

Enhancing the Performance of Multi-Cycle Path Analysis in an Industrial Setting

Higuchi, Hiroyuki
Fujitsu Laboratories Ltd. | Kyushu University

Matsunaga, Yusuke
Kyushu University

<https://hdl.handle.net/2324/6794471>

出版情報 : Asia and South Pacific Design Automation Conference 2004, pp.192-197, 2004-01. IEEE
バージョン :
権利関係 :



Enhancing the Performance of Multi-Cycle Path Analysis in an Industrial Setting

Hiroyuki Higuchi

Fujitsu Laboratories Ltd. / Kyushu University
e-mail: higuchi@labs.fujitsu.com

Yusuke Matsunaga

Kyushu University
e-mail: matsunaga@c.csce.kyushu-u.ac.jp

Abstract— In this paper we enhance the performance of multi-cycle path analysis in an industrial setting. Industrial designs are, in general, more complicated, but contain more information than fundamental sequential circuits. We show how such information is used for improving the quality and the efficiency of multi-cycle path analysis. Specifically, we propose local FSM learning to take into account reachability information. We also propose FF enable learning to accelerate multi-cycle path analysis. Experimental results show that our methods can handle large industrial designs with tens of thousands of FFs and detects more multi-cycle paths faster than conventional ones.

I. INTRODUCTION

Because of the steadily increasing demand for high performance integrated circuits, timing verification and optimization are becoming more and more important. The key to 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 used in logic synthesis, layout, ATPG for delay faults, and static timing analysis (STA). A false path in a circuit is a path that is never activated because of the circuit functionality and delay values of the circuit components. The problem of identifying false paths in logic circuits have been studied extensively in recent years [1, 2, 3, 4]. 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 a single clock cycle. Multi-cycle paths have also been investigated in literature to some extent [5, 6, 7, 8]. Multi-cycle path analysis methods can be grouped into path-based ones and non-path-based ones [5, 8]. Path-based methods suffer from the combinatorial explosion in the same way as false path analysis does. On the other hand, 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 combinatorial explosion. While it is rare for every path between an input and an output to be a false path, the same does not apply to multi-cycle paths. Two methods for non-path-based analysis have been proposed: symbolic-traversal-based one [6] and SAT-based one [7]. Though the SAT-based method is faster than the symbolic-traversal-based one, it is still not applicable

to large and complex circuits in industrial hardware designs.

Recently a method based on ATPG, especially based on logic implication, has been proposed for multi-cycle path analysis [9]. It has been experimentally shown that it is more efficient than conventional methods. The method is, however, only applicable to fully synchronous circuits with simple gates and FFs. We call these circuits as fundamental sequential circuits. The method cannot directly handle more complicated industrial designs with multiple clocks, complicated cells, and so on.

This paper describes how to apply implication-based multi-cycle path analysis to industrial designs and how to improve multi-cycle path analysis by using information in industrial design, such as local Finite State Machines (FSMs) and FF enable input information. To use such information efficiently in the framework of the implication-based analysis, we propose local FSM learning and FF enable learning. By using local FSM learning, reachable state information can be naturally taken into account. It is generally difficult to identify local FSMs, but it is not the case for industrial designs, because the names of FFs in a local FSM are likely to be similar. By using FF enable learning, the analysis for FF pairs with the same enable input does not have to be done again, which is effective especially for datapath parts.

The rest of this paper is organized as follows. In Section II we define terminologies and review the implication-based multi-cycle path analysis for fundamental sequential circuits. In Section III we show how to convert industrial designs into fundamental sequential circuits automatically. In Sections IV and V we propose local FSM learning and FF enable learning respectively. In Section VI we give experimental results for industrial designs.

II. PRELIMINARIES

A. Multi-Cycle Paths

We consider sequential circuits consisting of interconnections of combinational logic gates and clocked FFs. We assume the following: (1) Each FF is clocked by a single clock, and (2) there is no direct combinational feedbacks.

A *combinational path* P is an alternating sequence of gates and edges $\{g_0, e_0, g_1, e_1, \dots, g_{m-1}, e_{m-1}, g_m\}$, where g_0 is a primary input or an FF and g_m is a primary output or an FF. We call g_0 and g_m the *source* and the *sink* of the path respectively. 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 $g_i(t)$.

