Efficient Symbolic Simulation under the Extended Bounded Delay Model for Transition Mode Timing Analysis

Chihshun Ding and Massoud Pedram

CENG Technical Report 93-14

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4458

April 1993

Contents

List of Figures

List of Tables

1 Introduction

Timing analysis plays an important role in performance verification of logic networks. As sizes of networks grow, the graphical (topological) path delay estimates become less accurate. Therefore, more realistic timing analysis techniques must be exploited in order to get better delay estimates. Existing techniques are classified into two two groups: floating mode and transition mode. In the floating mode timing analysis, a single input vector is applied to the network. The state of each gate before application of the vector is assumed to be unknown or X. The viability [1] and Chen-Du criteria [2] are examples of this type of analysis. In the transition mode timing analysis, two input vectors are applied to the circuit. Enough time is allowed for the effect of the first vector to propagate through the circuit before the second vector is applied. Thus, the state of each gate before application of the second vector is known.

No delay model is really involved in the floating mode timing analysis. In contrast, delay models play an important role in transition mode timing analysis as different delay models produce different delay estimates. McGeer el at. [3] describe delay models which are often used during the transition mode timing analysis: the Fixed Binary Delay (FBPD) Model, the Fixed Binary Delay With Static Variation (FBPD-SV) Model, the Extended Bounded Delay (XBD) Model and the Extended Bounded Delay-0 (XBD0) Model. In the FBPD model, each gate in the network assumes a fixed delay and transitions at the gate output are between binary logic values. The FBPD-SV model is similar to FBPD model except that the delay of each gate f lies in a range $[f_{min}, f_{max}]$. The XBD mode also represent the delay as a range $[f_{min}, f_{max}]$. However, the output of the gate during the gate delay is assumed to be unknow or X. The transitions are between binary logic values and X. The XBD0 model is the same as XBD except that f_{min} in the delay range is always zero. In the same paper, it is shown that the timing analysis in transition mode under the XBD0 model is equivalent to the floating mode timing analysis. The connection between floating mode and transition mode is therefore established. Devadas et al. [4] gives an example to demonstrate that floating mode network delay could be different from transition mode network delay under other delay models.

Recently, transition mode timing analysis has received much attention [4, 5, 6, 3]. [4] assumes the FBPD and XBD0 delay models. The FBPD model ignores many real world factors such as gate delay variations and slope factors. In contrast, the XDB0 model exaggerates this variation by setting f_{min} to zero. [4] uses forward (from circuit inputs to circuit outputs) symbolic simulation to calculate the timing waveform for each gate in the network.

[3] assumes the XBD and XBD-0 models and introduces the concept of time quantum in order to reduce the computational complexity of waveform calculation. The main difficulty with this approach is how to determine the best tradeoff between accuracy and efficiency. Large time quantum size reduces the computation time and storage requirements at the cost of increased inaccuracy, and vise versa.

In this paper, we propose a new timing analysis approach which eliminates the need for quantizing the time axis while maintaining outmost accuracy. The delay model we assume is the XBD model. Our approach employs a backward (from circuit outputs toward circuit inputs) symbolic simulation technique. Given some time interval, we search for a pair of input vectors which produce waveforms other than constant one or constant zero at the network primary outputs. For each gate, the minimum number of symbolic formulas required at the gate input terminals are found. Conditions for each gate to assume value one or zero during some time interval are represented in a dual form[1]. This dual representation enables us to employ a local transformation procedure on symbolic formula to further reduce the number of necessary logic operations before passing the formula to the tautology checker. If the tautology check fails, then the time interval is decreased and the same routine is repeated until the true network delay is found. The first advantage of our approach is that it requires significantly less number of logic operations in the resulting symbolic formula than an conventional symbolic simulation approach. Secondly, since conditions for each gate to assume value one or zero are represented as dual forms, a very simple local transformation procedure is possible to detect important local implications in the symbolic formula before passing it to the tautology checker. Unnecessary backtrackings can be thus avoided.

The rest of the paper is organized as follows. Section 2 gives some useful definitions and notations. The relation between transition waveform and delay bounds are given in Section 3. In Section 4 and 5, we present the approach. The path sensitization criterion under XBD model is given in Section 6. Section 7 provides the result.

2 Notations and Definitions

Network $\mathcal{N}$ refers to a directed acyclic graph $\mathcal{G}$ with nodes representing gates and primary inputs, and edges representing connections. To simplify presentation, we assume that the network consists of simple gates[2]. A *simple gate* is either an inverter, AND, NAND, OR, or NOR gate. The *controlling value* c is defined as logic 1 (0) for OR and NOR (AND and NAND)gates. The *sensitizing value* s is defined as complement of the controlling value. (Neither is defined for inverters.) The *inversion* i is defined as logic 1 (0) for NAND and NOR (AND and OR) gates. If some input of a gate g assumes controlling value, then the gate's output will be $c \oplus i$. If every input assumes sensitizing value, then the gate's output will be $s \oplus i = \overline{c} \oplus i$.

We adopt the Extended Bounded Delay Model. The delay of a node is assumed to be independent of its input terminals. In reality, the delay through a node might depend on the input terminal the transition came from. Nevertheless, nodes with in-degree 1 and out-degree 1 can be inserted

[1]That is, the representation for one of them is obtained by replacing AND (OR) gates with OR (AND) gates and reversing the input polarities in the representation for the other.

[2]Section 5.5 describes extension of our technique to a network with complex gates.

into those input terminals to account for such effects. The network will receive two vectors, the first one v_1 is applied to primary inputs long enough for the network to stabilize before the second vector v_2 is applied at time zero. The primary inputs are assumed to have zero delays. Again, if the arrival time of the second vector is different from zero at some primary input, dummy nodes can be inserted into $\mathcal{G}$ in the way we just described.

