WSIM: A Symbolic Waveform Simulator

Piero Franco and Edward J. McCluskey

CRC Technical Report No. 94-4
(CSL TR No. 94-624)

CENTER FOR RELIABLE COMPUTING

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University, Stanford, California 94305-4055

ABSTRACT

A symbolic waveform simulator is proposed in this report. The delay of a faulty element is treated as a variable in the generation of the output waveform. Therefore, many timing simulations with different delay values do not have to be done to analyze the behavior of the circuit-under-test with the timing fault. The motivation for this work was to investigate delay testing by Output Waveform Analysis, where an accurate representation of the actual waveforms is required, although the simulator can be used for other applications as well (such as power analysis). Output Waveform Analysis will be briefly reviewed, followed by a description of both a simplified and a complete implementation of the waveform simulator, and simulation results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

1 INTRODUCTION

The difficulty of testing digital circuits for delay faults led to the proposal of delay testing by *Output Waveform Analysis* [Franco 91]. The aim of Output Waveform Analysis is to reduce the burden on the characteristics of input patterns required by performing a more thorough analysis of the circuit output response. The information in the shape of the actual output waveforms can be used to improve the thoroughness of delay fault testing.

The two classes of Output Waveform Analysis are shown in Fig. 1, together with a two-pattern delay test $<V_1,V_2>$ [Lesser 80] [Smith 85]. The *initializing vector* $<V_1>$ is applied and the transients in the circuit are allowed to settle. The *test vector* $<V_2>$ is then applied. In traditional delay testing, after a timed interval equal to the cycle time, T_C, of the circuit, the outputs are sampled and compared to the expected fault-free response. In *Post-Sampling Waveform Analysis* or *Stability Checking*, the circuit waveforms are checked for any changes after the sampling time, whereas information is extracted from the waveforms before the sampling time in *Pre-Sampling Waveform Analysis*. Different extraction functions are possible for Pre-sampling Waveform Analysis; the function used is computing the *average value* or integral of the waveform.

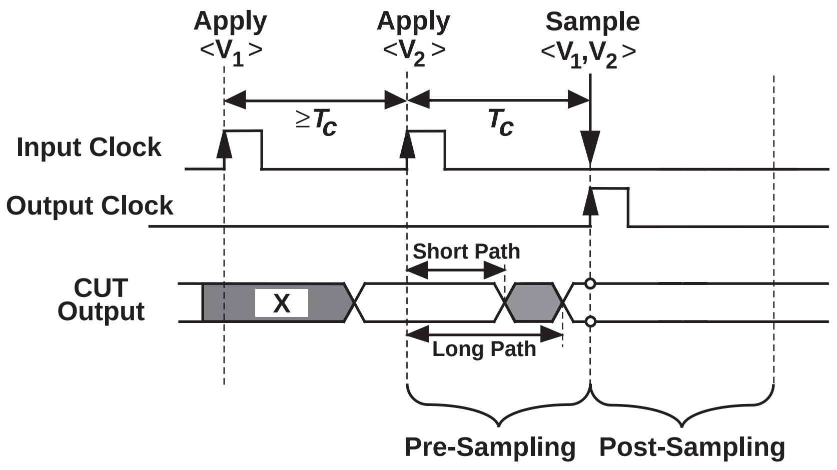

Figure 1. Output Waveform Analysis

Simulators are necessary to evaluate the effectiveness of Output Waveform Analysis, and existing simulators were found to be cumbersome or not suitable, since the actual waveforms at the circuit outputs need to be analyzed for many values of the variable delay. This report describes WSIM, a new simulator which is targeted to manipulating waveforms. WSIM allows one variable delay, and symbolically computes the circuit waveforms as a function of the variable delay. This eliminates the need to run a timing simulator many times with different delays to see the effect of different delay fault sizes. A simpler version of WSIM using no variable delays was also implemented, and is similar to a gate-level timing simulator.

Although WSIM was intended primarily as a tool for evaluating Output Waveform Analysis, it has other uses as well. An example is power estimation in digital circuits, where not only the stable waveforms, but also the transients should be considered to accurately estimate the switching current.

This report is organized as follows. Existing simulators are reviewed in Sec. 2, and their limitations for this application are noted. The simple version of WSIM with fixed delays in discussed in Sec. 3 to introduce the waveform representation used, followed by a description of the complete simulator with one variable delay in Sec. 4. Simple techniques to increase the speed of the simulators are mentioned in Sec. 5. Examples using WSIM are included in Sec. 6, and Sec. 7 concludes the report.

2 EXISTING SIMULATORS

Existing simulators are reviewed in this section and shown to have limitations for investigating Output Waveform Analysis. Timing simulators, delay fault simulators and symbolic simulators are discussed.

2.1 Timing Simulators

Timing simulators [Hitchcock 82], such as Verilog-XL can be used to generate the actual waveform at the outputs of the circuit under test. Verilog was used to analyze Output Waveform Analysis at the start of my work, but there are two main problems with this approach. First, the simulation needs to be rerun for many different delay values at the fault site. Second, there is limited flexibility to try non-standard circuit modeling. For example, it is difficult to consider the effect of capacitive coupling and parasitics on gate delay when multiple input changes occur [Franco 94].

2.2 Delay Fault Simulators

Delay fault simulators depend on the delay fault model adopted. Transition fault simulators use a slightly modified stuck-at fault simulator [Waicukauski 87], and only model “gross” or very large delay faults. Gate delay fault simulators [Pramanick 89] [Iyengar 90], generally compute the size of the smallest detected delay fault. This is done by using the earliest arrival and latest stabilization time for transitions, as shown in Fig. 2. Waveform information during the transitions is not kept, so the needed waveform analysis cannot be done (e.g. integration). Path delay fault simulators [Smith 85] do not use delay information, since “robust” tests that are independent of delays in other parts of the circuit are sought. Multi-valued logic systems are used which include stable signals, hazard-free transitions, and possible hazard pulses.

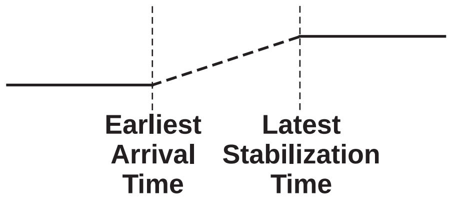

Figure 2. Simplified Waveform used in Delay Fault ATPG

2.3 Symbolic Simulators

Various symbolic simulators have been described in the literature, including both logic and timing simulators [Bryant 90].

