Multi-Clock Path Analysis Using Propositional Satisfiability

Kazuhiro Nakamura Shinji Maruoka Shinji Kimura Katsumasa Watanabe

Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0101 JAPAN
Tel: +81-743-72-5306
Fax: +81-743-72-5309
e-mail: {kazuhi-n,sinji-m,kimura,watanabe}@is.aist-nara.ac.jp

Abstract— We present a satisfiability based multi-clock path analysis method. The method uses propositional satisfiability (SAT) in the detection of multi-clock paths. We show a method to reduce the multi-clock path detection problems to SAT problems. We also show heuristics on the conversion from multi-level circuits into CNF formulae. We have applied our method to ISCAS89 benchmarks and other sample circuits. Experimental results show the improvement on the manipulatable size of circuits by using SAT.

I. Introduction

The clock frequency of a sequential logic circuit is decided based on the maximum delay time of the combinational parts in the circuit. The precise estimation of the maximum delay time is important in deciding the proper clock frequency[14].

In logic circuits, there may exist paths where the signal propagation can use 2 or more clock cycles, which are called multi-clock paths(or multi-cycle paths)[4],[2],[3]. In other words, each path in the circuit has the number of allowable clock cycles to propagate signals, and if the number is 2 or more, then the delay time of the path can be greater than the clock period. Hence multi-clock paths should be considered in the decision of clock frequency, timing verification, and logic optimization.

There have been proposed two multi-clock path detection methods[4],[2]. One is based on the analysis of state transition graphs of the controllers of microprocessors[4], the other is based on the symbolic state traversal of finite state machines (FSMs)[2]. Both of them detect multi-clock paths based on state traversal of finite state machines, and can only be applied to rather small circuits.

Propositional satisfiability (SAT) is a method to decide the satisfiability of Boolean formulae, and has been used in logic comparison and test pattern generation[5]. Recently, SAT is applied to the analysis of sequential circuits, such as symbolic model checking verification[1] and timing analysis of combinational circuits[10].

In this paper, we present a SAT-based multi-clock path analysis method. The method generates a propositional formula which is satisfiable if and only if the path is not a multi-clock path, and checks the satisfiability of the formula with a propositional prover. We introduce heuristics on the conversion from multi-level circuits to CNF formulae.

We have implemented our method and tested it on ISCAS89 benchmarks[8] and other sample circuits. Experimental results show the improvement on the manipulatable circuit size.

This paper is organized as follows. In the next section, we show preliminaries. In Section III, we present multi-clock path analysis. In Section IV, we present multi-clock path analysis using propositional satisfiability. In Section V, we present heuristics on the translation from a multi-level circuit into a CNF formula with taking the level of a gate into account. In Section VI, we show the experimental results.

II. Preliminaries

In this section, we show definitions of sequential circuits and propositional satisfiability.

A. Finite State Machine(FSM)

We show a definition of an FSM.

Definition 1 An FSM M is a 6-tuple $(S, \Sigma, \Gamma, \delta, \lambda, q_0)$ where

- S is a finite set of states,
- Σ is an input alphabet,
- Γ is an output alphabet,
- $\delta : S \times \Sigma \to S$ is a state transition function,
- $\lambda : S \times \Sigma \to \Gamma$ is an output function,
- $q_0 (\in S)$ is the initial state.

The behavior of M with respect to an input sequence $a_1 a_2 \ldots a_n$ $(a_i \in \Sigma)$ is a sequence of states $q_0 q_1 \ldots q_n$ $(q_i \in S)$ and a sequence of outputs $o_1 o_2 \ldots o_n$ $(o_i \in \Gamma)$, where each of the states and the outputs satisfies $q_i = \delta(q_{i-1}, a_i)$ and $o_i = \lambda(q_{i-1}, a_i)$.

Let Σ^* be the set of all input sequences over Σ, and let Σ^k be a set of input sequences with length k. We use ε as the sequence whose length is 0.

To represent the behavior of M, the domain of δ is extended, and $\delta^* : S \times \Sigma^* \to S$ is introduced.

Definition 2 $\delta^* : S \times \Sigma^* \to S$ is defined as follows:

- $\delta^*(q, \varepsilon) = q$
- $\delta^*(q, wa) = \delta(\delta^*(q, w), a),\ (a \in \Sigma, w \in \Sigma^*)$

The set RS of reachable states from the initial state q_0 is defined as follows.

