IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 7, Issue 4, Ver. II (Jul. - Aug. 2017), PP 38-47
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Identification of High-Risk Hardware Path-Delay Fault Locations and Evaluation of Their Impact

*Tadele Basazenew Ayele
[1](Addis Ababa Institute of Technology/ Addis Ababa University, Ethiopia)
Corresponding Author: Tadele Basazenew Ayele

Abstract:*If the delay of the manufactured network exceeds the specifications due to some physical defects or process variations, non-confidential logic values may be latched in memory elements. The method proposed here combines the static timing analysis techniques and the dynamic timing analysis techniques in combinational circuit timing verification. The framework proposed here combines the static timing analysis(STA) techniques and the dynamic timing analysis(DTA) techniques in the presence of transition delay fault model. Our framework takes the timing violations paths from STA and by using the proposed algorithm analysis this output for false path using dynamic timing analysis. We assumed that, for combinational circuits, paths with at least one negative slack node are considered as high-risk paths. Paths which have zero slack have a higher probability of being high-risk, so that they are considered here for testing. The Common path pessimism removal methods proposed in many literatures identifies the pessimism imposed by the false paths, but didn't consider the dynamic timing analysis based approach for false path identification. Experimental results show us that, the dynamic timing analysis based false path identification technique can be used as an alternative method of timing analysis during digital circuit design verification.*

Keywords: *High-Risk paths, Delay fault model, false paths, Pessimism*

Date of Submission: 15-08-2017 Date of acceptance: 05-09-2017

I. Introduction

Correct operation of a logic circuit requires not only performing the logic function correctly but also propagating the correct logic signals along paths within a specified time limit. Most researchers and digital circuit designers use static timing analysis(STA) to analyze the timing correctness of a digital circuit design. Static timing analysis is a key component of any integrated circuit (IC) chip design-closure flow, and is employed to obtain bounds on the fastest (early) and slowest (late) signal transition times for various timing tests and paths in the chip design[1,2,4,5]. Growing chip design sizes and complexities such as, increased number of clock domains, increased significance of crosstalk coupling, voltage islands, etc., as well as more complex and accurate timing models (e.g., current source models) leads to longer timing analysis run-times, thereby hindering designer productivity[1]. STA methods use the common path pessimism removal (CPPR) method to remove false paths from the timing report [1, 2]. But the CPPR methods have still so many criticisms from circuit designers and verification engineers [1, 2]. The CPPR methods consider the circuit design i.e. netlist and the cell library together. But here, our focus is on the circuit netlist only.

In Common path pessimism removal (CPPR) based methods, designers and researchers have used the STA techniques to reduce the pessimism of a timing analysis report. According to [1], the traditional CPPR methods uses a set of filters or fast-outs is then employed by CPPR algorithms to further prune the search space and to reduce pessimism of STA. Among common filters used in CPPR are path-limit, pre-CPPR slack-limit, and post-CPPR path-slack-limit. This search space pruning may leave some main circuit parts without exploring them. But in this paper, we used the 5-valued algebra for robust path delay faults proposed in [3] to identify the redundant paths, i.e. false timing paths so that to reduce the pessimism of the path. As it is stated in [3, 4], the 5-valued algebra is applicable to robust test and with some modification [4], it can be applicable for non-robust test and can remove a significant amount of pessimism. In this paper we used the output of STA tools slack report and we applied the Test vector on negative slack nodes only, so that we decrease the computational overhead. That is we combined methods of STA and algebraic methods to compute the true high-risk paths. It is stated in many literatures [5, 7], timing verification is performed after place and route of our design. But here, we are trying to show that, we simulated our design by using STA tools and if we get the timing violation, we take that violation and come back to our design at the gate-level netlist to verify by using the dynamic technique proposed in this paper.

VIJAY **S.** IYENGAR et.al in [8] proposes a method of delay fault detection threshold. They stated that Delay faults have the counterintuitive property that a test for a fault of one size need not be a test for a similar fault of a larger size. They propose a model for delay faults that can identify various fault sizes. In another work, Gordon L. Smith in [9] proposes a path based delay fault detection methodology. This work proposed a method of selection of paths in such a way that because the number of paths in a network can be excessively large, reduction of the number of paths in the path list is important. Following this analysis, in our work, we have targeted only high-risk paths so that, we can decrease the computational overhead. The difficulties associated with directly modeling of gate delay faults is also discussed. A combinational network is free of delay faults of any size if and only if there are no path faults. However, analysis of actual gate delays and delay fault sizes usually demonstrates that testing of a small subset of all paths is sufficient to assure with adequate confidence that all paths are free of path faults. These small subset of all paths that need to be tested to justify the whole circuit is free of delay fault are said to be high-risk paths. While it usually is not economical to explicitly include delays in a procedure that determines the paths that are delay testes, it is appropriate to consider delays during generation of the path list. In [10] and [11], researchers propose a statistical static timing analysis(SSTA) based method to identify false paths in STA and to reduce the pessimism.