Symbolic Fault Simulation [Cho 89], Symbolic Logic Simulation [Bose 89], and Symbolic Simulation-based Verification [Jain 92] use boolean variables together with the constants 0 and 1 in the simulation. In essence, the circuit is evaluated for many input combinations simultaneously.

The symbolic variable in the above simulators is used to represent a logic value. In Time-Symbolic Simulation [Ishiura 89, 90], and Symbolic Timing Verification [Amon 92], the symbolic delay is used to represent time. These simulators are used for timing verification. Variable delays are assigned to gate delay values, and it is determined whether the circuit will operate at the specified speed. These simulators are not suitable for analyzing Output Waveform Analysis, as they are designed to determine the maximum propagation delay through a circuit as a function of various variable delays, whereas the actual shape of the output waveform for particular input combinations and a variable delay fault is needed.

A symbolic simulator for delay fault diagnosis has been described [Girard 92]. The simulator is a 6-valued simulator consisting of stable signals, hazard-free transitions, and transitions with hazard pulses. Failing vectors are simulated and possible delay fault locations are found by intersecting the fault locations for the individual vectors. Timing information is not considered.

3 WAVEFORM SIMULATOR FOR FIXED DELAYS

The simplest implementation of WSIM uses fixed delays for gates. This implementation is a subset of the symbolic waveform simulator proposed that uses one variable delay, as described in the next section. An implementation of WSIM without this variable delay is first described in this section, to introduce the waveform representation and algorithms used. This simulator is similar to running a conventional timing simulator. It is more convenient however, as the internal representation simplifies the waveform analysis that needs to be performed.

The simulator is described in terms of a logically complete set of primitives (AND and NOT) and a separate DELAY element. Other gate primitives can easily be added if desired. The representation chosen for waveforms is central to the algorithms described here. The representation for a waveform where all delays are fixed is given in Definition 1. This representation is extended in to include a variable delay in Sec. 4.

Definition 1:

A waveform is represented as an ordered list of transitions (a_1 a_2 a_3 a_4 ...), where the real number constants a_i refer to the times of the transitions in the waveform, and $a_i < a_j$ if $i < j$. The transitions are extended to include variable delay in the next section. The BNF description of the syntax is:

```
transitions   ::= (point-in-time*)
point-in-time ::= const
```
(1)

By convention, the waveform starts at 0 at time negative infinity. A few examples of waveforms are given below:

() or (+∞)	:	Logic 0;
(-∞)	:	Logic 1;
(0)	:	Rising transition at time 0;
(-∞ 20 50)	:	A 1-hazard starting at time 20 and ending at time 50.

Logic functions and delay operations are separated in the simulator. Delays can be added either at the inputs or the output of a gate. For simplicity, equal rising and falling delays will be added to the outputs of gates in this report. The output waveform of a gate is computed as a function of the input waveforms, and then modified to account for the gate delay. The three primitives, NOT, AND and DELAY, are described below.

3.1 NOT Function

The NOT function for fixed delays is very simple, and shows the power of the waveform representation chosen. Only the first element of the list needs to be inspected, regardless of the number of transitions in the waveform. Rising transitions become falling transitions, and vice versa.

$$\left.\begin{aligned}\text{NOT}(-\infty\ a_1\ a_2\ a_3\ \ldots) &= (a_1\ a_2\ a_3\ \ldots)\\ \text{NOT}(a_1\ a_2\ a_3\ \ldots) &= (-\infty\ a_1\ a_2\ a_3\ \ldots)\end{aligned}\right\} \quad (2)$$

3.2 AND Function

The two-input AND function is described in this section; an *n*-input AND function is computed by taking 2 inputs at a time. Whereas the waveform was interpreted as a list of transitions for the NOT function, it is more convenient to consider the waveform as a series of pulses for the AND function. This is done by considering two terms at a time in the waveform representation. For example, the waveform (3 4 6 10) represents two pulses, the first between times 3 and 4, and the second between times 6 and 10.

The AND function is computed by considering one pulse in each input waveform at a time, until there are no more pulses. The three possible cases for a two-input AND function are shown in Fig. 3. The earliest transition is assumed to be in input waveform A, else the inputs are switched.

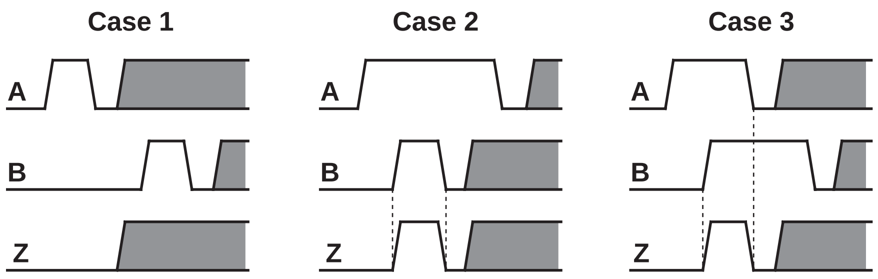

Figure 3. Three Cases for AND Function

Case 1: The first pulse in waveform A ends before the first pulse in waveform B starts. This means that the first pulse in A disappears. The AND function is repeated with the first pulse removed from waveform A, and waveform B kept the same.

Case 2: The first pulse in waveform B ends before the first pulse in waveform A ends. This means that the first pulse in B appears at the gate output. The AND function is repeated with waveform A unchanged, and the first pulse removed from waveform B.

Case 3: The first pulse in waveform A ends before the first pulse in waveform B ends. This means that a pulse starting from the beginning of the first pulse in B, and ending at the end of the first pulse in A, appears at the gate output. The AND function is repeated with waveform B unchanged, and the first pulse removed from waveform A.

After performing the above operations, the resulting waveform can sometimes be simplified. If there are two transitions at the same time, the transitions can be removed:

$$(\dots\ a_1\ a_2\ a_2\ a_3\ \dots) = (\dots\ a_1\ a_3\ \dots) \tag{3}$$

3.3 DELAY Function

Delay is added to a waveform in two steps. First transport delay is added, and then inertial delay is added. Transport delay is very simple. If a delay *del* is to be added, then every transition is delayed by *del*:

$$\text{TRANSPORT\_DELAY:}\quad (a_1\ a_2\ a_3\ \dots) \rightarrow (a_1{+}del\ a_2{+}del\ a_3{+}del\ \dots) \tag{4}$$

Inertial delay is more complex, as hazard pulses shorter than the gate delay are removed from the waveform. This is sometimes called high frequency rejection. Inertial delay is computed by analyzing the waveform as a series of pulses, and removing those that are shorter than the gate delay. Unlike the AND function, both positive and negative pulses must be considered. The algorithm is shown in Table. 1.

