Waiting False Path Analysis of Sequential Logic Circuits for Performance Optimization

Kazuhiro Nakamura, Kazuyoshi Takagi, Shinji Kimura and Katsumasa Watanabe
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, JAPAN
{kazuhi-n, ktakagi, kimura, watanabe}@is.aist-nara.ac.jp

Abstract

This paper introduces a new class of false path, which is sensitizable but does not affect the decision of the clock period. We call such false paths waiting false paths, which correspond to multi-cycle operations controlled by wait states. The allowable delay time of waiting false paths is greater than the clock period. When the number of allowable clock cycles for each path is determined, the delay of the path can be the product of the clock period and the allowable cycles. This paper presents a method to analyze allowable cycles and to detect waiting false paths based on symbolic traversal of FSM. We have applied our method to 30 ISCAS89 FSM benchmarks and found that 22 circuits include such paths. 11 circuits among them include such paths which are critical paths, where the delay is measured as the number of gates on the path. Informations on such paths can be used in the logic synthesis to reduce the number of gates and in the layout synthesis to reduce the size of gates.

1 Introduction

The clock frequency of a sequential logic circuit is decided based on the maximum delay of the combinational parts of the circuit. The precise estimation of the maximum delay is important in deciding the proper clock frequency. The maximum delay can be computed as the longest path of the weighted graph corresponding to the circuit, where nodes in the graph are logic gates in the circuit and the weight of the node is the delay time of each gate. In some case, such topological maximum delay paths cannot be sensitized with any input patterns and therefore become false paths.

Many works have been done to detect false paths and to maximize the clock frequency [1, 2, 3]. False paths are classified into combinational false paths and sequential false paths. Combinational false paths are the paths where we have no input patterns to sensitize these paths. Sequential false paths are the paths which can be sensitized only by unreachable states of the circuit. Note that these combinational and sequential false paths are non-sensitizable paths.

On the other hand, in the circuit designs, there exist paths which are sensitizable but do not affect the clock period. These paths are multi-cycle paths which have several clock cycles to propagate signals. We regard such paths as new class of false paths and call *waiting false paths* where the propagation of signals is waited on k (≥ 2) clock cycles, hence the delay time of the path can be greater than the clock period.

A timing verification technique which can handle multi-cycle operations in micro-processors has been developed [8]. The method generates STG(state transition graph) of the controller of microprocessor and checks the specified timing constraints by manipulating the relation of events represented by inequalities on delays of modules. However, the method aims at microprocessor-based designs, and hard to apply to gate level circuits because STGs of gate level circuits are too large to be represented.

In this paper, we handle gate level circuits using boolean function manipulations. We discuss the properties of waiting false paths and propose a detection method of waiting false paths in gate level circuits. The method computes allowable clock cycles of each path between registers and detects waiting false paths. There are two steps in computing allowable clock cycles of each path: update cycle analysis of registers and timing analysis of each path between registers, both of them are executed based on a symbolic state traversal of FSM using BDD's(Binary Decision Diagrams) [4, 5].

This paper is organized as follows. In the next section, we show preliminaries. In Section 3, we define waiting false paths. In Section 4, we show a method of analyzing update cycles of registers. In Section 5, we show a method to find allowable clock cycles between registers and a timing verification using the allowable clock cycles. In Section 6, we show the experimental results.

2 Preliminaries

In this section, we show a definition of a finite state machine (FSM) based on [7].

Definition 2.1 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 $a_1 a_2 \ldots$ $\ldots a_n$ $(a_i \in \Sigma)$ 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 state and the output satisfies $q_i = \delta(q_{i-1}, a_i),\ o_i = \lambda(q_{i-1}, a_i)$.

Let Σ^* be a 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 and λ are extended, and $\delta^* : S \times \Sigma^* \to S$ and $\lambda^* : S \times \Sigma^* \to \Gamma^*$ are introduced.