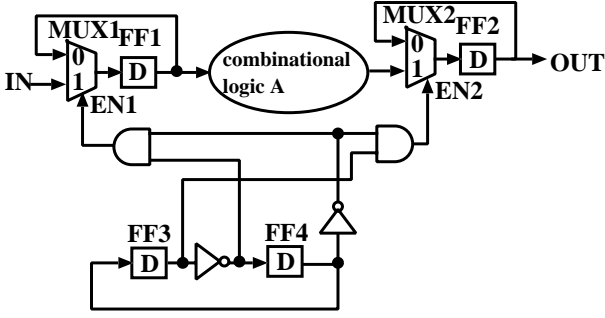


Fig. 1. Example of multi-cycle paths.

A *multi-cycle path* in a sequential circuit is a combinational path that does not have to propagate signals in a single clock cycle. A *k-cycle path* is a path that is allowed to use at most k clock cycles to propagate signals. A *single-cycle path* is a path that is not a multi-cycle path. 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.

B. Example of Multi-Cycle Paths

For example, consider a circuit shown in Fig.1. In the circuit IN and OUT are the primary input and the 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: $(0,0) \rightarrow (0,1) \rightarrow (1,1) \rightarrow (1,0) \rightarrow (0,0) \rightarrow \dots$. 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 logic A 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 cycles to propagate signals. As a consequence, the timing constraint of the paths can be relaxed from 1 clock cycle to 3 clock cycles. (FF_1, FF_2) is called a multi-cycle FF pair.

C. Multi-Cycle FF Pair Detection Problem

In [9] the following multi-cycle FF pair detection problem has been considered:

Input: A fully synchronous sequential circuit

Output: 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 the MC condition.

This condition is not exact because it does not take into account the following three issues:

- Path sensitization condition,
- Reachability of the state vector at time t from the reset states,

```

MCPSet MCPAnalysis_Orig(Circuit C) {
1.   $P \leftarrow \text{StructuralAnalysis}(C);$ 
    /* P is the set of the candidate MC pairs */
2.   $P \leftarrow \text{RandomPatternSimulation}(C, P);$ 
3.  for each  $p \in P$  {
4.    for each assignment of  $(FF_i(t), FF_j(t+1))$  {
5.       $FF_i(t+1) \leftarrow FF_i(t);$  /* transition at  $FF_i$  */
6.      Implication( $C, p$ );
7.      if MC condition is satisfied then continue;
8.      ATPG( $C, p$ );
9.      if a pattern is found or aborted then {
10.         $P \leftarrow P \setminus \{p\};$ 
11.        break; /* single cycle or aborted */
12.      }
13.    }
14.  }
15. return P;
}

```

Fig. 2. Original multi-cycle path analysis.

- Transient signal behavior at the input of FFs [9].

The first and the second ones make the analysis conservative. Therefore detected paths are always multi-cycle paths. We omit the first one in this paper. To handle the second one efficiently, we propose local FSM implication in Section IV. The third one may make the analysis optimistic. In [9] a method for handling this problem is shown. We do not consider the problem in this paper.

D. Implication-based Multi-Cycle Path Analysis

In this subsection we review implication-based multi-cycle path analysis proposed in [9].

Implication procedure is to assign as many mandatory values as possible by propagating already assigned values. The procedure is widely used in most ATPG systems. In [9] it is experimentally shown that most of the multi-cycle paths can be detected by only the implication procedures.

The overall flow of the implication-based multi-cycle path analysis algorithm is shown in Fig.2. In the figure, procedure *StructuralAnalysis* drops every FF pair that there is no combinational path between them. Procedure *RandomPatternSimulation* carries out random pattern simulation to drop FF pairs not satisfying the MC condition.

We shall now illustrate the main procedure *Implication* by using the circuit in Fig.1 again. Consider FF pair (FF_1, FF_2) . First the combinational logic part of the circuit is expanded into two timeframes as shown in Fig.3. Let us choose assignment $(FF_1(t), FF_2(t+1)) = (0,0)$. Value $FF_1(t)$ ($= 1$) is assigned to $FF_1(t+1)$ to analyze such the case that a rise transition occurs at FF_1 at time $t+1$. Fig. 3 shows the value assignment after implication procedure. In the figure, a signal value in a circle represents the value assigned before the implication procedure and the other values represent implied values.

