Delay Test Quality Evaluation Using Bounded Gate Delays

Soumitra Bose
Design Technology, Intel Corp.
Folsom, CA 95630

Vishwani D. Agrawal
Dept. of ECE, Auburn University
Auburn, AL 36849

Abstract: *Conventionally, path delay tests are derived in a delay-independent manner, which causes most faults to be robustly untestable. Many non-robust tests are invalidated by hazards caused primarily due to non-zero delays of off-path circuit elements. Thus, non-robust tests are of limited value when process variations change gate delays. We propose a bounded gate delay model for test quality evaluation and give a novel simulation algorithm that is less pessimistic than previous approaches. The key idea is that certain time-correlations among the multiple transitions at the inputs of a gate cannot cause hazard at its output. We maintain "ambiguity lists" for gates. These are propagated with events, similar to fault lists in a traditional concurrent fault simulation. They are used to suppress erroneous unknown states. Experimental results for ISCAS benchmarks with gate delay variation of* $\pm 14\%$ *show a miscorrelation of critical path delay as much as* 20%.

1 Introduction

Delay test [13, 16, 24] is the most widely accepted method of verifying that a manufactured design meets its timing constraints. Robust tests have the highest quality but few faults are robustly testable [10, 16]. 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 circuit to circuit. Only some non-robust tests are validatable [8, 23] given that certain other faults are robustly tested. Although the shortcomings of non-robust and validatable non-robust tests are known [15], attempts to enhance them have been limited [19].

Increasing process variation in sub-micron devices has prompted research in various directions, including Monte-Carlo techniques [14, 18, 27], bounded delay simulation [4, 11, 12, 17, 25], statistical timing analysis [2, 3, 26, 28] and the classical bounded delay timing analysis [7]. Timing analysis finds corners in the design process in a vector independent manner by using specified gate delays. This makes it unsuitable in a test environment, where test vectors must expose delays that exceed their expected values [13, 21]. Bounded delay simulation holds promise but there is no universally accepted correct algorithm.

We present a new algorithm for bounded delay simulation that is less pessimistic than previous approaches [13]. Thus, a more realistic coverage of delay defects is determined. Also, the simulator

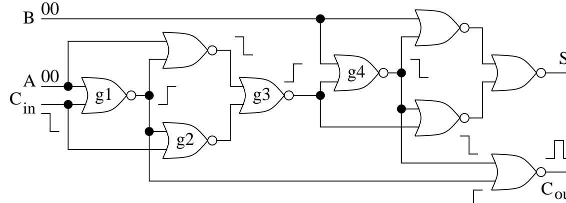


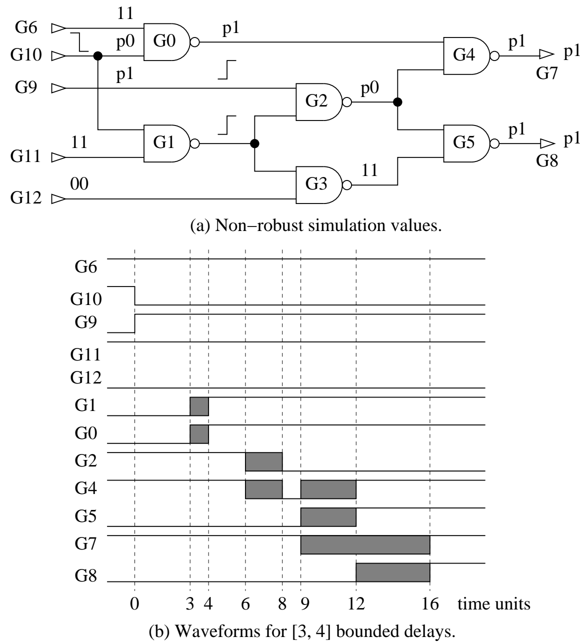
Figure 1: Non-robust sensitization in full adder.

allows the selection of tests for delay defect coverage. For the first time, delay faults can be simulated with an algorithm that allows imprecise gate delays due to variations in the manufacturing process. Unlike robust delay testing, more faults are found testable. Unlike non-robust delay tests that assume zero delay for all off-path gates, the tests found by this algorithm are not invalidated by other circuit delays. Section 4 shows that nominal path delays of the traditional non-robustly tested faults can have a miscorrelation of up to 20% when delays are imprecisely known. Thus, larger number of tests would be necessary to ensure quality.

