

A Timed Automaton-Based Method for Accurate Computation of Circuit Delay in the Presence of Cross-Talk

S. Taşiran¹ *, S. P. Khatri¹ *, S. Yovine² **, R. K. Brayton¹, and A. Sangiovanni-Vincentelli¹

¹ Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley

² VERIMAG, and
California-PATH, University of California at Berkeley

Abstract. We present a timed automaton-based method for accurate computation of the delays of combinational circuits. In our method, circuits are represented as networks of timed automata, one per circuit element. The state space of the network represents the evolution of the circuit over time and delay is computed by performing a symbolic traversal of this state space.

Based on the topological structure of the circuit, a partitioning of the network and a corresponding conjunctively decomposed OBDD representation of the state space is derived. The delay computation algorithm operates on this decomposed representation and, on a class of circuits, obtains performance orders of magnitude better than a non-specialized traversal algorithm.

We demonstrate the use of timed automata for accurate modeling of gate delay and cross-talk. We introduce a gate delay model which accurately represents transistor level delays. We also construct a timed automaton that models delay variations due to cross-talk for two capacitively coupled wires.

On a benchmark circuit, our algorithm achieves accuracy very close to that of a transistor level circuit simulator. We show that our algorithm is a powerful and accurate timing analyzer, with a cost significantly lower than transistor level circuit simulators, and an accuracy much higher than that of traditional timing analysis methods.

1 Introduction

The computation of delay for combinational circuits is a well-studied problem which, until recently, was considered solved. Efficient exact methods have been devised for computing the delays of acyclic combinational circuits ([DKM93], [LB94], [MSBS93], [YH95] among others). However, with the feature sizes of integrated circuits shrinking to sub-micron levels, some of the underlying assumptions of existing delay analysis methods are no longer valid. First, higher clock speeds require more accurate modeling of circuit delay, therefore, more

* Supported by SRC under grants DC-324-026 and DC-324-040.

** Supported by NSF under grant ECS 972514

sophisticated gate delay models are needed. For instance, for complex dynamic logic gates, gate delays depend on the relative timing of the input signals, as well as their values. Second, circuit level effects such as capacitive coupling between wires (also referred to as *cross-talk*) need to be taken into account. As a result, it is now commonly accepted that existing methods for modeling and computing delays are inadequate for deep sub-micron circuits (See, for instance, [CWS97,TKB97] and other related papers in the TAU '97 Workshop). Existing timing analysis schemes do not use sufficiently accurate gate delay models, and do not account for cross-talk.

Timed automata have been used to model the delay characteristics of gates and circuits [MP95,TAKB96,TB97,FK95]. Previously, timed automaton-based techniques have been restricted to the analysis of asynchronous circuits because they are generally smaller than synchronous circuits and require a more detailed temporal analysis than delay computation. Today's synchronous circuits with sub-micron feature sizes place stronger demands on timing analysis tools, and this new setting makes the expressiveness of timed automata desirable. Timed automata allow the use of very general gate delay models - including those in which a unique delay can be specified for every pair of input vectors. As a result, effects of cross-talk on delay can be incorporated. One contribution of this paper is the introduction of a new and accurate gate delay model, which efficiently accounts for input sequence dependent delays.

For large portions of a typical design, a less powerful modeling framework suffices. However, the analysis of certain critical portions of a high performance design must be performed with great accuracy, which is currently provided only by transistor level simulators. Exhaustive simulation of these sections of the circuit, on the other hand, is not computationally feasible because of the exponential number of possible input patterns. In this study, we present a timed automaton-based delay computation method. We represent input waveforms and circuits by networks of timed automata, with each automaton modeling a circuit element or input waveform. The state space of this network describes the possible behaviors of the circuit over time. Delay computation is then posed as a variant of the state-space traversal problem. Because all input patterns are covered, unlike simulation, the circuit delay computed by this algorithm is guaranteed to be correct.

State-space traversal of timed automata is a PSPACE-complete problem, and, like other traversal-based methods, it suffers from the state-space explosion problem for systems with large numbers of components. However, by exploiting certain topological properties of combinational circuits and the state spaces of the corresponding timed automata, we devise heuristics which enable us to handle systems that are much larger than could be handled with non-specialized traversal methods (Section 5.2).

The paper is organized as follows. In Section 2 we introduce timed automata. Section 3 discusses how circuits and waveforms are represented using timed automata. Section 4 describes our algorithm for computing maximum delays. Section 5 presents experimental results and contrasts our method with other ap-

proaches for delay computation. Section 6 summarizes our work, and discusses avenues for future research.

2 Timed automata

2.1 Notation

Let X be a finite set of real-valued variables. An X -valuation ν assigns a non-negative real value $\nu(x)$ to each variable $x \in X$. An X -predicate φ is a positive Boolean combination of constraints of the form $x \diamond k$, where k is a nonnegative integer constant, $x \in X$ is a variable, and \diamond is one of the following: \leq , \geq , $=$. Let P be a finite set of variables, each ranging over a finite type. A P -valuation ξ is an assignment of values to variables in P . A P -event is a pair $\langle \xi, \xi' \rangle$ consisting of P -valuations ξ and ξ' denoting the old and the new values of the variables in P . A P -predicate is a Boolean predicate on ξ and ξ' .

2.2 Timed Automata

A *timed automaton* A is a tuple $\langle S, S_0, O, I, X, \alpha, \mu, E \rangle$, where

- S is the finite set of locations, and $S_0 \subseteq S$ is the set of initial locations.
- O is the set of output variables, each ranging over a finite domain. An *output* of A is an O -valuation.
- I is the set of input variables, each ranging over a finite type. An *input* of A is an I -valuation, an *input-event* is an I -event, and an *input-predicate* is an I -predicate. An *observation* of A is a $(I \cup O)$ -valuation, and an *observation-event* of A is a $(I \cup O)$ -event.
- X is the finite set of real-valued variables, called *timers*.
- α is the invariant function that assigns an X -predicate $\alpha(s)$ to each location $s \in S$.
- μ is the output function that assigns the output $\mu(s)$ to each location $s \in S$.
- E is the finite set of edges. Each edge e is a tuple $\langle s, t, \varphi, \chi, R \rangle$ consisting of the source location s , the target location t , the X -predicate φ , the input-predicate χ , and a subset of clocks $R \subseteq X$ that specifies the timers to be reset to 0 after the edge is taken.