Given a node n, the *transition waveform* for n is a mapping $wf(n)$ from the set of real numbers R to the set $\{0, 1, X\}$. $wf(n)$ will be denoted by a *segment set*

$$\{(V_i, [t_i, h_i]) | i = 0, 1, \ldots, n, V_i \in \{0, 1, X\} \text{ and } V_i \neq V_{i+1}, t_0 = -\infty, h_n = \infty, t_i = h_{i-1}\}$$

where h_i and t_i are called the *segment bounds*. If $V_i = 1$ (0), then $[t_i, h_i]$ is called a 1-*segment* (0-*segment*). Similarly, if $V_i = X$, then $[t_i, h_i]$ will be called an X-*segment*. 0- and 1-segments are also referred as binary-segments. The waveforms at primary inputs are classified into four groups: $w_{00} = \{(0, [-\infty, \infty])\}$, $w_{01} = \{(0, [-\infty, 0]), (1, [0, \infty])\}$, $w_{10} = \{(1, [-\infty, 0]), (0, [0, \infty])\}$ and $w_{11} = \{(0, [-\infty, \infty])\}$.

A path is defined by a sequence of nodes and is denoted by $P = \{f_0, f_1, f_2, \ldots, f_m\}$ where f_0 is a primary input node. A partial path of P is denoted by $P_k = \{f_0, f_1, f_2, \ldots, f_k\}$. The *delay segment* of a path P is denoted by $d(P) = [min_{d(P)}, max_{d(P)}]$ where $min_{d(P)} = \sum_{f_i \in P} min_{f_i}$ and $max_{d(P)} = \sum_{f_i \in P} max_{f_i}$. $min_{d(P)}$ is called the *left bound* of $d(P)$ while $max_{d(P)}$ is called the *right bound* of $d(P)$. We will refer to $[min_{d(P)}, max_{d(P)}]$ simply as the "path delay". $d(P)$ is calculated without considering the side inputs. It represents the worst case scenario for path P, that is, if all side inputs assume constant sensitizing values and there is a transition at the head of path P, then the tail of path P will be X during $[min_{d(P)}, max_{d(P)}]$. The actual waveform at the tail of path P will, of course, depend on the waveform of the side inputs. For each node n, the collection of paths originating from primary inputs and terminating at n will be denoted by $P(n)$.

3 Calculation of Transition Waveforms

This section describes a procedure for calculating the transition waveforms at the internal nodes of the network. It also shows that binary segments start with the right bound of some path and end with the left bound of some (other) path. This result is used in section 4 to compute the minimum number of symbolic formula (needed for timing analysis) at each node. The section ends with a lemma that relates the type of segment seen as part of the waveform for a gate to the polarities of the path delays of some paths terminating at n.

3.1 Waveform Calculation under the Extended Bounded Delay Model

Given the delay range of a node n and the transition waveforms for its inputs, the transition

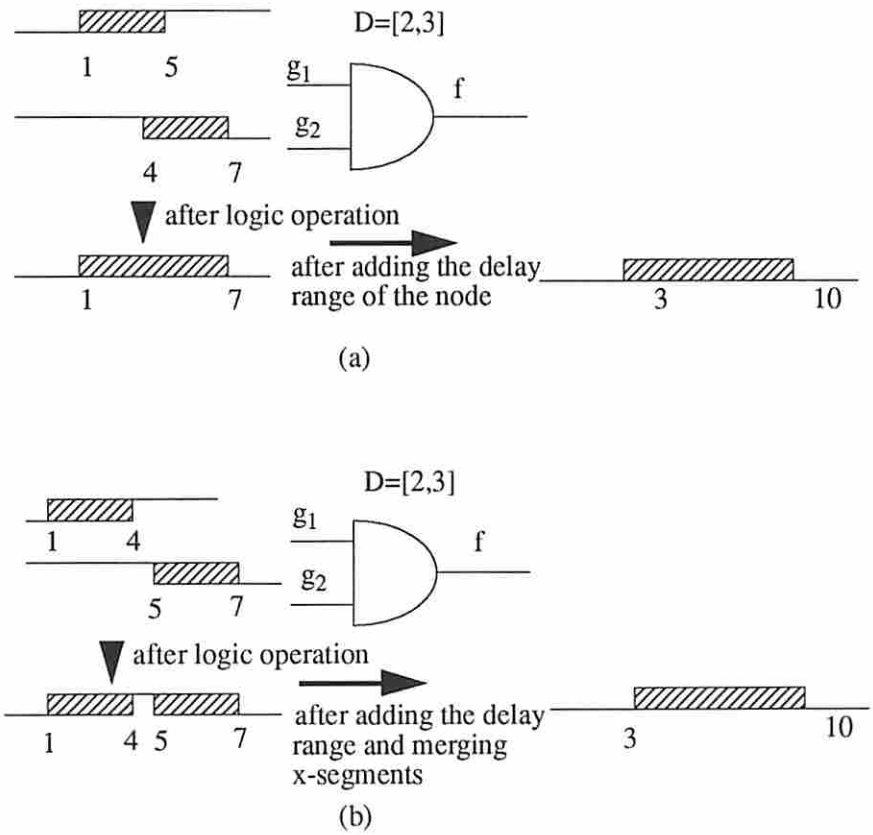


Figure 1: Examples of waveform calculation procedure.

waveform at the output of the node can be calculated by the following two procedures. First, we perform a ternary logic operation (corresponding to the function of the node) on the input waveforms. We then add the delay range of the node itself to each X-segment as shown in Figure 1. If any two X-segments overlap, we merge them into one segment.

The waveforms for all gates in the network (except primary inputs) alternate between binary values and X's. For example, it is possible to have a waveform which takes values 0,X,0,X,1...while 0,1,X,0,1...is not a legal waveform under the XBD model. As shown in Figure 1, node n assumes X starting from time 3, which is equal to 1 (when one of of its inputs assumes X) plus $d_{min} = 2$, and extending to time 10, which is equal to 7 (when one of its inputs assumes X) plus $d_{max} = 3$. If we apply induction on levels of the network, it is easily shown that X-segments in a waveform for node n always start with the left bound of some path P_m, and end with the right bound of some (other) path P_n. Similarly, the binary-segments start with the right bound of some path and end with the left bound of some path.

3.2 Polarity of Bounds for Path Delays

We distinguish boundary between an X segment and a 1 segment from that between an X segment and a 0 segment by assigning different polarities to the left and right bounds of path delays, as described next. This is useful when gates exhibit asymmetric fall and rise delays.

