

Computing Delay with Coupling Using Timed Automata

Serdar Taşiran*, Yuji Kukimoto and Robert K. Brayton
Department of Electrical Engineering and Computer Sciences,
University of California, Berkeley

Deep sub-micron circuits place new requirements on timing analysis tools: More accuracy is needed and new effects such as pattern dependent delays and cross-talk must be modeled. We propose a timed-automaton-based approach for computing delays of combinational circuits. Timed automata are used to represent delay models of circuit components and cross-talk effects, as well as sets of waveforms at nodes of the circuit. The uniform framework offered by timed automata has the advantage that accuracy and complexity can be traded-off without needing to change the algorithms used in the analysis. The expressiveness of timed automata is very useful for modeling deep sub-micron effects, however, analysis with timed automata has suffered from complexity problems and has been limited to systems with tens of components. To alleviate this problem, we propose a method that mimics image computation across a Boolean network and argue that by adopting a hierarchical view of combinational circuits and by using a proper set of heuristics for image computation, delay analysis with timed automata can be made practical.

1 Introduction and Motivation

Delay computation for combinational circuits is a well-studied problem. However, with the feature sizes of integrated circuits shrinking to sub-micron levels, some of the underlying assumptions of current delay analysis methods no longer remain valid:

- Higher clock speeds require more accurate modeling of circuit delay. Therefore, more sophisticated gate delay models are needed. For instance, with the shift from static CMOS to dynamic logic and with the use of more complex gates, input pattern dependence of gate delays becomes significant and needs to be incorporated into gate delay models. In certain cases, it is needed to compute delay at the transistor level.
- Wire delays become dominant and analysis of these delays is complicated by the fact that there is cross-talk between wires. Coupling from other nodes can affect the rise and fall times at dynamic nodes ([SN96]). Delay analysis algorithms must use parameters that are extracted from the layout and must take into account the cross-talk factor, which depends on the relative timing of signals as well as their logic values.

As a result, current algorithms for computing exact delay are inapplicable for deep sub-micron circuits. Moreover, simple topological delay is not a conservative estimate for delay, because it does not take cross-talk into account. A “worst-case topological analysis” assuming worst-case cross-talk on all wires at all times is likely to be too conservative and not suitable for high-performance circuits. This leaves circuit simulation at the transistor level as the only currently used method that can accurately compute circuit delay. This approach suffers from the fact that, for a large circuit, it is too costly to simulate all possible input patterns.

Timed automata have been used to represent the delay characteristics of gates ([MP95],[TAKB96],[TB97]). However, analysis and verification with timed automata is complex, and has been considered viable only for systems consisting of tens of components. This has restricted the use of timed automata to asynchronous circuits, which are generally smaller than synchronous circuits but require a more detailed analysis than delay computation. Efficient exact methods had already been devised for computing the delays of acyclic combinational circuits ([DKM93], [LB94], [MSBS93], [YH95]) and thus networks of timed automata appeared to be too general a model for this purpose.

As argued above, deep sub-micron circuits place different demands on timing analysis tools, and this new setting makes the expressiveness of the timed automata formalism desirable. However, the complexity

* Supported by SRC under Contract DC-324-026.

barrier still remains to be dealt with. In this study, we propose a timed automaton-based delay analysis method for combinational circuits. In practice, when performing analysis with timed automata, the most significant contributor to complexity is the number of timer variables used¹. We argue that delay computation can be accomplished using a moderate number of such variables by (i) employing a hierarchical view of combinational circuits, and (ii) computing the set of possible output waveforms by propagating the “image” of input waveforms across the circuit using heuristics to keep intermediate results manageable. To support our claim, we observe that, to achieve higher clock speeds, circuits are built to have fewer levels of gates, and for circuits that involve many levels of logic, only a small fraction of the gates are in transition at any given time. Later sections will make more precise how these observations can be used to alleviate verification complexity.

In section 2 we present timed automata, and discuss how circuits and sets of waveforms can be represented and manipulated using them. Section 3 describes the algorithms and heuristics for computing delay using timed automata and discusses the advantages of this approach. Section 4 summarizes the implementation issues that remain to be addressed.