None of the methods analyzed above are deterministic. So in this work, we propose a new deterministic method to improve the commonly used STA method and to get better result. In this work, we propose an STA based fault list generation and stuck-at fault application method. In this, work we detect delay faults from STA simulation, and bring the fault list to the dynamic delay fault simulation. Many research works proposed so many methods to avoid the pessimistic nature of static timing analysis, like the ones [1, 2,5,10, 11, 12], but non-of these consider the test vector based method used to identify the high-risk paths that may be false paths. In this paper we use the method of Static Timing Analysis(STA) to find high-risk paths and to selected high risk paths based on the their slack value. That is, in combinational circuits, paths that have negative slack have higher delay than paths with positive slack. And after identifying the critical paths, in the second phase, we apply test vectors on those paths to identify whether they are true paths or false paths. The test vector is applied only to those paths which are identified as high-risk by static timing analysis, so that it will decrease the overhead imposed by applying test vector on the whole circuit paths. The one in [13] describes redundancy identification as a form of false path identification, but didn't consider the STA output as a method of the timing violation identification mechanism.

The paper is organized as follows. Section II describes some background on static timing analysis methods based on the available literatures and circuit simulation to extract the delay parameters. Section III presents the proposed method of identifying the false and true paths and identifying the pessimism incurred in STA. Section IV presents the experimental analysis for high-Risk algorithm. And finally, section V concludes the paper.

II. Model To Select All High-Risk Paths

As stated in [5], [12], [7] and [12] a static timing analysis of a design typically provides a profile of the design's performance by measuring the timing propagation from inputs to outputs. Timing analysis computes the amount of time signals propagate in a circuit from its primary inputs (PIs) to its primary outputs (POs) through various circuit elements and interconnect.

Definition 1: In a combinational circuit, a high-risk path is a path with at least one negative slack node along the path from a certain primary input to a certain primary output in the static timing analysis.

Definition 1 explicitly specifies the combinational circuit only. The definition didn't consider the sequential circuits. Because, in sequential circuits, the path which has a negative slack node along it, is not always a high-risk paths. For sequential circuits, the criteria of high-risk path selection is that the sum of setup and hold values of the state elements should be positive, or else, it is a timing violation.

STA is a method of verifying expected timing characteristics of a circuit [1]. STA computes the arrival time, the required arrival time and the slack of a circuit node. Observing the numerical values of the slack, we can deduce whether the circuit meets the timing requirements or not. Given an arrival time and a required arrival time, the slack at a circuit node quantified how well timing constraints are met. That is, for combinational circuit timing analysis, a positive slack means the required time is satisfied, and a negative slack means the required time is in violation. Using definition 1, in our work, we consider paths with at least one negative slack node as high-risk paths.

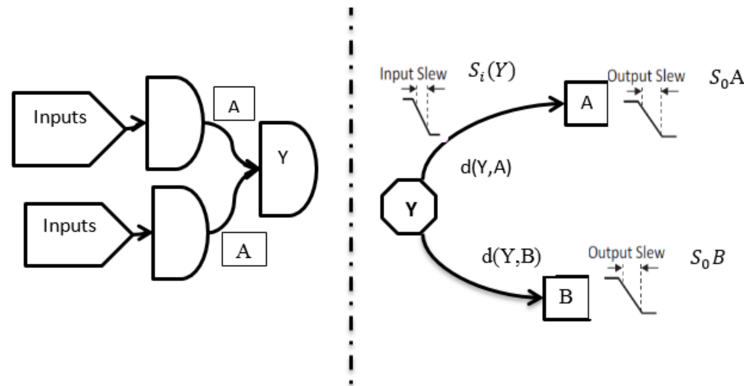

Figure 1: generic circuit (left) and delay model representation of a combinational element (right)

From Fig. 1, we can observe the two parts of the circuit. The one in the right is the circuit for STA representation and the one in the left is the dynamic timing analysis representation. The STA simulation includes the device library and the design together. If the design that is the picture in the left has a problem and creates the pessimism in the STA, we can verify it by using the dynamic timing analysis by applying a test vector at the primary inputs.

Actual arrival time[1,2, 5,7,12] is defined as starting from the primary inputs, arrival times (at) are computed by adding delays across a path, and performing the minimum (in early mode) or maximum (in late mode) of such accumulated times at a convergence point. For example, let $at^E(A\ )$ and $at^E(B)$ denote the early arrival times at pins A and B, respectively, in Fig 1. The most pessimistic early mode arrival time at the output pin Y is:

$$at^E(Y) = \min\left(at^E(A) + d^E(A,y), at^E(B) + d^E(B,y)\right) \dots\dots\dots\dots\dots\dots\dots\dots\dots (1)$$

By using the same concept of equation 1, as stated in [18], in late mode, the latest time that a signal transition can reach any given circuit node is computed. FromFig1, the most pessimistic late mode arrival time at Y is:

$$at^L(A) = \max\left(at^L(A) + d^L(A,y), at^L(B) + d^L(B,y)\right) \dots\dots\dots\dots\dots\dots. (2)$$

