AFTA: A Formal Delay Model for Functional Timing Analysis

V. Chandramouli, Jesse P. Whittemore, and Karem A. Sakallah
EECS Department,
The University of Michigan,
Ann Arbor, MI 48109-2122, USA

Abstract

Despite its importance, we find that a rigorous theoretical foundation for performing timing analysis has been lacking so far. As a result, we have initiated a research project that aims to provide such a foundation for functional timing analysis. As part of this work we have developed an abstract automaton based delay model that accounts for the various analog factors affecting delay, such as signals slopes, near simultaneous switching, etc., while at the same time accounting for circuit functionality. This paper presents this delay model.

1. Introduction

With the advent of deep submicron technologies, digital abstraction, which has served as an efficient means for doing synthesis and analysis, is widely perceived to be falling apart. Consequently, the notion of delay, which is at the core of timing analysis, is being questioned [15]. We believe that this is the result of the lack of a rigorous foundation for timing. Traditionally, both delay modeling and timing analysis algorithms have been developed by different schools of thought. Delay modeling has usually been dealt with by electrical engineers where the emphasis has been on accurate delay value *calculation* [9, 13, 16] rather than a *formalization* of the notion of delay. On the other hand, on the timing analysis front, computer scientists have used simplistic delay models like transport delay (with suitable "add-ons" like rise-fall, inertial, etc. for improved accuracy) with graph-based algorithms. Most initial approaches ignored circuit functionality and important "analog" effects like signal slopes and near-simultaneous switching of inputs (proximity effect) altogether leading to incorrect estimates of delay. In an attempt to account for circuit function, many different local sensitization conditions ([2, 6, 10, 12) have been proposed recently, none of which have been met with universal acceptance. Moreover, these approaches still do not account for effects like signal slopes. Thus, the field of timing modeling and analysis is marked by confusion.

To alleviate this problem, we are developing a rigorous theoretical framework at the University of Michigan for performing functional timing analysis that retains the effect of analog factors like signal slopes for accuracy. This theory has two components: 1) Waveform calculus [14], which links circuit function with timing, and 2) AFTA, which is the formal model of delay. This paper focuses on the latter, with the former being discussed elsewhere. This paper is organized in 5 sections as follows. In the next section, we very briefly review the relevant concepts of the waveform calculus. In Section 3, we present the semantics of AFTA and show how it differs from other models that bear a superficial similarity to AFTA. In Section 4, we describe the operation of AFTA with a couple of examples. Finally, we conclude the paper in Section 5.

2. The Waveform Calculus: A Resumè

The waveform calculus is a symbolic model of digital circuits that was proposed in [14] to link circuit timing and function. Symbolic waveforms are used in the calculus to model logic waveforms. Therefore, the waveform values are Boolean functions defined over an appropriate basis variable set rather than the Boolean constants 0 or 1. As an example, we show the symbolic simulation of the XOR circuit under the state-dependent delay model [17] shown in Fig. 1(a). BDDs [4] were used to manipulate the waveforms at the internal nodes. The waveforms at the primary inputs are typically represented by Boolean variables (in this case, x_1, x_2, and, x_3) which are called the basis variables. Symbolic waveforms thus represent a family of waveforms. For example, in Fig. 1(a), for different assignments of 0s and 1s to the basis variables, a variety of logical waveforms at z can be generated.

