E7.

USING TIMED BOOLEAN ALGEBRA TO SOLVE FALSE PATH PROBLEM IN TIMING ANALYSIS

Shiang-Tang Huang, Tai-Ming Parng, Jyuo-Ming Shyu

Department of Elextrical Engineering
National Taiwan University

Abstract

A new approach to 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 path delays as well as the input vectors for activating the sensitizable paths.

Introduction

A timing analyzer's major role is to report the exact maximal circuit delay and to extract critical paths of a combinational logic network. Most researches focus on the extraction of the critical paths. Depending on the techniques [1, 2, 3, 4, 5, 6, 7, 8] 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 false path problem is one of the major topics in recent researches [9, 10, 11, 12, 13, 14] in timing analysis. Another approach to solving the false path problem is to calculate the exact circuit delay by *trimming the false paths*.

In this paper we present a new approach to the 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. By performing algebraic manipulations and computations on the models, useful timing information can be extracted. The information include (1) each node's delay, and (2) the input vectors activating the sensitizable paths.

Timed Boolean Algebra

To facilitate modeling a logic network with timing information, the conventional Boolean Algebra is extended with a *delay operator*. The extended Boolean Algebra is referred to as the *Timed Boolean Algebra* in this paper.

Definition: The variables, expressions, and functions in the Timed Boolean Algebra are referred to as *timed boolean variables*, *timed boolean functions*, and *timed boolean functions*.

Definition: The timed boolean variable in the Timed Boolean Algebra is a functions of input data and time. Let I be the input data and t be a value about time, the time boolean variable B is

$$B : \mathcal{F}(I,t) \longrightarrow [0,1]$$

If $t < 0$, the value of B is U (unknown).

Definition: Given an input data I, the boolean value of timed boolean variable (expression) B at time t is denoted as $B(I,t)$.

The definitions of the four operators are described in the following.

1. Delay operator '[,]' : The first operand is a timed boolean variable (expression) B and the second operand is a value d representing the delay. $[B,d]$ is the result of applying the delay operator on B with delay d. The formula of $[B,d]$ is

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

   where I is the input vector.

2. OR-operator '+' and AND-Operator '·': The OR-operator and AND-operator performs *boolean-OR* and *boolean-OR* operation on the operands for all time t respectively.

3. NOT-operator '!' : This operator performs *boolean-Not* operation on the operand.

The precedence order of these operators is as follows:

Delay operator > NOT > AND > OR

Many laws and rules for Boolean Algebra still hold in the Timed Boolean Algebra. Here, we show three important theorems related to delay operator.

Substitution Theorem: Let $t_a, t_1, t_2, \cdots, t_n$ be non-negative value. A, B_1, B_2, $\cdots$, and B_n be boolean variables or expressions.

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

$$A = [B_1, t_a + t_1] + [B_2, t_a + t_2] + \cdots + [B_n, t_a + t_n]$$

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

$$A = [B_1, t_a + t_1] \cdot [B_2, t_a + t_2] \cdot \cdots \cdot [B_n, t_a + t_n]$$

Maximal Delay Theorem: Let B_1 and B_2 be two timed boolean expressions involving only primitive input variables and without delay operator.

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

Boolean Propagation Theorem: Let B_1 and B_2 be two timed boolean expressions involving only primitive input variables and without delay operator. If $t_1 < t_2$, then

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

Modeling Logic Element

Logical Path

Here, two definitions related to the logical path will be given, including the *logical node* and the *logical path*. The formula to calculate the path delay will be also presented.

Definition [Logical Node]: For each physical node **N** in a logic network, there are two *logical nodes*, denoted as N and $\overline{N}$. N models the timing and boolean behavior of the node **N** in the logic state 1, while $\overline{N}$ models the node **N** in the logic state 0.

Definition [Logical Path]: Let **N1** and **N2** be the input and output node of a logic element $L2$, $N1$ model the physical node **N1** in logic state $v1$ and $N2$ model the physical node **N2** in logic state $v2$. If the change of $N1$ from $\overline{v1}$ to $v1$ will cause $N2$ change from $\overline{v2}$ to $v2$, then there exists a *logical path segment* from $N1$ to $N2$ through L, denoted as $N1$-(L)-$N2$.

