A New Approach to Timing Analysis using Event Propagation and Temporal Logic

Arijit Mondal Partha. P. Chakrabarti C. R. Mandal

Dept. of CSE & Adv. VLSI Lab Dept. of CSE & School of IT

Indian Institute of Technology, Kharagpur, INDIA 721302

E-mail: {arijit,ppchak,chitta}@cse.iitkgp.ernet.in

Abstract

Present day designers require deep reasoning methods to analyze circuit timing. This includes analysis of effects of dynamic behavior (like glitches) on critical paths, simultaneous switching and identification of specific patterns and their timings. This paper proposes a novel approach that uses a combination of symbolic event propagation and temporal reasoning to extract timing properties of gate-level circuits. The formulation captures complex situations like trigerring of traditional false paths and simultaneous switching in a unified symbolic representation in addition to identifying false paths, critical paths as well as conditions for such situations. This information is then represented as an event-time graph. A simple temporal logic on events is proposed that can be used to formulate a wide class of useful queries for various input scenarios. These include maximum/minimum delays, transition times, duration of patterns, etc. An algorithm is developed that retrieves answers to such queries from the event-time graph. A complete BDD based implementation of this system has been made. Results on the ISCAS85 benchmarks indicate very interesting properties of these circuits.

1. Introduction

The estimation of timing behavior quickly and accurately is a challenging task due to the complexity of present day circuits. Timing analysis of a circuit traditionally involves estimating the critical delay of the circuit after eliminating false paths. In recent times static timing analysis (STA) has gained in popularity over dynamic simulation because of its speed and capability to handle large designs. However dynamic simulation can provide more detailed information about circuit behavior. This work presents a new approach using a combination of symbolic event propagation and formal logic to reason about circuit timing using a mixed approach that is capable of retrieving important details required in modern circuits while taking advantage of both methods.

We briefly discuss related work. The main focus in [2] lies in the determination of correct sensitization conditions for timing verification. [5] is based on the representation of conditional delay matrices (CDM) which combine module delays with event propagation conditions. A systematic analysis of delay computation based on a series of waveform models that capture signal behavior rigorously at different levels of detail is presented in [6]. The most general model, called the exact or W0 model, specifies each event occurring in a circuit signal. Algebraic Decision Diagram [1] based methods can be used to perform delay computation in combinational circuits. A model is proposed by Chen et. al. [3] to capture the delay phenomena associated with simultaneous to-controlling transitions.

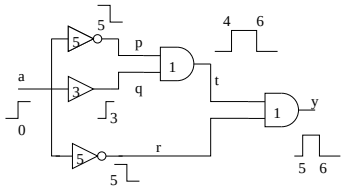

Figure 1. Example circuit where multiple transitions activate a false path

This work is motivated by the fact that existing tools normally fail to produce accurate results in the deep submicron region since they do not consider the impact of various situations relating to event propagation. For example, temporal proximity of two or more events *(simultaneous switching)* may have significant effect on the propagation time [3]. Most of the static timing analysis tools find out false paths and ignore them in further calculations. We have observed that multiple transitions at a node may activate a false path. Glitches (transition to high/low value for a small period) occur very often in circuits. In Figure 1, a circuit is shown where traditional methods report the path $\{a, r, y\}$ to be a false one (gate delays are shown inside the gate). However as shown in Figure 1, the glitch at node *t* sensitizes the path.

1530-1591/04 $20.00 (c) 2004 IEEE

The effects of such situations need to be considered to obtain more accurate results. Our approach here is to perform a combination of static and dynamic analysis, to enable deeper reasoning on timing behavior. Our method is capable of executing a combination of symbolic event/waveform propagation and temporal logic analysis based on user requirements. In order to provide a deeper insight on signal propagation in the circuit we need to have a versatile mechanism using which the designer can reason about waveforms and timings. For this we have developed a query language based on Temporal Logic in which one can express a timing query on the circuit under various scenarios. Algorithms to store timing information and retrieve query results efficiently are also proposed. Results on standard benchmarks (ISCAS85) are promising as we are able to retrieve some interesting information about these circuits not known earlier.

