Certified Timing Verification and the Transition Delay of a Logic Circuit

Srinivas Devadas, *Member, IEEE*, Kurt Keutzer, *Senior Member, IEEE*
Sharad Malik, *Member, IEEE*, and Albert Wang, *Member, IEEE*

***Abstract*— Most research in timing verification has implicitly assumed a single vector *floating* mode computation of delay which is an approximation of the multivector transition delay. In this paper we examine the transition delay of a circuit and demonstrate that the transition delay of a circuit can differ from the floating delay of a circuit. We then provide a procedure for directly calculating the transition delay of a circuit. The most practical benefit of this procedure is the fact that it not only results in a delay calculation but outputs a vector sequence that may be timing simulated to *certify* static timing verification.**

I. Introduction

THE LONGEST-PATH DELAY of a circuit is simply the sum of the cumulative delays of a circuit along the longest graphical path. This measure of delay is still used in most static timing verifiers but has the deficiency that it does not take into account *false paths.* To remedy this deficiency the *floating delay* of a circuit may be analyzed. 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 assumptions about the state of the circuit before the single vector is applied. A number of techniques have been proposed for computing the true floating delay of circuit, but a significant step was taken in [13] where a technique that provided correctness in the light of *monotone speed-ups* was demonstrated. Further improvements were made in [5], [7] where techniques that more precisely identified the critical path were presented. The transition delay of a circuit is the delay under a multivector dynamic-analysis condition that makes no assumptions about the state of the circuit before the vector sequence is applied. A circuit is presented in this paper whose true floating delay is greater than its transition delay; thus, the floating analysis condition itself has some deficiencies.

Meeting delay requirements is the most important constraint imposed on a circuit. For this reason verifying the timing of a circuit before manufacture is one of the most important tasks of a computer-aided design system. Unfortunately to date there has been no fully satisfactory approach to this problem. One solution to this problem is to simulate the behavior of the circuit using an accurate timing simulator. Simulators such as `SPICE` [1] are able to very accurately model the temporal behavior of a circuit. Accurate simulation has two significant problems: It is computationally expensive and its utility is limited by the vector set that is applied. The first problem can be addressed by using less accurate but more computationally efficient algorithms such as `CRYSTAL` [14]. Unfortunately, in any simulation-based approach the final result is only as good as the vector set that is applied. Simulation of all possible input stimuli is never an option, and if there is one unsimulated input stimulus that could cause the circuit to go slower, then the simulation results may lead to the manufacture of a circuit that will not run at the required speed.

An approach that avoids the problem of vector dependency is to use static timing verifiers [10], [14]. In this approach the delay of a circuit is determined to be the longest path in the circuit. This approach also has two significant problems: First, the timing models used in static timing verification are typically not as accurate as those in timing simulators such as `SPICE`. Secondly, there may not be any input stimulus that activates the longest path in the circuit as determined by the static timing verifier. Such paths are called false paths [2]. Thus static timing verifiers may be too pessimistic as regards the delay of the circuit. A potential solution to this problem is to eliminate from consideration those paths that are not statically sensitizable; however, it has been shown [3], [13] that paths which are not statically sensitizable may still contribute to the delay of the circuit. Thus simply eliminating these paths from consideration may result in too optimistic a notion of the delay of the circuit, and ultimately in a circuit that is slower than was required.

The obvious course of action is then to augment static timing analysis with techniques to eliminate from consideration only the false paths. This natural step also introduces three problems: The first problem is to accurately determine the false paths. While the work of [3], [13] and others made significant strides in this direction even the recent work of [13] cannot be said to correctly identify the paths that were responsible for the delay of the circuit.[1] It was not until the work of [5], [7] that the floating delay of a circuit could be accurately identified. The second problem is eliminating the

Manuscript received July 6, 1993; revised February 2, 1994 and March 16, 1994. This work was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-91-J-1968 and in part by a NSF Young Investigator Award with matching funds from Mitsubishi and IBM Corporation.

S. Devadas is with the Department of EECS, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.
K. Keutzer and A. Wang are with Synopsys, Inc., Mountain View, CA USA.
S. Malik is with the Department of Electronic Engineering, Princeton University, Princeton, NJ 08540 USA.

IEEE Log Number 9403168.

[1] Although it could accurately identify the delay of the circuit in the floating delay model.

false paths in a computationally efficient manner. The work of [7], [9] makes this procedure feasible on a wide variety of circuits, by considering sets of paths rather than individual paths. The third problem is that even with the most complete information, such as post-layout wire capacitances, there are still many potential inaccuracies in the modeling of the timing verifier and it is desirable to do a final timing simulation with the most accurate timing models.

Thus, despite the advances in the area of timing verification, any designer who relies heavily on the final performance of a circuit is reduced to the time consuming and error-prone task of identifying long paths and handwriting simulation vectors that will stimulate the paths that are determined to be critical by the timing verifier. In this paper we hope to provide a comprehensive solution to the problem of timing verification through *certified timing verification* which incorporates *transition delay* computation. In this approach the vector pair resulting from transition delay computation can be used in a subsequent timing simulation using a timing simulator with more accurate timing models, that take into account layout-level parasitic resistances and capacitances.

Our system, called TrueD, takes as input a combinational logic circuit and outputs an accurate floating delay of the circuit, as well an accurate transition delay with a corresponding set of simulation vectors that will allow certification of the results of static timing verification.

In Section II, we give the basic definitions and terminology used. We motivate the consideration of two-vector transition delay in Section III. In Section IV, we give a spectrum of transition delay models, that are applicable in a variety of design scenarios. In Section IV-B we point out the differences between the floating delay and transition delay of a circuit. We describe a technique for computing the transition delay in Section V, and results using these techniques are presented in Section VI. We describe the methodology of certified timing verification in detail in Section VII.

II. Definitions and Notation

In this section we introduce terminology that will allow us to discuss timing issues as well for temporal behavior.

