Delay Fault Diagnosis Using Timing Information

Zhiyuan Wang[1] Malgorzata Marek-Sadowska[1] Kun-Han Tsai[2] Janusz Rajski[2]

[1]Department of Electrical and Computer Engineering
University of California, Santa Barbara,CA,93106
{wzy,mms}@ece.ucsb.edu

[2]Mentor Graphics Corporation
Wilsonville, OR, 97070
{hans_tsai,janusz_rajski}@mentorg.com

Abstract

In modern technologies, process variations can be quite substantial, often causing design timing failures. It is essential that those errors be correctly and quickly diagnosed. Unfortunately, the resolution of the existing delay-fault diagnostic methodologies is still unsatisfactory. In this paper, we investigate the feasibility of using the circuit timing-information to guide the delay-fault diagnosis. We propose a novel and efficient diagnostic approach based on the timing window propagation (TWP) to achieve significantly better diagnostic results than those of an existing delay-fault diagnostic commercial tool. Besides locating the source of the timing errors, for each identified candidate our method determines the most probable delay defect size. The experimental results indicate that the new method diagnoses timing faults with very good resolution.

1. Introduction

Due to process-parameters variations, a circuit may fail to operate at the desired clock speed. A critical task for a failure analyzer is to locate quickly and accurately the cause of timing failures. The quality of failure analyzer is measured by their *resolution,* which is defined as a ratio of the number of true faults to the total number of reported candidates. Unfortunately, the existing delay fault-diagnostic methodologies suffer from very poor resolution. Industrial data suggest that on average, it may take about 240 hours to locate an open via defect by screening under the microscope the failure candidates reported by diagnostic tool. If too many candidates are reported by the diagnostic tool, the time-to-market requirement is hard to satisfy.

Delay fault diagnosis has been studied in the past. However, most of the existing works explicitly or implicitly assume that the delay-fault-model-based simulation results reflect exactly the delay-defect behavior in real silicon [4]-[7]. However, this assumption is generally not true. It is well known that different paths in a circuit may have different delays and slacks. A real delay defect can demonstrate itself only on those sensitized paths whose slacks are smaller than the delay-size of the defect. Other delay defects cannot be observed. The matching mechanism used in some of the existing techniques may produce wrong or inconclusive results due to mismatches between the delay fault simulation results and the real defect behavior.

The algorithms based on path-tracing alleviate this problem [6]. They back-trace the sensitized paths from each failing PO or a scan-cell. They work based on the assumption that the failure has been caused by a single fault. Therefore the real faulty site should be located in the intersection of the fanin cones of the observed failure POs or scan-cells. However, in practice, this method produces still too many candidates.

A more recent work advocates using statistical timing information to guide the delay defect diagnosis [8][9] which produces good diagnostic results. In this method it is assumed that the probability density functions (*pdf*'s) of each internal cell/interconnect are known. In reality, however, the accurate *pdf* information may not be easily available.

In [3] the authors describe an approach based on static timing information targeting the multiple delay-fault diagnosis. For each fault candidate, they try to use a robustly tested path and observe a fault-free situation to determine the upper and lower bounds for a suspect delay fault. The experimental results [3] show a much improved diagnostic resolution when compared to non-timing-based approaches. However, the resolution is still unsatisfactory for time-to-market requirements.

In this paper, we propose a delay defect diagnosis method based on the timing information extracted from the layout. We apply a simulation-based approach to diagnose the timing failure responses. For a given design, the circuit timing information is obtained from the SDF (standard delay format) file which contains the interconnect delay and gate pin-to-pin rising/falling delay. Our novel and efficient technique based on the *Timing Window Propagation* (*TWP*), diagnoses the failure responses caused by delay defects. Experimental results show that our method not only reports quite accurately the fault candidates (high resolution) but also suggests delay-defect sizes of the reported candidates.

The rest of the paper is organized as follows. In Section 2 we introduce the preliminary concepts and explain briefly the delay-fault diagnosis algorithm used in the

0-7695-2093-6/04 $20.00 © 2004 IEEE

existing commercial tool. In Section 3, we describe our diagnostic algorithm. In Section 4, we discuss some practical issues and the feasibility of our method. In Section 5, for industrial and ISCAS circuits, we report resolution, and performance. Section 6 concludes the paper.

