A New Approach to Solving False Path Problem in Timing Analysis

Shiang-Tang Huang Tai-Ming Parng

Dept. of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C

Jyuo-Min Shyu

Computer & Communication Research Labs.
Industrial Technology Research Institute
Hsinchu, Taiwan, R.O.C

Abstract

A new approach to solving the false path problem is proposed. The approach is based on an extended Boolean Algebra and is capable of modeling the logic and timing behavior of logic networks in terms of modified boolean functions. By applying algebraic manipulations, our approach can extract correct timing information such as path delays as well as the input vectors to activate the sensitizable paths. The approach has been implemented and tested on ISCAS benchmarks.

1 Introduction

A timing analyzer's major role is to extract critical paths of a logic network. Depending on the techniques [1, 2, 3] used, the reported paths may involve paths that are never activated by any input vector. These paths are usually called *false paths*, while the others are *sensitizable paths*. The presence of the *false paths* cause some loss of accuracy, i.e. an overestimated maximal delays. The false path problem, to extract exact maximal delays or to report the sensitizable paths, is one of the major topics in recent researches in timing analysis.

Since the rising delay and the falling delay of a logic element are usually different, to find the maximal delay of a logic network, the timing analyzer must know not only the sensitizable paths but also the logic states of all nodes on the paths. However, the existing researches [4, 5, 6, 8, 9] focus mainly on reporting sensitizable paths which have no logic state information and thus may derive longer delays. Consider the logic network shown in Fig. 1. There are four *physical paths*, namely **A-F-G**, **B-D-E-F-G**, **C-E-F-G**, and **C-G**. According to existing false path detection algorithms, the path **B-D-E-F-G** is *false* and the longest sensitizable path is **C-E-F-G**. Therefore, it will report 4 (rising delay of C2 + rising delay of C3 + rising delay of C4) as the maximal delay. However, by detailly considering the logic states of the nodes, there are eight *logical paths*, which involve information about the logic states of all nodes on the physical paths; five of them are sensitizable, namely A-F-G, $\overline{A}$-$\overline{F}$-$\overline{G}$, $\overline{C}$-$\overline{E}$-$\overline{F}$-$\overline{G}$, C-G, and $\overline{C}$-$\overline{G}$; three of them are false, namely $\overline{B}$-D-E-F-G, B-$\overline{D}$-$\overline{E}$-$\overline{F}$-$\overline{G}$, and C-E-F-G. The longest sensitizable path is the logical path $\overline{C}$-$\overline{E}$-$\overline{F}$-$\overline{G}$ and its path length (delay) is 3 (falling delay of C2 + falling delay of C3 + falling delay of C4). (Note that the path C-E-F-G is a false path and its path length is 4). From this example, we observed that for a physical path in a logic network, there may exist two or more *logical paths*, each of which may be either sensitizable or false. Thus, to get precise timing information, the circuit delay should be calculated based on logical paths and the timing analyzer should report the *longest sensitizable logical paths*.

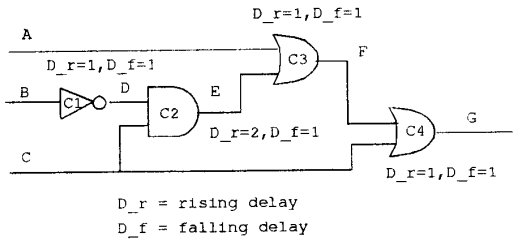

Figure 1: Path **C-E-F-G** is only sensitizable for **G=0**.

In this paper we present a new approach to solving false path problem in timing analysis. The approach is based on a formalism, called *Timed Boolean Algebra*, which is derived from the conventional Boolean Algebra by adding a *delay operator* for modeling the delay aspects of physical logic elements. Through the use of the *Timed Boolean Algebra*, the logic and timing behavior of each type of logic element and general logic networks can be modeled as boolean functions with delay operators. By performing algebraic manipulations and computations on the models, useful timing information can be extracted. The features of our approach are: (1) it reports circuits' delays based on logical paths, (2) the maximal rising and falling delays of each node are extracted separately, and (3) it derives the input vectors to activate the sensitizable paths, which are very useful in timing analysis.

In Section 2, a brief introduction to the *Timed Boolean Algebra* is given. In Section 3, the idea of

†This work was supported by the National Science Council R.O.C under Grant NSC 80-0404-E-002-26.

modeling the logic and timing behavior of commonly used logic elements are presented. Section 4 describes the modeling and reduction process of a logic network, while Section 5 delineates the method of deriving maximal delays from the results of section 4. Section 6 discusses the practical implementation considerations and Section 7 gives the experimental results. Finally, a conclusion is given in Section 8. Note that due to the limitation of space, all the proofs of the theorems have been omitted in this paper.