A **path** in a combinational circuit is an alternating sequence of vertices and edges, $\{g_0, e_0, \ldots, g_n, e_n, g_{n+1}\}$, where edge e_i, $0 \leq i \leq n$, connects the output of vertex g_i to an input of vertex g_{i+1}. For $1 \leq i \leq n$, g_i is a gate; g_0 is a primary input and g_{n+1} is a primary output. Each e_i is a net. With each vertex g (edge e) we associate a delay $d(g)(d(e))$.

The **length** of a path $P = \{e_0, g_0, e_1, \ldots, e_n, g_n, e_{n+1}\}$ is defined as $D(P) = \sum_{i=0}^{n} d(g_i) + \sum_{i=0}^{n+1} d(e_i)$. Ignoring sensitization conditions, the delay of a circuit as given strictly by the length of the longest path is called the **topological** or **graphical** delay.

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

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

Let $\pi = \{g_0, e_0, \ldots, g_n, e_n, g_{n+1}\}$ be a path. The inputs of g_i other than e_{i-1} are referred to as the **side-inputs** to π. If there exists an input vector v such that all the side-inputs along π settle to noncontrolling values on v then π is **statically sensitizable.**

The **critical path** is the longest sensitizable path in the circuit under the stated delay model. If a path is not sensitizable under the stated delay model then it is a **false path.** The precise definition of sensitization can vary depending on the mode of operation assumed. For our purposes here, we can assume that the sensitization of a path implies that an event propagates along the path from a primary input to a primary output of the circuit.

III. Circuit History: How Much is Enough?

We are interested in determining the delay of a circuit for a given delay model, but the real motivation is to determine the frequency at which a circuit can be clocked. There are a number of possible definitions of transition delay and each definition has implications on the issue of clocking frequency. A full consideration of these problems is beyond the scope of this paper, but in this section we introduce our notion of transition delay and show under what conditions it results in a valid clock clocking frequency.

Consider the operation of a synchronous digital circuit being clocked at period τ. At every clock period, the outputs are latched and a new set of inputs presented to the circuit. Let us examine the operation of a circuit over the period of application of a sequence of input vectors. Let v_0 be the vector applied at the present clock cycle, v_{-1} be the vector applied at the previous clock cycle and so on. In the **floating** mode of operation, the nodes are not assumed to be ideal capacitors and hence their state is unknown till it is set by the current vector. Thus, the timing behavior for v_0 is independent of all previous vectors. In the **transition** mode of operation, the circuit nodes are assumed to be ideal capacitors and retain their value set by the previous vectors till the current vector forces the voltage to change. Thus the timing response for v_0 is also a function of v_{-1} and possibly other previously applied vectors. In analyzing the timing response of the circuit, we would like to deal with as little history as possible while making no compromises on the accuracy. In this direction we first propose the following model of measuring the delay in the transition mode and subsequently justify it.

Let us assume that when vectors v_0 is applied, all circuit nodes have stabilized to their values under v_{-1}. In this case the effect is the same as if v_{-1} is given an arbitrary amount of time to settle. This mode of operation will be referred to as the *single stepping* transition mode and for the remainder of this paper and whenever we refer to transition delay it will be relative to this mode of operation. The input transition from v_{-1} to v_0 will result in some transitions at the circuit nodes.

Let δ be the time taken for the last transition at any of the output nodes for all possible vector changes v_{-1} to v_0. Thus, for all $\tau > \delta$ no transition will ever be observed at any of the outputs in the single stepping mode.

Let us now use this value of τ to clock the synchronous circuit. At time 0, when v_0 is applied it is possible that the circuit nodes may not have stabilized to their values under v_{-1}. (Note that the fact that the outputs have stabilized does not imply that all the circuit nodes have stabilized.) A simple sufficiency condition for a transition delay τ to be a valid clock period is for the state of the circuit to be the same whether 1) v_0 is applied only an interval of τ after v_{-1} or 2) v_0 is applied at an arbitrary interval (which is longer than ω) after v_{-1}. This is expressed in the following theorem which was originally stated in [8]. We provide a detailed proof here.

Theorem 3.1: Let C be a combinational subcircuit of a synchronous digital circuit. Let τ be the transition delay of the circuit derived using the single stepping mode of operation. Let ω be the length of the longest graphical path in C. If $\tau > \omega/2$ then τ is a valid clocking period for C.

Proof: Let v_{-1} be the vector applied at time $-\tau$ and let v_0 be the vector applied at time 0. Let e_{-1} refer to any event in the circuit that occurs after time 0 and is caused by v_{-1}, i.e., any event that is caused by v_{-1} but is still propagating at time 0 after v_0 is applied. Each event e_{-1} has traversed at least a distance corresponding to delay τ from the primary inputs and has at most a distance corresponding to $\omega - \tau$ to traverse to reach the primary outputs.

Now let us assume that the application of v_0 results in an event at the circuit outputs after time τ (making τ an erroneous clock period) and let e_0 be the last such event. If this is true then e_0 must propagate along a path, π, of length at least τ in the circuit. For each gate along π, the side inputs do not witness any event e_{-1} after e_0 has propagated along π. To see why this must be true, note that after time 0, e_0 propagates for at least time τ and e_{-1} can propagate for no more than $\omega - \tau$, which is strictly less than τ when $\tau > \omega/2$. Thus, as far as e_0 is concerned, its propagation along π looks exactly as the case for the single stepping mode. But, we know that for the single stepping mode e_0 cannot occur after time τ at the outputs. Thus, e_0 cannot occur at the circuit outputs after time τ even when the circuit is no longer operating in single stepping mode, but is being clocked with period τ. □

When this condition is true, then for events propagating along all paths of length at least τ, each gate will have settled to its value under v_{-1} by the time the event gets to that gate, which is the same as it would be in the single stepping mode. The restriction $\tau > \omega/2$ is not very stringent. Based on practical experience this property holds for most circuits. This is significant since it enables us to consider only the two vectors involved in the change at the inputs in the analysis of the timing response of this transition.

