False Timing Path Identification Using ATPG Techniques and Delay-Based Information

Jing Zeng, Magdy Abadir
High Performance Tools and Methodology
Motorola Inc.
Austin, TX78729
Jing.Zeng,M.Abadir@motorola.com

Jacob Abraham
Computer Engineering Research Center
University of Texas at Austin
Austin, TX78712
jaa@cerc.utexas.edu

ABSTRACT

A well-known problem in timing verification of VLSI circuits using static timing analysis tools is the generation of false timing paths. This leads to a pessimistic estimation of the processor speed and wasted engineering effort spent optimizing unsensitizable paths. Earlier results have shown how ATPG techniques can be used to identify false paths efficiently [6],[9], as well as how to bridge the gap between the physical design on which the static timing analysis is based and the test view on which ATPG technique is applied to identify false paths [9]. In this paper, we will demonstrate efficient techniques to identify more false timing paths by utilizing information from an ordered list of timing paths according to the delay information. More than 10% of additional false timing paths out of the total timing paths analyzed are identified compared to earlier results on the MPC7455, a Motorola processor executing to the PowerPC^{TM} [1] instruction set architecture.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: --Computer-aided design; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; F.2.3 [**Analysis of Algorithms and Problem Complexity**]: Tradeoffs between Complexity Measures; B.6.3 [**Logic Design**]: Design Aids–Optimization

General Terms

Algorithms, Performance

Keywords

static timing analysis, false timing paths, ATPG, timing slack

1. MOTIVATION

Static timing verification is a crucial step in the design methodology of high performance VLSI circuits. It handles the exponential number of paths in the circuit using efficient linear structural algorithm which ignores the circuit functionality. It is generally used iteratively to allow the timing issues with a certain number of paths to be fixed before it is run again. The cost of optimization and iteration is expensive and optimizing unsensitizable paths does not help to improve the performance. Thus effective techniques for identifying false paths and shortening the cycle of the iterative process are needed.

There have been different approaches at solving the false path identification(**FPI**) problem. Chen and Du used satisfiability(SAT)-based algorithms to check if the sensitizability functions can be satisfied[5]. Chang and Abraham proposed a path sensitization method[4]. Once false path sections have been identified, there are algorithms identifying them from the timing graph [2] [3]. Binary Decision Diagram(BDD)-based model checking and bounded SAT techniques were used at resolving false paths in synthesized logic blocks in [7]. In [1], ATPG techniques are used to remove false paths during timing analysis while expensive circuit modification technique is performed to resolve re-convergent fanout. In [8], a path delay fault identification prototype tool is used to determine circuit stabilization time. We make use of commercial ATPG tools at our design center. The tools can handle very large design models on the order of hundred million transistors.

Earlier results in [6] [9] demonstrate the effectiveness of ATPG techniques. Since they are not BDD-based, the problems with BDD blow-ups while analyzing timing paths for a full chip are not encountered.

Section 2 describes our terms, assumptions and our previously developed FPI technique. We then present our new FPI technique which utilizes information from timing report. We present our experimental results in Section 4, and our conclusions in Section 5.

[1]PowerPC is a trademark of the International Business Machines Corporation.

2. INTRODUCTION

2.1 Terms and Definitions

A **critical timing path** (P) is characterized by a set of n nodes x_1,x_2, ..., x_n and a set, $T = \{t_1, t_2, ..., t_n\}$, of signal transitions such that $t_i \in T$ represents the signal transition on x_i. We call nodes x_1, x_2, ..., x_n **on-nodes** for path P. If a circuit node connects to the same circuit element as x_i, but is not on path P, it is called a **side-node** of path P. Each transition t_i is characterized by a pair of booleans $< b_i, a_i >$ where b_i and a_i are the initial (or ***b***efore) and final (or ***a***fter) boolean values at node x_i, respectively. Note that b_i and a_i are always complementary to each other. We call the set $\{b_1, b_2, ..., b_n\}$ the **before** set and the set $\{a_1, a_2, ..., a_n\}$ the **after** set and test for their satisfiability.