Table 1. Inertial Delay Algorithm for Fixed Delays

```
Inertial_Delay(Waveform A, Delay del)
  While transitions remain in A do
    Begin
      If (second_transition(A) - first_transition(A)) < del
         Then /* remove first pulse */
            A = remove_first_2_transitions(A);
         Else /* transition remains */
            Add first_transition(A) to output waveform;
            A = remove_first_transition(A);
    End;
```

As an example, consider the waveform (0 2 3 6 8 10 11 12) shown in Fig. 5, and a gate delay *del* = 1.5.

	Output Waveform
The first pulse (0 2) > *del*, so transition (0) appears at the output:	(0)
The next pulse (2 3) < *del*, so transitions (2 3) disappear:	(0)
Pulse (6 8) > *del*, so transition (6) remains:	(0 6)
Pulse (8 10) > *del*, so transition (8) remains:	(0 6 8)
Pulse (10 11) < *del*, so transitions (10 11) disappear:	(0 6 8)
Pulse (12 ∞)* > *del*, so transition (12) remains:	(0 6 8 12)

* Note that transitions at ∞ can be added without affecting the waveform.

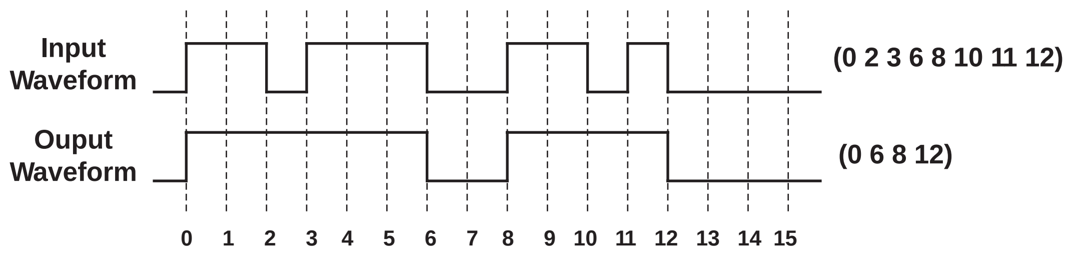

Figure 5. Inertial Delay Example (*del* = 1.5)

3.4 Example

A small example circuit is shown in Fig. 6. This example is included to show the waveform representation used for a few simple waveforms. The delay of the OR gates is 2 units, and the delay of the NAND gates is 1 unit.

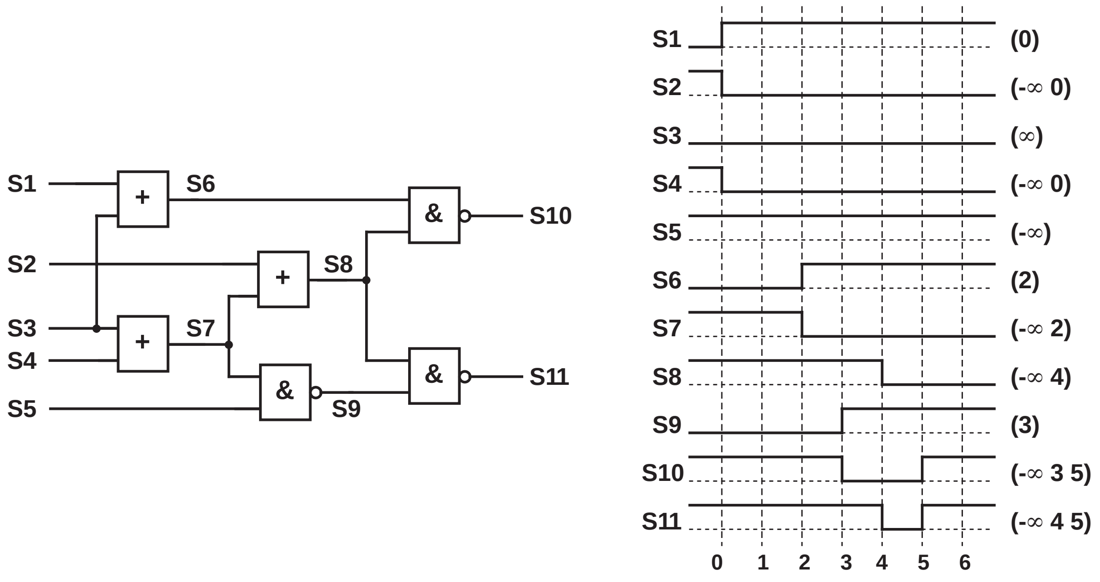

Figure 6. Example Circuit

3.5 Computing Output Waveform Analysis Information

Collecting the necessary information for Output Waveform Analysis is discussed in this section.

Stability Checking is very simple. The waveform is stable during the interval starting at α and ending at β, if there are no transitions in the waveform during this interval:

$$\texttt{STABLE}(<a_1\ a_2\ ...\ a_n>, \alpha, \beta)\text{: If for all } i\text{:, } a_i < \alpha \text{ OR } a_i > \beta \quad (5)$$

The integral values used in Pre-Sampling Waveform Analysis are computed by considering the waveform as a series of pulses, as was done for the AND gate. The integral of the waveform is the sum of the integral of the pulses, i.e.:

$$\int_{\alpha}^{\beta}(A\ B\ C\ D\ E)dt = \int_{\alpha}^{\beta}(A\ B)dt + \int_{\alpha}^{\beta}(C\ D)dt + \int_{\alpha}^{\beta}(E\ \infty)dt \tag{6}$$

Each pulse (*A B*) during the integration period contributes *B*-*A* to the total integral.

$$\int_{\alpha}^{\beta}(A\ B)dt = B - A \qquad if \quad \alpha \le A \le B \le \beta \tag{7}$$

Pulses that partly overlap with the integration period are narrowed, as shown below:

$$\left.\begin{aligned}\int_{\alpha}^{\beta}(A\ B)dt &= \int_{\alpha}^{\beta}(\alpha\ B)dt \qquad if \quad A < \alpha \\ \int_{\alpha}^{\beta}(A\ B)dt &= \int_{\alpha}^{\beta}(A\ \beta)dt \qquad if \quad B > \beta\end{aligned}\right\} \tag{8}$$

3.6 Implementation

A prototype of the simplified waveform simulator WSIM has been implemented in Common LISP. The simulator is similar to an event driven simulator, except that no explicit event queue is needed to schedule events. A levelized netlist is read in, and the simulation is performed on all gates in order. No optimizations have been implemented, and the core algorithm is only approximately 160 lines of LISP code. The data structures for waveforms are simply lists as have been described above, and built-in functions simplify operations. For example, once a two-input AND function is implemented, *n*-input ANDs are coded using the built-in function `reduce`, i.e. `(reduce 'Process_2_AND in-waves)`.