2 Motivation

Figure 1 shows a full adder, used extensively in the c6288 multiplier circuit of ISCAS85 combinational benchmarks [22]. A falling transition at the carry-in input, C_{in}, is non-robustly sensitized to the carry-out output, C_{out}, along the path C_{in}, g_1, C_{out}. Notice that the off-path sensitizing input, g_4, at the last gate, C_{out}, has a falling transition that is guaranteed to arrive later than the on-path transition at g_1 due to non-zero delays of gates $g2$, $g3$ and $g4$. A zero-delay non-robust simulator fails to recognize this temporal relationship and incorrectly marks this path fault as tested. The algorithm of this paper will eliminate the static hazard C_{out} and may, in turn, affect the detectability of other faults in the circuit.

Figure 2(a) shows two-vector simulation of ISCAS85 circuit $c17$. Signal value $p0$ ($p1$) means that a path has been non-robustly sensitized to the signal and that the final steady state value is 0 (1). The two longest paths sensitized non-robustly are $\{G10, G1, G2, G4, G7\}$ and $\{G10, G1, G2, G5, G8\}$. Figure 2(b) shows the waveforms obtained when all gates are assumed to have delay bounds in the range (min, max) = (3, 4) units. All primary inputs transitions are assumed to occur simultaneously at time 0, while gate outputs transition through an unknown



(a) Non−robust simulation values.

(b) Waveforms for [3, 4] bounded delays.

Figure 2: Unreliable failures in c17. All gates have (min, max) delays (3, 4).

(X) state. The waveform at $G7$ has a X value (shown in grey) from $t = 9$ to $t = 16$ units, and $G8$ has a X value from $t = 12$ to $t = 16$. If the outputs are sampled when they have indeterminate (X) values, failure detection is unreliable. Output $G7$ is guaranteed to fail if sampled before $t = 9$, while $t = 12$ is the guaranteed failing period for $G12$. Even though both $G7$ and $G8$ are destinations of paths with identical nominal delays, sampling times for guaranteed failure at the two outputs are sharply different.

Figure 3 [7, 11] illustrates the pessimism in most bounded delay simulation algorithms [13]. The (min, max) delay bounds are as shown. Ignoring the notation, which will be explained later, the shaded regions in the waveforms are time intervals when logic values are unknown (X). The unknown value at output E at time $t = 3$ is incorrect because due to time correlation, signal at B must fall before D rises. A correct algorithm for timing analysis is presented by Chakraborty *et al.* [7], though its complexity would prevent its use in logic simulation. Previous applications of min-max simulation to delay test [20] have been ad-hoc and lack any systematic algorithm.

3 Simulation Algorithm

Normally, an event-driven simulator will generate a hazard at the output of a two-input AND gate whenever the inputs have simultaneous rising and falling transitions and the time intervals of transitions overlap. However, notice that if the falling transition occurs before the rising transition then the gate cannot have a hazard. This is possible when the two transitions come from the same fanout stem traveling through paths with different inversion parity and

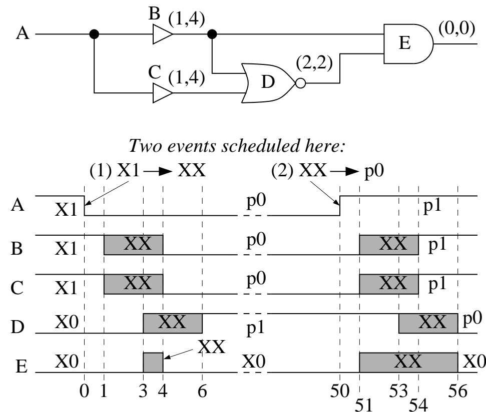


Figure 3: Illustrating pessimistic hazard simulation.

arrive at the reconvergence gate with correlated delays that guarantee a specific time succession. In our new simulation algorithm, described in this section, we attach an ancestral identity to every fanout node event. An ambiguity list entry for the transition identifies the fanout node and the time window in which the node produced the transition. As this transition progresses through the downstream logic, all fanout stems encountered are added to the list that is also propagated with the event. At a reconvergence gate receiving multiple transitions, the information in their lists allows us to determine how correlated transitions should interact and whether or not the gate output should have a hazard.