A *logical path* $P = N_0$-(L_1)-N_1-(L_2)-N_2-$\cdots$-(L_k)-N_k is formed by a sequence of logical path segments, N_0-(L_1)-N_1, N_1-(L_2)-N_2, $\cdots$, and N_{k-1}-(L_k)-N_k.

Definition [Path Delay]: Given a path segment N_0-(L_1)-N_1, the path delay of N_0-(L_1)-N_1 is denoted as $\mathbf{d}_{N_0\text{-}(L_1)\text{-}N_1}$. The path delay of a logical path $P = S_0$-(L_1)-S_1-(L_2)-S_2-$\cdots$-(L_k)-S_k denoted as $\mathbf{d}_P$ and $\mathbf{d}_P = \sum_{i=1}^{k} \mathbf{d}_{S_{i-1}\text{-}(L_i)\text{-}S}$

Boolean Model of Logic Element

To facilitate delay extraction, 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 one models the falling behavior. In the following subsections,

Modeling of Simple Gates : Let $\mathbf{A_i}$ be the ith input node of an n-input gate L, **B** be the output node, The timed boolean functions of each different kinds of simple gates are as follows:

1. **Inverter:**

$$\begin{aligned} B &= [\overline{A_1}, \mathbf{d}_{\overline{A_1}\text{-}(L)\text{-}B}] \\ \overline{B} &= [A_1, \mathbf{d}_{A_1\text{-}(L)\text{-}\overline{B}}] \end{aligned}$$

2. **N-input AND gate:**

$$\begin{aligned} B &= [A_1, \mathbf{d}_{A_1\text{-}(L)\text{-}B}] \cdot [A_2, \mathbf{d}_{A_2\text{-}(L)\text{-}B}] \cdot \cdots \cdot [A_n, \mathbf{d}_{A_n\text{-}(L)\text{-}B}] \\ \overline{B} &= [\overline{A_1}, \mathbf{d}_{\overline{A_1}\text{-}(L)\text{-}\overline{B}}] + [\overline{A_2}, \mathbf{d}_{\overline{A_2}\text{-}(L)\text{-}\overline{B}}] + \cdots + [\overline{A_n}, \mathbf{d}_{\overline{A_n}\text{-}(L)\text{-}\overline{B}}] \end{aligned}$$

Similarly, the models for other types of simple logic gates can be derived.

Modeling of XOR (XNOR) Gates : Here, we take a 2-input XNOR gate as examples to demonstrate the modeling of XOR and XNOR gates. Let L be the gate's name, $\mathbf{A_i}$ be the ith input node, and **B** be the output node, the timed boolean functions of 2-input XNOR gate G are as follows:

$$\begin{aligned} B &= [A_1, \mathbf{d}_{A_1\text{-}(L)\text{-}B}] \cdot [A_2, \mathbf{d}_{A_2\text{-}(L)\text{-}B}] + \\ &\quad [\overline{A_1}, \mathbf{d}_{\overline{A_1}\text{-}(L)\text{-}B}] \cdot [\overline{A_2}, \mathbf{d}_{\overline{A_2}\text{-}(L)\text{-}B}] \\ \overline{B} &= [A_1, \mathbf{d}_{A_1\text{-}(L)\text{-}\overline{B}}] \cdot [\overline{A_2}, \mathbf{d}_{\overline{A_2}\text{-}(L)\text{-}\overline{B}}] + \\ &\quad [\overline{A_1}, \mathbf{d}_{\overline{A_1}\text{-}(L)\text{-}\overline{B}}] \cdot [A_2, \mathbf{d}_{A_2\text{-}(L)\text{-}\overline{B}}] \end{aligned}$$

Modeling of Pass Transistors : The pass transistors in which the direction of signal flow is fixed can be modeled in terms of two timed boolean functions. Assume that the delays for signal propagating through the pass transistors' (L) channels are given and the signal flow direction is 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 path delays for logical paths B-(L)-C, B-(L)-$\overline{C}$ therefore are $\mathbf{d}_{on} + \mathbf{d}_{A\text{-}(L)\text{-}C}$ and $\mathbf{d}_{on} + \mathbf{d}_{\overline{A}\text{-}(L)\text{-}\overline{C}}$ respectively. The timed boolean function of the type and N-type pass transistors are as follows:

$$\begin{aligned} C &= [A, \mathbf{d}_{A\text{-}(L)\text{-}C}] \cdot [B, \mathbf{d}_{on} + \mathbf{d}_{A\text{-}(L)\text{-}C}] \\ \overline{C} &= [\overline{A}, \mathbf{d}_{\overline{A}\text{-}(L)\text{-}\overline{C}}] \cdot [B, \mathbf{d}_{on} + \mathbf{d}_{\overline{A}\text{-}(L)\text{-}\overline{C}}] \end{aligned}$$

Similarly, the model for P-type transistor can be derived.