Polarity of the left and right bounds of the delay segment of a path $P\{f_0, f_1, f_2, \ldots, f_m\}$ depends on the logic value of the input vectors at f_0 as well as the inversion parity on P[3]. The left bound $min_{d(P)}$ is called a *positive* left bound if $v_1(f_0) = \overline{v_2(f_0)} = 0$ and the inversion parity of P is zero, or if $v_1(f_0) = \overline{v_2(f_0)} = 1$ and the inversion parity is one. The corresponding right bound is called *positive* right bound. On the other hand, the left bound is called a *negative* left bound if $v_1(f_0) = \overline{v_2(f_0)} = 1$ and the inversion parity is zero, or if $v_1(f_0) = \overline{v_2(f_0)} = 0$ and the inversion parity is one. The corresponding right bound will also be called *negative* right bound. The reason we assign polarity to the delay segment bounds is that if P is sensitized at the positive left bound, the gate f_m should have a 0 to X transition. On the other hand, for P to be sensitized at the positive right bound, f_m should have a X to 1 transition. As a result, a positive left bound represents a possible transition from a 0-segment to an X-segment and a positive right bound represents a possible transition of an X-segment to a 1-segment. We formalize this notation in the following lemma.

Lemma 1 *Let wf(n) be a segment set representation of the waveform for gate n. Let s=(l,r) be some segment in set wf(n). Then,*

(a) If s is a 1-segment, then l is a positive right bound of the delay segment of some path p in

[3]The inversion parity of P is zero if there is an even number of inversion on that path. Otherwise, it is one.

P(n) and r is a negative left bound of some path p' in P(n);

(b) If s is a 0-segment, then l is a negative right bound of the delay segment of some path p in P(n) and r is a positive right bound of the delay segment of some path p' in P(n);

(c) If s is an X-segment, the l is a left bound (either positive or negative) of some path p in P(n), and r is a right bound (either positive or negative) of some path p' in P(n);

where p may be different from p'.

Despite of this discussion, we will limit our attention to gates with symmetric rise and fall delays in the remainder of the paper.

4 Transition Mode Timing Analysis under the XBD Model

This section gives an equation that if evaluated to true, would determine the longest path delay in the network. This equation is then converted to a set of symbolic formula in terms of the values of circuit inputs under the two input vectors. It is shown that our procedure generates the least number of logic operations in the symbolic formula and hence is very efficient.

4.1 Computing the True Delay of a Network

To determine the true delay of a network, we essentially need to find the earliest time t after which transition waveforms at the primary outputs are either constant one or zero. Before describing how to find such t, we introduce some more notation.

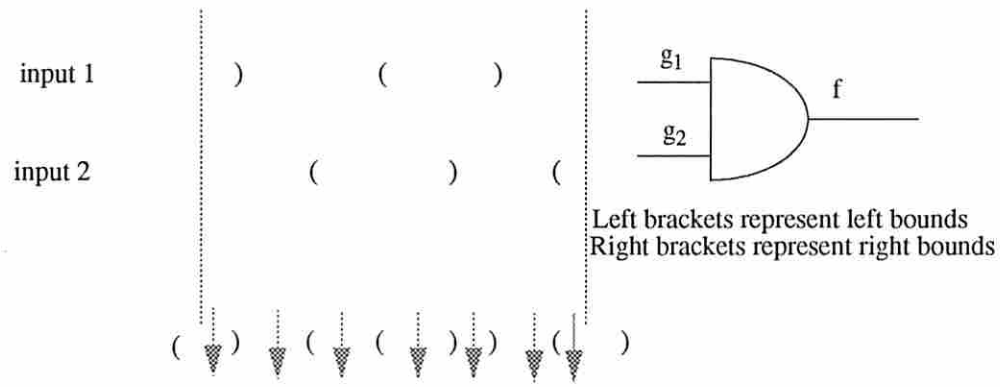
Let $S^f(C^f)$ denote the sensitizing (controlling) value of node f in the network. Nodes connected to the input pins of f might have a different sensitizing (controlling) values from $S^f(C^f)$. For clarity, we use subscript to denote the node that is assuming the value, and superscript to denote the node the sensitizing (or controlling) value is referred. Thus, $S_g^f(v_1)$ ($C_g^f(v_1)$) denotes the boolean function which causes (output of) node g assume steady state sensitizing (controlling) value of (output of) node f under the first vector. $S_g^f(v_2)$ ($C_g^f(v_2)$) is defined similarly. Since the first vector is applied to the primary inputs up to time zero, the output of any node in the network will remain $S_f^f(v_1)$ or $C_f^f(v_1)$ at least up to time zero. After time zero, the output of the node may undergo some transitions before settling to its steady state value under the second vector. The temporal behavior of g in interval $[t_1, t_2]$ will be denoted by $s_g^f(t_1, t_2)$ ($c_g^f(t_1, t_2)$) which represents the boolean function that forces node g to assume dynamic sensitizing (controlling) value of node f in the interval $[t_1, t_2]$. Note that $S_g^f(v_1)$ depends on n input variables, while $s_g^f(t_1, t_2)$ depends on $2n$ input variables. The inputs to node f are denoted as $FI(f)$.

The true delay of a network is the earliest t which causes the following boolean function evaluate to one:

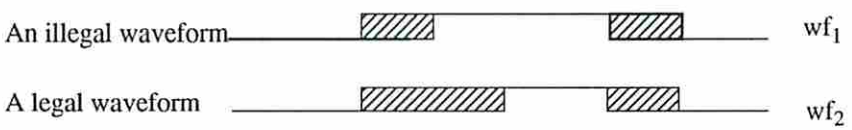
$$\Psi_N(t,\infty) = \prod_{f \in PO(N)} (s_f^f(t,\infty) + c_f^f(t,\infty)). \quad (1)$$

4.2 Derivation of Symbolic Formulas

$s_f^f(t,\infty)$ and $c_f^f(t,\infty)$ are in turn expressed as functions of the symbolic formulas at the inputs of node f.



(a)



(b)

Figure 2: Examples of legal and illegal waveforms

Let the delay range of node f be $[f_{min}, f_{max}]$, then all inputs to f must assume sensitizing values during $[t_1 - f_{max}, t_2 - f_{min}]$, that is,

$$s_f^f(t_1,t_2) = \prod_{g \in FI(f)} s_g^f(t_1 - f_{max}, t_2 - f_{min}). \quad (2)$$

Similarly, at any time instance during $[t_1 - f_{max}, t_2 - f_{min}]$, at least one input of f must assume controlling value. In the forward symbolic simulation, an individual symbolic formula is built for each time instance that the output of f may change. These formulas are then conjoined together. In our approach, we generate some symbolic formulas which are necessary for exact timing analysis as described below.