IV. Component Delay Models

All simulation techniques as well as timing analyzers make some assumptions about the possible variations in the delay of various circuit components. To avoid further confusion with other uses of the term "delay model" we call these component delay models. We now examine some of the models used and then specify the domain of this paper in terms of these.

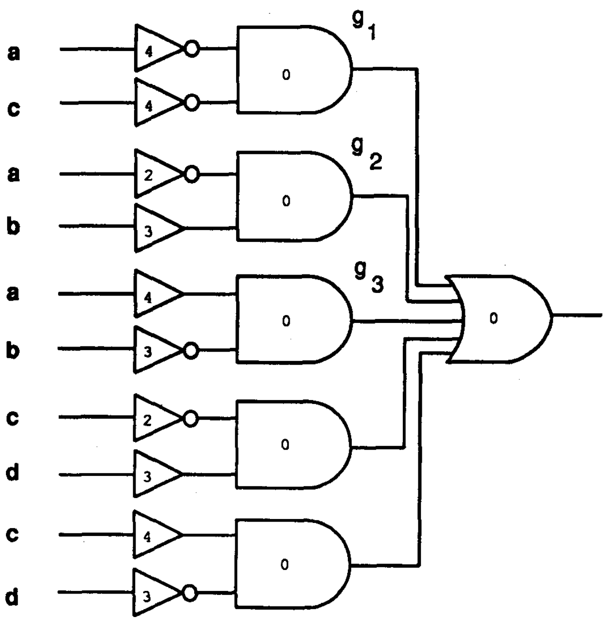


Fig. 1. Floating and transition delays differ.

The most common component delay model for a circuit component is one in which the delay is assumed to be a fixed number d. This is referred to as the **fixed delay** model. In this model a delay of 2 units on a gate indicates that the gate switches instantaneously to a logical 0 or 1 value but that the communication of this event to the output of the gate is delayed by 2 units. In reality this number is typically an upper bound on the expected delay, so in fact the actual delay may be any number bounded above by d. This potential speedup is incorporated in the **monotone speedup** model [13], which assumes that the delay for each component lies in the range [0, d].

The **bounded delay** model is more realistic about how much each gate can in fact be sped up. It specifies the delay as a range, $[d^l, d^u]$, given by the lower and upper bounds on the actual delays. There is some ambiguity in the literature as to what the bounded delay model means. It has been interpreted as either the switching delay or as the propagation delay. With the former interpretation, a **gate** delay of [2, 4] would imply that the gate would take somewhere between 2 and 4 units to make the transition between the two logical values. Ternary algebras [15] have been used to accommodate the unknown value of the gate output (which is neither a 0 or a 1) in the interval of uncertainty. If the bounded delay relates to the propagation delay then this implies that the gate switches instantly, but it is uncertain as to the time in the [2, 4] interval at which it will switch. Note that unlike the previous case, the gate output is always a 0 or a 1 and ternary algebras are not needed. In this paper, we use the propagation delay interpretation since we feel it is more realistic with respect to current technologies.

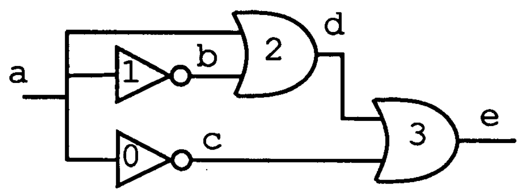


Fig. 2. Floating and transition delays differ—even with monotone speedup.

To simplify the presentation of this paper we consider only the fixed delay model and we restrict the delays in circuits to gates. This is still general enough to accommodate other delay quantities such as wire delays and pin-to-pin delays by introducing buffers with appropriate delays in the circuit. While more sophisticated delay parameters such as slope delays and separate rise and fall delays are not directly accommodated into the "delay lumped at a gate" paradigm, it can be shown with little effort that the results in this paper hold for even those models. Bounded delays are treated extensively in [11].

A. Instantaneous Glitches

Suppose that in a circuit simulation a rising event and a falling event arrive at precisely the same time at an AND gate with delay "2". In static timing verification, which aims at a robust though possibly pessimistic model of delay, it is commonly assumed that an "instantaneous glitch" event immediately occurs which is then communicated at the output of the AND gate after a delay of 2. In this paper our general assumption will be that such instantaneous glitches will not be able to overcome the inertial delay of a gate. This is more realistic given existing technologies.

B. Differences in the Floating Delay and the Transition Delay

It is easy to show that the floating and transition delay modes can give different results. In Fig. 1 we show a two-level circuit, resulting from a prime and irredundant cover, for which the transition delay (under fixed gate delays) is different from the floating delay. The number inside a gate corresponds to the delay of the gate.

Fig. 1 shows one of the ways that the transition delay of a circuit, assuming no monotonic speed-ups and no instantaneous glitches, can differ from the floating delay of a circuit. The floating delay of this circuit for a rising transition at the output is 4. In this example, on each transition on g_1 for $0 \rightarrow 1$ at least one of the other gates makes a $0 \rightarrow 1$ transition sooner. For example, let us look at the vector pair $\langle 1100, 0000\rangle$. The gate g_2 glitches to a 1 during the time interval [2, 3] then the gate g_3 glitches to a 1 during the time interval [3, 4] and as a result by the time the gate g_1 makes a $0 \rightarrow 1$ transition (at time 4), the output OR gate is already a 1.

However, since the work of [13] it has been considered to be necessary that speed-ups in the circuit should not result in increasing the delay of the circuit. In Fig. 1, if the input buffers/inverters to gates g_2 and g_3 speed up then the $0 \rightarrow 1 \rightarrow 0$ glitches of g_2 and g_3 on $\langle 1100, 0000\rangle$ settle to 0 before g_1 makes its $0 \rightarrow 1$ transition. As a result the transition delay of the circuit becomes equal to the floating delay.

Because of examples like this it has been conjectured that for any combinational circuit there exists a monotone speedup such that the transition delay equals the floating delay; however, in Fig. 2 we give a circuit in which the transition delay, with or without monotone speedup, is less than the floating delay.

