An Accurate Model for Ambiguity Delay Simulation

Thomas H. Krodel and Kurt J. Antreich
Institute of Computer Aided Design, Department of Electrical Engineering,
Technical University of Munich, D-8000 Munich 2, West Germany

Abstract
This paper presents a new approach for the accurate computation of ambiguously delayed waveforms which is implemented in the event-driven logic simulation system LDSIM. Six logic values are defined to be sets of up to four basic values. The mapping function representing the ambiguity delay model only manipulates these basic values. This principle results in a powerful and versatile ambiguity delay model that is clearly presented, easy to implement and guarantees correct delay computation of arbitrary six-valued waveforms with respect to the concept of superposition and causality.

1 Introduction
Since the design and the synthesis process of complex logic circuits is prone to errors (especially timing errors), a verification phase of the design is indispensable before manufacturing integrated circuits.

The main tool for design verification is a simulator that can exactly predict the behavior of the designed circuit. In order to model the timing of the circuit close to reality, there has to be a structure-oriented circuit description with powerful delay models. On the other hand, due to the ever increasing complexity of integrated circuits, the speed of a simulation run is important for the feasibility of the verification procedure. The verification tool most frequently used is a gate-level simulator because modeling and simulating at this level constitutes a good compromise between the accuracy of predictions about the timing behavior and the achievable simulation speed.

The accuracy of a simulator is determined by its simulation models. The fact that timing errors are hard to avoid in advance, especially when designing fast and complex asynchronous logic circuits, but difficult to detect before manufacturing other than by accurate simulation implies that special care has to be taken in choosing and implementing the delay model.

This paper presents a new structured, compact and versatile approach for ambiguity delay simulation, a simulation mode of increasing importance for the design verification of asynchronous high-speed logic circuits. The concepts described herein serve as a basis for several variations of delay models which are implemented in the logic design simulator LDSIM. This simulator incorporates a well-defined six-valued logic that comprises the logic states High, Low, Unknown, Rising, Falling and Changing. A logic value consists of up to four basic values. The mapping function for ambiguity delays only manipulates these basic values. Due to this principle, the mapping function is quite simple and always computes correct waveforms for arbitrary input waveforms with respect to the concepts of superposition and causality (the problem of common ambiguity will not be addressed here).

The paper is organized as follows: It starts with an introduction to the ambiguity delay model and continues with the definition of the six logic values and the definition of permitted value sequences. The mapping function is then presented in detail. We demonstrate the efficiency of the proposed approach by two examples. Finally, results of a commercial simulation run with LDSIM are included.

2 The Ambiguity Delay Model
A delay model of high precision and flexibility is the ambiguity delay model that takes into account minimum and maximum values for the delay. Advanced versions of this model distinguish between rising and falling transitions which is indispensable for the simulation of MOS circuits. Due to these features, the ambiguity delay model is capable of considering statistical variations of (transition-dependent) delay times. Furthermore, it enables the designer to affect the robustness of his design against timing problems by varying the min-max ratio. Of course, this ratio can be chosen to be 1, and the ambiguity delay model becomes a standard rise/fall-delay model.

Previously published implementations of the am-

2024/90/0000/0563$01.00 ©1990 IEEE

563
biguity delay model mostly use a table-coded technique, combined with the usual event scheduling algorithm (event-driven simulation) [CY71,Szy72, TSBP74]. Unfortunately, it cannot be deduced from these descriptions how (or that) the delay model reacts fault-free to arbitrary input waveforms.

Other papers [Bow82,BS87] treat a whole transition (from the beginning to the end) as one object (or event). The problem arising from this approach is that the statement made by such an object (concerning a whole period of time) may become invalid if another event affects the signal just being in a transition state (incomplete transition). One solution is to carry out an extra analysis of those 'hazard conditions', or to backtrack along the time-axis in order to correct the predictions about the shape of the transition—both cpu-time consuming actions.

The approach described in [CPG76] is based on two mathematically defined mapping functions which seem to yield correct results for all possible input sequences, although the definition of the second mapping function looks rather complicated. Furthermore, this approach is limited to three-valued logic. Similar to our approach, a concept of 'reference times' is used for the unscheduling of events.