Once again, in [5], [7], [12], and many other literatures, it is stated that, starting from the primary outputs, required arrival times (rat) are computed by subtracting the delays across a path, and performing the maximum (minimum) in early (late) mode of such accumulated times at a convergence point. For example, a generic interconnect (input Y, output A, B), the most pessimistic early mode required arrival time at the output pin Y is:

$$rat^E(Y) = \max\left(rat^E(A) - d^E(y,A), rat^E(B) - d^E(y,B)\right) \dots\dots\dots\dots\dots\dots\dots (3)$$

By using the same concept of equation 3, in late mode, the earliest time that a signal transition must reach a given circuit node is computed. From Fig 1, the most pessimistic late mode required arrival time at the input pin Y is:

$$rat^L(A) = \min\left(rat^L(A) - d^L(y,A), rat^L(B) - d^L(Y,B)\right) \dots\dots\dots\dots\dots\dots.. (4)$$

For proper circuit operation, the following must hold:

$$at^E \geq rat^E \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots. (5)$$
$$at^L \leq rat^L \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (6)$$

To quantify how well timing constraints are met at each circuit node, slacks can be computed based on Equations 5 and 6. That is, for combinational circuits, slacks are positive when the required times are met, and negative otherwise.

$$slack^E = at^E - rat^E \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots.. (7)$$
$$slack^L = rat^L - at^L \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots.. (8)$$

A slew rate is a rate of change [7]. The slew is typically measured in terms of the transition time, that is, the time it takes for a signal to transition between two specific levels [1, 2, 5, 6,7, 12]. Slew propagation is defined as circuit element delays and interconnect delays are functions of input slew(s_i); subsequent output slew (s_o) must be propagated. FromFig1, the early mode and late output slew at output pin Y are, respectively:

$$s_0{}^E(Y) = \min\left(s_0{}^E(A,Y), s_0{}^E(B,Y)\right) \dots\dots\dots\dots\dots (9)$$
$$s_0{}^L(Y) = \max\left(s_0{}^L(A,Y), s_0{}^L(B,Y)\right) \dots\dots\dots\dots\dots.. (10)$$

By using definition 1 and equations 5, 6, 7 and 8, we can identify which path is a high-risk paths. In this case a high-risk path is a path with at least one negative slack node along the path from a certain primary input to a certain primary output in the circuit's directed acyclic graph. As we know, a circuit path is a sequence of nodes from primary input towards primary output. So, if a certain path has a negative slack node, we call it

high-risk path. But this method of identifying high-risk paths can include false paths which have no important in circuit implementation, so that we have to get rid of these false paths using dynamic timing analysis. Here, we propose a deterministic algorithmic based false path and true-high-risk path identification for static timing analysis, so that we can identify the pessimism occurred by the false paths.

III. The Proposed Timing Analysis Framework And Algorithm

Static timing analysis[1,2] is a key component of any integrated circuit(IC) chip design timing-closure flow, and is employed to obtain bounds on the fastest (early) and slowest (late) signal transition times for various timing tests and paths in the chip design. But the STA that is popular in many VLSI hardware industries is prone to pessimism. That means it outputs false paths as True high-risk paths. There are different methods of common path pessimism removal [1, 2, 10, 11], but non-of them consider dynamic timing simulation technique. Here, in this paper, we propose a dynamic simulation based Pessimism removal technique to locate high-risk true paths and false high-risk paths. In [11], they have usedstatistical timing analysis using formal methods to analyses the pessimistic nature of STA output but didn't consider the deterministic method of identifying the false paths from the design netlist. The method presented here uses the traditional Static timing analysis (STA) as basis and we developed a framework and algorithm to identify false paths which create pessimism and we determine the timing delay of a circuit by applying a test vector by using 5-valued Boolean algebra on critical paths only. Here, we test the high-risk path for slow-to rise and slow-to-fall transitions. This is because, some nodes may be redundant for slow to rise fault but not redundant for slow to fall fault. The proposed framework is here:

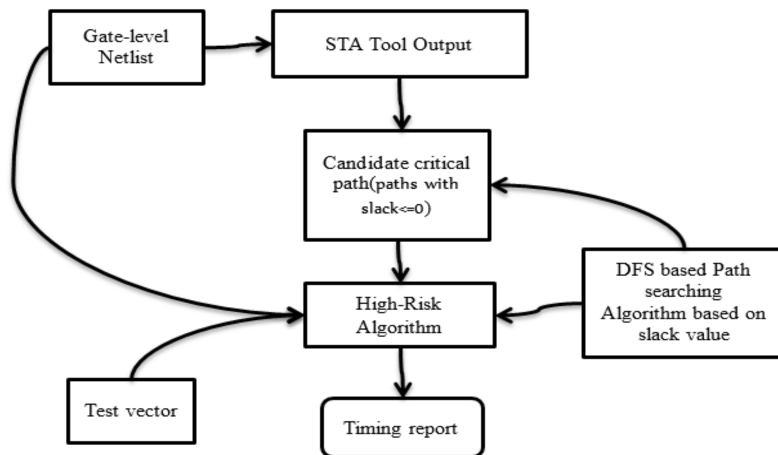

Figure2. The test vector based false path identification method and High-Risk Algorithm work flow

