

HDL Modeling for Finding Critical Timing Paths

Alireza Khalafi and Zainalabedin Navabi

Electrical and Computer Engineering Department

Faculty of Engineering, Campus No. 2, University of Tehran

14399, Tehran IRAN navabi@ece.ut.ac.ir

Abstract

Critical timing path in a logical circuit is the longest sensitizable path from primary inputs to primary outputs. Finding sensitizable paths is achieved by eliminating false paths. This paper presents VHDL simulation models for finding false paths. The algorithm used and VHDL modeling strategy will be discussed.

1. Introduction

An important issue in the design and verification of integrated circuits is finding worst case delay values. Worst case delay of a circuit is the maximum time between events on the inputs of the circuit and the time that output reaches to its stable value. Finding this delay is necessary to ensure that a design meets a set of timing constraints. The maximum delay calculation is also necessary for determining the maximum clock speed. A common approach for determination of worst case delay is Path Enumeration [6]. This method finds the longest Path in the circuit and delay of this path is considered as the worst case delay of circuit. The main problem with this method is overestimating the worst case delay value. In some circuits, the path with the longest delay may never be activated and events cannot propagate to the output of the circuit through this path. Paths through which no signal can propagate are called false paths and the problem of identifying them is called the *general false path problem* [2-5, 7-10]. Identifying false paths can help a designer not to waste his or her time on correcting delays of paths which have no contribution to the worst case delay of a design.

This paper presents VHDL modeling methodology solving the general false path problem. The next section introduces some of existing methods for this purpose. Section 3 discusses the false path detection algorithm, and Section 4 presents VHDL implementation of this method.

2. General False Path Problem

False path detection methods can be categorized into static and dynamic methods. Static methods are similar to justification step of the D-algorithm. In these methods a path is sensitizable and allows signal values to propagate through it, provided that the stable values of gates in the selected path are non-control. Figure 1 shows a combinational circuit with three inputs and one output [8]. Unit gate delay values are assumed.

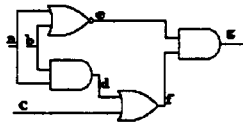


Figure 1. Example for critical path

The path enumeration method finds the worst case delay of this circuit to be 3. Considering path a-d-f-g of length 3, for events on "f" to propagate to "g", input "e" must be 1. On the other hand, for "a" to propagate to "d", line "b" must be 1 which implies "e" value of 0. Because of this inconsistency, the "a" to "f" path is a false path. Similarly, we can also show that the path b-d-f-g is a false path. Since these are the only paths which have a delay length of 3, we conclude that the delay of this circuit is less than 3. This result is reached by applying the static false path detection method. However, if we set "c" to 0 and change "a" and "b" from 1 to 0 at the same time, the events shown in Table 1 will occur. By referring to this table it is clear that the delay of this circuit is indeed 3, which is in contrast with the answer of the static method. The static method often underestimates worst case delay values in a circuit.

The main problem with static methods is that they check for false paths assuming that all signals in the circuit have reached their stable

values. However, the correct condition for a signal to propagate through a gate is that signals at the other inputs of the gate do not block its propagation only when it is making a transition. Therefore, it is not necessary that all other inputs have non-control stable values, even a small glitch having a non-control value can let signals at other gate inputs propagate forward.

Time	a	b	c	d	e	f	g
-1	1	1	0	1	0	1	0
0	0	0	0	1	0	1	0
1	0	0	0	0	1	1	0
2	0	0	0	0	1	0	0
3	0	0	0	0	1	0	1

Table 1

To consider such cases, we have chosen dynamic critical path analysis. For such analysis, several methods have been developed, we have implemented a method which mainly concentrates on finding false paths. In the next section we will introduce a dynamic algorithm and present comparisons with static algorithms.