The floating delay of the circuit in Fig. 2 is 5, and the associated floating delay vector is $\langle a = 1\rangle$. The transition delay of the circuit under the single stepping mode of operation is 3. We could give a full analysis of the transition delay for this circuit, but for our purpose here it suffices to give an intuitive argument why the transition delay is strictly less than the floating delay even with monotone speedup.

In single stepping mode, we apply an input vector when all circuit nodes are stabilized to their final values, in this case signal d and e to logic 1. Consider the case where the next input vector generates a rising transition at the input: It immediately forces signal d, which in turn forces e, to remain at logic 1. If the next input vector generates a falling transition, it immediately causes a rising transition at signal c which forces e to remain at 1 also. Therefore, no transition can be observed at the output in the single stepping mode of operation. The above argument remains valid under arbitrary monotone speedup of the circuit as will be demonstrated in the discussion in Section IV-C.

Thus, transition delay analysis using the single stepping mode returns a delay of 0. However, Theorem 3.1 only guarantees the validity of a clock period greater than $\omega/2 = 3$. For example with a clock period of 4, less than the floating delay of 5, the output of the circuit stays a stable 1.

It is interesting to note that the path $\{a, d, e\}$ of length 5 in Fig. 2 is statically sensitizable. While it has been known for some time that static sensitization can be too optimistic a condition to determine the delay of a circuit this example demonstrates that it can also be too pessimistic.

Circuits such as those in Figs. 1 and 2 motivate a further enquiry into the relationship between transition delay and floating delay.

C. Sources of Difference Between Floating and Transition Delay

In the floating delay mode the delay of a path π is computed on a single vector v_0. At a gate g, along π, that is controlled by v_0 it is implicitly assumed that if v_0 results in a static noncontrolling value at a side input e along π, then there always exists a vector sequence ending in v_0 that will result in that noncontrolling value on e when an event propagates along π due to v_0. In other words the vector sequence causes the noncontrolling value to be available when an event occurs along π, even if a speed-up is required to deliver that noncontrolling value. It will be useful to understand precisely why this assumption is not always valid, and why a violation of this assumption can result in the transition delay of a circuit being less than the floating delay.

It will be interesting to understand precisely why the transition delay of a circuit can differ from the floating delay. For

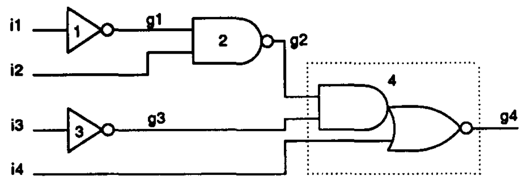


Fig. 3. A multilevel combinational circuit.

an event travelling down a path to be affected by the delay of a gate the event must propagate through the gate. The floating delay mode makes conservative assumptions about the initial state of a circuit when a vector is applied. Let us assume that a path π is determined to be responsible for the delay of the circuit in the floating delay mode. Let v_0 be the vector that sensitizes π to result in this delay in the floating delay mode. Let g be a gate along π whose output is a controlled value on v_0 and let e be a side-input of π at g that has a statically noncontrolling value on v_0. In floating delay analysis it is implicitly assumed that there exists a vector v_{-1} that will result in a noncontrolling value on e when an event propagates along π due to v_0; however, it may be the case that for any vector v_{-1}, e may be slow to transition to a noncontrolling value and thus the event along π is blocked at g.

Fig. 2 demonstrates this point. The vector $v_0 = \langle a = 1\rangle$ statically sensitizes the path $\{a, d, e\}$ and thus results in a delay of 5 in the floating delay mode. Consider the activity at gate d on the input pair $v_{-1} = \langle a = 0\rangle$ and $v_0 = \langle a = 1\rangle$. Gate b settles to a 0 on v_0 but only after the rising event associated with input a has reached the input of gate d. Thus the event on the path $\{a, d, e\}$ gets stopped at d. If the delay of gate b is reduced to 0, through a monotone speed-up, then an *instantaneous glitch* (see Section IV-A) occurs at the inputs to d, but such instantaneous glitches are assumed not to change the output of a gate. (We have a more complex example for which it is not possible to transmit even instantaneous glitches.) Thus, the floating delay assumptions are too conservative for this circuit.

V. Symbolic Simulation Under Fixed Gate Delays

A. Introduction

Having motivated transition delay let us now consider its computation. In order to compute the delay of a circuit under the transition mode, a strategy of symbolically simulating all possible input vector *pairs*, that can be applied to the circuit inputs, can be adopted. The possible resulting waveforms at each gate are encoded by a set of Boolean functions, one for each discrete time point. Each Boolean function is defined over the Boolean variables corresponding to the circuit inputs for the first and second vectors. We will consider the details of symbolic simulation for fixed gate delays in the following sections. For a symbolic simulation method for bounded gate delays, the reader is referred to [11].

In the fixed delay case, a single set of Boolean functions at each gate in the circuit suffices to capture all the information regarding the transitions occurring at the gate. We will begin with an illustrative example.

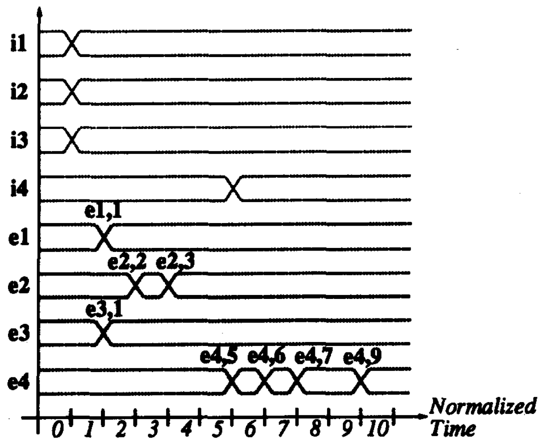


Fig. 4. Signal waveforms for the primary inputs and the outputsof the logic gates.

B. An Example

Consider the multilevel combinational circuit shown in Fig. 3.