Definition 2.2 $\delta^* : S \times \Sigma^* \to S$ is defined as follows:
- $\delta^*(q, \varepsilon) = q$
- $\delta^*(q, xa) = \delta(\delta^*(q, x), a),\ (a \in \Sigma, x \in \Sigma^*)$

Definition 2.3 $\lambda^* : S \times \Sigma^* \to \Gamma^*$ is defined as follows:
- $\lambda^*(q, \varepsilon) = \varepsilon$
- $\lambda^*(q, xa) = \lambda^*(q, x) \cdot \lambda(\delta^*(q, x), a),\ (a \in \Sigma, x \in \Sigma^*)$

Note that real sequential logic circuits consist of registers and combinational parts, and the registers are modeled as a state in the FSM model. In other words, a state of the FSM model corresponds to the tuple of these values of all registers at each time, and λ corresponds to the tuple of values. In the following, we focus on the

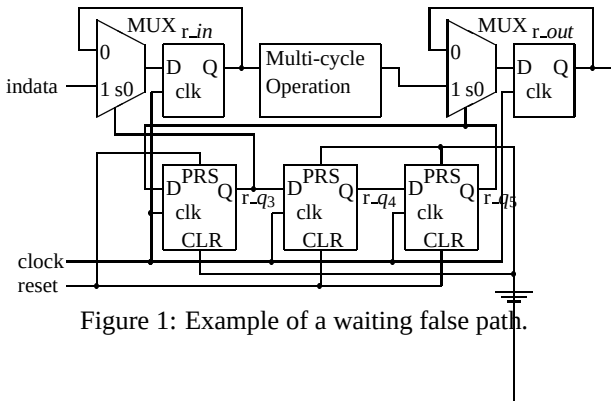

Figure 1: Example of a waiting false path.

values of specific registers, and use λ_r to denote the output function representing the value of a register r. It should be also noted that we deal only with edge-triggered flip-flops, but not with transparent latches.

3 Waiting False Paths

Waiting false paths are the paths in combinational parts which are sensitizable but do not affect the decision of the clock period. Fig.1 shows an example of waiting false 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 these registers changes as $(0,1,0)$, $(0,0,1)$, $(1,0,0)$, $(0,1,0)$, ... 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-cycle Operation" when $\text{r\_}q_5 = 1$. Since there is a wait state $(0,1,0)$, $\text{r\_}q_5$ is set to 1 two clocks after $\text{r\_}q_3$ is set to 1. Thus, there are 2 clock cycles for the computation of "Multi-cycle 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, the timing constraint of a path is

$$(\text{delay of the path}) \leq \{(\#\text{clocks}) \times (\text{clock period})\}$$

where #clocks denotes the clock cycles usable to propagate signals along the path. #clocks of each path are computed as follows:

1. Update cycle analysis of registers:
   we check whether the value of registers has been changed or not at each state, and compute the update cycle.
2. Timing analysis of each path between registers:
   we analyze the maximum allowable clock cycle of each path between registers using the update cycle of the input and the output registers.

In the following sections, we discuss update cycle analysis of registers and timing analysis of each path between registers.

4 Update Cycle Analysis of Registers

In the following, we focus on a 1-bit register(flip-flop), and introduce a set of states where the value of the register does not change during k clocks.

First, let RS be the set of reachable states from the initial state of FSM.

Definition 4.1 (Reachable states from the initial state)

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

```
ReachableStates(S0)
  S0:  the set of the initial states of FSM
  PS:  the set of present states
  NS:  the set of next states of PS
  RS:  the set of reachable states from S0
begin
  PS = S0; RS = S0;
  while (PS ≠ ∅) do
    NS = {δ(q,a) | q ∈ PS, a ∈ Σ};
    PS = NS − RS;
    RS = RS ∪ NS;
  end while;
  return RS;
end
```

Figure 2: Analysis of reachable states from the initial state.

Let S_0 be the set of initial states of FSM. From our FSM definition $S_0 = \{q_0\}$, RS is obtained with the procedure shown in Fig.2. In Fig.2, PS is the set of present states, and NS is the set of next states of PS. States are traversed until newly visited states become $\emptyset$.

The procedure is executed using a symbolic state traversal of finite state machines [4, 5]. In the symbolic state traversal, primary inputs and registers are represented as logical variables, and S_0, PS, NS, RS, δ are all described as logic functions or BDD's representing these functions. The manipulations of state sets such as $\cap, \cup, -$ are executed as logic operations such as AND, OR, and NOT.

Next, we define state sets $RS_k^r (\subseteq RS)$ where the value of register r does not change during k clock cycles. In other words, when starting from a state in RS_k^r, the value of the register does not change for all input sequence with length k. Formally, state sets RS_k^r is defined as follows:

Definition 4.2 (State sets where register r does not change during k clock cycles)

$$RS_k^r \triangleq \left\{ q \mid q \in RS, \forall x \in \Sigma^k, \lambda_r^*(q,x) \in \{0^k,\ 1^k\} \right\}$$

The following property is satisfied on RS_k^r.