3. False Path Detection Algorithm

This section describes the algorithm we have used for critical path analysis which is based on false path detection. The method used here has been proposed by H. C. Du, et. al., in a paper in the proceedings of 26th DAC [10]. In looking for false paths, this algorithm considers the steady state value of a signal as well as the time that the signal reaches this stable value. This is in contrast to the static methods that only consider the steady state value of a signal. The working of this algorithm will be shown by applying it to the example of Figure 2.

- (a) For each signal s_i define Max-arrive-time (s_i) as the maximum time it takes this signal to become stable with respect to changes on each of the circuit primary inputs. This is formulated as:

$$\begin{aligned} \text{If } s \text{ is the primary input, Max-arrive-time}(s) &= 0. \\ \text{If } s \text{ is the output of gate } g \text{ and } s_1, s_2, \dots, s_k \text{ are the inputs of the gate,} \\ \text{Max-arrive-time}(s) &= \text{Max}_{1 \leq i \leq k} (\text{Max-arrive-time}(s_i)) + \text{gate_delay}(g) \end{aligned}$$

In Figure 2-a these values are shown for every signal in the circuit.

- (b) For each signal s_i define Min-arrive-time (s_i) as the minimal time for this signal to become stable with respect to changes on each of the circuit primary inputs. This definition is formulated as follows:

$$\begin{aligned} \text{If } s \text{ is the primary input, Min-arrive-time}(s) &= 0. \\ \text{If } s \text{ is the output of gate } g \text{ and } s_1, s_2, \dots, s_k \text{ are the inputs to the gate,} \\ \text{Min-arrive-time}(s) &= \text{Min}_{1 \leq i \leq k} (\text{Min-arrive-time}(s_i)) + \text{gate_delay} \end{aligned}$$

Figure 2-b shows these values.

- (c) Select a path to be examined defined by a sequence of signals and gates. For example, the path $(s_0, g_0, s_1, g_1, \dots, g_{k-1}, s_k)$ describes s_i signals passing through g_i gates.
- (d) For every signal s_i on this path define Path-delay of s_i as the total delay from the primary input leading to this signal along the selected path. In other words, all gate delays along the path are added to form the delay of s_i .

In Figure 2-c we have shown the selected path and Path-delay for each signal.

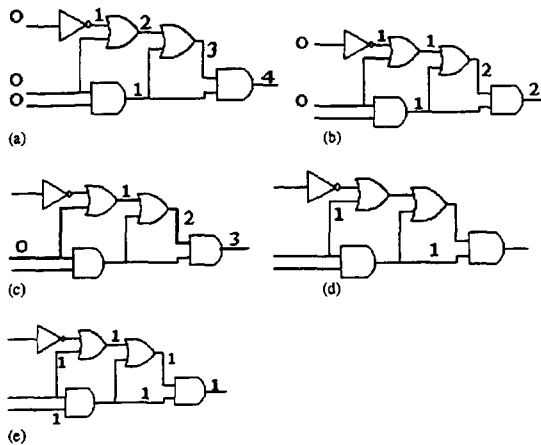


Figure 2. Steps of false path detection algorithm

(e) For each signal s_i along a selected path, if g_i is the gate which s_i is connected to, then for every input s_j ($i \neq j$) of g_i use the following rule to assign logic values to inputs of the gate g_i .

If Path-delay (s_i) > Max-arrive-time (s_j), set s_j to non-control value of g_i ,
 If Path-delay (s_i) < Min-arrive-time (s_j), set s_j to control value of g_i

Figure 2-d shows the values which have been assigned to signals according to the above rule for the path shown in Figure 2-c.

(f) For each gate propagate assigned values according to the following rules until there is no assignment in the circuit.

If (gate output) XOR (inversion of gate) is non-control value,
 set gate inputs to non-control value
 If one of gate inputs has a control value,
 set gate output to (control value) XOR (inversion of gate)
 If all gate inputs have non-control value,
 set gate output to (non-control value) XOR (inversion of gate)
 If only one input of the gate has not yet been assigned a value and
 (gate output) XOR (inversion of gate) is control value,
 set that input to control value

(g) If there is any inconsistency between values, then the path is false, otherwise it is sensitizable.