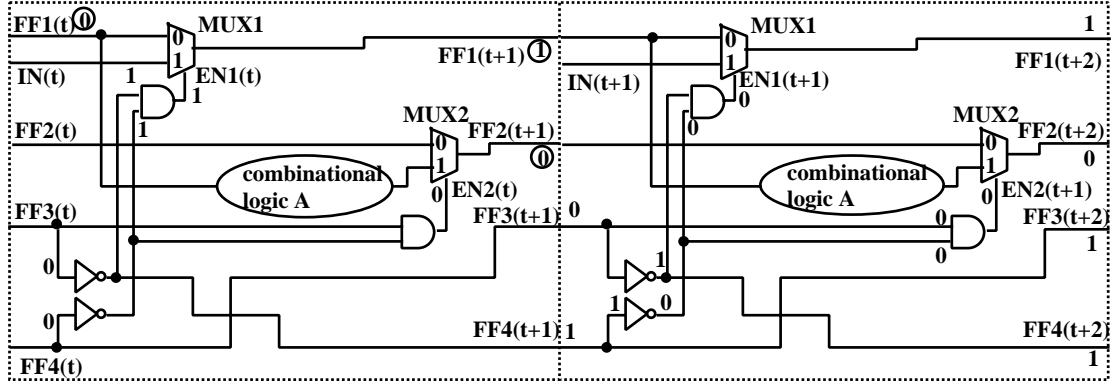


Fig. 3. Example of implication procedure.

Since we obtain $FF_2(t+2) = 0$, we can say that the signal at FF_2 never change at time $t+2$. Thus the MC condition is satisfied. This implies that 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, same analysis is done for 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 Procedure ATPG is applied. The procedure try to find such a pattern that violates the MC condition.

Actually this procedure is to detect 2-or-more-cycle paths. One can detect k -or-more-cycle paths ($k = 3, 4, \dots$) by extending k timeframes.

III. AUTOMATIC CONVERSION OF INDUSTRIAL SEQUENTIAL CIRCUITS INTO FUNDAMENTAL ONES

In this section we show how to convert industrial designs into fundamental sequential circuits to apply implication-based multi-cycle path analysis.

The inputs of the system is as follows: an HDL netlist, cell libraries, and timing constraint scripts for STA. Clock information is given by timing constraint scripts.

We automatically convert netlists into fundamental sequential circuits, which contain only the following components:

- Simple combinational logic gates (AND, OR, NAND, NOR, NOT, XOR, XNOR),
- Flip-flops with only one input and one output,
- Tri-state buffers,
- An implicit single clock driving the circuit, and
- Black boxes.

The inputs and the outputs of a black box are treated as pseudo-primary outputs and pseudo-primary inputs of the circuit respectively.

A. How to Handle Multiple Clocks

Typically industrial designs include more than one clock. These clocks can be partitioned into *clock groups*. Every pair

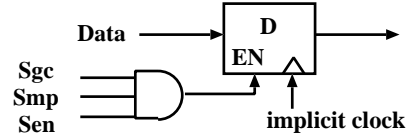


Fig. 4. Converted FF.

of the clocks in a clock group is synchronous. We assume that clock groups are specified in the given timing constraint scripts. We assume that every pair of clocks in different clock groups are asynchronous.

Our basic strategy to handle multiple clocks is as follows:

- Clock groups are processed one by one. Thus the analysis is done as many times as the number of the clock groups in a given circuit. In each analysis FFs driven by any clock groups other than the target clock group are considered as black boxes, because they are not synchronized with the target clock group.
- In each clock group, phases and clock cycles of the clocks may differ. Furthermore gated clocks may exist. We automatically convert a clock group into the following components: (1) A single implicit clock driving all the FFs, (2) Enabling conditions generated by a counter indicating the current phase.

The clock cycle of the single implicit clock driving all the FFs is the greatest common divisor of all the clock cycles and the differences of the phases. The counter counts up to the least common multiple of the clock cycles and the phase differences. Gated clocks are also converted into enabling conditions for FFs. An converted FF is shown in Fig.4. In the figure, S_{gc} , S_{mp} , and S_{en} are enable signals corresponding to the gated clock, the multi-phase clock, and the enable input of the original FF respectively.