3.1 Data Structures

Bounds on gate delays are specified by minimum and maximum values, assumed to be 0 for all input pins. All signals transition through an intermediate ambiguous (unknown) value, X, followed by a final Boolean value, 0 or 1. Transitions to X are scheduled with the minimum delay of the gate, while transitions to Boolean values, 0 or 1, are scheduled with the maximum gate delay. Signal transitions at primary inputs are modeled by injecting two back to back events with zero delay in the signal scheduler: (1) transition to X, and (2) transition to the final Boolean value [1]. The simulation state of gate G is given by:

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

Both **pv** and **cv** components can assume values from 0, 1, and X. After the occurrence of an event, the new logic value is assigned to **cv**, while the old **cv** value is assigned to **pv**.

The fault activation status, **fas**, can be either p or X. It is set to p when a path delay fault has been activated from some primary input to that gate, or set to X otherwise. Since all transitions pass through an intermediate unknown state, the **pv** component for a gate is always X if the **fas** component is p.

All signals are initialized by making **cv** and **pv** components equal to the corresponding Boolean values and setting **fas** components to X. The value of **cv** component determines the delay to be used for scheduling an event. An exception to this rule occurs when **fas** component at a gate has to be changed without any accompanying change in the current logic value **cv**. For example, consider that an input of an AND gate has a sensitized falling transition, while all other off-path inputs have non-controlling values. If an off-path input now transitions to X, then only **fas** component at the output will change. Such changes in **fas** component need to be propagated to the downstream logic with zero delay, and a *pseudo-event* is generated to make all fanouts active. Pseudo-events are identified by the signal scheduler from the absence of an associated change in **cv**.

In min-max delay simulation, Boolean **cv** values on off-path inputs imply that all ambiguities on those inputs have been resolved. Despite resemblance to the non-robust path sensitization criteria, these conditions are more stringent because of the intermediate X values associated with 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 $\langle \mathbf{pv}, \mathbf{fas}, \mathbf{cv} \rangle = \langle X, p, 0 \rangle$. In the rest of this paper, this state is abbreviated as $p0$. Similarly, $p1$ denotes the state $\langle \mathbf{pv}, \mathbf{fas}, \mathbf{cv} \rangle = \langle X, p, 1 \rangle$.

For the example of Figure 3 [7, 11], only the **fas** and **cv** components are shown for each signal. Thus, $X0$ indicates a **fas** value of X and a **cv** of 0. The (min, max) delays used for each gate are also shown in the figure. As stated earlier, the ambiguity in signal E at time $t = 3$ is erroneous and an algorithm for correct simulation is presented next.

3.2 Algorithm

General definitions for a gate G are:

- **dpv(G):** inputs of G that have dominating (controlling) **pv** values.
- **spv(G):** inputs of G that have sensitizing (non-controlling) **pv** values.

Gate G has an ambiguity list of elements consisting of:

1. Source gates of fanout points in the input cone of G 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. Several functions process ambiguity lists:

- **HZ(G):** fanout points 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 $f \in HZ(G)$.

When an event arrives at the input of gate G, the procedure of Section 3.1 determines the output. If the output value has a Boolean **cv** component (0 or 1), the ambiguity list at the gate output is set to *null*. If the **cv** component evaluates to X, the ambiguity lists at the inputs of G are used to evaluate the list at the output. The algorithm of Figure 4, may suppress the ambiguity at the output. If an event cancellation does not occur, new 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 4 iterates over all fanout points in the ambiguity list at any input of 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 4. 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 gate output is canceled, and the function returns a *null* list. If no event cancellation occurs, an entry for the fanout is added to the list (line 12).

The list evaluated by the algorithm of Figure 4 is used to update the ambiguity list of gate G after the minimum delay of the gate. This is achieved by letting the signal scheduler handle an additional type of events, the *ambiguity list update events*. These events merely add new elements to the ambiguity list of a gate. They are similar to the *fault list events* in a concurrent fault simulator [1, 6].

