An Implication-based Method to Detect Multi-Cycle Paths in Large Sequential Circuits

Hiroyuki Higuchi
Fujitsu Laboratories Ltd.
4-1-1, Kamikodanaka, Nakahara-Ku,
Kawasaki 211-8588, Japan
higuchi@flab.fujitsu.co.jp

ABSTRACT

This paper proposes a fast multi-cycle path analysis method for large sequential circuits. It determines whether or not all the paths between every flip-flop pair are multi-cycle paths. The proposed method is based on ATPG techniques, especially on implication techniques, to utilize circuit structure and multi-cycle path condition directly. The method also checks whether or not the multi-cycle path may be invalidated by static hazards in combinational logic parts. Experimental results show that our method is much faster than conventional ones.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids

General Terms

Algorithms, Designs, Verification

Keywords

multi-cycle path, sequential circuits, implication, ATPG

1. INTRODUCTION

Due to the steadily increasing demand for high performance integrated circuits, timing verification and optimization are becoming more and more important. The key for good timing verification and optimization is to compute circuit delay accurately and quickly.

The most common and easiest approach to compute circuit delay is based on topological delay. Topological delay can be too conservative because it ignores false paths and multi-cycle paths. False paths and multi-cycle paths relax timing constraints, which can be utilized in logic synthesis, layout, ATPG for delay faults, and static timing analysis. A false path in a circuit is a path that is never activated because of the circuit functionality and/or delay values of the circuit components. The problem of false paths in logic circuits have been studied extensively in recent years [1, 2, 3, 7]. Since the number of paths may be exponential in circuit size, path-based analysis may suffer from the combinatorial explosion.

A multi-cycle path in a sequential circuit is a combinational path that does not have to propagate signals in single clock cycle. Multi-cycle paths have also been investigated in literature to some extent [5, 8, 9, 10]. Multi-cycle path analysis methods can be grouped into path-based [5, 10] ones and non-path-based ones [8, 9]. Path-based methods suffer from the combinatorial explosion in the same way as false path analysis. Non-path-based methods determine whether or not all the paths between a given flip-flop(FF) pair are multi-cycle paths. Therefore they do not suffer from the combinational explosion. While it is rare for every path between an input and an output to be false paths, the same is not true of multi cycle paths. Two methods for non-path-based analysis have been proposed; symbolic traversal based one [8] and SAT-based one [9]. Though SAT-based method is faster than symbolic traversal based one, it is still not applicable to large circuits.

In this paper we propose a fast analysis method to detect multi cycle paths in large sequential circuits. Our method belongs to the category of non-path-based methods and is based on automatic test pattern generation(ATPG) techniques, especially on implication techniques, to utilize circuit structure and multi-cycle path conditions directly. We also show that conventional non-path-based methods for multi-cycle path analysis can be optimistic because they do not consider static hazards in combinational logic parts. Our method checks whether or not detected multi-cycle paths are valid even when static hazards are taken into account. It uses static sensitization and co-sensitization conditions.

The rest of this paper is organized as follows. In Section 2, we review definitions and notations. In Section 3, we define the problem of detecting multi-cycle paths. In Section 4, we propose a new multi-cycle path detection method. In Section 5, we also propose a method to check whether or not the multi-cycle paths detected by the method described in Section 4 are really multi-cycle paths under the existence of static hazards. In Section 5, we give experimental results.

2. DEFINITIONS

2.1 Circuits and Paths

We consider synchronous sequential circuits. They con-

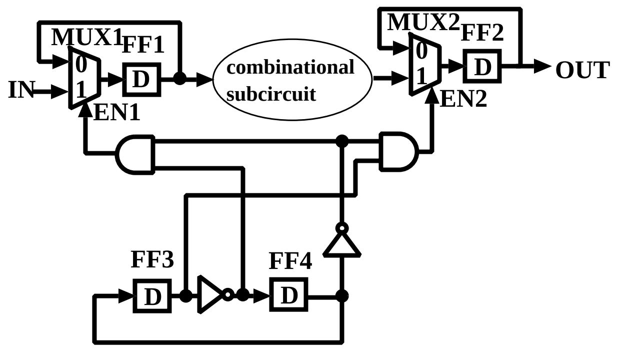

Figure 1: Example of multi-cycle paths

sist of an interconnection of combinational logic gates and clocked FFs. We assume that all the FFs are positive edge triggered D-FF driven by one clock. No direct combinational feedback is allowed.