In most realistic circuits, the number of basis variables will be large. Therefore, in order to deal with complexity, we introduce the notion of *functional* abstraction, where one or more of the basis variables can be abstracted away. In Fig. 1(b), we show the same simulation run with variable x_1 abstracted away. The resulting waveforms are now called partially-specified waveforms and the waveform values are now Boolean *function intervals* [3]. An additional mechanism in our calculus to cope with complexity is *temporal* abstraction, where transition instants that are not of interest are abstracted away. This is of particular importance for timing analysis where we are often concerned with the first and last event times. For example, in Fig. 1(c), the instant $t = 7$ has been abstracted away from the waveform z giving rise to the partially-specified waveform z'. Partially-specified waveforms are also convenient to represent signal slopes. For example, referring to Fig. 1(c), for the assignment $x_1 = 0, x_2 = 1,$ and, $x_3 = 1$, one inter-

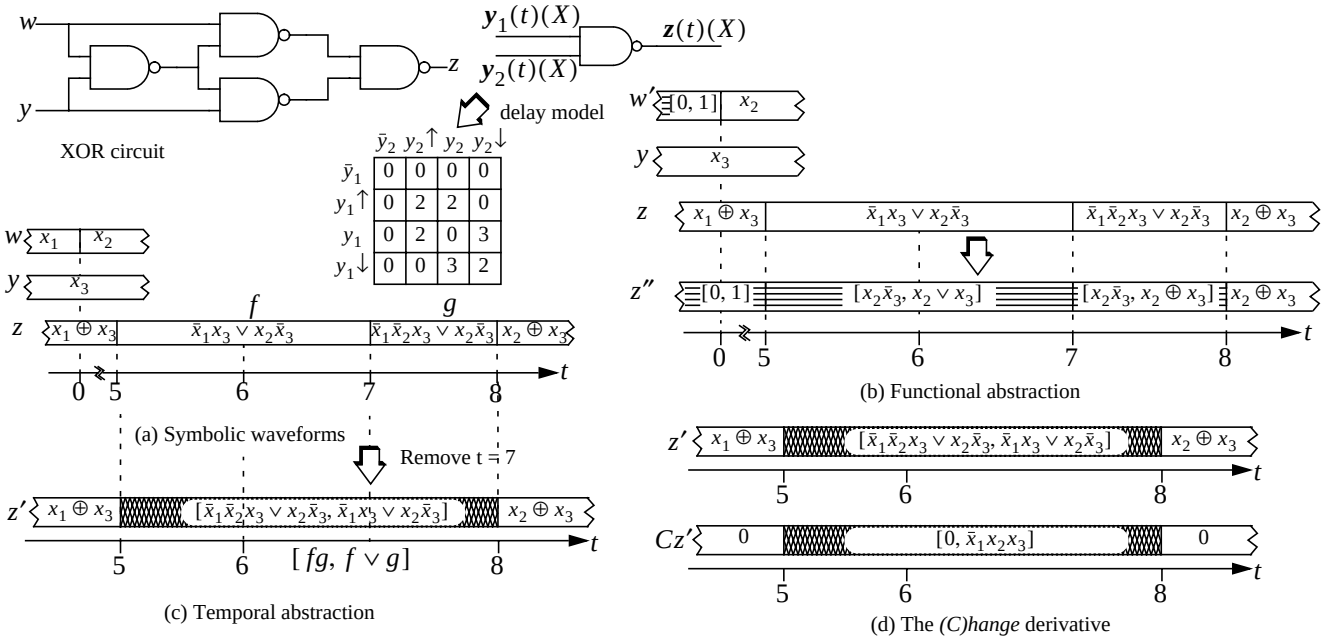


Fig. 1. Waveform calculus concepts

pretation can be that the signal is falling from $t = 5$ to $t = 8$.

The notion of waveform *derivatives* is central to the calculus since it captures the conditions under which a signal will change. The *(C)hange* derivative of a waveform $w(t)(X)$ is given by:

$$Cw(t)(X) = \lim_{\varepsilon \to 0} [\boldsymbol{w}(t-\varepsilon)(X) \oplus \boldsymbol{w}(t+\varepsilon)(X)] \qquad (3)$$

Note that this itself is a symbolic waveform as shown in Fig. 1(d). A non-zero value of the *C* derivative indicates the signal could potentially change. The *(R)ising* derivative is given by:

$$Rw(t)(X) = Cw(t)(X) \lim_{\varepsilon \to 0} \overline{w}(t-\varepsilon)(X) \qquad (4)$$

Similarly, the other three derivatives, *(F)alling, (H)igh, and (L)ow*, can be defined. For lack of space and ease of presentation, in the rest of the paper we will consider non-symbolic waveforms only and therefore drop the explicit dependence of the waveforms on *X*.

