Delay Fault Simulation with Bounded Gate Delay Model

Soumitra Bose*
Design Technology, Intel Corp.
Folsom, CA 95630
bose@ieee.org

Hillary Grimes and Vishwani D. Agrawal
Dept. of ECE, Auburn University
Auburn, AL 36849
grimehh@auburn.edu, vagrawal@eng.auburn.edu

Abstract

Previously reported work on path and gate delay tests fail to analyze path reconvergences when a bounded gate delay model is used. While robust path delay tests are of the highest quality, most path faults are only testable nonrobustly. Many non-robust tests are usually found but, in practice, are easily invalidated by hazards. The invalidation of non-robust tests occurs primarily due to non-zero delays of off-path circuit elements that may reconverge. Thus, non-robust tests are of limited value when process variations cause gate delays to vary. For gate delay faults, failure to recognize the correlations among the ambiguity waveforms at inputs of reconvergent gates cause fault coverages to be optimistic. This paper enhances a recently published ambiguity simulation algorithm [5] to accurately measure both non-robust path and gate delay coverages for the bounded delay model. Experimental results for the ISCAS circuits show accurate results are often 20-30% less than the optimistic ones that fail to analyze signal reconvergences.

1. Introduction

Delay test [11, 14, 21] is the most widely accepted method of verifying that a manufactured design meets its timing constraints. Robust tests have the highest quality, but it is well known that few faults are robustly testable [8, 14]. On the other hand, many faults have non-robust tests that only propagate hazards along the path of interest. Such tests can be invalidated by gate delays that are either imprecisely known or vary from one circuit to another. Only some non-robust tests are validatable [7, 20] given that certain other faults are robustly tested. Although the shortcomings of non-robust and validatable non-robust tests are known [13], attempts to enhance them have been limited [17].

With increasing process variations in sub-micron devices, the need to model imprecise circuit delays has prompted research in various directions, including Monte-Carlo techniques [12, 16, 24], bounded delay simulation [3, 9, 10, 15, 22], statistical timing analysis [1, 2, 23, 25] and the classical bounded delay timing analysis [6]. While timing analysis finds corners in the design process, it does so in a vector independent manner by using the maximum values of gate delays. This makes it unsuitable in a test environment, where test vectors are designed to expose delays that exceed their minimum values [11, 19]. Bounded delay simulation holds promise but there are few universally accepted correct algorithms. This paper further develops a recently presented simulation algorithm for test quality evaluation [5], and shows how slight modifications can be used for both path and gate delay fault simulation.

Test quality evaluation involves finding the earliest arrival and latest stabilization times for output signals of a circuit. Delay fault simulation involves finding the minimum size of a fault that guarantees detection at an observation point. This threshold, for faults that can impact a sensitized path to an output, does not necessarily equal the difference between the clock period and the earliest arrival time of a transition. This paper presents an algorithm to extract this parameter for both the gate and path delay models.

The inherent reason for inaccuracy in bounded delay simulation is the failure to recognize path reconvergences in circuits. At a reconvergent gate, the ambiguity waveforms at its inputs have correlations that are ignored by most algorithms [11]. As shown later in the paper in Section 5, the gate delay fault coverages obtained are too optimistic. For path delay faults, coverages obtained by our algorithm are more realistic than those from traditional zero-delay non-robust simulation [21]. For the first time, both path and gate delay faults can be simulated using a delay model that allows for process variations. Unlike robust delay test, more faults are found testable. Unlike traditional non-robust delay tests, the tests found by this algorithm are not invalidated by other circuit delays as long as individual gate delays lie within the specified bounds. Results in Section 5 show inaccuracies as high as 20-30% are possible when signal reconvergences are ignored in both gate and non-robust path delay fault simulation.

2. Motivation

The example of Figure 1 [6, 9] shows the error of most bounded delay simulation algorithms [11]. The (min, max) delay bounds for each gate are shown in the diagram. The notation used in the diagram is identical to the one used

* S. Bose is now with UBS Investment Bank, 1285 Avenue of the Americas, New York, NY 10019.

1-4244-1128-9/07/$25.00 © 2007 IEEE

commonly in path delay test [21]. The shaded regions in the waveforms are time intervals when logic values are unknown (*X*). Note that the unknown value shown at output *E* at time t = 3 is incorrect. Due to time correlations, signal at *B* must fall before *D* rises. A correct algorithm for timing analysis is presented by Chakraborty *et al.* [6], though its complexity would prevent the use in logic simulation. In the following paragraphs, we demonstrate the impact of incorrect hazard evaluation on both path and gate delay fault simulation algorithms.

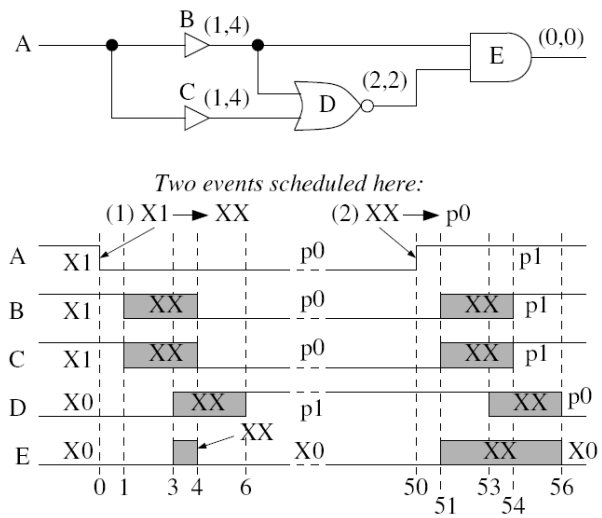


Figure 1: Illustrating pessimistic hazard simulation.

Figure 2(a) shows non-robust detection of the path delay fault (PDF) originating at *D*, and terminating at *K* along gates *F*, *H* and *J*. It is easy to show that no robust test exists for this fault [21]. The delay bounds for individual gates are shown in Figure 2(b), while signals waveforms are shown in Figure 2(c). This fault can be activated as a static hazard at the output the final gate on the path. However, this delay fault activation on a static hazard is incorrect due to a reason similar to Figure 1. The transition at *I* cannot precede the transition at *J* for the given delay bounds, and the static hazard at *K* is guaranteed not to occur.