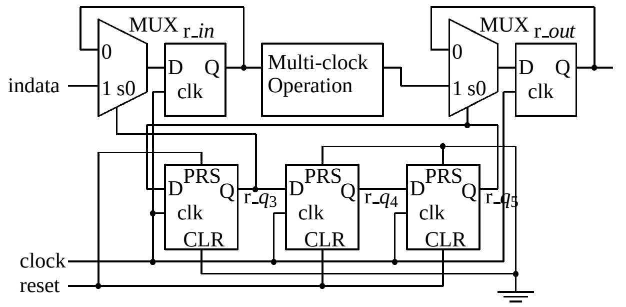


Fig. 1. Example of a multi-clock path.

Definition 3 (Reachable states from the initial state)

$$RS = \left\{ q \mid \exists w \in \Sigma^*,\ q = \delta^*(q_0, w) \right\}$$

Note that a symbol $a_i \in \Sigma$ corresponds to a tuple of values of primary inputs of sequential circuits, and a state $q_j \in S$ corresponds to a tuple of values of registers in the circuit.

B. Propositional Satisfiability

Let x_i be a propositional variable, and $\overline{x}_i$ be the negation of x_i. A variable and its negation are called *literals*. The disjunction (OR, $\vee$) of *literals* such as $x_1 \vee x_2$ and $x_1 \vee \overline{x}_2 \vee x_3$ is a *clause*. The conjunction of *clauses* such as $(x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2 \vee x_3)$ is a conjunctive normal form (CNF) formula. A CNF formula f over $x_1, ..., x_l$ is a CNF formula including only *literals* $x_1, \overline{x}_1, x_2, \overline{x}_2, ..., x_l, \overline{x}_l$.

Let $(x_1, ..., x_l)$ be a tuple of propositional variables and f be a CNF formula over $x_1, ..., x_l$. We can assign T(true) or F(false) for x_i. If x_i is assigned T(F), then $\overline{x}_i$ is assigned F(T). A *clause* has value T, if one of *literals* in the *clause* is assigned T. A CNF formula has value T, if all *clauses* have value T. A CNF formula f is *satisfiable* if f has value T with some value assignments for variables.

III. Multi-Clock Path Analysis

In this section we define multi-clock paths and show a detection method.

A. Multi-Clock Path

Multi-clock paths are the paths where the propagation of signals can use 2 or more clock cycles. Fig.1 shows an example of multi-clock paths. The upper part of Fig.1 shows the data path, and the lower part is the control registers. The initial state of the control registers after the reset signal is $(\text{r\_}q_3, \text{r\_}q_4, \text{r\_}q_5) = (1, 0, 0)$, and the state of registers changes as $(0, 1, 0) \rightarrow (0, 0, 1) \rightarrow (1, 0, 0) \rightarrow (0, 1, 0) \rightarrow \ldots$ synchronized with the clock signal.

At the data path, "r_*in*" is set to "indata" when $\text{r\_}q_3 = 1$, and "r_*out*" is set to the value of the output of "Multi-clock Operation" when $\text{r\_}q_5 = 1$. Since the state transition is $(1, 0, 0) \rightarrow (0, 1, 0) \rightarrow (0, 0, 1)$, "r_*out*" is set after 2 clocks with respect to the time when "r_*in*" is set. Thus we can use 2 clock cycles for the computation of "Multi-clock Operation", and therefore the timing constraint of the path from "r_*in*" to "r_*out*" is $(\text{the path delay}) \leq 2 \times (\text{clock period})$.

In general, we have the following constraint for each path between registers.

$$\begin{aligned}&(\text{the path delay})\\ &\qquad \leq (\text{allowable clock cycles}) \times (\text{clock period})\end{aligned}$$

The number of allowable clock cycles can be 2 or more.

Note that the reachable state set plays a key part on the multi-clock path detection. For example, the reachable state set of the above example is $\{(1,0,0), (0,1,0), (0,0,1)\}$, and if all states can be reached, then the signal between r_*in* and r_*out* should be propagated within 1 cycle when $(1,1,1) \rightarrow (1,1,1)$.

B. Detection Method Based on Symbolic Execution of FSM

Given a circuit, multi-clock paths are detected by analyzing FSM model of the circuit. At first, we compute the set *RS* of reachable states from the initial state.