2 Timed boolean algebra

The *Timed Boolean Algebra* is extended from conventional *Boolean Algebra* with a *delay operator*. In this section, we give a brief introduction to the major elements of the Timed Boolean Algebra, including some basic definitions, operators, and theorems.

Definition: Any variables, expressions and functions which may involve the delay operator are referred to as *timed boolean variables*, *timed boolean expressions*, and *timed boolean functions* respectively.

Definition : Any boolean variable which represents a primary input node of a logic network is called a *primitive timed boolean variable.*

Definition : A *primitive timed boolean expression* is a expression which involves only primitive boolean variables, AND-operators, and OR-operators.

2.1 Operators and basic operation rules

The four operators and basic algebraic operation rules in the Timed Boolean Algebra are described in the following.

1. Delay operator '[,]' : The first operand is a timed boolean expression B, while the second one is a non-negative delay value d. $[B, d]$ represents the result of applying the delay operator on B with delay d. The formal definition of $[B, d]$ is shown below.

$$[B, d](t) = \begin{cases} B(t-d) & t \geq d \\ Unknown & t < d \end{cases}$$

   where $B(t)$ and $[B, d](t)$ are the logic values of B and $[B, d]$ at time t, respectively. The expression $[B, d]$ is called a *'delay term'*. and the delay value d is referred to as 'the delay of expression B'.
2. OR-operator '+' and AND-operator '·': These two operators perform *boolean-OR* operation and *boolean-OR* on the operands.
3. NOT-operator '!' : This operator performs *boolean-Not* operation on the operand.

The precedence order of these operators is as follows:

Delay > NOT > AND > OR

Many rules for Boolean Algebra still hold in the Timed Boolean Algebra. For instance, the operations of operators '+' and '·' still obey the commutative, associate, and distributive laws.

2.2 Important theorems and definitions

Only the three theorems that have been applied in our approach are presented.

Substitution theorem: Let A, B_1, B_2, $\cdots$, and B_n be boolean variables or timed boolean expressions.

1. If $A = [B, t_a]$ and $B = [B_1, t_1] + [B_2, t_2] + \cdots + [B_n, t_n]$, then $A = [B_1, t_a + t_1] + [B_2, t_a + t_2] + \cdots + [B_n, t_a + t_n]$
2. If $A = [B, t_a]$ and $B = [B_1, t_1] \cdot [B_2, t_2] \cdot \cdots \cdot [B_n, t_n]$, then $A = [B_1, t_a + t_1] \cdot [B_2, t_a + t_2] \cdot \cdots \cdot [B_n, t_a + t_n]$

Maximal delay theorem: Let B_1 and B_2 be two primitive timed boolean expressions, then

$$[B_1, t_1] \cdot [B_2, t_2] = [B_1 \cdot B_2, \mathrm{Max}(t_1, t_2)]$$

Boolean propagation theorem: Let B_1 and B_2 be two primitive timed boolean variables or expressions. If $t_1 < t_2$, then

$$[B_1, t_1] + [B_2, t_2] = [B_1, t_1] + [(!B_1) \cdot B_2, t_2]$$

Finally, two important definitions are presented in the following.

Definition [primitive delay term] : A delay term is said to be a *primitive delay term (Pterm)*, if its first operand is a primitive variable or primitive boolean expression without delay operator.

Definition [sum-of-*Pterm* form] : A timed boolean expression is said to be a sum-of-*Pterm* expression if the expression is composed of only *Pterms* and OR-operators. The general form of a sum-of-*Pterm* expression is as follow:

$$[P_1, t_1] + [P_2, t_2] + \cdots + [P_n, t_n] \quad \text{or} \quad \sum_{i=1}^{n} [P_i, t_i]$$

where all $[P_i, t_i]$ are *Pterms*. A timed boolean function is a sum-of-*Pterm* function if its right hand side is a sum-of-*Pterm* expression.

3 Modeling logic elements

The logic and timing behavior of each element in a logic network is represented in terms of a pair of timed boolean functions; one models the rising behavior of the output node of the logic element; while the other models the falling behavior. In the following subsections, the modeling concept and the models for different logic element types are presented.

3.1 Modeling concept

For each physical node N in a logic network, there are two *logical nodes*, denoted as N and $\overline{N}$. N and $\overline{N}$ model the behavior of the node N in logic state 1 and 0 respectively. Let $N1$ model the physical node N1 in logic state $v1$ and $N2$ model the node N2 in logic state $v2$. If the change of $N1$ from $\overline{v1}$ to $v1$ may cause $N2$ to change from $\overline{v2}$ to $v2$, then there exists a *logical path* from $N1$ to $N2$, denoted as $N1$-$N2$. We use $\mathbf{d}_{N1,N2}$ to denote the path delay of $N1$-$N2$.