In Figure 2-e we have shown the final values for each signal and since there is no inconsistency we conclude that the selected path is not a false path. Rules described here have been implemented by modeling lines and gates in VHDL.

4. VHDL Implementation

We have used false path detection algorithm for worst case delay analysis and have implemented it in VHDL. As described in the previous section, the algorithm determines if a selected path is false independent of it being the longest path. Therefore, for finding the worst case delay of a circuit, the VHDL implementation examines all paths in the circuit by applying the false path detection algorithm to each and every path. When all paths have been processed, the one with the longest delay will be reported as the path having the worst case delay. This method is different from that reported in [11] which finds the K first longest paths.

VHDL types, utilities, and models for this implementation will be described here.

4.1 Signal Types and Utilities

Figure 3 shows types used in our VHDL models. The main type is "wcd". This type has nine enumeration elements. The 'X' element is for unknown values. Values '0' and '1' are the usual binary logic values. The "sta" value is used to start examination of a new path. The "rst"

enumeration element is used to reset the circuit for a new examination process. Finally, "nxp" and "nop" are used for selecting the next path to analyze. Type "which_input" is used by each gate to mark the input that is in a selected path.

The "wcd" type uses the resolution function "wiring" which is shown in Figure 4. Application of this resolution function will be discussed where VHDL models are described.

```

TYPE wcd IS ('X','1','0',sta,rst,cnf,esd,nxp,nop);
TYPE wcd_2d IS ARRAY(wcd,wcd) OF wcd;
TYPE wcd_vector IS ARRAY (NATURAL RANGE <=>) OF wcd;
TYPE which_input IS (inp1,inp2);
SUBTYPE wired_wcd IS wiring wcd;
  
```

Figure 3. Base Types

```

FUNCTION wiring(drivers:wcd_vector) RETURN wcd IS
VARIABLE temp:wcd:='X';
CONSTANT wiretable:wcd_2d:=
  (('X','1','0',sta,rst,cnf,esd,nxp,nop),
  ('1','1',cnf,sta,rst,cnf,esd,nxp,nop),
  ('0',cnf,'0',sta,rst,cnf,esd,nxp,nop),
  (sta,sta,sta,sta,rst,cnf,esd,nxp,nop),
  (rst,rst,rst,rst,rst,rst,esd,nxp,nop),
  (cnf,cnf,cnf,cnf,rst,cnf,cnf,cnf),
  (esd,esd,esd,esd,esd,cnf,esd,nxp,nop),
  (nxp,nxp,nxp,nxp,nxp,cnf,nxp,nop),
  (nop,nop,nop,nop,nop,cnf,nop,nop));
BEGIN
  FOR i IN drivers'RANGE LOOP
    temp:=wiretable(temp,drivers(i));
  END LOOP;
  RETURN temp;
END wiring;
  
```

Figure 4. Wiring Resolution Function

4.2 Gate Architecture

A gate model has inputs and outputs of type wired_wcd and INOUT mode. Each gate has a propagation delay, a control value and an inversion value which are passed to it via its generics. The entity part of a typical gate is shown in Figure 5.

```

ENTITY gate IS
  GENERIC(prop_delay:TIME;cont,inver:wcd);
  PORT(i1,i2,o: INOUT wired_wcd BUS);
END gate;
  
```

Figure 5. Entity part of a gate

There are three separate processes in the statement part of the gate architecture. These processes are for minimum and maximum arrive time calculation, path selection, and path examination. Figure 6 shows the general outline of a gate model structure. Min and Max arrival calculation process is initiated by placement of "esd" values on the primary inputs. Gates observing this initiation will calculate arrival time of signals on each of their inputs and record this information internal to the gate. When this is completed for all inputs of all gates, placement of the "nxp" value on the primary output will initiate the process of finding possible sensitized paths from primary outputs to the inputs. This is done by first selecting a path (as a result of placement of a "nxp") and then checking for its sensitizability. These two tasks are done by the second and third process statements of Figure 6 respectively. The placement of values on one hand and observing values on the other hand are done by VHDL models for the primary inputs and outputs. The following sections describe the three processes of gate models.

