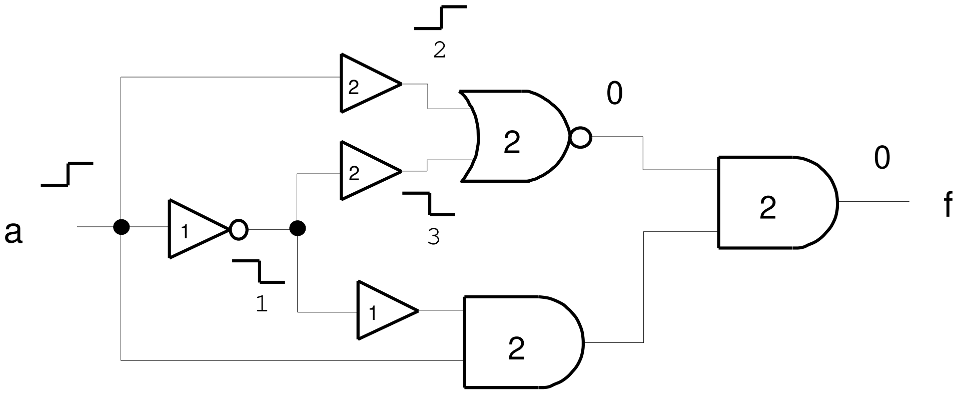
3 Delay under the Transition mode of Operation

The transition delay of a circuit is the delay under a multi-vector condition that makes no assumptions about the state of the circuit before the vector sequence is applied. There are several possible definitions for the transitions delay of a circuit. Consider a combinational circuit to which two input vectors v_1 and v_2 are applied in succession. In the transition mode of operation the circuit nodes are considered ideal capacitors and retain their value set by the previous vectors until the current vector forces the voltage to change. Therefore the timing response for vector v_2 is a function of v_1 and possibly other vectors applied previously. So the first question that arises is how much of the circuit history should one consider. Clearly one would like to deal with as little as possible without compromising accuracy. This has led to the definition of single stepping transition [24] mode in which one assumed that when vector v_2 is applied all circuit nodes have stabilized to their values under v_1. This is equivalent to assuming that v_1 was given an arbitrary amount of time for the circuit to settle. Other definitions can be considered which include more of the circuit history but we will restrict ourselves to the single stepping transition mode in this short summary of transition mode.

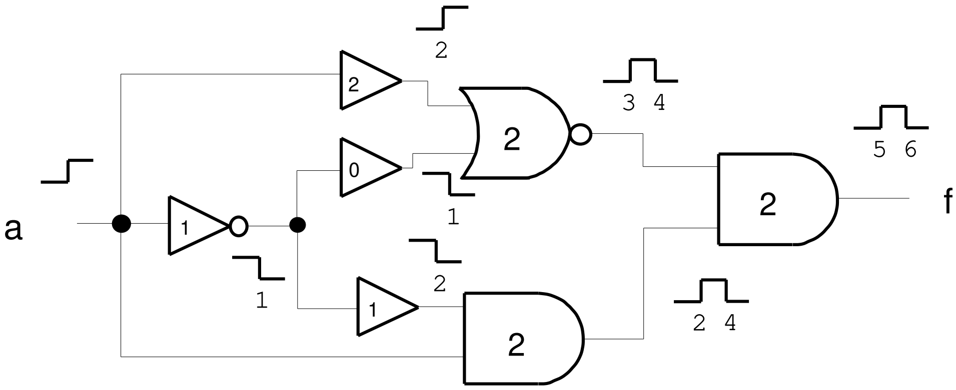
Consider the circuit of Figure 3(a), which is taken from [20]. Assume that the delays of each gate are fixed and shown inside the gates themselves. Given that this circuit is single input there are only two input vectors that can be used, but in order to determine the critical delay of the circuit we will have to simulate the two vector pairs corresponding to a rising and falling input transition. However, in this case it is easy to see that applying $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions on a does not change the output f from 0. Thus, one can conclude that the circuit has critical delay 0 under the transition mode of operation for the given fixed gate delays. Now consider the circuit of Figure 3(b) which is topologically identical to the circuit of Figure 3(a) but in which the buffer at the input to the NOR gate has been sped up from 2 to 0. One might expect that speeding up a component in a circuit would not increase the critical delay of a circuit. However, as shown, for the $0 \rightarrow 1$ transition on a, the output f switches both at time 5 and time 6, and the critical delay of the circuit is 6.

The above example shows that a sensitization condition based on transition mode and fixed gate delays is unacceptable since it does not satisfy the robustness criteria introduced in definition 2.19. This makes it unnaceptable in a worst-case design methodology where one is given the upper bounds on the gate delays (which could be the delays shown in each gate), and is required to report the (worst-case) critical path in the circuit.

Since in the previous example speeding up one of the gates produced a more acceptable result, one is lead to consider as a strategy using the transition mode of operation and monotone speedup. For that consider the circuit of Figure 4, which is identical to that in Figure 3(a), except that each gate delay can vary from 0 to its given upper bound, as defined in the monotone speedup component delay. As before, in order to determine the critical delay of the circuit we will have to simulate the two vector pairs corresponding to the following and rising transitions in the input. However, in this case simulating the circuit is a much more complicated process since the transitions at the internal gates may occur at varying times given the uncertainty in the gate delays. In Figure 4 we consider a rising input transition and show the waveforms that can appear at the outputs of each gate. For instance, the NOR gate can either stay at 0 or make a $0 \rightarrow 1 \rightarrow 0$ transition, where the



(a)



(b)

Figure 3: Transition mode with fixed delays.

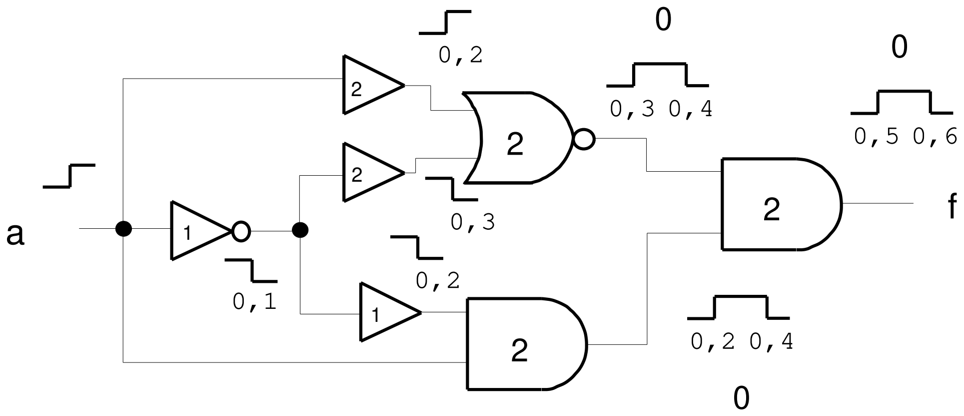


Figure 4: Transition mode with monotone speedup.

transitions can occur between $[0, 3]$ and $[0, 4]$ respectively, given the values shown. In order to determine the critical delay of the circuit, we must scan all the possible waveforms at output f and find the time at which the last transition occurs over all the waveforms, which provides us with a critical delay of 6. If one were to use a generic bounded delay component model the waveforms would be computed similarly [24].

A few facts should be clear at this point regarding the transition mode of operation. If a strategy based on a fixed delay model is unacceptable, usage of monotone speedup under the transition mode is a viable option for timing analysis for a worst-case design methodology. However, as mentioned before there are still several problems with this technique. Firstly the search space is 2^{2n} where n is the number of primary inputs to the circuit, since we may have to simulate each possible vector pair [4]. Secondly, monotone speedup delay simulation is significantly more complicated than fixed delay simulation as waveforms now have to be kept for each node. There are algorithms based on symbolic simulation that can compute the transition delay of a circuit but we will not discuss them here in detail. They use techniques for event suppression and for prunning the search space to improve the efficiency of the computation. For more details the reader is encouraged to consult references [26] and [24].

4 Delay under the Floating Mode of Operation

The floating delay of a circuit is the delay under a single-vector static analysis condition that considers the boolean behavior of the circuit but makes conservative, i.e. pessimistic, assumptions about the state of the circuit before the single vector is applied.

Attention and research in the floating mode of operation were sparked by its computational simplicity in contrast to transition mode. Under floating delay, delay is determined by a single vector, a simplification that makes it significantly easier to compute even under more complex component delay models.

An interesting result, first shown in [25], indicates that the circuit delay in the floating mode is an upper bound of the delay in the transition mode. That knowledge immediately tells us that the floating delay satisfies the correctness criteria although it says nothing about the tightness of the estimate. Due to this property and its relative computational simplicity, the floating delay is sometimes used as the starting point for algorithm computing transition mode delay. Nevertheless in many cases the floating mode delay may be accurate enough for the purposes of timing verification.

There has been significant research done in an effort to arrive at the correct sensitization criterion. A detailed history may be found in [28]. In this section, we will first introduce delay-independent sensitization conditions, since they were historically the first ones to be considered and are easier to understand. These will then be augmented to arrive at the correct sensitization criterion.

[4]or larger if one is not using single stepping transition mode and is using more of the nodes history.

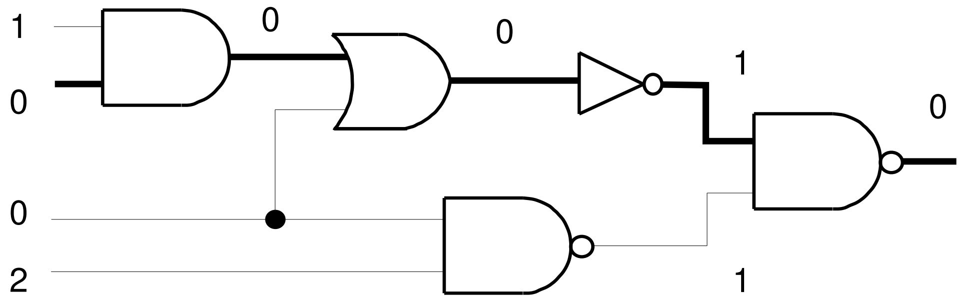


Figure 5: Static sensitization of a path.

4.1 Sufficient Sensitization Conditions

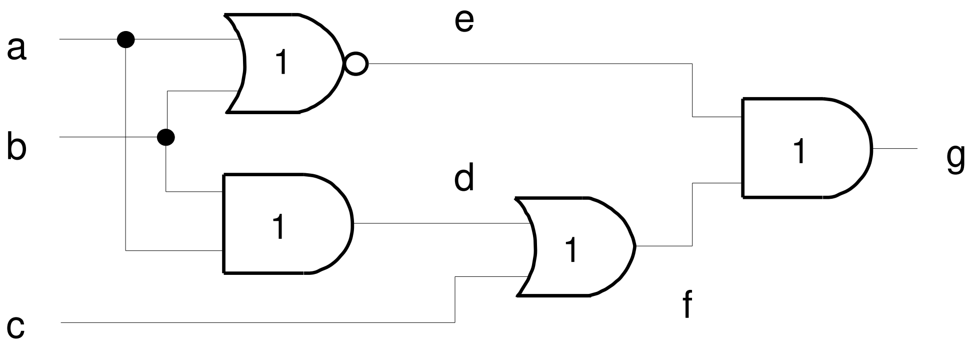
The most primitive of the sensitization conditions, and one that surprisingly is still widely use, is the concept of static sensitization which we define next:

Definition 4.1 (Static Sensitization Condition)
A path π is **statically sensitized** *by a vector if all the side-inputs along π have non-controlling stable values.*

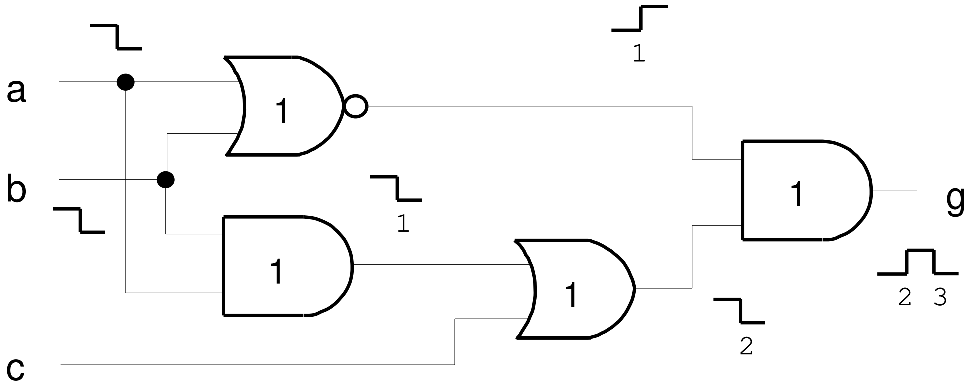
Static sensitization [16, 29] which ignores all dynamic propagation effects, is therefore delay-independent and resembles path sensitization approaches in testing.