A **combinational path** P in a sequential circuit is an alternating sequence of gates and edges $\{g_0, e_0, g_1, e_1, \ldots, g_{m-1}, e_{m-1}, g_m\}$, where g_0 is an primary input or an FF and g_m is an primary output or a fanin gate of an FF. Edge $e_i, 0 \leq i \leq m-1$, is an **on-input** of P which connects gate g_j to gate g_{j+1}. An edge is called a **side-input** of P with respect to e_i if it is connected to g_{i+1} but not originated from g_i. The **controlling value** at an input to a gate is the value that determines the output value of the gate independent of the other inputs. The **non-controlling value** to a gate is the complementary value of the controlling value. We say a gate g_i has the **controlled value** if one of its inputs has a controlling value; otherwise, we say g_i has a **non-controlled value**. A combinational path is simply called a path in this paper.

A signal value at the output of gate g_i is simply denoted by g_i. A signal value at the output of gate g_i at time t is denoted by $g_i(t)$.

2.2 Multi-Cycle Paths

A **multi-cycle path** in a sequential circuit is a path that does not have to propagate signals in single clock cycle. A **k-cycle path** is a path that is allowed to use k clock cycles to propagate signals. A **single-cycle path** is a path that is not a multi-cycle path. For example, consider a circuit shown in Fig.1. In the circuit IN and OUT are a primary input and a primary output respectively. FFs FF_3 and FF_4 constitute a 4-cycle gray code counter. The state transitions of the counter is as follows: $(0,0) \rightarrow (0,1) \rightarrow (1,1) \rightarrow (1,0) \rightarrow (0,0) \rightarrow \cdots$. Multiplexer MUX_1 selects primary input data IN when $(FF_3, FF_4) = (0,0)$. Then FF_1 is set to the value IN when $(FF_3, FF_4) = (0,1)$. On the other hand, MUX_2 selects the output of the combinational subcircuit when $(FF_3, FF_4) = (1,0)$. Then FF_2 is set to the value when $(FF_3, FF_4) = (0,0)$. Since the counter (FF_3, FF_4) requires 3 clocks to go from state (0,1) to state (0,0), the paths from FF_1 to FF_2 are 3-cycle paths. This means that the paths are allowed to use 3 clock cycle to propagate signals. As a consequence, the timing constraint of the paths can be relaxed from 1 clock cycle to 3 clock cycle.

A **multi-cycle FF pair** (FF_i, FF_j) is an ordered pair of FFs such that every path from FF_i to FF_j is a multi-cycle path. In Fig.1, FF pair (FF_1, FF_2) is a multi-cycle FF path.

2.3 Path Sensitization Conditions

A path is **statically sensitizable** if there exists an input vector such that all the side-inputs along the path settle to non-controlling values for that vector. This sensitization condition is delay-independent. In this paper we say that a set of paths is statically sensitizable if at least one of the paths is statically sensitizable.

A path $P = \{g_0, e_0, \ldots, g_{m-1}, e_{m-1}, g_m\}$ is **statically co-sensitizable** to a 1(0) if there exists an input vector v satisfying the following conditions [3]:

1. $g_m(v) = 1(0)$,
2. for each $g_i, 1 \leq i \leq m$, if g_i has a controlled value, then the edge e_{i-1} presents the controlling value of g_i.

In this paper we say that a path is statically co-sensitizable if the path is statically co-sensitizable to at least one logic value. Static co-sensitization condition is also delay-independent. Note that any statically sensitizable path is also statically co-sensitizable. In this paper we say that a set of paths is statically co-sensitizable if at least one of the paths is statically co-sensitizable.