[MIK+85] describes a timing analysis system featuring ambiguity delays. Although this paper does not contain an algorithm for general simulation, an example shows the notion of 'transition'—states being made up of separate (and therefore separable) rising and falling transitions. This principle is inherent in our approach.

The logic simulator DISIM, developed at AEG-Telefunken, Germany, also incorporates an ambiguity delay model. The system is described in [Jen74], but the algorithm of the delay model is left unknown to the reader.

Accurate logic simulation is not a matter of course. Problems even start with the specification of different rise-/fall-delays (without any ambiguity). For example, experiments (Fig. 6) have shown that some commercial simulators compute 'Unknown' to begin at time step 10 (which should rather be time step 8). The computation of an ambiguously delayed waveform normally gets complicated and/or yields inaccurate results as soon as consecutive transitions happen so fast that they influence each other (which, of course, cannot be precluded in general). In order to judge an algorithm for correctness we employ the concepts of superposition and causality.

Since logic values other than High and Low denote uncertainties, six-valued waveforms can be dissected into several possibly occurring waveforms. Furthermore, the ambiguity delay model leaves several possibilities for the delay of a transition, namely all those where the transition happens within the defined range. If the superposition of all possible (dissected and delayed) waveforms yields exactly the multi-valued waveform computed by the algorithm, the concept of superposition is valid.

The concept of causality states that the algorithm has to observe the relation between cause and effect. For example the (ending) falling transition of a positive pulse can never take place before the (starting) rising transition. This seems to be obvious, but has to be taken into account when the rise-delay is so much greater than the fall-delay that the falling transition is delayed to a time earlier than the beginning of the (delayed) rising transition.

In contrast to other implementations of the ambiguity delay model, our approach inherently observes these two concepts, thus always computing correct delayed waveforms on principle.

3 Logic Value Definition

The definition of the six employed logic values is based upon the binary values 0 and 1 [Hay86]. Since transitions have to be represented explicitly, we need distinct values for them. Therefore, we define four 2-tuples, called basic values, that denote two steady and two transition states:

\[
\begin{align*}
  h &= (1,1) \quad \text{high} \\
  l &= (0,0) \quad \text{low} \\
  r &= (0,1) \quad \text{rise} \\
  f &= (1,0) \quad \text{fall}
\end{align*}
\]

In order to express uncertainty about a signal state, a logic value is defined to be a set of basic values. Anyone of the states represented by those basic values that are elements of the set is possible to occur at the respective time step. The total number of 16 different sets that can be formed with 4 elements is reduced to 6 by the following constraints (where \( Z \) denotes a logic value):

\[
\begin{align*}
  \emptyset &\subseteq Z \subseteq \{ h, l, r, f \} \\
  r \in Z &\implies h \in Z \land l \in Z \\
  f \in Z &\implies h \in Z \land l \in Z
\end{align*}
\]

Eqn. (2) defines the logic values being sets with the basic values \( h, l, r \) and \( f \) as elements and excludes the empty set since signals are always in some state. Eqn. (3) and (4) claim that if there is a transition, the basic values representing the source and the destination of that transition always have to be present, too. This is necessary because we don't want to make any statements about the slope of a transition; the transition can be so fast that also the signal states \( h \) and \( l \) are part of the respective time step.
These three equations define the following six logic values:

\[
\begin{align*}
\mathcal{H} &= \{h\} \quad \text{High} \\
\mathcal{L} &= \{l\} \quad \text{Low} \\
\mathcal{U} &= \{h, l\} \quad \text{Unknown (static)} \\
\mathcal{R} &= \{h, l, r\} \quad \text{Rise} \\
\mathcal{F} &= \{h, l, f\} \quad \text{Fall} \\
\mathcal{C} &= \{h, l, r, f\} \quad \text{Change} \text{ (dynamic Unknown)}
\end{align*}
\] (5)

The logic values constitute a hierarchy of uncertainty. The more basic values a logic value consists of, the more uncertainty it describes. The most uncertain logic value is C, all other values are subsets of C.