Let r_{in} and r_{out} be registers, and there be a path from r_{in} to r_{out}. If the value of r_{out} does not change at the next clock of the clock when the value of r_{in} has just changed, then the propagation of signals can use 2 or more clock cycles. Note that the change of r_{in} does not contribute to the value of r_{out}. We obtain the following condition specifying the multi-clock property on a path from r_{in} to r_{out}:

$$\begin{aligned}&\forall q, q', q'' \in RS,\ \forall a, a' \in \Sigma,\\ &\quad \Big[\big(q' = \delta(q,\ a)\big) \wedge \big(q'' = \delta(q',\ a')\big) \qquad \text{(A)}\\ &\qquad \rightarrow \Big(\big(q(r_{in}) \neq q'(r_{in})\big) \rightarrow \big(q'(r_{out}) = q''(r_{out})\big)\Big)\Big].\end{aligned}$$

Formula (A) denotes that if the value of r_{in} changes at the state transition from q to q', then the value of r_{out} does not change at the state transition from q' to q'', where q, q' and q'' are reachable from the initial state. Similarly, the condition specifying the path can use 3 or more clock cycles can be obtained by extending Formula (A).

In the following, we focus on Formula (A) specifying the allowable clock cycles to be 2 or more. At present, the symbolic state traversal of the FSM is the best way to compute *RS*, and a BDD-based multi-clock path detection method has been proposed[2]. However, the method is hard to apply to large circuits. Hence, we relax the above condition a little and check the relaxed condition using propositional satisfiability.

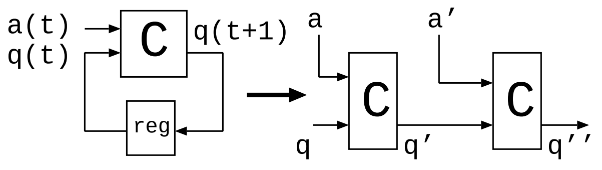
IV. Multi-Clock Path Analysis Using Satisfiability

In this section, we describe an algorithm that detects multi-clock paths based on propositional satisfiability.

SAT is not suitable for computing the reachable state set. Thus we abandon the computation and assume that all states are reachable from the initial state. The following formula is used in the multi-clock path analysis instead of Formula (A).

$$\begin{aligned}&\forall q, q', q'' \in S,\ \forall a, a' \in \Sigma,\\ &\quad \Big[\big(q' = \delta(q,\ a)\big) \wedge \big(q'' = \delta(q',\ a')\big) \qquad \text{(B)}\\ &\qquad \rightarrow \Big(\big(q(r_{in}) \neq q'(r_{in})\big) \rightarrow \big(q'(r_{out}) = q''(r_{out})\big)\Big)\Big].\end{aligned}$$

Since Formula (B) takes into account all states, there is a possibility that several multi-clock paths can not be detected by the analysis as we described in III.A.



(a) A sequential circuit (b) A time expansion

Fig. 2. Time expansion of a sequential circuit

A. Time Expansion

The analysis starts from the time expansion of sequential circuits, where a sequential circuit is translated into a multi-level combinational circuit. Fig.2 shows the idea of the time expansion: (a) is a sequential circuit and (b) is a time expanded combinational circuit. Since we consider Formula (B), the duplication of a circuit as shown in Fig.2(b) is enough. Symbols in Fig.2(b) correspond to those in Formula (B), where "C" implements the state transition function δ.

B. Propositional Formula for Multi-Clock Path Analysis

Based on Formula (B), we define a propositional formula F for detecting multi-clock paths.

Let r_{in} and r_{out} be the input and output registers to be checked. The following propositional formula F becomes true if the path from r_{in} to r_{out} can use 2 clock cycles:

$$F \overset{\triangle}{=} T(a,\, q,\, q') \wedge T(a',\, q',\, q'') \to \Big((q(r_{in}) \not\equiv q'(r_{in})) \to (q'(r_{out}) \equiv q''(r_{out}))\Big), \quad \text{(C)}$$

where $T(a,\, q,\, q') = 1$ iff $q' = \delta(q, a)$ and $T(a',\, q',\, q'') = 1$ iff $q'' = \delta(q', a')$, and $q(r)$ denotes the value of a register r under the state q. Note that $T(a,\, q,\, q')$ represents a logic formula which is implemented by a multi-level circuit as in Fig.2, where a, q and q' are binary coded and propositional variables have been assigned. Also note that $q(r_{in})$ and $q(r_{out})$ are propositional variables corresponding to r_{in} and r_{out} in the tuple of the state variables for the state q.