Given a delay assignment for a delay model, a path $P = \{g_0, e_0, \ldots, g_{m-1}, e_{m-1}, g_m\}$ is **sensitizable** (or **true**) if and only if there exists an input vector satisfying the following conditions at each gate $g_i, 1 \leq i \leq m$ [2, 3]:

1. If all inputs of g_i have non-controlling values, then g_{i-1} must be the last input of g_i to present the non-controlling value.
2. If at least one input of g_i has a controlling value, then g_{i-1} must be the earliest input of g_i to present the controlling value.

Note that for any delay assignment any sensitizable path is also statically co-sensitizable and any statically sensitizable path is also sensitizable. This means that static sensitization and static co-sensitization gives lower bound and upper bound to the exact sensitization condition respectively.

3. PROBLEM FORMULATION

3.1 Condition for Multi-Cycle FF Pairs

A pair of FFs (FF_i, FF_j) is a multi-cycle FF pair if and only if whenever a transition occurs at the source FF_i, at least one of the following two conditions is satisfied:

1. The transition is not propagated to the sink FF_j.
2. The transition is propagated to the sink, but
   (a) the transition at the sink is never observed at any primary output, and
   (b) for any FF FF_k, (FF_j, FF_k) is a multi-cycle FF pair under the assumption that a transition is propagated from FF_i to FF_j in the previous clock cycle.

It should be mentioned here that the signal at the source is not necessarily stable for more than one clock cycle before the signal is required at the sink. When the value after a transition at the source is overwritten before it is used at the sink, the transition never arrive at the sink. Then you can say that the path is infinite cycle path, that is, false in the situation.

Condition 2 is difficult to check because the analysis may require traversal of many states. In addition to this, Condition 2 can be viewed as some kind of timing budget borrowing from the subsequent FF pair. Thus we consider only Condition 1 in this paper. We further simplify the condition 1 by using the following relationship:

- $FF_j(t+1) = FF_j(t+2) \Rightarrow$ "A transition at FF_i at time $t+1$ is not propagated to FF_j." [1]

Thus we use the following sufficient condition to detect multi-cycle FF pairs:

- (FF_i, FF_j) is a multi-cycle FF pair if $FF_i(t) \neq FF_i(t+1) \Rightarrow FF_j(t+1) = FF_j(t+2)$.

Since this is a sufficient condition, FF pairs detected by using the condition are always multi-cycle FF pairs when we do not consider delay in combinational logic parts. The condition is the same as in [8, 9]. Note that while [8] take into account reachable states, [9] and our method assume that all the states can be reachable. [8] may detect more multi-cycle paths than [9] and ours.

3.2 Multi-Cycle FF Pair Detection Problem

In the next section we consider the following multi-cycle FF pair detection problem:

Inputs: a synchronous sequential circuit

Outputs: all the multi-cycle FF pairs (FF_i, FF_j) that satisfy the following condition:

$$FF_i(t) \neq FF_i(t+1) \Rightarrow FF_j(t+1) = FF_j(t+2)$$

We refer the condition above as MC condition.

4. IMPLICATION-BASED ANALYSIS FOR DETECTING MULTI-CYCLE PATHS

In this section we propose a fast algorithm for the multi-cycle FF pair detection problem described in the previous section. As you can see, the MC condition is nothing but implication relation. Thus our method utilizes implication procedure as much as possible to detect multi-cycle paths efficiently, while conventional non-path-based methods attempt to check the condition by BDD-based state traversal [8] or SAT solver [9].

4.1 Basic Flow of the Analysis

To check the MC condition, we need to deal with 2 time frames. Therefore we use the expanded combinational logic part in 2 time frames. The checking is done by using implication procedure and backtrack search for ATPG. Implication procedure directly attempts to check the MC condition. The backtrack search tries to prove that there is no input pattern which violates MC condition. Note that the step should also justify the premise of the MC condition, that is, there exists at least one pair of inputs and states which satisfies $FF_i(t) = FF_i(t+1)$. If the justification fails, the step answers that the MC condition holds.

The overall flow of our algorithm is as follows:

1) Check if there exists a path between the FFs in each FF pair. Drop the FF pair if there is no path.