The relationship between the derivatives and delay is explained in Fig. 2. We associate a zero delay output, $z^*(t)$ with each gate[1]. It is simply the instantaneous Boolean function of the input waveforms. From the waveform calculus machinery, we derive the conditions under which $z^*(t)$ will rise or fall (Fig. 2). Since delay is state dependent, we associate different delays with each of the six conditions. It is now the task of the delay model, AFTA, to produce the true output $z(t)$ from $z^*(t)$ using a delay value appropriate for the current input conditions. Referring to Fig. 2, if both inputs were falling, AFTA would make $z(t)$ rising *after* a delay of 2 units as shown.

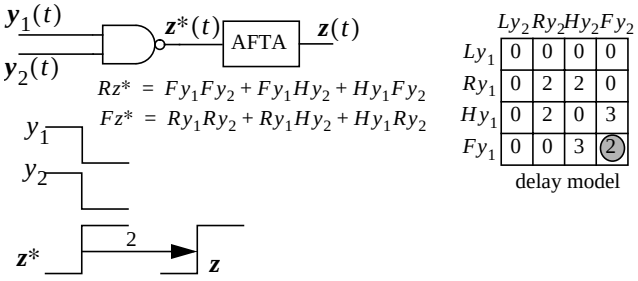
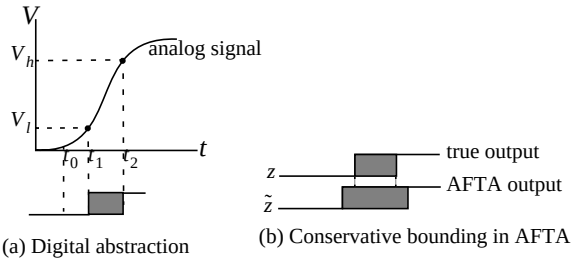


Fig. 2. From derivatives to delay

The notion of delay value computation is an orthogonal issue. We use the generic term “delay macromodel” to denote the entity that computes the delay value. The gate could be precharacterized and the delay macromodel implemented as a lookup-table; however, if such a precharacterization is not possible, the delay macromodel could be a simplified timing simulation for the specific input configuration. We assume that these delay values are available on demand through a suitable function call. The semantics of AFTA are now discussed in the following section.

1. We use the word “gate” to denote both the usual AND-OR-INV gates as well as channel connected components (CCC).



(a) Digital abstraction (b) Conservative bounding in AFTA

Fig. 3. Digital abstraction and bounding in AFTA

3. AFTA Semantics

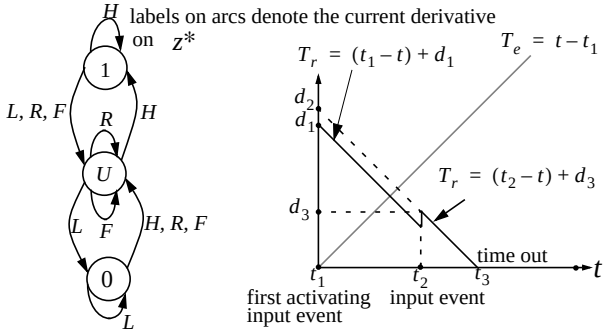
An analog signal can be abstracted into the digital domain as shown in Fig. 3(a). It has been shown that, for timing purposes, the unity differential gain points of a gate's DC voltage transfer curve, V_{il} and V_{ih}, are the appropriate candidates for the logic thresholds, V_l and V_h [5]. Thus, the output of a logic gate can be any one of $\{0, 1, U\}$, where U is interpreted to mean any voltage value lying between the 0 and 1 values. When sufficient input information is not available to compute the output, the guiding principle in AFTA is that of *conservative bounding* (see Fig. 3(b)): AFTA output should begin changing no later than the true output and should end changing no earlier than the true output.