A *state* σ of A is a pair $\langle s, \nu \rangle$ containing the location $s \in S$ and the X -valuation $\nu \in \alpha(s)$. The set of all states of A is denoted by Σ_A . The state $\langle s, \nu \rangle$ is *initial* if $s \in S_0$ and $\nu(x) = 0$ for all $x \in X$. Consider a state $\sigma = \langle s, \nu \rangle$ of the timed automaton A and a time increment δ . The automaton A can *wait* for δ in state σ , written $wait(\sigma, \delta)$, iff for all $0 \leq \delta' \leq \delta$, $(\nu + \delta') \models \alpha(s)$. A *timed event* γ of the timed automaton A is a tuple $\langle \delta, \xi, \xi' \rangle$ consisting of a non-negative real-valued increment δ and the observation-event $\langle \xi, \xi' \rangle$. Such an event means that the automaton can wait for the time period δ if $\delta > 0$ and then update its output variables from $\xi(O)$ to $\xi'(O)$ while the environment is updating the input variables from $\xi(I)$ to $\xi'(I)$. The set of all timed events of A is denoted Γ_A .

The timed automaton A corresponds to a labeled transition system over the state-space Σ_A with labels from Γ_A . For states $\sigma = \langle s, \nu \rangle$ and $\tau = \langle t, \mu \rangle$ in Σ_A , and a timed event $\gamma = \langle \delta, \xi, \xi' \rangle$ in Γ_A , define $\sigma \xrightarrow{\gamma} \tau$ iff $\xi(O) = \mu(s)$, $\xi'(O) = \mu(t)$, $wait(\sigma, \delta)$, and there exists an edge $\langle s, t, \varphi, \chi, R \rangle$ such that $(\nu + \delta) \models \varphi$, $\langle \xi, \xi' \rangle \models$

χ , and μ is obtained from ν by applying R as explained above. A *timed event sequence* $\bar{\gamma} = \gamma_0, \gamma_1, \dots, \gamma_{k-1}$ is a finite sequence of timed events $\gamma_i = \langle \delta_i, \xi_i, \xi_i' \rangle$ such that $\xi_{i+1} = \xi_i'$ for $0 \leq i < k - 1$. A *run* of A on a timed event sequence $\bar{\gamma}$ is a sequence of states $\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_k$ such that $\sigma_0 \xrightarrow{\gamma_0} \sigma_1 \xrightarrow{\gamma_1} \sigma_2 \xrightarrow{\gamma_2} \dots \xrightarrow{\gamma_{k-1}} \sigma_k$ in A . The timed event sequence $\bar{\gamma}$ is called a *trace* of A if there exists a run in A on $\bar{\gamma}$ starting from an initial state.

Composition of Timed Automata A timed automaton representing a circuit is obtained by *composing* the timed automata representing each component. The composition operation is analogous to that for FSMs: the product of the discrete state spaces is taken, and the set of timer variables for the composition consists of the union of the timers of the components. The composition of timed automata A and B is denoted by $A||B$. For a formal definition of composition for timed automata, please see [TAKB96].

3 Modeling Circuits and Waveforms

3.1 Modeling Sets of Waveforms

The maximum delays of combinational circuits are typically computed for the following two cases:

- *Floating-mode delay*: Inputs are allowed to change their values arbitrarily until time 0. After time 0 all inputs remain stable.
- *Two vector delay*: Inputs (and all intermediate nodes of the circuit) are stable until time 0. At time 0 the inputs may switch to new values and must remain stable thereafter.

The sets of input waveforms for both cases can be represented concisely by timed automata. Figure 1 shows the timed automaton for the two-vector delay model. \mathbf{i}_{old} and \mathbf{i}_{new} represent the vectors of primary inputs before and after time 0 ($t = 0$) respectively. \mathbf{i}_{old} and \mathbf{i}_{new} are selected non-deterministically, which enables the automaton in the figure to represent all input vector pairs.

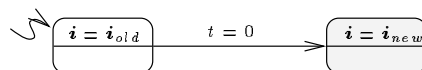


Fig. 1. Input Vector Automaton (Two-vector Delay Model)

It is also straightforward to model different arrival times at different primary inputs, asynchronous inputs, etc., with timed automata using extra timers.

3.2 Modeling Combinational Gates

Timed automata offer a great deal of expressive power for representing gate delay models. In the past, they have been used to model gates [MP95], [TAKB96], [TB97], [FK95] in the verification of asynchronous circuits and library elements. For example, in [TB97], a timed automaton representations for the inertial delay model was described. This model accounts for the possibility that short input pulses may not be reflected at a gate's output.

The gate delay models employed by conventional timing analysis methods have been somewhat simplistic and ad-hoc. The most primitive of these is the

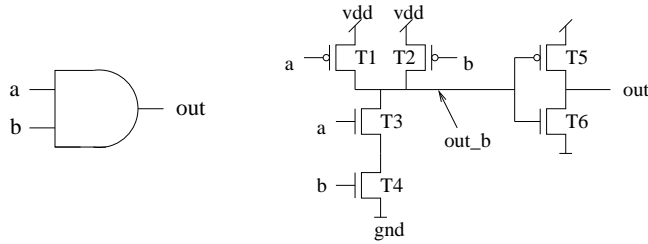


Fig. 2. AND gate: Transistor Level Description

unit delay model, where each gate is assumed to have a delay of one time unit. This was supplanted by the *fixed* delay model, where gates can have different but constant delays. Both of these models are too coarse as they do not allow for variations in gate delay. The *min-max* delay model addresses this problem to some degree by allowing variations in the delay of a gate: transitions at a gate's input are reflected at its output with a delay in the range $[d_{min}, d_{max}]$. In the *pin-delay* model, different input-to-output delays can be specified for different input pins of a gate. Neither the min-max model nor the pin-delay model are powerful enough for an accurate analysis because they can not express the dependency of gate delay on specific input values.

In practice gate delays have a strong dependence on the input vectors applied. This is illustrated by means of the static CMOS AND gate shown in Figure 2. For this gate, under the input sequence $a = b = 1 \rightarrow a = 1, b = 0$, let the gate delay be $t_{11 \rightarrow 10}^f$. The superscript f (r) indicates that the output is falling (rising), and the subscript represents the applied input vector sequence. Similarly, under the input sequence $a = b = 1 \rightarrow a = 0, b = 0$, let the gate delay be $t_{11 \rightarrow 00}^f$. In the second case, the delay of the gate is smaller. This is because, in the latter case, after the inputs transition, both transistors T1 and T2 are on, effectively doubling the current to charge the capacitance of node out_b. This results in a faster falling transition. In the first case, only transistor T2 is on after input b has switched, hence the falling transition is slower. The pin delay model does not distinguish between these two input sequences, and hence is not accurate enough for high performance designs. Some delay models used in the past allow the specification of separate rising and falling delays for the gate output. This is a useful property to model, but in the absence of input sequence dependence of the output delay, this model alone is not very useful.