2) Carry out random pattern simulation to drop FF pairs not satisfying the MC condition.

3) Expand the combinational logic part into 2 time frames. Let the first time frame be t.

4) For each remaining FF pair (FF_i, FF_j), do Steps 4.1)-4.2).

   4.1) For each possible assignment $(FF_i(t), FF_j(t+1)) = \{(0,0),(0,1),(1,0),(1,1)\}$, do Steps 4.1.1)-4.1.5).

      4.1.1) $FF_i(t+1) \leftarrow \overline{FF_i(t)}$.

      4.1.2) Carry out implication procedure, that is, assign as many mandatory values as possible.

      4.1.3) If $FF_j(t+2) = FF_j(t+1)$ or contradiction occurs during implication procedure, go to Steps 4.1). If $FF_j(t+2) = \overline{FF_j(t+1)}$, then identify (FF_i, FF_j) as a single-cycle FF pair and go to Step 4).

      4.1.4) Search an input pattern such that $FF_j(t+2) \neq FF_j(t+1)$.

      4.1.5) If an input pattern is found, identify (FF_i, FF_j) as a single-cycle FF pair and go to Step 4).

   4.2) Identify (FF_i, FF_j) as a multi-cycle FF pair. □

Though this algorithm is to detect multi-cycle FF pairs, it can be easily extended to detect k-cycle FF pairs ($k = 3, 4, \ldots$) by increasing the number of time frames in Step 3.

4.2 Example

For example, let us apply the algorithm described in the previous subsection to the circuit in Fig.1.

After Step 1, the following 9 FF pairs remain among 16 FF pairs: (FF_1, FF_1), (FF_1, FF_2), (FF_2, FF_2), (FF_3, FF_1), (FF_3, FF_2), (FF_3, FF_4), (FF_4, FF_1), (FF_4, FF_2), (FF_4, FF_3). After Step 2, the following 5 FF pairs remain: (FF_1, FF_1), (FF_1, FF_2), (FF_2, FF_2), (FF_3, FF_2), (FF_4, FF_1). Here we omit the explanation of the detail of random pattern simulation.

In Step 3 the combinational logic part is expanded into 2 time frames. In Step 4 let us choose (FF_1, FF_2), for example. In Step 4.1 let us choose assignment $(FF_1(t), FF_2(t+1))$=$(0,0)$. In Step 4.1.1 value $\overline{FF_1(t)}(=1)$ is assigned to $FF_1(t+1)$ to analyze such the case that a rise transition occurs at FF_1 at time $t+1$. Figure 2 shows the value assignment after implication procedure in Step 4.1.2. In the figure, a signal value in a circle represents the value assigned before the implication procedure and the other values represent implied values. Since we obtain $FF_2(t+2) = 0$, we can say that the signal at FF_2 never change at time $t+2$. This implies that the MC condition is satisfied. Thus when $(FF_1(t), FF_1(t+1), FF_2(t+1)) = (0,1,0)$, (FF_1, FF_2) are identified as a multi-cycle FF pair.

Similarly, the analysis is done for the assignments $(FF_1(t), FF_2(t+1)) = (0,1)$, $(1,0)$, $(1,1)$ one by one. If the FF pair cannot be identified as either single-cycle or multi-cycle FF pair, then the analysis is continued in Step 4.1.4 by

[1]The converse is not always true, because even when a transition at FF_i is not propagated to FF_j, a transition from another FF may be propagated.