There are two ways to run WSIM: simulating all input vectors together in one pass; or simulate single pairs of vectors in turn. For sequential circuits with feedback, it is necessary to simulate each pair of vectors individually. The advantage of simulating all vectors together is that each gate is only evaluated once, but the memory requirements can be high. Simulating single pairs of vectors is substantially slower. An effective alternative is a hybrid solution, where multiple simulation passes are performed, each with a fraction of the input vectors.

Table 2 shows the runtime of the simulator compiled under Lucid Common Lisp compared with Verilog-XL (with the accelerate flag, -a), both running on a SparcStation2 with 32MB of memory. The times shown are computed by subtracting the time for simulating 20 vectors from the time to simulate 120 and 220 random vectors, to eliminate the startup overhead for both cases. A single pass was used for WSIM, except for the (*) entries which required too much memory. (The overhead to read the netlist was higher for Verilog; probably due to the error checking done.) The times in the table represent the average of five simulation runs.

Table 2. Runtime of Verilog-XL and WSIM

Circuit	Verilog-XL (sec)		WSIM (sec) (all vectors together)	
	t(120)-t(20)	t(220)-t(20)	t(120)-t(20)	t(220)-t(20)
ALU181	1.2	2.2	0.3	0.7
c432	2.1	4.3	0.8	1.9
c499	1.9	3.9	1.1	2.7
c880	4.6	8.8	1.6	3.6
c1355	2.1	4.0	3.1	8.1
c1908	4.5	9.5	4.9	13.23
c2670	15.0	31.6	3.5	8.0
c3540	7.7	16.1	9.49	24.7
c5315	32.7	67.0	17.8	53.0
c6288	41.1	81.3	*	*
c7552	51.5	102.3	*	*

Simulations were done for both 100 and 200 vectors to see the runtime dependence on the number of vectors. Verilog runtimes are linear in vector size. WSIM runtimes are linear for the smaller circuits. For the larger circuits, multiple passes of WSIM would decrease the runtime.

4 SYMBOLIC WAVEFORM SIMULATOR WITH ONE VARIABLE DELAY

In the complete version of the symbolic waveform simulator WSIM, one of the nodes in the circuit is assumed to have an arbitrary delay. This allows the simulation of a localized delay fault or *gate delay fault* in the circuit. Simulating multiple varying delays or *path delay faults* is not practical using this simulator, unless the delays are correlated and track each other. The variable delay is a real number, and is denoted by δ in this report.

The biggest complication over the previous algorithm is that the waveform now depends on the value of the variable delay, and could be very different for different delays, as hazard pulses can disappear or new pulses appear. This is shown in Fig. 7. Assume both the transport and inertial delay of the AND gate are 2 units. If the variable delay δ which appears in waveform B exceeds 3 units, then the pulse in input A does not propagate through the AND gate due to the inertial delay of the gate.

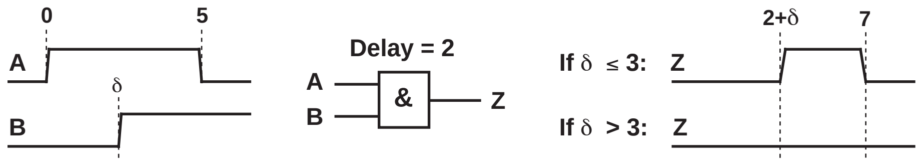

Figure 7. Hazard Pulse “Disappearing” Example

The waveform representation with one variable delay is more complex than before. The list of transitions, `transitions`, is similar to the case without the variable delay, except that the transitions can now depend on the variable delay. For example, a transition at time 3 + δ is represented as `point-in-time` = <3+δ>.

```
transitions    ::= (point-in-time*)                      }  (9)
point-in-time  ::= const | <const + variable-delay>
```

The complete waveform is now one or more wave packets, each valid for an interval of the variable delay.

```
waveform       ::= {wave-packet+}
wave-packet    ::= (interval transitions)
interval       ::= [start end]                          }  (10)
start          ::= const
end            ::= const
```

Note that each `wave-packet` is valid for all time, but only for the specified interval of the variable delay. Each list of transitions is ordered, meaning that a transition with variable delay cannot overlap other transitions, i.e. if `wave-packet` = ([α β] (x <y+δ> z)), then α > x-y, and β < z-y, in order for there to be no intersections.

Referring to the AND gate in Fig. 7, the waveform at input A is represented by {([0 ∞] (0 5))}. This means that the list of transitions (0 5) is independent of the variable delay and therefore valid for any value of the variable delay. Similarly, the waveform at input B is {([0 ∞] (<0+δ>))}. The list of transitions (<0+δ>) describes a single transition at time <0+δ>, or δ. The output waveform is expressed as two wave packets, one for δ≤3, and the other for δ>3:

$$Z = \{([0\ 3]\ (\langle 2+\delta \rangle\ 7))$$
$$([3\ \infty]\ ())\}$$

4.1 NOT Function

The NOT function is similar to before. The simple NOT function is applied to the `transitions` in each `wave-packet`.

4.2 AND Function

The AND function is more complicated, as new `wave-packets` can be generated. There are four levels of AND functions, each operating on successively simpler waveforms. The idea is to reduce the complexity of the waveforms until the variable delay is not a factor anymore.

Top-level `Process_n_AND` takes any number of waveforms. It simply calls the `Process_2_AND` function repeatedly on two inputs at a time. At present no sorting of inputs is done to improve efficiency.

The example in Fig. 8 will be used to illustrate how the AND function is performed.

The second level `Process_2_AND` function takes two `waveforms` as inputs. The overlap interval between the first wave packets of the two waveforms is found, and `Process_tr_AND` is called with the overlap interval and the first two wave packets. This is repeated until all the wave packets in the input waveforms are processed.

For the example in Fig. 8, there are two ranges of the variable delay that need to be considered separately. `Process_tr_AND` is called twice, for the two intervals [0 2] and [2 ∞].

`Process_tr_AND` takes two `transitions` and an `interval`. All possible values of the variable delay, δ, for which the two lists of transitions can intersect are found. These are the boundary conditions where hazard pulses can be created or eliminated. The lowest level `Process_simp_tr_AND` function is then called for the intervals between transitions.