2. Mathematical formulation

In our formulation we will consider a gate level specification of the netlist. Gates have pin-to-pin delays. In building the mathematical model, we will assume the circuit is purely combinational without any feedback loop, only a single transition can occur in one of the primary inputs and all the transitions are instantaneous. In addition we will assume that the interconnect has no delay.

Given an input stimulus to the circuit, a change or transition in value at the circuit node is called an *event*. The time at which the event occurs is called *event time.* The entire sequence of events occurring at a node x (say) over the time period of interest is called the *waveform* of x. If an event appears on output line z of a gate in response to an event j on input line x, then event j is said to propagate to z. The logical condition under which this occurs is called the *propagation condition*(*PC*). A path that cannot be sensitized under any input transition is called an *unsensitizable* or a *false path*. The delay of a circuit is defined as the difference in time between an occurrence of an input transition and the time at which the output stabilizes. Because circuit delays are determined by events that propagate to the outputs, it is important that event propagation be accurately characterized. The *critical path* in a circuit is the longest sensitizable path. The *circuit delay* is the length of the critical path.

We will use a few notations in the following discussion as described below. $a, b, a_1, a_2, \ldots, a_n$ are Boolean variables.

$$ab \equiv a \wedge b, \;\; a+b \equiv a \vee b, \;\; a-b \equiv a \wedge \bar{b}$$

$$\bigcup_{i=1}^{n} a_i = a_1 \vee a_2 \vee \ldots \vee a_n \qquad \bigcap_{i=1}^{n} a_i = a_1 \wedge a_2 \wedge \ldots \wedge a_n$$

2.1. Event propagation

We will first consider a 2-input AND gate to explain our formulation. Later we will present the general formulation for any gate. Consider an AND gate as shown in Figure 2. Any event at node x will propagate to the output node z if and only if the other input y is *true*, hence the PC = y, i.e., the non-controlling value for AND gate is 1. (In case of an OR gate the non-controlling value will be 0. For an XOR gate any event in any input will propagate to the output, thus the propagation condition for this gate is 1. A true condition holds for the NOT and BUFFER gates.) If d be the delay of the gate then the event will occur at the output after d units of time. Similarly in Figure 2 the PC for any event to propagate from node y to the node z is x. For each event we will store some more information along with the propagation condition. Let σ denote the set of all possible events that can occur at a node. σ at a node x, denoted by σ_x, will be defined as follows

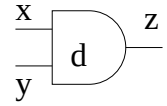

Figure 2. AND gate

$$\sigma_x = \left\{ (\lambda_x^i, t_x^i, v_x^i, tag_x^i, P_x^i) \right\} \qquad 0 \le i \le n_1$$

where,

λ_x^i : propagation condition (Boolean) that needs to be true for the event to occur
t_x^i : time at which the event occurs
v_x^i : value at node x after the ith event
tag_x^i : primary input where the transition has occured
P_x^i : previous nodes from which the event has propagated
n_1 : total number of possible events

The 0^{th} event is a special one. It describes the initial status of the node. For this event the *tag* and the *previous node* fields will be null. For the time being we will assume that no simultaneous switching takes place ie, mathematically $t_x^i \neq t_y^j \quad \forall i, j$. Let σ_z be the set of all possible events at node z in Figure 2. The ith event at node x will propagate to the output z with respect to the jth event at node y, if no event occurs at node y in the time interval between t_y^j and t_x^i and the value at node y has to be logic high (ie., v_y^j has to be true) and both ith and jth events have propagated to node x and y respectively for the same transition at the primary input (ie., both the events have the same tag). Let the propagated event at node z be the kth possible event, hence mathematically the propagation condition for it will be

$$\lambda_z^k = \lambda_x^i \left(\lambda_y^j - (\bigcup_{r=j+1}^{r=l} \lambda_y^r) \right) v_y^j \,,$$

where l is such that $t_y^l < t_x^i < t_y^{l+1}, \quad t_x^i \neq t_y^j \; \forall i, j$ For each such compatible j $(t_y^j < t_x^i)$, there will be a k. The *tag* for the k^{th} event at node z will be the same as the *tag* associated with the i^{th} event at node x. The *previous node* for the

k^{th} event at node z will be x, it means that the k^{th} event results from a single event at node x. The time at which the kth event at node z will occur is $(t_x^i + d)$.