Figure 2(a) shows a node f and the path delay bounds at its inputs g_1 and g_2. The method presented in [3] would generate (at least) 7 formulas. These formulas represent the logic values over segments separated by the path delay bounds. However, we can do even better, that is, in fact only three symbolic formulas are needed to determine $c_f^f(t_1, t_2)$.

Figure 2(b) shows the union of right and left bounds into a list. We add a pseudo left and a pseudo right bound to this list to represent the nearest left bound and right bound outside the interval. Two waveforms wf_1 and wf_2 are shown in this figure. wf_1 is not a legal waveform, while wf_2 is. The reason is that the binary-segments have to start with a right bound and end with a left bound (see lemma 1). If we were to use all 7 formulas, we essentially assume any possible waveform at f is legal which is not true. Indeed, if we know that a waveform assumes a controlling (sensitizing) value at time t, then the waveform will assume the same value from the nearest right bound to the left of time t, up to the nearest left bound to the right of time t. Such a segment which contains t and assumes the same logic value as that at time t is called the *stable segment* of t, $MS(t)$.

Figure 3 shows the stable segment for various time instances. We need to construct symbolic formulas at a minimum number of time instances that ensure f will assume controlling value from t_1 to t_2. For this example, time instance 1, 4 and 7 forms a *minimum (cardinality) covering set*, $MCS_f(t_1, t_2)$. Therefore,

$$c_f^f(t_1, t_2) = \prod_{t \in MCS_f(t1-max_f, t2-min_f)} \sum_{g \in FI(f)} c_g^f(t, t). \tag{3}$$

4.3 Computing the Minimum Covering Set

Our task is to find the minimum number of time instances, $MCS_f(t_1, t_2)$, such that the stable segment of each time instance in $MCS_f(t_1, t_2)$ overlaps with another, and these stable segments together cover $[t_1, t_2]$. As we see in Figure 3, some of the stable segments are contained by the others. They will be called dominated segments. The remaining segments are called dominating segments. The dominated segments could never be members of $MCS_f(t_1, t_2)$, since they could always be replaced by the corresponding dominating segments without increasing the cardinality of the minimum covering set. The key observation, however, is that once the dominated segments are

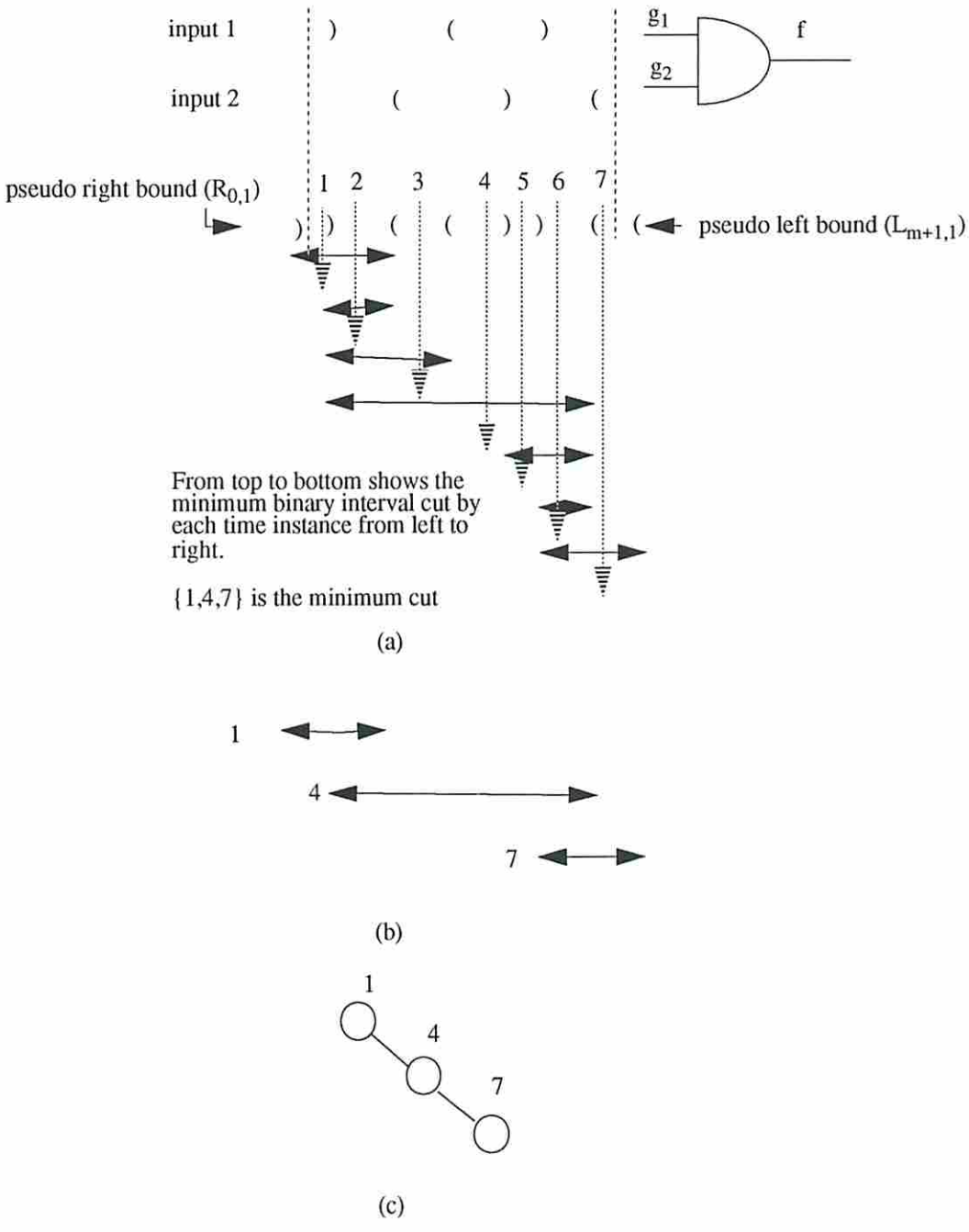


Figure 3: Minimum segments for each time instance

eliminated, the *interval graph*[4] corresponding to the remaining segments is *chain*[5] as shown next.