It has four primary inputs and one output and consists of four CMOS gates. (The gate surrounded by the dotted line is a complex gate having series-parallel connections only.) The circuit has a total of four gates: g_1, g_2, g_3, and g_4, in addition to the four primary input nodes. Fig. 4 shows the signal transitions at the primary input nodes as well as the possible transitions at the internal nodes of the network. The time points are in normalized units. The first three inputs, $i1, i2$, and $i3$, switch simultaneously between time periods 0 and 1. The fourth input, $i4$, is a late arriving signal that switches between the time points 5 and 6. In this example, the delays of both gate g_1 and gate g_3 are one time unit. Gate g_2 has a delay of two units while Gate g_4 has a delay of four units. $i4$ arrives five time units after the other three inputs.

Also shown in Fig. 4 are the waveforms, e_i's, representing the signals at the outputs of i^{th} logic gates. Each of the possible transitions $e_{i,j}$ represents either a low-to-high or high-to-low signal transition between $[j]^{\text{th}}$ and $[j+1]^{\text{th}}$ time points. The number of all possible transitions at a gate output is bounded by the sum of all possible transitions at the gate inputs. These transitions are delayed by the gate's propagation delay. If a gate is driven by the primary input signals, then the transitions at the gate output will be determined by the transitions of the primary input signals. Referring to the example, gate g_1 has waveform e_1 which contains only one possible transition, $e_{1,1}$, between the time points 1 and 2 because the total number of transitions at the input of gate g_1 at different time points is one. The earliest signal event will arrive at the gate output one time unit after $i1$ switches because the delay of gate g_1 is one time unit. Similarly, gate g_4 has a total of four possible transitions between the time points 5 and 10 because the number of transitions at the inputs of gate g_4 at distinct time points is

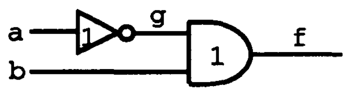


Fig. 5. An inverter-and circuit.

four. The earliest transition, originated from gate g_3, arrives at the gate output between the time points 5 and 6, since gate g_4 has a delay of four time units.

C. Unit-Delay Model

In this section, we describe how symbolic simulation can be used to compute signal transitions in a circuit under the unit-delay model. The unit-delay restriction will be relaxed in Section V-E.

Under the unit-delay model, the circuit activity caused by applying an input vector pair can be seen as occurring at discrete points in time implied by the unit-delay model, and all signals are stable between the time points. The central idea in our formulation is to describe the stable values of a signal in a particular time interval symbolically as a Boolean function of input variables over two input vectors (i.e., each implicant of f_i corresponds to an input vector pair which generates a 1 at signal f in time interval i). Determining whether a particular signal changes from interval i to $i+1$ now reduces to checking for logical equivalence of the functions in the two intervals. As a by-product, the input assignment that causes the two functions to differ gives us the input vector pair that generates a transition at time point between intervals i and $i+1$.

As a notational convention, we use subscripts to denote the time interval a variable is associated with. A special subscript "—" is used to denote input variables associated with the first vector.[2] The time at which the second vector is applied is a common reference point 0. So, for an input variable a, all a_i with $i < 0$ map to a_- and all a_i with $i >= 0$ map to a_0. Furthermore, for every function $f(g, h, \ldots)$ we construct a new function $f_i(g_{i-1}, h_{i-1}, \ldots)$ for a chosen i.

For example, consider the network in Fig. 5. For this network in the first time interval,

$$g_0 = \overline{a_-}$$
$$f_0 = g_{-1}b_{-1} = \overline{a_{-2}}b_{-1} = \overline{a_-}b_-$$

In the next time interval,

$$g_1 = \overline{a_0}$$
$$f_1 = g_0 b_0 = \overline{a_-}b_0 = \overline{a_-}b_0$$

Finally for the last interval,

$$f_2 = g_1 b_1 = \overline{a_0}b_0$$

For this example there are three possible transitions:

- g changes state from time interval 0 to 1,
- f changes state from time interval 0 to 1,
- f changes state from time interval 1 to 2.

[2]Note that we assume the single stepping mode of operation.

The symbolic formulas for these transitions are, respectively:

$$e_{g,1} = g_0 \oplus g_1 = \overline{a_-}a_0 + a_-\overline{a_0}$$
$$e_{f,1} = f_0 \oplus f_1 = \overline{a_-}b_-\overline{b_0} + \overline{a_-}\overline{b_-}b_0$$
$$e_{f,2} = f_1 \oplus f_2 = \overline{a_-}a_0b_0 + a_-\overline{a_0}b_0$$

Each implicant in $e_{f,2}$ gives an input vector pair which generates a transition of signal f at time 2 (between time interval 1 and 2). For example, implicant $\overline{a_-}a_0b_0$ corresponds to vector pair $v_1(a,b) = (0, X)$ and $v_2(a,b) = (1,1)$.

It should be pointed out that our formulation is more powerful than a procedure that just determines transitions of a signal at a particular time point. For example, finding an input vector pair that generates transition of f at both time 1 and 2 amounts to finding an implicant of $e_{f,1}e_{f,2}$ (e.g., $\overline{a_-}a_0\overline{b_-}b_0$). Also, the inputs need not be clocked at the same time. For example, if the second value of a is clocked at time t, all a_i with $i < t$ map to a_- and all a_i with $i \geq t$ map to a_t. This can be done on a per input basis.

As an efficiency concern, it is not necessary to generate function f_i for all time points. The following lemma gives a simple bound on the time points needed for a signal in the circuit.

Lemma 5.1 Let Δ and δ be the longest and shortest graphical delay to a signal f. The set $f_0, f_\delta, f_{\delta+1}, \ldots, f_{\Delta-1}, f_\Delta$, is sufficient to determine all possible transitions in the circuit.

The above lemma follows directly from the unit delay model. Signal f cannot change until the change in the nearest input propagates through, and will stop changing when the input furthest away finally arrives. All references to f_i map to f_Δ if $i > \Delta$ and to f_0 if $i < \delta$.