In the interval [0 2], in the example in Fig. 8, there is an intersection at $\delta=1$. This is because both the list of transitions trA = $(1\ \langle 3+\delta \rangle)$, and trB = $(-\infty\ 4)$ have a transition at time 4 when $\delta=1$. Therefore, `Process_simp_tr_AND` is called three times, for the intervals [0 1], [1 2], and [2 ∞].

`Process_simp_tr_AND` takes two `transitions` and the minimum value `min_delay` of the variable delay in the specified interval. There is no need to consider the variable delay anymore, as it is within limits that it cannot create or remove pulses. Therefore the function is similar to the non-symbolic AND function used in Sec 3.2.

The different wave packets are then combined to get the final waveform.

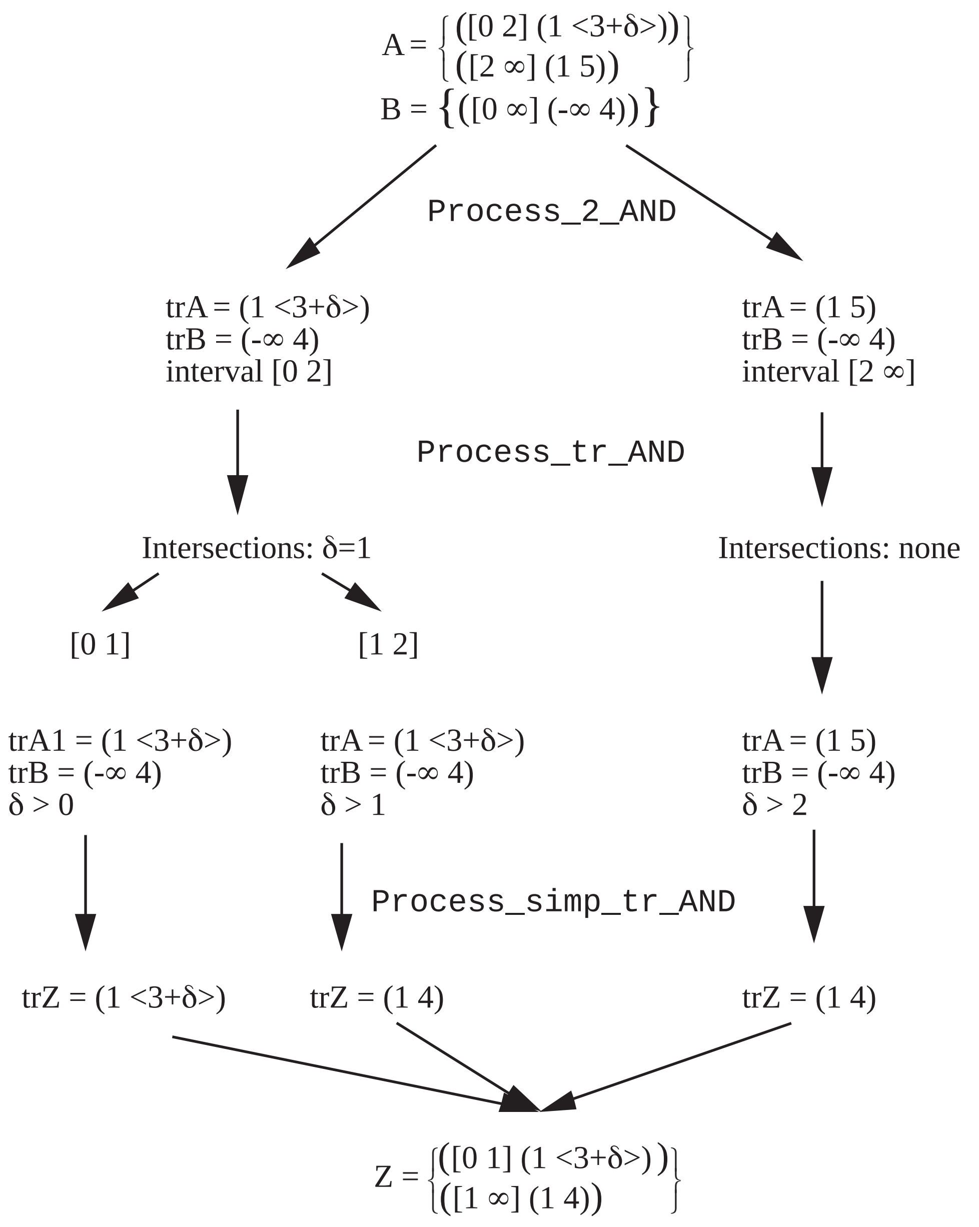

Figure 8. WSIM AND Gate Example

4.3 DELAY Function

Transport delay is added by delaying every transition by *del*. This is the same as in Sec. 3.3, except that every wave packet must be handled. Inertial delay is more complex, as pulses that depend on the variable delay δ could disappear for certain values of delay. The inertial delay function is implemented in levels, similar to the AND function.

The top level `Process_inertial_delay`, takes a waveform and a value for inertial delay, and calls `Simp_process_inertial_delay` on every wave packet. `Simp_process_inertial_delay` takes an interval, a list of transitions, and the value of inertial delay. Like the AND function, values of the variable delay δ for which transitions can intersect are found. Note that the value of inertial delay needs to be taken into account when computing the intersection points. The lowest level function `Simp_pr_inertial_delay`, is similar to the inertial delay function in Sec. 3.3.

As an example, consider the waveform A = {([0 7] (2 <3+δ> 10))} and an inertial delay of *del*=2. This waveform consists of only one wave packet, and it is only valid for δ≤7, else the transitions at <3+δ> and 10 will overlap.

`Simp_process_inertial_delay` finds the values of δ that will cause hazard pulses to be eliminated or appear. In this case, the values are 1 and 5. When δ=1, the first pulse will be (2 4), which is the same size as the inertial delay. This means that the first pulse will disappear for δ<1. When δ=5, the last pulse is (8 10), once again the size of the inertial delay. Therefore the lowest level function is called three times, with δ in the range [0 1], [1 5] and [5 7].

The resulting waveform is Z = {([0 1] (10)) ([1 5] (2 <3+δ> 10)) ([5 7] (2))}.

4.4 Examples

WSIM was run on the ISCAS’85 circuits to investigate the shape of the output waveforms. Two waveforms are shown as examples, with faults near the primary inputs.