Simultaneous switching: Using the previous notation, let us assume that the i^{th} event at node x happens simultaneously (in a very close interval of time - the length of this interval(ε) for a gate will be based on pin-to-pin delay values [3].) with the j^{th} event at node y. For an AND gate as in Figure 2, such an event can only propagate to the output if both the transitions are the same ie., both are either high transitions or low transitions. If one of them is a rise transition and the other is a fall transition then no event will propagate to the output. Hence mathematically

$$\lambda_z^k \quad = \quad \lambda_x^i \lambda_y^j (v_x^i v_y^j + \overline{v_x^i}\,\overline{v_y^j}\ )$$

The tag for the new event will be the same as the tag of the i^{th} event at node x or the tag for the j^{th} event at node y - this is because the events at node x and y have the same tag. The path for the k^{th} event will be (x,y). The time at which this event will occur is the delay of the gate due to simultaneous switching plus $(t_x^i + t_y^j)/2$. We will assume the delay value of a gate due to simultaneous switching is the average of pin-to-pin delays unless otherwise is specified.

Combining both: For a two input AND gate combining both the situations (single and multiple switching) we get the following PC

$$\lambda_z^k = (t_y^j <_\varepsilon t_x^i)\lambda_x^i \Big(\lambda_y^j - (\bigcup_{r=j+1}^{r=l} \lambda_y^r)\Big) v_y^j +$$

$$(t_x^i \equiv_\varepsilon t_y^j)\lambda_x^i \lambda_y^j (v_x^i v_y^j + \overline{v_x^i}\,\overline{v_y^j})$$

where l is such that $t_y^l < t_x^i < t_y^{l+1}$. $<_\varepsilon$ returns true if $t_y^j < (t_x^i - \varepsilon)$ holds and $\equiv_\varepsilon$ returns the truth of the temporal proximity of the ith and the jth events (ie., if $(t_x^i - \varepsilon) \le t_y^j \le (t_x^i + \varepsilon)$ holds). The *path* will be either (x) or (x,y) depending on whether a multiple switching occurs or not. The event time will be determined accordingly.

2.2. General formulation

Let us consider an arbitrary gate having n inputs and a single output. Let f be the functionality of the gate. Let $i_1, i_2, \cdots, i_n$ be the n inputs and y be the output ie., $y = f(i_1, i_2, \ldots, i_n)$. We want to find out the propagation condition for the jth event at node i_k with respect to the $r_m{}^{th}$ event at node i_m where $m = 1,2,\ldots,k-1,k+1,\ldots,n$. Let all these events be propagated from the same primary input x (say). Let the value of the kth node change from $\overline{i_k}$ to i_k after the occurrence of the jth event. The two values at the output, after and before the event, will be denoted by f_{i_k} and $f_{\overline{i_k}}$, where $f_{i_k} = f(i_1, i_2, \ldots, i_{k-1}, i_k, i_{k+1}, \ldots, i_n)$ and $f_{\overline{i_k}} = f(i_1, i_2, \ldots, i_{k-1}, \overline{i_k}, i_{k+1}, \ldots, i_n)$.

The condition for the said event to propagate to the output will be determined by the propagation condition of the said event and the other events at the other nodes after which the said event occurs and by the values of the all the nodes. We introduce a notation F which denotes the logical condition that needs to be satisfied arising from the values of different nodes. F_{p_1} denotes the changes in one input node, $F_{p_1 p_2 \cdots p_s}$ denotes the simultaneous changes in s input nodes where $p_i \in \{i_1, i_2, \ldots, i_n\}$. An event will propagate to the output if the transition at the input causes a transition at the output ie., the value of the output node will change after the occurrence of the event. Hence

$$F_{i_k} \quad = \quad f_{i_k}\,\overline{f_{\overline{i_k}}} + \overline{f_{i_k}}\,f_{\overline{i_k}}$$

Similarly, for simultaneous changes in s inputs,

$$F_{p_1 p_2 \cdots p_s} \quad = \quad f_{p_1 p_2 \cdots p_s}\,\overline{f}_{\overline{p_1}\overline{p_2}\cdots\overline{p_s}} + \overline{f}_{p_1 p_2 \cdots p_s}\,f_{\overline{p_1}\overline{p_2}\cdots\overline{p_s}}$$