D. Symbolic Event Suppression

To compute the transition delay of a circuit, it may not be necessary during symbolic simulation to store, or even generate, all the Boolean functions corresponding to each gate and each time point. Typically, one is interested in answering the question: "Is the delay of the circuit $\geq \delta$?" In this case, we only require the f_t's where $t \geq \delta - 1$ and f is the circuit output. We XOR $f_{\delta-1}$ with each such f_t and check the XOR'ed function for satisfiability to see if there is indeed a transition at time δ.

Techniques similar to the event suppression techniques described in [8] for the efficient simulation of a vector pair on a circuit can be used in the symbolic simulation procedure as well. For example, given a gate g in the circuit, let w_g be the length of the longest path from the gate output to the circuit output. We only need to compute the g_t's such that $t + w_g \geq \delta - 1$. The reason for this is simple. If the gate g makes a transition at time t_0, then this transition can appear at the circuit output no later than $t_0 + w_g$. If $t + w_g < \delta - 1$ then the transitions corresponding to the function g_t fall before the interval of interest at the circuit output and need not be computed at gate g.

E. General Delay Model

A gate with a large fanin may have several times the delay of an inverter. If one uses normalized time units, one can

TABLE I
STATISTICS OF BENCHMARK EXAMPLES

EX	inputs	outputs	literals	longest
C17	5	2	19	5
C432	36	7	405	19
C499	41	32	977	25
C880	60	26	718	20
C1355	41	32	1121	27
C1908	33	25	1225	34
C2670	233	140	1764	25
C3540	50	22	2332	41
C5315	178	123	3923	46
C6288	32	32	4752	123
C7552	207	108	5488	38
planet	13	25	894	11
sand	16	14	968	12
styr	14	15	1004	15
scf	33	63	1223	12

TABLE II
TRANSITION DELAY COMPUTATION ON BENCHMARK EXAMPLES FOR FIXED GATE DELAYS

EX	val	l. d.	f. d.	#check	CPU secs	t. d.
C17	1	5	5	1	0.03	5
C432	1	19	19	1	0.21	19
C499	1	25	25	1	0.79	25
C880	1	20	20	1	0.66	20
C1355	1	27	27	1	10	27
C1908	1	34	31	21	3675	31
C2670	0	25	24	2	232	24
C3540	0	41	39	10	182	39
C5315	1	46	45	9	11	45
C6288	1	123	122	2	812	122
C7552	1	38	37	9	16	37
planet	1	11	11	1	0.1	10
sand	1	12	12	1	0.2	11
styr	1	15	15	1	0.1	15
scf	1	12	12	1	0.2	11

always introduce unit-delay buffers at the output of gates in a circuit, which have a delay greater than unity, in order to model differing delays among logic gates.

F. Bounded Delay Model

The symbolic simulation algorithm can be extended to the case where the gate delays are variable and bounded within a given range. The reader is referred to Section IV of [11] for details regarding the extended method.

G. Checking Satisfiability

We maintain the various Boolean functions as multilevel logic networks during the course of symbolic simulation. The size of these networks is not much larger than the circuit itself. Alternatively, we could have used reduced, ordered Binary Decision Diagram (ROBDD) [4] representations for these functions, and propagated ROBDD's through the circuit.

Once we have the $f_{\delta-1} \oplus f_\delta$ function for the circuit output, we can determine if the function is satisfiable by constructing a reduced, ordered Binary Decision Diagram for the function, or using the satisfiability checking procedure of Larrabee [12]. In the case of circuits like multipliers, constructing ROBDD's for the Boolean functions is infeasible, but our method of maintaining the Boolean functions as multilevel networks, and the use of Larrabees' satisfiability checking algorithm succeeds in computing the transition delay of the circuit (*cf.* Section VI).

VI. EXPERIMENTAL RESULTS

In this section, we present preliminary experimental results in determining the transition delay of a circuit. Transition delay computation results in the procurement of vector pairs that each propagate a transition along a longest true path to the output. The set of vector pairs can be used to perform timing analysis under more sophisticated delay models, (e.g., using `SPICE`).

The techniques described in the previous sections have been implemented in the program `TrueD-T`. The statistics of the chosen benchmark circuits are shown in Table I. The first set of examples correspond to combinational circuits from the ISCAS combinational logic benchmark set. The second set correspond to state encoded, optimized and mapped finite state machine controllers from MCNC FSM benchmark set.

Results on applying the fixed delay simulation calculus to compute the transition delay of a circuit are given in Table II. We were able to exactly compute the transition delay under the fixed unit gate delay model for all the benchmark circuits, using the symbolic simulation algorithm described in Section V. In the table, val corresponds to the logical value the path was sensitized to, l.d. is the longest path delay, f.d. is the floating mode delay, and t.d. is the transition delay.

Note that for the finite state machine examples the set of input vectors in floating delay computation was restricted to $i@s$, with $s \in S$ where S is the set of reachable states. In transition delay computation, the set of input vector pairs $\langle i_1@s_1, i_2@s_2 \rangle$ were applied such that $s_1 \in S$ with s_2 being determined by the next state logic and $i_1@s_1$. The combinational logic benchmarks showed no difference between the floating delay and the transition delay. In virtually all of the combinational circuits, the longest floating mode sensitizable path is statically sensitizable, implying that the path is also sensitizable under fixed gate delays and the transition mode of operation. Differences between floating and transition delay in

TABLE III
TRANSITION DELAY COMPUTATION ON BENCHMARK EXAMPLES FOR BOUNDED GATE DELAYS

EX	val	l. d.	f. d.	#check	CPU secs	t. d.
C17	1	5	5	1	0.03	5
C432	1	19	19	1	0.21	19
C499	1	25	25	1	0.79	25
C880	1	20	20	1	0.66	20
C1355	1	27	27	1	15.76	27
C1908	1	34	31	21	3675	31
C2670	0	25	24	2	210	24
C3540	0	41	39	10	182	39
C5315	1	46	45	9	9	45
C6288	1	123	122	2	812	122
C7552	1	38	37	9	20	37
planet	1	11	11	1	0.1	10
sand	1	12	12	1	0.2	11
styr	1	15	15	1	0.1	15
scf	1	12	12	1	0.2	11

the finite state machine controller examples were observed due to the additional restriction on the applied input vector pairs.