Z1 = {([0 1] (-∞ <13+δ> 16 19))
([1 3] (-∞ 19))
([3 ∞] (-∞ 15 17 19))}

Z2 = { ([0 3] (-∞ 32 41 55))
([3 5] (-∞ 32 41 44 <44+δ> 55))
([5 ∞] (-∞ 32 41 44 52 55))}

Z1 is from the ALU181, and Z2 is from c3540. Both examples show that the output waveform is a complicated function of the variable delay δ. For some ranges of δ, the waveforms do not change, and then abruptly change as pulses appear or disappear. It is difficult to find the information above using a conventional timing simulator.

5 SPEEDING UP WSIM

Since the symbolic implementation of WSIM can be significantly slower than the simple implementation, it is worth trying to reduce unnecessary computations. Only nodes in a circuit with transitions need to be evaluated using WSIM. Simpler simulators can be used to pre-process the circuit to determine which nodes to evaluate. Two functions have been implemented: a forward 3-value simulator and reverse 4-value simulator. These are discussed below.

5.1 3-SIM

3-SIM uses the values: 0 for a constant zero; 1 for a constant one; and X for possible transitions. Any inputs that are not constant are assigned the value X, and then the gates are traversed in forward levelized order using the truth tables in Table 3. The function performed by 3-SIM is called "hazard simulation" and was described in [Eichelberger 65], where the value 1/2 was used instead of X.

Table 3. 3-SIM Truth Tables

0	1
1	0
X	X

NOT

	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

AND

5.2 4-SIM

After 3-SIM there are still nodes that do not have to be evaluated. For example, if a gate is not sensitized to the circuit outputs, then the input waveforms at the gate do not have to be evaluated (except when all transitions are needed, as in power estimation, for example). The simulator 4-SIM is used to find the nodes that need to be evaluated. 4-SIM uses four logic values: 0 and 1 for constants; W for nodes with possible transitions that need evaluation; and X for nodes with possible transitions that do not need evaluation.

4-SIM is run after 3-SIM, and traverses the circuit in reverse levelized order, changing X's to W's when necessary. The first step is to change all primary outputs that are X to W. For each gate, if the gate output is W, then all inputs that are X become W.

Table 4 shows the percentage of nodes in the ISCAS'85 benchmark circuits that need evaluation after running 3-SIM and 4-SIM. The percentages represent the circuit activity, and are the average over 120 random vectors.

Table 4. Percentage of Nodes that Need Evaluation after 3-SIM and 4-SIM

Circuit	After 3-SIM	After 4-SIM
ALU181	57.2	54.3
c432	61.5	50.8
c499	85.9	85.5
c880	59.9	54.2
c1355	82.4	82.2
c2670	67.6	54.2
c3540	62.9	46.6
c5315	66.5	47.0
c6288	87.9	87.9
c7552	70.6	65.4

6 RESULTS

A few examples using WSIM are shown in this section. Most of the examples involve Output Waveform Analysis.

6.1 Stability Checking: Number of Output Transitions per Vector

The greater the number of hazard pulses in the output waveforms, the greater the possibility of test invalidation for conventional delay testing. These pulses will be detected using Stability Checking [Franco 91]. Figure 9 shows the distribution of the number of transitions at the outputs of the ISCAS’85 combinational benchmark circuits. The distributions are based on counting the number of output transitions for each output for 120 random vectors. The fixed delay version of WSIM was used, since there were no variable delays. Some of the circuits have multiple transitions a small fraction of the time, while others have a reasonable fraction of multiple output transitions. C6288 is very different from the rest; there was even one output with 80 transitions.

