# Full Chip False Timing Path Identification: Applications to the PowerPC$^{TM}$ Microprocessors

Jing Zeng[†‡], Magdy S. Abadir[†], Jayanta Bhadra[†‡], and Jacob A. Abraham[‡]
[†]EDA Tools and Methodology,
Motorola ASP Somerset Design Center, Austin, TX 78729, U.S.A.
[‡]Computer Engineering Research Center,
The University of Texas at Austin, Austin, TX 78712, U.S.A.

## Abstract

*Static timing analysis sets the industry standard in the design methodology of high speed/performance microprocessors to determine whether timing requirements have been met. Unfortunately, not all the paths identified using such analysis can be sensitized. This leads to a pessimistic estimation of the processor speed. Also, no amount of engineering effort spent on optimizing such paths can improve the timing performance of the chip. In the past, we demonstrated initial results of how ATPG techniques can be used to identify false paths efficiently[1]. Due to the gap between the physical design on which the static timing analysis of the chip is based and the test view on which the ATPG techniques are applied to identify false paths, in many cases only sections of some of the paths in the full-chip were analyzed in our initial results. In this paper, we will fully analyze all the timing paths using the ATPG techniques, thus overcoming the gap between the testing and timing analysis techniques. This enables us to do false path identification at the full-chip level of the circuit. Results of applying our technique to the second generation G4 PowerPC$^{TM}$ will be presented.*

## 1. Introduction and motivation

Static timing analysis is an important component of timing verification of high speed, high performance ASIC designs such as microprocessors. The facts that it does not need a supply of vectors and that it considers all possible paths make it an excellent choice for this purpose. The inputs to it are the circuit models, the information about the circuit primitives and macros along with their pin-to-pin delays. The circuit models are extracted from the layouts taking into consideration of noise and interconnect delays in the circuits.

However, since most of the static timing analyzers do not consider the dynamic behavior of the circuit, they might report paths as critical when such paths are not sensitizable. Usually a critical path is defined to be one which does not meet the timing requirement of the circuit and is characterized by boolean transitions on all the nodes along the path. In this paper the definition of false critical path is the same as in [1]: a critical path for which the specified signal transitions at the nodes on the path cannot happen for any input combinations.

Usually an iterative approach is used by the designers where only a small percentage of the slowest structural critical paths are identified, analyzed and optimized. In each iteration, timing analysis is performed to identify a new set of critical paths for optimization. The number of paths identified may be very large, normally in the hundreds or sometimes thousands. The amount of engineering efforts involved in speeding up such a non-trivial number of paths is huge, and not necessarily well spent since many of the paths are not sensitizable. Previous research [2] indicated that for most circuits a good percentage of structural critical paths are not sensitizable. A technique, which is effective in identifiying the false paths, is needed to shorten the cycle of this iterative process. Once the false paths are identified, the designers can ignore them from the list of paths to be optimized. In [1] we demonstrated such a technique by using simple ATPG techniques. In this paper we show that the analysis can be done on full-chip circuit models and demonstrate its effectiveness by applying it on a state of the art custom-made microprocessor.

Different approaches have been attempted at solving the false path detection problem. [3],[4] used satisfiability-based algorithm and heuristics to check if the sensitizability functions can be satisfied or not. In [5] a path sensitization method is proposed. Work has been done at the transistor level [8] [9]. Also, once the false paths have been identified,

there are algorithms to identify them from the timing graph [10] [11]. In [6] and [7], model checking and bounded model checking techniques are attempted at resolving false paths in synthesized logic blocks in a chip. We make use of a commercial ATPG tool at our design center which can handle design models of size close to the chip level. Since our approach is not binary decision diagram(BDD)-based, we do not encounter BDD blowups while analyzing timing paths for a full chip.