A superficial study on static sensitization may lead one to think that it is the sensitization condition required. In fact, if one considers for instance the circuit of Figure 5, the path shown in bold has been statically sensitized by the vector 1002, where 2 stands for the unknown or undefined value. An event can propagate down the path by applying the vector pair $< 1102, 1002 >$. Also, we we go back to our example circuit with the false path, in Figure 2 one sees that no input vector can be found that statically sensitizes the path shown in bold. This is because in order to sensitize the bold path we require a 0 at the multiplexor input, which implies that either or both propagation signals have to be 0. But if `p0` or `p1` are 0 then gate `6` or gate `9` respectively will have a controlling value at the side-input to the bold path, which would violate the definition of static sensitization. We also know that we cannot find an input vector pair under the given gate delays that propagates a transition down the path.

Unfortunately it is known that static sensitization may underestimate the delay of a circuit [20, 30]. For example, consider the circuit of Figure 6, which is the example taken from [20]. If all gate delays are assumed to be 1, the paths $\{a, d, f, g\}$ and $\{b, d, f, g\}$ are not statically sensitizable. To see this, recall that in order to statically sensitize path $\{a, d, f, g\}$, a 1 must be present on e which implies that both a and b have to be 0. However, this in turn implies that we have two controlling values at the AND gate with output d and therefore the path is not statically sensitizable. A similar analysis can be done for the path $\{b, d, f, g\}$. Since the only two paths of length 3 are not statically sensitizable, this may lead us to conclude that the critical delay of the circuit is at most 2. However, that is not the case and one can see that by keeping c at 0 and toggling a and b from $1 \rightarrow 0$, which propagates a transition at time 3 to the output of the circuit. Therefore if a delay is set to the value of



(a)



(b)

Figure 6: Static sensitization underestimates delay.

the longest sensitizable path, the circuit will behave incorrectly on this input vector. Thus, a path may not be statically sensitizable but can be responsible for the delay of a circuit, or in other words, *static sensitization does not satisfy the correctness criteria.*

In [27] the following interesting result is introduced which again we include for completeness.

Proposition 4.2 *Under the floating mode of operation, static sensitization is a sufficient condition for a path to be responsible for the delay of a circuit.*

Proof. Consider a vector v_2 being applied to a circuit C where v_2 statically sensitizes path π to a 1. By definition, on v_2 all the side-inputs to π are at non-controlling values. Under the floating mode of operation, on any particular gate g on π, the values of the side-inputs of g on the previously applied vector can be assumed to be at non-controlling values. This implies that we have steady non-controlling values at each side-input. Thus the event propagating down π sees non-controlling values on all side-inputs and is able to propagate to the primary output. This of course implies that π is responsible for the delay of C. $\square$

As any sensitization condition used in the floating mode of operation, static sensitization makes pessimistic assumptions and therefore may, in general, overestimate the delay of a circuit. Since we have just seen that it can also underestimate the delay, then given a circuit it is not possible to decide whether the delay computed under the static sensitization condition is less than, equal to or greater than the real delay. It has been pointed out that static sensitization is an inadequate criteria because it fails to take into account the propagation delays of the circuit components. In particular it implies static conditions on the values of the path side inputs, whereas in reality these conditions only have to be met at the time the event propagating through the path reaches each gate.

4.2 Necessary Sensitization Conditions

In order to weaken the conditions imposed by static sensitization, the concept of static co-sensitization was developed [22].

Definition 4.3 (Static Co-Sensitization Conditions)
We say that an input vector w **statically co-sensitizes to a 1***(0) path* $\pi = \{v_0, e_0, \cdots, v_n, e_n, v_{n+1}\}$ *in a circuit C, if and only if 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} presents a controlling value.*

Immediately we notice that static co-sensitization is also a delay-independent condition similar to static sensitization, but is weaker than static sensitization. Consider Figure 7, where it is easy to see that the application of the vector 000 statically co-sensitizes the paths shown in bold $\{a, d, f, g\}$ and $\{b, d, f, g\}$ to a 0. If one were to use static co-sensitization as our delay criterion, one would report the correct delay of 3 for this circuit.

Unfortunately static co-sensitization can be pessimistic. To understand that consider the trivial circuit of Figure 8, taken from [27]. The bold path shown has length 6 and is

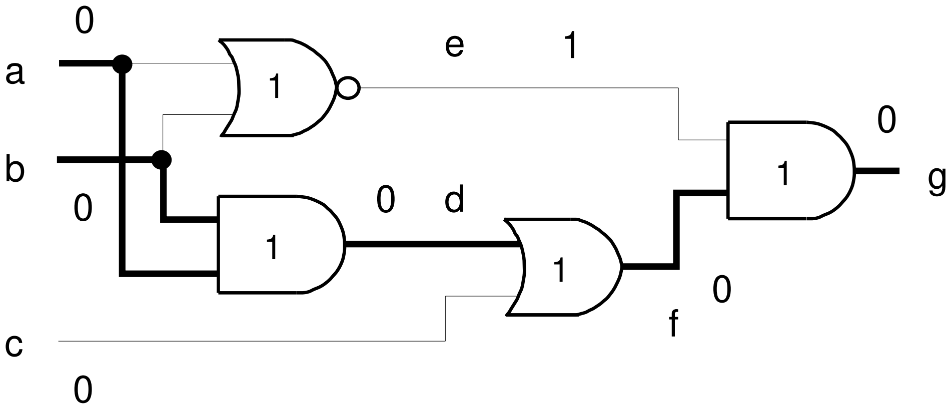


Figure 7: Static co-sensitization of paths.

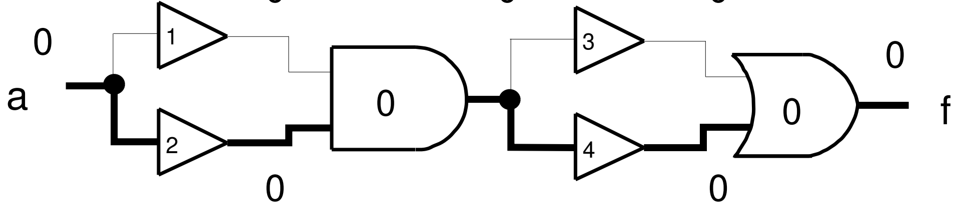


Figure 8: Static co-sensitization can be pessimistic.

statically co-sensitized by the vector $a = 0$. However, the critical delay of the circuit is no greater than 5 which can be seen by applying rising or falling transitions on a.

Again, in [27] the following result is proved:

Proposition 4.4 *Under the floating mode of operation, static co-sensitization is a necessary condition for a path to be responsible for the delay of a circuit.*

Proof. Consider a path π that is not statically co-sensitizable. Therefore, for any applied vector v_2 there is at least one gate g on π for which the input to g corresponding to π has a non-controlling value and some other side-input to g has a controlling value. It is clear that the controlling value at this side-input will always control the output of g. If the non-controlling value arrives before the controlling value, the gate output will not change and it will only go to the controlling value after the controlling value arrives. On the other hand, if the controlling value arrives before the non-controlling value, the gate output will be at the controlling value before the non-controlling value arrives. In either case, π is not responsible for the delay of the circuit on v_2. $\square$

Therefore, although correct, static co-sensitization does not seem to be the sensitization condition one would like to use.

4.3 True Floating Mode Sensitization

In the previous two sections, two delay-independent conditions, static sensitization and static co-sensitization were presented that are, respectively, sufficient and necessary for a path to be responsible for the delay of a circuit. The sensitization condition we seek, is both a necessary and sufficient condition for a path to be responsible for circuit delay under the floating mode of operation. Such a condition is delay-dependent and is stronger than static co-sensitization but weaker than static sensitization. It was proposed in [25, 32] and further developed in [33, 34, 35].

To understand the delay-dependent sensitization condition and the timed-calculus we are about to present, we will start by reviewing the fundamental assumptions made in delay-dependent single-vector analysis. It is enough for that to consider a two input AND gate under the fixed delay model [5]. Consider then the AND gate in Figure 9(a), which we assume has delay d and is embedded in a larger circuit. Since the AND gate is commutative, it is enough to consider three different cases corresponding to the application of vector pairs $< v_1, \ v_2 >$ to the circuit inputs. First we will consider rising transitions on both gate inputs. Then we will consider falling transitions and finally we will consider the case where one input is rising and the other is falling. Transitions in each input will happen at independent times, say at t_1 and t_2. If both input transitions are rising, then clearly the output of the gate will rise at a time given by $MAX(t_1, t_2) + d$. Recall that the abstraction under floating mode of operation only shows the value of v_2. Therefore in floating mode the previous transition is modeled by assuming that a 1 arrives at the first and second inputs to the AND gate at times t_1 and t_2 respectively, and a 1 appears at the output at time

[5] Since as we saw in floating mode the delay produced under this model is the same as that produced with monotone speedup and bounded delay.

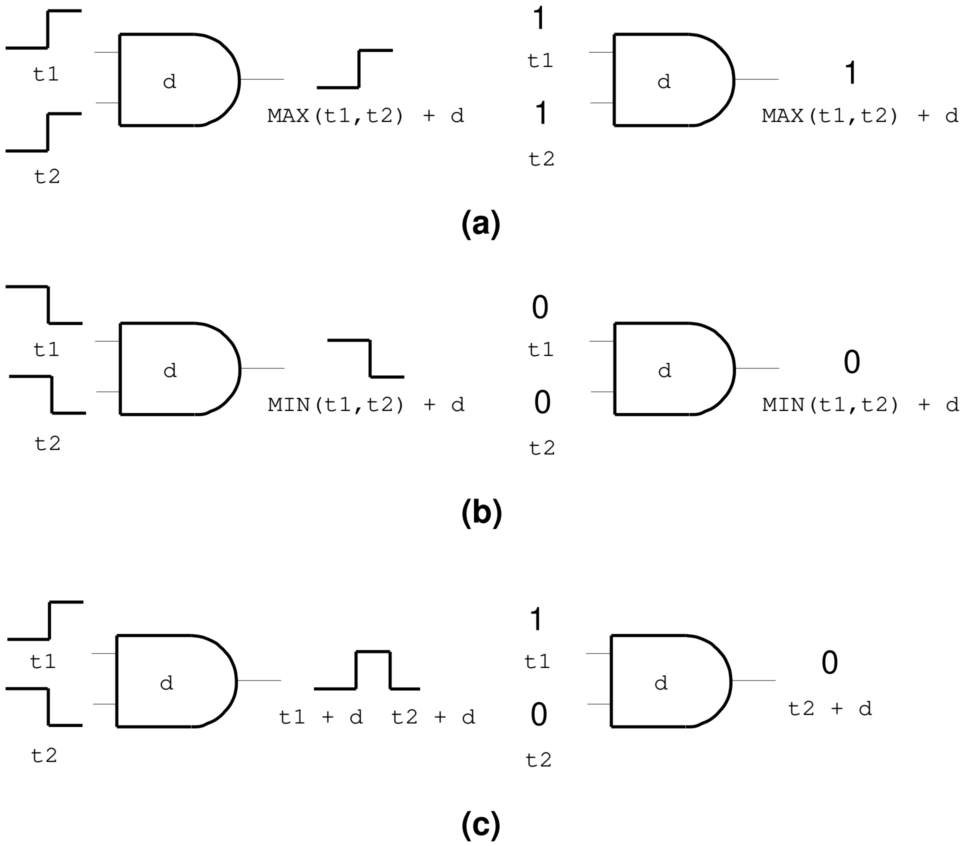


Figure 9: Fundamental assumptions made in floating mode operation.

$MAX(t_1, t_2) + d$. Similarly in Figure 9(b), two falling transitions at the AND gate inputs result in a falling transition at the output at time $MIN(t_1, t_2) + d$.

Now consider Figure 9(c), where a rising transition occurs at time t_1 on the first input to the AND gate and a falling transition occurs at time t_2 on the second input. Depending on the relationship between t_1 and t_2, the output will either stay at 0 (for $t_1 \geq t_2$) or have a small glitch to 1 (for $t_1 < t_2$). Since under the floating mode of operation, we only have the vector v_2, it is not possible to perform a timed two-vector simulation to determine whether the output would glitch or not (such a simulation was illustrated in Figure 4 in section 3 for the transition mode). Under floating mode all we know is that the 1 at the first input to the AND gate arrives at time t_1 and the 0 at the second input arrives at time t_2. The output of the AND gate obviously settles to 0 on v_2, but since we do not know the relation between t_1 and t_2 we do not know when it settles nor whether it always stays at 0 or whether it glitches (which will happen only if $t_1 < t_2$). In other words we do not know whether the output settles to 0 at $t = 0$ or at $t = t_2 + d$. In order to satisfy the correctness criteria stated in Definition 2.18 and therefore not to underestimate the delay, all single-vector sensitization conditions *have* to assume a pessimistic standing. In this case this corresponds to assuming that the glitch may indeed occur i.e., that $t_1 < t_2$, or equivalently that the 1 (the non-controlling value for the AND gate) arrives before the 0 (the controlling value for the AND gate). If a two-vector condition were being used, this corresponds to assuming that the values on the previous vector v_1 were non-controlling, that is, the subcircuit that is driving the first input can be sped up to cause the rising transition to arrive before the falling transition. Therefore in essence the robustness criteria is satisfied since it is embedded in the fact that our sensitization criteria will return the same value if monotone speedup is considered.

We can generalize the rules described in Figure 9 to represent a timed calculus for single-vector simulation. Such a calculus can be used to determine the correct floating mode delay of a circuit and the paths that are responsible for the delay under that vector. The result of such a computation is necessarily conservative since pessimistic assumptions were made regarding the state of the circuit nodes prior to application of the input vector.