Modeling of Output-wiring : The output nodes 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 of the pseudo element, the timed boolean functions are

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

Modeling Logic Networks

Any single-output logic network can also be modeled in terms of a pair of timed boolean functions. By using the models for the logic elements, a single-output logic network can be represented as a set of timed boolean functions. This set of timed boolean functions can be further combined and reduced into a pair of timed boolean functions by eliminating all non-primary input variables. For an N-output logic network, there will be $2N$ sum-of-*Pterm* functions in the final reduction results.

Example

In the following, we will use the example in Fig. 1 to demonstrate the modeling and reduction process of a logic network.

1. By the models for the logic elements, we obtain:

$$\begin{aligned} D &= [\overline{B},1] \quad , \quad \overline{D} = [B,1] \\ E &= [D,2]\cdot[C,2] \quad , \quad \overline{E} = [\overline{C},1]+[\overline{D},1] \\ F &= [A,1]+[E,1] \quad , \quad \overline{F} = [\overline{A},1]\cdot[\overline{E},1] \\ G &= [F,1]+[C,1] \quad , \quad \overline{G} = [\overline{F},1]\cdot[\overline{C},1] \end{aligned}$$

2. Combine functions $E = [D,2]\cdot[C,2]$ and $\overline{E} = [\overline{C},1]+[\overline{D},1]$.

$$\begin{aligned} E &= [\overline{B},3]\cdot[C,2] \\ &= [\overline{B}\cdot C,3] \\ \overline{E} &= [B,2]+[\overline{C},1] \end{aligned}$$

3. Combine $F = [A,1]+[E,1]$ and $\overline{F} = [\overline{A},1]\cdot[\overline{E},1]$.

$$\begin{aligned} F &= [A,1]+[\overline{B}\cdot C,4] \\ \overline{F} &= [\overline{A},1]\cdot([B,3]+[\overline{C},2]) \\ &= [\overline{A},1]\cdot[B,3]+[\overline{A},1]\cdot[\overline{C},2] \\ &= [\overline{A}\cdot B,3]+[\overline{A}\cdot\overline{C},2] \end{aligned}$$

4. Combine $G = [F,1]+[C,1]$ and $\overline{G} = [\overline{F},1]\cdot[\overline{C},1]$.

$$\begin{aligned} G &= [C,1]+[A,2]+[\overline{B}\cdot C,5] \\ \overline{G} &= [\overline{C},1]\cdot([\overline{A}\cdot B,4]+[\overline{A}\cdot\overline{C},3]) \\ &= [\overline{C},1]\cdot[\overline{A}\cdot B,4]+[\overline{C},1]\cdot[\overline{A}\cdot\overline{C},3] \\ &= [\overline{A}\cdot B\cdot\overline{C},4]+[\overline{A}\cdot\overline{C},3] \end{aligned}$$

At first view, from these functions, we may take 4 as the maximal delay of the logic network. In next section, we will show the maximal delay is only 3 based on the theorem presented later.

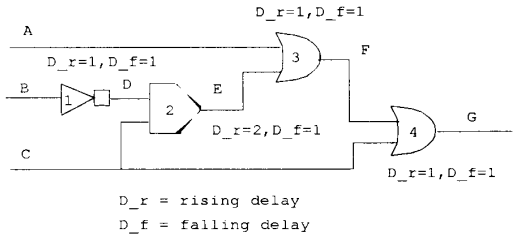

Figure 1: Path **C-(2)-E-(3)-F-(4)-G** is only sensitizable for **G**=0.

Procedure of Modeling a Logic Network

As shown in last subsection, the procedure of modeling a logic network is simple and can be stated as follows:

1. For each timed boolean functions whose right hand side involves only primitive input variables,
   (a) Substitute the timed boolean variables with their corresponding expressions.
   (b) Expand the resultant expressions by distribution laws.
   (c) Reduce the resultant expressions by maximal delay theorem.
2. Repeat step 1 until all timed boolean functions involves only primitive boolean variables.