Our initial results [1] demonstrated the effectiveness of our technique, though they were limited by the different approaches used in test generation and in timing analysis. Gate-level model is used for test generation with all the different design hierarchies flattened to a handful of gate-level primitives. On the other hand, transistor-level simulation for the custom blocks is needed for accurate timing to get worst case pin-to-pin delay estimation with different transition, loading types at the pins. A timing analyzer treats these pre-characterized design blocks as functional black-boxes and indicates the transitions on the pins of these design blocks on the timing paths. Unfortunately, many of these design blocks in timing analysis are of higher-level than the gate-level primitives understood by the ATPG tool, thus the ATPG tool does not understand the pins of these design blocks directly. We implemented a path extractor which analyzes all the custom designs treated as black-boxes by the timing analyzer and extracts all the paths associated with pin to pin pairs of a design block.

In the next section we provide a review of our false path detection methodology. We will also describe the gap between test generation and timing analysis technologies and our approach to overcome it in section 3. In section 4, we present our experiments and results. Section 5 concludes the paper.

## 2. Detection of false critical paths

A critical timing path ($P$) is characterized by a set of $n$ nodes $x_1, x_2, \ldots, x_n$ and a set, $T = \{t_1, t_2, \ldots, t_n\}$, of signal transitions such that $t_i \in T$ represents the signal transition on node $x_i$. 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, since we are concerned with the signal transition on every node along the path. We call the set $\{b_1, b_2, \ldots, b_n\}$ as the $Before$ set and the set $\{a_1, a_2, \ldots, a_n\}$ as the $After$ set and test for their satisfiability.

For example, in the following figure, there are 8 library cells and/or custom macros in the circuits, nodes $x_1$, $x_2$, ..., $x_7$ go through transitions $< 0,1 >$, $< 1,0 >$, $< 1,0 >$, $< 1,0 >$, $< 0,1 >$,$< 1,0 >$ and $< 1,0 >$.

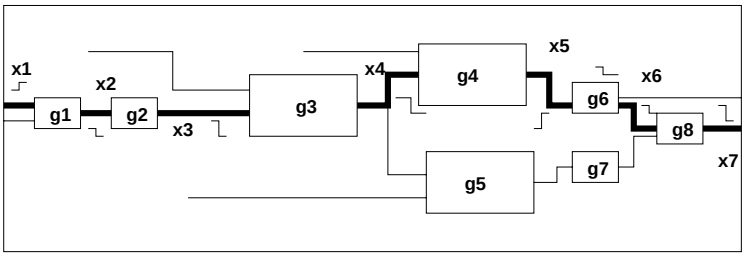


**Figure 1. Critical timing path**

The timing paths are considered to start from the primary inputs or latch outputs, often called the launch point, and end with the primary outputs or input to latches, or called the capture points. Each node $x_i$ is either the primary input or output of a latch, circuit primitive or custom macro. Each node depends combinationally upon a set of latches and primary inputs. Let the boolean function $f_i$ represent the cone of logic which feeds the node $x_i$. For any node $x_i$ to have transition $t_i$, the corresponding $f_i$ needs to assume values $b_i$ and $a_i$, respectively, in two successive time instances. Thus in order to test for a whole combinationally false path, the two boolean functions $(f_1 = b_1) \wedge (f_2 = b_2) \wedge \ldots \wedge (f_n = b_n)$ and $(f_1 = a_1) \wedge (f_2 = a_2) \wedge \ldots \wedge (f_n = a_n)$ need to be satisfied. Failing to satisfy the second function implies that $P$ is combinationally false, but it is not necessarily true for the first function.

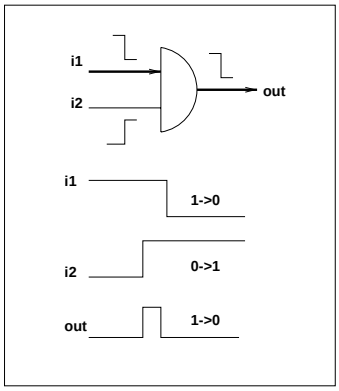


**Figure 2. Effect of side-inputs**

For example, for a 2-input AND gate, having inputs $i_1$ and $i_2$ and output $out$. Let us take the critical path $\{i_1, out\}$ and the $1 \rightarrow 0$ transition on both $i_1$ and $out$. If $i_1$, which is on the critical path, is undergoing a $1 \rightarrow 0$ and the side input, $i_2$, is undergoing a $0 \rightarrow 1$ transition then the above first function is not satisfied. But if the transition on $i_2$ happens before the transition on $i_1$ (which is a possibility since the path $\{i_1, out\}$ is the critical path) then there exists a functional test for this delay path.