To convey the intuition behind the approach more clearly, we refrain from a completely formal presentation and, instead, illustrate the key points using examples. We refer the interested reader to the Appendix and previously published work ([TAKB96] and [TB97]) for a more precise explanation of analysis with timed automata.

2 Modeling Circuits and Waveforms

In the same way that finite state machines are used to represent regular languages as well as input-output behaviors of sequential circuits, we use timed automata to represent sets of waveforms and input-output behaviors of combinational circuits. A timed automaton can be viewed as a finite state machine (FSM) augmented with real-valued stopwatch variables, called timers. Timers are used to keep track of the time that elapses between events: On certain transitions, the automaton resets a timer and later transitions are allowed to take place only when the timer value falls in a certain range. Each edge of the timed automaton corresponds to a change in the value of a signal in the circuit that it represents. Since these changes are not synchronized with a discrete clock, timed automata operate on a continuous timed domain. As an example, a timed automaton representing a buffer is depicted in figure 1, which is explained in the following section.

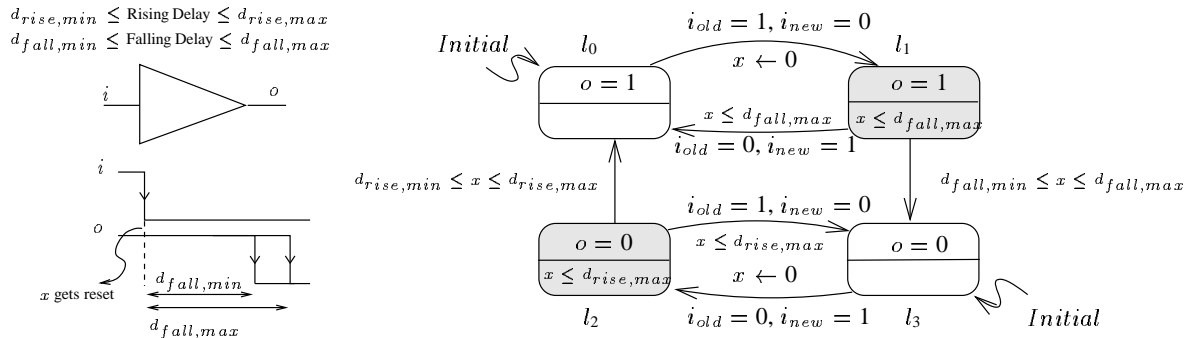


Figure 1: An inertial delay buffer with rising delay in the range $[d_{rise,min}, d_{rise,max}]$ and falling delay in the range $[d_{fall,min}, d_{fall,max}]$.

2.1 Gate Delay Models

Timed automata have been previously used to represent gate delay models ([MP95], [TAKB96], [TB97]) for the verification of asynchronous circuits. As an example, figure 1 (from [TB97]) depicts a timed automaton

¹See Section 2 for explanation of timer variables

representing the inertial delay model for a buffer as described in [MP95]. The inertial delay model specifies ranges $[d_{rise,min}, d_{rise,max}]$ and $[d_{fall,min}, d_{fall,max}]$ for the rising and falling delays. Suppose that the buffer starts at location l_0 representing the case where the input and output of the buffer are both 1 at power-up. When the input i goes low, the automaton responds by moving to l_1 and resetting timer x . x records the time that elapsed since i went low. In l_1 the output o is still high, but a change is pending. The automaton can move to l_3 when x is between $d_{fall,min}$ and $d_{fall,max}$ and at l_3 the output goes low. The use of x as a guard on the transition to l_3 ensures that the output follows the input within time $[d_{fall,min}, d_{fall,max}]$. If the input goes high again before the automaton moves to l_3 , the automaton may move back to l_0 . This represents the case when an input pulse lasting between $d_{fall,min}$ and $d_{fall,max}$ gets dropped by the buffer, i.e., it is not reflected at the output. If i goes high while $x < d_{fall,min}$, the automaton moves to l_0 , which models the fact that pulses lasting less than this minimum length are dropped.