In case of two input (i_1, i_2) AND gate the condition for an event in node i_1 to propagate is

$$F_{i_1} = i_1 i_2\,\overline{i_1 i_2} + \overline{i_1 i_2}\,\overline{i_1} i_2 = i_1 i_2 (i_1 + \overline{i_2}) + \overline{i_1} i_2 (\overline{i_1} + \overline{i_2}) = i_2$$

ie, the other input has to be true. If two events occur simultaneously at the nodes i_1, i_2, of an AND gate the event will propagate to the output if the following condition holds:

$$F_{i_1 i_2} \quad = \quad f_{i_1 i_2}\,\overline{f}_{\overline{i_1}\overline{i_2}} + \overline{f}_{i_1 i_2}\,f_{\overline{i_1}\overline{i_2}} = i_1 i_2 + \overline{i_1}\,\overline{i_2}$$

That is if both the transitions are the same only then will the event propagate.

Since, the jth event in the kth node occurs exactly after the $r_m{}^{th}$ event of the $i_m{}^{th}$ node where $m = 1,2,\ldots,k-1,k+1,\ldots,n$. Let us assume that among the n events s number of events (including the jth event at the kth node) occur simultaneously. Let the propagated event at the output y be the hth possible event at that node. Hence the propagation condition for the hth event at the output will be

$$\lambda_y^h = F_{p_1 p_2 \cdots p_s} \bigcap_{m=1}^{m=n} \Big(\lambda_{i_m}^{r_m} - (\bigcup_{q=r_m+1}^{l_m} \lambda_{i_m}^q)\Big)$$

where l_m is such that $t_{i_m}^{l_m} \le t_{i_k}^j < t_{i_m}^{l_m+1}, \quad \forall m = 1,2,\ldots n$ and the value will be $f(v_{i_1}^{r_1}, v_{i_2}^{r_2}, \ldots, v_{i_n}^{r_n})$ and the event time will be delay of the gate due to simultaneous switching at s nodes plus $(\sum_{i=1}^{i=s} t_{p_i}^{r_m})/s$.

After performing the previous calculation we will get the possible events at a node in the form of (λ, t, v, x, P). In order to express v in terms of x, the four mutually exclusive and exhaustive situations are listed in Table 1. In the table v_{new} denotes how the output follows the input. If the condition $xv + \overline{x}\overline{v}$ holds then the output will follow (x) the input otherwise the reverse transition $(\overline{x})$ will occur at the output. Now the possible event will be broken into two possible events of the form $(\lambda(xv + \overline{x}\overline{v}), t, x, x, P)$ and $(\lambda(\overline{x}v + x\overline{v}), t, \overline{x}, x, P)$. Finally, in any node all those events will be clubbed together which have the same *value* and *time*. The propagation condition will be OR-ed and *tag, previous node, values* will be remain unchanged.

It may be observed that we compute information for all possible single input changes for various input conditions symbolically using Binary Decision Diagrams

x	v	v_{new}
0	0	x
1	0	$\bar{x}$
0	1	$\bar{x}$
1	1	x

Table 1. Possible situations

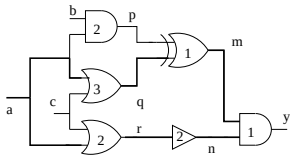

Figure 3. Example circuit for simultaneous switching

(BDDs) without having to run different computations for different input changes and input conditions.

Two examples are presented here. The first example shows how simultaneous switching behavior is captured in the formulation and the second one describes how consideration of glitches and subsequent activation of traditional false paths can be detected. In the first example we will assume the delay value for the gate in case of simultaneous switching is 80% of the normal gate delay.