The framework works in such a way that, first, we have to look for a node with negative slack in the circuit graph, and we have to parse through the gate-level netlist file to apply a 5-value propagation table to check whether the path through the negative slack node is false path or not. That is, the STA tool output file gives as the nodes with negative slack, we have to parse through the Verilog gate-level netlist to find the node expressed in STA tool output file, so that we apply the test vector to test whether the slow-to-rise or slow-to-fall transition delay fault is redundant or not. According to [14, 15, 16, 17], stuck-at fault redundancy and transition fault redundancy are different. So, applying a stuck-at fault test vector and transition fault test vector to identify false paths is a different idea. In transition delay fault test we use 2 set of test vector, say V1 and V2 to check for timing violation. V1 is an initialization (stabilization) vector and V2 is a propagation test vector. So in this work, we use transition test vector to search for transition fault redundancy so that, we can spot false paths in the design which leads to pessimism in the STA report.

In this framework, we have combined so many methods together, and we proposed a deterministic delay fault simulation model for combinational circuits. We use the algorithm described in [1, 2, 5, 6, 7, 12] to drive the parameters needed in STA analyzer. After analyzing the given circuit for longest path by using static timing analysis (STA), we have to identify the false paths, to avoid pessimism. To do that, we have to use a 5-value Boolean algebra (propagation table for transition delay faults). We use the Boolean algebra (propagation table) expressed in [3, 4] to identify the false paths and true high-risk paths. The Algorithm uses transition delay fault model as a delay fault model expressed in [15] and [16]. The Algorithm is shown below.

High-Risk Algorithm: - to compute the true high-risk paths under the dual fault injection (both slow-to-rise transition delay fault and slow-to-fall transition delay fault scenario

Input:

- **enumerative fault List(Gathered from STA output timing violations ($\forall\, Slack_{.} \leq 0$ ={PI, G1, G2,….Gn, P0})),**
- **Verilog gate-level Netlist,**
- **5-value algebra propagation table(transition delay fault test vector),**
- **The controllability and observability of the gate i/o**

Output:-

- **#true high-risk paths,**
- **#false paths**

$\forall Nodes, slack \leq 0$

- Report as high-risk node
- find the path from input to output through the high-risk node

$\textbf{Preport}$: $a\ path\ through\ at\ least\ one\ high - risk\ node\ as\ \ high - risk\ path$

If node is high risk
{
Check for slow-to-fall delay fault **and** slow-to-rise delay fault
}

Case slow-to-rise Delay fault:
$DFS\ for\ a\ Node\ with\ slack \leq\ 0$
For(i=0, i<= #negative slack node, i++){
Inject slow-to-rise delay fault
Faulty output of node i= 0 **AND**faulty input of node i
 Apply U0 to negative slack node
 Propagate U0 to the O/P
 O/P=U0
 Backtrack with easy-to-control node
 if(conflict)
  Redundant (false path)
  Break;
  }
 Repeat for any node slack<=0
if(#negative slack node==0)
 Break;

Case slow-to-fall delay fault:
$DFS\ for\ a\ Node\ with\ slack \leq 0$
For(i=0, i<= #negative slack node, i++){
Inject slow-to-fall delay fault
Faulty output of node i= 0 **OR**faulty input of node i
 Apply U1 to negative slack node
 PropagateU1 to O/P
 O/P=U1
 Backtrack with easy-to-control node
 If(conflict)
  Return redundant
  }
Repeat for any node slack<=0
if(#negative slack node==0)
 Break;

The input of the High-Risk Algorithm is the output of a static timing analyzer. The static Timing Analyzer reports all slack of a circuit node. Then the high-risk algorithm tests each negative slack node for slow-to-rise and slow-to-fall transition delay fault sensitizabality to remove or resynthesize the circuit or fault reduce the design netlistfor removing the false paths that create pessimism in the STA output if the pessimism in the STA is occurred by the designnetlist. The algorithm didn't consider the pessimism occurred due to the cell library. First the Fault list is gathered from STA tool output. The fault list is applied by enumerative method. This is because, the STA tool available today analysis almost all the Path delay faults. But a small number of paths violate the timing constraint. So that we can hand pick those paths from STA output and apply a Stuck at fault on them. Then, the Verilog gate-level netlist is tainted with a transition fault gathered from STA tool output. As we know, enumerative path delay fault simulation enumerates the path delay faults and a list of detected faults.

In this framework, we apply transition delay faults i.e. both slow-to-rise and slow-to-fall delay faults on negative slack nodes and we test for transition delay fault redundancy (testability). In STA, the false path removal method proposed in literatures like [1, 2] is CPPR (common path pessimism Removal) method, but here in this work, we just apply test vector on those nodes which violate timing in the STA simulation. That means we take list of nodes which violate the timing constraints from STA output, we inject transition delay fault on paths that contain those nodes which violate timing and we apply the test vector on the Verilog gate-level netlist to check for redundant transition delay faults.

