More Accurate Polynomial-Time Min-Max Timing Simulation *

Supratik Chakraborty David L. Dill
Computer Systems Laboratory
Stanford University
Stanford, CA 94305, USA
supratik@celery.stanford.edu dill@cs.stanford.edu

Abstract

We describe a polynomial-time algorithm for min-max timing simulation of combinational circuits. Our algorithm reports conservative bounds on the propagation delays from each primary input to each gate, for use in the timing verification of fundamental-mode asynchronous circuits. A new reconvergent fanout analysis technique is presented. Our algorithm produces more accurate results than previous polynomial-time (and some exponential-time) algorithms in the presence of reconvergent fanouts.

1. Introduction

Timing simulation is an important tool for high-performance asynchronous circuit design. However, statistical variations in IC processing conditions, operating conditions, etc., result in uncertainties in component delays which need to be taken into consideration when verifying timing-dependent circuit behavior, e.g., hazard-detection in asynchronous circuits. Timing simulation that considers component delays to vary within specified intervals and determines upper and lower bounds on signal transition times is called *min-max timing simulation*.

Exact min-max timing simulation is computationally intractable [13]. The complexity of the problem can be attributed to two primary causes. First, exponentially long sequences of transitions can result from a single input change in certain circuits [8]. Many event-driven simulators suffer from exponential worst-case behavior because of this problem [21, 2, 15, 10, 20, 19, 9]. Second, the number of different possible gate delay combinations can be exponential in the circuit-size. Since the circuit output depends on the choice of gate delays in general, determining the exact minimum and maximum gate switching times involves examining all such delay combinations in the worst-case. Several min-max timing simulators [15, 14, 20, 9] suffer from this problem in addition to the event proliferation problem. Although pruning techniques [8] can be used to reduce the number of events to be simulated, the effectiveness of pruning strongly depends on the choice of the interval during which the output is observed.

Fortunately, listing *all* of the transitions of a signal is not necessary for most practical purposes. Usually, it is sufficient to determine whether a signal is changing or stable during a temporal window. For example, hazard analysis in asynchronous circuits depends on whether a gate input has a spurious transition on it during the window when the gate is sensitized to this input. This suggests that a good tradeoff between accuracy and efficiency can be obtained by abstracting sequences of transitions using multi-valued logic and conservatively approximating the earliest and latest transition times of gates. The approximations should be conservative, so that our analysis never fails to report timing violations (e.g., fundamental-mode constraint violations) when they can occur. By being conservative, we, however, admit the possibility of false negatives: situations where our simulation predicts timing violations although these never actually occur in the real circuit. A major goal of this paper is to present techniques to increase the accuracy of min-max timing simulation, while remaining efficient.

Most existing timing simulators require the user to specify when the circuit inputs transition relative to each other. However, several practical asynchronous design styles, including [18, 23], generate circuits that have very few constraints on their input transition times for correct operation. Therefore, what we really need is a min-max timing simulator that determines bounds on the signal propagation delays from each primary input to each gate output for a given set of primary input transitions. These bounds can then be used to determine constraints for proper circuit operation, e.g., minimum delay between successive input transitions in fundamental-mode circuits. Previously, Ulrich, Lentz, Demba and Razdan [22] have reported a polynomial-time

*This work was supported by a grant from the Semiconductor Research Corporation, ref. 95-DJ-389

min-max timing simulator of this type. However, their results are extremely pessimistic in the presence of nested reconvergent fanouts. The challenge is to make the simulation as accurate as possible while remaining efficient.

In this paper, we first describe an abstract 13-valued signal algebra, similar to that used in [6], to succinctly represent signal transitions. This enables us to circumvent the event-proliferation problem, while retaining information about the initial and final states of the signal and about whether multiple transitions occurred in between. It also proves useful in detecting potential hazards in the circuit. Hazard analysis using multi-valued logic has been used in several different contexts over the last three decades [11, 4, 12, 16, 6]. More recently, Kung [16] proposed a 9-valued algebra for use in the hazard-free implementation of asynchronous circuits. Our 13-valued algebra differs from Kung's 9-valued algebra in that four additional logic values are used to model signal transitions which are either unknown to begin with or eventually become undefined. Waveforms like these arise during the analysis of certain types of circuits, e.g., 3D circuits with conditional inputs [23]. However, there does not exist any logic value in Kung's algebra to model such waveforms.

We then present a polynomial-time algorithm for min-max timing simulation of combinational circuits. Unlike other simulators [21, 10, 19], our simulation algorithm does not assume prior knowledge of when the primary inputs transition relative to each other, and computes bounds on the signal propagation delays from *each* primary input to each gate in the circuit. In order to improve the accuracy of simulation, we also present a polynomial-time reconvergent fanout analysis technique for detecting ordering of signal transitions in the circuit. Unlike previous exponential-time techniques [2, 19], our reconvergent fanout analysis is not based on event-graphs and does not require knowledge of the primary input transition times.

Our simulator produces timing results more accurate than those obtainable with previous polynomial-time min-max timing simulation algorithms [3, 22], particularly in the presence of nested reconvergent fanouts. Our polynomial-time reconvergent fanout analysis is also able to detect some event-orderings not detectable with a previously published exponential-time simulator [19]. As an application, we have used our min-max timing simulation algorithm to build an efficient timing analysis tool for 3D asynchronous circuits [23]. This work is reported in a companion paper [5].

Despite its advantages, our approach has its own limitations. We have been able to identify two situations where our algorithm is forced to behave conservatively and produce approximate results. These are described in more detail later in the paper. We conjecture that such situations rarely arise in real-life circuits and cannot be analyzed exactly unless one exhaustively simulates all possible events and gate delay combinations.

2. Delay model

Min-max timing simulation requires accurate characterization of the bounds on gate and wire delays. In this paper, we do not address the question of how to characterize these delays, but instead assume that we are given the delay interval for each gate. A delay interval associated with a gate is assumed to represent uncertainty in the *propagation delay* through the gate. In order to model delays more accurately, different delay intervals can be associated with a gate depending on whether the output is rising or falling, the input pattern, output capacitive load, etc. Since the delays of gates in the same chip track each other fairly well, the *tracking delay model* of Lam and Brayton [17] may be used to model gate delays. As a default, one may also use the nominal delays provided by the manufacturer and introduce a percentage variation (e.g., $\pm 10\%$).

3. Combinational timing analysis

In this section, we describe our approach to performing timing simulation of combinational circuits. The timing analysis of asynchronous circuits uses this as the core simulation engine.