2. Preliminary

In this paper we assume that a single fault is present in the circuit and it is modeled as a transition fault [1] (slow to rise, slow to fall, slow). Due to its simplicity, this model has been widely used in the majority of the delay fault testing and diagnosis works [12].

Since we will be referring to a commercial, delay-fault diagnostic tool, for the sake of completeness, we summarize this algorithm here. We call it Alg_{no_timing}. The Alg_{no_timing} is also based on the single-transition fault-model.

We say that the fault simulation result *covers* the failure response when the faulty effect can propagate to all the observed failure outputs. It is possible that the fault simulation may cause more primary outputs (PO) to fail.

Alg_{no_timing}:

1. Initialize the fault candidate list using the path-tracing technique. The initial fault candidates satisfy the following requirements which reduce the search space and improve diagnostic efficiency:

1.1 The fault must reside in the input cone of a failing PO of the given pattern.

1.2 If a failing pattern affects more than one PO, the candidate fault must reside in the intersection of all the input cones of the failing POs (single delay fault per pattern assumption).

2. Simulate each transition fault on the initial candidate list to see if the simulation results *cover* the observed failure responses. If they do, store the fault as a candidate and assign to it a weight equal to the number of patterns it explains in the current list.

3. After explaining the entire failing-pattern list, or when all faults in the initial list have been examined, the algorithm terminates and reports the possible candidate sites. Candidate faults are sorted by their weights. The fault with the greatest weight is reported first.

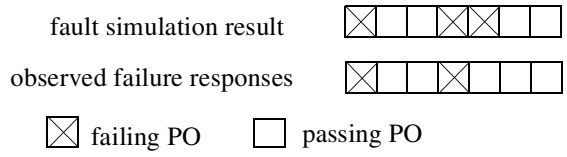


Fig. 1: Example of *cover*

Because of the *cover* relationship between the observed and simulated results, and based on the single-fault assumption, we conclude that if indeed a real single-delay fault has caused the timing failure, then that fault must be on the reported candidate list. However, the diagnostic results on real designs produce a large number of candidates. The purpose of this work is to improve further the diagnostic resolution and to prune the unlikely candidates.

3. Our Approach

In this work, we investigate the feasibility and efficiency of using the delay and timing information for delay fault diagnosis.

We obtain the circuit delay and timing information from the SDF files (refer to IEEE DASC P1497 SDF Standard [13] for more details about the file format). SDF file contains the internal gate timing/delay and the interconnect delay. For each pin-to-pin and interconnect, SDF provides the rising/falling transition delay if the transition occurs.

3.1 Timing-based Simulator

We have developed a framework which utilizes the delay information to emulate the real chip and tester timing behavior during at-speed testing. This framework allows us to evaluate our diagnostic algorithm. We do not include the implementation details here due to the page limit. Instead we give a simple example to show how we emulate the real circuit timing/delay for given test sets. In this example, we use a pair of delay values to represent the pin-to-pin rising and falling delays. All the examples discussed in this paper are based on this simplification. However, in our framework these values are replaced by pairs of delay ranges which span from the fastest rising (falling) delay to the slowest rising (falling) delay.

Consider the example in figure 2. The numbers shown beside each gate's inputs are the corresponding rising/falling delays from the input to the output of that gate. For example, if there is a transition on the output of the gate g_2 caused by the signal *A* rising transition, the delay from *A* to g_2 output *E* is 5. Here we ignore all the interconnect delays to simplify our explanation. For different patterns (structural or functional tests), the circuit delays and long(est) paths may vary.

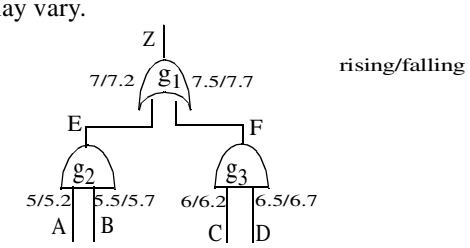


Fig. 2: Timing based simulation example

For the pattern *P1*: {*ABCD*}: {*r 1 r 1*} (*r* = rising), there are 2 rising transitions on *E* and *F*. Because the logic value *"1"* is controlling for the OR gate g_1, the circuit delay will be decided by the earliest transition, which occurs on *E*. So even though there is a longer delay on *F*, its effect cannot be propagated further. The longest path delay in the good circuit for this pattern is 5+7=12. The path which contributes to the circuit delay is from *A* to *Z*.