Figure 3 illustrates the drawback in a well known algorithm for gate delay detection threshold evaluation [11]. For a given transition fault *f*, the propagation rules are almost exactly the same as those of the corresponding stuck-at fault (sa0 for slow-to-rise or sa1 for slow-to-fall). For a signal *s*, we denote a logic value FPV that is identical to the logic value of *s* when the second pattern of the vector pair is applied to the circuit with the stuck-at fault corresponding to *f*. Bounds for all gate delays are shown in the figure. The table in that figure shows earliest arrival (EA) and latest stabilization (LS) times, fault propagation value (FPV), and other parameters from reference [11]. For each signal *s*, this table shows a reference fault size, $\rho(s)$, and two reference times, $RTa(s)$ and $RTb(s)$. If δ is the size of the gate delay fault in question, the logic waveform at *s* is guaranteed to be at $FPV(s)$ inside an interval of time from $RTa(s)$ to $RTb(s)+\delta$, provided $\delta>(s)$.

Input *A* is held constant while *B* undergoes a falling transition. If output *F* is sampled at t=10, the slow-to-fall fault at *B* is detected with a threshold of $10\text{-}RTB(F) = 6$. In simple terms, the earliest arrival time for the rising transition at output *F* is 4. Therefore, the minimum size of gate delay fault that is guaranteed to be detected by this vector is 6. However, careful analysis of the example shows the correct value to be 5. This error arises due to an incorrect hazard predicted at the output of gate *E*, which then propagates to *F*. A similar error is illustrated in Figure 4. Due to an incorrect static hazard at signal *F*, the detection threshold of the slow-to-fall transition fault at input *A* is incorrectly evaluated. Assuming the output is sampled at t = 12, the detection threshold is incorrectly evaluated as 8 instead of 6.

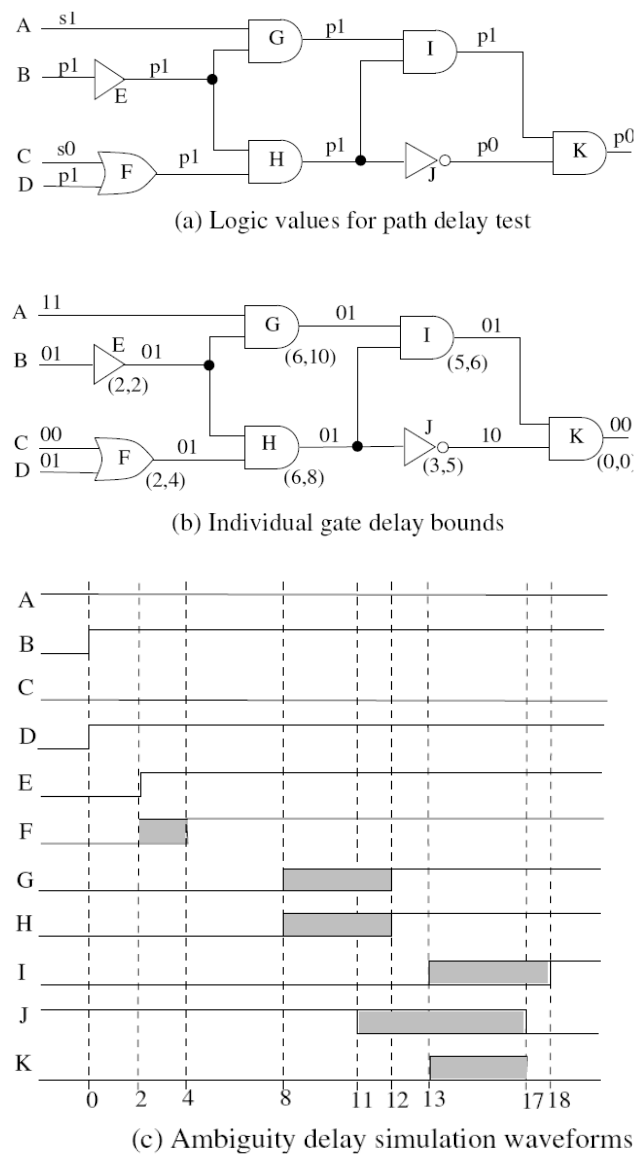
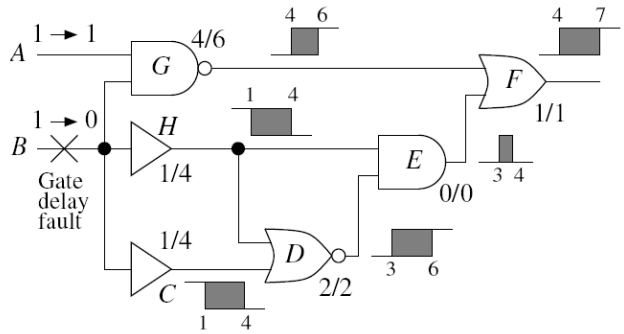


Figure 2: Erroneous non-robust detection of PDF.



signal	EA	LS	FPV	$\rho(s)$	RTA	RTB
A	∞	-∞	1	0	-∞	∞
B	0	0	1	0	-∞	0
C	1	4	1	0	-∞	1
D	3	6	0	0	-∞	3
E	3	4	0	0	-∞	3
F	4	7	0	0	-∞	4
G	4	6	0	0	-∞	4
H	1	4	1	0	-∞	1

Figure 3: Illustrating incorrect gate delay detection threshold.