1.1. DEMO OF HIGH-RISK ALGORITHM AND THE PROPOSED FRAMEWORK

Let's demonstrate the proposed algorithm and fault simulation technique with Examples. As you know, the actual arrival times are calculated based on the distance between nodes and the delay of the nodes themselves. These Delays are found from look-up table, i.e. STA simulation output. The following circuit is given with a Boolean formula, $F = AB + CD$ which is in canonical form.

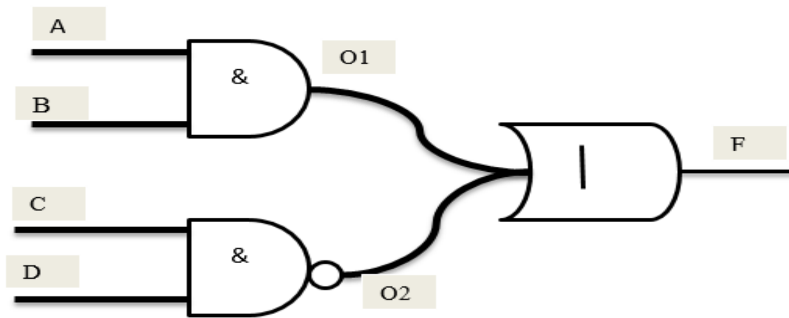

Figure 3:- example Circuit to demonstrate the STA and the proposed method to identify false path and pessimism.

The Directed Acyclic graph representation of the above circuit is given below. The nodes denoted as S and S0 are dummy nodes.

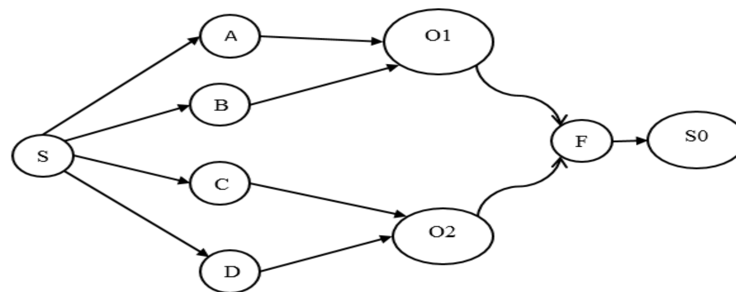

Figure 4: graph of figure 3

Assume that we have given the following parameters to calculate the delay and the high-risk paths. Note also that the values of the node and segment delays between the nodes are not shown in the graph above. The Actual arrival time and the required arrival time are calculated based on the look-up table value derived from the STA simulation.

TABLE 1: example table values taken from STA output for algorithm demonstration

circuit node	Actual Arrival Time	Required Arrival Time	Slack	Controllability and observability values
S	0	-0.3	-0.3	-
A	0	0.9	0.9	1/1/6
B	0	-0.35	-0.35	1/1/6
C	1.1	0.75	-0.3	1/1/6
D	0.6	0.9	0.3	1/1/6
O1	3.2	3.1	-0.1	2/3/4
O2	3.4	3.05	-0.35	2/3/4
F	5.65	5.3	-0.35	5/4/0
So	5.85	5.5	-0.35	-

From last column in table1, we try to show that the controllability (both 0 and 1) and observability values for the circuit in Fig 3. As it is stated in [4] for respective controllability and observability values, large numbers indicate that, it is difficult to control or observe. Now, let's check all negative slack nodes and paths for

slow-to-rise and slow-to-fall delay fault redundancy. Even if all false paths are not caused by redundant faults, we can find a significant number of false paths by testing the circuit for redundant faults. Now let's use the 5-value Algebra described in [3, 4, 15] find the redundant transition delay faults.
G= the set of all nodes which have negative slack.
G={S, B, C, O1, O2, F, S0}.

Now apply a stack-at fault on this nodes (test both for slow-to-rise and slow-to-fall delay fault redundancy). According to [14, 15, 17], the transition delay fault redundancy and stuck-at fault redundancy are different.

We have used a transition delay fault injection method proposed in [15] which is based on the PROOFS stuck-at fault simulator [17]. Transition delay fault means an infinite delay stuck-at fault[14,15, 17]. The transition delay faults are applied at the input signals; this is because we have assumed parallel fault simulation. In parallel fault simulation, the signal values are assumed to be faulty. So that, we applied transition delay fault at each output of the negative slack nodesusing the method proposed in [15] and [17]. According to [16], faults can be injected in to the design either by bit-masking or by inserting extra get in to the design. Here, we demonstrated by inserting extra get at each input. According to [16], AND gate is inserted to simulate slow to rise delay fault by making one input of the AND gate binary-zero and OR gate is inserted to simulate slow-to-fall delay fault by making one input of the OR gate binary-one. Let's see the applied slow-to-rise delay faults in Fig 5. The slow-to-rise delay fault is applied at the output of each G gate, Where G={S, B, C, O1, O2, F, S0}. Ignoring the dummy nodes, i.e. S and S0, we have 5 faulty sites (chosen based on the STA timing violation output). So that, we are going to inject 10 transition delay faults, i.e. 5 slow-to-rise transition delay faults and 5 slow-to-fall transition delay faults and test for redundancy(false path).