Definition 4.5 (Timed Calculus Rules)
`Controlling Value Rule:` *If the gate output is at a controlling value, pick the minimum among the delays of the controlling values at the gate inputs and ignore the non-controlling values. Add the gate delay to this value to obtain the delay at the gate output.*
`Non-Controlling Value Rule:` *If the gate output is at a non-controlling value, pick the maximum of all the delays at the gate inputs. Add the gate delay to the chosen value to obtain the delay at the gate output.*

For each vector v_2 we can now determine whether a path is responsible for floating mode delay under v_2, simply by simulating the circuit with that input and using the timed calculus. We then arrive at our sensitization condition:

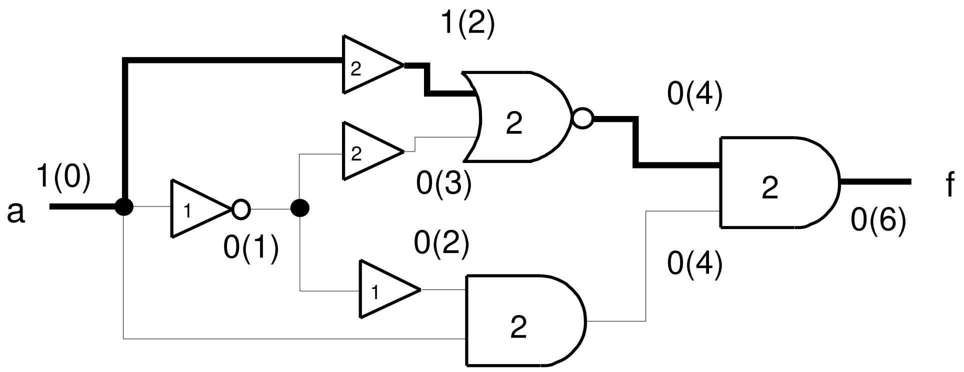


Figure 10: Floating delay sensitization conditions are robust.

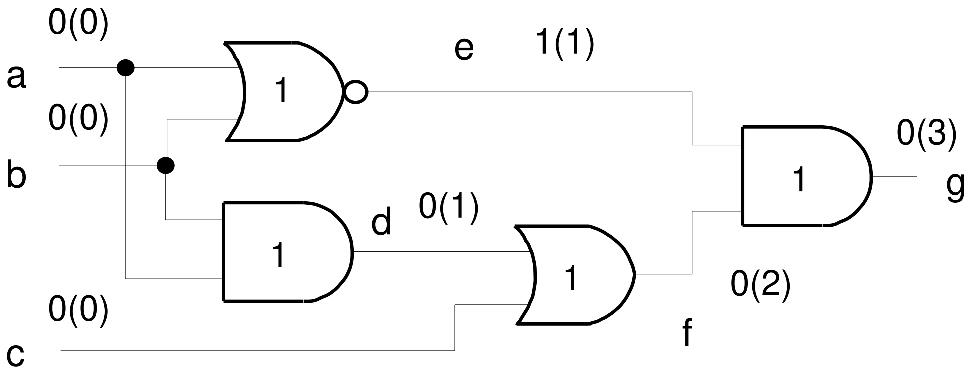


Figure 11: Floating delay sensitization is weaker than static sensitization.

Definition 4.6 (Floating Mode Sensitization)
A path P on a circuit is **sensitizable** *under the floating mode of operation if and only if for each gate along the path the following conditions are met:*

1. *If the gate output is at a controlling value, then the input to the gate corresponding to the path has to be at a controlling value, and furthermore has to have a delay no greater than the delays of the other inputs with controlling values.*

2. *If the gate output is at a non-controlling value, then the input to the gate corresponding to the path has to have a delay no smaller than the delays at the other inputs.*

It is instructive to apply the above conditions to determine the delay of some of the circuit that we have introduced previously. Consider the circuit of Figure 3(a) reproduced in Figure 10. Applying the vector $a = 1$ sensitizes the path of length 6 shown in bold illustrating that the sensitization condition takes into account monotone speedup (unlike transition mode fixed delay simulation) and therefore satisfies the robustness criteria stated in Definition 2.19. Since we are using the timed calculus, we not keep for each node a logical value and a delay value under the applied vector.

The circuit in Figure 11 reproduced from Figure 6 shows that this sensitization condition is weaker than static sensitization and does not fail the correctness criteria on this examples.

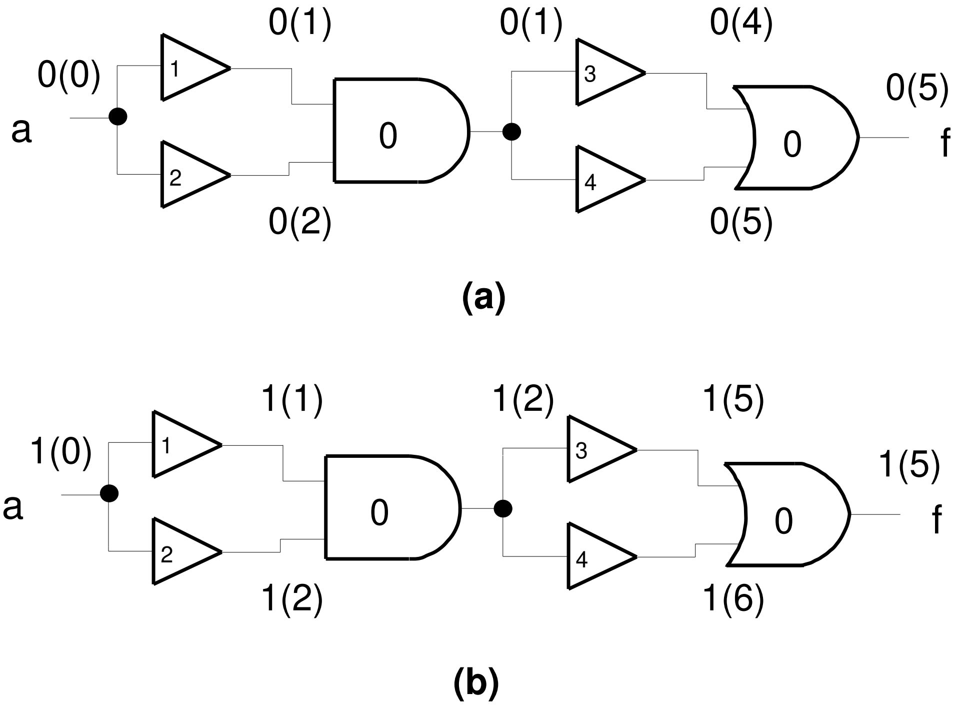


Figure 12: Floating delay sensitization is stronger than static co-sensitization.

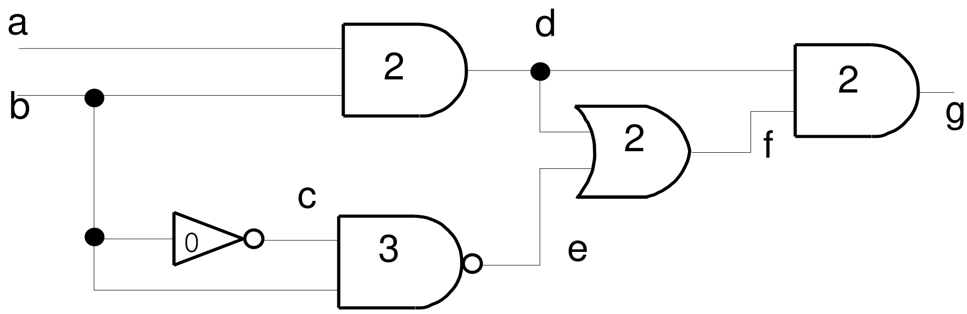


Figure 13: Floating delay sensitization may overestimate the delay.

Applying the vector 000 gives a floating mode delay of 3 as expected. The paths $\{a, d, f, g\}$ and $\{b, d, f, g\}$ which as we saw are not statically sensitizable can be seen to be responsible for the delay of the circuit.

Finally, consider the circuit of Figure 8 reproduced in Figure 12. Applying $a = 0$ and $a = 1$ results in a floating mode delay of 5 illustrating that the sensitization condition is stronger than static co-sensitization and again satisfies the correctness criteria for this circuit.

It can be shown that the sensitization criteria introduced is indeed both a necessary and a sufficient condition for a path to be responsible for the delay of the circuit under the floating mode of operation. A topologically-oriented formal proof can be seen in [25] while [22] uses the Equivalent Normal Form analysis, which is also reviewed there.

Although the sensitization condition introduced seems to satisfy the two criteria that we

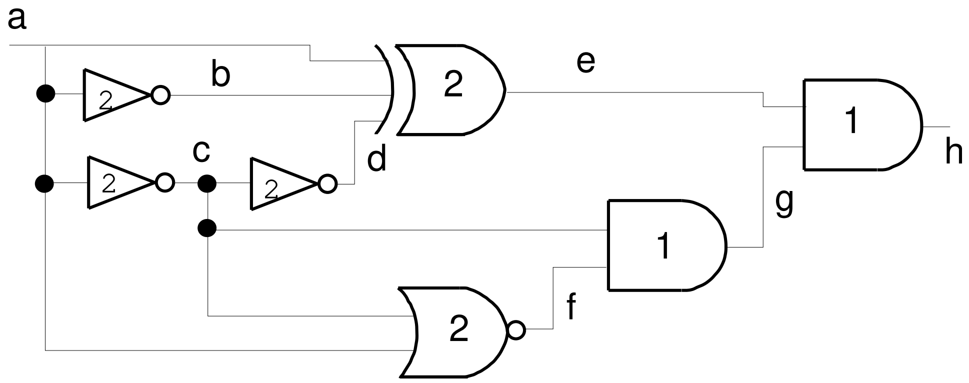


Figure 14: Floating delay sensitization may decide as false a path for which an event can propagate.

had set out to meet, one should not lose sight that they are computed under single-vector operation. As such they are not always guaranteed to compute the exact circuit delay, as was suggested in [25]. In fact, since in floating mode operation pessimistic assumptions are made regarding the arrival times of path side inputs, it is conceivable that the delay computed overestimates the exact delay. To see this consider the circuit in Figure 13 again taken from [31]. Node e cannot be part of a sensitizable path because d either assumes a controlling value with respect to the gates with outputs f or g and has a stable time less than e. Therefore we only need to consider paths containing d. For example, path $\{a, d, f, g\}$ is sensitizable since all the side inputs to the path verify the conditions for floating-mode sensitization (for instance node e assumes a controlling value with a stable time greater than d, which also assumes a controlling value). This would indicate a delay at least of 6. However, no transition can propagate to e, which always assumes value 1. Consequently no path containing both d and f is sensitizable, and the longest sensitizable path in the circuit is $\{a, d, g\}$ with a delay of 4. The sensitization criteria can overestimate the delay of the circuit because the values of the path side inputs prior to the last change is not known and can be set to a controlling value as the previous example shows.

Furthermore, although the sensitization criteria introduced in Definition 4.6 satisfies the correctness criteria, it may however consider as false a path that in fact is sensitizable according to Definition 2.10, but only when the length of that path is smaller than the real delay of the circuit. Consider the circuit in Figure 14 for which path $\{a, b, e, h\}$ is not floating mode sensitizable. To see this a simulation can be performed for both input values. In the case where $a = 0$, then $e = 0(6)$ but $g = 0(3)$ and the gate whose output is h is at a controlling value, but the input corresponding to the path arrives later than the side input, which violates the timing requirement of the sensitization condition. On the other hand, if $a = 1$ then $e = 1(6)$ and now $g = 5(0)$ and the gate whose output is h has a controlling value forced by a side-input to the path which immediately makes the path non-sensitizable. Consequently, no path containing e satisfies the conditions enumerated for floating mode sensitization. However, if we apply a falling transition on a, we can see that the event can be propagated along the path which therefore is sensitizable, in this case with delay 5. This path is not the critical path of this circuit, since path $\{a, c, f, g, h\}$ is sensitizable under our criteria, with delay 6 [6].

[6] Note that in the transition mode of operation with fixed delays, events do not propagate through this

While the floating mode sensitization condition discussed in this paper cannot be considered as accurate as simulating a complete vector set with an electrical or timing simulator, it can produce extremely accurate estimates with astonishing efficiency and makes assumptions at least as accurate as those of the current generation of timing verifiers. Other conditions have been proposed in the literature that compute the same delay value [20, 25]. In the next section we consider the issue of efficiently computing the critical delay of the circuit under the sensitization conditions presented here.

5 Floating Mode Delay Computation

In Section 4.3, we discussed and presented a sensitization condition that represents true floating mode delay. While this condition allows us to decide whether a particular path in the circuit is sensitizable or not, it does not address the question of determining which paths in the circuit should one apply that condition to. The question to be asked now is whether a method exists that can efficiently compute the true critical delay of a circuit.