3.1. 13-valued signal algebra

We use a *ternary logic model* to represent the initial and final states of a transitioning signal. A signal can transition to or from a *stable 0*, a *stable 1* or an *unknown value, X*, which may be 0, 1 or changing repeatedly.

Each signal waveform is represented as a triple: $\langle b, m, e\rangle$, where $b, e \in \{1, 0, X\}$, and $m \in \{1, 0, \uparrow, \downarrow, X\}$. b and e represent the ternary logic values at the beginning and end of the signal transition, while m represents the intermediate behavior. There are two *constant* values: $\langle 1, 1, 1\rangle$ and $\langle 0, 0, 0\rangle$; two *clean transitions*: $\langle 1, \downarrow, 0\rangle$ and $\langle 0, \uparrow, 1\rangle$; four *hazards*: $\langle 0, X, 0\rangle$, $\langle 1, X, 1\rangle$, $\langle 0, X, 1\rangle$ and $\langle 1, X, 0\rangle$; one *completely undefined* signal: $\langle X, X, X\rangle$; and four signals which either start off undefined or eventually become undefined: $\langle X, X, 1\rangle$, $\langle X, X, 0\rangle$, $\langle 1, X, X\rangle$ and $\langle 0, X, X\rangle$. Whenever the m-component of a signal waveform is X, there are *potentially* more than one transitions on it. An m-component of $\uparrow$ or $\downarrow$ indicates *exactly* one transition, while a 0 or 1 value of the m-component indicates no transition.

3.2. Modeling gates

Every Boolean function can be extended to operate on values from the 13-valued algebra. Consider an assignment of values $\langle b_i, m_i, e_i\rangle$ to n input variables. We define a *trajectory* to be a sequence of assignments of Boolean values to the input variables, where exactly one variable changes value from one element of the sequence to the next. The assignment of 13-valued waveforms to the inputs represents a set of trajectories in the points of an n-dimensional Boolean hypercube, where each trajectory starts in a subcube defined by the vector of start values (b_i for each input i) and ends in a subcube defined by the vector of end values (the e_i values).

When an input variable has a constant binary value for m_i, the Boolean value of the variable must have the value m_i at every point in the trajectory. If $m_i = \uparrow$ or $m_i = \downarrow$, the Boolean value of the variable must change once, in the proper direction, along every trajectory. If $m_i = X$, the Boolean value of the variable may change arbitrarily in the trajectory. Let $\langle b_r, m_r, e_r \rangle$ represent the value of the Boolean function B on the 13-valued input values. If B is constant throughout the start subcube, b_r is the constant value. Otherwise, $b_r = X$. The definition of e_r is similar. If the value of B throughout every trajectory is a constant 1 or 0, m_r has that value. If the value of B changes from 0 to 1 (1 to 0) exactly once on every trajectory, m_r has the value $\uparrow$ ($\downarrow$). Otherwise, m_r has the value X. For efficiency, a table of the behavior of standard gates on all 13-valued inputs can be precomputed. This table can be compressed by observing that the b, m and e components of the output waveform are determined solely by the b, m and e components, respectively, of the inputs. The b and e components are variables in a 3-valued system, while the m component is a 5-valued variable. Consequently, a table of size 5^n suffices for an n input function. However, three table lookups are needed in order to compute the output of a function. In contrast, one lookup in a table of size 13^n is needed using the representation of [6].

13-valued simulation proceeds in the obvious way. The primary inputs are set to 13-valued signals representing their transitions, then the values are propagated through each gate in the circuit using the 13-valued table for the gate type, until the outputs of all gates have been computed. For example, Figure 1 shows the simulation of falling transitions on inputs $A1$ and $A2$ of a circuit constructed from examples in [15]. In this example, gates $G5$ and $G10$ have potential static-0 hazards on their outputs.

It can be proved that the 13-valued simulation algorithm indicates a hazard at the output of a Boolean gate in a combinational circuit *if and only if* there exists a set of wire delays such that simulation using transitions between 0, 1 and X results in a hazard in the circuit. However, when wire delays are bounded by constants, the simulation gives only a conservative approximation to the actual hazard behavior. For example, suppose that in Figure 1 all gates have non-zero delays and all wires have zero delay. It follows that the $\langle 1 \downarrow 0 \rangle$ transition on $G2$ reaches $G5$ before the $\langle 0 \uparrow 1 \rangle$ on $G4$, thereby preventing the hazard on $Y1$. However, 13-valued simulation assumes an arbitrary delay in the wire connecting $G2$ and $G5$ – a static-0 hazard is therefore

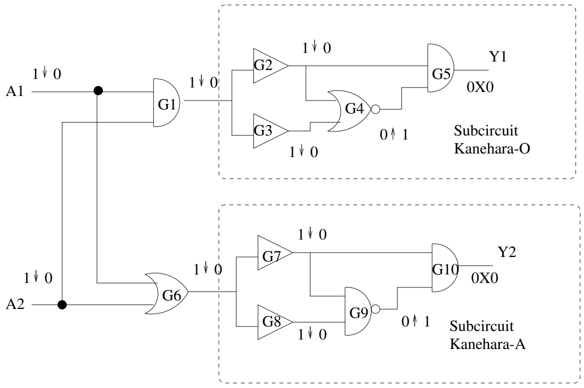


Figure 1. Example of 13-valued simulation

predicted on $Y1$.

3.3. Basic min-max timing simulation

We first describe the basic min-max timing simulation algorithm without reconvergent fanout analysis. For purposes of simulation, wire delays are modeled by inserting a buffer along each wire and associating the delay with the buffer; in the transformed circuit, all delays are associated with gates and wires have zero delay. To keep the notation simple, we assume that the delay from input p of gate G to its output lies in the interval $(t_{G,p}, T_{G,p})$. It should be understood that this interval is actually a function of the signal values, making it possible to model input-dependent delays and distinguish between rising and falling delays.

There are two inputs to our simulator. The first is a combinational circuit with minimum and maximum delay annotations on each gate. The second is an input stimulus consisting of 13-valued signals associated with the primary inputs. The output of our simulator is an assignment of a 13-valued signal to each gate G, conservatively modeling the output waveform resulting from the input. In addition, G is annotated with a set of intervals, one *for each primary input* i, representing the shortest and longest signal propagation delays from i to the output of G, for the given input stimulus. Note that the paths in the circuit through which a transition on primary input i propagates to the output of G may vary depending on the applied primary input stimulus. Consequently, signal propagation delays computed for one input stimulus are not valid, in general, for another stimulus.

