Diagnosis of Hold Time Defects

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. In this work, we analyze failures caused by the hold-time-violations. We investigate the feasibility of using circuit-timing information to guide the hold-time-fault diagnosis. We propose a novel and efficient diagnostic approach based on timing window propagation. For each identified candidate, our method locates the source of the hold-time violation and determines the most probable defect size. Experimental results indicate that the new method diagnoses hold-time related defects with very good resolution.

1. Introduction and Motivation

Sequential designs have to fulfill strict timing requirements. The *set-up time* requirement states that it is necessary for logic values at data inputs of the flip-flops to be stable before arrival of the active clock edge. The *hold time* requirement states that the data inputs should remain stable for a sufficient period of time after the active clock edge. The set-up time violation problem which can be modeled by delay-type defects has been extensively studied. In this paper, we study the hold-time defects and propose a diagnostic approach for them.

Hold time failures [7] may be caused by process variations, design tool limitations, crosstalk-induced speedup, IR-drop, clock skew or short paths. In modern technologies the polysilicon gate lengths have been scaled down below the wavelength of the light used in optical lithography processes. This results in systematic within-die fluctuations that can affect performance and functionality [3].

Among all those factors, two are the most likely to cause the hold-time violation. The first is the presence of short sensitizable paths (logic or scan) between flip-flops in the circuit. What constitutes a short path depends on the clock skew, path delay, and data hold time. Sometimes short paths can be eliminated by delay padding. Although delay padding could theoretically solve the short path problem, in practice it is an over-design increasing the chip area, power, and design time without improving performance [12].

The second typical cause of hold-time failure is clock skew between the flip-flops. Due to process variations and the inaccuracy of design tools, clock skew in a real chip may be very different from the designated skew value [1][4][5]. In [5], clock skew sources are classified into four categories: systematic offsets, random offsets, jitter, and drift. Systematic offsets are the skews caused by using the nominal-component SPICE-simulation results. Random offsets are due to intra-die process variations such as channel length and gate width variations. Drift is caused by temperature variation which results in low-frequency skew changes. Jitter, especially from the voltage noise, may cause high frequency timing variations. Most of the *zero skew* clock papers address only the systematic offsets. In [5], the authors mention jitter as the most challenging source of skew in modern high-performance micro-processors. Research results [5][9] have shown that PLL jitter and the uncertainties in the clock distribution network can cause difficulties in achieving accurate skew budgets. The data reported from IBM and Compaq show skew for the clock grids in the range of 50-70ps [1][12].

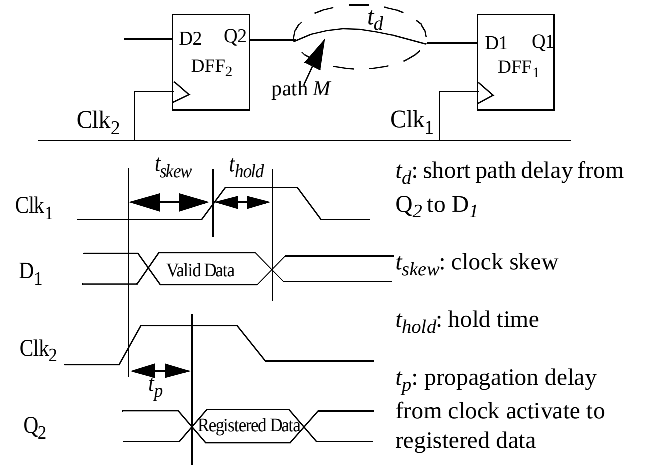


Fig. 1: Timing diagram

Figure 1 shows a timing diagram of a digital circuit. Let t_{hold} be the hold time for a flip-flop DFF$_1$, let t_{skew} be a clock skew between DFF$_1$ and DFF$_2$, and let t_p be the propagation delay from the Clk$_2$ activation to the DFF$_2$ registering data. Suppose that between DFF$_2$ and DFF$_1$ there is a sensitizable path M whose delay is t_d, and the following condition is satisfied:

$$t_d - t_{skew} + t_p < t_{hold} \qquad \text{Eq. (1)}$$

In such a case a hold-time failure may occur and wrong logic value may be registered by DFF_1.

All the papers on hold-time fault diagnosis [2][3][6]-[8][10][13][16] address this type of violation on short scan paths. Those approaches make explicit use of the scan chain properties, trying to localize a possible faulty scan cell in as narrow a range as possible. Those methods cannot be easily extended to diagnosing hold-time violations on the functional parts of the circuit.

Diagnosis of the hold time faults that occur on data paths has not yet been addressed. Here, we diagnose the hold-time defects occurring in circuits whose timing dependencies fulfill Eq.(1). We focus on two main causes of hold-time violation: the reduced t_d, and the increased t_{skew}.