Example 2.1 Consider the example in Figure 3. We will consider only the paths from primary input a to primary output y. Now we will determine the σ at various nodes. The possible events which have the propagation condition 0 (false) are not shown here.

$$\begin{aligned}
\sigma_p &= \{\ (1,0,\bar{a}b,null,null),\quad (b,2,a,a,a)\ \}\\
\sigma_q &= \{\ (1,0,\bar{a}+\underline{c},null,null),\quad (\bar{c},3,a,a,a)\ \}\\
\sigma_m &= \{\ (1,0,ac+\bar{a}b,null,null),\quad (ab\bar{c},3,a,a,p),\\
&\quad (b(\bar{a}+a\bar{c}),3,\bar{a},a,p),\quad (\bar{c}(\bar{a}+\bar{b}),4,a,a,q),\\
&\quad (ab\bar{c},4,\bar{a},a,q)\}\\
\sigma_r &= \{\ (1,0,\bar{a}+c,null,null),\quad (\bar{c},2,a,a,a)\ \}\\
\sigma_n &= \{\ (1,0,\bar{a}+c,null,null),\quad (\bar{c},4,a,a,r)\ \}\\
\sigma_y &= \{\ (1,0,\bar{a}b+ac,null,null),\quad (\bar{a}b,4,\bar{a},a,m),\\
&\quad (\bar{c}(\bar{a}+\bar{b}),4.8,a,a,(m,n))\}
\end{aligned}$$

Now one can tell not only the critical delay but also the conditions under which the critical path will be sensitized using the propagation condition associated with that event. The critical paths are highlighted in Figure 3. Well known methods like [5] estimate the critical delay as 5 in this case.

Example 2.2 Consider the circuit in Figure 1. The possible events at the output node will be

$$\sigma_y = \{(1,0,0,null,null),(a,5,a,a,t),(a,6,\bar{a},a,r)\}$$

Thus the critical delay for this circuit is 6 and this event will occur if there is a rise transition in the primary input as the propagation condition associated with this is a. In this case traditional methods report the critical delay as 5. Reporting lower critical delay is possibly more serious than overestimating it.

3. Reasoning about timing behavior

After performing the previous analysis, the designer may want to reason about the timing behavior of the circuit to have a deeper insight into the circuit dynamics. A user query may be related to the occurrence of an event or a series of events, the best/worst case timing information about the events, the duration between the events, etc. The user may even want to know the timing behavior of the circuit like - minimum delay, maximum delay, time at which the last glitch occurs, the maximum duration of a glitch or the maximum gap between two similar transitions under various input scenarios. For this we propose a query language based on Temporal Logic [4] which can capture various queries. We have developed a graph representation of the possible events at a node as a data structure called event-time graph and an algorithm to retrieve answers to the queries.

3.1. The event-time graph

We explain the event-time graph through an example. Let us consider the event-time graph as shown in Figure 4. The graph is generated using the σ (the set of all possible events) at a node by clubbing together those events which have same *value* and *time* irrespective of their *previous node* field. The new value of σ (containing *propagation condition*, *event time*, *value* and *tag*) is shown in Figure 4. The event-time graph can be partitioned into three parts namely $\mathbf{a}$, $\bar{\mathbf{a}}$ and $\mathbf{I}_0$ and these denote the set of direct follower events and inverse follower events of the primary input event and the initial state respectively. The node $N1$ in the graph denotes the initial state of the node and rest of the nodes in the graph denote the possible events. The nodes are sorted in terms of their time of occurrence. In the graph, we have shown all possible paths. Some of them may not exist for a node in a circuit. The node N_i can have a direct edge to node N_j if and only if the time of occurrence of the ith event is earlier than the jth event and the two nodes belong to the different partitions and $\lambda_i\lambda_j(\bigcap_{k=i+1}^{k=j-1}\bar{\lambda}_k) \neq 0$. The activation of a path in the graph depends on the λ values. There is no link between node $N2$ and $N4$ as both the nodes have same value a. This is because two similar transitions cannot occur at a circuit node consecutively unless the node undergoes a reverse transition in between. (Note that $\lambda_1\lambda_3 = \lambda_3\lambda_5 = \lambda_2\lambda_4 = \lambda_4\lambda_6 = 0$). For example to activate the path $N1-N3-N4-N7$ the condition $\lambda_2\lambda_3\lambda_6$ should hold and to activate the path $N1-N4-N7$ the condition $\lambda_3\lambda_6\lambda_2$ should hold. So we have a tripartite graph. We traverse this graph to extract answers to user queries.