Figure 5 shows the simulation algorithm in its entirety. List events are processed in line 1. Logic events are first checked for their values. If the new value is X at a fanout stem (line 2), a new entry is added to the

```
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
}
```

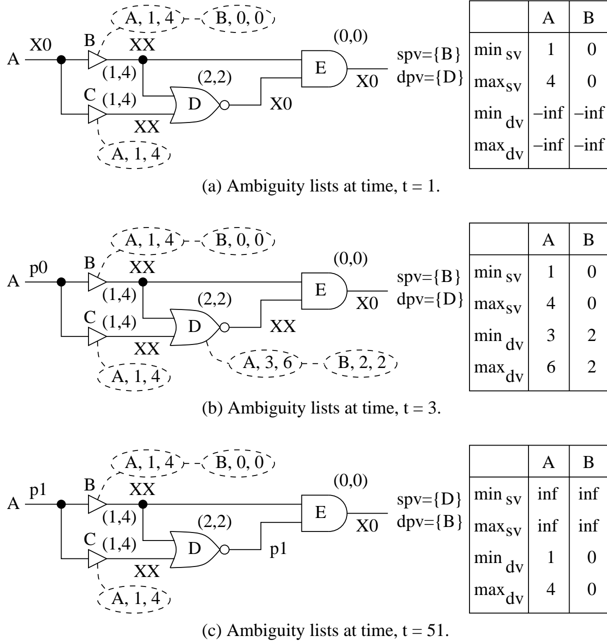
Figure 4: Ambiguity list evaluation at gate output.

```
while (more vectors to simulate) {
  inject new vector
  while (more events to simulate) {
    if (ambiguity_list_event) {                              (1)
      new_ambiguity_event=(f, d, D) at gate G
      add (f, d, D) to HZ(G)
    } else if (logic_event) {
      let new_logic_event=(G, nv, vfas)
      if (nv = X) and G a fanout stem                        (2)
        add (G, 0, 0) to HZ(G)
      else if nv is Boolean HZ(G)=∅                          (3)
      if (cv(G)≠nv)                                          (4)
        update pv(G), cv(G)
      update fas(G)                                          (5)
      for each fanout branch fb of G {                       (6)
        (fb_v, fb_fas, fb_d) = EvalGate(fb, G)
        (suppress, HL)=EvalAmbiguityList(fb)                 (7)
        if (suppress=1) continue                             (8)
        if (HL ≠ NULL)
          schedule e ∈ HL at fb with delay fb_d              (9)
        schedule (fb, fb_fas, fb_v) with delay fb_d          (10)
      }
    }
  }
}
```

Figure 5: Min-max simulation.

ambiguity list of the stem. If the new value is Boolean (line 3), the ambiguity list at the gate is set to *null*. As mentioned in Section 3.1, events that update **fas** value only are *pseudo-events* that propagate with zero delay. For such events, the **pv** and **cv** components are left unchanged (line 4). The **fas** component is always updated (line 5). After the values at the event site are updated, each fanout is processed (line 6). Function *EvalGate* returns three items: (1) a new value fb_v, (2) a new **fas** value fb_{fas}, and (3) a gate delay value, f_d, used for scheduling fanout fb. Furthermore, function *EvalAmbiguityList* returns two items: (1) a hazard suppression status and (2) an ambiguity list similar to the one evaluated by the function in Figure 4. After evaluation of the new value at the fanout, the ambiguity list at the fanout is obtained (necessary only if $fb_v = X$). If a hazard is evaluated at the output of the fanout branch and needs to be suppressed, then the event evaluated by *EvalGate* is skipped and processing continues with the next fanout branch (line 8). If the ambiguity list is non-empty, each element of the list is added to the signal scheduler (line 9) with the delay evaluated by the function *EvalGate*, fb_d. This delay equals the minimum delay of the fanout branch fb. Once all ambiguity list update events have been scheduled, the logic event at the fanout gate is scheduled with an identical delay (line 10).

Figure 6 illustrates this algorithm for the example of Figure 3. 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). Note that lines 2 and 3 of Figure 5 cause an ambiguity list to appear temporarily at primary input A at simulation time $t = 0$. This has the effect of adding entries for fanout stem A in the ambiguity lists at B and C, due to lines 7 and 9 of the algorithm of Figure 5. Since B is itself a fanout stem, an entry for B is added to the ambiguity list at B (line 3 of Figure 5). Since the **cv** component at D is 0, the ambiguity list at D is *null*. 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 4 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



(a) Ambiguity lists at time, t = 1.

(b) Ambiguity lists at time, t = 3.

(c) Ambiguity lists at time, t = 51.

Figure 6: Illustration of hazard suppression.

ambiguity checking condition fails for fanout stem B and this hazard would be suppressed. At time $t = 51$, the ambiguity lists in Figure 6(c) indicate an empty list at D and the hazard at E is deemed to be valid.