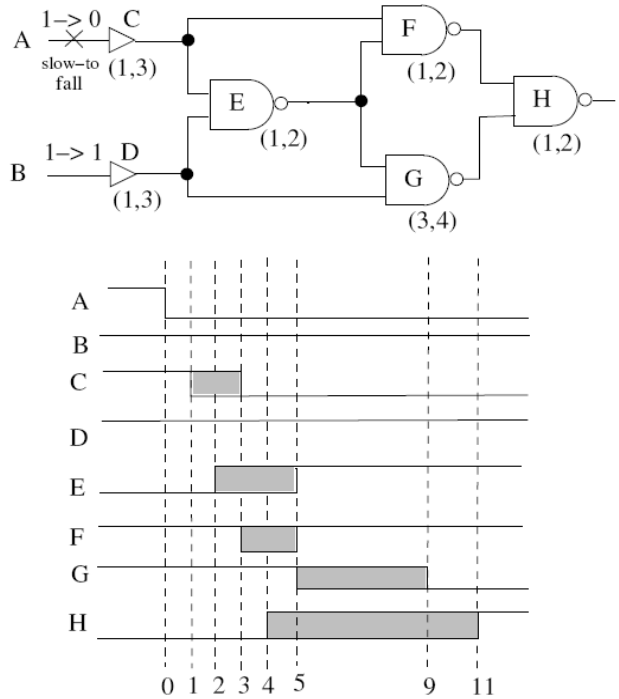
3. Non-Roboust Path Delay Faults

We present an algorithm for non-robust path delay simulation in this section. Since most of this work has been reported elsewhere [5], the presentation in this section will be brief. The algorithm can be summarized briefly as that of bounded delay logic simulation, with some modifications to evaluate the path fault detection status at each signal in the circuit. However, it should be pointed out that no algorithm for logic simulation that is widely accepted to be correct exists even today [6, 9].

The simulation state is maintained with the help of three records for each gate, *G*:

- **pv(G)**: previous value
- **fas(G)**: delay fault activation status
- **cv(G)**: current value

Both the **pv** and **cv** components can be one of 0, 1, or X values. The **pv** component represents the logic value of the gate prior to the last event obtained at the gate. The **cv** component represents the logic value after each of these events. The fault activation component, **fas**, can be one of two values: **p** or **X**. The **fas** component is set to **p** when a path delay fault has been activated from some primary input to that gate, and X otherwise. Since all transitions pass through an intermediate unknown state, the **pv** component for that gate is always *X* if the **fas** component is **p**.



signal	EA	LS	FPV	$\rho(s)$	RTA	RTB
A	0	0	1	0	-∞	0
B	∞	-∞	1	0	-∞	∞
C	1	3	1	0	-∞	1
D	∞	-∞	1	0	-∞	∞
E	2	5	0	0	-∞	2
F	3	5	1	0	-∞	3
G	5	9	1	0	-∞	5
H	4	11	0	0	-∞	4

Figure 4: Incorrect detection threshold for EXOR gate.

All signals are initialized by making the **cv** and **pv** components equal to a Boolean value and setting the **fas** component to *X*. The value of the **cv** component is used to determine the delay used for scheduling an event. An exception to this rule occurs when the **fas** component at a gate has to be changed, without any accompanying change to the current logic component **cv**. Such a situation occurs when an input of an AND gate has a sensitized falling transition, while all other off-path inputs have proper non-dominant values. If one of the off-path inputs now transition to *X*, only the **fas** component at the output changes. Such changes in the **fas** component need to be propagated to the downstream logic with zero delay, and a *pseudo-event* is propagated to all fanouts. Such a *pseudo-event* is easily differentiated by the signal scheduler by a change in the **fas** component only, without any change to **cv**.

In a bounded delay simulation environment, a Boolean **cv** value on the off-path input implies all ambiguities on these inputs have been resolved. While resembling the non-

```
EvalAmbiguityList(G) {
    ambiguity list l = ∅
    for all f ∈ HZ(in) for some input in of gate G {          // analyze fanout point f
        min_SV = max_SV = ∞, min_DV = max_DV = −∞   // initialize                                   ...(1)
        min_SV = min{d(j,f) | j ∈ spv(G)}    // min_SV = ∞ if spv(G)=∅ or f ∉ HZ(spv(G))             ...(2)
        max_SV = min{D(j,f) | j ∈ spv(G)}    // max_SV = −∞ if spv(G)=∅ or f ∉ HZ(spv(G))            ...(3)
        min_DV = max{d(j,f) | j ∈ dpv(G)}    // min_DV = ∞ if dpv(G)=∅ or f ∉ HZ(dpv(G))             ...(4)
        max_DV = max{D(j,f) | j ∈ dpv(G)}    // max_DV = −∞ if dpv(G)=∅ or f ∉ HZ(dpv(G))            ...(5)
        if min_SV = ∞                        // f ∉ HZ(spv(G))                                       ...(6)
            add f to list l with parameters (min_DV + d_G, max_DV + D_G)                            ...(7)
        else if min_DV = −∞                  // f ∉ HZ(dpv(G))                                       ...(8)
            add f to list l with parameters (min_SV + d_G, max_SV + D_G)                            ...(9)
        else if (min_DV ≥ max_SV) {          // suppress output hazard                               ...(10)
            suppress event and return ∅      // no need to analyze other f ∈ HZ(in)                  ...(11)
        }
        else {                               // fanouts that are propagated to output
            add f to list l with parameters (min_DV + d_G, max_SV + D_G)                            ...(12)
        }
    }
    return l
}
```

Figure 5: Ambiguity list evaluation at gate output.

robust path sensitization criteria, these conditions are actually more stringent due to the simulation algorithm that causes intermediate *X* values whenever a signal changes. The gate output signal transitions through an *X* state, and if off-path sensitizing values are obtained before the on-path transition, the output would be represented by the state <**pv**; **fas**; **cv**> = <*X*; *p*; 0>. In the rest of this paper, this state is abbreviated by *P0*. Similarly, *P1* denotes the state <**pv**; **fas**; **cv**> = <*X*; *p*; 1>.

The following general definitions for gate *G*, and its logic value, are used in the remainder of this section:

- **dpv(G)**: inputs with dominant **pv** values.
- **spv(G)**: inputs with sensitizing **pv** values.