Most methods to compute critical delay operate on a per-path basis. They are members of a broad generic class of methods parameterized by a sensitization condition and a search method. The sensitization condition defines what a true path constitutes, while the search method expresses itself in the data structures that are maintained and in the termination conditions for that same search. These parameters can be chosen rather independently. The choice of the search method affects the computational complexity of the algorithm while the choice of the sensitization criteria affects its correctness. Generically speaking, a collection of paths $\Pi = \{\pi_1, \cdots, \pi_m\}$ which are known to be true is kept and at each step of the algorithm, one such partial path, π_i is removed from the set and examined. If $\pi_i == \{c_0, g_0, c_1, \cdots, c_k, g_k, c_{k+1}\}$ is a full path (i.e. from a primary input to a primary output) then the search procedure stops to record this fact. If not, then some fanout of c_{k+1} which has not been examined yet is selected and the path is extended to it. The new path is checked for sensitization and if true it is inserted into the set of true paths, otherwise discarded. The procedure continues until a termination criteria is met. The fashion in which the algorithm proceeds and its termination criteria is defined by the search method. Two known methods have become quite popular [16]. These are the best-first and depth-first methods. In the depth first method the partial paths are maintained in a stack and every time a partial path is expanded, a new path is added to the top of the stack. The stack is last-in first-out and therefore that path will be examined next. When a full path is encountered its length is compared to that of the longest true path recorded so far. If it is longer, then it replaces it as the longest known full path; otherwise it is discarded. When the stack becomes empty the procedure terminates and the longest true path recorded is returned. In the best-first method, the partial true paths are kept in a priority queue ordered by the length of their longest extension (called potential full length or esperance of a path in [20]). The termination condition in this case is to reach a primary output, since by construction no longer path can exist. In any case, at any point in the algorithm, given a path, one must determine an input vector that sensitizes the path according to the chosen sensitization conditions. If no vector can be found, the path is discarded and the next path examined. This process is iterated until the longest true path is found. The delay

path and the delay of the circuit is indeed 5. Only under monotone speedup can the path become sensitizable and the delay of the circuit becomes 6.

of this path is reported as the critical delay of the circuit. Determining an input vector for a particular path could take $O(2^n)$ time where n is the number of primary inputs to the circuit and this operation has to be carried out for each chosen path. Many circuits such as parallel multipliers have thousands of longest paths, some of which are true and the others false. Per-path delay computation methods (reviewed in [20]) cannot be used on large circuits.

Clearly alternative strategies have to be developed that avoid per-path enumeration. In [22] a different approach is pioneered in which one attempts to directly answer the question of what the true critical delay of the circuit is and operate on sets of paths rather than a single path at a time. To this end the authors ask and try to answer the following question: "Is the delay of the circuit $\geq \delta$?". The value of δ to use may seem somewhat magical but in fact it is not critical. For instance one could consider initially setting δ to the length of the longest topological path in the circuit. If this path is false, then the answer to our question will be no and one can then progressively decrease δ until the answer to the posed question becomes yes. In other words, the algorithm will start from a large value and proceed to refine δ in order to improve the accuracy of the estimate. The process by which this refinement is controlled is not critical although some thought should be given to it as it will influence the computational effort needed.

It is in fact possible to find the true critical delay of a circuit without explicit path enumeration. A straightforward $O(2^n)$ algorithm for that would be to simulate each of the 2^n input vectors or minterms using the timed calculus introduced in Definition 4.5 and determine the longest delay seen at the circuit output. Although simple this can be time consuming if the number of inputs is large. However, this algorithm can be speeded up considerably using cube simulation rather than minterm simulation. This was already proven in [22] where a testing algorithm modified to include timing information was presented. In the following sections, an alternate algorithm will be detailed that accomplishes that same goal. The algorithm and its derivation can be found in [34, 27]. In section 5.1 we will start by generalizing the timed-calculus introduced in section 4.3 to allow unknown (or 2) values which in turn allows cube simulation instead of minterm simulation. In section 5.2 we will review the implicit enumeration algorithm PODEM [36] and show how one can use the PODEM search strategy in a timed test generation procedure that calculates the true floating delay.

5.1 Cube Simulation

In order to use PODEM or any other cube simulation method for delay computation, we need to generalize the time-calculus introduced in Definition 4.5 to allow logical values, 0, 1 and 2 (unknown). The new time-calculus must have the following properties:

`Completely specified vector:` In the case where inputs are completely specified, it is equivalent to the timed calculus of Definition 4.5.

`Incompletely specified vector:` Given an incompletely specified vector, it produces an upper and a lower bound on the achievable delay over any of the minterms in the vector.

We will derive such a timed-calculus for cube simulation for a two-input AND gate. Equivalent calculus for other gates would be obtained in similar fashion by taking into account the role of controlling and non-controlling values for those gates. Consider then a two-input AND gate for which we want to determine not only the logical value at the output but also lower and upper bounds on the achievable delay. We will assume that the delay of the AND gate, d is fixed. Each input to the AND gate is specified with a logical value in the set $\{0, 1, 2\}$ and a lower and upper bound on the delay corresponding to that value. For input i_1, the lower bound is l_1 and the upper bound is u_1. Similarly for i_2. We will now explain the calculation of the lower and upper bounds for some input combinations. Note since the gate is commutative we do not need to consider all combinations, as some are equivalent.

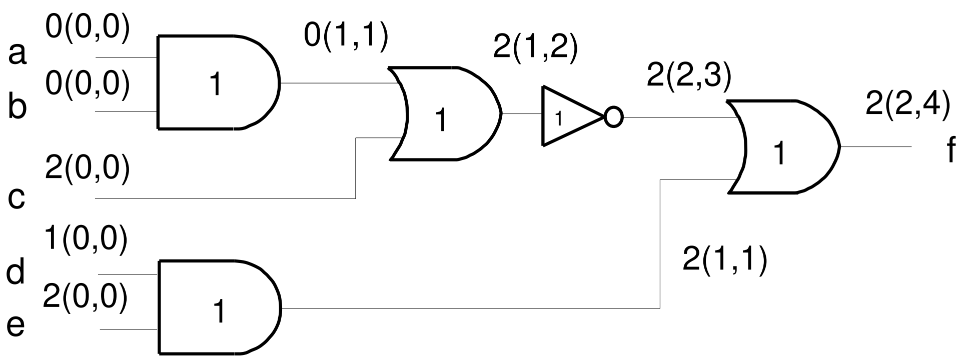
Clearly, when the inputs are in the subset $\{0, 1\}$, i.e., are completely specified, we must follow the rules of Figure 9. For instance when two 0's are present at the inputs of the AND gate, the lower and upper bounds at the output are respectively the minimum of the lower and upper bounds on the inputs. On the other hand, when two 1's are present at the gate inputs we choose the maximum of lower and upper bounds respectively to calculate the lower and upper bound at the output. Finally, when we have a 0 and a 1 we simply use the lower and upper bounds of the 0 input, the controlling value. It is only when the one or both of the inputs are not specified that new values have to be computed. When a 2 is at one of the inputs of the AND gate, the calculus must give the lower and upper bounds on the achievable delay with either (0 or 1) setting of the 2 value. For example, consider that one of the inputs, say i_1 is a 1 and i_2 is a 2. Immediately, we know that the logical value at the AND gate output is a 2. If i_2 were set to 0, the lower bound on the delay will be $l_2 + d$ and the upper bound will be $u_2 + d$. However, if i_2 were set to 1 the lower bound on the delay will be $MAX(l_1, l_2) + d$ and the upper bound will be $MAX(u_1, u_2) + d$. Our calculus must take into account both of these possibilities, but since $l_2 \leq MAX(l_1, l_2)$, l_2 represents the lower bound on the achievable delay for any setting on i_2, and similarly since $MAX(u_1, u_2) \geq u_2$, $MAX(u_1, u_2)$ also represents the upper bound on the achievable delay for any setting of i_2. Now consider the case where i_1 is a 0 and i_2 is still a 2. In this case, the value and the upper bound of i_1 are passed through to the output. To see this note that if i_2 were set to 1, its delay will not matter. On the other hand, if i_2 were set to 0, then the upper bound on the delay at the output will be $MIN(u_1, u_2) + d$. However, in this case $u_1 \geq MIN(u_1, u_2)$ and therefore we can safely pick $u_1 + d$ which satisfies both settings of i_2. By the same reasoning, the lower bound will be $MIN(l_1, l_2) + d$. The only combination left is when both inputs are 2's, in which case the output value will again be a 2, the lower bound on the achievable delay at the output is $MIN(l_1, l_2) + d$ and the upper bound is $MAX(u_1, u_2) + d$.

The timed calculus for cube simulation thus derived is given in Table 1:

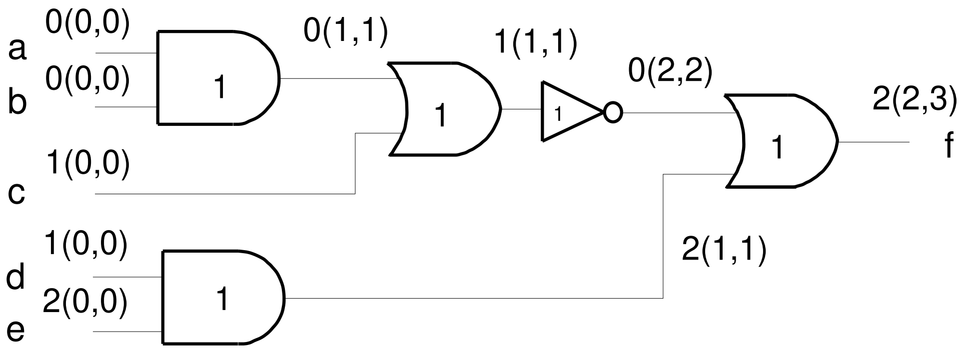
To exemplify a cube simulation using the timed calculus consider the circuit in Figure 15. Each connector is specified with a logical value and two delay values. For example, $0(1, 2)$ implies that the logical value of the wire is 0, the lower bound on the achievable delay is 1 and the upper bound is 2. The input vector to the circuit in Figure 15(a) has two 2 values implying that the values of those inputs are unknown. The output of the circuit is 2, and for the minterms that are contained in the input vector 00212, the maximum achievable delay is 4. When c is set to 1, as in Figure 15(b), the maximum achievable delay at the output is reduced to 3.

$i_1 \rightarrow$ $i_2 \downarrow$	0	1	2
0	0 $MIN(l_1, l_2) + d$ $MIN(u_1, u_2) + d$	0 $l_2 + d$ $u_2 + d$	0 $MIN(l_1, l_2) + d$ $u_2 + d$
1	0 $l_1 + d$ $u_1 + d$	1 $MAX(l_1, l_2) + d$ $MAX(u_1, u_2) + d$	2 $l_1 + d$ $MAX(u_1, u_2) + d$
2	0 $MIN(l_1, l_2) + d$ $u_1 + d$	2 $l_2 + d$ $MAX(u_1, u_2) + d$	2 $MIN(l_1, l_2) + d$ $MAX(u_1, u_2) + d$

Table 1: Timed Calculus for and AND gate with inputs in the set $\{0, 1, 2\}$.



(a)



(b)

Figure 15: Cube simulation using timed calculus.

Algorithm 5.1 (Podem Algorithm)

```
podem(po, lvalue)
{
   jlist = po with logical value lvalue;

   status = search_1(jlist);
   return(status);
}
```

It should be emphasized that the computed delays using the timed calculus merely give the range of the achievable delays over all the minterms contained in the partial input setting (input cube). The delay of a connector can in fact be a range rather than a constant even in the case when the logic value in that connector is known under some partial input setting v. A simple example of this is to consider a two-input AND gate with zero delay having an input, i_1 at 0 with $l_1 = u_1 = 4$. If the other input, i_2 is a 2 with $l_2 = u_2 = 3$, then the lower bound on the output is achieved when the second input is set to a 0; we obtain a delay of 3 at the output. The upper bound of 4 is achieved when the second input is set to a 1. In the following section we will briefly review the PODEM algorithm and then show how the timed-calculus just described can be used for cube simulation with PODEM.

5.2 The PODEM Algorithm

In this section we will start by reviewing the standard PODEM algorithm. Then we will show how to incorporate the timed-calculus just derived in order to use the algorithm for floating delay computation.

5.2.1 Timing-independent PODEM

PODEM is an algorithm that implicitly and exhaustively enumerates the input space using cubes rather than minterms in an effort to satisfy a given objective. In our case, the objective will be to justify a particular value (0 or 1) at the primary output of a circuit. The top-level **podem** routine is shown in Algorithm 5.1. It simply places the given primary output line `po` on a justification list `jlist` with an appropriate value, say 1, and calls a search procedure **search_1**.

The search procedures are described in Algorithms 5.2. There are two search procedures called **search_1** and **search_2**. Initially all primary inputs are unknown. Attempting to justify our objective, the procedure **search_1** calls a **backtrace** procedure which begins at the primary output `po` and tries to find a primary input whose logical value is currently unknown. Once such an input is found that input is set to a 1 value and an implication procedure, **imply** is called. The implication procedure in PODEM corresponds to standard three-valued cube simulation (without any delay information) which tries to justify the current output. Logical conflicts may result from the implication procedure. If no conflict occurs, **search_1** is called recursively. The procedure terminates successfully in **search_1** if the justification list is empty. If a logical conflict is produced in **search_1**, it was caused by the latest input setting. In that case we backtrack to the most recent primary input setting, change that input to a 0 value and call the **search_2** procedure shown in Figure 5.2. At this point failure may result if either the **backtrace** procedure is unable to find a primary input to set, or if the space has been completely enumerated without success in **search_2**. Either way, if conflict or failure is detected, the state of the network is restored to what it was immediately prior to the primary input setting that caused this failure.