The following data structures, which are associated with each gate and primary input G in the circuit, are updated as the algorithm executes:

1. A $value$ field which stores a 13-valued signal indicating the waveform at the gate output.

2. An array $del[1 \ldots max\_primary\_inputs]$ of tuples (t, T). $del[i].t$ stores a lower bound of the propaga-

tion delay of a transition on primary input i to gate G. $del[i].T$ stores an upper bound of the same delay.

The top-level algorithm in the timing simulation flow is depicted in Figure 2. For each gate, a 13-valued simulation is first performed to give an initial output waveform ignoring any information about the ordering of the input transitions. If the result is not a constant value or $\langle XXX\rangle$, a more detailed and costly analysis is performed by function *Min_Max_Analyze*, which utilizes information about the ordering of the input transitions to refine the 13-valued output of G, if possible, *e.g.*, $\langle 0X1\rangle$ refined to $\langle 0\uparrow 1\rangle$. It also yields information about the signal propagation delays from the primary inputs to the output of G.

Main Algorithm Steps:

1. Preprocessing:
   Sort the gates in topological order;
2. Initialization:
   (a) **for each** gate and primary input G
         for each primary input i
           G.del[i] $= (+\infty, -\infty)$;
           / G not affected by transition on i */*
   (b) **for each** transitioning primary input i
         i.del[i] $= [0, 0]$;
3. Simulation:
   for each gate G in topological order
     (a) Thirteen_Valued_Simulate(G);
     (b) if ($G.value \neq \langle 000\rangle, \langle 111\rangle$, or $\langle XXX\rangle$)
         Min_Max_Analyze(G);

Thirteen_Valued_Simulate(G: gate)

Looks up a precomputed table to determine
the 13-valued waveform on output of G from
the 13-valued waveforms on its inputs.
Information about ordering of input transitions
is neglected here.

Figure 2. Top-level algorithm

In order to explain how function *Min_Max_Analyze* works, we first define the notion of a gate G being sensitized to a transition on one of its inputs p: G is said to be sensitized to a transition on p if the transition on p can cause a transition on the output of G. The action of *Min_Max_Analyze* can now be summarized as follows: We look at each input p of G and determine whether G may be sensitized to the transition on p by analyzing the ordering of transitions on the other gate inputs. If G is sensitized to p, the signal propagation delays from the primary inputs to p are used to compute the corresponding delays from the primary inputs to G. Otherwise (i.e., if G is desensitized to the transition on p), we may be able to discover that a potential hazard on the output of G actually does not occur.

However, there is a source of inefficiency in this method. In order to determine whether G is sensitized to p, we must be able to detect any temporal ordering that exists between the transitions on the gate inputs. For example, consider an AND gate G with a rising input p and a falling input q (Figure 3). If q always falls before p rises, then G is not sensitized to the transition on p. Unfortunately, the problem of detecting all possible temporal orderings between gate input transitions and utilizing this information to determine the sensitizability of G to p is not efficiently solvable for circuits with reconvergent fanouts and bounded component delays. The problem becomes worse when the primary input transitions times are unspecified, as is the case here.

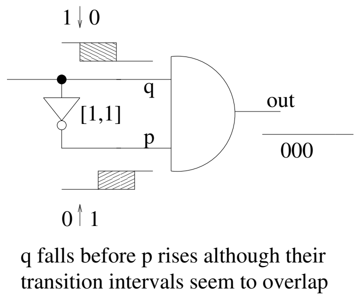
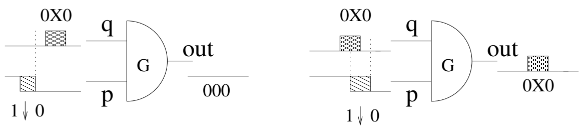


Figure 3. Example of input transition ordering

For purposes of efficiency, we will therefore compute a conservative approximation of the values on the other gate inputs when determining the sensitizability of G to the transition on p. The approximation is conservative in the sense that it never fails to report a sensitized gate. Our approximation strategy works as follows: Let the 13-valued waveform on gate input q (different from p) be $\langle b_q, m_q, e_q\rangle$. If we know that q always transitions before p transitions, then the value to which q transitions, i.e., $\langle e_q, e_q, e_q\rangle$, is used in determining the sensitizability of G to p. Similarly, if we know that q transitions after p, $\langle b_q, b_q, b_q\rangle$ is used. However, if we are unable to detect a temporal ordering between the transitions on p and q, then we use the value $\langle b_q, m_q, e_q\rangle$ in determining whether G is sensitized to p.

The rationale behind our approximation strategy is best understood from a simple example. Consider an AND gate with inputs p and q transitioning as shown in Figure 4. If we know that q starts transitioning only after p has transitioned (Figure 4(a)), then we can use the constant value at the beginning of q's transition ($\langle 000\rangle$) to determine that the gate is desensitized to the transition on p. A similar situation occurs if we know that q stabilizes to its final constant value before p starts transitioning. However, if we are unable to detect any temporal ordering between the transitions on p and q, we leave the value of q unchanged at $\langle 0X0\rangle$, so that the AND gate is potentially sensitized to the transition on p (Figure 4(b)). Thus, the value of q used is, in a sense, the "most-sensitizing value" that we can compute when determining the sensitizability of G to p. Note that in situations



G desensitized to transition on p

(a)

G potentially sensitized to transition on p

(b)

Figure 4. Example of gate sensitization and desensitization

where an ordering between the transitions on p and q cannot be detected, we can use information about the functionality of G to obtain a more refined most-sensitizing value of q. However, this does not yield any additional information when determining the sensitizability of G to p. For example, in the AND gate of Figure 4(b), the value of q that sensitizes G to the transition on p is $\langle 111\rangle$. Our approximation strategy however assigns the value $\langle 0X0\rangle$ to q, which allows the possibility of q being in the logical 1 state when p is transitioning. As a result, the gate remains sensitized to p. In summary, our strategy of leaving the value of q unchanged keeps the gate sensitized to p (if at all it was sensitized in the first place), without having to worry about the gate type and the sensitizing input value to use for each gate type. The simplicity of this method was the primary motivation behind our choice of the approximation strategy.