During simulation, each gate *G* has an ambiguity list, each element of which consists of the following:

1. Fanouts that are originators of ambiguity regions.
2. Minimum delay from above fanout to G.
3. Maximum delay from above fanout to G.

The ambiguity list is empty for signals that have attained definite values, and non-empty for those that have a **cv** value of *X*. To manipulate this list, we define some functions:

- **HZ(G)**: fanouts in ambiguity list of *G*.
- **d(G,f)**: minimum delay to *G* from $f \in HZ(G)$.
- **D(G,f)**: maximum delay to *G* from $f \in HZ(G)$.

When gate *G* is evaluated, the ambiguity lists at the inputs are used to evaluate the list at the output. This algorithm, shown in Figure 5, may suppress the ambiguity at the output. If such event cancellation does not occur, entries are added to the ambiguity list at the output, provided it evaluates to X. Additionally, if the gate output is a fanout stem, an entry for this gate is added to the list. The algorithm in Figure 5 iterates over all fanout points that appear in the ambiguity list at any input of the gate, G, being evaluated. The minimum and maximum delays of *G* are assumed to be d_G and D_G respectively. The inputs to *G* are classified into two types: (1) **spv**(*G*): those that have non-dominant (sensitizing) **pv** values and (2) **dpv**(*G*): those that have dominant sensitizing **pv** values. Inputs that have their **pv** values set to *X* belong to neither category. For fanouts that appear in the ambiguity lists of **spv**(*G*), the quantity min_{SV} denotes the minimum delay from the fanout that causes one of these inputs of *G* to attain a dominant value. Similarly, max_{SV} is the maximum delay from the fanout that causes at least one of the inputs of *G* to attain a dominant value. For fanouts that appear in the ambiguity lists of **dpv**(*G*), the quantity min_{DV} denotes the minimum delay from the fanout that causes one of the inputs of *G* to attain a non-dominant sensitizing value. Similarly, max_{DV} is the maximum delay from the fanout that causes one of the inputs of *G* to attain a non-dominant sensitizing value. These initializations are shown in lines 1-5 of Figure 5. For a fanout point that appears in the ambiguity list of only one of the above two types of gate inputs, appropriate entries are added to the ambiguity list returned by the function (lines 6-9), provided no hazard cancellation occurs during further analysis. *Hazard cancellation occurs whenever a fanout point is obtained that has the minimum delay to an input transitioning from a dominant value larger than the maximum delay to an*

input that is transitioning from a sensitizing value (lines 10-11). In such cases, the event at the output of the gate is cancelled, and an empty list is returned by the function. If no such event cancellation occurs, the result list is modified by adding an entry for this fanout (line 12).

Figure 6 illustrates this algorithm for the example of Figure 1. At simulation time t = 1, both signals *B* and *C* have their **cv** components set to *X*. The ambiguity lists at these two nodes are shown in Figure 6(a). This had the effect of adding entries for fanout stem *A* in the ambiguity lists at *B* and *C*. Since *B* is itself a fanout stem, an entry for *B* is added to the ambiguity list at *B*. Since the **cv** component at *D* is 0, the ambiguity list at *D* is empty. Gate *E* is never evaluated at time t=1. However, its **spv** set is {B} because this is the only input with a **pv** value equal to 1. Its **dpv** set is {D}. However, this input has no ambiguity list. If the condition shown in line 10 of Figure 5 is checked, it would fail for both fanout points in the cone. When the simulation advances to t = 3, the ambiguity lists are shown in Figure 6(b). Input *D* now has a non-empty ambiguity list. The ambiguity checking condition fails for fanout stem *B* and this hazard would be suppressed. At time t = 51, the ambiguity lists are shown in Figure 6(c). Since the ambiguity list at *D* is empty, the hazard at *E* is deemed to be valid.

Figure 7 shows ambiguity lists for the example of Figure 2 at time t = 13, when gate *K* is about to be evaluated. For each signal, the **pv** and **cv** values are also shown as a pair. For *I* and *J*, these values are 0*X* and 1*X* respectively. Only signals *I* and *J* have ambiguity lists, as shown in the figure. When the algorithm of Figure 5 is executed for gate *K*, the hazard suppression condition (line 10) is found to be satisfied for fanout *H*. The hazard at *K* is therefore suppressed, and *K* remains at 0 at t = 13. Hence, the path delay fault with *K* as destination is not assumed to be non-robustly activated, unlike the traditional delay fault simulation shown in Figure 2(a).

4. Gate Delay Faults

We present an algorithm which adds the ambiguous delay model to gate delay fault simulation in this section. We first review some definitions from previously published work [11] in Section 4.1, before presenting our algorithm in section 4.2.

4.1 Terminology and Previous Work

As in Section 3, gate delays are specified by their upper and lower bounds, and the stimulus to the circuit consists of a vector pair. It is assumed the circuit has stabilized after the first vector is applied, before applying the second vector. The time frame of reference, t = 0, is assumed to be the instant the second vector is applied to the primary inputs. For each gate, *G*, the following definitions are used:

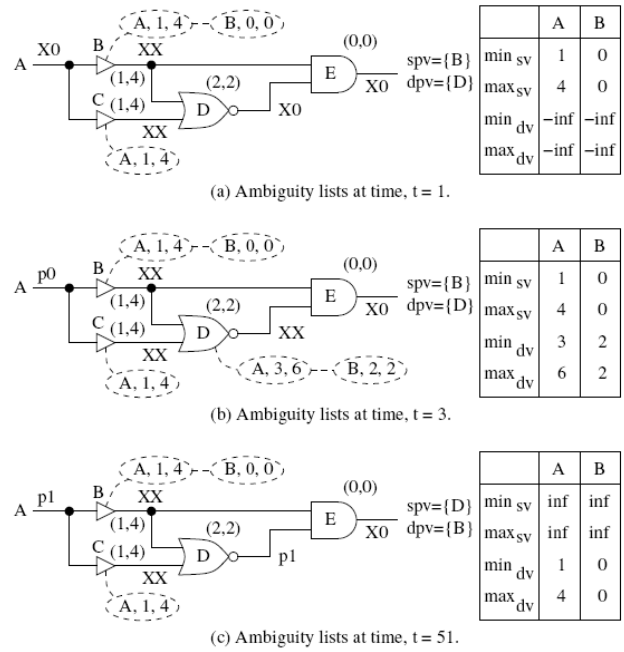


Figure 6: Illustration of hazard suppression.