Timed automata provide a powerful and uniform framework in which gate delay models of varying sophistication can be specified. One has the flexibility to assign a distinct delay for every possible sequence of input vectors (for an n input gate, there are $2^n \cdot (2^n - 1)$ of these). However, in practice, there are only a few sequences of input vectors that result in distinct delays. Many sequences of input vector transitions result in the same gate delay, and we make use of this property to simplify the timed automaton of a gate. For example, in Figure 2, the delays $t_{11 \rightarrow 01}^f$ and $t_{11 \rightarrow 10}^f$ have the same value, and so do the delays $t_{01 \rightarrow 11}^r$, $t_{10 \rightarrow 11}^r$, and $t_{00 \rightarrow 11}^r$. Therefore we group together these delays in our timed automaton model for the AND gate, resulting in a simpler delay model. In this way, our model incorporates input sequence dependent delays without becoming prohibitively large. A delay model in which different vector pairs are assigned distinct delays

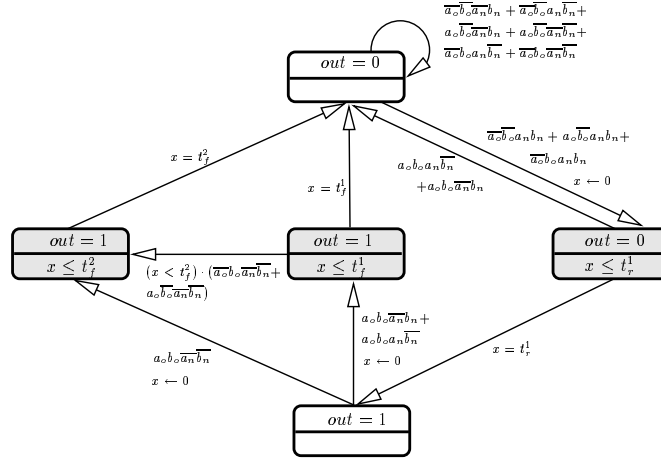


Fig. 3. Timed Automaton for AND gate in Figure 2

was previously introduced by [FK94]. However, this model requires a significantly larger number of transitions to model delays.¹

The timed automaton for an AND gate is shown in Figure 3. The inputs to the AND gate are a and b . a_o and a_n represent the old and new values of the input a respectively. Similarly for input b . If the input $a = b = 1$ is followed by $a = 1, b = 0$, then this condition is represented by $a_o b_o a_n \bar{b}_n$. Each oblong box in Figure 3 represents a *location*. “Transient” locations are shaded. For each location, the value of the gate output is listed on top. The invariant condition associated with the location is listed below its output value. If $a_o = a_n$ and $b_o = b_n$, we stay in the same location by means of a self-loop arc (not shown). Also, if an input changes and then returns to its original value while the automaton is still in its temporary location (referred to as a *glitch*), we return to the starting location. These glitching arcs are not drawn for ease of readability. For example, if we are in the location corresponding to $out = 0$, we remain in this location if condition $\bar{a}_o \bar{b}_o a_n \bar{b}_n$ occurs. This is because an AND gate has a 0 output under both the old and new input vectors. If, from the location corresponding to $out = 0$, condition $\bar{a}_o b_o a_n b_n$ occurs, then timer x is set to 0, and a transition is made to a temporary location where $out = 0$. After a delay of t_r^1 , a final transition is made to a location where $out = 1$. This corresponds to the gate output changing from 0 to 1 after a delay of t_r^1 , when inputs change from $a = 0, b = 1$ to $a = 1, b = 1$. However, if $a = 1, b = 1$ is followed by $a = 1, b = 0$ and $x < t_r^1$, the automaton returns to the location with $out = 0$.

When inputs change from $a = 1, b = 1$, depending on whether one or both inputs change to a 0 value, two delays are possible for this gate. If only one of the inputs changes to a 0 value, then the delay of the gate is t_f^1 . If both inputs change to a 0, then the delay of the gate is t_f^2 . As discussed earlier in this section, $t_f^1 > t_f^2$, because t_f^2 corresponds to the case where the current charging node out_b of Figure 2 is doubled. In either case, the timed automaton makes a transition to a temporary location, after setting timer x to 0. From this temporary location,

¹ For a 3-input NAND gate, the authors of [FK94] state that 109 transitions would be required in their gate model. In our model, only four distinct transitions are required.

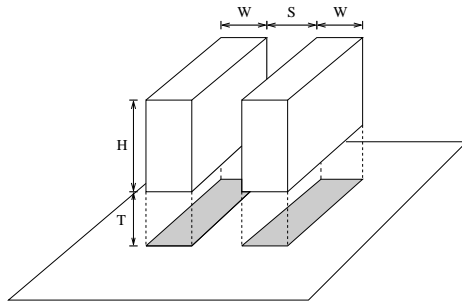


Fig. 4. Two Wires on a VLSI Integrated Circuit

it makes the final transition to the location corresponding to $out = 0$, after the appropriate delay. In case both inputs do not *simultaneously* change to a 0 value, the timed automaton makes a transition to the temporary location corresponding to one input change, from which it makes a transition to the temporary location corresponding to two input changes, if the second input arrives before t_j^2 .

Timed automata for other gates are constructed similarly. In the following, we refer to the electrical node in the circuit which is at ground potential as gnd , and the node which is at supply potential as vdd . For the transistor level representation of any gate, suppose there are n paths from the evaluation (i.e., output) node to vdd , and m paths from the evaluation node to gnd . In Figure 2, the evaluation node is out_b . Assuming that all paths to vdd and gnd have the same effective size of transistors, the timed automaton for the gate will have n distinct rising delays, and m distinct falling delays. In general, for a k input gate, each of the $2^k \cdot (2^k - 1)$ input sequences give rise to distinct delays at the output, as was suggested in [CL95]. However, in practice, the delays are tightly clustered around the $n + m$ distinct values we use.

For a general gate, we first determine the values of m and n . After this, we compute each distinct rise and fall delay by means of SPICE [N75], a transistor-level simulator, which gives us the exact delay taking into account the transistor level net-list of the gate. All SPICE simulations in this paper use $0.1 \mu\text{m}$ transistor models. Interconnections are assumed to be made of copper, and are assumed to correspond to a $0.1 \mu\text{m}$ fabrication process.

3.3 Modeling Cross-Talk between Wires

As the minimum feature size of VLSI fabrication processes decrease, certain electrical phenomena become significant. Figure 4 shows a graphical view of two wires on a integrated circuit. In most VLSI processes, $W = S$. As minimum feature sizes decrease, W (and S) decrease linearly, but T , the distance between wires on different metal layers decreases sub-linearly. As a result, the ratio of the capacitance between a wire and its neighboring wire to the capacitance between a wire and wires on other metal layers increases with diminishing feature sizes [NTRS97].