In the design phase, timing is estimated by static timing analysis (STA) based on parameters extracted from layout. To account for variability, the slowdown and setup times are computed using the worst-case library. The best-case library is used to evaluate the speedup or hold-time. It is possible that a circuit which passes STA may malfunction after it is manufactured. The purpose of this work is to develop a methodology to locate the probable cause of hold-time failures in manufactured faulty chips using failure responses observed from tester.

We propose a hold-time defect diagnostic method based on the timing information extracted from layout. We collect the observed failure responses and process them with the algorithm. We first determine whether the failure might possibly be caused by a hold-time fault. If so, our approach back-traces sensitizable paths from the failing outputs to find the initial candidate sites. Then we inject each candidate fault into the circuit, using the hold-time fault model, and perform timing-based fault-simulation. Our novel technique, the *Negative Timing Window Propagation* (*NTWP*), is very efficient. It not only reports the fault candidate-sites accurately but also suggests their probable causes, such as speedup on the short paths or increased clock skew. It also reports the amount of speed-up (skew) introduced by the reported candidates.

The rest of the paper is organized as follows. In Section 2, we introduce the preliminary concepts and analyze the hold-time fault behavior. In Section 3, we describe our diagnostic algorithm. In Section 4, we discuss the feasibility of our method, and its extensions to multiple hold-time faults. In Section 5, we report experimental results. Section 6 concludes the paper.

2. Preliminary

A good machine is a circuit with no defects. Logic values on the primary outputs (POs), or internal wires, of a good machine are the *good machine values* (*GMV*) for the corresponding test pattern *T*. For a simplified explanation, POs represent all the primary outputs and scan cells.

When multiple faults exist in the circuit, a test pattern may activate several faults and create multiple-fault behaviors.

Each failing pattern *p* in the given diagnostic test set *T* is classified into one of the following two types:

Type-1 failing pattern: *p* can activate only one fault and observe its effect. Other faults cannot be activated or their faulty effect cannot be observed.

Type-2 failing pattern: *p* can activate multiple faults and observe their faulty effects.

2.1 Hold Time Fault Analysis

To model the hold-time-violation defects, we use the hold time fault models similar to those proposed in [11].

Fault_Model_1: All the flip-flops, except the source flip-flop of a target path, receive the clock-activating edge within the bounds of timing constraints imposed on the skew. The source flip-flop receives the clock-edge earlier.

Fault_Model_2: All the flip-flops, except the sink flip-flop of a target path, receive the clock activating edge within the timing constraints imposed on the skew, whereas the sink flip-flop receives the clock-edge later.

These two models do not cover all the complex hold-time defect behaviors. They divide all the flip-flops into two groups. But in reality, due to complex relationships between the activation times on different flip-flops, these two models may not capture all the possible hold-time defects. Instead, they capture the most frequently occurring hold-time violations.

To activate and observe the failure responses caused by hold-time faults, we need to satisfy three conditions. Here, hold-time fault is defined as the situation that the t_{hold} of a flip-flop DFF_1 (in figure 1) fulfills the timing relationship stated by equation (1). A timing failure that might occur is that DFF_1 registers incorrect data.

Activation Condition: For a *fault_model_1*, a transition between the current and the next time frame must occur at the output of the source flip-flop of a path (Q_2). This is the necessary condition to activate the hold time fault [7]. For a *fault_model_2*, the transition must occur at the input of the sink flip-flop of a path (D1). This transition may be caused by multiple flip-flop sources.

Sensitization Condition occurs when the activation transition (at the source) is propagated to a sink flip-flop. This condition should be satisfied before clock capture for *fault_model_1*, and after clock capture for *fault_model_2*.

Timing Condition is $t_{hold} + t_{skew} - t_p > t_d$. There are four terms in this inequality and any of them may change due to the signal or design integrity problems. Path delay t_d might be reduced by crosstalk, whereas t_{skew}, t_{hold} might increase due to the process-parameter variations.

Those conditions form the necessary and sufficient conditions for hold-time failure to be activated and observed.

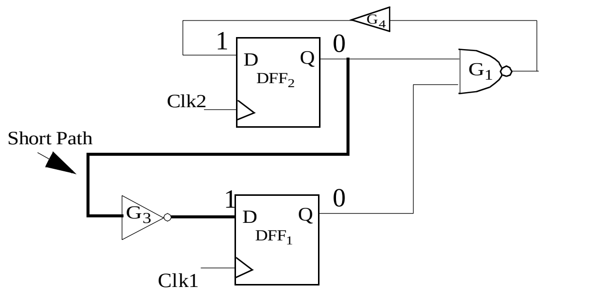
The hold time faults do not have to be detected by at-speed tests. A single cycle test using a slower clocking frequency (like stuck-at test) is sufficient to detect them. However, to diagnose a hold-time fault, timing information is required because of *timing conditions*. In practice, running the test at different frequencies can exacerbate the power consumption such that it can affect the test result for hold-time fault. Similarly, applying the test at different voltages can result in non-obvious results (lower voltages sometimes cause the hold-time failure to disappear).