4 Results

For ISCAS85 benchmarks, sets of random vectors were simulated with a conventional non-robust delay fault simulator. Next, simulations were performed using the algorithms presented in this paper. All gates were assumed to have a delay variation of $\pm 14\%$ around their nominal delay of 3.5 time units. This arbitrary choice means a bounded (min, max) gate delay of (3, 4). In general, the delay bounds may be obtained from the process, gate library and physical design data.

The cumulative nominal delay of paths sensitized in the first experiment was obtained by multiplying path length by the nominal delay of each gate. The maximum of such path delay, over all paths, was compared to the earliest time when the path output attained an unknown value. The latter is the earliest arrival time of on-path transition for some valid assignment of delays.

The results are presented in Table 1. Column 2 gives the total number, in millions, of path delay faults in these circuits. Column 3 shows the number, in thousands, of random vectors that were simulated. Column 4 shows the number of faults that were detected non-robustly by the conventional simulator, while columns 5 and 6 show the nominal and maximum delays of the longest path that was activated using our algorithm. Column 7 shows the time instant when the circuit settles to its steady state after the vector that sensitizes the longest nominal path has been applied. The waveforms at sensitized path destinations were monitored and the immediately preceding time instants when these outputs attain unknown values are reported in column 8. Note that for the circuit of Figure 2, these time values were 9 and 12 for outputs $G7$ and $G8$, respectively. These time values give the guaranteed failing frequency for the vector sequence [5, 20].

The numbers in column 8 are compared to nominal path delays (column 5), and column 9 gives the excess of their ratio above 1.0. This is the error in estimating the critical path delay, and hence the maximum frequency error, from the nominal behavior. When compared to minimum cumulative path delays, positive entries in column 8 are even higher, and denote path sensitization where the on-path transition arrives earlier than the off-path sensitizing values. Detection of such path delay faults is guaranteed only if the on-path delay exceeds a threshold value equal to the time interval between the arrival of on-path and off-path values. Negative values in column 8 denote either the invalidation of fault activation due to hazards or simultaneous activation of another path delay fault with a smaller delay. This situation is best illustrated by output $G7$ in Figure 2. Even though path {G10,G1,G2,G4,G7} is sensitized, a shorter path {G9,G2,G4,G7} determines the quality of the test at $G7$. As evident from column 9, errors in measuring the critical path delay can be as high as 20%.

Columns 10 and 11 give the numbers of events, in millions, processed by the simulator. List events in column 11 correspond to the hazard list manipulation (line 1 in Figure 5). The number of events can be high for bounded delay simulation and methods to speed up the simulation have been proposed [9]. However, since vector application and output sampling were not staggered temporally, the only heuristic used to speed up simulation was an inertial gate delay model. The inertial delay threshold used was equal to the difference between the minimum and maximum delay of gates. Column 12 shows the runtimes for a Pentium 4 Linux workstation.

5 Conclusion

The new bounded delay simulation algorithm eliminates many hazards, making the result less pessimistic and more realistic. Further analysis would reveal that the detected delay faults are of two types, i.e., those with monotonic detection sizes and those with multiple detectable sizes. Monotonic behavior guarantees that all delay faults at a gate above a threshold are detected by the same test. We believe this viewpoint overcomes several known difficulties with other fault models like transition and path delay faults. A comparison of our results with bounded delay static timing analysis [28] can be used for enhancing the tests and for identifying false paths. A fault simulation algorithm to find the delay thresholds for all gates above which delay fault detection is monotonic will be discussed in a future paper. Vector-specific yield evaluation is another application worth exploring.

Table 1: Comparison of nominal and earliest hazard arrival delays.

circuit	faults $\times 10^6$	vectors $\times 10^3$	non-rob. detect	longest active nom	max	last event	earliest ambig	fmax error (%)	events $\times 10^6$ logic	list	CPU s
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
c432	0.17	20	9995	63	72	72	66	+4.8	5.9	7.4	25.9
c880	0.02	20	6679	87.5	100	100	75	-14.3	13.3	15.9	53.4
c1355	8.34	20	498256	87.5	100	100	97	+10.6	25.6	138.5	845
c1908	1.46	20	24383	133	152	156	111	-16.5	26.5	73.2	458
c2670	1.36	20	41592	108.5	124	124	108	-0.46	36.2	78.5	426
c3540	57.3	20	355248	168	192	192	145	-13.7	62.5	330.3	2506
C5315	2.68	20	89167	168	192	192	135	-19.6	72.7	129.9	1262
C6288	2200	5	38×10^6	381.5	436	436	418	+9.6	63	375	3900
C7552	1.45	5	68886	147	168	168	136	-7.5	25.9	74.9	1671

References

[1] M. Abramovici, M. A.Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. IEEE Press, 1990.