- *EA(G):* Earliest arrival time for signal at *G* in the second vector.
- *LS(G):* Latest stabilization time for *G* in the second vector.
- *IV(G):* Final logic value of *G* after first vector (before second vector is applied).
- *FV(G):* Final logic value of *G* after second vector is applied.

For a slow-to-rise (slow-to-fall) delay fault at a gate output, the corresponding stuck-at fault is the stuck-at-0 (stuck-at-1) fault at the same fault site. For a gate *G*, we denote a fault-propagating value, *FPV(G)*, that is identical to the logic value of *G* when the second pattern of the vector pair is applied to the circuit with the corresponding stuck-at fault at *G*. At the fault site *g*,

$$FPV(g) = IV(g)$$

For a gate, *G*, outside the cone of influence of fault site *g*,

$$FPV(G) = FV(G)$$

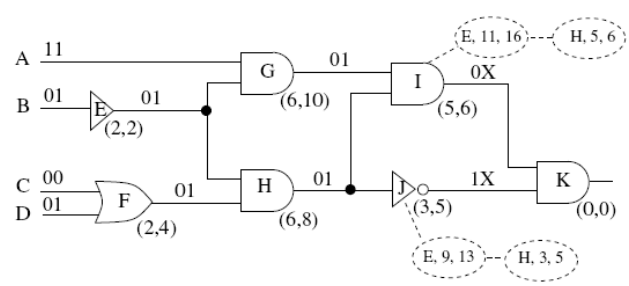


Figure 7: Ambiguity lists at t = 13 for Fig 2.

For a gate G inside the logic cone of the fault site g, $FPV(G)$ is evaluated using FPV values at its inputs and the usual rules of Boolean logic [11].

For a given delay fault f, a reference fault size, $\rho(G)$, and two reference times, $RTa(G)$ and $RTb(G)$, are evaluated for each gate G. Certain assertions are made about the signal waveform at G. The logic value at G is guaranteed to be at $FPV(G)$ between the time interval $RTa(G)$ to $RTb(G)+\rho(G)$, provided the size of the delay fault f exceeds $\rho(G)$.

4.2 Threshold Evaluation

For gates that lie outside the forward logic cone of the gate delay fault, the three reference quantities are evaluated identically as in the original paper [11]:

$$\rho(G) = 0$$
$$RTa(G) = LS(G)$$
$$RTb(G) = +\infty$$

At the fault site,

$$\rho(G) = 0$$
$$RTa(G) = -\infty$$
$$RTb(G) = EA(G)$$

If the fault site happens to be a fanout stem, a delay ambiguity list is also created at the fault site, as explained next. Each gate G has an ambiguity list, each element of which consists of the following:

1. Gate ids of fanout points that lie in the forward cone of the delay fault site.
2. Minimum delay from above fanout to G.
3. Maximum delay from above fanout to G.

Unlike the ambiguity lists evaluated during logic simlation in Section 3, these lists are evaluated for each gate in the forward cone of the fault site. As in Section 3, the following functions are defined for these lists:

- **HZ(G):** fanouts in ambiguity list at G.
- **d(G,f):** minimum delay to G from $f \in HZ(G)$.
- **D(G,f):** maximum delay to G from $f \in HZ(G)$.

We next show how the reference size $\rho(G)$, and the reference times, $RTa(G)$ and $RTb(G)$, are evaluated for gates in the forward logic cone of the fault site. When these quantities are evaluated for a gate G, the ambiguity lists at the inputs of G are used to evaluate the list at the output. If the output of gate G is a fanout stem, then an entry for gate G is added to the ambiguity list. There are two cases that are considered separately below:

Case 1: Assume G has no inputs that have a dominant FPV value. Define $In(G)$ as the minimum delay at some input of G that overcomes the inertia of G so as to observe the delay at the gate output:

$$In(G) = max\{0, max\{RTa(i)\}+D(G)-min\{RTb(i)\}\}$$

where i is any input of G. With the above definition of $In(G)$, the reference quantities are evaluated as follows:

$$\rho(G) = max\{max\{\rho(i)\},\ In(G)\}$$
$$RTa(G) = max\{RTa(i)\} + D(G)$$
$$RTb(G) = min\{RTb(i)\} + d(G)$$

The ambibuity list at the output of G is also updated. For every fanout stem f in the ambiguity list at any input, i, of G, a new entry is created at the output of G:

$$d(G, f) = min\{d(i, f)\} + d(G)$$
$$D(G, f) = max\{D(i, f)\} + D(G)$$

Case 2: Now assume G has inputs with both dominant and sensitizing values FPV values. Let M be an input of G that has a dominant FPV value, while m be an input with a sensitizing FPV value. For all fanout points that are elements of ambiguity lists at inputs with both FPV values, the following quantities are evaluated:

$$min_{DV}(f) = max\{d(M, f)\}$$
$$max_{SV}(f) = max\{D(m, f)\}$$

There are two cases possible:

- If $min_{DV}(f) \geq max_{SV}(f)$, the output signal at G is stable, so hazard cancelation occurs. The following reference values are used at the output:

  $$\rho(G) = 0$$
  $$RTa(G) = -\infty$$
  $$RTb(G) = \infty$$

  The ambiguity list at the output of G is set to the empty (NULL) list

- Assume no f exists with $min_{DV}(f) \geq max_{SV}(f)$. We choose input i with dominant FPV such that $\rho(i)$ is minimum among all such inputs of G.

  $$\rho(G) = max\{\rho(i), RTa(i) + D(G) - RTb(i)\}$$
  $$RTa(G) = RTa(i) + D(G)$$
  $$RTb(G) = RTb(i) + d(G)$$