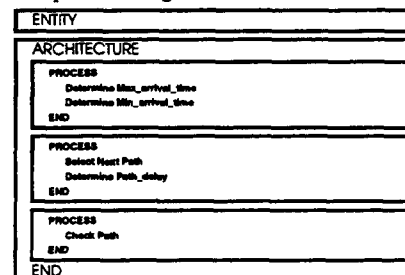


Figure 6. General gate structure

4.2.1 Max Min Arrival Time

The first process of Figure 6 is responsible for the determination of Max-arrive-time and Min-arrive-time of the gate inputs. Figure 7 shows the VHDL code for this process.

```

PROCESS
  VARIABLE cur_time : TIME;
BEGIN
  -- estimates MAX delay
  WHILE NOT(i1 = esd AND i2 = esd) LOOP
    WAIT ON i1,i2;
    IF NOT i1'STABLE THEN maxi1 <= NOW; END IF;
    IF NOT i2'STABLE THEN maxi2 <= NOW; END IF;
  END LOOP;
  o <= esd AFTER prop_delay;
  WAIT UNTIL i1 /= esd AND i2 /= esd;
  o <= NULL;
  ...
  -- estimates MIN delay
  WHILE NOT(i1 = esd AND i2 = esd) LOOP
    WAIT ON i1,i2;
    IF NOT i1'STABLE THEN mini1 <= NOW - cur_time; END IF;
    IF NOT i2'STABLE THEN mini2 <= NOW - cur_time; END IF;
    o <= esd AFTER prop_delay;
  END LOOP;
  WAIT UNTIL i1 /= esd AND i2 /= esd;
  o <= NULL;
  WAIT;
END PROCESS;

```

Figure 7. Determining Max & Min arrival times

This process is called only once for a given gate and has two parts for calculating Min and Max -arrive times. Initially the "esd" value is assigned to all primary inputs to initiate the process of estimating delay values. After a reset, "esd" values are again assigned to the primary inputs for the second time. The first time "esd" values reach gate inputs, this process calculates Max-arrive-time, and the second time these values arrive at the gate inputs, the gate calculates its Min-arrive time.

The first part of the process statement determines the Max-arrive-time for each input. A multi-input gate in a circuit waits until "esd" values are propagated to its inputs. The arrival time of the "esd" on each of the inputs is recorded as the Max-arrive-time for that input. After receiving "esd" on all inputs, the gate waits for the amount of time equal to its gate delay value and then assigns "esd" to its output.

The second part of the process determines Min-arrive-time for each input. This part is activated when "esd" values appear on the gate inputs for the second time. As in the first part, a gate in the circuit waits for the "esd" value to appear on its inputs and records the time of arrival as the Min-arrive-time for that input. After the appearance of the first "esd" on any of the gate inputs, the gate waits for an amount of time equal to its gate delay, and then assigns an "esd" value to its output.

4.2.2 Path Selection

The main task of the second process of Figure 6 is selecting a path to check for sensitizability. Figure 8 shows the VHDL details of this process.

The method which we have used for searching all the paths in a circuit is based on the depth first search algorithm. In the VHDL implementation of this method we first assign "npx" to the primary output of the circuit. Each gate that sees "npx" on its output propagates it to its first input. This process continues until "npx" reaches a primary input component. The primary input VHDL model checks to see if it has received "npx" in the last search. If it has not received "npx" it means that this is a new path and the primary input port responds with assigning 'X' to the gate input. When this transaction is seen by a gate connected to this primary input, the gate knows that the path has been found. At this time the gate waits for its delay value and then assigns 'X' to its output. Other gates do the same until 'X' reaches a primary output of the circuit. At this point an input to output path has been identified.