For the pattern *P2*: {*ABCD*}: {*f 1 f f*} (*f* =falling) there are 2 falling transitions on *E* and *F*, because *0* is a control-

ling value for the *AND* gates g_2 and *g3*. The circuit delay will be decided by the latest transition, which occurs on *F*. Note that *F*'s delay is defined by *C* and not by *D*, whose delay is longer. The longest path delay in a good circuit mode for this pattern is 6.2+7.7=13.9. The path which determines the circuit delay is from C to Z.

3.2 Timing Window Propagation (*TWP*)

In this section, we describe our novel simulation technique, *the timing window propagation* (*TWP*). The purpose of this technique is to determine, for a given fault candidate, its capability of explaining the failing and passing pattern responses.

We define the *timing window (TW)* of a fault candidate as a delay range *[a,b]* whose lower bound is the smallest possible delay size and the upper bound is the biggest possible delay size.

We define a conventional interval-arithmetic on the timing windows. Assuming $T_1 = [t_a,t_b]$ and $T_2 = [t_x,t_y]$, we have:

$$T_1 + T_2 = [t_a + t_x,\ t_b + t_y]$$
$$T_1 - T_2 = [t_a - t_x,\ t_b - t_y]$$
$$T_1 \cap T_2 = [max(t_a, t_x),\ min(t_b, t_y)]$$
$$T_1 \cup T_2 = [min(t_a, t_x),\ max(t_b, t_y)]$$

For a given pattern *P* and a fault candidate *f* under evaluation, we begin the timing-fault simulation from the faulty location *f* by assigning there an initial timing window *(TW) [0,T]*. The lower bound of this window is *0*, which means that the minimum delay for *f* is *0*. The initial upper bound for *f* is the operating clock period T. For each candidate, those windows will be propagated (and possibly shrunk) along the sensitized paths.

The examples in figure 3 show how the timing windows are updated as they propagate through an *AND* gate. For other gate types and combinations of rising and falling signals, the rules are similar. If the *f*'s faulty effect propagates to a signal line, the corresponding *TW* can also propagate to this line. The symbol *R* (*F*) stands for rising(falling) transition on the signal line.

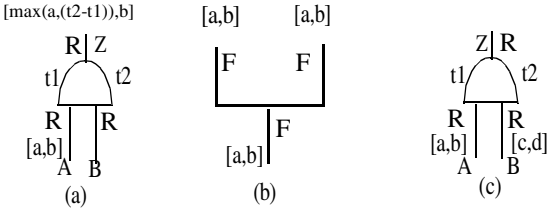


Fig. 3: Timing Window Propagation Examples

The example in Figure 3(a) shows how the lower bound of a *TW* is updated. Suppose the faulty effect propagates to *A* and the *TW* on *A* is *[a,b]*, and the current simulating pattern produces rising transitions on both *A* and *B*. To propagate the delay effect from *A* to *Z*, the delay value on *A* and *B* must satisfy the condition $t_1 + a > t_2$, where *t1* and *t2* are the fault-free delay values calculated as described in Section 3.1. In Figure 3 examples, we assume that the pin-to-pin rising and falling delay values are the same for *A-Z* and *B-Z*. The lower bound of the *TW* must be equal to or greater than $max(a, (t_2-t_1))$. The new *TW* at *Z* becomes $[max(a, (t_2-t_1)),b]$.

For fanout branches as shown in Fig. 3(b), the *TW*s simply propagate to each branch from the stem.

In Fig. 3(c) we have the fanout reconvergence situation. The faulty effect propagated to both inputs from different fanout branches of the failure's source. The timing window at *A* is $TW_A = [a,b]$ and at *B* is $TW_B = [c,d]$. Suppose that the *AND* gate's inputs have rising transitions. We first derive the delay value range at Z before deriving the TW_Z range at *Z*. In this example, the delay range at *A* is $D_A = [t_1+a, t_1+b]$ and the delay range at *B* is $D_B = [t_2+c, t_2+d]$. The delay range at *Z* is the $[max(t_1+a,t_2+c), max(t_1+b, t_2+d)]$.