After sorting and removing duplicates, the list containing the left and right bounds of path delays which fall in the interval $[t_1, t_2]$ can be written as $List = \{R_{1,1}R_{1,2}\ldots R_{1,j_1}L_{1,1}\ldots L_{1,k_1} R_{2,1}R_{2,2}\ldots R_{2,j_2}L_{2,1}\ldots L_{2,k_2}\ldots R_{m,j_m}\ldots L_{m,k_m}\}$, where R and L denote the right and left bounds, respectively. If the first symbol in the list is an L, then $\{R_{1,1}R_{1,2}\ldots R_{1,j_1}\}$ are treated as empty. If the last symbol in the list is an R, then $\{L_{m,1}\ldots L_{m,k_m}\}$ will be treated as empty. After that an $R_{0,1}$ and an $L_{m+1,1}$ are appended to the head and tail of this list respectively to account for the nearest left and right bounds outside the interval, that is, $List' = \{R_{0,1}\} \bigcup List \bigcup \{L_{m+1,1}\}$.

The dominance relation is described as in the following lemmas.

Lemma 2 *If $L_{i-1,j_{i-1}} \leq t < R_{i,1}$, $1 \leq i \leq m$, then the stable segment of t, $MS(t)$, will dominate the stable segments of t', where $R_{i-1,j_{i-1}} \leq t' < L_{i,1}$.*

Proof : Since $L_{i-1,j_{i-1}} \leq t < R_{i,1}$, $MS(t) = [R_{i-1,j_{i-1}}, L_{i,1}]$. Assume that the lemma is not true, there exists a t'' such that $MS(t'') = [R', L']$, where $R' < R_{i-1,j_{i-1}}$ and $L_{i,1} < L'$. Since $MS(t'')$ start with nearest right bound and end with the nearest left bound, therefore $R' \geq R_{i-1,j_{i-1}}$ and $L_{i-1,1} \geq L'$. A contradiction. □

Lemma 3 *If the last symbol in List is an L, then the stable segment of time t, where $L_{m,k_m} \leq t < t_2$, will dominate the stable segment of time t', where $R_{m,j_m} \leq t' < t_2$.*

Proof : Same reason as above. □

If the *List* itself is an empty list, we still need to put an time instance in MCS_f. In this case, any time instance within $[t_1, t_2]$ will do the job as their stable segment will at least extend over the nearest right and left bounds outside $[t_1, t_2]$. We pick $\{t_2 - \epsilon\}$. Next we will prove that the dominating stable segments described above altogether form a chain on the corresponding interval graph, therefore they should be all included in MCS_f.

Theorem 1 *Let $S = \{t_i | i = 1, \ldots, m$, and $t_i = R_{i,1} - \epsilon\}$ if the last symbol in List is an R, otherwise $S = \{t_i | i = 1, \ldots, m+1$, $t_i = R_{i,1} - \epsilon$ for $i \neq m+1$ and $t_{m+1} = t_2 - \epsilon\}$. If we build the interval graph corresponding to the stable segments of all the elements in S, then this graph is a chain.*

Proof : We prove the first case as the second case is similar. $MS(t_i) = [R_{i-1,j(i-1)}, L_{i,1}]$. The left bounds of $MS(t_i)$ are strictly increasing and the right bounds are also strictly increasing. Furthermore, since $L_{i,1} < R_{i+1,j_{i+1}}$, $MS(t_i)$ doesn't overlap with $MS(t_j)$ where $i + 2 \leq j \leq m$. Therefore, t_i's form a chain in the overlap graph. □

[4] We define the overlap graph as G=(V,E), where $V = \{v_i | v_i$ represents interval $I_i\}$. An edge is defined between v_i and v_j if I_i and Ij overlap.

[5] A chain is a graph, in which there are a unique source and sink vertices, and only one path between them.

Corollary 1 *Node f will assume controlling value in interval $[t_1, t_2]$ exactly if Equation 3 is true.*

Using the notion of a minimum covering set, Equation 2 can be written as:

$$s_f^f(t_1, t_2) = \prod_{t \in MCS_f(t1-max_f, t2-min_f)} \prod_{g \in FI(f)} s_g^f(t, t). \tag{4}$$

The advantage of this "dual" representation of c_f^f and s_f^f will be discussed next.

5 Efficient Representation of Symbolic Formulas

This section starts with a multi-level network representation of symbolic formulas, followed by a local transformation procedure that will reduce the number of nodes in the multi-level network even further and reduce the number of backtrackings during the tautology checking of the network delay equation. Extension to complex gates and some degenerate cases of XBD model are discussed.

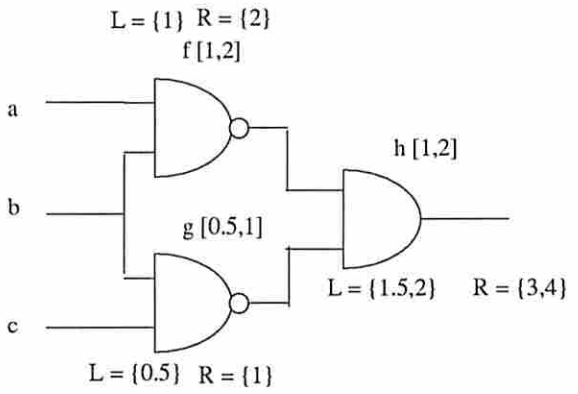
5.1 Symbolic Network Representation

The reason we base the above discussion on sensitizing and controlling values is that it is easier to demonstrate the idea on simple gates. In actual implementation, those symbolic formulas are labeled as logic one or logic zero instead, and are represented by a multi-level network. Each node in this network represents a logic operation in the symbolic formula while each edge represents the destination of the operation. We use $f^1(t_1, t_2)$ ($f^0(t_1, t_2)$) to denote conditions for node f to assume value one (zero) during time interval $[t_1, t_2]$.