Property 4.1 (Property on RS_k^r)

$$RS = RS_0^r = RS_1^r \supseteq RS_2^r \supseteq RS_3^r \supseteq RS_4^r \ldots$$

Note that, if there is a strongly connected component where the value of r is the same in any state, then $RS_k^r = RS_{k+1}^r$ with some k. Let K_r be the maximum number of k such that $RS_{k-1}^r \neq RS_k^r$.

Lemma 4.1 The following formula holds with $k > 2$.

$$RS_k^r = \left\{ q \mid q \in RS_{k-1}^r \wedge \forall a \in \Sigma :\ \delta(q,a) \in RS_{k-1}^r \right\}$$

Proof: ($\supseteq$) Let q be an element of the set of the right side. Since $q \in RS_{k-1}^r$, $\lambda_r^*(q,x) \in \{0^{k-1},\ 1^{k-1}\}$ for all $x \in \Sigma^{k-1}$. Since $q' = \delta(q,a) \in RS_{k-1}^r$, $\lambda_r^*(q',x) \in \{0^{k-1},\ 1^{k-1}\}$ for all $x \in \Sigma^{k-1}$. Thus, for any $w \in \Sigma^k$, there exist $a \in \Sigma$ and $x \in \Sigma^{k-1}$ s.t. $w = ax$ and $\lambda_r^*(q,w) \in \{0^k,\ 1^k\}$. They say that $q \in RS_k^r$.
($\subseteq$) Let q be an element of the set RS_k^r, then $q \in RS_{k-1}^r$. From the definition of RS_k^r, for any $ax \in \Sigma^k$, $\lambda_r^*(\delta(q,a),x) \in \{0^{k-1},\ 1^{k-1}\}$. Hence, for any $a \in \Sigma$, $\delta(q,a) \in RS_{k-1}^r$. ■

We show a procedure to compute the state sets RS_k^r based on Lemma 4.1.

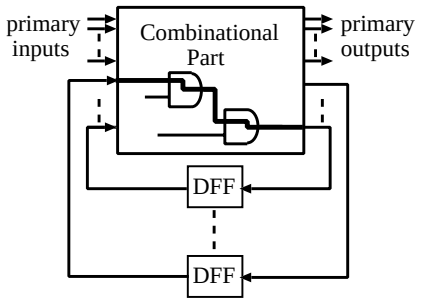

Figure 3: Paths between registers.

1. Compute the set *RS* of reachable states from the initial state of FSM, and then let RS_1^r be *RS*.
2. Using the state set *RS*, compute the state set RS_2^r, where $RS_2^r = \{q \mid q \in RS, \forall a_1a_2 \in \Sigma^2, \lambda_r(q,a_1) = \lambda_r(\delta(q,a_1),a_2)\}$.
3. Using RS_2^r, $RS_k^r (k \geq 3)$ is computed as follows. We also compute K_r.
   $k = 3$;
   `while(`$RS_{k-2}^r \neq RS_{k-1}^r$`) do`
   $RS_k^r = \{q | q \in RS_{k-1}^r \wedge \forall a \in \Sigma : \delta(q,a) \in RS_{k-1}^r\}$
   $k = k + 1$;
   `end while`
   $K_r = k - 2$;

First, state sets RS_1^r and RS_2^r are computed using symbolic state traversal in step 1 and step 2. Second, state set RS_3^r is computed from RS_2^r, and similarly, we can obtain state sets RS_4^r, RS_5^r, ... with Lemma 4.1 in step 3. Note that RS_k^r is computed until $RS_{k-2}^r = RS_{k-1}^r$, in some case $RS_{k-2}^r = RS_{k-1}^r = \emptyset$.

On the other hand, we define the set CS_r of states where the value of register *r* has just changed. Formally, state set CS_r is defined as follows:

Definition 4.3 (State set where register *r* has just changed)

$$CS_r \stackrel{\triangle}{=} \{q \mid \exists a_1a_2 \in \Sigma^2,\ q' \in RS,\ q = \delta(q',a_1),\ \lambda_r(q',a_1) \neq \lambda_r(q,a_2)\}$$

The CS_r can be computed similarly based on the symbolic state traversal.

5 Timing Analysis of Each Path Between Registers

5.1 Interval of Value Changes

In general, combinational parts of sequential logic circuits have many paths between registers as shown in Fig.3, and the maximum allowable clock cycle of each path may be different. We would like to know the clock cycles between the input register change and the output register change. That may vary on each state and the minimum cycle of allowable clocks decides the clock period. Formally, the following property is satisfied.