If $\overline{F}$ is unsatisfiable, then F is always true for any q, a and a', and therefore we can decide that the allowable clock cycle is 2 or more. We translate $\overline{F}$ into CNF formulae, and check the unsatisfiability of $\overline{F}$ using a SAT prover. We can obtain CNF of $\overline{F}$ using a translation rule $a \not\equiv b \Rightarrow (a \vee b) \wedge (\overline{a} \vee \overline{b})$:

$$\begin{aligned}\overline{F} &= T(a,\, q,\, q') \wedge T(a',\, q',\, q'') \\ &\wedge (q(r_{in}) \vee q'(r_{in})) \wedge (\overline{q(r_{in})} \vee \overline{q'(r_{in})}) \\ &\wedge (q'(r_{out}) \vee \overline{q''(r_{out})}) \wedge (\overline{q'(r_{out})} \vee q''(r_{out})).\end{aligned} \quad \text{(D)}$$

Note that $T(a,\, q,\, q')$ and $T(a',\, q',\, q'')$ are the same logic formula with different variables. In the following we discuss a method to generate CNF of $T(a,\, q,\, q')$.

V. Conversion to CNF formula

In this section we show a method to translate from multi-level combinational circuits into CNF formulae.

In general, CNF formulae can be generated from multi-level combinational circuits using the following translation rules: $(a \not\equiv b) \Rightarrow (a \vee b) \wedge (\overline{a} \vee \overline{b})$, $(a \equiv b) \Rightarrow (a \vee \overline{b}) \wedge (\overline{a} \vee b)$ and $(a \wedge b) \vee c \Rightarrow (a \vee c) \wedge (b \vee c)$. The size of resulting CNF formulae may grow exponentially, and we should introduce heuristic techniques in the conversion. For example, let f_1 and f_2 be sub-formulae which are not CNF, and we consider a formula $(f_1 \not\equiv f_2)$. The formula is translated into $(f_1 \vee f_2) \wedge (\overline{f_1} \vee \overline{f_2})$ using the above rules, and the size of the formula can be twice. If f_2 is $f_{21} \wedge f_{22}$, then f_1 should be duplicated and the size of CNF can be exponential with respect to the size of the original formula in the worst case.

To avoid this problem, we apply a method which introduces new propositional variables and reduces the size of CNF formulae without affecting the satisfiability of formulae[13]. For example, if we introduce new variables x_1 and x_2 for f_1 and f_2, then we obtain an equivalent formula $(x_1 \not\equiv x_2) \wedge (x_1 \equiv f_1) \wedge (x_2 \equiv f_2)$. In the translation of $x_1 \equiv f_1$ into CNF, the size of CNF formulae does not grow exponentially, since x_1 is a propositional variable. Note that x is manipulated as a primary input variable.

In [13], new variables are inserted for all logic gates. We adapt the translation method, and improve the method by introducing new heuristic techniques on the insertion of new variables based on the depth of logic gates from primary output and newly introduced variables. In the following, we assume that multi-level combinational logic circuits consist of AND, OR, NOT and XOR($\not\equiv$) gates.

To control the number of introduced variables and the size of generated CNF formulae, we introduce heuristic techniques. Fig.3 shows the idea. We classify 4 cases as shown in the figure, and introduce 4-level parameters. In Fig.3, "v" denotes logic gates which we focus on, "Var" denote new variables which we introduced. "XOR-level", "OR-level", "AND-level" and "NOT-level" are thresholds that denote the maximum depth from introduced variables. We introduce new variables at the focusing gate v when the number of levels of "v" from the nearest variables exceeds the thresholds. By this, the circuit is divided into sub-circuits with almost the same level. For example, the OR gate in Fig.3(b) is considered. If the level of the OR gate from the nearest variable is over "OR-level" and "pre_v" is not a NOT gate, then we introduce new variables for fan-ins of the OR gate. We show the effect of this method on reducing CNF size in Sect. VI.

Fig.4 shows an algorithm for introducing new variables. The analysis starts from the primary outputs of multi-level logic circuits, and goes backward toward primary inputs based on the depth first search algorithm. At first *lev* is set to 1, and then *lev* is incremented for every recursive calls of `DepthFirst()`. If the value of *lev* exceeds the threshold decided by the type of gates, then new variables are inserted for fan-ins by `NewVar()`.