As the example above demonstrates, the expressiveness offered by timed automata enables the use of more elaborate gate delay models than those used by conventional delay analysis methods. An example of a significant effect that can be conveniently incorporated into a timed automaton delay model is the dependence of delay on the relative arrival times of the inputs. Consider an n input NAND gate with inputs $\vec{x} = x_1, \dots, x_n$ and output $y = \overline{x_1} \cdot \overline{x_2} \cdot \dots \cdot \overline{x_n}$. Suppose that $y = 1$ initially and that the inputs switch from $\vec{x}_{old} = x_{1,old}, \dots, x_{n,old}$ to $\vec{x}_{new} = 1, 1, \dots, 1$, and suppose that x_i is the latest arriving input. Conventional delay methods would stipulate that the output y stabilizes at $t_i + d_{i,max}$, where $d_{i,max}$ is the maximum pin-to-pin delay from x_i to y . However, especially for a dynamic CMOS logic gate, the actual delay depends on the times that other inputs arrive at, as well as the values $\vec{x}_{old} = x_{1,old}, \dots, x_{n,old}$. This effect may be negligible for static CMOS gates with few inputs, but for high performance designs, complex dynamic logic gates with large numbers of inputs are frequently used, and for these types of gates, the dependence of delay on this effect is significant ([CL95]).

In the next section we present timed automata more formally, and describe the features that are useful in modeling delay characteristics.

2.2 Timed Automata

Let X be a finite set of real-valued variables. An X -valuation ν assigns a nonnegative 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 \mathbf{f} is an assignment of values to variables in P . A P -event is a pair $\langle \mathbf{f}, \mathbf{f}' \rangle$ consisting of P -valuations \mathbf{f} and \mathbf{f}' denoting the old and the new values of the variables in P . A P -predicate is a Boolean predicate on \mathbf{f} and \mathbf{f}' .

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 type. 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.
- 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 reset function R that specifies for each timer x its value after the edge is taken, $R(x)$:
 - (i) $R(x) = 0$: x is reset to 0.
 - (ii) $R(x) = y$, where $y \in X$: x takes on the value of y right before the transition. The capability to assign the value of one timer to another enables the reuse of timer variables and facilitates minimization of timer variables ([DY96]). If $y = x$, x is left unchanged.

- (iii) $R(x) = f(\lfloor \cdot \rfloor)$, where, for $n \in \mathbb{N}$, $f(n)$ is an interval $[m_{min}, m_{max}]^2$. The intuitive interpretation is the following: f is a look-up table that specifies a range of possible values for x by looking at the integer part of its current value. x can take on any value in the interval $[m_{min}, m_{max}]$. This capability is useful for modeling coupling between timed automata.

Since timers take on values from \mathbb{R} , the state space of a timed automaton is infinite. Analysis is performed by partitioning this space into a finite number of equivalence classes ([AD94]). The version of timed automata presented above is different from that of [AD94], but its state space can be partitioned into a finite number of equivalence classes in exactly the same way as [AD94]. A proof of this fact appears in the appendix.

2.3 Modeling Wire Delay and Cross-talk

Wires constitute a significant part of delay in deep-submicron combinational circuits. The analysis and modeling of wire delays are complicated by the fact that coupling from nearby wires needs to be taken into account. The delay characteristics of a wire segment depend on the technology used and the circuit layout. For a given wire, these characteristics can be derived from 3-D physical extraction and simulation of possible input patterns and coupling from nearby wires. It is feasible to run detailed simulation at this level locally and derive delay characteristics precisely. Simulating all possible input patterns and interactions for an entire circuit, however, is clearly infeasible. As described in the previous section, for gate delays this problem is solved by constructing a higher level gate delay model which is a conservative abstraction of the underlying analog behavior. We apply the same approach to wire delay. After performing detailed analysis of wire delay at the analog level, a more abstract model is constructed and is represented by a timed automaton. This model must encapsulate all analog behavior at a more abstract level and must involve few parameters.