To account for all such cases, we test the simultaneous

satisfiability of the values in the *Before* set unioned with the *non-controlling* values on the side inputs of all the gates on the path.

If we let:

1. $e_a$ = True iff $x_i = a_i$, for all $i$ can be justified simultaneously
2. $e_b$ = True iff $x_i = b_i$ for all $i$ can be justified simultaneously
3. $e_n$ = True iff respective non-controlling values can be assigned simultaneously at all side-inputs.

Thus the following is our algorithm:

Given a path $P$ with $e_a$, $e_b$ and $e_n$
if $e_a = false$, then
    $P$ is a false path
else
    if $e_n = true$ then
        $P$ is a true critical path
    else if
        $e_b = false$ then $P$ is a false path
    else $P$ maybe a true critical path

This takes care of those paths which do not satisfy the *Before* sets but have functional tests for them.

Note that our algorithm is also general enough to consider the kind of paths which satisfy both $e_a$ and $e_b$, but not $e_n$.

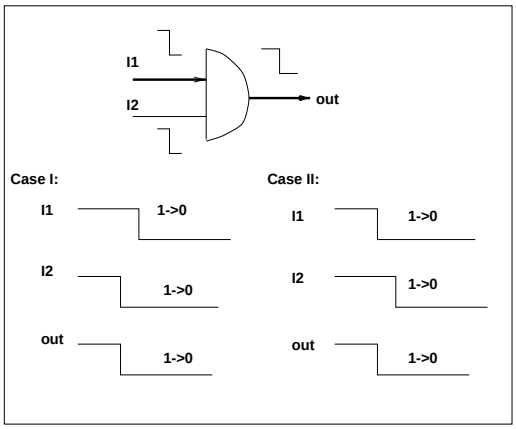


**Figure 3. Path considered for timing purpose**

For the $AND$ gate in the figure 3, even though the transition obtained at the output of the gate is not necessarily associated with the transition of the input on the path of consideration(POC, indicated by the arrowed line), it still helps the performance of the circuit to speed up the POC. There are two cases here. In case I, the POC *IS* the critical path, even though the transition propagated is associated with the side path. Optimizing POC would allow propagation of the transition on POC to the output of the gate and allow the timing of the path to be met. In case II, even though POC is not the most critical path since it meets the timing earlier than its side path, it is still a critical path and needs optimization.

Based on our algorithm, only those critical paths which are truly false are safely removed. Note that in this paper we are concerned with only combinationally false paths.

## 3. Methodology

A reasonably accurate way of estimating the performance of a processor is through the transistor-level simulation. Unfortunately transistor-level simulation is prohibitively expensive for large designs. The realistic approach in the industry is to perform circuit-level simulation on all the custom circuits(including gate primitives and custom macros), characterizing the worst case pin to pin delays for all of them, then the structural timing analysis is performed on the whole processor with the pre-characterized custom blocks treated as black-boxes.

To identify a false path identified by a timing analyzer, we can check the satisfiability of $e_a$, $e_b$ and $e_n$ for the path by setting the corresponding values at the nodes along the path simultaneously using the commands in the ATPG tool.

There are 4 kinds of return status after running ATPG tool:

1. a set of node values is returned satisfying the condition under check.
2. abort.
3. redundant.
4. ATPG_untestable

In case 3 and 4, the ATPG tool finds the logic expression unsatisfiable either due to the redundant nature of the logic or under the given constraints. In either of these cases, the POC is not sensitizable and thus can be eliminated from consideration safely.

Most of the pre-characterized design blocks are not gate level primitives understood by the ATPG tool. To specify the nodes on the POC which are ports for these design blocks, we analyze the gate-level models for the blocks to figure out the gate primitives inside the blocks which are connected to these ports.