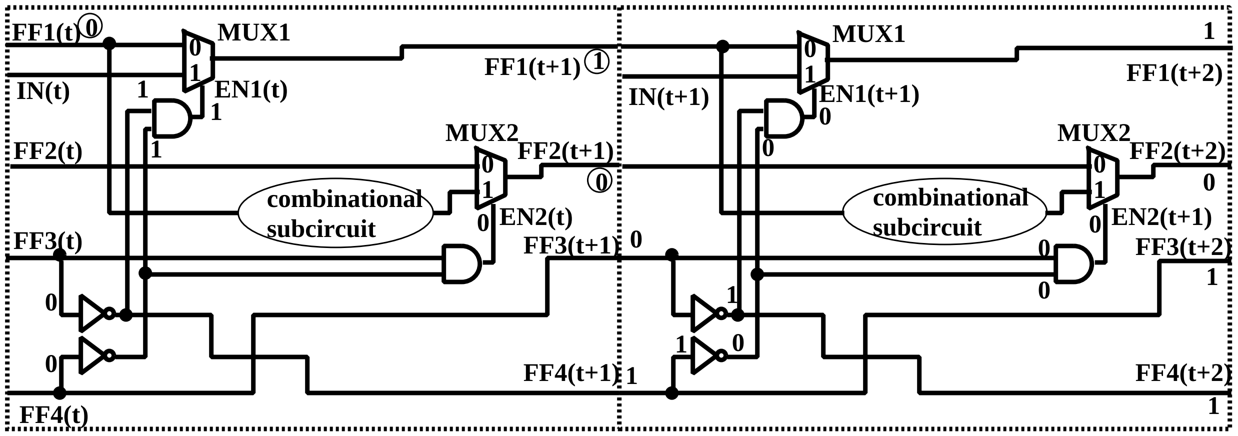

Figure 2: Example of implication procedure

using ATPG. In the circuit shown in Fig.1, all 5 candidates after random pattern simulation are identified as multi-cycle FF pairs.

4.3 Random Pattern Simulation

Random pattern simulation is done by parallel pattern simulation. Specifically, a random pattern of one word length is assigned to each primary input and each output of FFs and simulation is done for two clock cycles. Then it is checked whether or not for each FF pair (FF_i, FF_j) there is a bit position such that the following condition is satisfied:

$$FF_i(t) \neq FF_i(t+1) \wedge FF_j(t+1) \neq FF_j(t+2).$$

If there is such a bit position, the pair is dropped as a single-cycle FF pair. The above logic operation is done by bit-wise operation in parallel. The simulation for 2 clocks are repeated for a given number of patterns.

4.4 Implication

Local implication procedure attempts to assign as many mandatory values as possible by propagating already assigned values. A contradiction during implication procedure indicates that the combination of already assigned values is impossible. In that case there is no need to check whether or not the given FF pair is a multi-cycle FF pair for the assignment.

Global implication like static learning [11] can detect more implication relations than local implication.

4.5 ATPG

Multi-cycle FF pair detection problem has the following properties:

- Several value assignments have already been done before ATPG is applied.
- The target fault is likely to be redundant.

From these point of view, we adopted D-algorithm based method because it assigns values to internal nodes directly and tries to detect contradictions faster than PODEM based method.

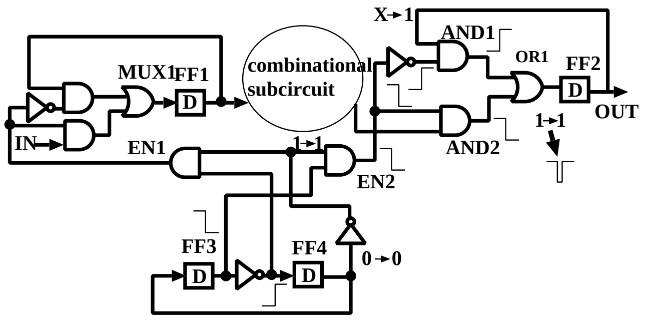

Figure 3: Example of static hazards

5. TAKING INTO ACCOUNT STATIC HAZARDS IN COMBINATIONAL LOGIC

In this section we show that multi-cycle FF pairs detected by our method and the conventional non-path-based methods are not necessarily multi-cycle FF pairs if we take into account static hazards in the combinational logic part. Then we give methods to detect possible static hazards in delay-independent manners.

5.1 Effects of Static Hazards

The following proposition is not always true if we take into account static hazards in the combinational logic part:

- If the MC condition for an FF pair is true, then the FF pair is a multi-cycle FF pair.