[2] A. Agarwal, D. Blaauw, and V. Zolotov, "Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations," in *Proc. Int. Conf. CAD*, Nov. 2003, pp. 900–907.

[3] S. Bhardwaj, S. Vrudhula, and D. Blaauw, "Probability Distribution of Signal Arrival Times Using Bayesian Networks," *IEEE Trans. CAD*, vol. 24, no. 11, pp. 1784–1794, Nov. 2005.

[4] 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, Jan. 1991.

[5] S. Bose, P. Agrawal, and V. D. Agrawal, "Delay Fault Testability Evaluation Through Timing Simulation," in *Proc. Third Great Lakes Symp. VLSI*, Mar. 1993, pp. 18–21.

[6] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer, 2000.

[7] S. Chakraborty, D. L. Dill, and K. Y. Yun, "Min-Max Timing Analysis and an Application to Asynchronous Circuits," *Proc. IEEE*, vol. 87, pp. 332–346, 1999.

[8] S. Devadas and K. Kuetzer, "Validatable Nonrobust Delay Fault Testable Circuits Via Logic Synthesis," *IEEE Trans. CAD*, vol. 11, pp. 1559–1573, 1992.

[9] S. Devadas, K. Kuetzer, S. Malik, and A. Wang, "Event Suppression: Improving the Efficiency of Timing Simulation for Synchronous Digital Circuits," *IEEE Trans. CAD*, vol. 13, pp. 814–822, June 1994.

[10] F. Fink, K. Fuchs, and M. H. Schulz, "Dynamite: An efficient automatic test pattern generation system for path delay faults," *IEEE Trans. CAD*, vol. 10, pp. 1323–1335, Oct. 1991.

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

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

[13] 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.

[14] H. F. Jyu, S. Malik, S. Devadas, and K. W. Kuetzer, "Statistical Timing Analysis of Combinational Logic Circuits," *IEEE Trans. VLSI Syst.*, vol. 1, no. 2, pp. 126–137, June 1993.

[15] H. Konuk, "On Invalidation Mechanisms for Non-Robust Delay Tests," in *Proc. Int. Test Conf.*, Oct. 2000, pp. 393–399.

[16] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. CAD*, vol. 6, no. 5, pp. 694–703, Sept. 1987.

[17] M. Linderman and M. Leeser, "Simulation of Digital Circuits in the Presence of Uncertainty," in *Proc. Int. Conf. CAD*, Nov. 1994, pp. 248–251.

[18] J. J. Liou, A. Krstić, 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. Des. Auto. Conf.*, June 2002, pp. 566–569.

[19] S. Padmanaban and S. Tragoudas, "An Implicit Path Delay Fault Diagnosis Methodology," *IEEE Trans. CAD*, vol. 22, pp. 1399–1408, Oct. 2003.

[20] Q. Peng, V. D. Agrawal, and J. Savir, "On the Guaranteed Failing and Working Frequencies in Path Delay Fault Analysis," in *Proc. 16th IEEE Instrument. Meas. Tech.*, Mar. 1999, pp. 1794–1799.

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

[22] W. Qiu and D. M. H. Walker, "Testing the Path Delay Faults of ISCAS85 Circuit c6288," in *Proc. Int. Workshop Microproc. Test. Verif.*, 2003, pp. 19–24.

[23] S. M. Reddy, C. J. Lin, and S. Patil, "An Automatic Test Pattern Generator for Path Delay Faults," in *Proc. Int. Conf. CAD*, Nov. 1987, pp. 284–287.

[24] G. L. Smith, "Model for Delay Faults Based Upon Paths," in *Proc. Int. Test Conf.*, Oct. 1985, pp. 342–349.

[25] E. G. Ulrich, K. P. L. ans S. Demba, and R. Razdan, "Concurrent Min-Max Simulation," in *Proc. Design Automation Workshop*, June 1991, pp. 554–557.

[26] 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.

[27] 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.

[28] 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," *IEEE Trans. CAD*, vol. 25, no. 6, pp. 1182–1191, June 2006.