Since, they are dummy nodes, we didn't apply stack-at fault at S and S0. As we can see in [16] and [17], slow-to-rise transition delay fault is injected by the addition of K+1-input AND gate for K- clock cycle slow to rise fault. Here we insert 2-input AND gate for 1-clock cycle slow-to-rise transition delay fault simulation. Here, B/0 means slow-to-rise delay fault at node B. The same is true for every element of G, G={S, B, C, O1, O2, F, S0}.

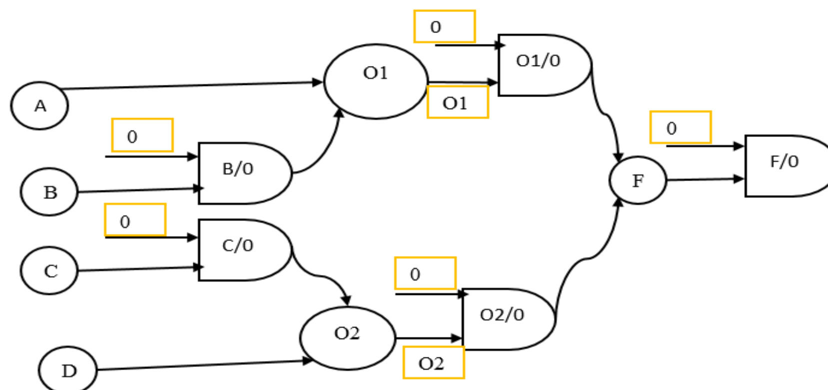

Figure 5: slow-to-rise transition delay faults injected at each negative slack node

Since we use the same fault list for slow-to-fall transition and slow-to-rise transition delay faults, the insertion points for slow-to-fall transition delay fault is also the same as slow-to-rise transition delay fault. So once again, the fault sites are, G={S, B, C, O1, O2, F, S0}. Here, also, we ignore S and S0, because they are dummy nodes.

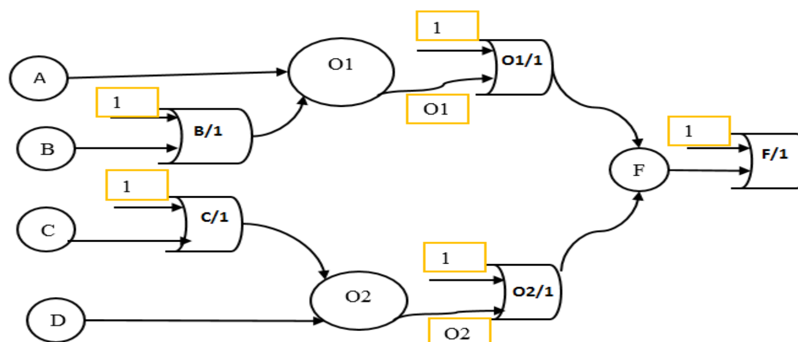

Figure 6: slow-to-fall transition delay fault injected at each negative slack node

Here, all nodes G={S, B, C, O1, O2, F, S0}, are going to be tested for slow-to-rise transition delay fault by inserting a slow-to-rise transition delay fault model at each output by using robust path delay testing criteria. Let's first identify paths which have negative slack nodes.
Path1: S->C->O2->F->S0
Path2: S->B->O1->F->S0
Path3: S->D->O2->F->S0
Path4: S->A->O1->F->S0

All paths have negative slack nodes. To check whether all these are true-paths or false paths, let's check the static sensitizability of the paths by using 5-value Algebra. Assume Path2 is under investigation for now. Assume, B has a slow-to-fall transition delay fault. This is because slack of B is negative (timing violation occurs). Now we are going to check that B is testable (static sensitizable under the 5-valued logic). The steps are explained in [4], but in [4], they didn't use the STA output as a basis to analysis the pessimism.

Step1: Assign B=U1
Step 2: O1=U1=>F=U1=>fault propagation
Step 3: A=U1=> backtracking,using table 1, A is easy to control than O2, so that it has to be determined before O2.
Step 4: F=U1=>O2=S0=>backtracking from Robust test in [3] and [4]
Step 5: O2=S0=> conflict for C and D using Robust test. That means, from propagation table proposed in [3], C and D can have multiple values which is undecided from Robust Delay fault simulation. That means slow to fall transition delay fault at B is redundant and path 2 is a false path under robust delay fault simulation.