It should be mentioned that logic values containing more than one basic value only describe the possibility of the corresponding signal states to occur. Therefore, the value \(\mathcal{R}\) does not imply a rising transition to happen but only excludes a falling one. Only after analyzing a whole sequence like \(CRRH\) it can be determined that one rising transition is certain to occur—anywhere during the time steps denoted by \(\mathcal{R}\).

4 Permitted Value Sequences

A precondition for the mapping function is that there are no implicit transitions in a waveform, i.e. every transition that may take place has to be represented by the corresponding basic value. Therefore, a sequence like \(\mathcal{LH}\) will be inadmissible.

The constraints excluding implicit transitions can easily be formulated with the help of basic values. Any logic value change is characterized by the manipulation of the corresponding set of basic values, i.e. by the addition and/or the removal of basic values. The number of all possible set manipulations is reduced by the following constraints (where \(Z^T\) denotes a logic value at time step \(T\)):

\[
\begin{align*}
h \notin Z^T \land h \in Z^{T+1} &\Rightarrow r \notin Z^T \land r \in Z^{T+1} \\
l \notin Z^T \land l \in Z^{T+1} &\Rightarrow f \notin Z^T \land f \in Z^{T+1} \\
h \in Z^T \land h \notin Z^{T+1} &\Rightarrow f \in Z^T \land f \notin Z^{T+1} \\
l \in Z^T \land l \notin Z^{T+1} &\Rightarrow r \in Z^T \land r \notin Z^{T+1}
\end{align*}
\] (6) (7) (8) (9)

Since basic value \(h\) can only be reached by a rising transition, Eqn. (6) claims that the beginning of basic value \(h\) has to be accompanied by the beginning of basic value \(r\). Similarly, leaving basic value \(h\) needs a falling transition, i.e. basic value \(f\) (Eqn. (8)). For reasons of symmetry analogous constraints are effective for basic value \(l\) (Eqn. (7) and (9)). These four constraints define 20 permissible logic value changes; they are illustrated in Fig. 1.

It is important to note that the restrictions on logic value changes do not constitute any limitations of the simulation algorithm. If the user specifies an illegal sequence of logic values at the primary inputs, the simulator generates the missing transition by itself. Nevertheless, it is possible for arbitrary logic value changes to take place within one time step. This is accomplished by the methods of ternary simulation [Eic65.BS87b] that are incorporated in the basic simulation algorithm (these methods are useful for detecting static and dynamic hazards). There, each time step is divided into two halves whereby two logic value changes become possible to happen during one time step. It can be seen in Fig. 1 that arbitrary logic value changes are always obtainable in no more than two steps.

5 The Delay Function

The circuit model implicitly provides delays at the output of elements. Therefore, elements consist of two kinds of units (Fig. 2): A unit performing the logic operations and, for each element output, a unit that delays waveforms according to the four delay parameters (minimum and maximum delay of the rising and the falling transition, Eqn. (10)).

\[
0 \leq T_{Rmin} \leq T_{Rmax} \quad 0 \leq T_{Fmin} \leq T_{Fmax}
\] (10)

The delay function operates in two steps. The first step constitutes a mapping from the (internal) waveform \(z\) to an auxiliary waveform \(V\). This is the temporal part of the delay function because it provides a delay of waveform \(z\) according to Eqn. (10) and the concept of superposition. The second step, the mapping from waveform \(V\) to the output waveform \(Z\), modifies waveform \(V\) in consideration of the concept of causality.
5.1 The Mapping Function, Part I

Bearing in mind that a logic value is made up of basic values, we can characterize a waveform of logic values by regions of presence of the four basic values. Since the mapping function treats each basic value separately, there is an implicit dissection of the waveform into regions of basic values whereby the concept of superposition is observed.