The ambiguity list at G is evaluated as in case 1. For every fanout stem f in the list at any input, i, of G, a new entry is created at the output of G:

$$d(G, f) = min\{d(i, f)\} + d(G)$$
$$D(G, f) = max\{D(i, f)\} + D(G)$$

Figure 8 shows the ambiguity lists and the three reference quantities for the example of Figure 4 at time t = 3, when gate F is about to be evaluated. Gate C was evaluated at time t = 1, and gate E was evaluated at time t = 2. Evaluating the three reference quantities and updating the ambiguity list for gate F falls under case 2 above, and the following quantities are evaluated:

$$min_{DV}(f) = 1$$
$$max_{SV}(f) = 0$$

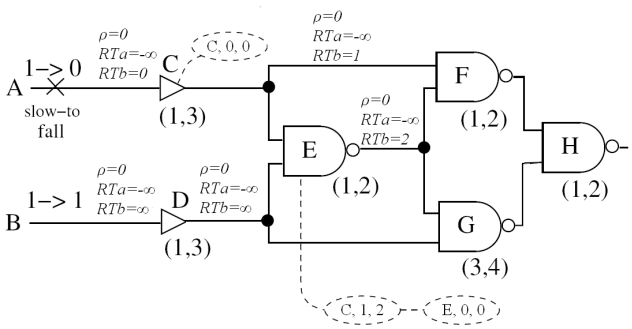


Figure 8: Ambiguity lists at time t = 3.

Since $min_{DV}(f) \geq max_{SV}(f)$, the hazard is suppressed, and the output of gate *F* is stable. The ambiguity list at *F* is set to null, $\rho(G)=0$, $RTa(G)=-\infty$, and $RTb(G)=\infty$.

Figure 9 shows the results obtained when this algorithm is used for the examples shown in both Figures 3 and 4. This figure shows *FPV* and reference times, along with the ambiguity lists evaluated for all signals in the circuits. For the example of Figure 3, the output sampling time was assumed to be t = 10. The correct detection threshold (=10-*RTb(F)*) is evaluated as 5, while the original algorithm in Figure 3 evaluates to 6. For Figure 4, assuming an output sampling period of 12, the erroneous detection threshold is 8 (=12-*RTb(H)*). This threshold is now correctly evaluated as 6.

signal	FPV	ρ	RTa	RTb	Ambiguity list
A	1	0	$-\infty$	∞	$\emptyset$
B	1	0	$-\infty$	0	(B,0,0)
C	1	0	$-\infty$	1	(B,1,4)
D	0	0	$-\infty$	3	(B,3,6),(H,2,2)
E	0	0	$-\infty$	∞	$\emptyset$
F	0	0	$-\infty$	5	(B,5,7)
G	0	0	$-\infty$	4	(B,4,6)
H	1	0	$-\infty$	1	(B,1,4),(H,0,0)

(a) Values for Figure 3

signal	FPV	ρ	RTa	RTb	Ambiguity list
A	1	0	$-\infty$	0	$\emptyset$
B	1	0	$-\infty$	∞	$\emptyset$
C	1	0	$-\infty$	1	(C,0,0)
D	1	0	$-\infty$	∞	$\emptyset$
E	0	0	$-\infty$	2	(C,1,2),(E,0,0)
F	1	0	$-\infty$	∞	$\emptyset$
G	1	0	$-\infty$	5	(C,4,6),(E,3,4)
H	0	0	$-\infty$	6	(C,5,8), (E,4,6)

(a) Values for Figure 4

Figure 9: Modified gate delay threshold.

5. Results

The value of the ambiguous delay model is demonstrated by two sets of experimental data presented in this section. The first experiment shows the error for path delay fault simulation using a non-robust zero-delay model. Bounded delay simulation eliminates many faults that cannot be detected, assuming gate delays that lie within specified bounds. The second experiment shows the error in small delay defect simulation when circuit reconvergences are ignored.

Path delay fault simulation results for both ISCAS combinational and scan versions of the ISCAS sequential benchmarks are shown in Table 1. Column 2 shows the total number of path delay faults for each of these circuits. Column 3 shows the length of the test sequence in thousands of clock cycles. These test sequences were generated randomly, with lengths chosen to achieve reasonable coverage in reasonable simulation time. Column 4 shows the non-robust (NR) fault coverage attained using a zero-delay model, as in conventional path delay simulation. Column 5 shows fault coverage obtained with the bounded min-max (MM) delay simulation algorithm of Section 3. The data in columns 4 and 5 clearly show a drop in coverage when the bounded delay model is used. Columns 6 and 7 show the effect of hazards on the longest sensitized path. Column 6 shows the earliest signal arrival time at the destination of the longest sensitized path when the zero-delay model is used. This is evaluated by adding the minimum delays of gates that lie on the longest sensitized path. Column 7 is obtained directly from the waveform of the signal at the destination of the longest sensitized path. The delay shown in column 7 can be used to evaluate the *guaranteed failure frequency* of the circuit [4, 18]. Columns 8, 9, and 10 show the number of events required for the simulation of these circuits. The number of events required for zero-delay simulation is a fraction of that required for any multiple-delay timing simulation. For bounded delay simulation, columns 9 and 10 show the number of logic and list events respectively. Columns 11 and 12 show the runtimes for the two types of path delay simulation. The data structures used in the path delay simulator were very simple and can be improved a lot using some of the recent work published in the literature. A simple path numbering scheme with a hash table to store all previously detected faults was used for the sake of simplicity. The goal of our experiment was to demonstrate the extent of the error when hazards and realistic delays are ignored. Efficient implementation of path fault dictionaries was not the primary goal in generating this data.