AFTA has three *discrete* states corresponding to the three logic values of a signal $\{0, U, 1\}$. In addition to these, AFTA has two continuous real-valued timers, T_r and T_e. T_r denotes the time remaining before the output changes to its next scheduled value in response to the input change. T_e denotes the time elapsed since the first input event that "activated" (to be made clear later) AFTA. Whenever the inputs to the gate change, a new value for T_r is computed using the delay macromodels. The total state, or simply the state, in AFTA consists of both the discrete and continuous states: (z, T_e, T_r). State transitions in AFTA occur in response to:

1. input events, or

2. time-outs indicated by $T_r = 0$.

A state transition involves a change in the output logic value or a change in the values of T_e and T_r. The changes in the output logic value obey the transition diagram in Fig. 4(a), and occur only when T_r expires (i.e. $T_r = 0$). The labels on the arcs are the derivatives of z^*. As will become evident later on, this transition diagram models the *eventual* logic value that the gate output attains, and is able to account for inertia in the gate response. Input events, on the other hand, affect the elapsed and remaining time components of AFTA state. As is evident from the self-loops in



(a) Discrete state transition function (b) Operation of the timers

```
LOOP: while(Tr >0)
{
 if (an input event at t = t* )
 {
  if (first activating event){ t*1 = t* ;}
  else{ Te = t* – t1* ;}
 Tr* = M(current output, inputs, Te );
 } /* end of if */
 else{
  if (AFTA is stable){ Te = 0 ; Tr = ∞ }
  else { Te = t – t1* ; Tr = (t* – t) + Tr* }
 ;
 } /* end of else */
} /* end while */
Tr* = M(current output, inputs, Te );
if (Tr* ==0)
{
   change output logic value according to
diagram in Fig.4(a);
  Tr* = M(current output, inputs, Te );
   Te = t – t1* ;
} /* end of if */
goto LOOP;
```

(c) The pseudo-code illustrating the operation of AFTA

Fig. 4. The semantics of AFTA

states 0 and 1, when the z^* derivative is L with all inputs to the gate stable and not changing and AFTA is in state 0 (or z^* derivative is H with all inputs stable and AFTA is in state 1), AFTA is said to be in a *stable* configuration. All other situations are considered unstable and AFTA is said to be *activated.* The self-loop for state U when the z^* derivative is R or F (i.e. changing) is in keeping with conservative bounding, since if z^* is changing the output should also be changing and should remain changing till at least z^* becomes stable.

Initially, before any input event occurs, AFTA is assumed to be in a stable configuration with $T_e = 0$. When an input event occurs at time t^*, T_e becomes a free running variable and starts to increase linearly with time if it happens to be the *first* activating transition. An activating

transition is the one that takes AFTA from a stable to an unstable configuration. It continues to increase, modeling elapsed time, till AFTA reaches a stable configuration once again. Thus, T_e defines a window on the inputs, and all input events within this window are used by the delay macromodels to compute delay. Note that by resetting T_e to 0 when AFTA is in a stable configuration, we are ignoring the effect of all previous input events while computing delay. When an input event occurs, T_r is set to T_r^*, a value determined by an appropriate delay macromodel. Subsequently, T_r becomes a free running variable decreasing with time. A logic change in z occurs when T_r becomes 0 and is determined by the current z^* derivative and the state transition diagram in Fig. 4(a). The operation of the two timers is further illustrated in Fig. 4(b). An input event at time t_1, assumed to be the first activating one, sets T_r to d_1, which is computed by calling an appropriate delay macromodel, and enables T_e. From then on these two timers become free running, with their trajectories as shown. At t_2 another input event occurs, causing a re-evaluation of the delay to d_2 (proximity effect, for instance). However, we must subtract the amount of time that AFTA has been in the current state which results in the value d_3 being loaded into T_r. Note that this event does not affect T_e since it is not the first activating one. Subsequently, T_r becomes free running once again and when T_r finally expires at t_3 (assuming no further input event has occurred), the output will change depending on the current z^* derivative according to Fig. 4(a).