Next we determine the TW_Z. We need to consider two cases separately.

Case 1: $D_A \cap D_B = \Phi$ which implies that the delay on one of the inputs dominates the delay at *Z* regardless the delay changes at the other input. In this case, we simply propagate the *TW* from the input which has a bigger delay to *Z*.

Case 2: $D_A \cap D_B \neq \Phi$ which implies that both inputs may contribute to the delay at *Z*. We will analyze the overlapped delay range only. The non-overlapped delay range is the same as the case *1* above. Here, we assume $D_A=D_B$.

Case 2.1 If $TW_A \cap TW_B \neq \Phi$ then $TW_Z = TW_A \cup TW_B$, which means that a defect of any size in the timing window *A* or *B* could produce a failure at *Z*. We merge these two timing windows with no loss of information.

Case 2.2 If $TW_A \cap TW_B = \Phi$ then TW_A and TW_B are disjoint. It is possible to merge these two *TW*s to form a bigger timing window. But this will degrade the resolution. This is so because some delay values which are neither in TW_A nor in TW_B could be included within the new timing window. To overcome this problem, we can save all the information about TW_A and TW_B and propagate them. However, the more the disjoint timing windows we keep, the more the performance of the simulation degrades. In our implementation we store up to four disjoint timing windows on each signal. If the number of disjoint timing windows exceeds four, we merge the closest windows. Experimental results show that this heuristic achieves good resolution and performance trade-off.

We perform the timing window propagation applying all test patterns for the given fault candidate. If the timing window can propagate to any primary output under a given pattern *P*, we use a tuple with four components, $\{f, P, PO_i, TW_i\}$ to record this information. It records that the candidate *f*'s timing window TW_i can propagate to the primary output PO_i under the pattern *P*.

So far we have discussed only how the lower bound of a timing window is updated. Now we explain how to update the upper bound using the passing point information.

Suppose that for a given pattern *P* we cannot observe a delay failure on the output PO_i at the capture time. If, however, we recorded a tuple at PO_i under *P*, the upper bound of TW_i must be smaller than $T - D_{PO_i}$, where *T* is the operating clock and D_{PO_i} is the delay value at PO_i calculated from the fault-free circuit simulation.

In Section 3.4 we will explain how to extract a useful information from those tuples and further prune the unlikely candidates.

3.3 The Diagnostic Algorithm

We propose a novel algorithm to diagnose the delay defects using the timing information and delay simulation. Our algorithm is based on the assumption that each failing pattern attributes its response to a single fault location. However, our algorithm is capable of identifying multiple fault locations as long as each failing pattern is affected by one fault only. We make another assumption that, if two candidates have the same explanation capability for a set of failing and passing patterns, and we have the delay size information for each candidate, the smaller delay size candidate has a higher probability of being the real defect. Our experimental results and the results in [8] support this assumption.

The algorithm proceeds as follows:

1. Back-tracing diagnosis: The set of initial fault candidates, $F_{initial}$, is determined by using the algorithm $\boldsymbol{Alg_{no\_timing}}$.

2. Timing Window Propagation (*TWP*) and Pruning: For each candidate in the $F_{initial}$ we apply the *TWP* technique, simulate the patterns, and record the tuples. We prune the unlikely candidates checking the rules (which will be described in detail in the Section 3.4). The candidate set after pruning is denoted as F_{after_TWP}.

3. Refinement: If a group of candidates in the F_{after_TWP} has the same explanation capability for a set of failing and passing patterns, we use the tuple information obtained from the observation points to deduce the most likely delay size of each candidate in this group. Then the candidates are ranked by their delay sizes. Candidates with smaller sizes are reported earlier.

3.4 Pruning Rules

Here we introduce the rules for pruning the impossible candidates. The rules are derived based on the timing windows relations at the observation points.