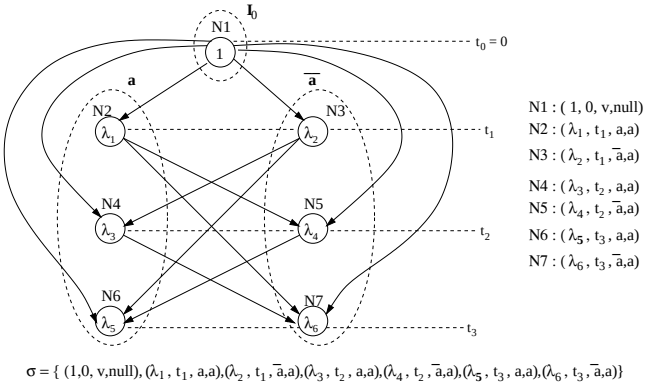

Figure 4. Graph structure for all possible waveforms at a node

3.2. Query language:

Let S be the scenarios under which the query will be evaluated, Q be the cost function over situation which can be *max* or *min*. C, C' be the cost function over the event which can be *max*, *min* or *val*. $0, 1$ denotes the logic value at a node in the circuit. x represets some logic value at a circuit node i.e., $0/1$. A node in the circuit will be referred by its name. U is the until operator [4] of Temporal Logic.

A scenario can be specified by the values at the different inputs and the inputs line where a transition can occur. The value at an input line can be $0, 1$ or x, where x denotes that it can be either 0 or 1. Consider the circuit in Figure 5. To explain our query language in a simplistic manner let us consider the following scenarios $S_1 = [a:1; b:1; c:1; d:0; e:1;]\,[a;]$, $S_2 = [a:1; b:1; c:0; d:1; e:0;]\,[a;]$ and $S_3 = [a:x; b:1; c:0; d:x; e:0;]\,[a;]$.

The scenario S_1 denotes a high transition at node a and the rest of the primary inputs (b, c, d, e) will be at $1, 1, 0, 1$ respectively. Similarly for S_2. The scenarios captured by S_3 is that any transition (high/low) can occur in the node a and the primary input lines b, c, e will be at $1, 0, 1$ respectively and the node d will be at either 0 or 1. The waveforms are shown for scenarios S_1 and S_2 in Figure 5.

Now the syntax for our query (F) :

$$
\begin{array}{rcl}
F & : & [S]_{Q(t)}\ C(t_1)\,[f] \mid [S]_{Q(t)}\ C'(t_1,t_2)\,[f] \\
f & : & (term\ U\ term) \mid (term\ U\ C(t_1)\ f) \\
 & & \mid (term\ U\ C'(t_1,t_2)\ f) \\
term & : & node.value \mid \neg\, node.value \\
node & : & y \mid Y \\
value & : & 0 \mid 1 \mid x
\end{array}
$$

3.3. Examples and informal semantics

Here we will provide an informal explanation of the semantics of our query language using examples. All the examples will be evaluated under the scenarios S_1, S_2 and S_3. The pin-to-pin delays are given in Figure 6. We will assume that the average of pin-to-pin delays as the delay for simultaneous switching for the gate. Note that, for S_2 simultaneous switching occurs at nodes $w3$ and $w6$.

Example 3.1 Determine the time of occurrence of the first event at node *y (min delay)*

$$[S]_{min(t)}\ \ min(t_1)\ \ [(\ y.x\,U\,\neg y.x\ )]$$

In the query $[(\ y.x\,U\,\neg y.x\ )]$ means at node y, x will be true until $\neg x$ holds which means a transition (high/low) at the node. The function $min(t_1)$ denotes the minimum value of all possible t_1's and t_1 represents the time of transition associated with the U operator. The function $min(t)$ associated with the scenario denotes that we will take the minimum value of $min(t_1)$ over all the situations specified by S. For S_1 the min delay is 4.66, for S_2 the answer is 4.72 and for S_3 the value will be 4.66.