Timing paths start from primary inputs or outputs of sequential elements, often called the **launch points**, and end with primary outputs or inputs to sequential elements, which are called the **capture points**. Timing paths with outputs of sequential elements as their launch points and inputs of sequential elements as their capture points are called **latch to latch timing paths**.

A path is **sensitizable** if a two-pattern test (v_1, v_2) activates a transition at its launch point and propagates to its capture point. We call the time frame during which v_2 is applied the **current time frame**, while that during which v_1 is applied the **previous time frame**.

Timing paths are generally listed in the timing reports based on their timing slacks. **Timing slack** is defined as the required arrival time minus the actual arrival time at a capture point of a timing path. Static time analysis can be configured to run so that only the single worst case timing path is generated for each capture point in the circuit. We call these paths **main paths**. Main paths can be ordered based on their timing slacks at the capture points with the path with the worst slack showing up first in the report. Besides the main path for a particular capture point, paths converging to the same capture point, with differences in timing slacks from that of the main path within a given threshold, can be generated. These paths are called **subpaths**. Subpaths and their corresponding main path form a group of **converging paths**.

For the scope of this paper, we restrict our analysis to identifying false timing paths which violate logic constraints within the combinational blocks in a design. We will be using the terms *timing paths* and *paths* interchangeably.

2.2 Previous Work: FPI using Logic Info

Previously developed FPI techniques took into consideration of the following logic conditions.

1. e_a = True iff x_i = a_i, for all i can be justified simultaneously when evaluated in the current time frame.

2. e_n = True iff respective non-controlling values can be assigned simultaneously at all side-nodes in the current time frame.

3. e_b = True iff x_i = b_i for all i can be justified simultaneously in the previous time frame.

The subscripts of e_a, e_n and e_b indicate the evaluation of different criteria. e_a describes the *e*valuation of *a*fter value criteria. e_n describes the *e*valuation of *n*on-controlling value criteria while e_b describes the *e*valuation of *b*efore value criteria.

The following algorithm, which we will refer to as algorithm *A*, was presented in [6], [9].

```
Given a path $P$ with $e_a$, $e_b$ and $e_n$
if $e_a = false$, then
    $P$ is a false path($f_a$)
else
    if $e_n = true$ then
        $P$ is an active critical path
    else if
        $e_b = false$ where $e_n = false$
        then $P$ is a false path($f_b$)
    else $P$ is an active critical path
```

The labels in parenthesis represent the group of paths which are identified false at a particular step in the algorithm. This notation will be used to describe other algorithms later on. We also call the *false paths* identified in algorithm A *logically false paths* since only logic information is utilized for false path identification. If a path is not identified as logically false, we call it a *logically active path*.

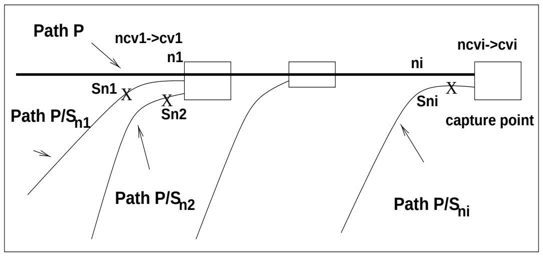

Figure 1: General Multiple Race Condition

3. SLOW PATH CONDITION: FPI USING LOGIC AND DELAY INFORMATION

Due to reconvergent fanouts in the circuits, race conditions between timing paths exist. In a **race condition**, more than one inputs of a gate on the path under consideration(PUC) transition from NCV(non-controlling value) to CV(controlling value). Here e_a and e_b are satisfied for the inputs/output of the gate, but e_n is violated. The transition at the output of the gate is associated with that of the input with earlier timing than others. Unless the arrival times of the transitions associated with the side-nodes of a gate are slower than that associated with the on-node, path PUC is not sensitized.

Any input of the gate in a race condition which is a side-node of the PUC is called S_{n_i}. All the S_{n_i} for the PUC form a set **SN**. The path which forms race condition with PUC at S_{n_i} is called P/S_{n_i}. All the S_{n_i} corresponding to P/S_{n_i} with longer delay than the PUC form a set **SSN**. *SSN* is a subset of *SN*.