Rule 1: For two tuples *{f, P, PO_1, TW_1}* and *{f, P, PO_2, TW_2}*, if at-speed failure responses are observed on two primary outputs PO_1 and PO_2, but $TW_1 \cap TW_2 = \Phi$, then *f* is not a candidate. In this case the *timing windows* have a conflict as shown in figure 4.

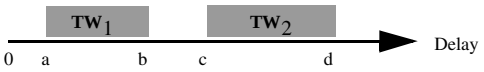


Fig. 4: Timing Window Conflicts

This rule eliminates those cases when there is no consistent delay value for *f* which would propagate its effect to both failing *POs*.

Rule 2: We calculate slacks along the paths propagating the failure *f* to PO_1 and to PO_2. If the timing failure is only observed on PO_1 but not on PO_2, and the slack s_1 to PO_1 is bigger than s_2, the slack to PO_2, then f is not the candidate. Figure 5 illustrates this situation.

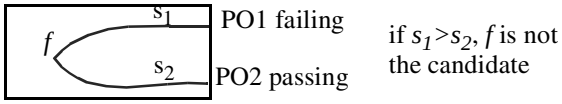


Fig. 5: Example of Rule 2.

Rule 3: For a tuple *{f, P, PO_1, TW_1}*, if we observe a failure on PO_1, but the summation of the upper bound of TW_1 and the path delay value calculated on PO_1 is less than the clock period T, *f* is not the candidate. Figure 6 shows this situation where *t1* is the fault-free delay value calculated as described in section 3.1.

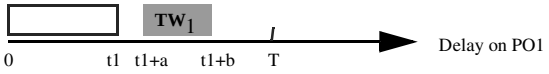


Fig. 6: Rule 3.

Rule 4: If we do not observe failure on PO_1, but the summation of the lower bound of TW_1 and the delay value calculated on PO_1 is greater than the clock period *T*, *f* is not the candidate for *P*.

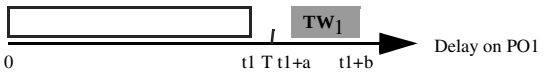


Fig. 7: Rule 4.

For each candidate and every failing/passing pattern, after applying the above checking rules, we prune many unlikely candidates. In our implementation, instead of deleting a candidate which fails the rules, we assign a penalty for each failed rule. After simulating all the fault candidates, we rank them based on their penalty scores. The better-matched candidates have a higher rank.

3.5 Refinement

In the refinement step, we consider globally all the failing and passing patterns which can be explained by the fault candidates. For each candidate fault, we combine the tuples for different observation points and different patterns, and then we determine the timing window distribution of each candidate.

For each fault we collect *TW*s and construct the delay distribution graph, *wave(t)*. The *wave(t)* is built such that for each delay *t*, we assign a value equal to the number of

TWs which cover *t*. For each fault, the delay value/range which has the highest weight is the greatest possible delay size for this fault. We measure the waveforms of the candidates by their integrals from 0 to *t*, which yield the total area *Area(t)* covered by a *wave(t)*:

$Area(t) = \int_0^t wave(t))dt$. We define the percentage function as $Percentage(t) = Area(t)/Area(T)$. For each candidate we determine three points - t_{lb}, t_{mid}, and t_{ub} - by setting the *percentage(t)* to be 0.3, 0.5, 0.7 and solving the percentage function.

A fault candidate which has a clustered distribution graph is more meaningful than the candidate which has a sparse distribution graph. The clustered distribution demonstrates that the candidate's delay size is within small range as long as it can be activated. This can be quantified by the density function $density(f) = [Area(t_{ub})-Area(t_{lb})] / (t_{ub} - t_{lb})$. A higher density indicates that the candidate's delay value is more clustered at a smaller delay range. In the example in figure 8, when all waveforms cover the same area, the candidate which has the delay distribution (a) is better than the candidate which has the delay distribution (b).

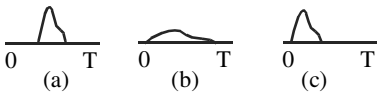


Fig. 8: delay distribution graph example

After obtaining all the distribution graphs for all the faults, it may happen that two faults have about the same density. Based on our assumption stated in Section 3.2, the fault which has a smaller t_{mid} is better than the fault which has a bigger t_{mid}. In figure 8, if the three candidates have the same capability to explain all the failing and passing pattern responses (and their waveforms cover the same area), we can use the density function and t_{mid} to rank them. In this example, the ranks of three candidates are (c) > (a) > (b).