IV. LOCAL FSM LEARNING

As we mentioned in Subsection II.C, one of the most optimistic feature of the analysis in [9] comes from the lack of

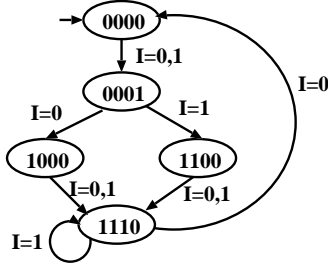


Fig. 5. State transition graph of FSM M .

the reachable state information. Reachable state information enables us to analyze multi-cycle paths more accurately, because we do not have to check the MC condition in Subsection II.C for the unreachable states. There are several issues, however, to use reachable state information in large industrial designs. First, reachability analysis of a the whole of a large sequential circuit is not practical because of the intractability of the problem. Second, reachable state information is usually represented by sets of the reachable states or state transition graphs. We need to choose better representation which fits into the implication-based analysis. To settle these issues, we propose local FSM learning.

A. Local FSM Identification

For local FSM learning, we need to identify local FSMs beforehand. A large sequential circuit, in general, consists of a lot of small local FSMs, instead of a single large FSM. Identifying FSMs in fundamental sequential circuits is difficult because there is no hints. We use names of instances of FFs in industrial designs as hints about identifying local FSMs. The identification is fully automatic. Each local FSM is usually small enough to enumerate the reachable states implicitly by using BDDs [10, 11].

Similar names of instances of FFs do not necessarily imply a local FSM. Though wrong groups of FFs may be identified as an FSM, even in such a case, the resulting implication relations are valid, because that information are actually extracted from the identified group of FFs. We limit the number of FFs for FSMs and skip reachability analysis for larger FSMs. We also limit BDD sizes during reachability analysis.

B. Local FSM Learning

One of the most important issues of local FSM learning is how to use reachable state information of local FSMs in implication-based multi-cycle path analysis. Once a local FSM is identified, reachability analysis is applied to the FSM. In this subsection we propose a method for extracting implication relations by using reachability analysis.

We can extract two kinds of implication relations: the one is implication relations in the same timeframe, and the other is those across timeframes.

For example, consider FSM M with the state transition graph as shown in Fig.5. Here I is the input of M and FF_4, FF_3, FF_2, FF_1 are FFs in M . The reset state is 0000. The

reachable state set of M is

$$(FF_4, FF_3, FF_2, FF_1) = \{0000, 0001, 1000, 1100, 1110\}.$$

An example of the generated implication relations between FFs in the same timeframe is as follows:

$$\{FF_1(t) = 1 \Rightarrow FF_4(t) = FF_3(t) = FF_2(t) = 0\}.$$

This relation is satisfied in every timeframe t .

Extracted implication relationships can be treated in the same way as those generated by static learning.

Consider FSM M again. M goes to state 1110 two clock after it goes to state 0001. Thus state 0001 at time t implies state 1110 at time $t + 2$. This is an example of implication across timeframes. Actually this relation can be simplified as follows:

$$FF_1(t) = 1 \Rightarrow FF_2(t + 2) = 1,$$

because the following condition is satisfied:

$$FF_1(t) = 1 \Rightarrow FF_4(t) = FF_3(t) = FF_2(t) = 0.$$

Thus, in this case, extracted implication relation is normal implication relation between two signals. In general, they are relations from a state vector to signals. To handle these relations, the data structure of implication relations should be extended so that implied values are attached to state vectors instead of signals.

V. FF ENABLE LEARNING

FF enable inputs give us useful information on the condition under which values of FFs change. In particular, the information is important in our method, because multiple clock information and gated clock information are converted into enable inputs. When FF enable inputs are shared by many FFs, learning of implication relations between FF enable inputs prunes the search space.

When enable inputs EN_i, EN_j of FF_i, FF_j satisfy the following condition, the FF pair always satisfy the MC condition:

$$EN_i(t) = 1 \Rightarrow EN_j(t + 1) = 0.$$

Here it is assumed that the enable inputs are active when the value is 1. We call the condition as EN-MC condition. Apparently, EN-MC condition is a sufficient condition for MC condition but the converse is not always true.

In the original algorithm, the MC condition is checked for each combination of rising and falling transition at the source and the sink as shown in Subsection II.D. If the EN-MC condition is checked in the procedures, directions of the transitions at the source and the sink do not necessarily be taken into account.