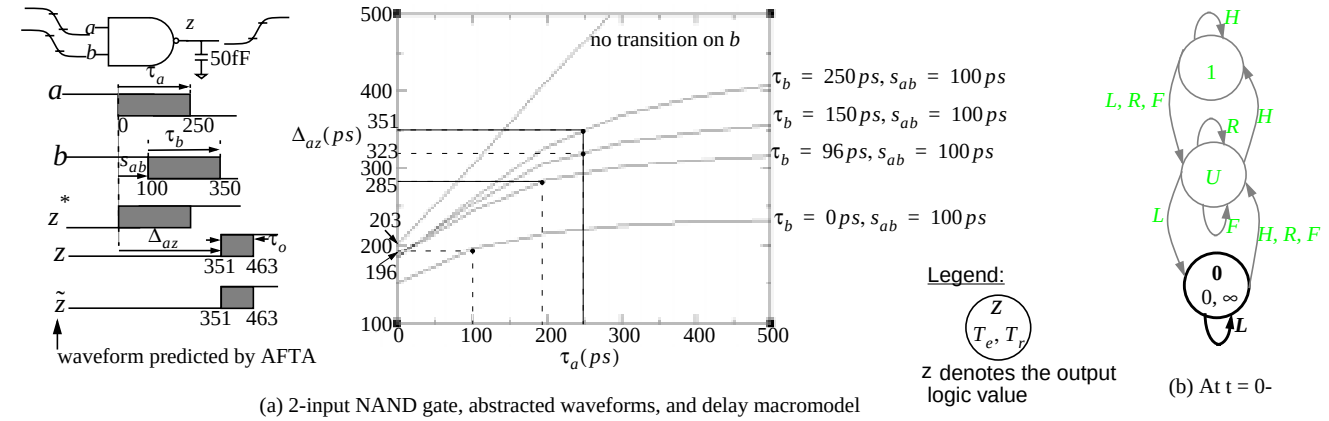
We can summarize the operation of AFTA in the pseudo-code of Fig. 4(c). A couple of observations can be made from the pseudo-code. First, the function that computes the value for T_r, M, depends on the current output logic value and the various delay macromodels which in turn depend on the inputs within the window specified by T_e. Second, when T_r expires, a new value of T_r is computed using the most recent knowledge of the inputs. It is only if this second computation also yields 0 that the output logic value is changed according to Fig. 4(a). This is because since the time when T_r was originally set, additional information about the inputs may have become available and this must be accounted for, else AFTA may change the output at an incorrect instant. This point will be made clear in the examples of the next section.

Automaton based delay models have been proposed before in the literature, although for different purposes. In [8], a logic automaton was proposed using the discrete-event system specification (DEVS) formalism introduced by Ziegler [18]. However, the emphasis was on event-driven simulation rather than functional timing analysis. Moreover, a simplified transistor level model based on the charging and discharging of load capacitance through a resistor was used to compute actual delay values. Such a simplified model is clearly inadequate for deep submicron designs. More recently, in [7], the functional and timing models of a gate were linked using timed-transition tables to enable efficient VHDL simulation. However, the low-level modeling suffered from inaccuracies since the transistors were replaced by equivalent *RC* networks. Sufficient examples showing the efficacy of their model in accounting for state dependent effects were not presented. In addition, both these approaches lacked a theoretical framework that clearly linked circuit function with timing, and did not show how to compute the early and late signal arrival times, the central task in timing analysis. Another work that uses an automaton for timing, albeit not for computing gate outputs, is presented in [1]. A timed-automaton formalism to model real-time systems is introduced and the focus is on analyzing asynchronous sequential circuits. While AFTA may bear a superficial similarity to the timed-automaton or the logic automaton, what sets it apart is its specific role to facilitate accurate functional timing analysis.