For example, let us assume that the circuit shown in Fig.1 is technology-mapped as in Fig.3. Only each multiplexer is replaced with 2 AND gates, 1 OR gates, and 1 NOT gate. Let us consider FF pair (FF_3, FF_2). The pair was identified as a multi-cycle FF pair according to the MC condition. The value assignment in Fig.3 shows the implied assignment when $(FF_3(t), FF_3(t+1), FF_2(t+1)) = (1, 0, 1)$ and $FF_2(t+2) = 1$. Assignment $FF_2(t+2) = 1$ is added because the FF pair satisfies the MC condition. From Fig.3 you can see that the input of FF_2 may have a static hazard when the output of AND_1 is slower than that of AND_2.

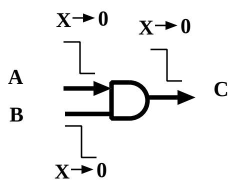

Figure 4: Unsensitizable situation in static sensitization

Since the method shown in Section 4 and conventional non-path-based methods only consider the stable function value, they overlook static hazards. When the FF pair is regarded as a multi-cycle FF pair and its timing constraints are relaxed, problems may occur. For example, assume that the output of AND1 become much slower by either replacing the AND gate with slower one or detouring the routing. Then the signal transition at AND1 exceeds the clock boundary. That may violate the circuit functionality. Therefore it is important to check the existence of static hazards for identified multi-cycle FF pairs.

It requires a lot of computation cost to exactly detect static hazards for a given FF pair, because static hazards can occur at every edge on the paths between the FF pair and in addition it should be checked whether or not each hazards can be propagated. In this paper we utilize static sensitization condition and static co-sensitization condition to detect static hazards. By using these conditions, we can analyze static hazards in delay-independent manners.

5.2 Checking based on Static Sensitization

In case of Fig.3 static sensitization condition can find the static hazard, because the path through AND_1 and OR_1 is statically sensitizable. Thus static sensitization can determine whether or not a static hazard occurs. However checking based on static sensitization may introduce conservative results on propagating the static hazard, because an sensitizable path is not always statically sensitizable. For example, Fig.4 shows a case where a path from A to C is not statically sensitizable, because B has a controlling value and blocks the path from A to C. Note that the values in the first clock cycle is unknown(X) because we should take into account static hazards. Suppose that the path from A to C is detected as a multi-cycle path by our method shown in Section 4 and that checking based on static sensitization find a static hazard and the static hazard disappears at the AND gate in Fig.4. This implies that the static hazard disappears under the condition that the side-input B blocks the on-input at the second clock. Threfore if a path from B to C is also detected by a multi-cycle path and the timing constraint is relaxed, then the static hazard may propagate, which means that the timing constraint of path from A to C cannot be relaxed. Thus multi-cycle FF pairs verified by static sensitization condition may introduce dependency between them.

5.3 Checking based on Static Co-Sensitization

Since a sensitizable path is always statically co-sensitizable, it does not suffer from dependency between multi-cycle paths. That is, multi-cycle FF pairs verified by static co-sensitization are always multi-cycle FF pairs. However there may not be static hazards on the path, because static co-sensitization gives upper bound to the exact sensitization. In Fig.4 the path from A to C is statically co-sensitizable.

6. EXPERIMENTAL RESULTS

We have implemented a multi-cycle FF pair detection program based on the method described so far. The program is written in C++ language and runs on Pentium III(866MHz, main memory 512MByte)+FreeBSD4.5R+gcc version 2.95.3 with -O2 option. ATPG techniques are based on [6]. The experiments were done on ISCAS89 benchmark circuits [4].

In the experiments random pattern simulation was continued until no FF pairs were dropped during 32×10 consecutive patterns. The number of backtracks in ATPG was limited to 50 except for "s9234.1","s9234","s13207.1","prolog". For these circuits the maximum numbers of backtracks were set to 650,150,150,51, and static learning [11] was done before ATPG.