Since one fault may not be sensitizable by a certain delay fault testing criteria but may be senstitizable by other testing criteria. Let's take a look at the non-robust test for slow-to-fall transition delay fault. According to [15], a non-robust test becomes invalid if the transition on any non-robust off-input arrives later than the transition on the corresponding on-input. That means the non-robust test becomes invalid when slack of the off path input is negative. Since, the slack of O1 is negative, by non-robust test, we can't verify that O2=S0 or O2=U0. The non-robust transition delay fault simulation described in [3] and [4] pointed out that S0<-U0, and S1<-U1. If we make O2=U0. Still we can't determine the two values i.e. C and D together for value assignments. This implies that Path 2 is a false path under robust and non-robust delay fault criteria considering the transition delay fault at node B. This methodology applies to all paths that have at least one negative slack node. But this claim may be invalidated by fault reduction at node B.

After we get the false path, the critical paths we mentioned using STA Techniques is not real. So we have to Resynthesize our circuit to get a different circuit(in case the circuit may not be in canonical form) or a reduced fault list(using fault reduction techniques). The same, process applied to other paths until we test all negative slack nodes for redundancy (false path), and resynthesize the circuit if we get false paths. False paths can be eliminated from the report by using good synthesis method or algorithm and fault reduction techniques. After resynthesize and fault reduction, the faulty node i.e. B will not be eliminated. Resynthesize and fault reduction just creates the same and simplified circuit in canonical form. In our case, the circuit,$F = AB + CD$ is in canonical form, so that, the redundant slow-to-rise and slow-to-fall fault at node B, comes due-to the fault reduction problem in our gate-level design netlist.

Here the false path and the Pessimism of the STA may be cause by different factors. These are either design error or library design error. If it is a design error or a fault reduction problem, we have to correct our design and correctly reduce our transition fault. If the pessimism is caused by cell library design error, we have to change the library source or redesign our library. So our framework and algorithm used to identify the timing violations caused by the design error.

IV. Experimental Analysis

The high-risk algorithm is implemented by C++ on top of the University of California, Santa Barbara Podem based compiled code transition fault simulator [18, 19]. First, we have changed the Podem algorithm by the High-risk algorithm and the podem based transition table in to the Delay fault based transition table using the five value algebra proposed in [3]. The experimental section has two phases.

Phase 1:- This is STA analysis phase. This analysis produces the path delay fault report in the form of slack. Collect all negative slack nodes from the output file and apply a transition delay fault on those paths on a gate level netlist of the same circuit. We use the TAU2013 Timing Analysis Contest static timing analysis software called IITimer to analyze and find the negative slack nodes. Here we use the ISCAS85 benchmark circuit used TAU2013 Timing Analysis Contest for STA analysis.

TABLE 2: IITimerSTA output and the generated transition delay faults

STA-output file	#nodes violate timing	#Slow-to-rise delay faults injected	#Slow to fall delay faults injected
C17	8	8	8
C432	15	15	15
C499	48	48	48
C880	87	87	87
C1355	58	58	58
C1908	82	82	82

These circuits are ISCAS85 benchmark circuits. But in TAU2013 Timing Analysis Contest, they had used them in STA analysis and we use their report for this simulation. The reason that the numbers in this table are the same for both slow-to-rise faults and slow-to-fall faults is that, we inject and test the same amount of both slow-to-rise and slow-to fall delay faults. That means, as we have stated in the high-risk algorithm, we apply the same amount of both slow-to-rise and slow to fall faults on the high-risk paths twice for each case.

Phase 2: apply the high-risk algorithm to determine whether the path under test is false path or true high-risk path. After getting the negative slack nodes, we parse the negative slack nodes, inject transition delay faults as described above and applied a 5-valued logic to identify false paths and True high-risk paths. After the high-risk paths and nodes are found by STA, we used our tool to analyze the trueness or false ness of the high-risk paths revealed by STA. We used the ISCAS85 benchmark circuits that are used in TAU2013 for STA timing analysis for our Deterministic timing analysis purpose.

TABLE 3: the output of high-risk algorithm

Circuit name	#Total transition delay faults	#Detected transition faults by Random vector	Fault coverage by random vector (%)	CPU time by random vector	#Detected sequential transition delay faults by ATPG	Fault coverage by transition fault vector (%)	CPU time by High-Risk	#Undetected faults	#False path fault
C17	34	34	100	0.00	23	67.65	0.00	11	0
C432	1110	202	18.20	0.04	5	0.45	0.02	1105	908
C499	2390	2308	96.57	12.65	2171	90.48	0.08	219	16
C880	2104	1203	57.18%	0.23	971	46.15	0.05	1133	901
C1355	2726	1724	63.24	0.73	263	9.65	0.06	2463	1002
C1908	3816	2196	57.55	0.86	615	16.12	0.07	3201	1620

In this table, we present the simulation output of the high-risk algorithm. The algorithm applies high-risk faults only on high-risk nodes only. We identified the number of transition fault redundant paths i.e. false paths in the last column of table 3. Using these false paths, we can either fault reduce the circuit or resynthesize it to get the canonical number of faults that give the true high-risk paths. From table 3, we can observe that, the CPU overhead due to random pattern and due to High-risk algorithm is different. High-risk algorithm injects only small amount of high-risk faults on those high-risk paths mentioned in table 2 only, so that, it takes an almost zero CPU time. But the random vector based algorithm, injects a random amount of slow-to-rise and slow to fall faults, so that it takes a longer amount of CPU time. Let's take a look at the C1908 benchmark circuit. Using the random based algorithm, the fault injector injects a random number of faults and a random amount of test vector on those faults so that it takes 0.86 CPU time. But using the high-risk algorithm on C1908, it injects only 82 slow to-rise and 82 slow to fall faults and test only those faults at a time, so that it takes only 0.07 CPU time. This implies that, testing only high-risk path delay faults saves a significant amount of CPU time.