For any single output logic element with fixed signal flow direction, the logic relation between the logical nodes can be modeled by AND-operators and OR-operators, while the delay relations can be modeled by delay-operators. Thus, any logic element can be modeled in terms of two timed boolean functions: one function models the physical output node in logic state 1; the other models the physical node in logic state 0.

3.2 Modeling of simple gates

Let $\mathbf{A_i}$ be the ith input node of an n-input gate, **B** be the output node. The functions of an inverter and an N-input NAND gate are as follows:

1. **Inverter:**
   $B = [\overline{A_1}, \mathbf{d}_{\overline{A_1},B}]$
   $\overline{B} = [A_1, \mathbf{d}_{A_1,\overline{B}}]$
2. **N-input NAND gate:**
   $B = [\overline{A_1}, \mathbf{d}_{\overline{A_1},B}] + \cdots + [\overline{A_n}, \mathbf{d}_{\overline{A_n},B}]$
   $\overline{B} = [A_1, \mathbf{d}_{A_1,\overline{B}}] \cdot [A_2, \mathbf{d}_{A_2,\overline{B}}] \cdot \cdots \cdot [A_n, \mathbf{d}_{A_n,\overline{B}}]$

The models for other types of logic gates can be derived in a similar way.

3.3 Modeling of pass transistors

The pass transistors with fixed signal flow direction can also be modeled in terms of two functions. Assume that the propagation delays through the channels are given and the signal flows from the source terminal **A** to the drain terminal **C**. Let **B** be the gate terminal, and $\mathbf{d}_{on}$ be the delay time to turn on the transistor. The functions of the N-type pass transistors are as follows:

$C = [A, \mathbf{d}_{A,C}] \cdot [B, \mathbf{d}_{on} + \mathbf{d}_{A,C}]$
$\overline{C} = [\overline{A}, \mathbf{d}_{\overline{A},\overline{C}}] \cdot [B, \mathbf{d}_{on} + \mathbf{d}_{\overline{A},\overline{C}}]$

The models of P-type transistor can be derived similarly.

3.4 Modeling of output-wiring

The pass transistors' outputs may be wired together (*output-wiring*) and the output-wiring is modeled as a pseudo logic element. Let $\mathbf{A_i}$ be the ith input node and **B** be the output node, the timed boolean functions are:

$B = [A_1, 0] + [A_2, 0] + \cdots + [A_n, 0]$
$\overline{B} = [\overline{A_1}, 0] + [\overline{A_2}, 2] + \cdots + [\overline{A_n}, 0]$

4 Modeling logic networks

By using the models for logic elements, a N-output logic network can be represented as a set of boolean functions. This set of boolean functions can be further combined and reduced into a set of sum-of-*Pterm* functions by eliminating all non-primary input variables. After this process, the logic network is modeled in terms of 2N sum-of-*Pterm* functions.

First, the example in Fig. 1 is used to demonstrate the procedure of modeling of a logic network.

1. By the models for the logic elements, we obtain
   $D = [\overline{B}, 1]$, $\overline{D} = [B, 1]$
   $E = [D, 2] \cdot [C, 2]$, $\overline{E} = [\overline{C}, 1] + [\overline{D}, 1]$
   $F = [A, 1] + [E, 1]$, $\overline{F} = [\overline{A}, 1] \cdot [\overline{E}, 1]$
   $G = [F, 1] + [C, 1]$, $\overline{G} = [\overline{F}, 1] \cdot [\overline{C}, 1]$
2. Reduce $E = [D, 2] \cdot [C, 2]$ and $\overline{E} = [\overline{C}, 1] + [\overline{D}, 1]$.
   $E = [\overline{B}, 3] \cdot [C, 2]$, $\overline{E} = [B, 2] + [\overline{C}, 1]$
   $= [\overline{B} \cdot C, 3]$
3. Reduce $F = [A, 1] + [E, 1]$ and $\overline{F} = [\overline{A}, 1] \cdot [\overline{E}, 1]$.
   $F = [A, 1] + [\overline{B} \cdot C, 4]$
   $\overline{F} = [\overline{A}, 1] \cdot ([B, 3] + [\overline{C}, 2])$
   $= [\overline{A}, 1] \cdot [B, 3] + [\overline{A}, 1] \cdot [\overline{C}, 2]$
   $= [\overline{A} \cdot B, 3] + [\overline{A} \cdot \overline{C}, 2]$
4. Reduce $G = [F, 1] + [C, 1]$ and $\overline{G} = [\overline{F}, 1] \cdot [\overline{C}, 1]$.
   $G = [C, 1] + [A, 2] + [\overline{B} \cdot C, 5]$
   $\overline{G} = [\overline{C}, 1] \cdot ([\overline{A} \cdot B, 4] + [\overline{A} \cdot \overline{C}, 3])$
   $= [\overline{C}, 1] \cdot [\overline{A} \cdot B, 4] + [\overline{C}, 1] \cdot [\overline{A} \cdot \overline{C}, 3]$
   $= [\overline{A} \cdot B \cdot \overline{C}, 4] + [\overline{A} \cdot \overline{C}, 3]$