The complete diagnosis flow is shown in figure 9.

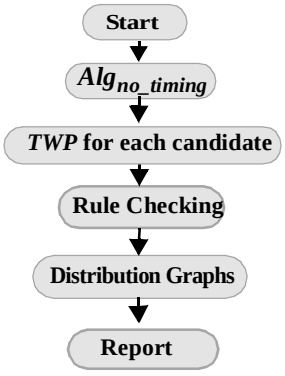
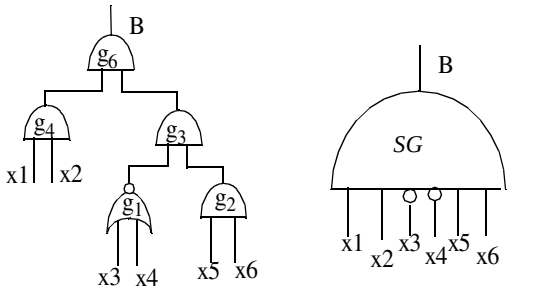


Fig. 9: Diagnosis flow.

4. Practical issues

4.1 Storing the timing information

Since the timing information may require a huge space, storing it efficiently is an issue. It is common to find in a circuit the networks of gates which logically correspond to a large AND/OR gate. Such structures are referred to as *super gates* and can be efficiently detected [10]. We propose using super gates to reduce the storage requirements. The delay information can be stored for super gates instead of for individual gates.



a) Before transformation b) After transformation

Fig. 10: SG transformation

We will use the example in Fig. 10 to explain how to store the delay information using the supergates (*SGs*). Suppose we obtained the timing information (rising/falling, interconnect) for each gate and interconnect in fig. 10(a). Because the extracted super gate is fanout-free, the delay value of any path from X_i to B (either rising or falling transition on that path) can be lumped to the boundary of the *SG*. All the information we need to store are the lumped delays and the inversion status on the boundary of *SG*.

The experimental data on ISCAS circuits show that the supergate circuit transformation yields a factor of 5-7 storage space reduction.

4.2 Propagating *TWPs*

To achieve higher performance and memory-efficiency, we use the *PPSFP* (Parallel Pattern Single Fault Propagation) technique to simulate 32 patterns at a time. We apply the event-driven simulation technique to improve the memory efficiency.

5. Experimental Results

We evaluated the diagnostic capabilities of our algorithm using the delay fault simulation framework. Since obtaining accurate timing information has not been the main purpose of this work, we used the static timing/delay information as a substitute for accurate timing information obtained from either SPICE simulation or SDF files.

In our experiments, we used ISCAS'85 and full-scan versions of ISCAS'89 benchmark circuits bigger than 1K gates. We injected randomly into the circuits, random-size delay-faults (slow-to-rise, slow-to-fall, or slow), performed delay-fault simulation, and collected the failure responses.

The failure responses were captured when the path delay values on the observation points were bigger than the system operating clock. The 100%-fault-test-coverage transition-fault test-pattern set was generated by a commercial tool. We fed those failure responses into an existing commercial delay-fault diagnostic tool. The candidates reported by the commercial tool served as the initial fault candidates for our algorithm. The experimental results show that our algorithm can significantly improve the diagnosis resolution. Also, when we use the refinement step, the real fault sites have much higher rank than other candidates.

In Table 1, we list the pertinent data for the circuits used in our experiments. The first and second columns list the circuit's name and size. The third column shows the size of the transition fault tests sets.

Table 1: Circuit Information

Circuit	*# Gates*	*# Pat.*	*Circuit*	*# Gates*	*# Pat.*
C5315	2631	186	C6288	2480	80
C7552	3883	287	s9234	6325	364
s13207	10139	549	s38417	27320	301
s35932	20492	78	s38584	24083	360

Our diagnostic algorithm ranks the identified faults so that the most probable faults (based on the matching and refinement steps) are reported first. It is important that the rank of the first fault which matches the injected fault on the ordered list is as low as possible. The position of the first true fault is often referred to as the *first hit rank* (*FHR*). A low *FHR* can save time on chip screening for a failure analyzer.