VI. Experimental Results

We have implemented a SAT-based multi-clock path analyzer in C language and compared it with a symbolic execution based multi-clock path analyzer using BDD's[2]. We have also implemented BDD-based analyzer without reachability. The SAT-based analyzer reads circuit descriptions in SLIF[16], and produces CNF formulae in DIMACS format[9] for detecting multi-clock paths. The satisfiability of the formula is checked

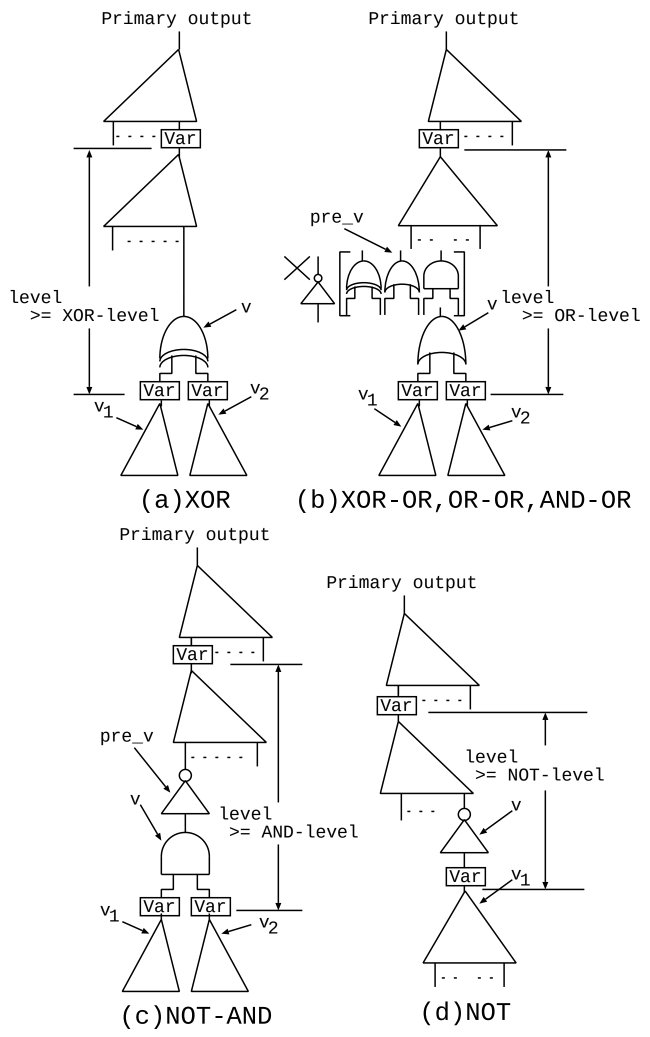


Fig. 3. Internal variables

with H. Zhang's SATO[6] / J. Silva's GRASP[15]. Both of the SAT provers are based on Davis-Putnam method[7].

We have analyzed the ISCAS89 benchmarks[8] and other sample circuits designed by our laboratory on a PC (CPU Pentium II 500MHz, Main memory 512MB).

A. Application to ISCAS Benchmarks

We have applied 3 analysis methods to 30 ISCAS89 benchmarks. (1) Method1: BDD-based algorithm which takes into account the reachable state set[2]. (2) Method2: BDD-based algorithm which assumes that all states are reachable, (3) Method3: SAT-based algorithm which assumes that all states are reachable. We used SATO as a SAT prover.