Example 3.2 Find out the time of occurrence of the last event at node *y (max delay)*

$$[S]_{max(t)}\ \ max(t_1)\ \ [(\ y.x\,U\,\neg y.x\ )]$$

This is similar to the previous one except for the functions. Here we are interested in the last event at node y over some situations. The max delay for S_1, S_2 and S_3 will be 4.98, 4.72 and 4.82 respectively.

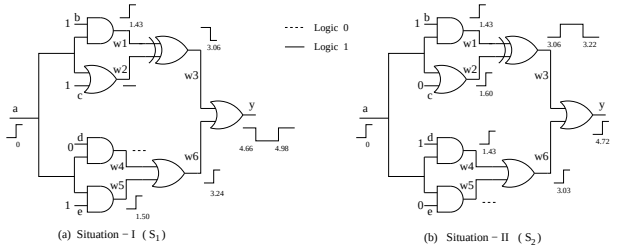

Figure 5. Example circuit for query

Example 3.3 Determine the time of occurrence of the last glitch

$$[S]_{max(t)}\ \ max(t_1)\ \ [(\ y.x\ U\ \ val(t_1)\ \ (\ \neg y.x\,U\,y.x\ ))]$$

In this case, we express the occurrence of two events consecutively at node y. The node y undergoes a series of transitions of the form $x - \overline{x} - x$ ie., either $1-0-1$ or $0-1-0$. The $val(t_1)$ function returns the value at which the event (event represented by the subformula) has occurred and $max(t_1)$ returns the maximum value of the start time for the series of transitions (the argument t_1 in *max* function is associated with the first U operator of the query formula). For S_1 the time at which last glitch occurs is 4.66. For S_2, no such glitch occurs and we return *null*. For S_3 the value will be 4.66.

Example 3.4 Determine the maximum timing gap between two successive changes at node y

$$[S]_{max(t)}\ \ max(t_2 - t_1)\ [(\ y.x\,U\ val(t_1)\ (\ \neg y.x\,U\,y.x\ ))]$$

Here we are interested in the duration of the glitch. In the function $max(t_2 - t_1)$, the argument t_2 is related with the value returned by the subformula $(\neg y.x U y.x)$ and t_1 is related with the U operator in the top level of the formula. For S_1 the value of this query is 0.23 and for S_2 the value will be *null* and for S_3 the answer is 0.16.

Example 3.5 Find out the maximum timing gap between two successive similar transitions at node y

$$[S]_{max(t)}\ max(t_2 - t_1)\ [(y.x\ U\ val(t_2)\ (\neg y.x\ U\ val(t_1)\ (y.x\ U\ \neg y.x))]$$

This query represents a series of transition of the form $1-0-1-0$ or $0-1-0-1$. In this case the function $val(t_2)$ returns the value returned by the inner most $val(t_1)$ function since the argument t_2 is associated the subformula. The function $max(t_2 - t_1)$ returns the maximum value for such situations. Finally, over all the situations the maximum value is taken. For all the situations the value will be *null* since at the output there are no two similar transitions.

AND2		AND3		AND4		AND5		AND6	
Y	A 1.50	Y	A 1.53	Y	A 1.56	Y	A 1.64	Y	A 1.68
Y	B 1.43	Y	B 1.59	Y	B 1.53	Y	B 1.63	Y	B 1.67
NAND6			C 1.42	Y	C 1.48	Y	C 1.69	Y	C 1.65
Y	A 1.34	NAND5		Y	D 1.42	Y	D 1.55	Y	D 1.62
Y	B 1.37	Y	A 1.36	NAND4		Y	E 1.59	Y	E 1.58
Y	C 1.49	Y	B 1.38	Y	A 1.43	NAND3		Y	F 1.53
Y	D 1.42	Y	C 1.41	Y	B 1.45	Y	A 1.52	NAND2	
Y	E 1.44	Y	D 1.43	Y	C 1.47	Y	B 1.54	Y	A 1.64
Y	F 1.46	Y	E 1.45	Y	D 1.59	Y	C 1.56	Y	B 1.65