We now show how AFTA can handle the signal slopes and the proximity effect by presenting different input situations that could occur in practice. These examples have a timing simulation flavor since we will be using fully specified waveforms (i.e. there is no uncertainty as to when the signals will transition). While AFTA will be eventually used with the more general partially specified waveforms which arise in timing analysis, these examples nevertheless show both the power of the AFTA model in accounting for analog effects as well as the inherent limitations of doing timing simulation at the gate level using a discrete system.

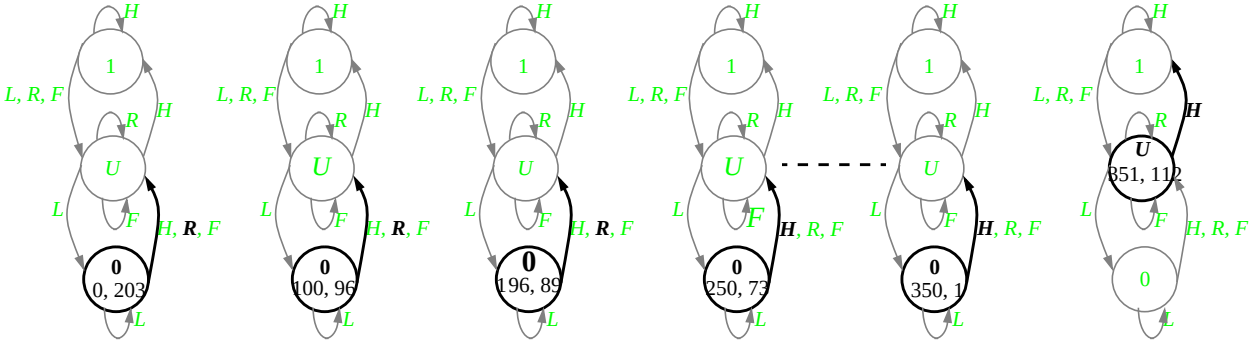
4. Explaining AFTA Through Examples

The examples in this section have been constructed by simulating actual circuits with HSPICE [11] using a $0.5\mu m$ CMOS technology and performing a suitable digital abstraction of the analog waveforms used in the simulation. In general, the errors in doing timing analysis with AFTA would be from two sources: 1) Delay macromodel errors and 2) AFTA errors. In the following examples, we used HSPICE as the delay macromodel for delay while doing the simulation to remove the effect of macromodeling errors. This enabled us to concentrate on errors (if any) induced by AFTA alone. In the examples, since an AFTA state involves a discrete component and two continuous real-valued components, we explain AFTA's operation by appropriate snapshots in time that capture the evolution of the two continuous components of state. The active state is shown in **bold** font in all the figures.



(a) 2-input NAND gate, abstracted waveforms, and delay macromodel

(b) At t = 0-



(c) At t = 0 (d) At t = 100ps (e) At t = 196ps (f) At t = 250ps (g) Finally, at t = 350ps (h) At t = 351ps