As mentioned, each gate waits for its delay value before propagating 'X' to its output. This is done so that the time when a gate receives 'X' on its input is the same as the delay between primary inputs and the input of this gate. When this signal propagates to a primary output, the accumulated delay on the signal is the same as the Path_delay value. Therefore, we are selecting a path as well as finding its delay value.

The above discussion presented the case that a "npx" has not been seen by a primary input component during the previous processing. If an "npx" has occurred on this input, the primary input component will assign "nop" to its output (gate input). A gate receiving this value will know

that there are no more paths through its specified input. Such a gate will set another one of its inputs to "npx" and waits for the response. If the response is 'X', the path has been found and gate sets its output to 'X' to propagate path selection verification to the primary output. On the other hand, if this gate receives "nop", it sets its output to "nop" to indicate to the next gate that there are no other paths through any of its inputs. When a "nop" is seen on a primary output, it indicates that all possible paths to the output have been examined for sensitizability.

```

Process selects next path to analyze.
PROCESS
  VARIABLE cur_time : TIME;
BEGIN
  WAIT UNTIL o = npx;
  cur_time := NOW;
  IF on_path_input = inp1 THEN
    i1 <= npx;
    WAIT ON i1 TRANSACTION; WAIT ON i1 TRANSACTION;
    i1 <= NULL;
    WAIT ON i1 TRANSACTION;
    IF i1 = 'X' THEN
      path_delay <= NOW - cur_time;
      is_on_path <= TRUE;
      WAIT FOR prop_delay;
      o <= 'X';
      WAIT ON o TRANSACTION; WAIT ON o TRANSACTION;
    END IF;
  END IF;
  IF on_path_input = inp2 OR i1 = nop THEN
    i2 <= 'X';
    WAIT ON i2 TRANSACTION;
    i2 <= npx;
    WAIT ON i2 TRANSACTION; WAIT ON i2 TRANSACTION;
    i2 <= NULL;
    WAIT ON i2 TRANSACTION;
    IF i2 = nop THEN
      on_path_input <= inp1; is_on_path <= FALSE; o <= nop;
    ELSE
      path_delay <= NOW - cur_time;
      on_path_input <= inp2; is_on_path <= TRUE;
      WAIT FOR prop_delay;
      o <= 'X';
    END IF;
    WAIT ON o TRANSACTION; WAIT ON o TRANSACTION;
    o <= 'X';
  END IF;
END PROCESS;

```

Figure 8. Process for Selecting paths

4.2.3 Checking Path

The third process in our gate architecture is responsible for assigning values to signals and propagating them through gates. For this consideration, there are two groups of gates in a circuit. The first group are the gates on the path selected by the path selection process. These gates use the rules in part (e) of the algorithm to assign values to their inputs. After assigning values to their inputs, these gates propagate values by using the rules in part (f) of the algorithm. The second group of gates are those that are not on the selected path. These gates only propagate values on their inputs or outputs according to the rules in part (f). Figure 9 shows the VHDL code of this process.

The process of Figure 9 is initiated by a "sta" value that is propagated to it from primary inputs. When initiated, this gate process causes gates to evaluate their output values in delta times and after waiting for a short time, the results will be assigned to their outputs. An inconsistency in circuit occurs if a 1 and a 0 appear on a signal at the same time. In this case, a node resolution function returns "cnf" which will propagate to the primary output. Therefore, if the primary output is "cnf" then the path is false, otherwise it is sensitizable.

Each time after selecting a path and checking for sensitizability of the selected path, a new path must be selected. To select the next path, the circuit must be reset to its initial state. This is done by assigning "rst" value to primary inputs which causes all gates in the circuit to remove all the drivers from their inputs and output and be prepared for examining the next path.