Although we use a single clock design as an example illustrating those three conditions, the hold-time violation can also occur in multi-clock and wave-pipelined designs. The analysis in this work is also applicable to those situations.

In this study, we assume that a single fault is present in the circuit and that it is modeled by one of the two introduced fault models. We also assume that the hold-time fault can manifest itself only on the data paths. If a chip fails on scan paths during the loading or unloading phases, we can apply the methods in [3][7] to identify the faulty locations efficiently before we perform the logic test.

2.2 Hold-time Fault Injection

Our diagnostic method is simulation-based. To determine



Pattern P:

1. Scan-in both flip-flops with logic value 0
2. Pulse clock
3. Scan-out value from both flip-flop both flip-flops which have logic value 1.

Fig. 2: Example of Hold-Time Fault Injection

whether a candidate site could be the source of a failure, we inject a hold-time fault and perform simulation. The hold-time fault injection is different from the injection methods for other fault models. For a *fault_model_1*, if there is a transition at the path source flip-flop's output (activation condition), we use the next time-frame good-machine value (*GMV*) to replace the current time-frame *GMV*. There are subtle differences between the hold-time fault and the delay fault, and their detection conditions. In a delay-fault test, the observation points must first be established with the *failing value* (typically by scanning it in). Then the sample will overwrite the value with a *passing* value. If the passing value does not arrive by the setup time, then the failing value remains, it is scanned out, and delay failure response is observed. In a hold-time test, the observation point must be initialized with a value that coincides with the correct value. If there is shoot-through of the value in the flip-flop previous to the launch flip-flop, then the test will fail. For this reason some delay tests may not be useful for hold-time detection because even if there is a hold-time fault in the real chip, there would be no mismatch at the tester. The ATPG method for hold-time fault in [11] could be used to produce high quality tests for hold-time faults.

From now on, our analyses will be based on *fault_model_1*. For clarity, we omit the scan circuitry. In Section 4, we will discuss the differences between the two fault models from the perspective of simulation and diagnosis. Figure 2 shows a simple example, how to inject the hold-time fault based on *fault_model_1*. Suppose there is a hold time fault *f* (*fault_model_1*) on DFF_1 along the $path_1$: DFF_2-$G3$-DFF_1. The good machine value for {DFF_1, DFF_2} in time frame 0 is scanned-in as {0,0}. After a good machine simulation, the next time-frame's *GMVs* on the data input of both flip-flops are equal to logic 1. Since DFF_2 satisfies the necessary hold-time fault activation condition, i.e., there exists a transition between the current time frame and the next one. For DFF_2 we use the next time frame's *GMV*, which is logic 1, and replace the current time frame's *GMV*, logic 0.

3. Our Approach

In this work, we demonstrate feasibility and efficiency of using timing information such as path delay and clock skew for hold-time fault diagnosis.

We obtain the circuit delay and timing information from the *SDF* files [17] which contain the gate timing, delays, and the interconnect delay. For each pin-to-pin and interconnect, *SDF* provides the rising/falling transition delays.

Our diagnostic algorithm uses the observed failure responses as the input. We first check the activation and sensitization conditions to see if the cause could be the hold-time violation. If so, our algorithm back-traces from the observed failing outputs along the sensitizable paths and identifies the initial fault candidates, $F_{initial}$. Each candidate in $F_{initial}$ is injected (one at a time) into the circuit as a hold-time fault. The algorithm based on the timing and input test pattern performs the functional timing simulation. Once the faulty effect has been propagated to an observation point, we check the timing conditions. If the condition stated by Eq. (1) is fulfilled, we say the hold-time faulty effect can be observed at this point. Finally, we weight each candidate by its faulty behavior capability to match the observed failure responses.

3.1 Functional Timing Simulator

We have developed a framework which allows us to evaluate the 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 behavior for given test sets. All examples discussed in this paper use a pair of delay values to represent the pin-to-pin rising and falling delays. However, in our framework these values are represented by pairs of delay ranges. Interconnect delays are taken into account in our framework.

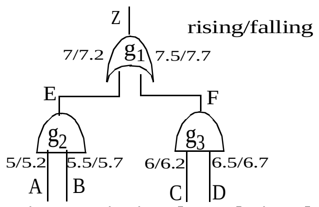


Fig. 3: Timing based simulation example

Consider the example in Figure 3. The numbers beside each gate's inputs are the corresponding rising/falling delays from the input to the output of that gate. We ignore all the interconnect delays to simplify our explanation.

For the pattern P_1: {*ABCD*}: {*r 1 r 1*} (*r* = rising), there are rising transitions on *E* and *F*. Because the logic value *"1"* is a controlling value 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 P_2: {*ABCD*}: {*f 1 f f*} (*f* =falling) there are falling transitions on *E* and *F*, because *0* is a non-controlling value for the *OR* gate g_1. 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. A path from C to Z determines the circuit delay.