The operation of the mapping function (part I, from 2 to 5) is to delay the beginning and the end of each region of basic values according to the following delay times:

<table>
<thead>
<tr>
<th>Region</th>
<th>Delay of the beginning (TB)</th>
<th>Delay of the end (TE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>TRmin</td>
<td>TFmax</td>
</tr>
<tr>
<td>l</td>
<td>TRmin</td>
<td>TFmax</td>
</tr>
<tr>
<td>r</td>
<td>TRmin</td>
<td>TRmax</td>
</tr>
<tr>
<td>f</td>
<td>TFmin</td>
<td>TFmax</td>
</tr>
</tbody>
</table>

Considering the definition of the four delay parameters (Eqn. (10)), the delay times for regions r and f are obvious. For region h, the delay of the beginning, TB = TRmin, can be deduced from Eqn. (6) (region h starts with a rising transition), and the delay of the end, TE = TFmax, from Eqn. (8) (region h ends with a falling transition). Similar considerations result in the delay times for basic value region l.

The event-driven implementation of the first step of the mapping function is illustrated by two examples with basic value region h (TB < TE, Fig. 3 and TB > TE, Fig. 4). The events (arrows) transport the instruction to start or to end the basic value region. If events overlap, i.e. the sequence of generation is not the same as the sequence of execution (Fig. 3, T: 10–13; Fig. 4, T: 11–15), the out-of-date event has to be ignored (otherwise, the basic value region would end at time step 15, Fig. 4). This is done by the concept of 'reference times': Upon successful execution of an event, the variable Tref is set to the generation time of that event. But if there is an event to be executed that was generated at an earlier time than the time currently stored in Tref, it is out of date and will be discarded (Fig. 3, T: 13; Fig. 4, T: 15). This principle of unscheduling avoids cpu-time consuming manipulations of the event queue.

5.2 The Mapping Function, Part II

The auxiliary waveform V, generated by the first step of the mapping function, does not always contain valid logic values; it may happen that Eqn. (3) and/or (4) are violated. This means that there may exist basic values r and f where there are not both of the basic values h and l present, indicating a violation of the concept of causality (Fig. 6, T: 5). Therefore, the second step of the mapping function removes basic values r and/or f according to Eqn. (12), yielding the final output waveform Z.

6 Examples and Results

The efficiency and correctness of the proposed approach is demonstrated by two examples. The graphical parts show the existing basic values at each time step (h, l, r, f).

Fig. 5 shows a short pulse to be delayed. After the delay, the ambiguity regions of the rising and the falling transition overlap. Therefore, the value H (h, T: 3) does not occur any more at Z, the value C is generated instead. In this example, the second part of the mapping function has no effect on Z (Z = V).
Figure 5: Example 1

Figure 6: Example 2

Even if the delay is not ambiguous (Fig. 6) our approach leads to an improved accuracy (compared to some commercially available simulators, for example). Value \( U \) seems to be delayed by 4 units (T: 5+9) although no delay time amounts to 4 units. But if we dissect waveform \( z \) into the two possible waveforms, we will see that the first waveform, a pulse, will be swallowed since the rise-delay is longer than the fall-delay plus pulse width (mapping function, part I, removes fat time step \( 5 \)). The second waveform consists of one logic value change from \( L \) to \( H \) at time step 2; this rising transition is delayed to time step 8. Superposing the two delayed waveforms yields exactly waveform \( Z \).

Finally, Tab. 1 shows some statistics obtained by simulating a commercial asynchronous circuit that is employed in the area of telecommunication (size: approx. 730 gate equivalents). The simulation was done on a VAXstation 2000 running ULTRIX.

### 7 Conclusions

A new approach for ambiguity delay simulation has been presented. The algorithm computes always accurate delayed waveforms for arbitrary input waveforms and observes the concept of superposition and causality. This is due to the fact that logic values are defined to be sets of basic values and the mapping function directly manipulates these basic values in a straightforward manner. This approach has successfully been implemented in the logic simulation system LDSIM.

| Input Vectors | 978 |
| Simulated Time Range [Time Units] | 24 450 000 |
| Logic Value Changes | 47 465 |
| Element Calculations | 73 363 |
| Generated Events | 63 319 |
| CPU-Time [Sec] (VAXstation 2000) | 36.48 |
| Events per Second | 1 735 |

Table 1: Statistics of a Simulation Run

Curate delayed waveforms for arbitrary input waveforms and observes the concept of superposition and causality. This is due to the fact that logic values are defined to be sets of basic values and the mapping function directly manipulates these basic values in a straightforward manner. This approach has successfully been implemented in the logic simulation system LDSIM.

### References