The formal procedure of modeling a logic network is simple and can be stated as follows:

1. For each function whose right hand side variables that are already modeled as sum-of-*Pterm* functions, we perform following reduction process:
   (a) Substitute the variables with their corresponding sum-of-*Pterm* expressions.
   (b) Apply the distribution laws to expand the resultant expressions.
   (c) Apply the maximal delay theorem to reduce the resultant expressions into sum-of-*Pterm* expressions.
2. Repeat step 1 until the functions for each output node are in the sum-of-*Pterm* form.

5 Extracting maximal delays

In this section, the method of extracting maximal delays from the sum-of-*Pterm* functions is presented.

Theorem: For each *Pterm* $[P_i, t_i]$ in the the sum-of-*Pterm* function $P = \sum_{i=1}^{n} [P_i, t_i]$, its ***sensitizability*** is

$$\mathbf{SENS}([P_i, t_i]) = P_i \cdot \overline{(\sum_{\forall P_j,\, t_j < t_i} P_j)} = P_i \cdot \prod_{\forall P_j,\, t_j < t_i} \overline{P_j}$$

If the sensitizability is not equal to 0, $[P_i, t_i]$ is a *sensitizable term*. Otherwise, $[P_i, t_i]$ is a *false term*.

Definition [possible delay set] : The *possible delay set* of a sum-of-*Pterm* function $P = \sum_{i=1}^{n} [P_i, t_i]$ is the union of delay value t_i of all sensitizable terms $[P_i, t_i]$. The maximal delay of P is the maximal value of its possible delay set.

Again, the logic network in Fig. 1 is used to demonstrate the delay extracting procedure. The two sum-of-*Pterm* functions are

$G = [C, 1] + [A, 2] + [\overline{B} \cdot C, 5]$
$\overline{G} = [\overline{A} \cdot B \cdot \overline{C}, 4] + [\overline{A} \cdot \overline{C}, 3]$

The sensitizabilities of all the *Pterms* are as follows:

$\mathbf{SENS}([C, 1]) = C$
$\mathbf{SENS}([A, 2]) = A \cdot \overline{(C)} = A \cdot \overline{C}$
$\mathbf{SENS}([\overline{B} \cdot C, 5]) = \overline{B} \cdot C \cdot \overline{(A + C)} = 0$
$\mathbf{SENS}([\overline{A} \cdot \overline{C}, 3]) = \overline{A} \cdot \overline{C}$
$\mathbf{SENS}([\overline{A} \cdot B \cdot \overline{C}, 4]) = (\overline{A} \cdot B \cdot \overline{C}) \cdot \overline{(\overline{A} \cdot \overline{C})} = 0$

So the maximal rising delay of node **G** is 2 and the input vector must make $A \cdot \overline{C}$ true. The maximal falling delay of node **G** is 3 and the input vector must make $\overline{A} \cdot \overline{C}$ true.

6 Implementation considerations

To apply the Timed Boolean Algebra in practical timing analysis applications, the problem of manipulating large timed boolean functions needs to be addressed. To reduce the number of in-memory *Pterms* and hence the memory required, three rules have been developed as follows:

Table 1: Experimental results

CKT	Improved Nodes	CPU (sec)	Memory (K bytes)	Sensitizable Terms	False Terms
c432	0	348.55	1425	60347	6842
c499	0	4.80	330	20952	384
c880	0	609.54	2924	159322	1462
c1355	160	427.56	1207	99994	2336
c1908	56	655.04	4375	194792	38391
c2670	134	1800.94	4633	196721	8560
c3540	259	969.67	15489	417809	22103
c5315	666	1374.93	15836	397711	13358
c6288	462	19729.89	14030	1488118	449324
c7552	957	1753.57	23631	443468	18905

1. **Remove false terms as early as possible.** Whenever the function of a logical node is derived, the false *Pterms* are removed immediately.
2. **Perform depth-first traversal.** As soon as the functions of some logical nodes are no longer needed, i.e. all the succeeding nodes have been processed, the memory used by the functions is released.
3. **Replace a complex timed function with a single *Pterm*.** Given a sum-of-*Pterm* function $P = \sum_{i=1}^{n}[P_i, t_i]$, where all $[P_1, t_i]$ are sensitizable terms, if n is larger than a given threshold (Cutting Threshold), it is replaced by $P = [P, \mathrm{Max}_{i=1}^{n}(t_i)]$.