Unlike the static timing analyzer, our simulator considers the circuit's functionality for each input pattern.

3.2 Negative Timing Window Propagation (*NTWP*)

In this section, we describe our novel simulation technique which determines fault candidate's capability of explaining the failing and passing pattern responses.

To emulate the speed-up effects in the circuit, we introduce a *negative timing window (NTW)* of a fault candidate as a delay interval *[a,b]* with non-positive values of *a* and *b*. The absolute value of *a*, *Abs(a)*, is the maximum speed-up size, and *Abs(b)* is the minimum speed-up size. *NTW* also captures a situation when the source flip-flop of the data path receives the active clock-edge earlier *(fault_model_1*).

We perform a conventional interval-arithmetic on the timing windows. Let $T_1 = [t_a,t_b]$ 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* in the initial fault list $F_{initial}$, we begin the timing-fault simulation by assigning at the faulty location *f* the initial negative timing window *(NTW) [-L,0]. L* is the longest path delay in the circuit. For each candidate, those windows will be propagated (and possibly shrunk) along the sensitized paths.

The examples in figure 4 show how the negative 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 faulty effect of *f* propagates to a signal line, the corresponding *NTW* can also propagate there. The symbol *R* (*F*) stands for a rising (falling) transition on the signal line.

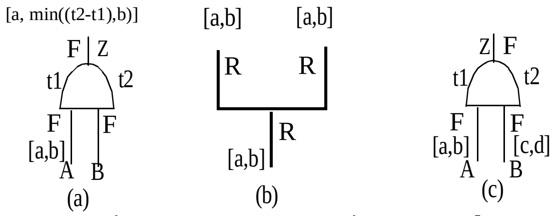


Fig. 4: NTW Propagation Examples

The example in Figure 4(a) shows how the upper bound of an *NTW* is updated. Suppose the faulty effect propagates to *A*, the *NTW* on *A* is *[a,b]*, and the current simulating pattern produces falling transitions on both *A* and *B*. To propagate the speed-up effect from *A* to *Z*, the delay value on *A* and *B* must satisfy the condition $t_1 + b < t_2$, where t_1 and t_2 are the fault-free delay values calculated as described in Section 3.1. In Figure 4 examples, we assume that the pin-to-pin rising and falling delay values are the same for *A-Z* and *B-Z*. The upper bound of the *NTW* must be equal to or smaller than $min((t_2\text{-}t_1), b)$. The new *NTW* at *Z* becomes $[a, min((t_2\text{-}t_1), b)]$.

In Fig. 4(b), the *NTWs* simply propagate to each branch from the stem. In Fig. 4(c) we have a reconverging fanout. The faulty effects propagate to both inputs from different fanout branches of the failure's source. The negative timing window at *A* is $NTW_A = [a,b]$ and at *B* is $NTW_B = [c,d]$. Suppose that the inputs on the *AND* gate have falling transitions. We first derive the delay value range at Z followed by the NTW_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 $[min(t_1+a,t_2+c), min(t_1+b, t_2+d)]$ (*a,b,c and d* are non-positive values).

To determine the NTW_Z, we consider two cases.

Case 1: $D_A \cap D_B = \Phi$ which implies that the delay on one of the inputs dominates the delay at *Z* regardless of the delay changes at the other input. In this case, we simply propagate the *NTW* from the input which has a bigger speed-up 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 $NTW_A \cap NTW_B \neq \Phi$ then $NTW_Z = NTW_A \cup NTW_B$, which means that a defect of any size in the negative timing window *A* or *B* could produce a speed-up at *Z*. We merge these two negative timing windows with no loss of information.

Case 2.2 If $NTW_A \cap NTW_B = \Phi$, we can apply a union operation on these two *NTWs* and form a wider window. Doing so will degrade the resolution because some delay values which are neither in NTW_A nor in NTW_B could be included in the new window. To overcome this problem, we can retain NTW_A and NTW_B and propagate them. The more disjoint 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. Experimentally, this heuristic achieves a good trade off between resolution and performance.

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

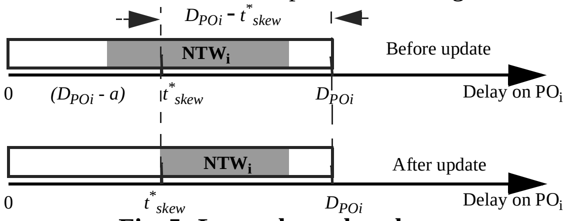


Fig. 5: Lower bound update

Suppose that at the capture time, for a given pattern *T* we cannot observe delay failure on the output flip-flop PO_i. Let M be a sensitizable path to PO_i passing through the fault site. If we have recorded a tuple at PO_i for a pattern *P*, the lower bound of NTW_i, say *a*, must be bigger than $t^*_{skew} - D_{PO_i}$, where $t^*_{skew} = t_{skew} + t_{hold} - t_p$. t_{skew} is the clock skew between the source flip-flop and the sink flip-flop of the path M, and D_{PO_i} is the path delay value at PO_i calculated for the fault-free circuit. t_p represents the propagation delay from the source FF clock activation time to the source FF registering-data time. t_{hold} is the hold-time requirement for the sink flip-flop of the path M. This is illustrated in figure 5.