One way to decide whether a PUC with race conditions is sensitized, is to keep track of the transition arrival time at each of its on-node and S_{n_i} node. This can be an expensive approach. It is more efficient to utilize the delay information of converging timing paths from the timing report. So for a timing path *P*, which may or may not be a main path, we have the following.

Theorem I. A timing path *P* with race condition, is false if there is not a logically active P/S_{ni} of longer delay than P at any S_{ni}.

Proof: In Figure 1, an “x” indicates a S_{n_i} node.

Take for example, two side-nodes S_{n1}, S_{n2} and the corresponding on-node n_1 in a race condition. If either of the two side paths P/S_{n1}, P/S_{n2} corresponding to S_{n1}, S_{n2} has been identified as logically false, then it would not help to sensitize the transition of path P. If either of the two side paths P/S_{n1}, P/S_{n2} corresponding to S_{n1}, S_{n2}, is of shorter delay than path *P*, it would kill the propagation of the transition on node n_1.

The same condition needs to be true for all S_{ni}s *simultaneously*, otherwise P is a false path $\square$

We call the FPI condition in Theorem I the *Slow Path Condition*.

The comparison of timing slacks of paths depends on how accurately delays are estimated. A threshold is needed to estimate how close the slacks of two timing paths can be and still be differentiated. In a race condition, if one timing path has a slightly better slack than another one, it can activate the other one, if it ends up with worse timing in real silicon. We take this into consideration in our experiments. The false paths identified based on the slow path condition are not logically false paths. Whether they can be sensitized depending on the delays of other logically active paths.

To check Theorem I, we can identify the exact locations of race conditions where e_a, e_b are satisfied, but e_n is violated. We can then check if there are logically active paths of longer delays associated with these nodes. This can be expensive. Instead, we identify the upper bound of the locations of race conditions while allowing the PUC sensitization using efficient structural analysis as follows.

Corollary I. A timing path P with race conditions is false if race conditions happen at nodes which are not intersections between path P and every one of its converging paths with longer delays.

The converging paths of P with longer delays can be identified from the delay information in the timing report. Their intersections with path P can be identified by comparing path nodes. Corollary I avoids explicitly stating exactly where the race conditions are, but provides an upper limit for them for path P to be sensitized.

We can make the upper bound of race condition locations tighter and further identify false paths. If a converging path P' with longer delay is logically false, then its associated intersection with path P cannot be a location for race condition. A path P with race conditions is false if none of its converging paths with longer delays is logically active. Thus we have Corollary II.

Corollary II. A timing path P with race condition(s) is false if none of its converging paths with longer delays is logically active.

We call the FPI condition defined by Corollary I and II a *revised slow path condition* since it does not attempt to identify the exact locations of race conditions, only the upper bound. The identified upper bound is called **PRN**.

Note if a path P violates e_n, it could, 1). violate e_a; 2). violate e_b where e_n is violated; 3). have a race condition. For the first two items, P is false by definition. In general, if e_s and e_n can be checked to eliminate the third possibilty, then P is false whatever the exact cause of its being false. This allows earlier and more efficient identification of false paths.

A main path with a race condition violates Corollary I.

Corollary III. A main path P with a race condition is false.

3.1 Improved FPI Algorithm

Let e_s = True iff the set *PRN* is non-empty. e_s describes the *e*valuation of the revised *s*low path condition. Given a path P with e_a, e_s, e_n and e_b, Figure 2 shows our improved algorithm *B*. The main advantages over algorithm A are the following.

- Additional false paths(f_s and f_n) are identified.
- The checking of the satisfiability of e_n and e_a simultaneously makes sure that a specific transition can be sensitized.
- The locations where e_b is checked are identified more efficiently using the corollaries which do not take additional ATPG run time, compared to being checked exactly where e_n is violated in algorithm A.
- The checking of e_n after e_s is done at a node $\notin$ PRN which is identified efficiently using the corollaries, while e_n is checked everywhere in algorithm A.
- Paths which fail e_b can be identified false as early as during the checking of e_s.