The fixed delay model may not be realistic for use in certified timing verification since there will always be some statistical variation in the predicted gate delays. We next experimented with the bounded gate delay model, where all gates have upper bounds equal to unity, and lower bounds equaling zero.[3] We have been able to obtain vector pairs that validate the floating delay for all the ISCAS-85 benchmark circuits under the bounded gate delay model. The results are shown in Table III.

The CPU times in Tables II and III are on a SUN-4 workstation and correspond to the time required for floating delay computation using the method of [9] plus transition delay computation. The time required for transition delay calculation is typically a small fraction of the time required for floating delay computation.

We are currently experimenting with random-logic circuits to see if logic optimization affects the transition delay of a circuit.

VII. CERTIFIED TIME VERIFICATION

In Section IV-B we demonstrated that there can be a difference in the floating delay and transition delay of a circuit. In Section V we gave a procedure that actually computed the transition delay of a circuit assuming fixed delays and no monotonic speed-up. This procedure produces a vector pair that sensitizes the critical path in the transition delay mode. Finally, in Section VI we presented results on applying the transition delay computation procedure to a number of benchmark examples. As Table II indicates, on benchmark circuits the transition delay typically does not differ from the floating delay, but we claim that it is still useful to compute the transition delay because of the utility of the resulting vector set for certifying the delay of the circuit. In this section we briefly outline a procedure for certifying the results of static timing verification.

[3] The monotone speed-up condition.

The transition delay procedure works on the question "Is there a path with transition delay greater than or equal to δ?" The first step is to identify the upper bound on the delay of the circuit. As the transition delay of a circuit is bounded above by the true floating delay [5] the natural value for δ is the true floating delay. This can efficiently be computed using techniques described in [7], [9]. The derived value of δ is then passed to the symbolic simulation procedure described in Section V, or for computation using bounded delays the procedure described in [11] may be employed. The nominal use of these transition delay computation procedures is to return a single vector sequence which sensitizes an event along some path, perhaps only one, of length at least δ. Whichever approach is applied let us call the resulting vector sequence V.

The circuit model, perhaps with speed-ups required by the transition delay computation procedure, is then given to the timing simulator of choice. The vector sequence V is then applied. In general the results of the timing simulation should not give delay values that are worse than the results of the transition delay calculation. If this happens it means that the delays used in the transition delay calculation were not pessimistic enough; these should be modified and the delay calculation re-run.

The most complex case is when the timing simulation of V reports a delay γ that is less than δ. If there is sufficient confidence in the coverage of the vector set then an aggressive designer may opt to clock the circuit at γ. Another approach is to further investigate the range of possible clocking speeds using statistical methods [11]. The hope here is that the statistical analysis procedures will give a quantitative notion of what percentage of parts are likely to run at each speed in the range between γ and δ.

Using a combination of transition delay computation and timing simulation in this way gives a greater predictability to the post-manufacture delay of the circuit.

VIII. CONCLUSION

In this paper we demonstrated for the first time that the transition delay of a circuit *can differ* from the floating delay even in the presence of arbitrary monotonic speed-ups in the circuit. This result is used to motivate the derivation of a procedure which directly computes the transition delay of a circuit. The output of the transition-delay computation procedure is a vector sequence which excites an event along the longest sensitizable path of the circuit under consideration.

While this theoretical framework for the analysis of transition delay is in itself useful for understanding the relationship between static and dynamic delay analysis, we envision the most practical application of these results in *certified timing verification*. In such a scenario the upper bound on circuit delay is first derived by means of a floating delay calculation. The transition delay of the circuit is then derived using the transition delay calculator, and a vector sequence for sensitizing the critical paths of the circuit is produced as a by-product of this delay calculation. This vector sequence can then be applied using a timing simulator equipped with more accurate timing models. Such a procedure promises to give the high accuracy of timing simulation with the computational efficiency and the comprehensive path coverage of static timing verification.

A. Work in Progress

A great deal of work remains to be done in order to understand completely the relationship between transition and floating delay. Even the definition of transition delay requires further examination. The common *single-stepping* definition of transition delay together with a simple sufficiency condition for a valid clock frequency was presented in Section III, but a full consideration of the relationship between transition delay and clocking frequency remains to be done. Encouraging progress toward resolving this question with regard to floating delay was presented in [6]. Correct computation of transition delay seems to become enmeshed in many technology specific issues, such as the instantaneous glitches discussed in Section IV-A, and these issues also require further resolution. Furthermore, while a distinction between floating delay and transition delay has been drawn in this paper we presently have no clear idea of how fundamental this difference is. We have presented circuits in which a difference occurs and we have derived a number of circuit properties that give sufficiency conditions under which the two delay modes give the same value but we have not closed the gap with precise necessary and sufficient conditions under which those two delay modes give the same value. Finally, while we see the immediate practical applications of this work in certified timing verification and delay fault testing, we hope that resolution of the issues discussed in this section will ultimately eliminate the need of timing simulation for synchronous digital circuits altogether.

Acknowledgment

The authors acknowledge V. Agrawal for questioning the accuracy of floating delay computation; this probing question motivated us to pursue this work. Thanks also to R. McGeer and R. Rudell, for interesting discussions on delay computation. Thanks to the reviewers for constructive comments and to Reviewer 3 for suggesting the use of finite state machine for transition delay computation.

References

[1] P. Antognetti and G. Massobrio, *Semiconductor Device Modeling with SPICE*. New York: McGraw Hill, 1988.

[2] J. Benkoski, E. Meersch, L. Claesen, and H. De Man, "Efficient algorithms for solving the false path problem in timing verification," in *Proc. Int. Conf. on Computer-Aided Design*, 1987.

[3] D. Brand and V. Iyengar, "Timing analysis using functional analysis," in *IEEE Trans. Comput.*, vol. 37, Oct. 1988.