We implemented a path extractor which analyzes all the custom designs and extracts all the paths associated with a pin to pin pair for a custom design.

For example, for the design block in the figure 4, which is not a gate-level primitive. It contains gate-level primitives $I_1$, $I_2$ and $I_3$ along with inputs $IN_1$,$IN_2$ and output $OUT$. The design block would then be specified as:

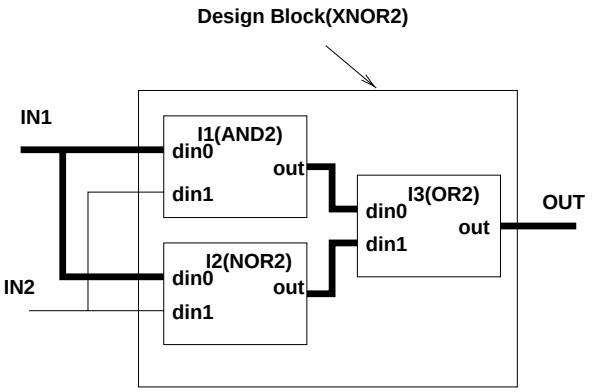


**Figure 4. Ports of custom design blocks**

1. $IN_1$: /I1/din0, where $din0$ is an input port for $I1$.
2. $IN_2$: /I2/din0, where $din0$ is an input port for $I2$.
3. $OUT$: /I3/out, where $out$ is an output port for $I3$.

There are 2 paths between $IN1$ and $OUT$, one is activated when $IN2$ is 0, the other when $IN2$ is 1. Our path extractor would extract both.

The flow of our methodology is shown in the figure 5. We use the set of critical paths identified by the timing analyzer as the starting point of our false paths identification technique.

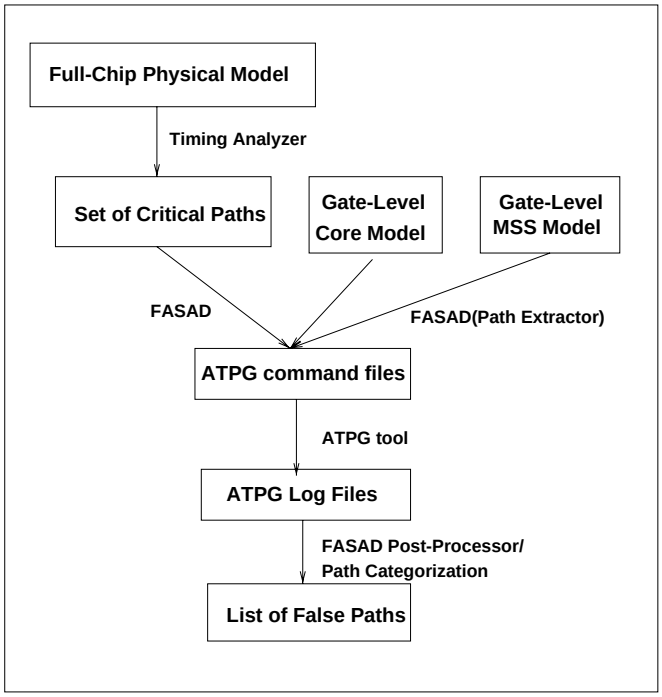


**Figure 5. The flow in the method**

## 4. Experiments and results

We ran our experiments on the second G4 PowerPC$^{TM}$ microprocessor with its statistics shown in the following table.

**Table 1. The Second Generation G4 PowerPC$^{TM}$ Microprocessor Statistics**

| # of transistors | # of signal pins | # of LSSD latches |
|---|---|---|
| 33 million | 484 | 91263 |

Due to the size of the design, separate core and memory test models were created for this chip even though the structural timing analysis was carried out at the chip-level. We utilized both models identifying false paths in the full-chip.