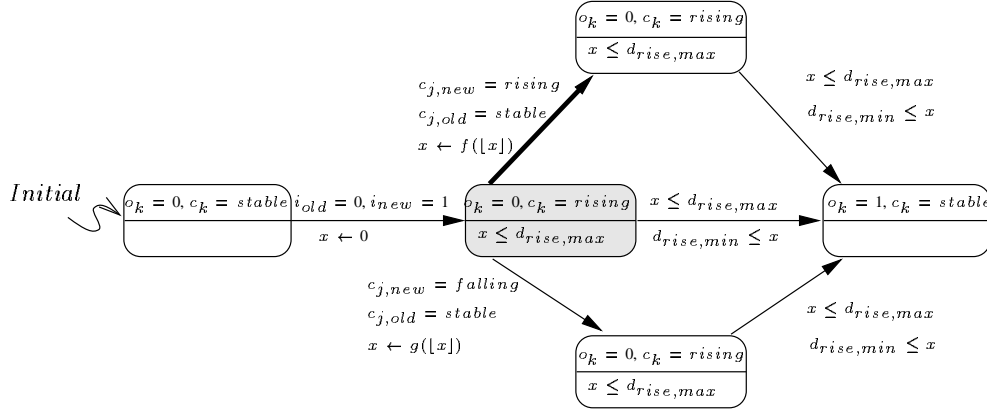


Figure 2: Delay model for wire k incorporating cross-talk from wire j . i is the signal that drives wire k .

Figure 2 shows one possibility for such a wire delay model. Let A_k denote the timed automaton representing wire k . A_k has two outputs o_k and c_k . o_k takes on values from the set $\{0, 1\}$ and represents the digital value of the signal at the output end of wire k , while c_k takes on values from the set $\{rising, falling, stable\}$ and indicates whether wire k is making a transition or has a stable signal on it. If the signal on wire j affects wire k , c_j is an input to the automaton representing wire k . Suppose that i , the signal that drives wire k makes a transition from 0 to 1 and in response to this, A_k moves to the shaded location where, if x has value \bar{x} , within $[d_{rise,min} - \bar{x}, d_{rise,max} - \bar{x}]$ time units, a transition is taken and the output o_k becomes 1.

A typical scenario is where another wire j runs parallel to wire k and because of this, if j starts rising while k is rising, this reduces the time k takes to rise. Suppose while A_k is in the shaded location c_j changes from *stable* to *rising*. The coupling between the wires is modeled by the “reset” function f on the marked edge. f will increase the value of x to reflect the fact that now o_k will take less time to

²See the Appendix for the restrictions on the form of f

rise. x can take on any value from the interval $f(\lfloor \bar{x} \rfloor) = [n_{min}, n_{max}]$. Let \tilde{x} be the new value of x . Now a transition to the location where $o_k = 1$ is possible when $d_{rise,min} \leq \tilde{x} \leq d_{rise,max}$, i.e., within $[d_{rise,min} - \tilde{x}, d_{rise,max} - \tilde{x}]$ time units. Thus, f should be such that if the signal on wire k has been rising for $\lfloor \bar{x}, \bar{x} + 1 \rfloor$ time units, the remaining time to rise for o_k with positive coupling from j should fall inside the interval $[d_{rise,min} - f(\lfloor \bar{x} \rfloor), d_{rise,max} - f(\lfloor \bar{x} \rfloor)]$. Such $d_{rise,min}$, $d_{rise,max}$ and a look-up table f can either be determined analytically or computed by simulating the two wires. By choosing the range $[n_{min}, n_{max}]$ for \tilde{x} large enough, we can ensure conservative modeling of the effects that cross-talk can have on the rise time of x . Choosing the time unit to be smaller will yield a more accurate model but to larger verification complexity. It is possible to refine the delay model by incorporating more cases for possible interactions between wires. This example is aimed at demonstrating that timed automata have enough expressiveness to capture cross-talk effects and represent them in a way that captures all behavior obtained from simulation.

2.4 Modeling Sets of Waveforms

For combinational circuits, the two interesting kinds of primary input waveforms are the following:

- (*Floating delay*): The inputs are allowed to change their values arbitrarily until time 0. After time 0 all inputs remain stable.
- (*Two vector delay*): The inputs and all intermediate nodes of the circuit are stable at some value prior to time 0. At time 0 the inputs switch to new values and remain stable thereafter.