We will illustrate the PODEM algorithm with a simple example. Consider the circuit in Figure 16 for which we wish to find an input vector that sets the output to a 1. We begin with all the inputs set to unknown values, $a = b = c = 2$. Simulation or implication of these input values results in a 2 on all wires including the output. The algorithm then backtraces all the way to the input lines and a is set to 1. Forward implication immediately sets the output to a 0 as shown in Figure 16(a). Since our objective is a 1 at the output, this corresponds to a conflict. The algorithm then backtracks again to the most recent primary input setting and changes it, which in this case leads to setting a to 0. But implication of the input vector $\langle a, b, c\rangle = 022$ results in a value of 2 for the output. We therefore backtrace looking for the next primary input that has not yet been set. This happens to be b which is set to a 1. Implication of the input vector $\langle a, b, c\rangle = 012$ leads to another conflict in the value of the output as illustrated in Figure 16(b). To try and resolve the conflict again we backtrack to the most recently set primary input, in this case b and change its value to 0. Simulation of the vector $\langle a, b, c\rangle = 002$ results in an output of 2 as shown in Figure 17(a). The procedure is then recursed setting the remaining unknown input, c to 1 which then sets the output to the required value of 1, as shown in Figure 17(b). The search procedure ends in success. Note that in this case only 6 input vectors were simulated while the input space is of size 8.

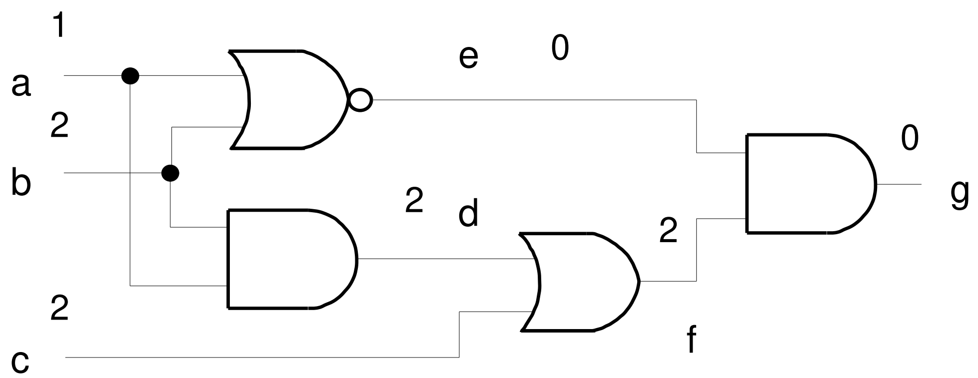
The flow of control in the PODEM algorithm can also be understood with the help of a binary decision tree. For the circuit in Figure 16 the binary decision tree corresponding to the application of the algorithm is shown in Figure 18. Each node in the tree corresponds to a primary input, and the left and right edges from each node correspond to input settings of 1 and 0, respectively. The leaves of the tree correspond to primary output settings. The algorithm starts at the top of the tree and descends through its left edge. If a conflict is obtained, the algorithm backtracks by going up the tree from the edge it last descended up to the node corresponding to the last input setting. Then the algorithm descends through the right edge of that node which corresponds to a change in the value of that input. There is also a simple correspondence between traversing the tree and procedure invocation. For example, descending through the left edge of a node is equivalent to a call to **search_1**, while descending through the right edge is equivalent to a call to **search_2**. Every time one descends through either edge, a call to **imply** is made to determine the value imposed at the output. Similarly going up the tree is equivalent to backtracking and restoring the state of the network to what it was prior to the most recent input assignment.

5.2.2 Timed Test Generation in PODEM

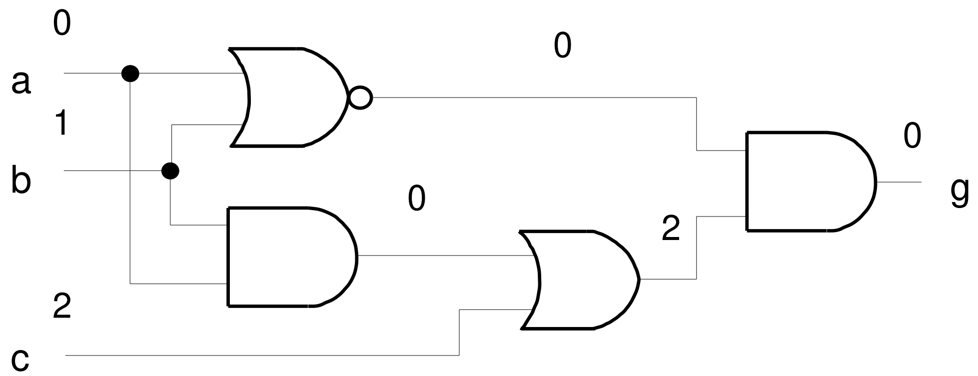
In this section we will show how to apply the PODEM algorithm just described to the problem of floating delay computation. In order to use the PODEM algorithm for floating delay computation, we have to use the timed calculus of Table 1 in the implication procedure.

Algorithm 5.2 (Search Procedures for PODEM)

```
search_1(jlist)
{
   if(length of jlist is zero) return SUCCEED;

   if (backtrace(po, po_value, &pi, &pi_value) == FALSE)
      return(FAILED);

   if (imply(pi, pi_value, jlist) != IMPLY_CONFLICT) {
      search_status = search_1(jlist) ;
      if (search_status == FAILED) {
         restore the state of the network to what it was prior to
           the most recent primary input assignment ;
         search_status = search_2(jlist, pi, 1 - pi_value);
      }
   } else {
      restore the state of the network ;
      search_status = search_2(jlist, pi, 1 - pi_value) ;
   }
   return(search_status);
}


search_2(jlist, pi, pi_value)
{
   backtracks = backtracks + 1 ;
   if (backtracks > BACKTRACK_LIMIT)
      return(ABORTED);
   if (imply(pi, pi_value, jlist) != IMPLY_CONFLICT) {
      search_status = search_1(jlist);
      if (search_status == FAILED)
         restore the state of the network;
   } else {
      search_status = FAILED;
      restore the state of the network;
   }
   return(search_status);
}
```



(a)



(b)

Figure 16: PODEM algorithm example.

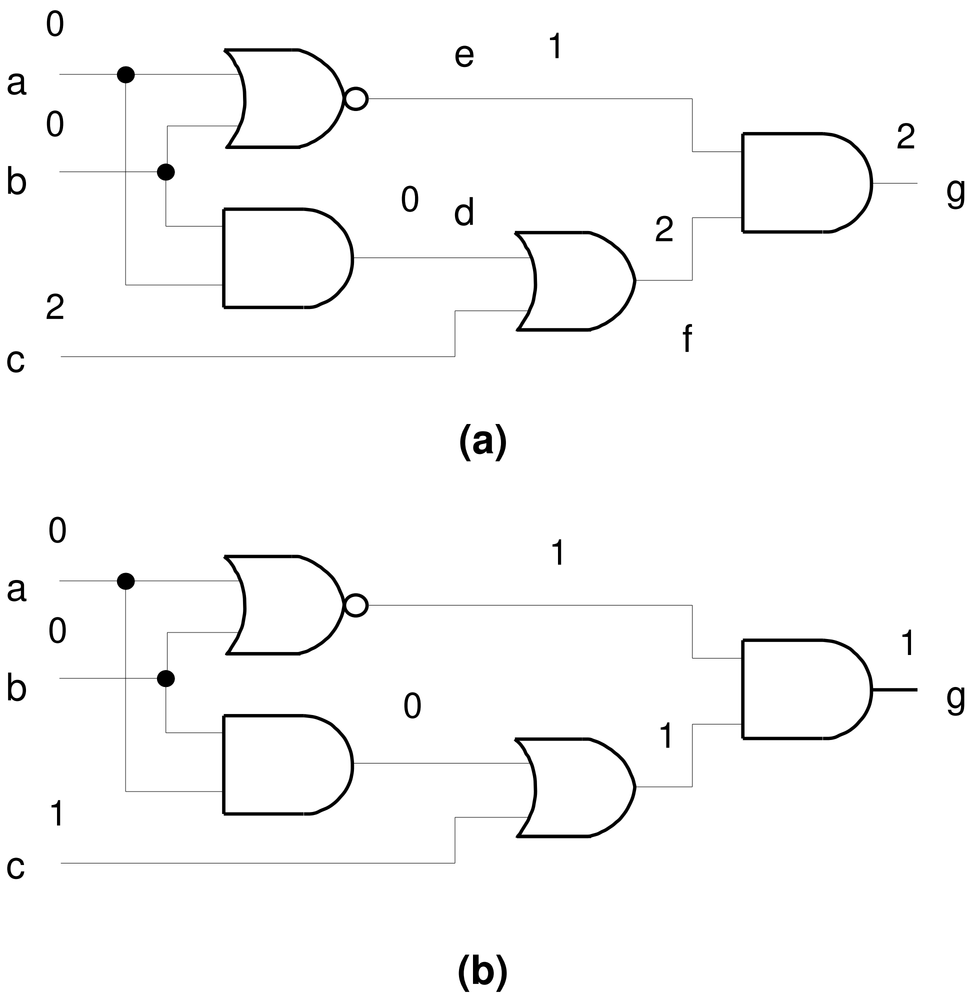


Figure 17: PODEM algorithm example (continued).

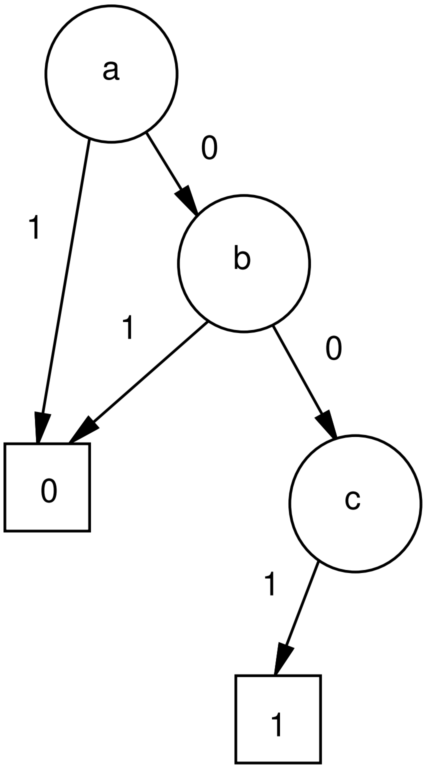


Figure 18: Binary decision tree for PODEM.

We will be attempting to justify both a logical value on a wire as well as a delay value, and therefore the procedure is termed *timed test generation* [37].

The timed test generation procedure is given a logical value $L \in \{0, 1\}$ and a number δ, and asked to find an input vector, if such a vector exists, which sets the output of the circuit to L and which results in a floating mode delay $\geq \delta$.

In general terms, the timed procedure follows the same steps as the standard PODEM algorithm. However, since both logic and delay values are now being considered, some differences arise, mainly in the implication procedure. Rather than a purely logical calculus, the implication procedure now uses the timed calculus of Table 1. Therefore conflicts occurring during implication may now be logical conflicts or time conflicts. As before, logical conflicts correspond to the case when the output is set to the value $\overline{L}$. A time conflict occurs when the output is set to L but the upper bound on the delay at the output is strictly less than δ. In this case too the algorithm has to backtrack since we know that we cannot find an input vector within the cube corresponding to the current settings of the primary inputs that has a delay $\geq \delta$. The procedure ends successfully if the output has been set to L and the lower bound on the computed delay at the output is $\geq \delta$, i.e., both the logical and the delay value have been justified.