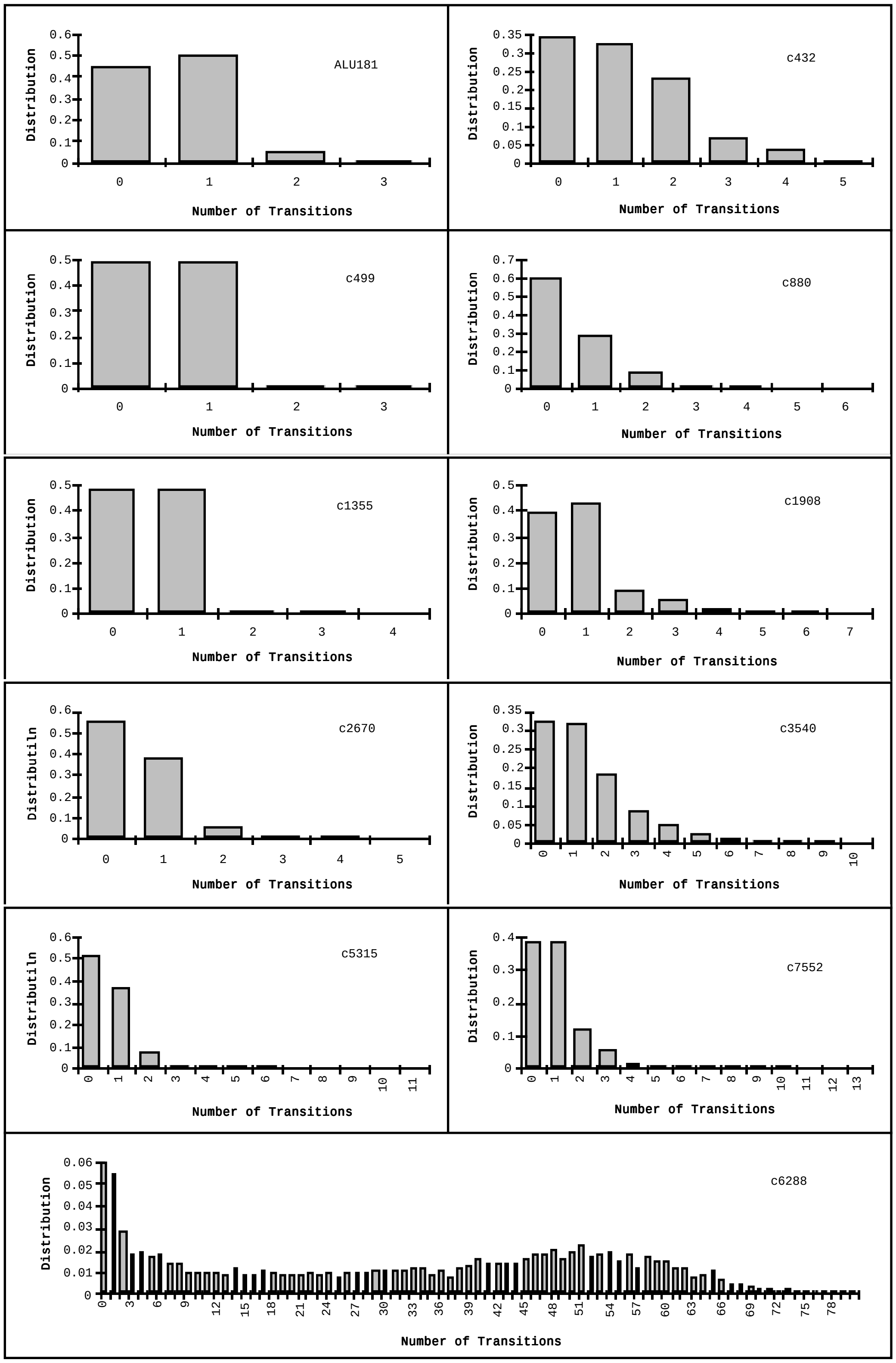

Figure 9. Distribution of Number of Output Transitions

6.2 Power Estimation -- Number of Transitions per Vector

By repeating the analysis in Sec. 6.1 for all nodes instead of only the output nodes, the switching activity in circuits can be estimated. This is useful for computing dynamic power dissipation, which depends on the number of times node capacitances are charged and discharged. Table 5 shows the average number of transitions per node, per input vector for all nodes, as well as the output nodes. (The average value of the output nodes is the mean of the distributions in Fig. 9.)

Table 5. Average number of transitions per node per vector

Circuit	All Nodes	Output Nodes
ALU181	0.423	0.613
c432	0.545	1.131
c499	0.520	0.526
c880	0.523	0.527
c1355	0.739	0.536
c1908	0.790	0.886
c2670	0.500	0.513
c3540	0.649	1.370
c5315	0.744	0.634
c6288	12.956	31.660
c7552	0.874	1.081

6.3 Delay Fault Coverage Using Integration

A simple fault simulator has been implemented based on the implementation of WSIM with fixed delays. Basically, the fault-free circuit is simulated, and the integral values as well as the sampled values are stored. Various sized delay faults are then injected into each node in turn, and the circuit is resimulated. To compare integration and sampling fairly, only detectable delay faults were injected, since smaller delay faults could not be detected by sampling. The timing slack [Hitchcock 82] is computed for each node, and then delay faults larger than the slack are injected. Figures 10-12 show the results for some of the benchmark circuits. The “sampling” curve is the fault coverage for conventional delay testing, and the other curves are for different forms of integration. A fault was considered detected if the integral differed from the fault-free integral by more than T_C/RES, where RES is the resolution of the integrator. The curve with f=0.5 was computed by only integrating over the last half of the cycle.