Fig. 5. AFTA operation for slope and proximity

Example 1: Slope and proximity in AFTA Consider two falling inputs to the NAND gate as shown in Fig. 5(a). The delay is now a function of both the transition times and the temporal separation between the two inputs. The delay macromodel, $\Delta_{az}(\tau_a, \tau_b, s_{ab})$, is also plotted in the figure, as a function of τ_a, parametrized by τ_b, for a fixed separation of $s_{ab} = 100ps$. The initial state of AFTA is shown in Fig. 5(b). At $t = 0$, a starts to fall (causing z^* derivative to be R) which activates T_e and sets T_r to $\Delta(0, 0, \infty) = 203ps$ (delay when no transition on b) as shown in Fig. 5(c). At $t = 100ps$, b starts to fall (note that the z^* derivative is still R) causing T_r to be set to $\Delta_{az}(100, 0, 100) - T_e = 196 - 100 = 96ps$ (see Fig. 5(d)). Note how the value of T_e defines the proximity window. At $t = 196ps$, T_r expires and a recomputation with the most recent input information sets T_r to $\Delta_{az}(196, 96, 100) - 196 = 285 - 196 = 89ps$ as shown in Fig. 5(e). Before T_r expires, however, at $t = 250ps$, a becomes low (z^* derivative is now H) and this sets T_r to $\Delta_{az}(250, 150, 100) - 250 = 323 - 250 = 73ps$ as shown in Fig. 5(f). Skipping a few intermediate steps, eventually, at $t = 350ps$ b becomes low (z^* derivative remains H) and now we have the complete information to set T_r to $\Delta_{az}(250, 250, 100) - 350 = 351 - 350 = 1ps$ (see Fig. 5(g)). 1ps later, at $t = 351ps$, T_r expires and subsequent recomputation still yields a value of 0 since the inputs haven't changed. Therefore, AFTA makes the output logic U since the z^* derivative at this instant is H, as shown in Fig. 5(h). T_r is set to $\tau_o(250, 250, 100) = 112ps$ using the output transition time macromodel (not shown here). Finally, at $t = 463ps$, T_r expires and subsequent recomputation still yields a value of 0. Therefore, AFTA makes the output logic 1 since z^* derivative is H. T_r is set to ∞ since the output will remain 1 unless the inputs change and T_e is reset to 0. The actual output and the predicted output in this case are the same and thus we see that AFTA can successfully handle, in addition to the input slope effects, the proximity effect too.

Example 2: Anomalous response of AFTA Suppose input a rises and input b falls as shown in Fig. 6(a). Both the true and AFTA outputs are also shown. As can be seen, AFTA underestimates the end of the first transition and overestimates the onset of the second transition on the output. When input a has finished rising, though input b has not yet crossed the V_h threshold, it starts affecting the output, slowing it down. AFTA, however cannot detect this since it assumes a signal to be changing only if it crosses

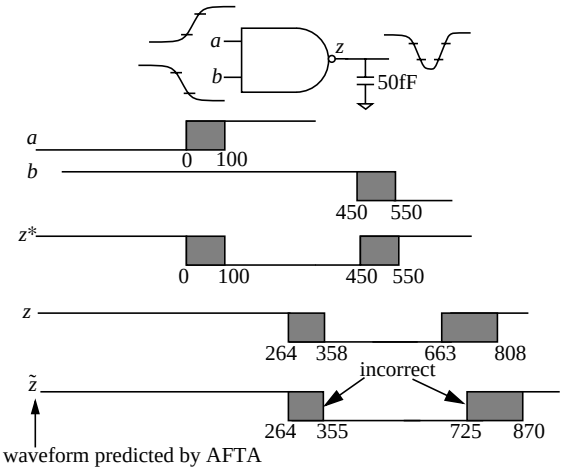


Fig. 6. Anomalous output of AFTA

the appropriate threshold. This causes AFTA to underestimate the end of the first transition. In computing the response to b, AFTA ignores the effect of a since it assumes transition on a occurred sufficiently in the past to affect the output. This is so because AFTA enters a stable configuration (which resets T_e) before the transition on b activates AFTA again. It thus assumes the output to be at 0V while computing the delay for the output to rise which is not the case. The output has not yet quite reached 0V before the transition on b makes it rise again. Consequently, the true output finishes its transition sooner than predicted by AFTA. For a correct computation, effects of both a and b together should be considered. This example shows some of the difficulties in performing timing simulation with an AFTA like approach.