Consider the circuit of Figure 19 which is replicated from Figure 16. In this case however, not only logical values are indicated, but delay values are also used. Our objective is to justify a 1(3) at the output of the circuit. The delay of each gate is indicated inside the gate. Initially, all the inputs are set to 2 values with zero delay. Implication results in a 2 logical value at the output with a delay range $(2, 3)$ as shown in Figure 19(a). Backtracing

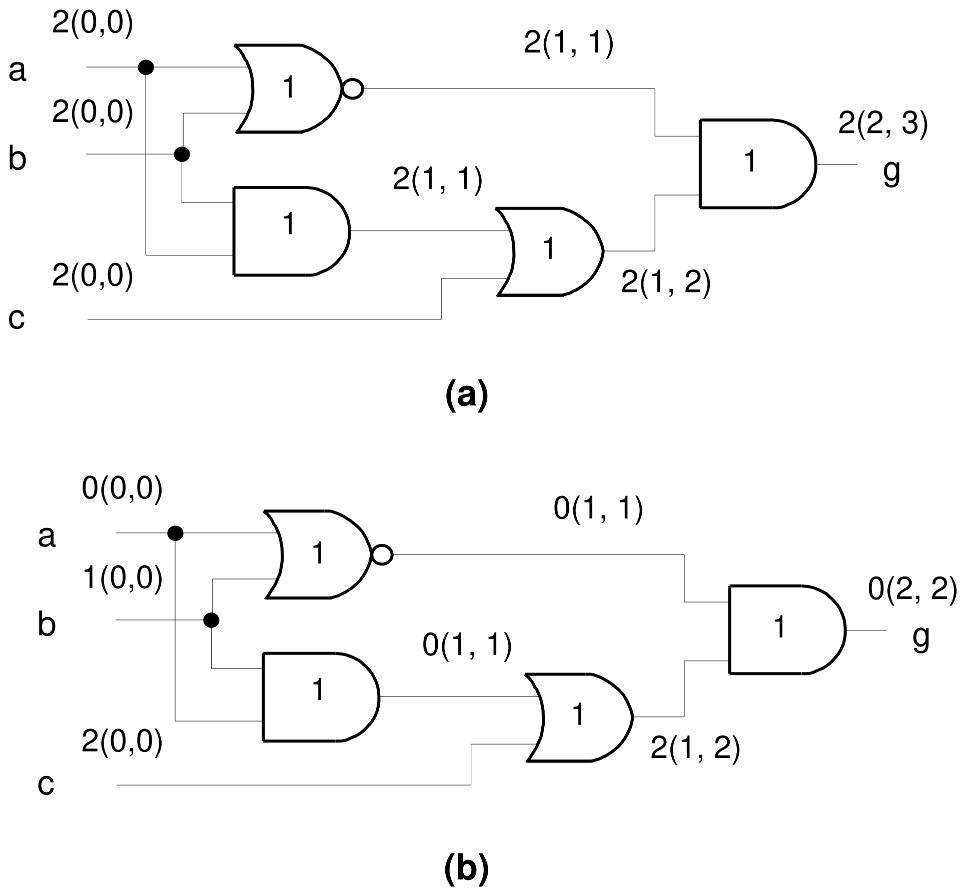


Figure 19: Timed test generation example.

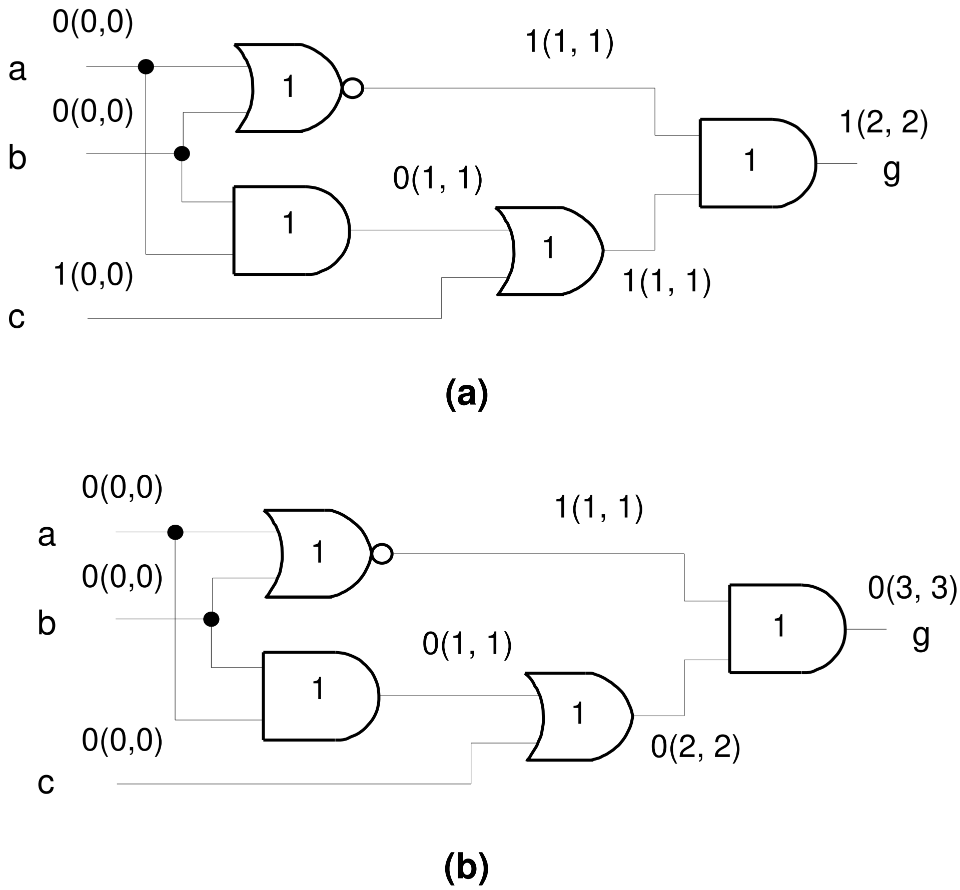


Figure 20: Timed test generation example (continued).

sets a to 1 and implication immediately sets the output to 0 which is a logical conflict. We backtrack to the most recently set primary input, a and change its value to 0. Forward implication in this case results in a 2 at the output with a delay range of $(2, 3)$. Backtrace sets b to 1, resulting in a 0 at the output, a logical conflict, as shown in Figure 19(b). We backtrack to set b to 0. The output is unknown with the input setting corresponding to $\langle a, b, c\rangle = 002$. Backtracing now and setting c to 1 results in the output being set to 1, but with a delay of $(2, 2)$, as shown in Figure 20(a). We note at this time that the logical value at the output is the one we want to justify, but there is a time conflict since the upper bound on the delay is strictly less than the required delay. So again we backtrack, finally setting c to 0 which results in a 0 at the output, a logical conflict, as shown in Figure 20(b). At this point we have tried cubes covering all possible input combinations and we have failed to find an input vector that results in a 1 at the output of the given circuit with floating mode delay 3. Note that in order to reach this conclusion we did not have to simulate all of the 2^3 different input minterms. We must conclude that it is not possible to justify a 1(3) at the output. However, it is possible to find an input vector that results in a delay of 3 that produces a 0 at the output, as shown in Figure 20(b). Therefore we conclude that the floating mode delay of this circuit is 3.

In our discussion of the PODEM algorithm so far we have failed to discuss much of the **imply** and **backtrace** procedures. The **imply** procedure is a simple cube simulation given the current input settings and really needs no further explanation. The **backtrace** procedure however, deserves some further study. In PODEM, the backtrace procedure is called when an implication of the current input settings leads to an unknown value at the the primary output of the circuit. The backtrace begins at the circuit's output and traces back through the gates in the circuit to a primary input that is still at the unknown value. The gates traversed all have the property that their outputs are at unknown values. Before backtracing, the output is set at the required value which by itself may dictate the required value at these gates. However in general there will be choices as to which path the backtrace follows. Since there is no way to determine the best choice, the backtrace procedure uses heuristics in following paths that begin from a primary input with an unknown value which when set will likely set the output to the desired value. The particular value that the primary input is set to, either 0 or 1 can also be decided heuristically. Similarly to what we saw happening to the implication procedure, the backtrace in the timed test generation method is similar to the purely logical backtrace of PODEM except that it uses both the logical and desired delay value at the output to choose what path to follow. In this case, the backtrace procedure may be called when the primary output is at the 2 value, or when the lower and upper bounds on the computed delay at the output are not equal. This is illustrated in Figure 21, where we show a fragment of a circuit with a partial setting of its primary inputs. Logical values and upper bounds on the delay value for each wire are shown in the figure. Assume that the desired value at the primary output is 1(10), shown in bold, and that under the current input setting cube simulation has led to a value of 2(11) at the output.

In this case backtracing begins at the OR gate connected to the primary output. Since we want to justify a 1 at the primary output and the other two inputs to the OR gate are 0, we can infer that we require a 1 at the first input to the OR gate. Furthermore, by taking into account the gate delay, the delay value required at the first input is 9. Backtracing continues and the AND gate is now examined. We know that we require a 1(9) at its output, as shown in bold in the figure. This means that all of its inputs have to be a 1. The first

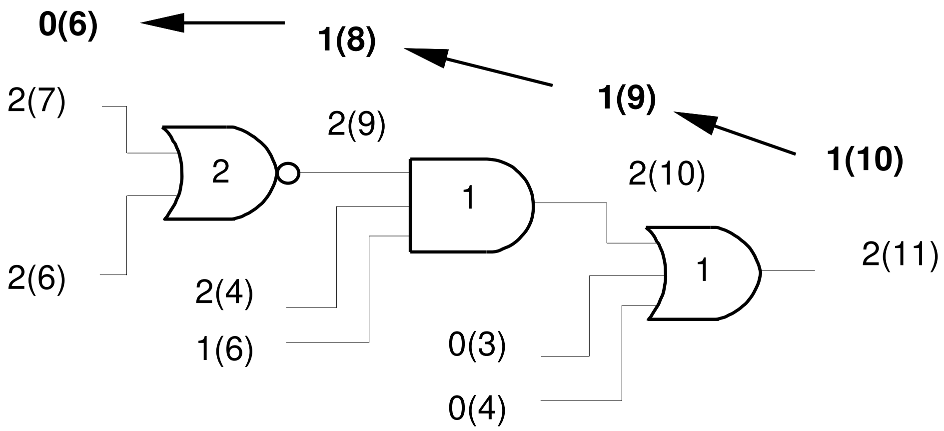


Figure 21: Backtrace example.

and second inputs to the AND gate are at 2's and therefore a choice must be made at this point. We will choose to follow the path corresponding to the first input because that is the only path that can satisfy the delay value of 9 at the AND gate output. For that a value of 1(8) is required at the first input which is the output of the NOR gate. Both inputs of this gate are at unknown values, and it is possible for either input to provide a 0(6) value. The backtrace procedure randomly selects one of the inputs and continues till it reaches a primary input.

6 Three Timing Analysis Papers

This section is a brief summary of three important papers in the field, indicating their contributions, and relating them to the description in the previous sections. For the most part we will try to summarize the ideas in the papers and leave out the details of any implementation or algorithm. Whenever appropriate we will refer to the sections on this paper that deal or present the same topic.

6.1 McGeer and Brayton

The first paper we discuss is entitled "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network", authored by Patrick C. McGeer and Robert K. Brayton and was presented at the 26^{th} ACM/IEEE Design Automation Conference in 1989 [20]. This paper considers the elimination of false paths in combinational circuits. An example with a false path is presented to motivate the discussion. This paper classifies all timing algorithms in a generic, parameterized class. According to the authors a set of partial true paths is maintained and at each step of the algorithm, one element P of the of the set is picked and checked whether it terminates in a primary output. If so, the path is a so-called full path, otherwise it is expanded along some unexamined fanout g, as long as a boolean condition $\gamma(P, g)$. If $\gamma(P, g)$ is satisfied, then $\{P, g\}$ is a true path and added to the collection. The procedure then continues until a termination criteria is met.

This paper takes a procedural approach to the critical path problem. A generic procedure is used that takes as inputs a search method and a definition of what constitutes a true path, that is, sensitization conditions. The preferred search method seems to be best-first and the authors use the correctness and robustness criteria which we introduced in Definitions 2.18 and 2.19 and make an analysis of four sensitization algorithms based on these criteria. It is instructive to follow this discussion because it introduces some very interesting ideas. The first of this is the notion of boolean difference which in turn depends on the definition of co-factor.

Definition 6.1 (Co-Factor of a Function with respect to an input)
Given a boolean function f and an input x, the co-factor of f with respect to x is the function f_x such that $f_x = f$ if $x = 1$.

If f is written in sum-of-products form the co-factor of f with respect to x will consist of the sum of the cubes containing x evaluated for $x = 1$.

Definition 6.2 (Boolean Difference)
The boolean difference of a function f with respect to an input x is the function:

$$\frac{\partial f}{\partial x} = f_x \oplus f_{\bar{x}}$$

Simply put this means that for the boolean difference to be satisfied, a change in x must translate directly into a change in f. In some sense the boolean difference is the boolean counterpart of a derivative.

Definition 6.3 (Smoothing Operator)
The smoothing operator is a boolean operator whose application to a function results in the set of points where the functions is satisfied regardless of a given condition.

For instance the set of points where $f = 1$ assuming $x = 1$ is, as we know, f_x. Similarly, the set of points where $f = 1$ when $x = 0$ is $f_{\bar{x}}$. Therefore, the set of points where $f = 1$ regardless of the values of x is given by the application of the smoothing operator to f with condition x, that is:

$$\mathcal{S}_x f = f_x + f_{\bar{x}}$$

In the sum-of-products representation, $\mathcal{S}_x f$ is equivalent to deleting every occurrence of x and $\bar{x}$.

With this notation the authors proceed to define four sensitization conditions. Static sensitization which we already described in section 4.1, follows from the observation that for a path $P = \{c_0, g_0, c_1, \cdots, c_n, g_n, c_{n+1}\}$ to be true, an event must be able to propagate through the path. In other words, changing the values of c_i must change the value of c_{i+1}, that is $\frac{\partial c_{i+1}}{\partial c_i}$ must be satisfied and this must be true for all connections in the path. Static sensitization can then be written formally as a single condition:

$$\gamma(\{c_0, g_0, c_1, \cdots, c_n, g_n, c_{n+1}\}) = \prod_{i=0}^{n} \frac{\partial c_{i+1}}{\partial c_i}$$

Trying to avoid the delay-independence characteristic of static sensitization, the paper then quickly reviews a sensitization condition named dynamic sensitization. The idea behind dynamic sensitization is that the boolean difference computed at each gate in the path need not be satisfied statically and instead only has to be satisfied at the time the event propagates through each gate. This implies a two-vector condition with v_1 applied at $t = -\infty$ and v_2 applied at $t = 0$. This can be written as

$$\frac{\partial c_{i+1}}{\partial c_i}(v_1, v_2, \tau_i) = 1$$

where τ_i is the computed time at which the event arrives at the gate whose output is c_{i+1}. This sensitization condition turns out to be a correct but not robust criteria and an example is shown to prove that point.