One of the effects of this increased capacitance to neighboring wires is a large variation in the delay of the wire. If the neighboring wires switch in the same direction as the wire of interest, the delay of the wire is decreased, and if they switch in the opposite direction, its delay increases. The effect of the transition

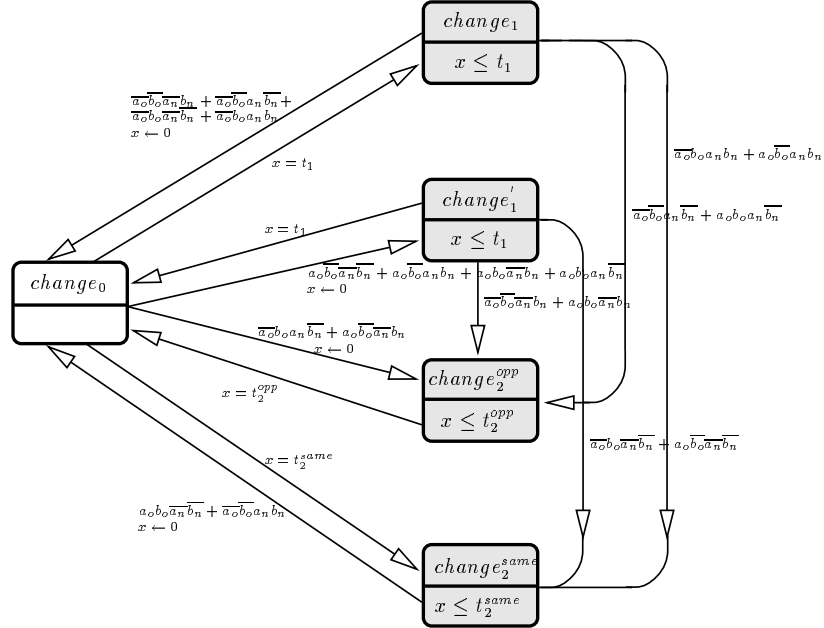


Fig. 5. Timed Automaton Model for Delay of a Pair of Wires

activity of a wire on its neighboring wires is referred to as *cross-talk*. A SPICE simulation of three neighboring wires of length $150 \mu\text{m}$ in a $0.1 \mu\text{m}$ process shows a 2:1 variation in the delay of the center wire due to cross-talk. In the set-up for this experiment, the lines were driven by inverters, whose P devices were 4.5 times larger than a minimum P device, and whose N devices were 3 times larger than a minimum N device. The parasitics for this configuration of wires were determined using a 3-dimensional parasitic extractor called SPACE [SPC]. VLSI processes with a $0.1 \mu\text{m}$ feature size are still a few years away from production, but the above simulation indicates that delay variation effects due to cross-talk will be a major problem in the future. For this reason, it is becoming increasingly important for circuit timing analyzers to incorporate the effect of cross-talk.

We model the effect of cross-talk between two wires by the timed automaton shown in Figure 5. In this figure, the location $change_0$ represents the condition when both wires have a stable value. From this location, if one of the wires switches, a transition is made to temporary location $change_1$ or $change_1'$. While in these temporary locations, if the other wire switches, an transition is made to a location which models both wires switching. There are two such locations, $change_2^{same}$, which models the two wires switching in the same direction, and $change_2^{opp}$ which models the two wires switching in opposite directions. The delay associated with $change_1$ and $change_1'$ is t_1 . The delays associated with $change_2^{same}$ and $change_2^{opp}$ are t_2^{same} and t_2^{opp} respectively. These delays are determined by running SPICE on the physical configuration of wires encountered. In general, $t_2^{same} \leq t_1 \leq t_2^{opp}$.

4 Delay Computation with Timed Automata

We pose the delay computation problem as follows: Given a combinational circuit, described as an interconnection of circuit components, and a set of primary input waveforms, we want to determine the latest time that primary outputs become stable. The set of input waveforms is represented by a timed automaton I as described in section 3.1. The circuit is described as an interconnection of timed automata, G_1, \dots, G_n . The delay parameters of these are determined by SPICE simulation of the transistor-level circuit description, as described in the example in Figure 2. The delay computation problem is then formally stated in the following manner: Let

$$F = (I \parallel G_1 \parallel G_2 \parallel \dots \parallel G_n)$$

represent the evolution of the circuit over time for the primary input waveforms described by I . Let us denote by G_{o_1}, \dots, G_{o_m} the circuit components whose outputs are primary outputs of the circuit. For each j , define E_{o_j} to be the edges $\langle s_{o_j}, t_{o_j}, \varphi, \chi, R \rangle$ of G_{o_j} for which the primary outputs at s_{o_j} and t_{o_j} are different. The delay of the circuit is the latest time that edge in some E_{o_j} can be traversed in F . Denote by E_{switch} the set of edges of F whose projection onto some G_{o_j} lies in E_{o_j} , and let $S_{switch} \subseteq \Sigma_F$ be the set of states of F that have an outgoing E_{switch} transition. The goal is then to compute the largest time elapsed on paths from initial states of F to S_{switch} . The most straightforward way to obtain this information is to traverse the state-space of F . However, the size of this state-space is often very large. Automata networks describing circuits have certain special properties, which enable significant improvements to the traversal techniques. The rest of this section elaborates on these properties and how they are used for making state-space exploration more efficient.

4.1 A Region Automaton with Integer Delays

Traversal techniques for the dense state-spaces of timed automata can be broadly classified into two categories. The first class of methods use a set of inequalities to represent a convex subset of the timer space, and state sets are represented by pairing locations with sets of such convex subsets (See, [LPW97], for instance). These methods are not suitable for our purposes because they represent locations explicitly, and the boolean component (therefore the number of locations) of the automaton F is often considerably large. Our delay computation technique is based on the second category of methods, called “region automaton”-based methods [AD94]. The key feature of these methods is the division of the state-space into a finite number of “regions”, each of which is one equivalence class of a relation (“region equivalence”) defined on Σ_F . Each region makes up one state of a finite automaton, called the region automaton, and a transition relation is defined on the regions such that the essential information about F is captured. This finite automaton can then be analyzed using OBDD-based methods. In many cases this enables one to handle large state spaces.

Region equivalence for timed automata as defined in [AD94] has to distinguish between clock valuations which have different orderings of the fractional parts of clock values. This contributes a factor of k to the state-space, where k

is the total number of timers used in F . For a subclass of timed automata, this component of the state space can be eliminated by using the following fact proven in [HMP92]: if a timed automaton uses no strict inequalities in clock predicates, for each run of the automaton, there exists a run that makes transitions only at integer time points, and goes through the same sequence of locations. In fact, such a run can be obtained as follows: Let $0 \leq \varepsilon \leq 1$ be arbitrary. For a transition γ , let t_γ denote the time it is taken. If the fractional part of t_γ is $\leq \varepsilon$, round t_γ to the nearest integer from below, otherwise, round it up the nearest integer from above. The run obtained in this manner is also a run of the automaton. It follows that, for delay computation purposes, timers can be treated as integer valued variables which increase at the same rate.