Custom designed blocks are characterized at the transistor-level before running timing analysis. Timing analysis is run at the chip-level to obtain a list of critical timing paths. The output of the timing analysis consists of a set of critical paths along with the transition for every node on each path. Our tool $FA$l$S$le p$A$th $D$etector (FASAD) post-processes the paths generated by the timing analyzer using information from its path extraction capacity. It takes the processed critical paths as input and generates command files for the ATPG tool to perform a combinational false path analysis on the critical paths. The results of the ATPG runs are analyzed by FASAD and the false paths are identified.

All our runs were performed on a Sun Solaris with 4GB memory. We generated 557 most critical paths on the second generation G4 PowerPC$^{TM}$ processor using the timing analyzer. We ran FASAD along with its post-processor on them as shown in the figure 5 on these 557 paths. Out of these 557 paths, 111 are multi-cycle paths utilizing cycle stealing technique and will not be considered in the scope of this paper. That leaves us with a total of 446 paths to analyze.

The time taken to generate the command files for the ATPG tool is around 5 minutes. The core and memory models are loaded in different ATPG sessions and run in parallel (it takes about 20 minutes to load each model). The ATPG tool takes around 30 minutes to run the command files for all the 446 paths and analyze them. The ATPG tool produced a log file which is then analyzed using the post-processor of FASAD. It identified 95 paths as unsensitizable. The time taken for this classification is around 5 minutes. It takes around 1 hour to identify 95 paths from 446 initial timing paths.

Comparing to our initial results[1](where about only 5% false paths were identified), it underlies the importance of path extraction of custom blocks allowing the false path

identification analysis on all types of design blocks on the chip.

## 5. Conclusions and future work

To overcome the difference in ATPG and timing analysis techniques, we implemented a circuit-level path extractor to extract paths between input and output pin pairs for custom designed blocks. We were able to analyze full-chip paths and identified 21% of these as false paths.

So far we have worked on identifying false paths using scan test vectors. Future work will include identifying non-functionally false paths taking into consideration the sequential behavior of the circuits.

It is desirable for the characterization tool to keep record of the activation vector associated with the pin to pin delay since there are generally multiple paths associated with a particular input and output pair of the design block. Unfortunately, currently the characterization process only records worst case pin to pin delays of blocks, but not the activation vectors associated with them. Circuit-level analysis and test generation tools can be used to help the characterization process to capture both the delay and activation vector efficiently.

## 6. Acknowledgments

We acknowledge the valuable help from Mike Figley, Bruce Long, Robert Bailey, George Joos, Ashu Razdan, Dawit Belete of Motorola Inc. PowerPC is a trademark of the International Business Machines Corporation, used under license therefrom.

## References

[1] 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$^{TM}$ Microprocessor" in *Proceedings of the IEEE 2000 Custom Integrated Circuits Conference*, 71-74, May 2000.

[2] K. T. Cheng, H. C. Chen. "Classification and Identification of Nonrobust Untestable Path Delay Faults" in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(8):845–853, August 1996.

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

[4] S. T. Huang, T. M. Parng and J. M. Shyu. "A Polynomial-Time Heuristic Approach to Solving the False Path Problem", in *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 43(5)386-396, May 1996.

[5] H. Chang. "Strategies for Design and test of High Performance Systems", *Ph.D. Dissertation*, The University of Texas at Austin, August 1993.

[6] R. Raimi and J. A. Abraham. "Detecting False Timing Paths", in *Proceedings of the Design Automation Conference*, 737-741, 1999.

[7] R. Raimi and J. A. Abraham. "False Timing Paths and Environment Modeling: Experiments on PowerPC Microprocessors", in *Proceedings of High-Level Design, Validation and Test*, 1998.

[8] C. P. R. Liu. "Transistor level Synthesis and Hierarchical Timing Optimization for CMOS Combinational Circuits", *Ph.D. Dissertation*, The University of Texas at Austin, August 1999.

[9] K. T. Lee and J. A. Abraham. "Critical Path identification and Delay Tests of Dynamic Circuits", in *Proceedings of the International Test Conference*, 421-430, 1999.

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

[11] D. Blaauw et al. "Removing user-specified false paths from timing graphs", in *Proceedings of the Design Automation Conference*, 270-273, 2000.