3.2 A More Efficient FPI Algorithm

The checking of e_a, e_n and e_s in algorithm B can be done either in parallel or in serial where only the paths not identified false by e_a need to be checked further. Either way it requires duplication of

```
if $e_a$ = false, then P is false($f_a$)
else
    if $e_a$ and $e_n$ are true, then P stays active
    else
        if $e_s$ = false, then P is false($f_s$)
        elsif $e_n$ = false at node $x_i \notin$ PRN
            then P is false($f_n$)
        elsif $e_b$ = false at node $x_i \in$ PRN
            then P is false($f_b$)
        else P stays active
```

Figure 2: Algorithm B

```
if $e_a$ and $e_n$ are true,  then P stays active
else
    if $e_s$ = false, then P is false($f_s$)
    elsif $e_n$ = false at node $x_i \notin$ PRN
        then P is false($f_n$)
    elsif $e_a$ = false  then P is false($f_a$)
    elsif $e_b$ = false at node $x_i \in$ PRN
        then P is false($f_b$)
    else P stays active
```

Figure 3: Algorithm C

resources or a lot of run time of an ATPG tool. To minimize the amount of logical checking, we observe the following.

Observation: the set of timing paths which fail the simultaneous satisfaction of e_a and e_n is the upper bound of all the false timing paths we identified in algorithm B.

With e_a, e_s, e_n and e_b, we propose algorithm C(Figure 3) for path P with the following main efficiency over algorithm B.

- The upper bound of all the false paths is identified by e_a and e_n. This reduces the amount of paths that need be checked by further logic conditions, especially e_a. As in algorithm B, the upper bound is further reduced by checking e_s.

Note in algorithm B, we check e_s on the timing paths which fail e_a and e_n together, but not e_a by itself. Now we perform the check on *ALL* the timing paths which fail e_a and e_n. This tradeoff is minimal for structural analysis. Also, paths which fail e_a by itself and e_b can be identified as early as during the checking of e_s.

4. Experimentation Results

A circuit made of boxes and RC networks has been analyzed. The timing behaviors of the boxes are precharacterized under numerous environmental parameters using transistor-level simulation. The RC nets are precharacterized for the estimation of interconnet timing behavior using extraction tool. Timing analysis is then performed on the whole chip. The output of the timing analysis consists of a set of critical paths.

Our FPI engine translates timing paths into delay paths for ATPG tools[9]. It checks the satisfiability of different criteria for the path by setting the corresponding values at the nodes along the path simultaneously using the ATPG tool commands. Based on the status returned after running ATPG tool with command files, we use our

latch to latch timing paths	Algorithms									FPI improv of B,C over A
	A	B				C				
	fa	fa	fs	fn	fb	fs	fa	fn	fb	
61	6	6	9	5	0	12	3	5	0	3.3
332	26	26	36	6	0	54	8	6	0	2.6
566	54	54	56	6	0	95	15	6	0	2.1

Figure 4: False Path Identified using Different Algorithms

FPI engine again to identify a list of false paths. Additional command files are fed to the ATPG tool and the log files from the ATPG tool are analyzed for further logic checking.

# of Transistors	# of IO pins	# of latches
33 million	281	90k

Table 1: Statistics for MPC7455

We ran our experiments on MPC7455 microprocessor. All runs were performed on a 400MHz Ultra60 running Solaris 5.6 with 1GB memory. Three sets of most critical timing paths were generated using the timing analyzer. We simplified the issue by analyzing only most critical latch to latch timing paths, but it is straightforward to extend the analysis to other kinds of paths. The cycle time target is 950ps. The threshold for generating subpaths was 3ps. The threshold for differentiating between the timing slacks of two timing paths was set to be 0.01ps. Any two timing paths with timing slacks of less than 0.01ps difference are considered to be of comparable delay. False paths identified in different groups following the convention described earlier are shown in Figure 4.

Our additional checking of f_s and f_n was effective in identifying false paths. This highlights the need of taking into consideration delay-based information of the paths besides logic value justification when performing FPI. The last column in Figure 4 shows the number of false paths identified using algorithms B and C, both utilizing f_s and f_n, over algorithm A. At least twice as many false paths are identified using f_s and f_n, or at least 10% additional false timing paths out of the total timing paths analyzed in all three data groups. Note only f_a is identified in algorithm A since identifying f_b without structural analysis information is expensive.