The implementation of function *Min_Max_Analyze* (Figure 6) calls function *Most_Sensitizing_Value* to compute qep_i, the most-sensitizing value on gate input $\underline{q}$ at the $\underline{e}$arliest time a transition on primary input $\underline{i}$ reaches $\underline{p}$. Similarly, it computes qlp_i, the most-sensitizing value at the latest time that a transition on i reaches p (see Figure 5). If an ordering of gate input transitions is detected during this process, the ordering information is used to determine whether a potential hazard on the output of gate G actually does not occur. The qep_i values are then used to determine whether G is sensitized to p at the earliest time a transition on i reaches p. If so, we check to see whether the earliest change in p could cause the output of G to change earlier than previously estimated. If the answer is "yes", the minimum signal propagation delay from primary input i to G is updated to this new smaller value. *Min_Max_Analyze* then performs a similar analysis for the maximum propagation delay from i to G using the qlp_i values. The entire process is repeated for each transitioning gate input p and primary input i.

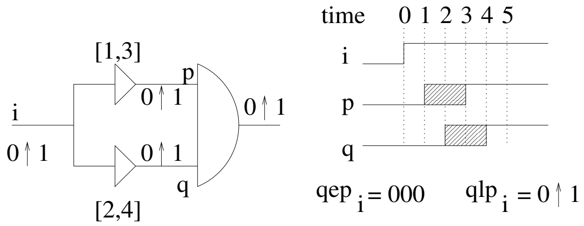


Figure 5. Computation of qep_i and qlp_i

<u>**Min_Max_Analyze** (**G: gate**)</u>

1. **for each** transitioning input p of G
   (a) **for each** other input q of G
      /* *Let qep_i = most-sensitizing value of q at the* */
      /* *earliest time a transition on i reaches p* */
      /* *qlp_i = most-sensitizing value of q at the* */
      /* *latest time a transition on i reaches p* */
      if (q.value $\neq \langle 000\rangle$, $\langle 111\rangle$ or $\langle XXX\rangle$)
         $\{qep_i, qlp_i$: pr. inp i affects $p\}$ =
            Most_Sensitizing_Value(G, p, q);
      else
         $qep_i = qlp_i =$
            q.value for all pr. inp. i affecting p;
   (b) **if** (G has a potential hazard)
      Use information about ordering between gate input transitions to determine if hazard actually does not occur;
      if (hazard actually does not occur)
         Update G.value to hazard-free value;
         if (new G.value == $\langle 000\rangle$ or $\langle 111\rangle$)
            G.del[i] = $(+\infty, -\infty)$ for all pr. inp. i;
            return;
   (c) Using qep_i values
         if G is sensitized to transition on p
         G.del[i].t = min(G.del[i].t, p.del[i].t $+t_{G,p}$);
      Using qlp_i values
         if G is sensitized to transition on p
         G.del[i].T = max(G.del[i].T, p.del[i].T $+T_{G,p}$);

Figure 6. Basic min-max analysis algorithm

For a gate G with inputs p and q, function *Most_Sensitizing_Value* computes the most-sensitizing value of q at the earliest and latest times that a transition on a primary input reaches p. In order to accomplish this, the implementation of *Most_Sensitizing_Value* (Figure 7) first determines whether the transitions on p and q are temporally ordered. If both p and q are affected by transitions propagating from only one primary input i, bounds on the signal propagation delays from i to p and q can be used to detect such an ordering. However, if there are several primary inputs affecting p or q or both, the simple version of the algorithm presented here conservatively assumes that the transitions on p and q could overlap. In the next subsection, we discuss a much more sophisticated technique that performs reconvergent fanout analysis to detect ordering of signal transitions in a circuit. Information about such ordering of transitions can then be used to compute more accurate values of qep_i and qlp_i.

Most_Sensitizing_Value(G: gate; p, q: gate input)

1. (i) q_transitions_after_p =
    p_transitions_after_q = $false$;
  (ii) **if** (both p and q affected by only one pr. inp. i)
    if (q.del[i].t $>$ p.del[i].T)
      q_transitions_after_p = $true$;
    else if (p.del[i].t $>$ q.del[i].T)
      p_transitions_after_q = $true$;
2. **if** (q_transitions_after_p)
    for each pr. inp. i affecting p
      $qep_i = qlp_i = \langle b_q, b_q, b_q \rangle$;
  else if (p_transitions_after_q)
    for each pr. inp. i affecting p
      $qep_i = qlp_i = \langle e_q, e_q, e_q \rangle$;
  else
    / No ordering of trans. on p and q detected */*
    for each pr. inp. i affecting p
      if (both p and q affected by only one pr. inp. i)
        (a) **if** (q.del[i].t $>$ p.del[i].t)
          $qep_i = \langle b_q, b_q, b_q \rangle$;
        (b) **if** (q.del[i].T $<$ p.del[i].T)
          $qlp_i = \langle e_q, e_q, e_q \rangle$;
        In all other cases, $qep_i = qlp_i = \langle b_q, m_q, e_q \rangle$;
3. **return** $\{qep_i, qlp_i\}$;

Figure 7. Simple version of algorithm for finding most-sensitizing values

The complexity of the algorithm presented above is $O(n_g.n_{pi}.n_{fanin}^2)$ where n_g is the number of gates, n_{pi} is the number of primary inputs and n_{fanin} is the maximum fanin of a gate. For technological reasons, n_{fanin} is a small number, typically 4 or 5. The observed complexity of the algorithm is therefore $O(n_g.n_{pi})$. When the relative transition times of all primary inputs are known, a greatly simplified version of this algorithm can be applied. Since it is no longer necessary to maintain separate propagation delays for each input, the factor of n_{pi} is reduced to 1.

Let us now apply our basic min-max timing simulation algorithm to the circuit shown in Figure 8. This is essentially the same circuit that we examined in Figure 1, the only difference being that each gate is now associated with a delay interval ($[min, max]$) representing the delay from each input of the gate to its output. All wire delays are assumed to be zero. The result of our min-max timing simulation is shown in Figure 8. Note that the circuit is symmetric with respect to $A1$ and $A2$ for the given primary input stimulus. Consequently, the signal propagation delay from $A2$ to a gate is exactly the same as the corresponding delay from $A1$ to the same gate. A closer inspection of the timing simulation results indicates that our basic algorithm gives accurate signal propagation delay bounds and 13-valued waveforms for $G1$, $G2$, $G3$, $G4$, $G6$, $G7$ and $G8$. The bounds computed for $G9$ are conservative although the 13-valued waveform on its output is accurate. However, for gates $G5$ and $G10$, both the 13-valued waveforms and the delay bounds are conservative. In the next subsection, we show how our polynomial-time reconvergent fanout analysis technique enables us to compute accurate bounds and 13-valued waveforms for each of these gates.