V. Conclusion

If a certain designer assisted with both STA and dynamic timing delay fault simulation, we can get a better and quick result. The STA simulation includes the design gate-level netlist and cell library simulation together. So, we don't know, whether the STA pessimism is from the Library or from our design gate-level netlist. So using the dynamic delay fault simulation, we can verify whether false paths exist in the design, so that we can identify from where the false path reports in the STA came from. If the design has a false path, we can correct it, if the library has a problem that creates the false path, we can change the library or we can redesign it if we can. Since, circuit density is increasing, and size is decreasing, in the future, we will extend our method to Statistical sense.

References

[1]. Tsung-Wei Huang,et.al, "UI-Timer: An Ultra-Fast Clock Network Pessimism", IEEE, 2014
[2]. Tsung-Wei Huang, "OpenTimer: A High-Performance Timing Analysis Tool", IEEE, 2015
[3]. CHIN JEN LIN and SUDHAKAR M. REDDY, "On Delay Fault Testing in Logic Circuits", 1987 IEEE
[4]. michael l. bushnell, essentials of electronic testing for digital, memory and mixed-signal vlsi circuits(Kluwer Academic Publishers, 2002)417-439

[5]. T. 1. Kirkpatrick N. R. Clark, “PERT as an Aid to Logic Design”, IBM JOURNAL * MARCH 1966
[6]. John Lillis, Chung-Kuan Cheng, Ting-Ting Y. Lin and Ching-Yen Ho, “New Performance Driven Routing Techniques With Explicit Area/Delay Tradeoffand Simultaneous Wire Sizing*”, 1996 ACM
[7]. J. Bhasker and Rakesh Chadha, static timing analysis for nanometer designs a practical approach(Springer Science+Business Media, LLC, 2009)
[8]. VIJAY S. IYENGAR,“On Computing the Sizes of Detected Delay Faults”, IEFE TRANSACTIONS ON COMPUTER AIDED DFSIGN VOL 9 NO 7 MARCH 1990
[9]. Gordon L. Smith, “Model For Delay Faults Based Upon Paths” , International Business Machines Corporation
[10]. Jing-JiaLiou, Angela Krstic, Li-C. Wang and Kwang-Ting Cheng, “False-Path-Aware Statistical Timing Analysis and E,fficient Path Selection for Delay Testing and Timing Validiation”, DAC 2002, June 10-14,2002, New Orleans, Louisiana, USA.
[11]. Shihheng Tsai and Chung-Yang (Ric) Huang, “A False-Path Aware Formal Static Timing Analyzer Considering Simultaneous Input Transitions”, DAC’09, July 26-31, 2009, San Francisco, California, USA
[12]. Chunyu Wang, “Common path pessimism removal in static timing analysis”, MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY, Missouri, Summer 2015
[13]. Kurt Keutzef, SharadMaliktAle.xanderSaldanhd, “Is Redundancy Necessary to Reduce Delay?”,27th ACM/IEEE Design Automation Conference, 1990
[14]. John A. Waicukauski, Eric Lindbloom, Barry K. Rosen, and Vijay S. Iyengar, “Transition Fault Simulation”, 1987 IEEE
[15]. Angela Krstic, and Kwang-Ting (Tim) Cheng, delay fault testing for vlsi circuits (SPRINGER SCIENCE+BUSINESS MEDIA, LLC, 1998)
[16]. Thomas M. Niemann, Wu-Tung Cheng Jan& H. Pate, “PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator”, 1990 IEEE
[17]. Shetye K, “Transition Delay Faults”, TERM PAPER, ELEC 7250, SPRING 2004
[18]. Ernst G. Ulrich, Vishwani D. Agrawal and Jack H. Arabian, concurrent and comparative discrete event simulation(SPRINGER SCIENCE+BUSINESS MEDIA, LLC, 1994
[19]. Peter M. Maurer, “Levelized Compiled Code Multi-Delay Logic Simulation”, TMS/DEVS 2015, April 12 - 15, 2015, Alexandria, VA

Tadele Basazenew Ayele. “Identification of High-Risk Hardware Path-Delay Fault Locations and Evaluation of Their Impact.” IOSR Journal of VLSI and Signal Processing (IOSR-JVSP) , vol. 7, no. 4, 2017, pp. 38–47.