In our context, timer predicates refer to the time elapsed between two transitions in node voltages, which are analog waveforms over real-valued time. It does not make any physical sense to specify that the time elapsed between two analog transitions can be any value less than (or greater than) but not equal to c time units. The set of possible elapsed times is simply rounded up to the nearest closed interval with integer end-points. Therefore, combinational circuits can be modeled without strict inequalities and the above integer interpretation for timers can be used. Then, a run $\sigma_0 \xrightarrow{\gamma_0} \sigma_1 \xrightarrow{\gamma_1} \sigma_2 \xrightarrow{\gamma_2} \dots \xrightarrow{\gamma_{k-1}} \sigma_k$ can be viewed as an interleaving of *time passage transitions* and *control transitions* as follows: $\sigma_0 \xrightarrow{\delta_0} (\sigma_0 + \delta_0) \xrightarrow{\langle \xi_0, \xi'_0 \rangle} \sigma_1 \xrightarrow{\delta_1} (\sigma_1 + \delta_1) \xrightarrow{\langle \xi_1, \xi'_1 \rangle} \dots \xrightarrow{\langle \xi_{k-1}, \xi'_{k-1} \rangle} \sigma_k$, where $\sigma_i + \delta_i$ is shorthand for all timers in σ_i being incremented by $\delta_i \in \mathbb{N}$. Note that time passage transitions with $\delta_i \geq 1$ can be realized by a sequence of $\delta = 1$ transitions. We have found that taking time steps of one at a time results in more efficient ordered binary decision diagram (OBDD)-based analysis algorithms and makes the formulation of the delay problem easier, as will be seen in Section 4. Clearly, the reachability and delay properties of the automaton remain the same.

With these observations, the timed automaton takes the form of a finite-state machine with states of the form $\sigma = \langle s, \nu \rangle$, where ν is an integer-valued clock evaluation. The transition relation T_A of a timed automaton A is given as $T_A^\delta \cup T_A^c$ where

$$T_A^\delta = \{ \langle \sigma, \xi, \sigma + \delta, \xi \rangle \mid \sigma \in \Sigma_A, \delta = 1, \text{ and } \text{wait}(\sigma, \delta) \text{ and } \xi \text{ is an observation} \}$$

represents the *time passage transitions*, where all variables except timers remain constant, and

$$T_A^c = \{ \langle \sigma, \xi, \sigma', \xi' \rangle \mid \sigma, \sigma' \in \Sigma_A, \sigma' \text{ is obtained from } \sigma \text{ by the traversal of some edge } e \in E \text{ on observation event } \langle \xi, \xi' \rangle. \}$$

represents the *control transitions*, where the location of the automaton changes. In the rest of the paper, the timed automata will be identified with this discrete transition structure. Our delay computation algorithm is essentially a breadth-first search of this discrete state-space using OBDDs to represent state sets and transition relations.

4.2 The Region Automaton is Acyclic

The transition structure of a timed automaton representing a combinational circuit is acyclic for the following reason: The primary input waveforms that we consider in delay analysis remain constant after a certain point in time (Section 3.1). Combinational circuits are designed to stabilize to a certain final state after the inputs become constant. If F had a cycle, it could be traversed an unbounded number of times, which would point to an instability in the circuit.² A cycle in the transition structure of F corresponds to a cycle in each component of F . Observe that all such cycles involve a change in some signal in the circuit. Therefore, such a cycle in F points to oscillations in some circuit nodes. For a correctly designed combinational circuit, this sort of behavior should not occur³.

While exploring a large state transition graph, the bottleneck is often the size of the representation for the set of explored states. However, if an acyclic state transition graph is traversed in breadth-first manner starting with the set of initial states, one does not need to store all of the traversed states: Denoting by $S^{(k)}$ the set of states that can be reached from the initial states by traversing exactly k edges, it suffices to store $S^{(k)}$ at the k th step of the traversal. This allows a (heuristic) memory-time trade-off: Typically, memory is saved by storing a smaller set, but a state may be visited more than once, which results in re-computation. For an arbitrary system, the amount of re-computation could be prohibitively large. However, the delay of a combinational circuit is bounded by the delay of the longest topological path, which places a polynomial bound on k .

In practice, we found that this approach results in significant savings in memory (See Section 5). Our technique, like many OBDD based methods, is memory limited, and by representing $S^{(k)}$ only, we were able to handle circuits that we could not have handled otherwise.

4.3 The Algorithm

For uniformity of notation, let us rename the automata comprising F to C_1, \dots, C_m , such that $F = (C_1 \parallel \dots \parallel C_m)$. Let $T_1^\delta \cup T_1^c, T_2^\delta \cup T_2^c, \dots, T_m^\delta \cup T_m^c$ be the transition relations of C_1, \dots, C_m . Recall that the outputs σ_j of each timed automaton C_j are a deterministic function of its location given by $\sigma_j = \mu_j(s_j)$. Therefore, for the purposes of traversing the state-space of F , it is possible to express all transition relations in terms of the state variables of the C_j 's. Then the transition relation of component j can be expressed solely in terms of state variables in the form $T_j(\sigma_j, \sigma_{\rightarrow j}, \sigma'_j, \sigma'_{\rightarrow j})$. Here $\sigma_{\rightarrow j}$ denotes the set of state variables of fan-ins of C_j .

With these, the structure of our delay computation algorithm is described in Figure 6

² Note that F incorporates automata that generate the primary input waveforms, and is thus a closed system. Therefore, it is possible for every path in F to be traversed.

³ If the circuit does have unstable behavior, the algorithm that will be described in Section 4.3 will not converge. However, note that the delay of the circuit is bounded from above by the delay of the longest topological path. Using this fact, we can limit the number of iterations, and also determine whether the circuit stabilizes.

```

 $k \leftarrow 0, t \leftarrow 0, S^{(0)} = S_1^{(0)} \wedge S_1^{(0)} \wedge \dots \wedge S_m^{(0)}$ 
repeat
  repeat
     $S^{(k+1)} = \text{ONESTEP}(S^{(k)}, \langle T_1^C, \dots, T_m^C \rangle)$ 
     $k \leftarrow k + 1$ 
  while  $S^{(k)}(\sigma) \neq S^{(k-1)}(\sigma)$ 
     $S^{(k+1)} = \text{ONESTEP}(S^{(k)}, \langle T_1^\delta, \dots, T_m^\delta \rangle)$ 
     $k \leftarrow k + 1$ 
   $t \leftarrow t + 1$ 
while  $S^{(k)}(\sigma) \neq S^{(k-1)}(\sigma)$ 