From our experiments, we found that *L* must be preset to a value bigger than the possible speed-up (skew) defect size. Otherwise, the upper bound modifications may cause the upper bound to become smaller than *-L* and produce no candidate. Presetting *L* to the longest path delay value guarantees that this situation will not happen. In Section 3.4 we will explain how to extract useful information from those tuples and how to prune further the unlikely candidates.

3.3 The Diagnostic Algorithm

Our algorithm to diagnose the hold-time related defects is based on the timing information and delay simulation. We make an assumption that each failing pattern can be attributed to a single fault location. Our algorithm is capable of identifying multiple fault locations as long as each failing pattern is affected by only one fault. We also assume that, if two candidates have the same explanation capabilities for a set of failing and passing patterns, the candidate with the smaller speedup size has a higher probability of being the real defect.

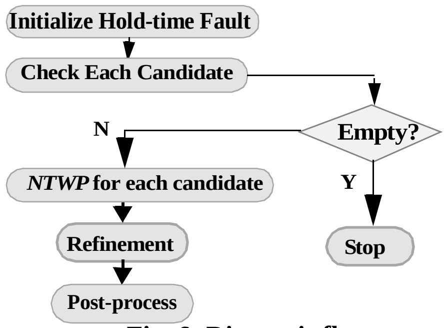


Fig. 6: Diagnosis flow

In Figure 6, we state our algorithm for diagnosing hold-time defects modeled by the *fault_model_1*:

For a failing pattern *T*, we path-trace from each failing flip-flop and initialize the fault candidate list $F_{initial}$. All the flip-flops in the failing flip-flop's fanin cone and the failing FF itself are the initial fault candidates. Based on the single-fault per pattern assumption, the initial candidate faults must reside in the intersection of fanin cones of different failing FFs. If this is not the case, we eliminate the fault from $F_{initial}$.

Before performing the diagnosis, we verify the possibility that the failure is caused by a hold-time defect.

Step 1: If for a pattern *T* there is no sensitizable path from the candidate site to the failing flip-flop, eliminate this fault from the $F_{initial}$ list.

Step 2. (Hold-time *fault_model_1*) If for a pattern *T* in the good machine there is no transition at the fault candidate, eliminate this fault from the $F_{initial}$.

For each candidate in $F_{initial}$ we check to see if the activation and sensitization conditions (Section 2) are fulfilled. If any of the conditions is not satisfied, this candidate is eliminated from the initial fault list. This checking step is performed for both fault models, even though we presented the details only for the fault-model_1.

If $F_{initial}$ is not empty, continue. Otherwise, the failure is not caused by a hold-time fault. The algorithm stops. For each candidate in the $F_{initial}$ we apply the *NTWP* technique, simulate the patterns, and record the tuples. We prune the unlikely candidates and report the candidate set F_{after_NTWP}. In the refinement step, if a group of candidates has the same explanation capability, we deduce the most likely speedup size for each candidate. The candidates are ranked by their speedup sizes. Candidates with smaller speedup sizes are reported earlier. Finally, we post-process the most probable candidate sites and paths and deduce the most possible causes of failures.

3.4 Pruning Rules

The rules for pruning the impossible candidates are derived based on the relations between the negative timing windows at the observation points.

Rule 1: For two tuples *{f, P, PO_1, NTW_1}* and *{f, P, PO_2, NTW_2}*, if 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 *negative timing windows* have a conflict as shown in figure 7.

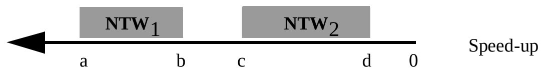


Fig. 7: Timing Window Conflicts

This rule eliminates those candidates which introduce inconsistent speed-up values when propagated to different failing *POs*.

Rule 2: If for a tuple *{f, P, PO_1, NTW_1}* the summation of the lower bound of NTW_1 and the path delay value calculated on PO_1, t_1+a, is bigger than t^*_{skew}, where $t^*_{skew} = t_{skew} + t_{hold} - t_p$, then *f* is not a candidate. From our previous analysis (Eq. (1)), we know that if a path delay is bigger than t^*_{skew}, the hold-time violation does not occur. Figure 8 shows a situation where t_1 is the fault-free path delay value calculated as described in section 3.1.