For example, consider the circuit shown in Fig.1 again. When one recognizes EN_1 and EN_2 as the enable inputs for sources FF_1 and FF_2 respectively, one can say that the FF pair is a multi-cycle path without specifying the values of FF_1 and FF_2 .

A verified implication relation between FF enable inputs are stored to use it later. We call this procedure FF enable learning.

MCPSet **MCPAnalysis**(Circuit C , Library L , Clock CK) {

1. $C \leftarrow \text{Flattening}(C)$;
2. $CC \leftarrow \text{Conversion}(C, L, CK)$;
/* CC is the circuit after conversion */
3. LocalFsmLearning(CC);
4. $P \leftarrow \text{StructuralAnalysis}(CC)$;
/* P is the set of the candidate MC pairs */
5. $P \leftarrow \text{RandomPatternSimulation}(CC)$;
6. **for** each $p \in P$ {
7. EN-MCAnalysis(CC, p);
8. **if** p is EN-MC pair **then** {
9. LearnEN-MC(p);
10. **continue**;
11. }
12. ATPG(CC, p);
13. **if** pattern found or aborted **then**
14. $P \leftarrow P \setminus \{p\}$; /* single cycle or aborted */
17. }
18. **return** P ;
- }

Fig. 6. Enhanced multi-cycle path analysis.

This kind of learning is useful for datapath parts, because the FFs of one word often share the same enable input.

The overall flow of the algorithm for industrial designs is shown in Fig.6. Procedures added or changed from Fig.2 are underlined. Procedure Flattening flattens module hierarchy. Procedure Conversion converts the original industrial designs into fundamental sequential circuits as shown in Section III. Procedure EN-MCAnalysis is to check the EN-MC condition. If an enable input pair of FF pair p is learned, learned results are used. Otherwise It uses Procedures Implication and ATPG. The iterations of assignment for $(FF_i(t), FF_j(t+1))$ in Fig.2 are implemented by backtracking in Procedure ATPG to use value assignments by Procedure EN-MCAnalysis.

VI. 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 SPARC64 IV(702MHz, main memory 8GB) + Solaris 8 + gcc version 2.95.3 with -O2 option. ATPG techniques are based on [12]. In the experiments, random pattern simulation was continued until no FF pairs were dropped during $32 \text{ (bit)} \times 10 \text{ (word)}$ consecutive patterns. The number of backtracks in ATPG was limited to 50. The experiments were done on three industrial designs. Table I shows the characteristics of the designs. In Table I Columns “#cell”, “#PI”, “#PO”, “#FF”, “#ck grp”, and “#ck” show the numbers of the cells, the primary inputs, the primary outputs, the FFs, the clock groups, and the clocks respectively. Cells include not only simple gates but also complex cells such as full adders, multiplexers, and RAMs. The number of the

TABLE I
CHARACTERISTICS OF THE EXAMPLES

circuit	#cell	#PI	#PO	#FF	#ck grp	#ck
EX1	36,489	101	87	8,672	2	2
EX2	33,522	209	277	5,586	3	12
EX3	241,707	115	115	24,827	6	43

TABLE III
COMPARISON OF CPU TIMES

circuit	ck grp	CPU(sec)			
		wo FSM wo EN	w FSM wo EN	wo FSM w EN	w FSM w EN
EX1	A	525	588	340	456
	B	101	116	76	103
EX2	A	1,597	2,015	798	1,007
	B	83	83	77	84
	C	26	29	26	26
EX3	A	63,834	64,309	32,367	30,359

clocks does not include the number of gated clocks or inverted clocks.

Table II shows the experimental results of detecting multi-cycle FF pairs. In Table II Column “ck grp” shows the names of the clock groups. For “EX3”, only the clock group driving the largest number of FFs is listed. Column “#black box” shows the number of the cells treated as black boxes in the analysis. Column “#local FSM” shows the number of the identified local FSMs. Column “#added imp” shows the added implication relations. In the experiments, implication relations across timeframes were not considered. Columns “#remaining FF pairs before rsim” and “#remaining FF pairs after rsim” show the numbers of the remaining FF pairs before and after random pattern simulation. Column “#MC pairs” shows the number of the multi-cycle FF pairs detected by the proposed method. Column “#aborted pairs” shows the number of the aborted FF pairs because of backtrack limits. Column “abort ratio” indicates how many FF pairs were aborted because of backtrack limits among the remaining FF pairs before random pattern simulation. Column “CPU(sec)” shows total CPU times in seconds.