Note that the third rules may cause some loss of accuracy, but it can guarantee the extracted delays of each logical nodes are the upper bounds.

7 Experimental results

A tool based on the procedures and rules presented in previous sections has been implemented in C on SUN4/60 workstations (SPARCstation1). The test cases used are the ISCAS[10] benchmarks and the cutting thresholds are assigned to be 2000. Without loss of generality, all the logic elements in these circuits are assumed to be of *unit delay*. The detailed experimental results, including the number of improved nodes (whose extracted delay is less than the critical delay), the CPU time, the memory size required, etc., are shown in Table 1. From this table, we see that (1) many nodes have improved delay values, (2) the memory size required is reasonable (less than 25M bytes), and (3) the run-time is acceptable. More detailed information about the number of improved delay units and the number of nodes with improvement are shown in Table 2.

Since the results obtained with different cutting thresholds are different, a better way to perform timing analysis on a logic network is to analyze several times by different thresholds and take the best result among these runs as the final result.

8 Conclusion

To solve the false path problem in timing analysis, we present a new methodology, which is based on a formalism called Timed Boolean Algebra. The operators of the Timed Boolean Algebra facilitate the modeling of the logic and timing behavior of logic networks, while the algebraic laws and theorems provide the theoretical basis for processing the models and extracting

Table 2: Improved delays and node number

Impro. delay	Number of nodes with improvement						
	c1355	c1908	c2670	c3540	c5315	c6288	c7552
1	96	23	112	178	494	164	645
2			10	57	144	43	243
3			3		19	107	38
4	64		3	10	6	104	12
5			1	10	2	36	12
6		20	2				2
7		9	1	3	1	1	
8						5	2
9			2	1		2	1
12		2					
13							2
14		2					

important timing information needed in timing analysis. There are two innovative ideas involved in our approach: (1) use algebraic method to deal with the problem of delay analysis, (2) the rising and falling delays of each node are extracted separately. A tool has been developed and tested on public ISCAS benchmarks. The examples and experimental results show that our approach is very practical and promising.

References

[1] R. B. Hitchcock, Sr., G. L. Smith, and D. D. Cheng, "Timing Analysis of Computer Hardware," *IBM. J. Res. Develop.*, vol. 26, no. 1, pp. 100-106, Jan. 1982.

[2] H.C. Yen, S. Ghanta, H.C. Du, "A Path Selection Algorithm for Timing Analysis," *Proc. of 25th Design Automation Conf.*, 1988.

[3] Steve H.C. Yan, David H.C. Du, S. Ghanta, "Efficient Algorithms for Extracting the K Most Critical Paths in Timing Analysis," *Proc. of 26th Design Automation Conf.*, 1989.

[4] Robert B. Hitchcock, Sr., "Timing Verification and the Timing Analysis Program," *Proc. of 19th Design Automation Conf.*, pp. 594-604, 1982.

[5] J. Benkoski, E. Vanden Meersch, L. Claesen, and H. De Man "Efficient Algorithms for Solving the False Path Problem in Timing Verification," *IEEE Inter. Conf. on Computer-Aided Design (ICCAD'87)*, 1987.

[6] David H.C. Du, Steve H.C. Yen, and S. Ghanta "On the General False Path Problem in Timing Analysis," *Proc. of 26th Design Automation Conf.*, 1989.

[7] Daniel Brand, Vijay S. Iyengar, "Timing Analysis Using Functionsl Analysis," *IEEE Tran. on Computers*, VOL 37, NO. 10, Oct. 1988.

[8] Patrick C. McGeer, Robert K. Brayton, "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network," *Proc. of 26th Design Automation Conf.*, 1989.

[9] Patrick C. McGeer, Robert K. Brayton, "Timing Analysis in Precharge/Unate Networks," *Proc. of 27th Design Automation Conf.*, 1990.

[10] ISCAS-85 Benchmarks, *Special Session: Recent Algorithms for Gate Level ATPG with Fault Simulation and Their Performance Assessment*, IEEE International Symposium on Circuits and Systems, June 1985.