```

Fig. 6. Algorithm COMPUTEDELAY

where ONESTEP (described in Figure 7) computes the states reached from $S^{(k)}$ by traversing one edge in the transition relation given by $T = T_1 \wedge T_2 \wedge \dots \wedge T_m$

$\text{ONESTEP}(S^{(k)}, \langle T_1, \dots, T_m \rangle)$

$$S^{(k,k+1)}(\sigma, \sigma') = S^{(k)} \wedge \bigwedge_{i=1}^m T_i$$

$$S^{(k+1)}(\sigma') = (\exists \sigma) S^{(k,k+1)}(\sigma, \sigma')$$

Fig. 7. Algorithm ONESTEP

The relation $S^{(k,k+1)}$ represents the outgoing transitions from $S^{(k)}(\sigma)$ as a function of σ where the σ and σ' are variables corresponding to the present and next states, respectively. The algorithm repeats the following loop until the circuit stabilizes: First, control transitions are explored until all states reachable through them are computed, then a time increment of 1 is taken. The delay of the circuit is then the maximum t such that $S^{(k,k+1)}$ includes an edge for which there is a change in a primary output⁴. Note that minimum delay computation, as well as any timed safety property check can easily be incorporated into this scheme.

The practical limitation in applying the algorithm above is the size of the OBDD for $S^{(k)}$. In the next section we present a method for efficiently computing and representing $S^{(k)}$'s.

4.4 A Conjunctively Decomposed Representation

$S^{(k)}(\sigma)$ typically has a large number of variables in its support. In order to get compact (monolithic) OBDD representations, it is necessary to choose a good order of the variables. Heuristically, variables that are strongly correlated must be close to each other in the order, and variables encoding correlated integers must be interleaved. For the problem at hand, it is in general not possible to find a total order that satisfies these constraints, especially because all variables corresponding to timers are correlated with each other [BMPY97]. To deal with this difficulty, we employ a conjunctively decomposed representation for the state sets $S^{(k)}$. The decomposition corresponds to a “slicing” of the circuit in the following manner. The circuit is partitioned into slices SL_1, \dots, SL_p , where each slice SL_i consists of a set of circuit elements (C_i 's). The slices cover the circuit and no C_i belongs to more than one slice. The SL 's are ordered topologically, i.e., if $a < b$ no C_i 's in SL_b can be in the transitive fan-in of a C_j in SL_a . (See Figure 9 for an example.).

⁴ It is a simple OBDD operation to check whether $S^{(k,k+1)}$ contains such an edge.

$S^{(k)}$ is then represented by a collection of relations, each corresponding to a slice. We construct $S_1^{(k)}(\sigma_0)$, $S_1^{(k)}(\sigma_0, \sigma_1)$, $S_2^{(k)}(\sigma_1, \sigma_2)$, \dots , $S_p^{(k)}(\sigma_{p-1}, \sigma_p)$ such that each $S_j^{(k)}$ specifies the set of states of SL_j at step k of the algorithm. Note that each $S_j^{(k)}$ has σ_{j-1} in its support in addition to σ_j . Intuitively, $S_j^{(k)}$ specifies what assignments to σ_j are part of $S^{(k)}$ for each assignment to σ_{j-1} , i.e., the correlation between the state variables of adjacent slices is captured. Since each $S_j^{(k)}$ has much fewer variables in its support, the total size of the $S_j^{(k)}$'s is in general much smaller than a monolithic representation for $S^{(k)}$.

$$\begin{aligned} \text{For } j = 1 \text{ to } p \\ S_j^{(k,k+1)}(\sigma_{j-1}, \sigma_j, \sigma'_{j-1}, \sigma'_j) &= \left((\exists \sigma_{j-2}, \sigma'_{j-2}) S_{j-1}^{(k,k+1)} \right) \wedge S_j^{(k)} \wedge \bigwedge_{C_i \in SL_j} T_i \\ S_j^{(k+1)}(\sigma'_{j-1}, \sigma'_j) &= (\exists \sigma_{j-1}, \sigma_j) S_j^{(k,k+1)} \end{aligned}$$

Fig. 8. Algorithm ONESTEP'

The modified algorithm described in Figure 8 performs one step of reachability computation operating on one slice at a time. For slice j , slice $j - 1$ provides the inputs, and since the correlation between the states is stored by $S_j^{(k)}$, for each state, the corresponding inputs are supplied. Ideally, for such a decomposed representation, one would like the following equalities to hold: (i) $S^{(k)} = \bigwedge_{j=1}^p S_j^{(k)}$, and (ii) $S_j^{(k)} = (\exists \tau_j) S^{(k)}$, where τ_j is the set of variables not in the support of $S^{(k)}$. This would be the case if, at the k th iteration of the algorithm, the state variables of slice k were only correlated with those of slice $k - 1$. However, for an arbitrary slicing and arbitrary circuit components, state variables in non-adjacent slices may be correlated. For algorithm ONESTEP', it can be proven by induction that $\bigwedge_{j=1}^p S_j^{(k)}$ is a superset of $S^{(k)}$, and thus the slicing based delay computation method is conservative.

In practice, we have not found any cases where the results of the slicing method is different from the results of the monolithic but exact method. We offer the following intuitive explanation for this fact: Combinational circuit elements have bounded memory, and thus, their states are only strongly correlated with the states of other elements close-by. Let us call a circuit component “active” if it is not in a stable state. Suppose that the circuit is sliced in such a way that the topological delay through each slice is roughly equal. Then, at any given time, the active elements are contained in a portion of the circuit consisting of two adjacent slices. Our algorithm performs traversal by sweeping the circuit using a window consisting of two slices. If transitions at any point in time are confined to two adjacent slices, then next-state computation as performed by ONESTEP' is exact. We are now working on an exact algorithm based on this intuition. The algorithm dynamically focuses only on the active portion of the circuit.

5 Experimental Results

As our benchmark circuits, we use n -bit adders built by cascading 2-bit Carry Skip Adder (CSA) blocks (Figure 9). These circuits are known to have false paths, therefore, even in the absence of cross-talk, a topological analysis over-approximates delay.

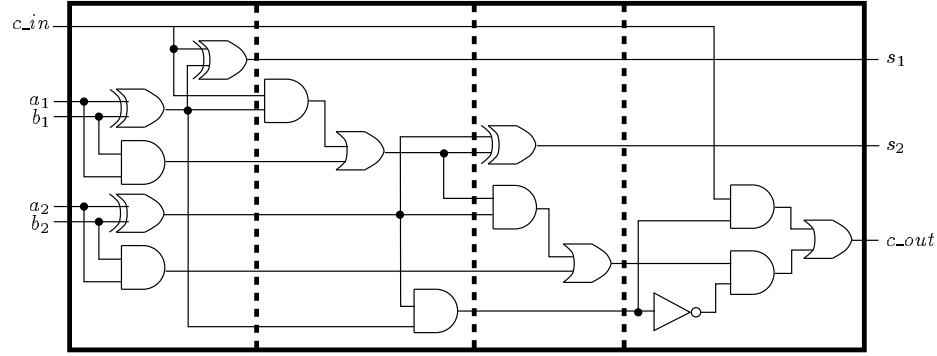


Fig. 9. Two-bit carry-skip adder block. The dashed lines indicate the boundaries between the slices.