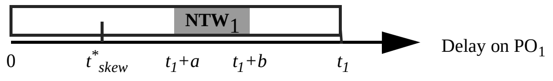


Fig. 8: Rule 2.

Rule 3: If we do not observe a failure on PO_1, but the summation of the upper bound of NTW_1 and the delay value calculated on PO_1, t_1+b, is smaller than t^*_{skew}, then *f* is not a candidate for *P*. Based on Eq. (1), if t^*_{skew} is bigger than t_1+b, we should observe the hold-time failure on PO_1.

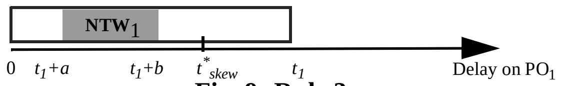


Fig. 9: Rule 3.

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 all the failing and passing patterns which can be explained by the fault candidates. For each fault we collect *NTWs* and construct a *wave(t)*. The *wave(t)* is built such that for each delay *t*, we assign a value equal to the number of *NTWs* which cover *t*. For each fault, the most probable is presumed to be the delay value/range with the highest weight. We measure the waveforms of the candidates by their integrals from *-L* to *0*, which yield the total area *Area(t)* covered by a *wave(t)*:

$Area(t) = \int_{-L}^{0} wave(t))dt$. We define the *ratio function* as $Ratio(t) = Area(t)/Area(-L)$. For each candidate we obtain three points -- t_{lb}, t_{mid}, and t_{ub} -- by setting the *ratio(t)* to be 0.3, 0.5, 0.7.

For locating the fault candidate, a clustered distribution graph *w(t)* is more meaningful than a sparse distribution graph. 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 speedup range. In figure 10, all waveforms cover the same area. The candidate which has the speedup distribution (c) is more probable than the candidate which has the speedup distribution (b).

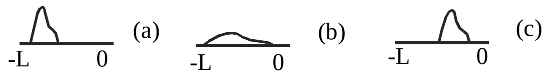


Fig. 10: Speedup distribution graph example

After obtaining the distribution graphs for all the faults, we might find that two faults have about the same density. Based on our assumption stated in Section 3.3, a fault which has a smaller t_{mid} is a better candidate than the fault which has a bigger t_{mid}. In figure 10, the ranks of three candidates are (c) > (a) > (b).

3.6 Post-processing the Candidates

After the refinement step, the diagnostic algorithm terminates with a list of fault candidates and sensitizable paths from the candidate sites to the observed failing outputs. In the post-processing step, we report possible candidates based on assumptions about the defects. Here we consider short path speed-up and clock skew as possible causes of failures. Since it is impossible to distinguish which of those two failure causes is the culprit, we report faulty sites based on each of those assumptions along with the sizes of timing failures. A failure analyzer could decide which failure to target first, based on manufacturing experience.

4. Discussion

When we inject a hold-time *fault_model_1* into a circuit, we use the next time frame *GMV* to replace the current *GMV* on the transitioning path-source flip-flop. In case of the *fault_model_2*, the sink flip-flop of a path receives the clock edge later, and to inject a fault, every flip-flop except for the sink flip-flop needs to replace its current time frame *GMV* with the next time frame *GMV.* The fault simulation process is the same as for the *fault_model_1*.

For the *fault_model_1*, we use the current time frame to perform back tracing, identify the sensitizable paths, and use the source/sink flip-flops of those paths as the initial fault candidates. For the *fault_model_2*, we perform good machine simulation based on the next time frame. We perform back-tracing, based on the next time frame *GMV,* identify sensitizable paths, and use the source/sink flip-flops of those paths as the initial fault candidates.

For the *fault_model_1*, we inject the initial *NTW* as *[-L,0]* at the fault site and perform timing-based simulation in the *current* time frame. For the *fault_model_2*, we inject the initial *NTW [-L,0]* at each transitioning flip-flop for which there exists a sensitizable path terminating at the failing observation point in the next time frame of the pattern.

In practice, multiple flip-flops could be the source of hold-time failures as shown in figure 11. However, we found that even though multiple hold-time faults exist in the circuit, many failing patterns can activate and observe the faulty effect of only one of them. These failing patterns fall into our *type-1* failing pattern category.

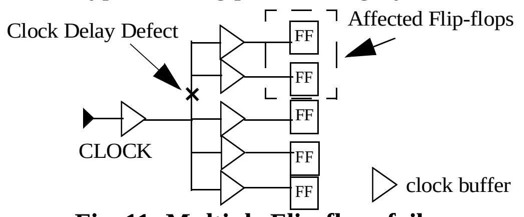


Fig. 11: Multiple Flip-flops failure

The diagnostic algorithm proposed in Section 3.3 can be altered to diagnose failure responses caused by multiple hold-time-fault sites.

A. Initial Fault Candidates

For multiple-fault diagnosis, the initial candidate fault must reside in at least one of the fanin cones of different failing flip-flops of the given pattern.