By far the most important contribution of this paper is the notion of *viability*, which is defined as follows:

Definition 6.4 (Viable Path)
A path $P = \{c_0, g_0, c_1, \cdots, c_n, g_n, c_{n+1}\}$ is **viable** *under an input vector v if and only if for each c_{i+1} and each side input $g \neq c_i$ to c_{i+1} either:*

1. *g is set to its sensitizing value by v, or*
2. *g terminates a viable path under v of length at least the length of the partial path $\{c_0, g_0, \cdots, c_i\}$.*

Viability can be shown to be both a correct and robust sensitization condition. For networks of simple gates in [30] it is shown that this condition is equivalent to stating that for a path to be viable, all side inputs either have a non-controlling value or have a stable time greater than or equal to the arrival time of the event through the path. In [25] it is shown that both the true floating mode criteria and viability compute the same circuit delay. Given the previous definition the authors proceed to introduce a viability function such that the viability of a path becomes equivalent to satisfying that boolean function. This function is quite complicated as it involves computing the viability of side-paths, which can be updated in the search procedure as each path is extended. For simplicity we will leave out the detailed description of the function.

A forth sensitization condition, originally presented in [19] is also discussed. Interestingly enough this condition is very similar to viability and in fact the authors claim it to be an approximation that will yield a path no shorter and possibly longer than the viability condition. This approximation is based on an ordering of the inputs at each gate and a simplification whereby only inputs up to the path connector in that ordering are considered in the computation of the viability function. The quality of the delay estimate is therefore critically dependent upon that ordering.

The second part of McGeer and Brayton's paper is concerned with the computation of the delay of a circuit using the viability condition. For that purpose a dynamic programming procedure is described, which itself is based on a best-first search method. This dynamic procedure is aimed at improving the efficiency of the computation of the viability condition along every side path to the path under consideration. A very detailed description of the algorithm is given but we will not repeat it here.

The paper ends with an interesting description of an implementation of a timing environment within a logic synthesis system. This timing environment is said to be parameterized on five axes: search method (best-first or depth-first), sensitization criteria (static, viability, etc), satisfiability test (to determine whether the sensitization function is satisfied), representation of function (graph-based, versus sum-of-products) and delay model. The selection of parameters is made by the user which has therefore many degrees of freedom. A large number of experimental results is shown that seems to indicate that the viability conditions lead to accurate estimates of the delay. Surprisingly enough on the majority of these experiments static sensitization computes the same delay as viability which leaves some doubts about the relative improvement in accuracy versus the computational cost of the more complicated viability function.

Overall this paper is extremely important because it points out the important issues in timing verification. It also manages to introduce a framework upon which known sensitization criteria can be compared on their own merits, independently of the implementation and the environment in which they are used.

6.2 Chen and Du

The second paper we will discuss is entitled "Path Sensitization in Critical Path Problem", authored by Hsi-Chuan Chen and David H. C. Du and was presented at the International Conference for Computer-Aided Design of VLSI Circuit in 1991 [32]. This paper is an improved, albeit short version of [25], which was later expanded into [38]. The main focus of this paper lies in the determination of correct sensitization conditions for timing verification. The authors main concern relates to correctness, in the sense that the critical delay should not be overestimated. Robustness is not an issue and the fixed component delay model is used throughout. As in the previous paper, this paper also introduces a framework such that various path sensitization criteria can be compared in a unified way.

The approach followed in this paper is topologically-based in the sense that the sensitization criteria deals with unions and intersections of sets based on the connections between the gates in the circuit. To understand their approach we start with some notational definitions. We should add at this point that we felt that the notation chosen is somewhat awkward and verbose which leads easily into confusion.

The `controlling value` to a gate g_i is denoted $C(g_i)$ while the `non-controlling value` is denoted $NC(g_i)$ and is of course the complementary of the former. Their definitions can be seen in Definition 2.16. A few more definitions are needed as follows, where for consistency we make use of the terminology "connector", instead of "lead" which is used in Chen and Du's paper.

Definition 6.5 (Stable Value and Stable Time)
The logic values stabilized at a gate input c_i and output c_{i+1} under an input vector v are called the **stable values** *of c_i and c_{i+1} under v and are denoted by $sv(c_i, v)$ and $sv(c_{i+1}, v)$. The times when c_i and c_{i+1} become stable under v are called the* **stable times** *of c_i and c_{i+1} and are denoted by $st(c_i, v)$ and $st(c_{i+1}, v)$.*

Definition 6.6 (Dominate)
Given an input vector v and a gate g_i of which the connection c_i is an input, the connection c_i is considered to **dominate** *g_i under v if either of the following conditions is satisfied:*

1. *c_i is the earliest controlling input*
2. *c_i is the latest non-controlling input, given all its side inputs are non-controlling inputs*

As we can easily see, this condition is the same as the one proposed in Definition 4.6.

Definition 6.7 (Activate)
Partial path $P_k = \{c_0, g_0, \cdots, g_{k-1}, c_k\}$ is defined to be **activated** *by input vector v if each on-input c_i of P_k dominates its succeeding gate g_{i+1} under v.*

Activation of a path is therefore equivalent to the condition that an event be propagated through the path. Therefore with no surprise, the authors define sensitization of a path as:

Definition 6.8 (Sensitization of a Path)
A path is sensitizable if there is at least an input vector to activate it.

According to this criteria, determining the sensitizability of a path is equivalent to determining the existence of primary input vectors that can sensitize the path. This is done by computing the set of primary inputs vectors $SENV(P)$ that sensitizes each path P. To help in this endeavor the following sets are defined:

- The set of primary input vectors allowing c_i to be a controlling (non-controlling) input is defined as $CV(c_i)$ ($NV(c_i)$) respectively.
- The set of primary input vectors allowing c_i to be a controlling value with stable time no earlier than time t is defined as $GEV_c(c_i, t)$. The set of primary input vectors allowing c_i to be a non-controlling value with stable time no later than time t is defined as $LEV_n(c_i, t)$.

The determination of whether a given path P is sensitizable follows from computing these sets for subpaths of P and using set arithmetic to arrive at the appropriate condition. In order not to confuse the description we will skip this step entirely. For details, see either [32] or [38]. The authors also claim that as long as a path sensitization criteria is only concerned with critical paths the requirement on being the latest non-controlling input to arrive can be relaxed. Instead it is enough to require all other inputs to be non-controlling as well. This allows them to loosen the criteria which has computational implications. The path sensitization criteria introduced in this paper is claimed to be exact, i.e. to achieve 100% accuracy, which is incorrect as we known that it can overestimate the delay of some circuits.

The last part of the paper is spent comparing various sensitization criteria under the new-developed framework. The first criteria to be studied is from [16], here named $BenV$, which to no surprise is shown to possible underestimate the delay. A criteria presented in [21], and referred here as $DuYenV$ which uses precomputed timing information regarding the length of the longest and shortest paths from the primary inputs to any connector in the path is reviewed. It is shown that it can possibly overestimate the delay. Another criteria, initially presented in [39] and named $PerrV$ also uses precomputed data and is also shown to overestimate the delay. This criteria maintains for each connector an upper bound on the latest stable time of that connector. This value is decreased whenever there is a sufficient knowledge that no sensitizable partial path terminating at that connector exists with length equivalent to the current bound. Finally viability, here referred to as $ViableV$ is also compared which is known to compute the same delay as the sensitization condition presented in this paper [25]. The paper ends with a very interesting summary which shows the relative merits of each sensitization condition with respect to both the paths that it sensitizes and the delay estimates it returns. The summary is as follows:

- Regarding the sensitization of a path P:

$$BenV(P) \subseteq SENV(P) \subseteq SENV_{loose}(P) \subseteq ViableV(P)$$
$$SENV(P) \subseteq DuYenV(P)$$
$$SENV_{loose}(P) \subseteq PerrV(P)$$

- Regarding the delay estimate:

$$BenV \leq SENV = SENV_{loose} = ViableV(P) \leq DuYenV, PerrV$$

No mention is made in this paper of computational or implementation issues. That discussion can be found in the longer version [38]. The importance of this paper is related to the sensitization criteria here developed. As in McGeer and Brayton, Chen and Du also introduce a framework upon which known sensitization criteria can be compared on their own merits.

6.3 Devadas, Keutzer and Malik

The third paper we study is entitled "Delay Computation in Combinational Logic Circuits: Theory and Algorithms", authored by Srinivas Devadas, Kurt Keutzer and Sharad Malik and was presented at the International Conference for Computer-Aided Design of VLSI Circuit in 1991 [22]. There are two issues pursued in this paper. The first one is to derive a set of necessary and sufficient conditions for a path to be true. The second one is to develop a delay computation algorithm that handles entire sets of paths simultaneously. Special relevance is given to this second issue and most of the paper is dedicated to it and deservingly so because of the two it is the one to which the least research had been dedicated before.

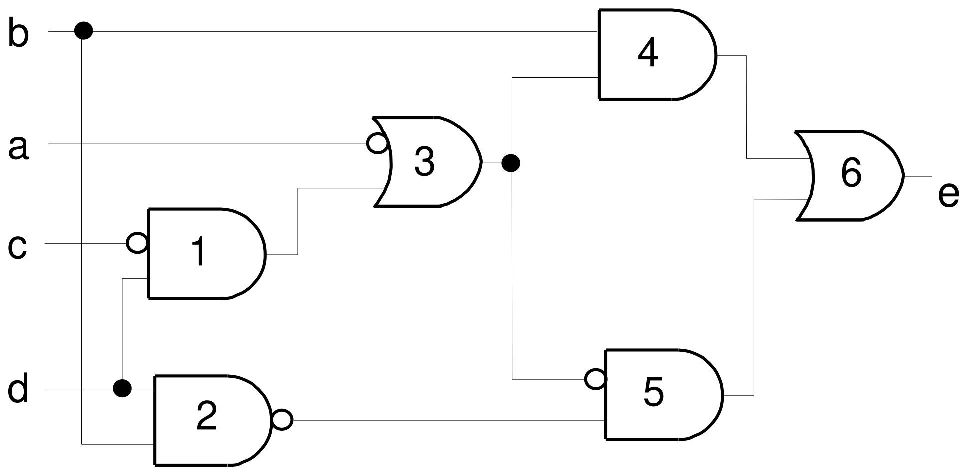


Figure 22: Example circuit for computing the ENF.

The approach presented in this paper to analyze the boolean and temporal behavior of logic circuitry is to use a representation called the *equivalent normal form* or ENF [40]. The ENF of a multi-level circuit circuit is a two-level representation that retains path information as well as logic functionality. Computing the ENF of a circuit is a simple matter. All it involves is to unfold the circuit and make it internal fanout free, which may involve duplicating gates and then using De Morgan's laws, push the inverters back to the primary inputs. For simplicity Devadas' paper handles only simple gates, AND, OR and NOT. This results in a leaf-directed acyclic graph as a circuit representation and the ENF is then constructed starting from the primary inputs of the circuit. For instance, the ENF expression corresponding to the output of an AND gate g with inputs a_α, b_β is a sum-of-products expression over tagged literals, $E_g = a_{\alpha \cup \{g\}} \cdot b_{\beta \cup \{g\}}$. It is instructive to consider an example and see what the result of computing a circuit's ENF is.