TableI shows the statistics. In the table, "#In", "#Reg" and "#R-pair" are the number of primary inputs, registers and connected pairs of registers in the circuit respectively. "#Rep", "#M-pair" and "Time" are the number of repetitions in computing reachable states of the circuit, the number of pairs of registers whose path is a multi-clock path(2-cycle), and the elapsed CPU seconds obtained by the time command respectively. "#Var", "#Cl" and "#Lit" are the maximum number of propositional variables, clauses and literals of the formula which is used in our method. Multi-clock paths on 21 circuits are found by Method1[2]: ex2, ex3, ex4, ex5, ex6, ex7, s208, s298, s344, s349, s382, s386, s420, s444, s510, s526, s526n, s641, s713, s838, s953. Multi-clock paths on only 16 circuits can be found by Method2 and Method3: ex2, ex3, ex4, ex5, ex6, ex7, s298, s344, s349, s382, s386, s444, s510, s526, s526n, s953. Because of the effect of unreachable states, multi-clock paths in s208, s420, s641, s713 and s838 can not be found by Method2 and Method3, and the total number of paths is small on Method2 and Method3 for detected circuits. The total number of "#M-pair" in Method1 is 1101, that in Method2 and Method3 is only 590 and 636. Note that s1423 has memory overflow on BDD-based methods, and Method3 can detect s1423. The elapsed CPU time of Method3 is the smallest.

```
DepthFirst ($v, pre\_v, lev$)
  $v$:  a gate;
  $pre\_v$:  the previous gate of $v$;
  $lev$:  a level from $v$ to variable;
  $v_1, v_2$:  the next gate of $v$;
  XOR-level, OR-level,
    AND-level, NOT-level:  thresholds;
begin
  switch ($v$)
    case ($v$ == Var) :  return;
    case ($v$ == XOR): /*Fig.3(a)*/
      if (XOR-level <= $lev$)
        NewVar($v_1$); NewVar($v_2$); $lev$ = 1;
      else
        $lev$ = $lev$ + 1;
      DepthFirst($v_1, v, lev$); DepthFirst($v_2, v, lev$);
      return;
    case ($v ==$ OR and $pre\_v \neq$ NOT): /*Fig.3(b)*/
      if (OR-level <= $lev$)
        NewVar($v_1$); NewVar($v_2$); $lev$ = 1;
      else
        $lev$ = $lev$ + 1;
      DepthFirst($v_1, v, lev$); DepthFirst ($v_2, v, lev$);
      return;
    case ($v ==$ AND and $pre\_v ==$ NOT): /*Fig.3(c)*/
      if (AND-level <= $lev$)
        NewVar($v_1$); NewVar($v_2$); $lev$ = 1;
      else
        $lev$ = $lev$ + 1;
      DepthFirst($v_1, v, lev$); DepthFirst($v_2, v, lev$);
      return;
    case ($v$ == NOT) : /* Fig.3(d) */
      if (NOT-level <= $lev$)
        NewVar($v_1$); $lev$ = 1;
      else
        $lev$ = $lev$ + 1;
      DepthFirst ($v_1, v, lev$);
      return;
  end switch;
end
```

Fig. 4. An algorithm for introducing new variables.

B. Application to Sample Circuits

We have applied our SAT-based method and the BDD-based method in [2] to several circuits which are designed in our laboratory: prime250 (a prime number generator which computes prime numbers less than 250, about 400 gates), s-div8 (an 8-bit sequential divider, about 500 gates), prime999 (a prime number generator which computes prime numbers less than 999, about 400 gates), bezier8 (an 8-bit bezier curve generator, about 900 gates), forsen (an edge detection circuit for im-

TABLE I
RESULT OF MULTI-CLOCK PATH ANALYSIS OF ISCAS BENCHMARK CIRCUITS.

Circuits				Method1 (Reachability, BDD)			Method2 (BDD)		Method3 (SAT)				
Name	#In	#Reg	#R-pair	#Rep	#M-pair	Time	#M-pair	Time	#M-pair	#Var	#Cl	#Lit	Time
daio	1	4	6	5	0	0.8	0	2.8	0	18	46	124	0.7
ex2	2	19	342	6	306	1.0	238	2.1	238	195	1250	3554	4.6
ex3	2	10	90	5	80	0.8	45	1.7	45	102	570	1626	1.2
ex4	6	14	169	14	135	0.9	135	1.7	135	116	436	1172	1.5
ex5	2	9	72	4	48	0.8	30	1.5	30	93	488	1380	1.0
ex6	5	9	61	1	61	0.5	9	1.6	9	99	638	2032	0.9
ex7	2	10	90	5	77	0.8	53	1.8	53	102	614	1816	1.2
s27	4	3	4	3	0	0.6	0	1.5	0	23	48	124	0.5
s208	11	8	28	17	18	0.9	0	1.6	0	78	240	700	0.5
s298	3	14	56	19	4	0.7	3	1.7	3	112	468	1438	0.9
s344	9	15	74	7	1	1.9	1	1.7	1	155	480	1296	1.3
s349	9	15	74	7	1	1.8	1	1.6	1	155	484	1312	1.3
s382	3	21	131	151	13	11.6	13	9.9	13	175	622	1738	2.0
s386	7	6	30	8	4	0.9	4	1.5	4	78	500	1824	0.9
s420	19	16	72	17	62	0.9	0	1.7	0	158	496	1420	1.1
s444	3	21	131	151	13	13.3	13	6.3	13	221	844	2392	2.6
s510	19	6	30	47	7	0.9	2	1.6	2	192	814	2588	1.4
s526	3	21	123	151	8	10.3	7	20.1	7	207	1030	3344	2.5
s526n	3	21	123	151	8	10.5	7	37.7	7	203	1024	3338	2.5
s641	35	19	100	7	38	9.3	0	5.3	0	259	656	1818	1.9
s713	35	19	100	7	38	10.2	0	6.0	0	293	844	2344	2.6
s820	18	5	20	11	0	0.9	0	1.5	0	221	1504	5768	1.7
s832	18	5	20	11	0	0.8	0	1.7	0	217	1538	5902	1.6
s838	35	32	160	17	150	1.1	0	101.1	0	318	1008	2860	3.0
s953	16	29	150	11	29	19.9	29	6.2	29	469	1702	4848	6.2
s1196	14	18	20	3	0	253.4	0	10.9	0	318	1290	3856	2.2
s1238	14	18	20	3	0	200.2	0	12.6	0	322	1338	4030	2.4
s1423	17	74	1694	—	Mem $>$ 1G	—	Mem $>$ 1G	—	46	654	2238	6298	60.4
s1488	8	6	30	23	0	0.9	0	1.6	0	222	1576	5836	2.0
s1494	8	6	30	23	0	0.8	0	1.7	0	222	1608	5980	2.0
Total number of M-pair				1101			590		636				