The results in Table 2 have been obtained by averaging 100 random-injected test-cases for each benchmark circuit. Since the first hit rank is not reported by the commercial tool, we list only the first hit rank of our algorithm. We also list the number of candidate sites reported by the commercial tool and our algorithm. The reported sites are representative faults of equivalence classes after transition fault collapsing.

Table 2: Experimental Results for ISCAS Circuits

Circuit	*Initial # Faults*	*Reported # Faults*	*FHR*	*Runtime (sec)*
C5315	31.3	6.9	4.4	3.75
C6288	33.2	11.2	4.6	1.27
C7552	47.1	6.5	2.2	5.66
s9234	33.2	5.7	3.6	10.33
s13207	26.3	5.9	2.5	23.09
s38417	35.8	5.3	2.7	32.98
s35932	30.6	5.7	3.1	10.23
s38584	28.6	5.4	2.3	38.47
Average	***33.2***	***6.58***	***3.1***	***15.72***

Our algorithm significantly increases the diagnostic resolution and produces very low first hit ranks.

The performance of our algorithm is proportional to the circuit size. The runtime depends on the number of initial fault candidates and on the number of test patterns used for diagnosisis.

6. Conclusion and Future Work

In this work, we have investigated the feasibility of using timing/delay information to guide the delay defect diagnosis. Our algorithm improves significantly the diagnostic resolution.

In the future we will use the timing information to study the timing failures caused by multiple delay defects and distributed delay defects.

Acknowledgement

This work was supported by the California MICRO Program and Mentor Graphics Corporation.

References

[1] Z. Barzilai and B. Rosen, "Comparison of ac self-testing procedures", *Proc. Intl. Test Conf,* 1983. pp. 89–94.

[2] K.-T. Cheng, H.C. Chen, "Delay testing for nonrobust untestable circuits", *Proc. Intl. Test Conf,* 1993. pp. 954-961

[3] J.G. Dastidar, N.A. Touba, "A systematic approach for diagnosing multiple delay faults", *Proc.* IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 1998. pp. 211-216.

[4] J.G. Dastidar, N.A. Touba, "Adaptive techniques for improving delay fault diagnosis", *Proc*. VLSI Test Symposium,1999, pp. 168-172

[5] P. Girard, C. Landrault, S. Pravossoudovitch, "An alternative to fault simulation for delay-fault diagnosis", *Proc.* European Conference on Design Automation, 1992. pp. 274 -279.

[6] P. Girard, C. Landrault, S. Pravossoudovitch, "Delay-fault diagnosis by critical-path tracing", IEEE Design & Test of Computers, 1992, pp.27-32

[7] Y.-C. Hsu, S.K. Gupta," A new path-oriented effect-cause methodology to diagnose delay failures", *Proc. Intl. Test Conf,* 1998, pp. 758-767.

[8] A. Krstic, L.-C. Wang, K.-T. Cheng, J.-J. Liou, T.M. Mak, "Enhancing diagnosis resolution for delay defects based upon statistical timing and statistical fault models", *Proc.* Design Automation Conference, 2003. pp. 668-673.

[9] A. Krstic, L.-C. Wang, K.-T. Cheng, J.-J. Liou,M. S. Abadir,"Delay Defect Diagnosis Based Upon Statistical Timing Models – The First Step", *Proc.* Design Automation and Test in Europe, 2003.

[10]K.-H. Tsai, J. Rajski, M. Marek-Sadowska, "Star test: the theory and its applications", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 19 Issue: 9, 2000,pp. 1052 -1064.

[11]N. Tendolkar, R. Raina, R. Woltenberg, X. Lin, B. Swanson, G. Aldrich, "Novel techniques for achieving high at-speed transition fault test coverage for Motorola's microprocessors based on PowerPC/spl trade/ instruction set architecture", *Proc.* VLSI Test Symposium, 2002. pp.3 -8

[12]"IEEE DASC Standard Delay Format (SDF)", http://www.eda.org/sdf/