We conducted two sets of experiments. The first set, described in Section 5.1 compares the performance and accuracy of conventional delay analysis methods, a circuit simulator and the algorithm described in this paper. The second set, described in Section 5.2 demonstrates that our algorithm for traversing the state-space scales much better than a non-specialized OBDD based method.

5.1 Comparison with conventional methods

For this experiment, we constructed a 4-bit adder using two CSAs (Figure 10). Two configurations of this circuit were used:

K_1 : The c_{out} output of CSA1 and primary input A3 were neighboring wires in the circuit layout, resulting in a possible delay variation for both due to cross-talk.

K_2 : No cross-talk exists between any wires.

These configurations were meant to be “proof-of-concept” circuits, meant to highlight the difference between various algorithms. For an industrial strength tool, cross-talk information needs to be extracted for the whole circuit from the layout. Here, we performed this only for the pair of wires c_{out} and A3.

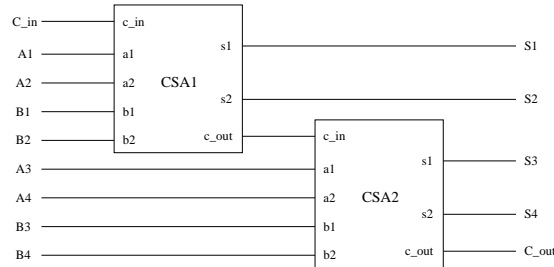


Fig. 10. A 4-bit CSA

Four different algorithms were tested on this circuit:

A_1 : *Circuit Simulation*: The circuit was modeled in SPICE, and exact delays were computed by simulating the input vector pairs causing the largest delay. The run-time for exhaustive simulation of all input vector pairs was estimated by multiplying the run-time for one vector pair by the number of possible pairs. Observe that, for this circuit, it was possible to determine the input

vector pairs causing worst-case behavior. This is in general not possible, and exhaustive simulation needs to be performed.

- A_2 : *Our Approach*: We first created timed automata for each gate using the ideas described in Section 3.2. This was done for each of the 4 gate types in the circuit of Figure 9. Similarly, a timed automaton model for cross-talk between two wires was constructed as discussed in section 3.3. The value of different delays was determined by a transistor level simulation, using SPICE. For the gate delay models, the numbers obtained were rounded to the nearest multiple of 5 ps. The maximum delay of the circuit was computed using the algorithm described section 4. Our algorithms were implemented on the verification platform MOCHA [AH+98].
- A_3 : *“Exact timing analysis”*: This is the algorithm described in [MSBS93], implemented on the SIS platform. This algorithm does not account for cross-talk, but reports the delay of the longest true path. Although this method is not truly exact, it is referred to as “exact timing analysis” in the delay computation literature. The implementation we used did not allow input-dependent delays, so worst-case delay parameters obtained from SPICE simulation were used for gates and wires.
- A_4 : *Topological delay analysis*: This was done using SIS [SSL+92]. This method does not model cross-talk, and does not account for false paths. Again, worst-case delay parameters obtained from SPICE simulation were used for gates and wires.

The input vector sequence which results in the maximum delay for the 4-bit CSA also results in the c_{out} output of CSA1 switching in the opposite direction as the primary input A3. However, these signals are significantly separated in time, so there is no increase in circuit delay due to cross-talk between the wires. An algorithm which is not cross-talk-aware will assign a worst-case delay for each of the wires, and hence estimate a larger delay for configuration K_1 than for configuration K_2 , whereas, in reality, the two configurations result in the same delay.

Method	With cross-talk (K_1)		Without cross-talk (K_2)	
	run-time(s)	max delay (ps)	run-time(s)	max delay (ps)
A_1	2.936×10^6	611	3.09×10^6	616
A_2	602	660	617	660
A_3	1.745	770	1.619	740
A_4	0.1	920	0.1	890

Table 1. Experimental Results

The results of these runs are described in table 5.1. SPICE models circuit behavior most closely, and we take the delay that SPICE computes as our reference. Note that the maximum delay that SPICE computes for the two configurations is almost the same, as expected.

The results obtained from A_2 are promising:

- Our method correctly determines that the delay of the circuit is identical whether configuration K_1 or K_2 is used.

- The computed delay is within 10% of the true circuit delay as computed by SPICE.⁵ The other two schemes have higher estimates.
- The run-time of the timed-automata scheme is three orders of magnitude less than the estimated run-times for SPICE.

Since algorithm A_3 cannot detect that cross-talk does not actually take place in K_1 , it reports a larger delay than for K_2 . This is due to the fact that cross-talk in this case was modeled using buffers whose delays are the worst-case delay under cross-talk (t_2^{opp} in Figure 5). Also, the value of delay computed by this scheme for configuration K_2 is larger than that computed by timed automata. This is because the delay model used by exact timing analysis is a pin-delay model, and suffers from the drawback described in section 3.2.

The topological timing analysis scheme (algorithm A_4) has the lowest run-times, but gives the most inaccurate results. False paths in the circuit are not detected, and further, cross-talk is not modeled.

From these results, it is clear that the method we developed is powerful, significantly faster than transistor-level simulation, and much more accurate than other timing analysis methods.

5.2 Comparison with non-specialized traversal

To quantify the improvement brought about by our traversal heuristics, we compared their performance with a non-specialized traversal algorithm. The non-specialized algorithm made use of a partitioned representation of the transition relation while keeping the representation of $S^{(k)}$'s monolithic. The algorithms were run on a family of n -bit carry-skip adders (n-CSA) as described earlier. For this set of experiments, we used the same delay models with a coarser time discretization (1 time unit = approx. 14 ps) in order to demonstrate how the algorithm's performance scales with circuit size. The results presented in Table 2 show that our algorithm scales remarkably well. In contrast, the non-specialized algorithm ran out of space (1GB) for the 4-bit CSA and was not able to complete even with the help of dynamic variable reordering heuristics. Observe that, with the help of our heuristics, we were able to handle models with thousands of OBDD variables.