Mem $>$ 1G: we have analyzed on a DEC AlphaServer8400 (CPU alpha21164A 617MHz $\times$ 10, Main memory 8GB).

age processing, about 900 gates), pcpu16 (a 16-bit pipelined processor, about 9,000 gates), mul64 (a circuit including a 64-bit combinational adder array multiplier, about 47,000 gates), SpchRecog (a HMM-based speech recognition circuit, , about 40,000 gates). They have been designed to include multi-clock paths: those of forsen are to meet the clock constraints, those of SpchRecog are for memory access constraints, and those of others are for experiments.

We have found multi-clock register pairs on all circuits by Method3. TableII shows the statistics. Almost all circuits, BDD-based methods suffer from memory overflow or time-overflow. We used SATO as a SAT prover if not specified. For only SpchRecog, we should use GRASP to overcome the number of registers more than 1000.

C. Effects of Thresholds for Introducing New Variables

We have tested properties of thresholds which are used to introduce new variables in our algorithm.

TableIII shows statistics of multi-clock path detection of s1423 and mul16. In the table, "Threshold: a, b, c, d" denote "XOR-level", "OR-level", "AND-level" and "NOT-level" in Fig.3. Note that our algorithm with (1, 1, 1, 2) introduces variables for all logic gates. From these tables, the number of literals becomes the smallest when the 'Threshold" is (1, 4, 4, 5), and the elapsed CPU time is the shortest when the "Threshold" is (1, 9, 9, 10). The statistics show that the reduction of the size of CNF formula does not always contribute to reducing the time for analysis. Note that the statistics in TableI and TableII are obtained by using (1, 4, 4, 5) because the size of CNF is more important in the manipulation.

VII. CONCLUSIONS

In this paper, we have presented multi-clock path analysis method based on propositional satisfiability and shown experimental results.

The method reduces multi-clock path detection problems into SAT problems using the time expansion. The SAT-based algorithm enables us to apply multi-clock path analysis to large circuits that can not be analyzed with the symbolic execution based algorithm.

We have applied our method to ISCAS89 benchmarks and other sample circuits. Experimental results show the improvement on the manipulatable size of circuits by using satisfiability.

The problem is that the SAT-based algorithm can only detect a subset of multi-clock paths detectable by the symbolic

TABLE II

RESULTS OF MULTI-CLOCK PATH ANALYSIS OF SAMPLE CIRCUITS

Circuits				Method1 (Reachability, BDD)			Method2 (BDD)		Method3 (SAT)				
Name	#In	#Reg	#R-pair	#Rep	#M-pair	Time	#M-pair	Time	#M-pair	#Var	#Cl	#Lit	Time
prime250	2	35	718	11959	570	774.3	413	116606.6	413	435	1662	4768	17.4
s-div8	10	47	514	—	Mem>1G	—	—	Time>48h	33	557	2000	5620	20.5
prime999	2	44	1219	—	Mem>1G	—	—	Time>48h	480	550	2258	6498	37.1
bezier8	34	64	684	—	Mem>1G	—	Mem>1G	—	58	2121	9074	27264	37.1
forsen	187	255	1739	—	Mem>1G	—	Mem>1G	—	1739	2547	956	24922	199.9
pcpu16	40	779	6744	—	Mem>1G	—	Mem>1G	—	0	4675	22462	71762	16292.9
mul64	259	258	12866	—	Mem>1G	—	Mem>1G	—	12480	39666	160114	451304	97560.6
SpchRecog	37	1316	134443	—	Mem>1G	—	Mem>1G	—	57573	22256	105252	328272	547.5h†

Mem>1G: we have analyzed on a DEC AlphaServer8400 (CPU alpha21164A 617MHz × 10, Main memory 8GB).
SpchRecog: we have analyzed on 6 Sun Ultra2 workstations (UltraSPARC-IIi 333MHz, Main memory 512MB) in parallel.
†: The total CPU time using GRASP.