B. Diagnosis Step

First we perform the diagnostic algorithm based on the *single fault per pattern* assumption to find candidates which could explain some failing patterns. If multiple faults exist in the circuit, but each of them has a test pattern which can activate and observe only one faulty behavior (*type-1* failing pattern), then every fault site could be identified correctly and easily. However, if every failing pattern is affected by multiple faults (*type-2* failing pattern), the proposed algorithm may face some difficulties. To overcome this limitation, we use a failing-PO-partition technique proposed in [14]. The basic idea of this technique is to partition the failing POs into groups such that each group of failing POs is affected by only one or few faults. After the partitioning, most of the *type-2* failing patterns are transformed into several *type-1* failing patterns so that the proposed algorithm can be applied further. This partitioning technique is very useful especially for big industrial designs, most of which are full-scanned and very flat, with a large number of observation points.

Experimental results have shown that most of the failing patterns are of *type-1*. For small number of test cases whose failures exhibit only *type-2* patterns, our partition technique can help find the correct candidates.

C. Post-processing Step

After finding multiple locations by using *type-1* patterns and the failing-PO-partition technique, we rank the candidate sites by their capability of explaining failure responses. Then we group those higher ranked candidates together. If layout information is available, based on either speedup or clock skew assumptions, we can find the most likely defect locations.

5. Experimental Results

5.1 Results for Single Hold-time Fault

Since obtaining accurate timing information is not the main purpose of this work, we used static timing information instead of accurate timing information which could be obtained from SPICE simulation or SDF files.

We evaluated diagnostic capabilities of our algorithm using the hold-time fault-simulation framework. Our experiments were set up as follows. We randomly injected a hold-time fault into the circuit, performed simulations, and collected the failure responses for the given tests. We fed the failure responses into the diagnostic tool using the same tests. These algorithms reported possible faulty locations. For each circuit, we performed 100 times random fault injections to obtain different "faulty" circuits and average diagnostic results. The test set for diagnosis was generated by a commercial tool targeting all testable stuck-at faults.

Table 1: Results for Single Fault

Circuit	*# of gates*	*# of Init. f*	*# of Cand.*	*FHR*	*T (sec)*
B14	5.7k	6.1	2.2	1	0.78
B15	10.7k	7.4	2.4	1	1.68
B17	29.7k	8.6	3.3	1	5.25
B18	81.7k	8.2	3.4	1	27.47
B20	11.4k	9.2	2.9	1	2.04
B21	11.8k	8.8	2.8	1	1.98
B22	18.0k	7.8	3.1	1	3.78
Avg.		***8.01***	***2.87***	***1***	***6.14***

Our diagnostic algorithm reports lists of paths and the source/sink flip-flops of those paths. In addition to paths, we assign weights to the flip-flops. The weight of a flip-flop is equal to the number of reported paths which contain it. We order the flip-flops with the highest weight reported first. A rank is a position on the ordered list of fault sites, the first fault of which matches the injected fault site. This is the *first hit rank* (*FHR*).

Table 1 shows the diagnostic results for full scanned ITC99 benchmark circuits whose sizes are bigger than 5K gates. The first and second columns show the circuit names and sizes. The third column shows the average number of the initial fault candidates before using the timing information to prune the unlikely candidates. The fourth column is the number of candidates reported by the algorithm. A candidate consists of a number of paths from source flip-flop to sink flip-flop.

The fifth column is the average first hit rank of our algorithm. The last column shows the runtime (*T*) of the algorithm. The algorithm was implemented in C language, and we ran all the experiments on a PC under Linux OS with a 2GHz CPU and 1GB memory.

Resolution is the ratio of the number of injected faults over the count of total reported candidates. Our algorithm can locate the injected fault locations with a very low first-hit rank and good resolution. Because we only injected one fault into a circuit, our algorithm can always find the best faulty site and rank it as the number one candidate.

5.2 Results for Multiple Hold-time Faults

In the second round of experiments, we randomly injected two or three hold-time faults into the circuits and collected the failure responses. If too many flip-flops seemed to be the source of hold-time failure, we simply screened the clock network instead of performing diagnosis. For each circuit and a given number of faults, we performed 100 times random fault injections to get varying "faulty" circuits. Then we obtained the average diagnosis results for those test cases. Table 2 shows the experimental results to diagnose the failure responses caused by multiple hold-time faults. The first column shows the circuit name. The second column shows the number of fault candidates reported by different injected fault numbers before using timing information. In the table, "***2-f***" refers to the results for two hold-time faults injections and "***3-f***" to three hold-time faults. The third column shows the number of reported candidates after using timing information. The last column shows the diagnosability (*DA*) for different fault density. *Diagnosability* is defined as the ratio of the correctly identified fault count over the total number of injected faults.