4.3 I/O Ports

Our models are activated by values assigned to primary inputs and outputs. The generated values at the primary ports are then processed by the corresponding components. Therefore, we need an entity for handling primary inputs and outputs. The architecture body for this entity is shown in Figure 10. This architecture is called *behavioral of io_ports* and consists of a generate part and a process statement. The code in the

generate statement is responsible for selecting a path. In this part, each primary input that receives "nxp" value for the first time, responds with 'X'. When this input receives the "nxp" value for the second time, it responds with "nop". This informs the following gate that it must search the new path through its other inputs.

The other part of the *io_ports* is a process which is a collection of sequential statements that assign values to primary inputs and outputs and control the program flow. It also writes the result to an external file. The first two tasks in this process force gates to determine Max-arrive-time and Min-arrive-times of their inputs. The next statement is a loop statement that selects and analyses each path in the circuit. After examination of all paths, the delay of the longest sensitizable path is taken as the worst case delay of the circuit.

```

-- justify and propagate values
PROCESS
BEGIN
-- wait until receive start signal
WAIT UNTIL i1 = sta AND i2 = sta;
o <= sta;
WAIT UNTIL i1 /= sta AND i2 /= sta;
o <= NULL;
-- if is on path, assign values to inputs
IF on_path_input = inp1
AND is_on_path = TRUE THEN
IF max_i2 < path_delay THEN i2 <= NOT cont;
ELSIF min_i2 > path_delay THEN i1 <= cont; END IF;
ELSIF on_path_input = inp2
AND is_on_path = TRUE THEN
IF max_i1 < path_delay THEN i1 <= NOT cont;
ELSIF min_i1 > path_delay THEN i2 <= cont; END IF;
END IF;
-- propagate input and output values
WHILE i1 /= rst AND i2 /= rst LOOP
IF i1 = cnf OR i2 = cnf THEN o <= cnf;
ELSIF (o XOR inver) = NOT cont THEN i1 <= NOT cont; i2 <= NOT cont;
ELSIF i1 = cont OR i2 = cont THEN o <= cont XOR inver;
ELSIF i1 = NOT cont THEN o <= i2 XOR inver;
IF (o XOR inver) = cont THEN i2 <= cont; END IF;
ELSIF i2 = NOT cont THEN o <= i1 XOR inver;
IF (o XOR inver) = cont THEN i1 <= cont; END IF;
END IF;
WAIT ON i1, i2, o;
END LOOP;
-- reset circuit
i1 <= NULL; i2 <= NULL; o <= rst;
WAIT UNTIL i1 /= rst AND i2 /= rst;
o <= NULL;
END PROCESS;

```

Figure 9. Process for Assigning and Propagating Values

```

ENTITY io_ports IS
PORT(inputs :INOUT wired_wcd_vector; output :INOUT wired_wcd);
END io_ports;

ARCHITECTURE behavioral OF io_ports IS
SIGNAL search :BOOLEAN :=FALSE;
BEGIN
-- this part selects next path to examine --
g1:FOR i IN inputs'RANGE GENERATE
p1:PROCESS
BEGIN
WAIT UNTIL inputs(i) = nxp; input(i) reply with 'X' (new path)
WAIT UNTIL inputs(i) = nxp; input(i) reply with "nop" (no path)
END PROCESS p1;
END GENERATE;

PROCESS
VARIABLE path_delay : TIME:=0 NS;
BEGIN
-- Estimates Min-arrive-time delays
-- Main Loop (check every path in circuit) --
WHILE TRUE LOOP
Select next path to analyze
Get selected path's delay;
EXIT WHEN no other path;
IF path_delay > worst_case_delay THEN
Start path analysis
IF path is not false THEN
THEN worst_case_delay := path_delay; END IF;
Reset circuit;
END IF;
END LOOP;
write worst_case_delay to a file
WAIT;
END PROCESS;
END behavioral;

```

Figure 10. Pseudo code of IO_Ports Entity

4.4 Primitive Gates

Primitive gates are easily defined by selecting appropriate control and inversion values of a logical gate. Models for basic primitive gates and

inverter have been developed. The architecture for a NAND gate is shown in Figure 11.