Property 5.1 (Interval of value change) Let *in* and *out* be registers, the interval of value change from *in* to *out* is *k*, when the following condition is satisfied:

$$CS_{in} \subseteq RS_k^{out}$$

5.2 Interval Analysis of Value Changes

Let *in* and *out* be register, and there be a path from *in* to *out*. From the Property 5.1, if $CS_{in} \subseteq RS_k^{out}$, then for $i < k$, $CS_{in} \subseteq RS_i^{out}$. Hence, the maximum allowable clock cycle of the path from *in* to *out* is the maximum number of *i* satisfying $CS_{in} \subseteq RS_i^{out}$. The maximum number of *i* is computed as the procedure shown in Fig.4, where *i* is incremented from 2 to K_{out} while $CS_{in} \not\subseteq RS_i^{out}$. If *i* is equal to K_{out}, then the allowable clock cycles of the path is infinite. Note that allowable clock cycles of a path becomes infinite when the destination register of the path never changes.

If the maximum allowable clock cycle of some paths is *k* ($\geq$ 2), then the path becomes false with respect to the clock, and the allowable delay time of the path is clock period multiplied by *k*.

```
Interval(in,out)
  CS_in:the state set where the value of in
changes
  RS_i^out:the state set where the value of out
       does not change during i clock cycles
  i:the interval of the value change between
in and out
begin
  if (K_out == 1) then
    return ∞;
  end if
  i = 2;
  while (CS_in ⊆ RS_i^out) do
    if (i == K_out) then
      return ∞;
    end if
    i = i + 1;
  end while
  return i - 1;
end
```

Figure 4: Analysis of the interval of value change between registers.

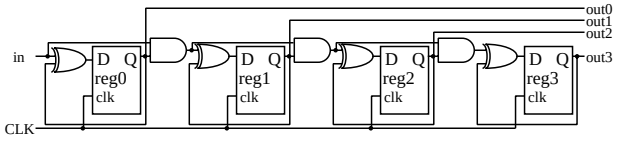

Figure 5: 4-bit synchronous up counter.

6 Experimental Results

We have implemented the ideas proposed in this paper in C language. We have applied the method to a 4-bit synchronous up counter, a prime number generator and the ISCAS89 benchmarks [6] on a Sun Ultra2 workstation (CPU UltraSPARC II 300MHz, Main memory 512MB).

6.1 Evaluation on a 4-bit Synchronous Up Counter

Fig.5 shows a 4-bit synchronous up counter which we have applied our method. The counter consists of 4 registers reg0, reg1, reg2 and reg3. The value of these registers changes as $(0,0,0,0)$, $(1,0,0,0)$, $(0,1,0,0)$, ... starting from $(0,0,0,0)$. The results of the update cycle detection are shown in Table 1. In the table, "Path" shows the path between registers, and "Clock cycles" shows the maximum allowable clock cycle of the path. For instance, the maximum al-

Table 1: Results for 4-bit synchronous up counter.

Path	Clock cycles	Path	Clock cycles
reg0 $\longrightarrow$ reg1	1	reg2 $\longrightarrow$ reg3	4
reg1 $\longrightarrow$ reg2	2	reg1 $\longrightarrow$ reg3	2
reg0 $\longrightarrow$ reg2	1	reg0 $\longrightarrow$ reg3	1

lowable clock cycle of the path from reg0 to reg1 is 1, and that of the path from reg1 to reg2 is 2 etc. The elapsed CPU time was 3.1 seconds on the Sun Ultra2.

6.2 Evaluation on a Prime Number Generator

A prime number generator is a circuit which computes all prime numbers (≤ 250) starting from 2. We have designed the prime number generator in VHDL. The VHDL design is non-pipeline and includes hardware loop structures. We obtained a gate level circuit from the VHDL design with the Synopsys Design Compiler, where the circuit consists of 35 registers and about 500 gates. We have applied our method to the circuit, and found 476 register pairs with allowable clock cycles greater than 1 in 772 register pairs. The detected register pairs were pairs of control registers of the loop and data registers controlled by wait states as shown in the Sect. 3. The number of repetitions in computing reachable states of the circuit was 11959. The elapsed CPU time was 3619.0 seconds on the Sun Ultra2.

We have checked critical paths in the prime number circuit, where the delay is measured as the number of gates on the path. We have found 16 critical paths (43 gates), and 14 of them have been detected by our method as paths whose allowable clock cycles are more than 2.