6 Conclusions and Future Work

We have addressed delay problems due to decreasing minimum feature sizes of VLSI circuits. The need for more accurate timing analysis methods, and also for cross-talk-aware timing analysis was fulfilled by: (i) new gate delay models, and (ii) an accurate, timed automaton-based analysis scheme. The advantages of our approach are:

⁵ This discrepancy is partly due to the fact that in the timed-automaton models for gates, delays are rounded up to the nearest multiple of 5 ps. The more important reason, however, has to do with the fact that we compute gate delays assuming a nominal loading. In the circuit, each instance of any gate drives a different load. This fact is naturally taken into account by SPICE. If we had incorporated this factor into our delay models, we would have had a larger number of models, but our results would have been closer to SPICE.

no. of CSAs	no. of timers	circuit delay (ps.)	no. of BDD vars	BDD mem.(MB)	CPU time
2	30	634	526	29	2 min
3	45	767	820	58	8 min.
4	60	900	1114	78	13 min.
5	75	1034	1408	85	21 min.
6	90	1167	1702	102*	30 min.
8	120	1440	1180	63*	15 min.
16	240	2480	2362	78**	2.8 hr.

Table 2. Experimental Results for n-bit CSA Adders. * denotes dynamic OBDD variable reordering. * denotes a coarser time discretization (1 time unit = approx. 35 ps.). The second column gives the number of timer variables used in modeling each circuit. The fourth and fifth column indicate the number of OBDD variables in the circuit representation and the total memory used by OBDDs.

- Circuit delay is computed much more accurately than algorithms A_3 and A_4 of Section 5.1, since the delay model we use is more realistic. This remains true even in the absence of cross-talk.
- The method models cross-talk between wires in an integrated circuit. This feature is absent from algorithms A_3 and A_4 .
- Our preliminary implementation shows reasonable run-times and the run-time of our method is orders of magnitude less than exhaustive circuit simulation. We expect to achieve significant speed-ups with a better implementation

Our method cannot currently handle large circuits. For large circuits, we propose that our scheme be applied to determine the delay of a critical portion by modeling only the path and other nodes in its “electrical neighborhood”. By pruning away irrelevant portions of the circuit, our algorithms may be able to analyze the critical portions. We intend to test this conjecture on industrial circuits.

Future work in terms of the computational approach will proceed in two directions: (i) An exact method that dynamically determines the set of active elements and performs computation only on that part of the circuit, and, (ii) the use of an output load dependent delay model for this scheme.

References

- [AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Comp. Sci.*, 126:183–235, 1994.
- [AH+98] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, S. Taşiran. MOCHA: Modularity in Model Checking To appear in *Intl. Conf. on Computer-Aided Verification, CAV ’98*
- [BMPY97] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some Progress in the Symbolic Verification of Timed Automata. In *Proceedings of the 9th Intl. Conf. on Computer-Aided Verification, CAV ’97*, LNCS 1254, pages 191–201, Springer-Verlag, 1997.
- [CL95] B. S. Carlson and S.-J. Lee. Delay optimization of digital CMOS VLSI circuits by transistor reordering. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.14, (no.10), pages 1183-92, Oct. 1995
- [CWS97] V. Chandramouli, J. Whittemore, and K. Sakallah. AFTA: A Delay Model for Functional Timing Analysis *Proceedings of the 1997 ACM/IEEE International*

- Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 5–14, 1997, Austin, Texas.
- [DKM93] S. Devadas, K. Keutzer and S. Malik. Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms In *IEEE Transactions on Computer-Aided Design*, 12(12): 1913–1923, December 1993.
- [HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *Proceedings of the 19th International Colloquium on Automata, Languages, and Programming (ICALP 1992)*, Lecture Notes in Computer Science 623, Springer-Verlag, 1992, pp. 545–558.
- [FK95] J. Fröbl, and T. Kropf. Verifying Real-Time Properties of MOS-Transistor Circuits In *European Design and Test Conference (EDTC)*, pp. 314–319, Paris, March 1995.
- [FK94] J. Fröbl, and T. Kropf. A New Model to Uniformly Represent the Function and Timing of MOS Circuits and its Application to VHDL Simulation In *European Design and Test Conference (EDTC)*, pp. 343–348, March 1994.
- [LB94] W.K.C. Lam, and Robert K. Brayton. *Timed Boolean Functions- A Unified Formalism for Exact Timing Analysis* ISBN 0-7923-945402, Kluwer Academic Publishers, 1994.
- [LPW97] Kim G. Larsen, Paul Pettersson and Wang Yi. UPPAAL: Status & Developments In *Proceedings of the 9th International Conference on Computer-Aided Verification, CAV '97*. Haifa, Israel, 22–25 June 1997.
- [MP95] O. Maler, A. Pnueli. Timing Analysis of Asynchronous Circuits Using Timed Automata In *ACM Intl. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 249–257, November 1995.
- [M96] K. L. McMillan. A conjunctively decomposed Boolean representation for symbolic model checking. In *Proceedings of the 8th Intl. Conf. on Computer-Aided Verification, CAV '96*, LNCS 1102, pages 13–25, Springer-Verlag, 1996.
- [MSBS93] P. McGeer, A. Saldanha, R. K. Brayton and A. L. Sangiovanni-Vincentelli. Delay Models and Exact Timing Analysis In *Logic Synthesis and Optimization*, pages 167–189, T. Sasao, ed., Kluwer Academic Publishers, 1993.
- [NTRS97] The National Technology Roadmap for Semiconductors, 1997 <http://notes.sematech.org/97melec.htm>
- [N75] L. Nagel SPICE: A Computer Program to Simulate Computer Circuits University of California, Berkeley UCB/ERL Memo M520 May 1995
- [SSL+92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, A. L. Sangiovanni-Vincentelli SIS: A System for Sequential Circuit Synthesis Electronics Research Laboratory, Univ. of California, Berkeley, CA 94720 UCB/ERL M92/41, May 1992
- [TAKB96] S. Taşiran, R. Alur, R. P. Kurshan, and R. K. Brayton. Verifying Abstractions of Timed Systems. In *Proceedings of the 7th Intl. Conf. on Concurrency Theory, CONCUR '96*, LNCS 1119, pages 546–562, Springer-Verlag, 1996.
- [SPC] Physical Design Modeling and Verification Project (SPACE Project).. <http://cas.et.tudelft.nl/research/space.html>
- [TB97] S. Taşiran and R. K. Brayton. STARI: A Case Study in Compositional and Hierarchical Timing Verification. In *Proceedings of the 9th Intl. Conf. on Computer-Aided Verification, CAV '97*, LNCS 1254, pages 191–201, Springer-Verlag, 1997.
- [TKB97] S. Taşiran, Y. Kukimoto and R. K. Brayton. Computing Delay with Coupling Using Timed Automata. In *Proceedings of the 1997 ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 232–244, 1997, Austin, Texas.
- [YH95] H. Yalcin and John P. Hayes. Hierarchical Timing Analysis Using Conditional Delays. In *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design, ICCAD '95*, pages 371–377, 1995