Table 2 presents results for gate delay faults for the same ISCAS circuits using vectors identical to those used in the previous experiment. Column 2 shows the number of gate delay faults in these circuits. Both the slow-to-rise and slow-to-fall transitions at gate outputs have been counted. For transition faults at the inputs of gates, circuit models need to be slightly changed by inserting buffers at gate inputs, and considering output faults for these new buffers. This is a commonly used assumption in the literature [11]. Column 3 shows the fault coverage achieved for these

Table 1: Path delay fault simulation results.

circuit (1)	path faults (2)	test len. $\times 10^3$ (3)	faults tested % NR (4)	faults tested % MM (5)	longest path NR (6)	longest path MM (7)	total events$\times 10^6$ NR (8)	total events$\times 10^6$ MM-logic (9)	total events$\times 10^6$ MM-list (10)	CPU (s) NR (11)	CPU (s) MM (12)
c432	0.168×10^6	1000	40.3	12.3	54	54	75	295	374	81	738
c499	0.018×10^6	20	96.9	71.2	45	36	2.37	12	53	7.5	211
c880	0.017×10^6	20	55.9	38.6	84	75	3.45	13.3	15.9	3.3	30.3
c1355	8.35×10^6	200	46.3	16.5	97	97	69.3	235	1390	140	1448
c1908	1.458×10^6	20	40.6	13.8	146	120	7.3	26	73	23.3	421
c2670	1.356×10^6	20	11.8	9.4	115	108	10.1	38	79	10.4	322
c3540	57.35×10^6	200	6.14	1.8	186	169	172	621	1020	192	2150
c5315	2.68×10^6	200	24.9	8.1	164	139	211	738	1302	213	2359
c7552	1.452×10^6	200	30.2	10.4	157	144	298	1040	2502	312	3998
s298	464	200	88.4	80.8	32	30	7.3	31	14.5	7.1	47.8
s344	716	200	95.5	90.2	70	63	11.3	39.9	29.4	10.1	60.4
s349	726	200	94.2	86.9	66	66	10.7	38.1	29.8	9.8	59.3
s382	802	200	94.4	91.0	33	33	9.5	40.5	21.3	10.4	55.8
s386	416	200	100	100	36	36	6.7	35.5	18.6	9.1	50.7
s400	898	200	92.8	82.5	33	33	9.9	41.9	23.2	10.4	58.9
s420	950	200	76	60.1	47	42	8.95	36.9	12.2	8.8	41.0
s444	1072	200	82.7	74.3	42	42	10.8	45.5	26.8	11.5	66.2
s510	740	200	100	100	42	42	10.9	45.6	32.8	11.2	74.4
s641	3446	200	68.2	60.3	247	213	21.6	75.4	44.5	19.4	96.9
s1196	6198	200	70.5	57.2	84	72	21.9	99.2	64.0	25.1	181
s1488	1926	200	99.7	99.2	63	54	31.1	145	97.1	36.5	312
s5378	27048	200	65.1	64.2	89	86	122	478	821	112	1548
s9234	489710	200	9.6	7.3	214	201	248	845	1100	336	2245

circuits. For gate delay faults, coverages are usually very high the slow-to-transition at the gate output is always detected with an appropriate threshold. For the coverage data in column 3, the magnitude of fault detection has been ignored. Any fault at the output of a gate that transitions is assumed to be detectable, provided its magnitude is large enough. Columns 4 and 5 consider detected faults that lie on paths with length at least 70% of the longest path in the circuit. These faults are detected with a magnitude of at most 30% of the longest path in the circuit. Column 4 (labeled "IRW") presents this data for the algorithm by Iyengar *et al.* [11], while column 5 (labeled "MM") presents the same data using the algorithm outlined in this paper. For this data, about half the critical gate delay faults can be erroneously assumed to be detected if signal reconvergences are ignored.

6. Comparison to Previous Work

A signal may have multiple ambiguity regions before attaining its final steady value. While the work presented in [5] finds the leading edge of the first ambiguity region, this work would find the minimum delay that would shift the last ambiguity region so as to guarantee detection. This is best illustrated by the waveform at *G4* in Figure 10. Figure 10 shows the waveforms for simulation of benchmark circuit c17. All primary inputs are assumed to transition at time t = 0, and the (min, max) delays for all gates are (3, 4). Assuming *G4* is observable, the hazard between t=8 and t=9 may be sufficient for detection. This hazard is largely ignored by the work presented in [5].

7. Conclusion

As shown in Tables 1 and 2, fault coverages obtained from algorithms that fail to analyze signal reconvergences are often 20-30% higher than realistic coverages where gate delays are assumed to vary due to process variations. Similar to concurrent simulation of stuck faults, the algorithm presented in this paper maintains lists at each node of the circuit. Unlike traditional concurrent fault simulation, these lists maintain information about delay bounds from fanout points. While this work presents a step in the right direction, the efficiency of the simulation algorithm can be improved further, and will be the subject of a forth coming paper.

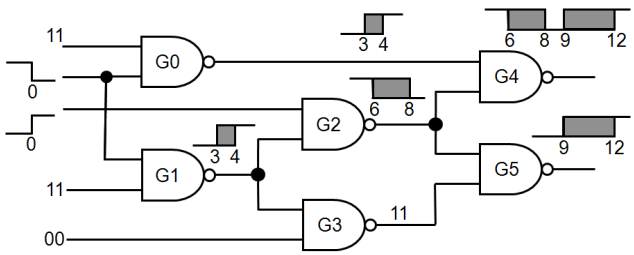


Figure 10: Simulation of benchmark circuit c17.

Table 2: Gate delay fault simulation results.

circuit (1)	gate faults (2)	coverage (%) (3)	70 % coverage IRW (4)	70 % coverage MM (5)
c432	420	97.1	42.6	42.6
c499	564	97.9	96.9	71.2
c880	952	98.7	35.6	15.6
c1355	1252	99.0	59.2	54.2
c1908	1890	98.9	35.9	26.1
c2670	3298	98.9	26.1	21.4
c3540	3496	70.7	16.7	8.1
c5315	5230	99.8	12.4	8.2
c7552	7666	99.2	12.0	9.0
s298	328	96.3	16.6	12.9
s344	440	96.8	13.7	12.3
s349	438	97.3	21.1	12.4
s382	434	97.2	35.7	18.5
s386	386	96.9	58.3	27.1
s400	444	96.8	35.8	18.1
s420	544	92.0	21.7	5.4
s444	480	97.5	33.1	23.0
s510	514	97.7	17.2	5.5
s641	966	98.1	21.8	10.2
s1196	1200	99.0	22.5	9.9
s1488	1400	99.1	21.6	5.4
s5378	6428	99.1	12.85	4.5
s9234	12200	89.7	42.2	36.0

8. References

[1] A. Agarwal, D. Blaauw, and V. Zolotov, "Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations," in *Proc. International Conf on Computer Aided Design*, November 2003, pp. 900-907.