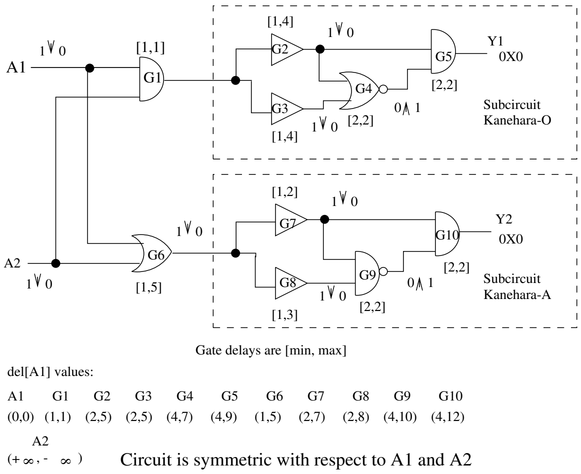


Figure 8. Basic min-max algorithm applied to our example.

3.4. Reconvergent fanout analysis

It is well-known that the accuracy of min-max timing simulation can be significantly improved if reconvergent fanouts in circuits are analyzed carefully [3]. This might be able to identify ordering of transitions on gate inputs which would otherwise seem to overlap in their transition intervals. In this subsection, we present a new technique for analyzing reconvergent fanouts in circuits and for detecting temporal ordering of signal transitions. Our analysis has a worst-case complexity that is polynomial in the circuit-size, and unlike previous approaches [2, 19], is not based on event-graphs.

Given a combinational circuit and a set of primary input transitions, we first define the following two relations on the gates (by gates, we mean internal gates and primary inputs):

- Gates G_1 and G_2 are said to satisfy G_2 *waits_for* G_1, if G_2 must wait for the transition on G_1 to propagate to one of its inputs before it can start transitioning. Figure 9(a) shows an example of this relation. In this example, the output of $G2$ cannot start transitioning until the $\langle 0 \uparrow 1 \rangle$ transition on $G1$ propagates to $G2$. Therefore, $G2$ *waits_for* $G1$. Similarly, $G2$ also *waits_for* $G3$. We represent this relation graphically by a *waits_for graph*, where G_1 and G_2 are represented as nodes and a solid directed edge is drawn from G_1 to G_2 (Figure 9(a)). We also assign a weight, $d_{1,2}$, to this edge, where $d_{1,2}$ = minimum propagation delay from G_1 to G_2. Mathematically, if t_i denotes the time when G_i *starts transitioning*, $t_2 \geq t_1 + d_{1,2}$.

- G_1 and G_2 are said to satisfy G_2 *yields_to* G_1, if G_2 must end transitioning when the last transition on G_1 propagates to G_2 or even before. Figure 9(b) shows an example. Here, the output of $G2$ must settle to its final value of 1 either when the $\langle 1 \downarrow 0 \rangle$ transition on $G1$ reaches $G2$ or before (if the $\langle 1 \downarrow 0 \rangle$ transition on $G3$ reaches $G2$ before the transition on $G1$ does). Consequently, $G2$ *yields_to* $G1$. In a similar manner, $G2$ also *yields_to* $G3$. This relation is graphically represented by a *yields_to graph*, where a dotted directed edge is drawn from G_2 to G_1 (Figure 9(b)). If $D_{1,2}$ represents the maximum propagation delay from G_1 to G_2, we assign the weight $-D_{1,2}$ to this edge. Mathematically, if T_i denotes the time when G_i *ends transitioning*, then $T_2 \leq T_1 + D_{1,2}$.

Each of the above two relations defines a partial order on the gates, and the corresponding graphs are therefore directed acyclic graphs (DAGs). Figure 10 shows these DAGs for the circuit and primary input stimulus of Figure 8, superimposed on each other. For the sake of clarity, we have not shown all the edges that would result if we computed the transitive closure of each relation.

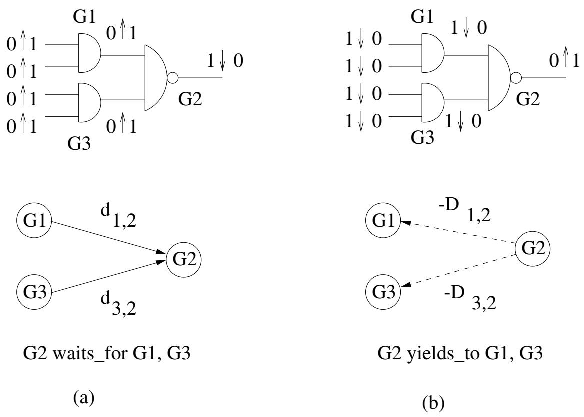


Figure 9. Illustration of *waits_for* and *yields_to* relations.

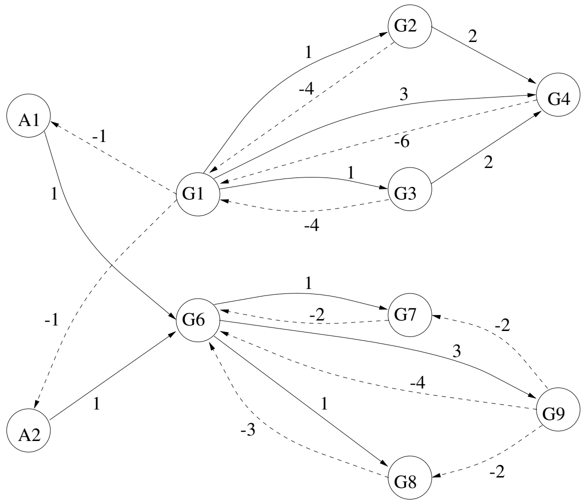


Figure 10. *waits_for* and *yields_to* DAGs for Figure 8

Let us now see how the information represented in Figure 10 can be used to detect ordering of signal transitions and increase the accuracy of simulation. First, note that a *waits_for* (solid) edge of weight 2 exists from $G2$ to $G4$. This implies that $t_4 \geq t_2 + 2$. However, we know from our 13-valued annotations (Figure 8) that each of $G4$ and $G2$ has a single transition on it. Consequently, t_4 = transition time of $G4$ and t_2 = transition time of $G2$. The *waits_for* edge therefore implies that $G4$ transitions at least 2 time units after $G2$. It follows that the potential hazard on $Y1$ (Figure 8) actually does not occur. Now consider the following subgraph of the superimposed DAGs of Figure 10: $G7 \xrightarrow{-2} G6 \xrightarrow{3} G9$. The information represented by this subgraph can be expressed by the following two inequations:

$$T_6 \geq T_7 - 2; \quad t_9 \geq t_6 + 3$$

which, on rearranging, gives

$$t_9 - T_7 \geq 1 + (t_6 - T_6).$$

Since we know from our 13-valued annotations (Figure 8) that each of $G6$, $G7$ and $G9$ has a single transition on it, we have $t_6 = T_6$ = transition time of $G6$, t_9 = transition time of $G9$, and T_7 = transition time of $G7$. Therefore, $G9$ transitions at least 1 time unit after $G7$; it follows that the hazard on $Y2$ (Figure 8) is eliminated as well. The information represented by the *waits_for* and *yields_to* DAGs can be further used to determine that if the transition on $A1$ reaches $G8$ at the latest possible time (8 time units after $A1$ transitions), then $G7$ would have already transitioned to 0 by that time, desensitizing $G9$ to the transition on $G8$. As a result, the maximum signal propagation delay from $A1$ to $G9$ is 9 time units (not 10 time units as computed in Figure 8).

The method for detecting event orderings outlined above can be generalized as follows. We first superimpose the *waits_for* and *yields_to* DAGs for a given circuit and primary input stimulus. A *directed path* in the resulting graph is a sequence of vertices $\langle G_1, G_2, \ldots G_k \rangle$ such that a *waits_for* or *yields_to* edge exists from G_i to G_{i+1} $\forall i \in \{1, \ldots k-1\}$. In order to detect whether gate G_j transitions after gate G_i, we now need to find a directed path, $\langle G_i, G_k, \ldots G_n, G_j \rangle$, from G_i to G_j in the superimposed graph, such that the following properties hold:

1. Each gate G_r (other than G_i and G_j) in the path has a single transition on it (no hazards).

2. The total weight of the path, which is the summation of the weights of edges in the path, is greater than 0.

3. If G_i has a potential hazard on it, the first edge in the path, $\langle G_i, G_k \rangle$, is a *yields_to* edge.

4. If G_j has a potential hazard on it, the last edge in the path, $\langle G_n, G_j \rangle$, is a *waits_for* edge.

The rationale behind this method can be best understood from a simple example. Consider a directed path (in the superimposed graph) from X to A, as shown in Figure 11. The information represented by this path can be expressed by the following system of inequations:

$$T_Y \geq T_X - D_1; \quad t_Z \geq t_Y + d_2$$
$$T_W \geq T_Z - D_3; \quad t_A \geq t_W + d_4$$

which can be reorganized to give

$$t_A - T_X \geq \; (-D_1 + d_2 - D_3 + d_4) + (t_Y - T_Y) + (T_Z - t_Z) + (t_W - T_W).$$

Since each of Y, Z and W has a single transition on it, the last three parenthesized terms reduce to 0. Therefore, if the sum of weights on the edges, $(-D_1 + d_2 - D_3 + d_4)$, is greater than 0, we have $t_A > T_X$. In other words, A starts transitioning only after X has finished transitioning.

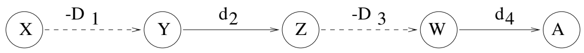


Figure 11. A path in the superimposed DAGs.

The problem of finding a path of positive length can now be solved with the Bellman-Ford [7] longest path algorithm, with the additional constraints 1, 3 and 4 mentioned above. However, this has a complexity $O(n^3)$ where n is the number of nodes in the graph. In order to reduce the complexity, we can use several approximations. For example, we can compute the transitive closure of the *waits_for* and *yields_to* relations and then look for paths of a given maximum length (say 2) in the resulting superimposed graph. To reduce the complexity further, we may also choose not to compute the transitive closures completely. Our implementation uses this approximation strategy, which still detects a significant number of event orderings.

We now describe how to efficiently compute the *waits_for* and *yields_to* DAGs, for a given circuit and input stimulus. We process gates in the circuit in topological order. Consider a gate G with a potential transition on its output, and a transitioning input p of G. To determine whether $G \; waits\_for \; p$, we hold p at its initial state ($\langle b_p, b_p, b_p \rangle$) and allow all other transitioning inputs of G to transition. If a 13-valued evaluation of G now shows that G *does not* have a potential transition on its output, we know that G must wait for p to transition in order for it to transition. A *waits_for* edge is therefore drawn from p to G and assigned the weight $d_{p,G}$. We also consider another situation which often arises in practical circuits. Suppose the set of transitioning inputs of G is $\{p_i, \ldots p_n\}$. If each $p_k \in \{p_i, \ldots p_n\}$ satisfies $p_k \; waits\_for \; G'$, for some gate G', then we must also have

$G\ waits\_for\ G'$. This is because a stable gate must wait for at least one of its inputs to transition, and each transitioning input, in turn, must wait for G' to transition. Once we have computed these simple relations, the transitive closure can be easily computed. While computing the edge weights in the *waits_for* graph, if we have the edges $G'' \xrightarrow{a} G' \xrightarrow{b} G$ and if the current weight of edge $G'' \rightarrow G$ is c, the new weight of $G'' \rightarrow G$ becomes $\max(c, a + b)$. The procedure for computing the *yields_to* DAG is very similar.

The complexity of our min/max algorithm (which computes the transitive closure partially and then looks for paths of maximum length 2) is now $O(n_g.n_{pi}.n_{fanin}^2.n_g^*)$, where n_g^* = number of incoming *waits_for* edges or outgoing *yields_to* edges at a gate. The worst-case value of $n_g^* = n_g$. However, in reality, n_g^* never grows beyond 10 for all our benchmarks, which includes all the ISCAS85 circuits.

Figure 12 shows how the results of our basic min-max timing simulation algorithm can be improved using our reconvergent fanout analysis technique. Note that $G9.del[A1] = [4, 9]$ and $G7.del[A1] = [2, 7]$, so the transitions on $G7$ and $G9$ would appear to overlap if we used only the del arrays, as in [3]. If we were to analyze these circuits using the reconvergent fanout analysis proposed in [22], then we would find that in subcircuit *Kanehara-O*, a *glitch* is produced at $Y1$ by *P-machines* originating from $A1$ and $A2$, and *also* by an *S-machine* originating from $G1$. The analysis in [22] would therefore indicate the presence of a *glitch* on $Y1$. Similarly, their analysis would indicate the possibility of a *glitch* on $Y2$.