It is important to note that AFTA does over estimate the final stable time which is fine since it is still conservative. It is the intermediate points that are not conservative. However, in timing analysis, these intermediate time instants would be abstracted away in which case AFTA response would conservatively bound the true response.

5. Conclusions

We have presented a new automaton based delay model, called AFTA, that is intended for use in a functional timing analyzer. Unlike previous approaches in developing an abstract delay model, AFTA accounts for analog effects like signal slope and proximity, at the gate level. Lack of space prevents us from describing our recent extensions of AFTA to work with symbolic waveforms using both *temporal* and *functional* abstraction [14] which forms the basis for functional timing analysis.

Acknowledgments

This research was supported in part by NSF under grant MIP-9404632 and ARPA under grant DAAH04-94-G-0327.

References

[1] R. Alur and D. Dill, “Automata for Modeling Real-Time Systems,” *Automata, Languages, and Programming*, M.S. Paterson (Ed.), in series *Lecture Notes in Computer Science* (ed. G. Goos and J. Hartmanis), pp. 322-335, Springer-Verlag, 1990.

[2] J. Benkoski and e. al, “Timing Verification Using Statically Sensitizable Paths,” *IEEE Trans. on CAD of ICs and Systems*, 9(10):1073-1084, 1990.

[3] F. M. Brown, *Boolean Reasoning*, ed. J. Allen, Kluwer Academic Publishers, 1990.

[4] R. E. Bryant, “Graph-Based Algorithms for Boolean Function Manipulation”, *IEEE Trans. on Computers*, 35(8):677-691, 1986.

[5] V. Chandramouli and K. A. Sakallah, “Selection of Voltage Thresholds for Delay Measurement”, *Analog Integrated Circuits and Signal Processing, Kluwer Acedemic Press*, 14(1/2): 9-28, 1997.

[6] H. Chen and D. H.-C. Du, “Path Sensitization in Critical Path Problem,” *IEEE Trans. on CAD of ICs and Systems*, 12(2):196-207, 1993.

[7] J. Fröβl and T. Kropf, “A New Model to Uniformly Represent the Function and Timing of MOS Circuits and its Application to VHDL Simulation” in *Proc. European Design and Test Conference*, pp. 343-348, 1994.

[8] M. Heydemann and D. Dure, “The Logic Automaton Approach to Accurate and Efficient Gate and Functional Level Simulation”, in *Proc. 25th IEEE/ACM Design Automation Conference*, pp. 250-253, 1988.

[9] A. Kayssi, “A Methodology For the Construction of Accurate Timing Macromodels for Digital Circuits”, PhD, University of Michigan, EECS Dept., 1993.

[10] P. C. McGeer and R. K. Brayton, “Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network,” in *Proc. 26th Design Automation Conference*, pp. 561-567, 1989.

[11] Meta-Software, HSPICE H92A User's Manual.

[12] S. Perremans, L. Claesen, and H. DeMan, “Static Timing Analysis of Dynamically Sensitizable Paths,” in *Proc. 26th Design Automation Conference*, pp. 568-573, 1989.

[13] L. T. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic Circuit and System Simulation Methods*, McGraw-Hill, 1995.

[14] K. A. Sakallah, “Dynamic Modeling of Logic Gate Circuits,” Technical Report CSE-TR-253-95, University of Michigan, July 1995.

[15] M. Shoji, *The Dynamics of Digital Excitation*, Kluwer Academic Publishers, 1998.

[16] P. Subramaniam, “Table Models for Timing Simulation”, *Proc. Custom Integrated Circuits Conference*, pp. 310-314, 1984.

[17] S.-Z. Sun, D. H. C. Du, and H.-C. Chen, Efficient Timing Analysis for CMOS Circuits Considering Data Dependent Delays, in *Proc. IEEE International Conference on Computer Design (ICCD)*, pp. 156-159, 1994.

[18] B. P. Ziegler, *Multi-Facetted Modeling and Discrete Event Simulation*, Academic Press, 1984.