Consider then Figure 22 for which the ENF for primary output e is required (the numbers on the gates are identifiers). From Figure 23 where the appropriate transformations have been performed (gates keep the number of those they are duplicates of) it is easy to see that the the ENF is:

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

The example illustrates that 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. For each path π in the circuit there is a tagged literal l_π in the ENF. l is a literal of the input variable of π and its tag is the set of gates along π. As an example, consider the path from b through gates 4 and 6 to e. 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. Furthermore, the ENF preserves the timing behavior of a circuit which can be proven with the help of the two-level representation based on the

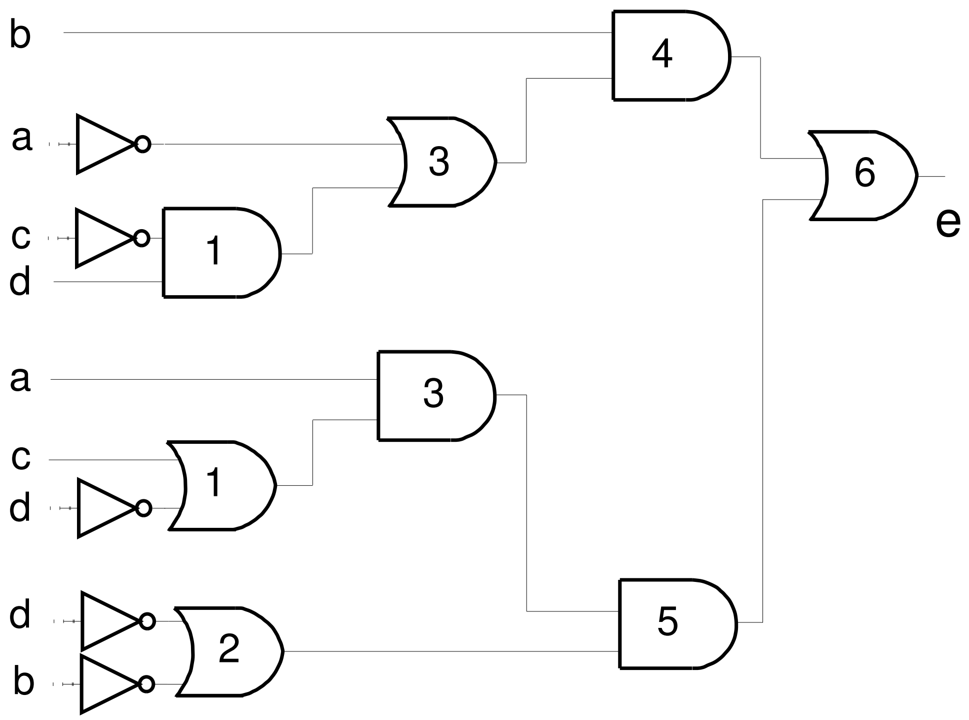


Figure 23: Example circuit after unfolding and pushing the inverters back to the inputs.

ENF. In the floating mode of operation, using the ENF as a circuit description, after the application of the single-input vector, different cubes will settle to their final values at different times, and the output will settle as soon as the first cube settles to its final value. Thus, the delay for of the circuit is determined by the time taken for the first cube to settle. From this condition the authors note that for a path π to be true, there must be some cube q containing the tagged literal l_π that evaluates to 1 for some vector v. This leads then to a necessary condition which is exactly the same as encountered for static co-sensitization. At this point the authors compare this condition with that of viability [20] and note that viability does not imply static-co-sensitization which may lead a path to be considered viable even though it is false, which can be wasteful. Nevertheless the authors quickly note that even though viability may incorrectly identify a path as being false, the delay computed is always correct (i.e. it satisfies the correctness criteria). Since static co-sensitization is simply a necessary condition, the authors then derive sufficient conditions for sensitization. These conditions, along with a topological interpretation are similar to those of Definition 4.6. This sensitization condition can be directly applied to any path in the circuit. In this paper, Devadas *et al.* extend this to set of paths. For this, they consider sets $\Pi = \{\pi_1, \cdots, \pi_m\}$ containing all paths of length at least δ. The idea is to determine necessary and sufficient conditions for there to be at least one true path in the set. The following theorem gives the necessary result.

Theorem 6.9 ()
Let C be a circuit C with ENF E and $\Pi = \{\pi_1, \cdots, \pi_m\}$ the set of all paths of length at least δ. At least one π_i is true for the rising transition if and only if there exists a vector v such that C settles to a 1 on v and for all cubes q for which $q(v) = 1$, there exists some l_{π_i} satisfying $l_{\pi_i} \in q, \pi_i \in \Pi$.

The authors then proceed to make a connection between their condition and testing which is expressed in another similar theorem. We include it for comparison.

Theorem 6.10 ()
Let C be a circuit C with ENF E and $\Pi = \{\pi_1, \cdots, \pi_m\}$ the set of all paths of length at least δ. At least one π_i is true for the rising transition if and only if there exists a test for the multifault $\{l_{\pi_i}, stuck - at - 0\}$.

The remaining part of the paper consists in the development and study of an algorithm for delay analysis whose origins are in the testing field. The algorithm works by attempting to answer the question: *"Is the delay of the circuit $\geq \delta$?"*. It is based on the classic *D-calculus* notation [41] and amounts to determining a test for a multifault. Even though we will not go into a detailed explanation of the algorithm, it is instructive to consider some interesting points in its derivation. We should note that although it is simple to determine the test for the multifault from the leaf-DAG circuit, one would rather avoid having to create that representation of the circuit. The authors show that it is possible to work directly on the original circuit by developing a technique they call *timed test generation*. Basically, the fault propagation in the test procedure is restricted to paths in Π by considering timing information. That implies keeping some information for every connector in the circuit, namely the lengths of the longest paths from that edge to the primary outputs considering even and odd parity of inverters along that subpath. The D-calculus is then used to propagate faults and an extra variable containing timing data is kept along with the D values that one is trying to propagate. While evaluating a value at the output of a gate each fanout connector is considered separately and both the D-calculus and timing information are considered. This avoids gate duplication and allows one to work directly on the initial circuit.

In the final sections on the paper the authors present results for a set of circuits. They show that their algorithm is able to effectively compute the delay of some troublesome circuits in a very short period, sometimes with orders of magnitude improvement. As the previous two papers, this one is also important because it addresses an issue that had mostly been left out. The ability to work on sets of paths instead of having to consider one path at a time is invaluable and makes it possible to study circuits that have large numbers of long paths.

7 Conclusions

This paper introduced the problem of Critical Path determination in the context of Timing Verification. It is known that the speed of a circuit is related to the length of the longest path through which a signal can travel, also known as the longest true path or critical path of the circuit. Determination of the critical path of the circuit and its length is the aim of timing verification.

The paper focused on the floating mode of operation and addressed two main problems, namely the sensitization conditions for a path to be true and the methods that lead to efficient computation of the true delay of a circuit. Three relevant papers on the subject which cover the last few years of research in the field were covered and their findings compared. Various sensitization conditions were described and a true floating mode sensitization criteria was motivated and presented. A method that efficiently computes the circuit delay using this sensitization condition was also discussed.

Acknowledgments

I would like to thank Prof. Srinivas Devadas for having brought this subject to my attention and for having patiently helped me to understand it. Many, if not all of the figures in this text were also gently provided by him. I would also like to acknowledge the assistance of my friend and colleague João Paulo Silva, at the University of Michigan, Ann Harbor for having pointed me out several important references, explained several details and made some papers available to me.

References

[1] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Tech. Rep. ERL M520, Electronics Research Laboratory Report, University of California, Berkeley, Berkeley, California, May 1975.

[2] B. R. Chawla, H. K. Gummel, and P. Kozah, "MOTIS - An MOS timing simulator," *IEEE Transactions on Circuits and Systems*, vol. CAS-22, pp. 901–909, December 1975.

[3] R. A. Saleh, J. E. Kleckner, and A. R. Newton, "Iterated timing analysis and SPLICE1," in *International Conference on Computer Aided-Design*, (Santa Clara, California), September 1983.

[4] C. F. Chen and P. Subramanyam, "The second generation MOTIS timing simulator - An efficient and accurate approach for general MOS circuit," in *International Symposium on Circuits and Systems*, (Montreal, Canada), May 1984.

[5] Y. H. Kim, J. E. Kleckner, R. A. Saleh, and A. R. Newton, "Electrical-logic simulation," in *International Conference on Computer Aided-Design*, 1984.

[6] P. Odryna and S. Nassif, "The ADEPT timing simulation algorithm," *VLSI Systems Design*, March 1986.

[7] L. Vidigal, S. Nassif, and S. Director, "CINNAMON: Coupled INtegration and Nodal Analysis of MOs Networks," in 23^{rd} *ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), June 1986.

[8] R. Bauer, A. Ng, A. Raghunathan, and C. T. M Saake, "Simulating MOS VLSI circuits using super-Crystal," in *VLSI Conference*, North-Holland, 1987.

[9] R. Bauer, J. Fang, A. Ng, and R. Brayton, "XPSim: A MOS VLSI simulator," in *International Conference on Computer Aided-Design*, (Santa Clara), November 1988.

[10] Y. H. Kim, S. H. Hwang, and A. R. Newton, "Electrical-logic simulation and its applications," *IEEE Transaction on CAD*, vol. 8, pp. 8–22, January 1988.

[11] R. B. Hitchcock, G. L. Smith, and D. D. Cheng, "Timing analysis of computer hardware," *IBM Journal of Research and Development*, vol. 26, pp. 100–105, January 1982.

[12] R. B. Hitchcock, "Timing verification and the timing analysis program," in 19^{th} *ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 594–604, June 1982.

[13] J. K. Ousterhout, "Crystal: A timing analyzer for nMOS VLSI circuits," in *Third Caltech Conference on VLSI* (R. Bryant, ed.), (Rockville MD), pp. 57–69, Computer Science Press, 1983.

[14] J. K. Ousterhout, "A switch-Level timing verifier for MOS VLSI," *IEEE Transaction on CAD*, vol. 4, pp. 336–348, June 1985.

[15] N. P. Jouppi, "TV: An nMOS timing analyzer," in *Third Caltech Conference on VLSI* (R. Bryant, ed.), (Rockville MD), pp. 71–85, Computer Science Press, 1983.

[16] J. Benkoski, E. V. Meersch, L. Claesen, and H. De Man, "Efficient algorithms for solving the false path problem in timing verification," in *International Conference on Computer Aided-Design*, pp. 44–47, November 1987.

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

[18] V. Hrapcenko, "Depth and delay in a network," *Soviet Math. Dokl.*, vol. 19, no. 4, 1978.

[19] D. Brand and V. S. Iyengar, "Timing analysis using functional relationships," in *International Conference on Computer Aided-Design*, pp. 126–129, November 1986.

[20] P. C. McGeer and R. K. Brayton, "Efficient algorithms for computing the longest viable path in a combinational network," in 26^{th} *ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 561–567, June 1989.

[21] D. H. C. Du, S. H. C. Yen, and S. Ghanta, "On the general false path problem in timing analysis," in 26^{th} *ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 555–560, June 1989.

[22] S. Devadas, K. Keutzer, and S. Malik, "Delay computation in combinational logic circuits: Theory and algorithms," in *International Conference on Computer Aided-Design*, pp. 176–179, November 1991.

[23] S.-T. Huang, T.-M. Parng, and J.-M. Shyu, "A new approach to solving false path problems in timing analysis," in *International Conference on Computer Aided-Design*, pp. 216–219, November 1991.

[24] S. Devadas, K. Keutzer, S. Malik, and A. Wang, "Certified timing verification and the transition delay of a logic circuit," in 29^{th} *ACM/IEEE Design Automation Conference*, pp. 549–555, June 1992.

[25] H.-C. Chen and D. H. C. Chu, "Path sensitization in critical path problems," in *Proceedings ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Tau 1990*, August 1990.

[26] S. Devadas, K. Keutzer, S. Malik, and A. Wang, "Event suppression: Improving the efficiency of timing simulation for synchronous digital circuits," in *Proceedings of the Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, pp. 195–209, March 1992.

[27] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis*. New York, NY: McGraw-Hill, in preparation.

[28] P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal domains in Logic Design*. Norwell, Massachusetts: Kluwer Academic Publishers, 1991.

[29] J. Benkoski, E. V. Meersch, L. J. M. Claesen, and H. De Man, "Timing verification using statically sensitizable paths," *IEEE Transaction on CAD*, vol. 9, pp. 1073–1084, October 1990.

[30] K. Keutzer, S. Malik, and A. Saldanha, "Is redundancy necessary to reduce delay?," *IEEE Transaction on CAD*, vol. 10, pp. 427–435, April 1991.

[31] J. M. Silva and K. A. Sakallah, "A comparison of path sensitization criteria for timing analysis," in *accepted for publication in the ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Tau 1993*, September 1993.

[32] H.-C. Chen and D. H. C. Du, "Path sensitization in critical path problem," in *International Conference on Computer Aided-Design*, pp. 208–211, November 1991.

[33] S. Devadas, K. Keutzer, and S.Malik, "Path sensitization conditions and delay computation in combinational logic circuits," in *MCNC International Workshop in Logic Synthesis*, May 1991.

[34] S. Devadas, K. Keutzer, S. Malik, and A. Wang, "Computation of floating mode delay in combinational logic circuits: Practice and implementation," in *International Symposium on Logic Synthesis and Microprocessor Architecture*, pp. 68–75, July 1992.

[35] S. Devadas, K. Keutzer, S. Malik, and A. Wang, "Computation of floating mode delay in combinational logic circuits: Practice and implementation," in *ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Tau 1992*, 1992.

[36] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Transactions on Computers*, vol. C-30, pp. 215–222, March 1981.

[37] S. Devadas, K. Keutzer, S. Malik, and A. Wang, "Certified timing verification and the transition delay of a logic circuit," in *Proceedings ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Tau 1992*, 1992.

[38] H.-C. Chen and D. H. C. Chu, "Path sensitization in critical path problems," *IEEE Transaction on CAD*, vol. 12, pp. 196–207, February 1993.

[39] S. Perremans, L. Claesen, and H. De Man, "Static timing analysis of dynamically sensitizable paths," in 26^{th} *ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 568–573, June 1989.

[40] D. B. Armstrong, "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets," *IEEE Transactions on Computers*, vol. EC-15, pp. 66–73, February 1966.

[41] J. P. Roth, "Diagnosis of Automata Failures: a calculus and a method," *IBM journal of Research and Development*, vol. 10, pp. 278–291, July 1966.