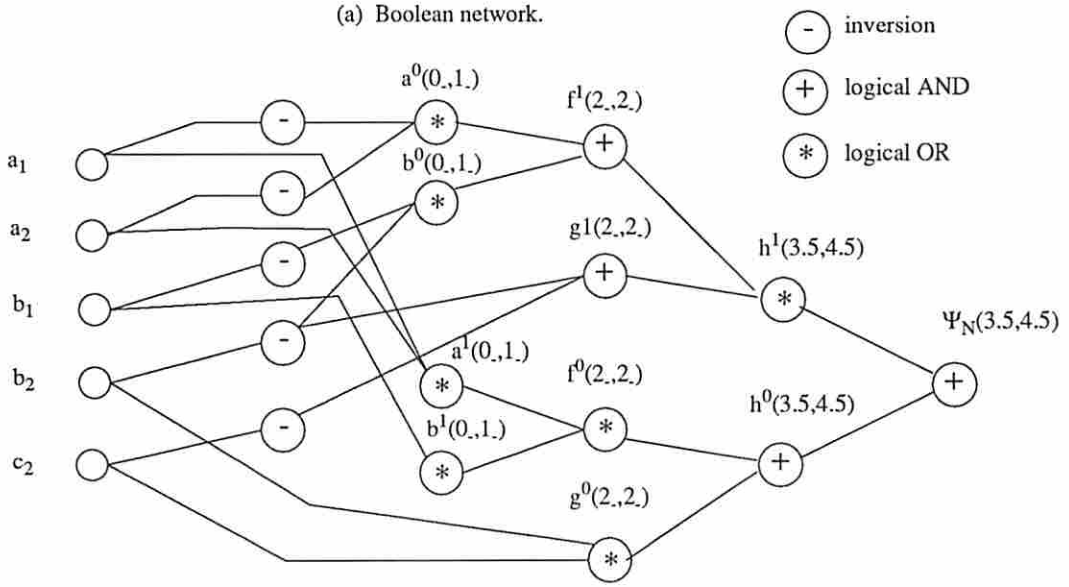
When we finally visit a primary input f, depending on the values of t_1 and t_2, there are three possible expressions for $f^1(t_1, t_2)$ ($f^0(t_1, t_2)$). If $t_1 > 0$, then $f^1(t_1, t_2)$ ($f^0(t_1, t_2)$) will be $v_2[f]$ ($\overline{v_2[f]}$), where $v_2[f]$ denotes the value of f under the second input vector. Similarly, if $t_2 < 0$, then $f^1(t_1, t_2)$ ($f^0(t_1, t_2)$) will be $v_1[f]$ ($\overline{v_1[f]}$). Otherwise, it will be $v_1[f]v_2[f]$ ($\overline{v_1[f]} \cdot \overline{v_2[f]}$).

5.2 An Example

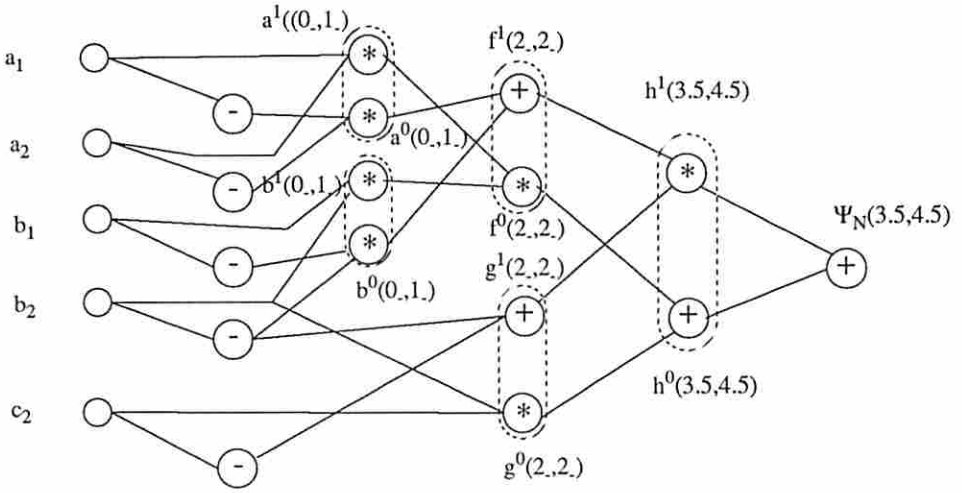
Figure 4(a) shows a network with three gates f, g and h. The gate delays are shown by the gate names. R and L are the set of right and left bounds, respectively. We will use Equations 3 and 4 in this example. Suppose we would like to find out if the network delay is greater than 3.5. Figure 4(b) shows the multi-level network corresponding to the symbolic formula $\Psi_N(3.5, 4.5)$. $\Psi_N(3.5, 4.5) = h^1(3.5, 4.5) + h^0(3.5, 4.5)$. Now, $MCS_h(3.5-2, 4.5-1) = MCS_h(1.5, 3.5) = \{2\_\}$ where $2\_$ represent the nearest time instance to the left of time 2. Therefore $h^1(3.5, 4.5) = f^1(2\_, 2\_)g^1(2\_, 2\_)$. $MCS_f(2\_ - 2, 2\_ - 1) = MCS_f(0\_, 1\_)$. $f^1(2\_, 2\_) = a^0(0\_, 1\_) + b^0(0\_, 1\_) = (\overline{a_1} \cdot \overline{a_2} + \overline{b_1} \cdot \overline{b_2})$,



(a) Boolean network.



(b) Symbolic network.



(c) Rearranged symbolic network.

Figure 4: An Example.

$MCS_f(2\_ - 1, 2\_ - 0.5) = MCS_f(1\_, 1.5\_)$. Therefore $g_1(2\_, 2\_) = \overline{b_2} + \overline{c_2}$. Calculation of $h^0(3.5, 4.5)$ is carried out similarly (see Figure 4(b)).

We rearrange the symbolic network in Figure 4(b) to obtain that in Figure 4(c). Note that the nodes encircled together are dual of one another and were built for the same node in the original network.

5.3 The Encoding Scheme

There is a unique implicit encoding scheme imposed on the ternary logic by our approach. If we treat g^1 as the first bit and g^0 as the second bit, we essentially represent logic one as 10, logic zero as 01 and all the other cases as 00. 11 is an illegal code. This is different from the encoding scheme adopted in most ternary logic systems.