We can see that the number of false paths identified using algorithm C changed from those identified using algorithm B. e_s identifies false paths under category e_a in algorithm B efficiently during the structural analysis. This results in the number under the f_a using algorithm C being much less than that using algorithm B. The number of timing paths needing to be checked using f_a is also reduced. In general, algorithm C performs much less logic checking than algorithm B, thus algorithm C takes much less time than algorithm B since the run time of the algorithms mainly come from running ATPG tool for logical checking. The run time for the 3 sets of timing paths is listed in Table 2. The ATPG abort limit is set to 100. We can see algorithm C takes comparable amount of time to algorithm A, which is about 50% of what it takes for algorithm B. With a set abort limit, in the worst case, the overall run time scales linearly with the total number of paths. If the logic checking for most timing paths takes much less time than the abort limit, the ratio of overall run time for two sets of timing paths can be smaller than the ratio of timing path number. From Table 2, we see between the first two data groups, the ratio of timing path number is 5.4(332/61), while the ratio of run time is 2.3. The same is true for comparison between all the data groups. The loading of the gate-level model takes around 1 hour is not included in the run time since it is an one-time cost and is the same for all algorithms.

latch to latch timing paths	A(mins)	B(mins)	C(mins)
61	30	63	33
332	68	141	73
566	94	195	101

Table 2: Comparison of Algorithms

5. Conclusions

New techniques are demonstrated to perform FPI utilizing additional delay-based information of the timing paths. We are able to identify at least 20% of the total timing paths within the different data groups as false. The new techniques can efficiently identify at least 10% more false paths out of all the timing paths compared to the earlier results. We experimented with a single threshold value for differentiating between slacks of timing paths for this paper. More threshold values will be experimented with for future experiments. Their impacts on the number of false timing paths identified and run time will be explored. So far we have worked on identifying false paths using scan test vectors. Future work will include identifying functionally false paths taking into consideration of the sequential/functional behavior of the circuits.

4. REFERENCES

[1] P. Ashar, S. Malik. "Functional timing analysis using ATPG", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 1025-1030, Aug. 1995.

[2] K. P. Belkhale, A. J. Suess. "Timing Analysis with known False Sub-graphs" in *Proceedings of ACM/IEEE International Conference on Computer-Aided Design*, pp. 736-739, 1995.

[3] D. Blaauw, R. Panda and A. Das. "Removing user-specified false paths from timing graphs", in *Proceedings of 36th ACM/IEEE Design Automation Conference*, pp. 270-273, 2000.

[4] H. Chang and J. Abraham "VIPER: An Efficient Vigorously Sensitizable Path Extractor"in *Proceedings of 30th ACM/IEEE Design Automation Conference*, pp. 112-117, 1993.

[5] H. C. Chen, D. and H. C. Du. "Path Sensitization in Critical Path Problem," in *Proceedings of IEEE International Conference on Computer-Aided Design*, pp. 208-211, 1991.

[6] J. Bhadra, M. S. Abadir, J. A. Abraham. "A Quick and Inexpensive Method to Identify False Critical Paths Using ATPG Techniques: an Experiment with a PowerPC Microprocessor" in *Proceedings of IEEE 2000 Custom Integrated Circuits Conference*, pp. 71-74, May. 2000.

[7] R. Raimi and J. A. Abraham. "Detecting False Timing Paths: Experiments on PowerPC(TM) Microprocessors", in *Proceedings of 36th ACM/IEEE Design Automation Conference*, pp. 737-741, 1999.

[8] A. Sivaraman, A. J. Strojwas. "Timing Analysis Based on Primitive Path Delay Fault Identification", in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 182-189, 1997.

[9] J. Zeng, M. S. Abadir, J. Bhadra, J. A. Abraham. "Full chip false timing path identification: applications to the powerPC microprocessors " in *Proceedings of Design, Automation and Test in Europe*, pp. 514-518, 2001.