TABLE III

THE INFLUENCE OF THE THRESHOLD ON MULTI-CLOCK PATH ANALYSIS.

Threshold	Method3 (SAT)			
a, b, c ,d	#Var	#Cl	#Lit	Time
1, 14, 14, 15	414	4604	25222	95.7
1, 9, 9, 10	460	2248	7896	47.5
1, 4, 4, 5	654	2238	6290	57.8
1, 1, 1, 2	1256	3232	7500	83.1

(a) Statistics for s1423

Threshold	Method3 (SAT)			
a, b, c, d	#Var	#Cl	#Lit	Time
1, 14, 14, 15	2324	44034	440910	799.8
1, 9, 9, 10	2398	11164	35570	138.6
1, 4, 4, 5	2824	10890	30800	179.4
1, 1, 1, 2	4714	14084	35168	369.3

(b) Statistics for mul16

execution based algorithm. We should develop SAT-based manipulations of reachable states.

ACKNOWLEDGMENTS

We would like to express our thanks to Research Associate Takashi Horiyama, Qiang Zhu and other members of Watanabe Laboratory at Nara Institute of Science and Technology for their discussions and comments. We also thank to VDEC at the University of Tokyo for their support for the usage of CAD tools. The part of this work is supported by a Grant in Aid for Scientific Research from the Ministry of Education, Science, Sports and Culture, Specified Area Research 09245101 and 11480068, and a grant form Nikkei BP IP design award.

REFERENCES

[1] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic Model Checking using SAT procedures instead of BDDs," *Proc. of DAC'99*, June 1999.

[2] K. Nakamura, K. Takagi, S. Kimura and K. Watanabe, "Waiting False Path Analysis of Sequential Logic Circuits for Performance Optimization," *Proc. of ICCAD'98*, pp. 392-395, Nov. 1998.

[3] K. Nakamura, S. Kimura, K. Takagi and K. Watanabe, "Timing Verification of Sequential Logic Circuits Based on Controlled Multi-clock Path Analysis," *IEICE Transactions on Fundamentals*, Vol. E81-A NO. 12, pp. 2515-2520, Dec. 1998.

[4] Anurag P. Gupta and Daniel P. Siewiorek, "Automated Multi-Cycle Symbolic Timing Verification of Microprocessor-based Designs," *Proc. of 31st DAC*, pp. 113-119, 1994.

[5] Tracy Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Transaction on Computer-Aided Design*, pp. 4-15, Jan. 1992.

[6] H. Zhang, "SATO: An Efficient Propositional Prover," *Proc. of International Conference on Automated Deduction'97*, pp. 272-275, Jan. 1997.

[7] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *Journal of the Association for Computing Machinery*, 7, pp. 201-215, 1960.

[8] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *IEEE 1989 International Symposium on Circuits and Systems Proceedings*, pp. 1929-1934, May 1989.

[9] D. S. Johnson, and M. A. Trick, editors, "The second DIMACS implementation challenge," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1993. (see `http://dimacs.rutgers.edu/Challenges/`)

[10] L. G. Silva, J. P. M. Silva, L. M. Silveira, and K. A. Sakallah, "Timing Analysis Using Propositional Satisfiability," *IEEE International Conference on Electronics, Circuits and Systems*, Sep. 1998.

[11] O. Coudert, C. Berthet, and J. C Madre, "Verification of Sequential Machines Based on symbolic Execution," *Automatic Verification Methods for Finite State Systems(LNCS 407)*, pp. 365-373, Sep. 1989.

[12] R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293-318, Sep. 1992.

[13] D. Plaisted and S. Greenbaum, "A Structure-Preserving Clause Form Translation," *Journal of Symbolic Computation 2*, pp. 293-304, 1986.

[14] P. C. McGeer, A. Saldanha, R. K. Brayton and A. Sangiovanni-Vincentelli, "Delay Models and Exact Timing Analysis," *Logic Synthesis and Optimization*, Kluwer Academic Publishers, pp. 167-189, 1993, ed. T. Sasao.

[15] J. Silva and K. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Transactions on Computers*, pp. 506-521, May, 1999.

[16] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide," *Technical Report 1991-IWLS-UG-Saeyang*, Jan. 1991.