Table 1 shows the experimental results of detecting multi-cycle FF pairs without checking static hazards. We compared our method with conventional SAT-based method [9]. CPU time reported in [9] was measured on Pentium II (500MHz, main memory 512MByte). In table 1, Columns "In", "FF", "FF-pair" show the numbers of the primary inputs, FFs, and the topologically connected FF pairs respectively. Column "ours" shows the results of our methods. Column "[9]" represents the results of conventional SAT-based method [9]. Column "MC-pair" indicates the number of detected multi-cycle FF pairs. Column "CPU(sec)" represents CPU time in seconds. Entry "–" means that the data were not reported in [9].

Table1 indicates that our method can detect multi-cycle FF pairs efficiently for all the benchmark circuits. All the FF pairs were identified as single-cycle FF pairs or multi-cycle FF pairs. For circuits with more than 1,000 FFs, our method can detect all the multi-cycle FF pairs in reasonable time. The table also shows that the number of multi-cycle FF pairs is approximately one-tenth as large as the number of all the topologically connected FF pairs.

The proposed method is much faster than the conventional SAT-based one [9], though machines on which CPU times were measured are different. The number of the multi-cycle FF pairs detected by our method is the same as that by conventional one except for the self-loop FF pairs, which seem to be excluded beforehand in the conventional method.

Table 2 shows the total numbers of identified FF pairs and CPU times in each analysis step. Column "Sim.", "Implication", and "ATPG" represent random pattern simulation, implication procedure, and ATPG respectively. Rows "single cycle" and "multi cycle" indicates the numbers of FF pairs identified as single-cycle paths and multi-cycle paths in each step respectively. Data in the parentheses show the ratio of the number of the detected FF pairs. The table shows that almost all the single-cycle paths can be dropped by random pattern simulation and more than 80% of all the multi-cycle paths can be detected by implication procedure. It is probably the main reason why our method is faster than the SAT-based method.

Table 3 shows the total number of multi-cycle paths after static hazard checking and total CPU times required for the checking. Row "before" indicates data before static hazard checking. Rows "sensitize" and "co-sensitize" represetns data after checking based on static sensitization and static

Table 1: Experimental results of detecting multi-cycle FF pairs without checking static hazards

				ours		[7]
circuit	In	FF	FF-pair	MC pair	CPU (sec)	CPU (sec)
s27	4	3	7	0	0.3	0.5
s208.1	10	8	36	28	0.3	-
s208	11	8	36	0	0.3	0.5
s298	3	14	70	3	0.3	0.9
s344	9	15	89	1	0.3	1.3
s349	9	15	89	1	0.3	1.3
s382	3	21	146	13	0.3	2.0
s386	7	6	36	4	0.3	0.9
s400	3	21	146	13	0.3	-
s420.1	18	16	136	120	0.4	-
s420	19	16	88	0	0.3	1.1
s444	3	21	146	13	0.3	2.6
s499	1	22	484	379	0.6	-
s510	19	6	36	3	0.3	1.4
s526	3	21	144	7	0.4	2.5
s526n	3	21	144	7	0.3	2.5
s635	2	32	528	0	0.5	-
s641	35	19	115	1	0.3	1.9
s713	35	19	115	1	0.3	2.6
s820	18	5	25	0	0.3	1.7
s832	18	5	25	0	0.3	1.6
s838.1	34	32	528	496	0.9	-
s838	35	32	192	0	0.3	3.0
s938	34	32	538	496	0.9	-
s953	16	29	156	29	0.5	6.2
s967	16	29	156	29	0.5	-
s991	65	19	71	20	0.4	-
s1196	14	18	20	0	0.3	2.2
s1238	14	18	20	0	0.4	2.4
s1423	17	74	1765	47	0.4	60.4
s1488	8	6	36	0	0.3	2.0
s1494	8	6	36	0	0.4	2.0
prolog	36	136	578	29	0.6	-
s1269	18	37	288	3	0.3	-
s1512	29	57	513	1	0.4	-
s3271	26	116	899	0	0.5	-
s3330	40	132	549	0	0.5	-
s3384	43	183	1831	0	0.5	-
s4863	49	104	628	0	1.0	-
s5378	35	179	1200	55	0.9	-
s6669	83	239	2179	0	0.6	-
s9234.1	36	211	2681	37	12.3	-
s9234	19	228	2830	168	11.7	-
s13207.1	62	638	3411	580	18.8	-
s13207	31	669	3716	937	13.2	-
s15850.1	77	534	11873	320	7.6	-
s15850	14	597	15363	3756	9.6	-
s35932	35	1728	4763	0	2.3	-
s38417	28	1636	33852	240	36.1	-
s38584.1	38	1426	16372	17	9.2	-
s38584	12	1452	17978	1622	13.9	-
Total		10908	125504	9476	152.4	(103.5)