5.4 Reducing Formulas by Local Transformation

In the multi-level network corresponding to the symbolic formula, subnetworks of the form shown in Figure 5(a) are likely to occur. We will refer to this structure as *local inversion implication* because $o_2 = \overline{o_1}$. For example, $g^1(2\_, 2\_)$ and $g^0(2\_, 2\_)$ and their transitive fanin nodes in Figure 4(c) exhibit this structure. The reason that this structure shows up so often is that logic values of some nodes in the network, after certain time instances, will remain fixed at one or zero. For example, the longest delay through gate g is 1 which is smaller than $3.5 - 2 = 1.5$. Therefore, g will have settled to its stable value under the second vector at time 1.5. Another less obvious example is when the associated interval $[t_1, t_2]$ for a symbolic node does not overlap with any delay segments of the node that it is built for.

These structure must be transformed into the structure shown in Figure 5(b). If they are left in the symbolic network, the performance of those tautology checkers based on CNF or $D-$algorithm will greatly deteriorate[6]. Outputs o_1 and o_2 in Figure 5(a) have zero parity to any primary output of the multi-level network. The tautology checker needs to find an input assignment to force at least one of the primary outputs to zero. If the tautology checker is unware of the fact that $o_1 = \overline{o_2}$, it will try to find an input assignments which sets both o_1 and o_2 to zero. Thus, it will try all possible input assignments until it fails. These backtrackings could have be avoided if structure Figure 5(a) is converted to that in Figure 5(b). This procedure is carried out from the primary inputs to the primary outputs. Only one pass is adequate, since structure in Figure 5(a) could only happen among symbolic nodes corresponding to the same node in the original network.

[6]This is especially true when such structures are located at a distance from the primary inputs.

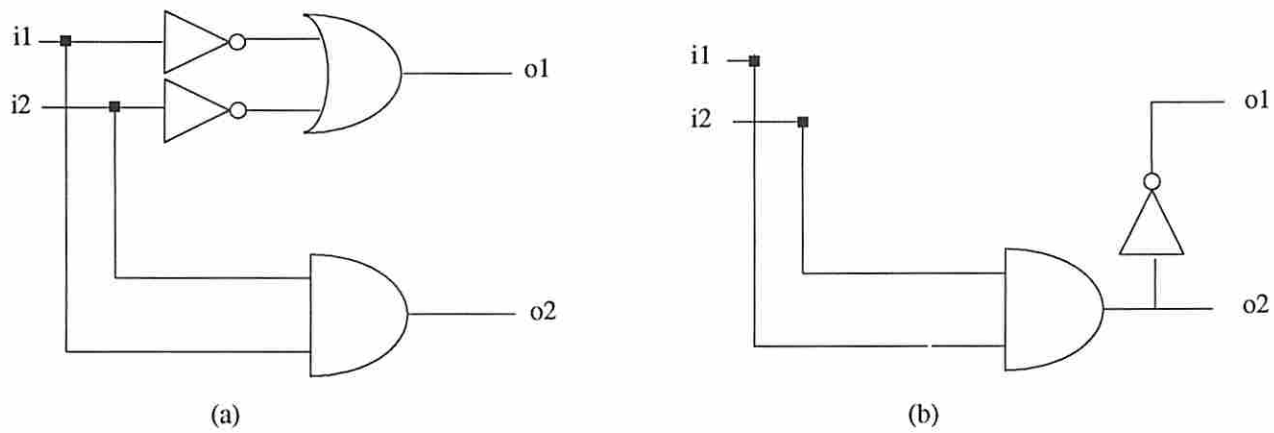


Figure 5: Local transformation procedure in the symbolic formula.

5.5 Symbolic Formulas for Complex Gates

We adopt a technique similar to that of [3]. Let f be a node. Let $p_1, \ldots, p_n$ be on-set primes of f and $q_1, \ldots, q_m$ be the on-set primes of $\overline{f}$. Let $g_1, \ldots, g_r$ be the inputs to gate f. Let $p_i^j \in \{0, 1, X\}$ be the logic value for input g_j in prime p_i. We define $g^X = 1$. Then the formulas for $g^1(t_1, t_2)$ and $g^0(t_1, t_2)$ are written as:

$$f^1(t_1, t_2) = \prod_{t \in MT_n(t1-max_f, t2-min_f)} \sum_{i=1}^{n} \prod_{j=1}^{r} g_j^{p_i^j}(t, t) \tag{5}$$

$$f^0(t_1, t_2) = \prod_{t \in MT_n(t1-max_f, t2-min_f)} \sum_{i=1}^{m} \prod_{j=1}^{r} g_j^{q_i^j}(t, t) \tag{6}$$

When the on-set of f or $\overline{f}$ consists of a single prime, such as the case for sensitizing values in the above discussion, further reduction on the symbolic formulas is possible, as demonstrated by Equation 2.

However, the above representation has one disadvantage. It is hard to detect local inversion implication in the symbolic formula, since the on-set of f and $\overline{f}$ are not in dual form. Therefore, we use the following representation.

$$f^1(t_1, t_2) = \prod_{t \in MT_n(t1-max_f, t2-min_f)} \sum_{i=1}^{n} \prod_{j=1}^{r} g_j^{p_i^j}(t, t) \tag{7}$$

$$f^0(t_1,t_2) = \prod_{t \in MT_n(t1-max_f,t2-min_f)} \prod_{i=1}^{n} \sum_{j=1}^{r} \overline{g_j^{p_i^j}}(t,t) \tag{8}$$

An additional advantage of this dual representation of f^1 and f^0 is that we can pick the more concise representation (in terms of CNF versus DNF) for f^1. Whatever the choice of f^1, f^0 will represented by its dual.

5.6 The degenerate Cases of XBD0 Model

If we restrict the lower delay bound to zero, the XBD model will degenerate to XBD0 model. As demonstrated in [3], timing analysis under XBD0 model in transition mode is equivalent to the floating mode timing analysis. A very important property of both is the *monotone waveform property* which states if at time $t > 0$ some node f assumes binary value, then at any time $t' \geq t$, f will assume the same binary value. If we apply our technique to XBD0 model, then every time interval $[t_1, t_2]$ where $t_1 > 0$ will consist of only right delay bounds. Thus we can pick the time instance that is at the beginning of interval. That is, our formulation automatically satisfies the monotone waveform property.