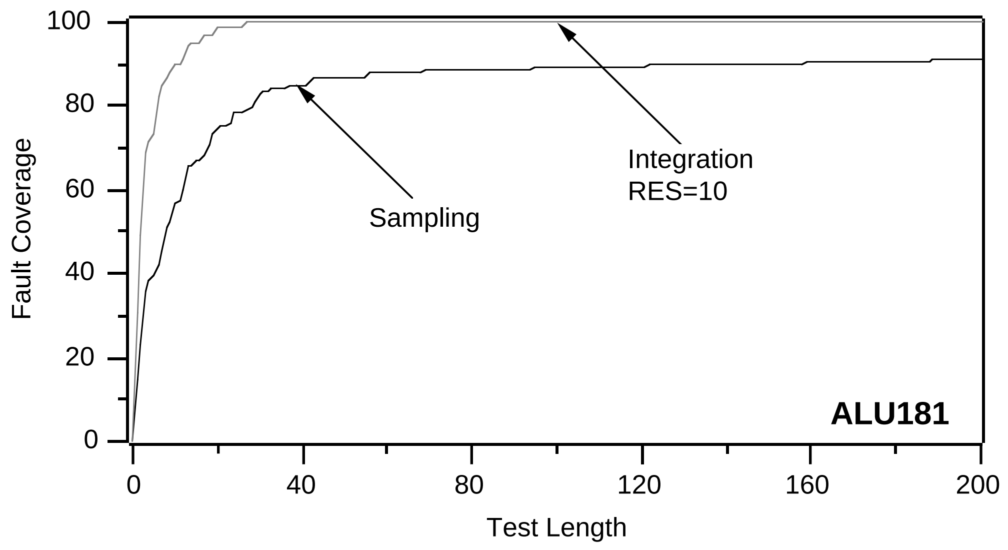

Figure 10. Fault Coverage for ALU181

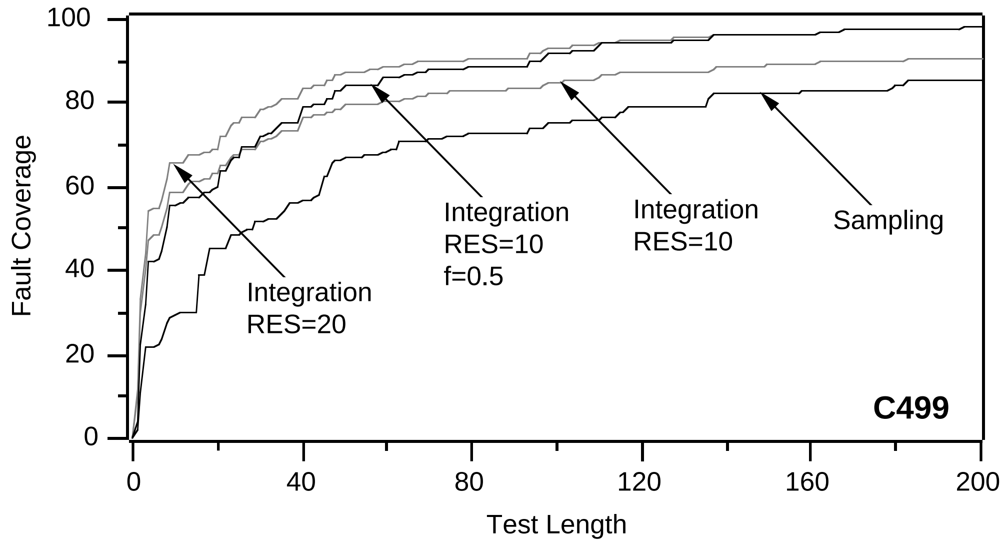

Figure 11. Fault Coverage for c432

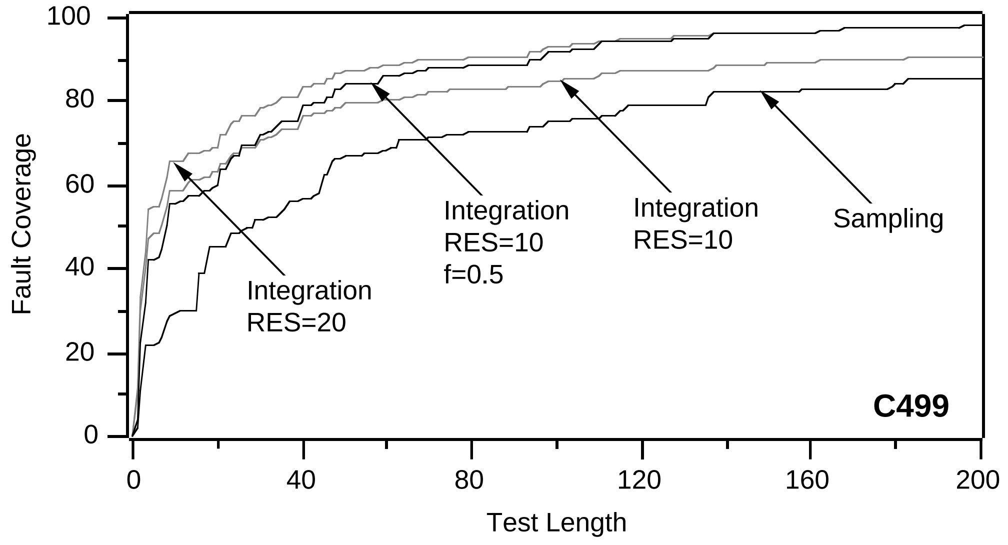

Figure 12. Fault Coverage for c499

7 CONCLUSION

A new waveform representation and algorithm for waveform simulation have been presented in this report. By processing all transitions for each gate in turn, there is no need for an explicit event queue. The main motivation was that existing simulators were not suitable for evaluating Output Waveform Analysis.

The fixed delay implementation of WSIM is comparable in speed to Verilog for timing simulation, and is faster when computing waveform information. WSIM with one variable delay is more complex as much more information is computed, but seems feasible as long as only one pair of inputs is simulated at a time. The only alternative would be to run a timing simulator repeatedly for many different values of the variable delay.

WSIM is also suitable for other applications, such as dynamic power estimation, where the actual number of transitions at each node is needed.

ACKNOWLEDGMENTS

The authors wish to thank LaNae Avra and Samy Makar for their comments and suggestions. This work was supported in part by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization and administered through the Office of Naval Research under Contracts No. N00014-85-K-0600 and N00014-92-J-1782, and in part by the National Science Foundation under Grants No. MIP-8709128 and No. MIP-9107760.

REFERENCES

[Amon 92] Amon, T., and G. Borriello, "An Approach to Symbolic Timing Verification," *Proc. 29th Design Automation Conf.*, Anaheim, CA, pp. 410-413, June 8-12, 1992.

[Bose 89] Bose, S., and A.L. Fisher, "Verifying Pipelined Hardware using Symbolic Logic Simulation," *Proc. ICCD*, Cambridge, MA, pp. 217-221, Oct. 2-4, 1989.

[Bryant 90] Bryant, R.E., "Symbolic Simulation -- Techniques and Applications," *Proc. 27th Design Automation Conf.*, Orlando, FL, pp. 517-521, June 24-28, 1990.

[Cho 89] Cho, K., and R.E. Bryant, "Test Pattern Generation for Sequential MOS Circuits by Symbolic Fault Simulation," *Proc. 26th Design Automation Conf.*, Las Vegas, NV, pp. 418-423, June 25-29, 1989.

[Eichelberger 65] Eichelberger, E. B., "Hazard detection in combinational and sequential switching circuits," *IBM J. Res. Develop.*, Vol. 9, pp. 90-99, Mar. 1965.

[Franco 91] Franco, P., and E.J. McCluskey, "Delay Testing of Digital Circuits by Output Waveform Analysis," *Proc. 1991 Int. Test Conf.*, Nashville, TN, pp. 798-807, Oct. 26-30, 1991.

[Franco 94b] Franco, P., and E.J. McCluskey, "Three-Pattern Tests for Delay Faults," *Proc. 12th IEEE VLSI Test Sym.*, Cherry Hill, NJ, pp. 452-456, Apr. 25-28, 1994.

[Girard 92] Girard, P., C. Landrault, and S. Pravossoudovitch, "Delay-Fault Diagnosis Based on Critical Path Tracing from Symbolic Simulation," *Proc. ISCAS*, San Diego, CA, pp. 1133-1136, May 10-13, 1992.

[Hitchcock 82] Hitchcock, R.B., G.L. Smith, and D.D. Cheng, "Timing Analysis of Computer Hardware," *IBM J. Res. Develop.*, Vol. 26, No. 1, pp. 100-105, Jan. 1982.

[Ishiura 89] Ishiura, N., M. Takahashi, and S. Yajima, "Time-Symbolic Simulation for Accurate Timing Verification of Asynchronous Behavior of Logic Circuits," *Proc. 26th Design Automation Conf.*, Las Vegas, NV, pp.497-502, June 25-29, 1989.

[Ishiura 90] Ishiura, N., Y. Deguchi, and S. Yajima, "Coded Time-Symbolic Simulation Using Shared Binary Decision Diagram," *Proc. 27th Design Automation Conf.*, Orlando, FL, pp. 130-135, June 24-28, 1990.

[Jain 92] Jain, P., and G. Gopalakrishnan, "Some Techniques for Efficient Symbolic Simulation-based Verification," *Proc. ICCD*, pp. 598-602, Cambridge, MA, Oct. 11-14, 1992.

[Lesser 80] Lesser, J.D., and J.J. Shedletsky, "An Experimental Delay Test Generator for LSI Logic," *IEEE Trans. Computers*, Vol. C-29, No. 3, pp. 235-248, Mar. 1980.

[Pramanick 89] Pramanick, A.K., and S.M. Reddy, "On the Computation of the Ranges of Detected Delay Fault Sizes," *Proc. Int. Conf. on Comp. Aided Design*, pp. 126-129, Nov. 1989.

[Smith 85] Smith, G.L., "Model for Delay Faults Based Upon Paths," *Proc. 1985 Int. Test Conf.*, Philadelphia, PA, pp. 342-349, Nov. 19-21, 1985.

[Waicukauski 87] Waicukauski, J.A., E. Lindbloom, B.K. Rosen, and V.S. Iyengar, "Transition Fault Simulation," *IEEE Design & Test*, pp. 32-38, Apr. 1987.