Table 2: Results of each analysis step

	Sim.	Implication	ATPG
single cycle	107974 (86%)	240 (0.2%)	7914 (6.3%)
multi cycle	0 (0%)	7712 (6.1%)	1764 (1.4%)
CPU(sec)	31.0	29.0	74.4

Table 3: Results of static hazard checking

	MC-pair	CPU(sec)
before	9,476	152.4
sensitize	8,084	57.1
co-sensitize	5,570	62.5

co-sensitization respectively. It shows that one-tenth of the multi-cycle FF pairs detected without static hazard checking may have static hazards at the input of FFs and three-tenth of them may depend on one another. CPU times required to detect static hazards are practical.

7. CONCLUSIONS

We have proposed a fast multi-cycle path analysis method for large sequential circuits. We have also shown that conventional non-path-based methods can be optimistic because they do not consider static hazards and introduced a method to detect static hazard in delay-independent manners. Experimental results indicates that the method is much faster than conventional methods.

8. REFERENCES

[1] D. Brand and V. S. Iyengar. Timing Analysis using Functional Relationships. In *Proc. of IEEE International Conference on CAD-86*, pages 126–129, Nov. 1986.

[2] H.-C. Chen and D. H.-C. Du. Path Sensitization in Critical Path Problem. *IEEE Trans. Comput.-Aided Design Integrated Circuits*, 12(2):196–207, Feb. 1993.

[3] S. Devadas, K. Keutzer, and S. Malik. Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms. *IEEE Trans. Comput.-Aided Design Integrated Circuits*, 12(12):1913–1923, Dec. 1993.

[4] D. B. F. Brglez and k. Kozminski. Combinational Profiles of Sequential Benchmark Circuits. In *Proc. of the International Symposium on Circuits and Systems*, May 1989.

[5] A. P. Gupta and D. P. Siewiorek. Automated Multi-Cycle Symbolic Timing Verification of Microprocessor-based Designs. In *Proc. of the 31th ACM/IEEE Design Automation Conference*, pages 113–119, 1994.

[6] Y. Matsunaga and M. Fujita. Enhanced Unique Sensitization for Efficient Test Generation. *IEICE Trans. on Information and Systems*, E76-D(9):1114–1120, Sept. 1993.

[7] P. C. McGeer, A. Saldanha, R. K. Brayton, and A. Sangiovanni-Vincentelli. Delay Models and Exact Timing Analysis. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 167–189. Kluwer Academic Publishers, 1993.

[8] K. Nakamura, K. Takagi, S. Kimura, and K. Watanabe. Waiting False Path Analysis of Sequential Logic Circuits for Performance Optimization. In *Proc. of IEEE/ACM International Conference on CAD-98*, pages 388–393, Nov. 1997.

[9] K. Nakamura, S. Maruoka, S. Kimura, and K. Watanabe. Multi-Cycpe Path Detection based on Propositional Satisfiability with CNF Simplification using Adaptive Variable Insertion. *IEICE Trans. on Fundamentals*, E83-A(12):2600–2607, Dec. 2000.

[10] W.-C. Lai, A. Krstic, and K.-T. Cheng. Functionally Testable Path Delay Faults on a Microprocessor. *IEEE Design & Test of Computers*, oct 2000.

[11] M. Schulz, E. Trishler, and T. Sarfert. SOCRATES: A Highly Efficient Automatic Test Pattern Generation System. In *Proc. of International Test Conference*, pages 1016–1026, 1987.