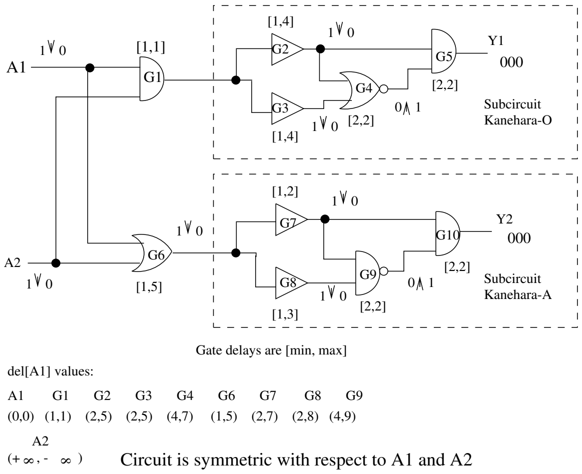
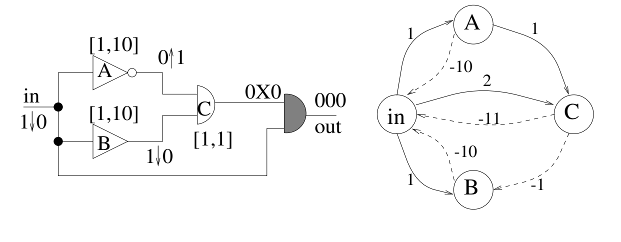


Figure 12. More accurate simulation with reconvergent fanout analysis.

In asynchronous and other timing-critical circuits, hazards generated at internal gates may actually get masked before reaching the primary outputs. Figure 13(a), adapted from [19], shows an example. All wire delays are assumed to be zero and the delay from each input of a gate to its output is indicated by a $[min, max]$ interval associated with the gate. An inspection of Figure 13(a) reveals that the $\langle 1 \downarrow 0 \rangle$ transition on in reaches the shaded AND gate before the output of gate C can transition from its initial logic 0 state. Consequently, the shaded gate becomes desensitized to the glitch that later appears on the output of C, and the output of the circuit remains stable in the logic 0 state. However,



(a) (b)

Figure 13. Event ordering not detected by MTV

simulators like MTV [19], Scald [21], Clover [10] or [2] fail to detect this phenomenon. Linderman notes this shortcoming of MTV in [19] and attributes it to the fact that MTV does not record the causality of hazardous transitions in its event-graph. As a result, it cannot detect temporal ordering of hazardous transitions. In contrast, event orderings of this type are easily detected using our reconvergent fanout analysis algorithm. Applying our technique to this example, we obtain the DAGs depicted in Figure 13(b). In the resulting graph, there exists a *waits_for* edge from in to C because each of the two inputs A and B of C must *wait_for* in. Consequently, C must *wait_for* in and the associated minimum delay is seen to be 2 time units. The presence of this *waits_for* edge in Figure 13(b) indicates to our algorithm that C must transition after in. Consequently, our algorithm correctly predicts the absence of a hazard on out. The ability to detect such temporal ordering involving hazardous transitions proves extremely useful in simulating timing-critical circuits.

3.5. Pathological cases

We have been able to identify two pathological situations where our simulator is forced to produce conservative approximations to the true circuit behavior. The first consists of a situation where a large number of interleaved events are generated on two inputs of a gate in response to a primary input stimulus. The timing of the interleaved events may be such that the gate output always remains constant. However, our algorithm approximates the sequence of interleaved events using $X's$, and detects two overlapping X transitions on the gate inputs, thereby producing an X at the output. The second situation arises when there does not exist any absolute ordering of the transitions on the inputs of a gate. However, the circuit might be so designed that for every possible ordering, the waveform at the gate output remains the same. Since our reconvergent fanout analysis can detect only absolute event orderings, we are unable to detect this phenomenon. However, we conjecture that such situations do not commonly occur in real-life circuits, and cannot be analyzed exactly unless one exhaustively simulates all possible events and gate delay combinations.

4. Experimental results

Our combinational timing simulation technique is primarily meant for use in the timing analysis and verification of asynchronous circuits. We have incorporated our algorithm in a tool for analyzing the timing constraints of 3D asynchronous finite state machines [23]. Detailed results obtained by applying our timing analysis tool to a suite of 3D benchmark circuits are presented in a companion paper [5]. In this section, we intend to illustrate the effectiveness of our reconvergent fanout analysis technique, and show that our algorithm runs fast enough even when analyzing circuits with a few hundred primary inputs and several thousand gates.

Table 1 shows how our reconvergent fanout analysis technique gives significantly less conservative constraints than those obtained with the basic min-max timing simulation algorithm. In this table, we report hold time (HT) constraints and fundamental-mode (FM) constraints of 3D circuits obtained using our 3D timing analysis tool [5]. Columns titled "-b" represent results obtained without using our reconvergent fanout analysis algorithm while those titled "-r" give results obtained using our reconvergent fanout analysis algorithm. For the 3D benchmarks in Table 1, the figures in the "-r" column also give the exact timing constraints for the circuits under consideration. In general, however, we expect our min-max timing simulation to yield conservative approximations to the true timing requirements in the worst-case, while yielding reasonably accurate results for most practical circuits.

For each gate in the benchmark circuits, we estimated gate delays for a Hitachi CMOS gate library [1]. Each gate has a nominal delay given by: $t = t_0 + K \times C$, where $C = 0.4\times$ *Effective fanout* and *Effective fanout* $= \sum_{all\ fanout\ gates}(Normalized\ load\ of\ fanout\ gate)$. The t_0, K and *normalized load* figures have been obtained from the Hitachi data book [1] and are different for rising and falling transitions. An inverter with a fanout of 4 has a rising delay of 2.12ns and a falling delay of 1.74ns in this library. The delay figures shown in Table 1 are normalized with respect to the delay of an inverter with fanout 4 (FO4 delay), which is assumed to be $2ns$ here. Our experiments assume the actual gate delays to vary within $(0.9 \times nominal\_delay, 1.1 \times nominal\_delay)$. We also assume zero delays for all wires, since we do not have post-layout information about wire-delays in the circuit. However, wire-delays can be modeled by insertion of delay buffers, as has been noted earlier. This would increase the number of gates in the circuit, thereby causing a degradation in the performance of our algorithm. Nevertheless, the number of delay buffers inserted is at most the number of wires (connections) in the circuit. Since our algorithm is essentially linear in the number of gates, we do not expect any significant performance degradation even if wire-delays are incorporated in the analysis.