As for the FBPD model, the upper bound equals to the lower bound. There is no X in the transition waveform for any gate. At any time instance, node f will assume either one or zero. Our dual representation of the symbolic formula and the local duality transformation will also recognize such property.

6 Path Sensitization Criterion

The false path problem is an important issue in performance optimization of logic circuits. Next will solve the problem under the model.

For a path to be a true path, all side inputs to the gate f_k along the path (which is denoted by $SD(f_k)$) should satisfy the path sensitization criterion as described below.

Consider f_k on a path P. Assume partial path P_{k-1} is a true path, we derive boolean function that must evaluate to one in order for $P_k = P_{k-1} \bigcup \{f_k\}$ to be a true path. Let $d(P_{k-1})$ denote delay of path P_{k-1}. Therefore the output of node f_{k-1} will make a X to either controlling or sensitizing transition. In the first case, every input in $SD(f_k)$ must assume X or sensitizing value at time $d(P_{k-1})$. In the second case, every input in $SD(f_k)$ must assume sensitizing value at time $d(P_{k-1})$. Therefore for P_k to be a true path under the condition that P_{k-1} is a true path, the following equation must hold true:

$$\{c_{f_{k-1}}^{f_k}(max_{d(P_{k-1})}) \cdot \prod_{g \in SD(f_k)} \overline{c_g^{f_k}(max_{d(P_{k-1})}, max_{d(P_{k-1})})} \; + \\ s_{f_{k-1}}^{f_k}(max_{d(P_{k-1})}) \prod_{g \in SD(f_k)} s_g^{f_k}(max_{d(P_{k-1})}, max_{d(P_{k-1})})\} \quad (9)$$

The requirement for f_0 is different from the other nodes since it is a primary input, that is, $f0$ should assume opposite values under the first and second vectors:

$$f_0(v1)\overline{f_0(v2)} + \overline{f_0(v1)}f_0(v2)\} \quad (10)$$

Equation 9 and 10 give a recursive definition of true paths. The following equation gives a non-recursive definition:

$$\{f_0(v1)\overline{f_0(v2)} + \overline{f_0(v1)}f_0(v2)\} \cdot \{\prod_{i=1}^{k}\{c_{f_{i-1}}^{f_i}(max_{d(P_{k-1})}) \cdot \prod_{f \in SD(f_i)} \overline{c_f^{f_i}(max_{d(P_{i-1})}, max_{d(P_{i-1})})} \\ + s_{f_{i-1}}^{f_i}(max_{d(P_{k-1})}) \prod_{f \in SD(f_i)} s_f^{f_i}(max_{d(P_{i-1})}, max_{d(P_{i-1})})\}\} \quad (11)$$

7 Experimental Results

Circuit	longest topological delay	LB=0			LB=0.5			LB=0.8		
		total time	sat time	true delay	total time	sat time	true delay	total time	sat time	true delay
C880	40.8	4.0	0.1	40.8	6.2	0.1	40.8	6.4	0.1	40.8
C1335	36.2	5.7	0.1	36.2	8.9	0.1	36.2	9.3	0.1	36.2
C1908	49.7	699.8	476.2	47.9	874.2	470.1	47.9	865.0	468.7	47.9
C2670	35.0	5.3	0.1	35.0	6.7	0.1	35.0	6.6	0.1	35.0
C5315	59.0	98.1	38.1	58.8	136.4	37.2	58.8	138.5	37.4	58.8
C7552	81.5	36.7	0.2	81.5	68.5	0.2	81.5	67.3	0.2	81.5

Table 1: Experimental results on 6 ISCAS benchmarks.

We have implemented the proposed approach in C under the SIS environment. The gate delays are obtained after technology mapping. We assume symmetric rise and fall delays. Since there is no

delay variation information available, the lower bound is assumed to be a constant factor away from the upper bound. The tautology checker is a CNF-based satisfiability check procedure provided by SIS. [7]. Table 1 shows the results for ISCAS85 benchmarks under three different lower bound ratios (except C6288 for which, the memory requirementfor storing the delay bounds was too high, and C3540, where the tautology checker didn't finish after two hours of computation).

The LB is the lower bound ratio. When LB equals to zero, the XBD degenerate to the XBD0 model. The total time column gives the total run time, which includes the preprocessing time. The sat time column gives the amount of time spent on tautology checking. Both of them are in seconds. We use a linear search method to obtain the true delays[7]. The decremental step is 0.1ns, which is the resolution of the gate delays. In all these circuits, the true path delays are the same for three lower bounds.

8 Conclusions

In this paper, we have presented a symbolic simulation technique for performing transition mode timing analysis under the Extended Bounded Delay Model. Our procedure generates the minimum number of symbolic formulas at each internal node. These formulas are compactly stored in a symbolic network. The network is then processed to improve the performance of the tautology checking algorithm. The procedure has been implemented and the results are provided.

References

[1] P. McGeer and R. Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *Proceedings of the 26th Design Automation Conference*, pages 561–567, June 1989.

[2] H. Chen and D. H. C. Du. Path sensitization in critical path problem. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, pages 208–211, November 1991.

[3] P. McGeer, A. Saldanha, P. R. Stephan, and R. Brayton. Delay models and sensitization criteria in the false path problem. pages 76–83, 1992.

[7] For floating mode and transition mode under the XDB0 model, it is feasible to use a binary search method to find the delay. The main reason is that the monotone waveform property will keep the number of nodes in the symbolic formula within a acceptable range. If the lower bounds of the gate delays are very close to the upper bound, it will in general behave more like in FBPD model. It should use a linear search method. Here we use a linear search method for all three cases to give a fair comparison.

[4] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Certified timing verification and the transition delay of a logic circuit. In *Proceedings of the 29th Design Automation Conference*, pages 549–550, June 1992.

[5] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Delay computation in combinational circuits: Pratice and implementation. In *Proceedings of the 29th Design Automation Conference*, pages 68–75, June 1992.

[6] S. Devadas, K. Keutzer, and S. Malik. Delay computation in combinational circuits: Therory and algorithm. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, November 1991.

[7] T. Larrabee. Test pattern generation using boolean satisfiability. IEEE *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 4–15, January 1992.