The discussion thus far has focused on two-input gates. Gate models with more inputs can easily be developed by cascading basic two-input gates. In this case, the gate delay must be handled by the last primitive gate, and all other primitive gates must have a delay of zero.

```

ENTITY nand2 IS
GENERIC(prop_delay : TIME); PORT(i1,i2,o: INOUT wired_wcd);
END nand2;

ARCHITECTURE structural OF nand2 IS
COMPONENT p_gate
GENERIC(prop_delay:TIME; cont,inver:wcd);
PORT(i1,i2,o: INOUT wired_wcd BUS);
END COMPONENT;
FOR all: p_gate USE ENTITY WORK.gate(behavioral);
BEGIN
p1:p_gate GENERIC MAP(prop_delay,'0','1') PORT MAP (i1,i2,o);
END structural;

```

Figure 11. A NAND gate

4.5 Test Bench

The test bench for activating the models and performing the false path identification on a circuit is a simply an instantiation list of the gates of the circuit. A partial listing is depicted in Figure 12.

```

p1:ports PORT MAP (input,output);
g1:nand GENERIC MAP (1 NS) PORT MAP (input(0),input(1),h3);
g2:nand GENERIC MAP (1 NS) PORT MAP (input(0),h3,h1);
g3:nand GENERIC MAP (1 NS) PORT MAP (h3,input(1),h2);
g4:nand GENERIC MAP (1 NS) PORT MAP (h1,h2,output);

```

Figure 12. Instantiation List

CONCLUSIONS

This paper presented VHDL models for solving general false path problem. Our models are based on dynamic false path detection algorithms and therefore give better results than static approaches. These models examine all paths in the circuit and find the longest non false path and therefore will be useful for critical path analysis and other timing applications. Application of these models is in formation of an integrated HDL based design and simulation environment.

REFERENCES

1. Z. Navabi, "VHDL: Analysis and Modeling of Digital Systems," McGraw-Hill, New York, 1993.
2. J.Benkoski, E.V.Meersch, H.D.Man, "Efficient Algorithms for solving the false path problem in timing verification", ICCAD 1987.
3. D.Brand, V.Iyengar, "Timing analysis using functional analysis", IEEE transactions on Computer, Oct 1988
4. D.Brand, V.S.Iyengar, "Timing analysis using functional relationship", IEEE transactions on Computer, Oct 1988
5. H. C. Du, L. R. Liu, H. C. Chen, "Critical path selection for performance optimization", IEEE transactions on Computer Aided Design, 1993.
6. R.Hitchcock, "Timing verification and timing analysis program", 19th ACM/IEEE Design Automation Conference, 1982.
7. W. Li, S. M. Reddy, S. K. Sahni, "On the path selection in combinational logic circuit", IEEE transactions on Computer Aided Design, 1989.
8. C. Mcgeer, R. Brayton, "Efficient Algorithms for computing the longest viable path in combinational network", 27th ACM/IEEE design Automation Conference, 1989.
9. S. Perremans, L. Claesen, H. Deman, "Static timing analysis of dynamically sensitizable paths", 26th ACM/IEEE Design Automation Conference, 1989.
10. Du, H. C., Yen, H. C., Ghanta, S., "On the General Fault Path Problem in Timing Analysis", Proceeding of the 26th Design Automation Conference, June, 1989.
11. H. C. Yens, S. Ghanata, H. C. Du, "Efficient algorithms for extracting the K most critical paths in timing analysis", 26th ACM/IEEE Design Automation Conference, 1989.