Table 2: Results for Multiple Hold-time Faults

Circuit	***# of init. f***		***# of Cand.***		***DA***	
	2-f	***3-f***	***2-f***	***3-f***	***2-f***	***3-f***
B14	11.7	13.2	4.2	5.8	0.98	0.96
B15	13.3	16.4	4.3	6.1	0.99	0.99
B17	18.2	22.5	5.5	7.2	0.99	0.97
B18	18.8	24.7	5.7	9.3	0.99	0.99
B20	14.2	20.4	4.4	6.3	0.98	0.98
B21	19.7	32.2	3.9	6.8	0.99	0.98
B22	21.3	27.1	6.2	9.6	0.95	0.94
Avg.	***16.7***	***22.4***	***4.89***	***7.3***	***0.98***	***0.97***

The results suggest that for the majority of cases our approach can identify the injected faulty locations with good resolution. By using timing information, our approach significantly reduces the number of reported candidates. The FHR data for all those cases are less than 1.2 on average. The performance of our algorithm is linear with the number of initial fault candidates.

6. Conclusions

We have proposed timing-information-based diagnosis of hold-time related defects. We developed a novel yet simple timing-driven hold-time fault simulation method. We proposed a diagnostic approach of the hold-time violations which could be caused by short path speed-up or clock skew. The proposed approaches achieve very good resolution and high performance.

Acknowledgement

This work was supported by the California MICRO Program and Mentor Graphics Corporation. We also thank Sudhakar M. Reddy for his helpful comments on the hold time failure diagnosis problem.

References

[1] D.Bailey and B.Benschneider, "Clocking design and analysis for a 600-MHz alpha microprocessor", IEEE J. Solid-State Circuits, vol.34, pp.1478-1491, Nov. 1999.

[2] S. Edirisooriya and G. Edirisooriya, "Diagnosis of Scan Failures", *Proc.* VLSI Test Symposium 1995, pp.250-255.

[3] R. Guo, S. Venkataraman, "A technique for fault diagnosis of defects in scan chains", *Proc. Intl. Test Conf*. 2001. pp.268-277.

[4] D. Harris, M. Horowitz, D. Liu, "Timing analysis including clock skew", IEEE Trans. on CAD, vol. 18, (11), Nov. 1999, pp.1608 - 1618.

[5] D. Harris, S. Naffziger, "Statistical clock skew modeling with data delay variations", IEEE Trans. on VLSI Systems, vol.:9, (6) Dec. 2001, pp. 888 - 898.

[6] Y. Huang, W.T. Cheng, C. Hsieh, H.Y. Tseng, A. Huang, Y.T. Hung, "Efficient diagnosis for multiple intermittent scan chain hold-time faults", Asian Test Symposium, 2003, pp.44 - 49.

[7] Y. Huang, W.T. Cheng, S. M. Reddy, C.J. Hsieh, Y.T. Hung, "Statistical Diagnosis for Intermittent Scan Chain Hold-Time Fault", *Proc. Intl. Test Conf*, 2003, pp. 319-328.

[8] S. Kundu, "Diagnosing Scan Chain Faults", IEEE Trans.On VLSI Systems, vol. 2,(4) 1994, pp.512-516.

[9] N.A. Kurd, J.S. Barkarullah, R.O. Dizon, T.D. Fletcher, P.D. Madland, "A multi gigahertz clocking scheme for the Pentium$^{(R)}$ 4 microprocessor", IEEE J. of Solid-State Circuits, vol. 36, (11) Nov. 2001,pp. 1647 - 1653.

[10]S. Narayanan, A. Das, "An Efficient Scheme to Diagnose Scan Chains", *Proc. Intl Test Conf.*, 1997, pp.704-713.

[11]S.M. Reddy, I. Pomeranz, A. Murakami, M. Ohta, "On validating data hold times for flip-flops in sequential circuits", *Proc. Intl. Test Conf*, 2000, pp. 317 - 325.

[12]P.J. Restle, et. al, "A clock distribution network for microprocessors", IEEE J. of Solid-State Circuits, vol.36, (5) May 2001, pp. 792 - 799.

[13]K. Stanley, "High-Accuracy Flush-and-Scan Software Diagnostic," IEEE Design & Test of Computers, 2001, pp.56-62.

[14]Z. Wang, K-H. Tsai, M. Marek-Sadowska, J. Rajski, "An Efficient and Effective Methodology on the Multiple Fault Diagnosis", *Proc. Intl Test Conf*, 2003.

[15]Z. Wang, M. Marek-Sadowska, K.H. Tsai, J. Rajski, "Delay Fault Diagnosis Using Timing Information", Int. Symp. on Quality Electronic Design, 2004.

[16]Y. Wu, "Diagnosis of scan chain failures", Defect and Fault Tolerance in VLSI Systems, *Proc*. IEEE Int. Symp. on 1998, pp. 217 - 222.

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