After this procedure, each function is eventually transformed into a special form called the sum-of-*Pterm* form. This general form is expressed as $P = \sum_{i=1}^{n}[P_i,t_i]$, where each P_i involves only primitive input variables and AND operators. And $[P_i,t_i]$ is referred to as the *Pterm* of function P. The sum-of-*Pterm* form is analogous to the sum-of-product form in conventional Boolean Algebra.

Extract Timing Information

We have shown that a logic network can be modeled by pairs of functions in the sum-of-*Pterm* form. Now, we give the theorem and procedure to extract timing information from these functions.

Theorem [Sensitizable 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

$$P_i \cdot \overline{\Big(\sum_{\forall P_j,\, \mathbf{d}_{P_j}<\mathbf{d}_{P_i}} P_j\Big)}$$

where $\mathbf{d}_{P_i}$ denotes the delay value of P_i. If the sensitizability is not equivalent to 0, $[P_i,t_i]$ is referred to as a *sensitizable term*. Otherwise, $[P_i,t_i]$ is a *false term*.

Proof : Derived from Boolean Propagation Theorem.

Definition [Possible Delay Set] : The *possible delay set* of a timed boolean function $P = \sum_{i=1}^{n}[P_i, t_i]$ in sum-of-*Pterm* form is the union of delay value t_i of all sensitizable terms $[P_i, t_i]$. And the maximal delay to logical node P is the maximal value of possible delay set of the timed boolean function P.

Case Study

Take the logic network in Fig. 1 as an example. The two functions are $G = [C,1]+[A,2]+[\overline{B}\cdot C,5]$ and $\overline{G} = [\overline{A}\cdot B\cdot\overline{C},4]+[\overline{A}\cdot\overline{C},3]$. The sensitizability of each *Pterm* is listed below:

$$\begin{aligned} [C,1] &\longrightarrow C \\ [A,2] &\longrightarrow A\cdot\overline{C} \\ [\overline{B}\cdot C,5] &\longrightarrow 0 \\ [\overline{A}\cdot\overline{C},3] &\longrightarrow \overline{A}\cdot\overline{C} \\ [\overline{A}\cdot B\cdot\overline{C},4] &\longrightarrow 0 \end{aligned}$$

So, the possible delay sets for G and $\overline{G}$ are {1,2} and {3} respectively. The maximal delays of G and the input vector must satisfy $A\cdot\bar{c}$, while the maximal delay of $\overline{G}$ is 3 and the input vector must satisfy $\overline{A}\cdot\overline{C}$.

Implementation and Experimental Results

Although the Timed Boolean Algebra provides a strongly theoretic basis for timing analysis, for the logic network with large number of gates, the functions are too complex to be represented. Thus, a heuristic approach was developed. It can get more precise delays than the ones extracted by the methods without false path detection. It also guarantees the the reported delays are upper bounds.

Representation of Timed Boolean Functions

The problem of representing timed boolean functions is the exponentially growing number of *Pterms*. To reduce the number of *Pterms* (memory), several rules was developed:

1. Based on the **Boolean Propagation Theorem**, the false terms can be removed. Whenever each timed boolean function is derived, the sensitizabilities of all *Pterms* are calculated and all *false terms* are removed immediately.

2. The procedure of modeling a logic network performs a *depth-first* traversal algorithm. As soon as the succeeding nodes are all processed, the memory used by the node is freed.

3. Given a timed boolean function $P = \sum_{i=1}^{n}[P_i, t_i]$, if n is larger than some threshold (*cutting threshold*), it is replaced by
$$P = [P, \max_{i=1}^{n}(t_i)]$$

The first and second rules cause no loss of the accuracy of the proposed algorithm, while the third one guarantees the extracted delays be the upper bounds.

Experimental Results

The proposed heuristic algorithm for timing analysis have been implemented in C and runs on SUN workstations. The test cases used are the ISCAS circuits. Without loss of the generality, the delays of all logic elements in these circuits are assumed to be *unit delay*. The characteristics of these circuits are listed in table 1.