Table II indicates that our method can handle industrial designs in reasonable CPU time. The ration of aborted FF pairs are low enough, below 1%. The number of MC paths are approximately proportional to the number of FFs. CPU times largely depend on the number of the remaining FF pairs after random pattern simulation.

Table III shows how the proposed techniques affect CPU times. In the table, “w FSM” and “wo FSM” represent analysis with and without local FSM learning respectively. Similarly, “EN” represents FF enable learning. As for FF enable learning, CPU times are much reduced. Computation costs for local FSM learning are not so high. Table IV shows how the numbers of MC FF pairs are affected by local FSM learning. Local FSM learning can detect more multi-cycle paths, espe-

TABLE II
EXPERIMENTAL RESULTS

circuit	ck grp	#FF	#black box	#local FSM	#added imp	#remaining FF pairs		#MC pairs	#aborted pairs	abort ratio	CPU (sec)
						before rsim	after rsim				
EX1	A	7,584	2,400	39	54	88,350	7,710	3,958	811	0.9%	456
	B	1,022	14,168	19	1	13,964	3,021	367	76	0.5%	103
EX2	A	4,966	1,395	43	1,036	216,468	151,334	6,128	555	0.3%	1,007
	B	646	11,771	4	2	7,329	2,326	32	14	0.2%	84
	C	10	13,074	1	16	53	53	47	0	0%	26
EX3	A	10,125	34,699	63	365	750,762	366,215	16,647	6,900	0.9%	30,359

TABLE IV
COMPARISON OF THE NUMBER OF MC FF PAIRS

circuit	ck grp	#MC pair		#aborted pair	
		wo FSM	w FSM	wo FSM	w FSM
EX1	A	3,875	3,915	892	870
	B	367	367	76	76
EX2	A	5,283	6,128	1,220	555
	B	10	32	7	14
	C	44	47	0	0
EX3	A	16,481	16,647	5,478	6,900

cially for “EX2”.

VII. CONCLUSIONS

We have shown how to improve implication-based multi-cycle path analysis to handle industrial designs. We have proposed local FSM learning to take into account reachability information and FF enable learning to accelerate multi-cycle path analysis. Experimental results are encouraging in terms of the quality and the efficiency of multi-cycle path analysis for industrial designs.

REFERENCES

- [1] D. Brand and V. S. Iyengar. Timing analysis using functional relationships. In *Proceedings of IEEE International Conference on CAD-86*, pages 126–129, November 1986.
- [2] H.-C. Chen and D. H.-C. Du. Path sensitization in critical path problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 12(2):196–207, February 1993.
- [3] S. Devadas, K. Keutzer, and S. Malik. Computation of floating mode delay in combinational circuits: Theory and algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 12(12):1913–1923, December 1993.
- [4] 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.
- [5] A. P. Gupta and D. P. Siewiorek. Automated multi-cycle symbolic timing verification of microprocessor-based designs. In *Proceedings of the 31st ACM/IEEE Design Automation Conference*, pages 113–119, 1994.
- [6] K. Nakamura, K. Takagi, S. Kimura, and K. Watanabe. Waiting false path analysis of sequential logic circuits for performance optimization. In *Proceedings of IEEE/ACM International Conference on CAD-98*, pages 392–395, November 1998.
- [7] K. Nakamura, S. Maruoka, S. Kimura, and K. Watanabe. Multi-cycle path detection based on propositional satisfiability with cnf simplification using adaptive variable insertion. *IEICE Trans. on Fundamentals*, E83-A(12):2600–2607, December 2000.
- [8] W.-C. Lai, A. Krstic, and K.-T. Cheng. Functionally testable path delay faults on a microprocessor. *IEEE Design & Test of Computers*, oct 2000.
- [9] H. Higuchi. An implication-based method to detect multi-cycle paths in large sequential circuits. In *Proceedings of the 39th ACM/IEEE Design Automation Conference*, pages 164–169, June 2002.
- [10] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.
- [11] C. Berthet O. Coudert and J. C. Madre. Verification of sequential machines using boolean functional vectors. In *Proc. IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, November 1989.
- [12] Y. Matsunaga and M. Fujita. Enhanced unique sensitization for efficient test generation. *IEICE Trans. on Information and Systems*, E76-D(9):1114–1120, September 1993.