In Synopsys Design Compiler, multi-cycle paths can be handled with "set_multicycle_path" command. We have constrained the area of the circuit and multi-cycle paths with the result of the waiting false path analysis. We have optimized the prime number generator with Synopsys Design Compiler using lsi_10k library, and found that synthesis of the circuit with information of multi-cycle paths yields a slight smaller circuit: 5% in combinational area, 3% in total cell area.

These results show that many waiting false paths exist in sequential logic circuits, and synthesis of these circuits with the information yields a slightly smaller circuit.

6.3 Evaluation on ISCAS Benchmarks

We have applied our detection method to 30 ISCAS89 benchmarks [6] and found multi-cycle register pairs with allowable clock cycles greater than 1 on 22 circuits. Table2 shows the statistics. In the table, "#reg" is the number of all registers in the circuit, "#rep" is the number of repetitions in computing reachable states of the circuit, "#all_pairs" is the number of all register pairs which have paths between them, "#reg_pairs" is the number of register pairs whose maximum allowable clock cycle are greater than 1.

We have checked critical paths in 30 benchmarks. The critical paths whose allowable clock cycles are more than 2 have been found in 11 circuits. In Table2, † denotes that the circuit includes such critical paths.

7 Conclusions

In this paper, we have introduced a new kind of false path called waiting false paths and shown a method to detect the waiting false paths. Waiting false paths exist when input and output registers of the paths are guarded with wait states, and the delay time of the path can be greater than the clock period. We have proposed a method to analyze allowable clock cycles of each path and to detect waiting false paths based on symbolic traversal. We have applied the detection method to ISCAS89 benchmarks and found waiting false paths on 22 circuits in 30 finite state machine benchmarks.

Informations on waiting false paths can be use in the logic synthesis to reduce the number of gates and in the layout synthesis to reduce the size of gates.

Table 2: Result of waiting false path analysis of ISCAS benchmark circuits.

Circuit	#reg	#rep	#all_pairs /#reg_pairs	CPU Time
daio	4	5	6/0	3 [sec]
ex1†	20	10	380/357	3.0 [sec]
ex2†	19	6	342/306	3.0 [sec]
ex3†	10	5	90/80	3.0 [sec]
ex4†	14	14	169/135	3.0 [sec]
ex5†	9	4	72/48	3.0 [sec]
ex6†	9	1	61/61	3.0 [sec]
ex7	10	5	90/77	3.0 [sec]
s27	3	3	4/0	2.0 [sec]
s208†	8	17	28/18	3.0 [sec]
s298	14	19	56/4	4.0 [sec]
s344	15	7	74/1	79.0 [sec]
s349	15	7	74/1	50.0 [sec]
s382	21	151	131/13	150.0 [sec]
s386	6	8	30/4	3.0 [sec]
s420†	16	17	72/62	3.0 [sec]
s444	21	151	131/13	112.0 [sec]
s510	6	47	30/7	3.0 [sec]
s526	21	151	123/8	204.0 [sec]
s526n	21	151	123/8	225.0 [sec]
s641†	19	7	100/38	55.0 [sec]
s713†	19	7	100/38	55.0 [sec]
s820	5	11	20/0	3.0 [sec]
s832	5	11	20/0	3.0 [sec]
s838†	32	17	160/150	4.0 [sec]
s953	29	11	150/29	279.0 [sec]
s1196	18	3	20/0	802.0 [sec]
s1238	18	3	20/0	1074.0 [sec]
s1488	6	23	36/0	3.0 [sec]
s1494	6	23	36/0	3.0 [sec]

Acknowledgment

We would like to express our thanks to all members of Watanabe Laboratory at Nara Institute of Science and Technology for their discussions and comments. The Synopsys Design Compiler is licensed in the education program of VLSI Design and Education Center(VDEC), the University of Tokyo. We should thank to reviewers for their helpful comments.

References

[1] J. Benkowski, E.V. Meersch, L.J.M. Claesen, and H. DE MAN. "Timing Verification Using Statistically Sensitizing Path". In *IEEE Trans. on CAD, vol.CAD-9, no.10*, pp.1073-1084, Oct. 1990.

[2] H.C. Chen, and D.H. Du. "Path Sensitization in Critical Path Problem". In *Proc. of ICCAD-91*, pp.208-221, 1991.

[3] P. Ashar, S. Dey, and S. Malik. "Exploiting Multicycle False Paths in the Performance Optimization of Sequential Logic Circuits". In *IEEE Trans. on CAD, vol.14, no.9*, pp.1067-1075, Sep. 1995.

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

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

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

[7] J.E. Hopcroft and J.D. Ullman. "Introduction to Automata Theory, Languages and Computation". *Addison-Wesley*, 1979.

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