The computer used is SUN4/60 workstation (SPARCstation1) and its CPU performance is about 12.5 MIPS. The detailed experimental results are shown in table 2. From these tables, we can see (1) the proposed algorithm is an efficient one. Most experiments take les than 1000 seconds. (2) many nodes' delays have been improved, and (3) the memory required is small (less than 25M bytes). More detailed information about the delay units improved among those improved nodes are shown in table 3.

Our experience shows that applying larger *cutting threshold* can obtain better results. Thus, it is better to run each logic network several times by applying different *cutting thresholds* and take the best result among these runs as the final result.

Conclusion

To address the false path problem in timing analysis, we present a new methodology, which is based on a formalism called Timed Boolean Algebra. The algebra is derived by extending the conventional Boolean Algebra with a delay operator. The operators of the Timed Boolean Algebra facilitate the modeling of the logic and timing behavior of logic networks and provide the theoretic basis for extracting important timing information. There are three innovative ideas involved in our approach: (1) extend the conventional Boolean Algebra with a delay operator, (2) use algebraic method to deal with the problem of reporting sensitizable paths and delay analysis, (3) propose the logical path concept and use it in modeling logic element. To reduce the memory usage, a heuristic approach has been developed. The experimental results show that our approach is a good and efficient one.

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. 100-116, Jan. 1982.

[2] Lionel C. Bening, and etc., "Developments in Logic Network Path Delay Analysis," *Proc. of 19th ACM/IEEE DAC*, pp. 605-615, 1982.

[3] Norman P. Jouppi, "Timing Analysis for nMOS VLSI," *Proc. of 20th ACM/IEEE DAC*, pp. 411-418, 1983.

[4] Edward Chan, "Development of a Timing Analysis Program for Multiple Clocked Network," *Proc. of 22nd ACM/IEEE DAC*, pp. 816-819, 1985.

[5] Michiaki Muraoka, Hirokazu Iida, and etc., "ACTAS: An accurate Timing Analysis System for VLSI," *Proc. of 22nd ACM/IEEE DAC*, pp. 152-158, 1985.

[6] Norman P. Jouppi, "Timing Analysis and Performance Improvement of MOS VLSI Designs," *IEEE Tran. on CAD*, VOL. CAD-6, NO. 4, JULY 1987.

[7] H.C. Yen, S. Ghanta, H.C. Du, "A Path Selection Algorithm for Timing Analysis," *Proc. of 25th ACM/IEEE DAC*, pp. 720-723, 1988.

[8] 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 ACM/IEEE DAC*, pp. 649-653, 1989.

[9] Robert B. Hitchcock, Sr., "Timing Verification and the Timing Analysis Program," *Proc. of 19th ACM/IEEE DAC*, pp. 594-604, 1982.

[10] Daniel Brand, and Vijay S. Iyengar "Timing Analysis Using Functional Relationships," *Proc. of 23rd ACM/IEEE DAC*, 1986

[11] J. Benkoski, E. Vanden Meersch, L. Claesen, and H. De Man "Efficient Algorithms for Solving the False Path Problem in Timing Verification," *Proc. of 24th ACM/IEEE DAC*, 1987.

[12] David H.C. Du, Steve H.C. Yen, and S. Ghanta "On the General False Path Problem in Timing Analysis," *Proc. of 26th ACM/IEEE DAC*, 1989.

[13] Patrick C. McGeer, Robert K. Brayton, "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network," *Proc. of 26th ACM/IEEE DAC*, pp. 561-567, 1989.

[14] Patrick C. McGeer, Robert K. Brayton, "Timing Analysis in Precharge/Unate Networks," *Proc. of 27th ACM/IEEE DAC*, pp. 124-129, 1990.

CKT	Physical Nodes			Gates	Maximal delay
	Input	Output	Internal		
C432	36	7	153	160	17
C499	41	32	170	202	11
C880	60	26	357	383	24
C1355	41	22	153	546	24
C1908	33	25	855	880	40
C2670	157	64	1129	1193	32
C3540	50	20	1647	1669	47
C5315	178	123	2184	2307	49
C6288	32	32	2384	2416	124
C7552	206	107	3405	3512	43

Table 1: Characteristics of ISCAS benchmarks

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	1458118	49324
c7552	957	1753.57	23631	443465	18905

Table 2: Results of ISCAS benchmarks (threshold : 2000)

Delay	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					

Table 3: Improved delays and node number (threshold : 2000)