OR3		NOR4		OR2					
Y	A 1.79	Y	A 1.65	Y	A 1.60				
Y	B 1.54	Y	B 1.61	Y	B 1.74				
Y	C 1.80	Y	C 1.55	OR4					
NOR2		Y	D 1.48	Y	A 1.56				
Y	A 1.71	NOR3		Y	B 1.72				
Y	B 1.54	Y	A 1.70	Y	C 1.84				
BUF		Y	B 1.61	Y	D 1.93				
Y	A 1.59	Y	C 1.59	XOR2					
		INV		Y	A 1.55				
		Y	A 1.81	Y	B 1.61				

Figure 6. Delay table (values in ns)

4. Results

We have implemented a tool using our proposed method. This tool is capable of performing timing analysis of circuits and can answer various queries. We have taken results using the ISCAS85 benchmarks using the five queries in Examples 3.1 to 3.5. In order to have realistic values for gate delays of modern circuits, we have used the table in Figure 6. We have assumed a fixed delay for rise and fall transitions and both the delays are same. Since the delay values for simultaneous switching are not available, the average of pin-to-pin delays are taken as the delay for the simultaneous switching. Each query was evaluated for fifty different randomly generated scenarios and in each scenario twenty of the input lines were randomly kept at x while others were made 0 or 1 and finally over all the scenarios an appropriate *min* or *max* value is taken depending on the cases. (For the query in example 3.1 we take the minimum over all the scenarios and for rest of the queries we take the maximum over all the scenarios.) We generated the event-time graph based on the symbolic timing analysis approach of Section 2 and then executed the queries on that graph. The results are listed in Table 2. The memory requirement for evaluation of a query is significantly dominated by the memory usage of the BDD package. The results show that the occurrence of more than one event is very common ie., the node undergoes various transitions before the stabilization even if there is only a single transition in the primary input. Interestingly these are not known to occur for unit delay models.

		Ex-3.1	Ex-3.2	Ex-3.3	Ex-3.4	Ex-3.5
C17	(q)	3.28	4.95	3.29	1.65	NULL
	(t)	0	0	0	0	0
C432	(q)	6.11	31.40	29.96	19.96	19.92
	(t)	79.65	79.83	79.37	79.24	79.43
C499	(q)	1.63	18.04	17.98	11.39	13.17
	(t)	14.31	14.37	14.87	14.87	15.22
C880	(q)	3.01	34.08	26.96	7.66	NULL
	(t)	4.33	4.35	4.56	4.53	4.68
C1355	(q)	4.87	37.76	34.52	27.83	26.13
	(t)	18.85	18.71	19.59	19.48	20.23
C1908	(q)	4.88	61.30	59.33	46.92	45.56
	(t)	20.18	19.81	20.63	20.61	21.21
C3540	(q)	9.29	67.36	65.56	33.16	35.06
	(t)	12.83	12.91	13.17	13.11	13.29
C5315	(q)	3.18	6.41	NULL	NULL	NULL
	(t)	0.24	0.23	0.24	0.25	0.26

(q) - query output (in nano-seconds)

(t) - average execution time per query (in seconds)

Table 2. Results on ISCAS85 benchmark circuit

Acknowledgements

Dr. P.P. Chakrabarti acknowledges Dept. of Science & Technology, Govt. of India for partial support of this work.

References

[1] R. I. Bahar, H. Cho, G. D. Hatchel, E. Macii, and F. Somensi. Timing analysis of combinational circuits using add's. *IEEE European Conference on Design Automation*, pages 625–629, 1994.

[2] H. C. Chen and D. H. C. Du. Path sensitization in critical path problem. *International Conference for Computer-Aided Design of VLSI Circuit*, pages 208–211, 1991.

[3] L. C. Chen, S. K. Gupta, and M. A. Breuer. A new gate delay model for simultaneous switching and its application. *Proceedings of Design Automation Conference*, pages 289–294, 2001.

[4] E. M. Clarke, J. O. Grumberg, and A. P. Doron. *Model Checking*. The MIT Press, Cambridge, Massachusetts, London, England, 2001.

[5] H. Yalcin and J. P. Hayes. Hierarchical timing analysis using conditional delays. *ICCAD, Digest of Technical Papers, San Jose, California*, pages 371–377, 1995.

[6] H. Yalcin and J. P. Hayes. Event propagation condition in circuit delay computation. *ACM Transactions on Design Automation of Electronic Systems*, 2(3):249–280, July 1997.