3D Benchmark	HT-b (FO4 delays)	HT-r (FO4 delays)	FM-b (FO4 delays)	FM-r (FO4 delays)
ircv	7.140	7.140	**4.998**	**4.789**
ircv-bm	**5.130**	**4.488**	**2.995**	**1.959**
trcv	**6.525**	**5.594**	**3.201**	**2.715**
trcv-bm	**5.637**	**5.501**	2.073	2.073
tsend-bm	**5.915**	**5.860**	**2.737**	**2.600**
biu-dma2fifo	7.390	7.390	**3.402**	**2.550**
biu-fifo2dma	4.753	4.753	**2.074**	**1.357**
scsi-targ-send	4.172	4.172	**0.628**	**0.540**

Table 1. Comparison of results on 3D benchmarks

Since all our 3D asynchronous benchmarks were relatively small in size ($<$ 1000 gates), we decided to evaluate the efficiency of our algorithm by running it on the entire suite of ISCAS85 combinational benchmarks. In each case, we forced a rising transition on each primary input and simulated the entire circuit. *Our analysis gives upper and lower bounds on the signal propagation delay from each primary input to each gate in the circuit.* The results in Table 2 indicate that our algorithm is able to handle reasonably large circuits within very reasonable times. The CPU times shown are on a SUN SparcStation 2, and do not include the time required to read in the circuit and topologically sort the gates.

Unfortunately, the tools described in [9, 15, 14] were unavailable for performance and accuracy comparisons. We also did not find it meaningful to compare our tool with MTV [19] because MTV does not compute signal propagation delays from each primary input to each internal gate. Nevertheless, we have demonstrated that there exist situations where our simulator can detect ordering of hazardous transitions that MTV cannot detect.

5. Conclusion

We have described a polynomial-time min-max timing simulation algorithm for computing signal propagation de-

Benchmark	Gates	Inputs	Outputs	Analysis Time(s)
c6288	2416	32	32	72.5
c7552	3512	207	108	92.9
c1355	546	41	32	2.1
c5315	2307	178	123	33.2
c3540	1669	50	22	32.8
c2670	1193	157	64	18.1
c1908	880	33	25	17.3
c880	383	60	26	3.8
c499	202	41	32	2.8
c432	160	36	7	3.4
c17	6	6	2	0.0

Table 2. Computing signal propagation delays from each input in ISCAS85 benchmarks

lay bounds from each primary input to each internal gate in a combinational circuit. Such an algorithm is extremely useful for timing analysis and verification of asynchronous circuits. We have also described a new polynomial-time reconvergent fanout analysis technique that can detect absolute orderings of signal transitions, thereby producing more accurate results than previous polynomial-time min-max timing simulation algorithms.

We have applied the algorithm presented here for efficient timing analysis of 3D asynchronous finite-state machines. We believe that there are numerous other applications of our timing simulation algorithm in the domain of timing analysis of asynchronous circuits. The unique advantage of our algorithm is that, being polynomial-complexity, it can easily be applied to large circuits without significant performance degradation. One potential application could be simulation of asynchronous circuits to determine the count and extent of internal hazards, from which power requirement estimates could be obtained. Another application could be to determine if cross-talk occurs between two physically close lines in a circuit. This could happen, for example, if the transition intervals of these two lines overlap.

References

[1] *Hitachi High Speed CMOS Gate Array: HG62E Series Design Manual.*

[2] K. Bowden. Design goals and implementation techniques for time-based digital simulation and hazard detection. In *Proceedings of the 1982 International Test Conference*, pages 147–152, 1982.

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

[4] M. A. Breuer and L. Harrison. Procedures for eliminating static and dynamic hazards in test generation. *IEEE Transactions on Computers*, C-23:1069–1078, Oct. 1974.

[5] S. Chakraborty, D. L. Dill, K. Y. Yun, and K.-Y. Chang. Timing analysis for extended burst-mode circuits. In *Proceedings of the Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 1997.

[6] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell. Delay fault models and test generation for random logic sequential circuits. In *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pages 165–72, 1992.

[7] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[8] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Event suppression: Improving the efficiency of timing simulation for synchronous digital circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 13(6):814–822, June 1994.

[9] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Verification of asynchronous interface circuits with bounded wire delays. *IEEE Journal of VLSI Signal Processing*, 7(1-2):161–182, February 1994.

[10] D. Doukas and A. S. LaPaugh. Clover: A timing constraints verification system. *ACM International Workshop on Timing Issues in the Specification and Synthesis of Digitial Systems*, 1990.

[11] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9, Mar. 1965.

[12] G. Fantauzzi. An algebraic model for the analysis of logical circuits. *IEEE Transactions on Computers*, C-23:576–581, June 1974.

[13] N. Ishiura. *Studies on Logic Simulation and Hardware Description Languages*. PhD thesis, Kyoto University, Kyoto, Japan, Dec. 1990.

[14] N. Ishiura, Y. Deguchi, and S. Yajima. Coded time-symbolic simulation using shared binary decision diagram. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 130–145, 1990.

[15] N. Ishiura, M. Takahashi, and S. Yajima. Time symbolic simulation for accurate timing verification of asynchronous behavior of logic circuits. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 497–502, 1989.

[16] D. S. Kung. Hazard-non-increasing gate-level optimization algorithms. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design*, pages 631–634. IEEE Computer Society Press, November 1992.

[17] W. Lam and R. Brayton. *Timed Boolean Functions: A Unified Formalism for Exact Timing Analysis*. Kluwer Academic Publishers, 1994.

[18] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Synthesis of hazard-free asynchronous circuits with bounded wire delays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):61–86, Jan. 1995.

[19] M. H. Linderman. *Simulation of Digital Circuits in the Presence of Uncertainty*. PhD thesis, Cornell University, 1994.

[20] A. Martello, S. Levitan, and D. Chiarulli. Timing verification using hdtv. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 118–173, 1990.

[21] T. M. McWilliams. Verification of timing constraints on large digital systems. In *Proceedings of the 17th ACM/IEEE Design Automation Conference*, pages 139–147, 1980.

[22] E. Ulrich, K. Lentz, S. Demba, and R. Razdan. Concurrent min-max simulation. In *EDAC. Proceedings of the European Conference on Design Automation*, pages 554–557, 1991.

[23] K. Y. Yun and D. L. Dill. Automatic synthesis of 3D asynchronous finite-state machines. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design*, pages 576–580. IEEE Computer Society Press, November 1992.