[2] S. Bhardwaj, S. Vrudhula, and D. Blaauw, "Probability Distribution of Signal Arrival Times Using Bayesian Networks," in *IEEE Trans. Computer Aided Design*, volume 24, No 11, November 2005, pp. 1784-1794.

[3] J. W. Bierbauer, J. A. Eiseman, F. A. Fazal, and J. J. Kulikowski, "System Simulation with MIDAS," *AT&T Tech. J.*, vol. 70, no. 1, pp. 36-51, January 1991.

[4] S. Bose, P. Agrawal, and V. D. Agrawal, "Delay Fault Testability Evaluation Through Timing Simulation," in *Proc. Third Great Lakes Symposium on VLSI*, March 1993, pp. 18-21.

[5] S. Bose and V. D. Agrawal, "Delay Test Quality Evaluation using Bounded Gate Delays," in *Proc. 25th IEEE VLSI Test Symp.*, May 2007, pp. 23-28.

[6] S. Chakraborty, D. L. Dill, and K. Y. Yun, "Min-Max Timing Analysis and an Application to Asynchronous Circuits," *Proc. IEEE*, vol. 87, no. 2, pp. 332-346, Feb. 1999.

[7] S. Devadas and K. Kuetzer, "Validatable Nonrobust Delay Fault Testable Circuits Via Logic Synthesis," *IEEE Trans. CAD*, vol. 11, no. 12, pp. 1559-1573, December 1992.

[8] F. Fink, K. Fuchs, and M. H. Schulz, "Dynamite: An efficient automatic test pattern generation system for path delay faults," in *IEEE Transactions on Computer Aided Design*, volume 10, Oct 1991, pp. 1323-1335.

[9] N. Ishiura, Y. Deguchi, and S. Yajima, "Coded Time Symbolic Simulation Using Shared Binary Decision Diagrams," in *Proc. Design Automation Conference*, June 1990, pp. 130-135.

[10] N. Ishiura, M. Takahashi, and S. Yajima, "Time Symbolic Simulation for Accurate Timing Verification," in *Proc. Design Automation Conf*, June 1989, pp. 497-502.

[11] V. S. Iyengar, B. K. Rosen, and J. A. Waicukauski, "On Computing the Sizes of Detected Delay Faults," *IEEE Trans. CAD*, vol. 9, pp. 299-312, Mar. 1990.

[12] H. F. Jyu, S. Malik, S. Devadas, and K. W. Kuetzer, "Statistical Timing Analysis of Combinational Logic Circuits," in *IEEE Transactions on Very Large Scale Integration Systems*, volume 1, No 2, June 1993, pp. 126-137.

[13] H. Konuk, "On Invalidation Mechanisms for Non-Robust Delay Tests," in *Proc. International Test Conference*, Oct 2000, pp. 393-399.

[14] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," in *IEEE Trans. Computer Aided Design*, volume 6, No 5, September 1987, pp. 694-703.

[15] M. Linderman and M. Leeser, "Simulation of Digital Circuits in the Presence of Uncertainty," in *Proc. International Conf on Computer Aided Design*, November 1994, pp. 248-251.

[16] J. J. Liou, A. Krstic, L. C. Wang, and K. T. Cheng, "False Path Aware Statistical Timing Analysis and Efficient Path Selection for Delay Testing and Timing Validation," in *Proc. Design Automation Conf*, June 2002, pp. 566-569.

[17] S. Padmanaban and S. Tragoudas, "An Implicit Path Delay Fault Diagnosis Methodology," in *IEEE Transactions on Computer Aided Design*, volume 22, No 10, October 2003, pp. 1399-1408.

[18] Q. Peng, V. D. Agrawal, and J. Savir, "On the Guaranteed Failing and Working Frequencies in Path Delay Fault Analysis," in *Proc. 16th IEEE*

Instrumentation and Mesurement Technology, March 1999, pp. 1794-1799.

[19] A. K. Pramanick and S. M. Reddy, "On the Fault Coverage of Gate Delay Fault Detecting Tests," *IEEE Trans. Computer-Aided Design*, vol. 16, no. 1, pp. 78-94, Jan. 1997.

[20] S. M. Reddy, C. J. Lin, and S. Patil, "An Automatic Test Pattern Generator for Path Delay Faults," in *Proc. International Conference on Computer Aided Design*, November 1987, pp. 284-287.

[21] G. L. Smith, "Model for Delay Faults Based Upon Paths," in *Proc. International Test Conference*, Oct 1985, pp. 342-349.

[22] E. G. Ulrich, K. P. Lentz, S. Demba, and R. Razdan, "Concurrent Min-Max Simulation," in *Proc. Design Automation Conference*, June 1991, pp. 554-557.

[23] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First Order Incremental Block Based Statistical Timing Analysis," in *Proc. Design Automation Conf*, June 2004, pp. 331-336.

[24] K. Yang, L. C. Wang, K. T. Cheng, and S. Kundu, "On Statistical Correlation Based Path Selection for Timing Validation," in *Proc. VLSI Design, Automation and Test*, April 2005, pp. 8-11.

[25] L. Zhang, W. Chen, Y. Hu, and C. C. Chen, "Statistical Static Timing Analysis With Conditional Linear MAX/MIN Approximation and Extended Canonical Timing Model," in *IEEE Trans. Computer Aided Design*, volume 25, No 6, June 2006, pp. 1182-1191.