[4] R. Bryant, "Graph-based algorithms for Boolean function manipulation," in *IEEE Trans. Comput.*, vol. C-35, pp. 677–691, Aug. 1986.

[5] H.-C. Chen and D. Du, "Path sensitization in critical path problem," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 2, pp. 196–207, Feb. 1993.

[6] S.-W. Cheng, H.-C. Chen, D. Du, and A. Lim, "The role of long and short paths in circuit performance optimization," in *Proc. Design Automation Conf.*, 1992.

[7] S. Devadas, K. Keutzer, and S. Malik, "Computation on floating mode delay in combinational logic circuits: Theory and algorithms," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 12, pp. 1913–1923, Dec. 1993.

[8] S. Devadas, K. Keutzer, S. Malik, and A. Wang, "Event suppression: Improving the efficiency of timing simulation for synchronous digital circuits," *IEEE Trans. Computer-Aided Design*, vol. 13, no. 6, pp. 814–822, June 1994.

[9] ——, "Computation of floating mode delay in logic circuits: practice and implementation," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 12, pp. 1924–1936, Dec. 1993.

[10] N. P. Jouppi, "TV: An nMOS timing analyzer," in*Proceedings, Third Caltech Conference on* vlsi, R. Bryant, Ed. Rockville MD: Computer Science, 1983, pp. 71–85.

[11] H. Jyu, S. Malik, S. Devadas, and K. Keutzer, "Statistical timing analysis of combinational logic circuits," *IEEE Trans. VLSI Syst.*, vol. 1, no. 2, pp. 126–137, June 1993.

[12] T. Larrabee, "Test pattern generation using Boolean satisfiability," in *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 4–15, Jan. 1992.

[13] P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design*. New York: Kluwer Academic, 1991.

[14] John K. Osterhout, "Crystal: A timing analyzer for nmos vlsi circuits," in *Proceedings, Third Caltech Conference on* vlsi, R. Bryant, Ed. Rockville MD: Computer Science, 1983, pp. 57–69.

[15] C.-J. Seger and R. E. Bryant, "Modelling of circuit delays in symbolic simulation," in *IFIP Int. Wkshp. on Applied Formal Methods for Correct VLSI Design*, Nov. 1989, pp. 625–639.

Srinivas Devadas (S'87–M'88) received the B. Tech degree in electrical engineering from the Indian Institute of Technology, Madras in 1985 and the M.S. and Ph.D. degrees, also in electrical engineering from the University of California, Berkeley, in 1986 and 1988 respectively.

Since August 1988, he has been with the Massachusetts Institute of Technology, Cambridge, and is currently an Associate Professor of Electrical Engineering and Computer Science. He held the Analog Devices Career Development Chair of Electrical Engineering from 1989 to 1991. His research interests span all aspects of synthesis of VLSI circuits, with emphasis on optimization techniques for synthesis at the logic, layout and architectural levels, testing of VLSI circuits, formal verification, hardware/software co-design, design-for-testability methods and interactions between synthesis and testability of VLSI systems..

Dr. Devadas is a member of ACM. He has received five Best Paper awards at CAD conferences and journals, including the 1990 IEEE Transactions on CAD Best Paper award. In 1992, he received a NSF Young Investigator Award. He has served on the technical program committees of several conferences and workshops including the International Conference on Computer Design and the International Conference on Computer-Aided Design

Kurt Keutzer (S'83–M'84–SM'94) received the B.S. degree in mathematics from Maharishi International University in 1978 and the M.S. and Ph.D. degrees in computer science from Indiana University in 1981 and 1984, respectively.

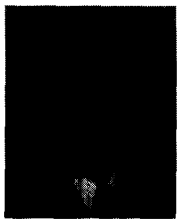
In 1984 he joined AT&T Bell Laboratories where he worked to apply various computer-science disciplines to practical problems in computer-aided design. In 1991 he joined Synopsys, Inc. where he continues his work as Director of Research. His research in technology mapping led to the inclusion of a paper in the anthology "Twenty-five Years of Electronic Design Automation". He presently serves on the editorial boards of three journals: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems; Formal Methods in System Design; and Integration—The VLSI Journal.

Dr. Keutzer currently serves on the technical program committees of DAC and EuroDAC and has served on numerous other technical program and executive committees in recent years. His investigations into synthesis for testability, asynchronous synthesis, and timing verification have led to DAC Best Paper Awards in 1990 and 1991, as well as an ICCAD Distinguished Paper citation in 1991 and an ICCD Best Paper Award in 1992.

Sharad Malik (S'88–M'90) received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, New Delhi in 1985 and the M.S. and Ph.D. degrees in computer science from the University of California, Berkeley in 1987 and 1990, respectively.

Currently he is an Assistant Professor with the Department of Electrical Engineering, Princeton University, Princeton, NJ. His research interests are in the synthesis and verification of digital systems.

Dr. Malik has received the President of India's Gold Medal for academic excellence (1985), the IBM Faculty Development Award (1991), an NSF Research Initiation Award (1992), a Best Paper Award at the IEEE International Conference on Computer Design (1992), the Princeton University Engineering Council Excellence in Teaching Award (1993, 1994), the Walter C. Johnson Prize for Teaching Excellence (1993), Rhinestein Award for Junior Faculty (1994) and the NSF Young Investigator Award (1994).



Albert Wang (S'84-M'89) received the B.S. degree in computer engineering and applied mathematics from the University of California, San Diego in 1984 and the Ph.D. degree in computer science from the University of California, Berkeley in 1989.

He is a Staff Research Engineer at Synopsys Inc., Mountain View, CA. His research interest has been in the area of synthesis, optimization and verification of synchronous and asynchronous digital circuits for area, speed, and power. His current interest involves all aspects of computer-aided designs of embedded systems, with emphasis on retargetable compilation technologies.

Dr. Wang received the Darlington Award from IEEE Circuits and Systems Society in 1987. His thesis shared the Sakrison Memorial Prize for the best dissertation in EECS at Berkeley in 1989.