Figure 3: Automaton representing the set of input vectors for the two-vector delay model. \vec{i}_{old} and \vec{i}_{new} represent the vectors of primary inputs before and after time 0. \vec{i}_{old} and \vec{i}_{new} are selected non-deterministically, which enables the automaton to represent all input vector pairs.

These sets of input waveforms can be represented concisely by timed automata (See figure 3 for the 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.

2.5 Hierarchical Representation with Timed Automata

Composition of Timed Automata

For a set of interconnected circuit components, a timed automaton representing the entire system is obtained by *composing* the timed automata representing each component ([TAKB96]). The composition operation is analogous to the one 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 $A \parallel B$.

Variable Hiding (“Smoothing”)

Another common operation on timed automata is *variable hiding*. Given a timed automaton $A = \langle S, S_0, O, I, X, \alpha, \mu, E \rangle$, hiding an input variable i removes i from I and replaces each input predicate χ on an edge with the predicate $(\exists i, i') (\chi)$, which does not depend on i . Hiding an output variable removes it from O and the output function μ is restricted to the remaining set of variables $O \setminus \{o\}$. The hiding operation is used to obtain a representation only in terms of the variables of interest. The operation of hiding internal variables of a composition of automata is often referred to as “smoothing”. The automaton obtained from A by hiding variable y is denoted by $(\exists y)A$.

A Hierarchical View of Combinational Circuits

As has been observed by [KB97] and [YH95] and as will be clearer in section 3, hierarchical representation and verification are essential for being able to handle large circuits. The theory and algorithms for hierarchical reasoning with timed automata had been studied previously in [TAKB96] and [LLPY97].

In the approach we propose, a combinational circuit is represented hierarchically as follows: leaf-level components are connected (or coupled) groups of gates, transistors and wires, and higher level modules consist of interconnections of modules from lower levels. Timed automata represent the functional and timing characteristics of each leaf level model. Automata representing higher level modules are obtained from lower-level ones by composition and hiding of internal variables. To keep the automata of higher level modules small, we often resort to abstraction. In this case, the high level automaton overapproximates the behavior of the interconnection of lower level models. By controlling the amount of abstraction used, one can trade-off accuracy and efficiency of analysis. Previous research on performing minimization of products of timed automata ([DY96],[LPY95]) reports encouraging results from algorithms related to this problem.

We now illustrate the hierarchical representation of the preceding paragraph using the example of Figure 4. The circuit consists of four identical NAND gates, each of which is modeled by an ideal NAND gate followed by an inertial delay buffer as in Figure 1. These constitute the leaf-level modules in the hierarchy. Composing these four models we obtain a timed automaton that represents the behavior of the signals $a, b, c, x, y,$ and z . This automaton has $4^4 = 256$ locations and uses 4 timer variables. Since the behaviors of the internal signals x, y and z do not need to be represented, we smooth these variables to obtain a timed automaton model for the module XNOR. This timed automaton specifies exactly what waveforms are possible at c for each pair of input waveforms at a and b . Since no abstraction has been performed at this point, there is no reduction in the number of locations and timers. If we insist on an exact model, little or no further reduction is possible in the number of timers and locations. Suppose, however, that for the delay computation we have at hand, modeling XNOR at this level of detail is unnecessary and that it is sufficient to specify the earliest possible time c becomes unstable and the latest possible time it takes on a stable value. The behavior of c in the unstable intervals does not need to be specified exactly. Then we can model XNOR by the automaton shown in figure 5. The number of locations and timers has been reduced significantly at the expense of allowing more behavior at c than is possible in the original circuit. By picking d_{min}, d_{max} and the look-up table f properly, we can ensure that the higher level model is conservative but is tight enough for our application.

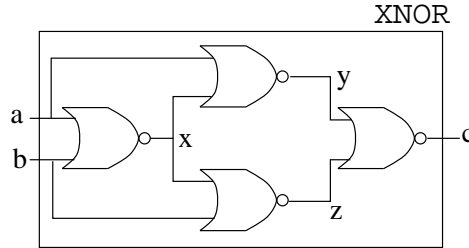


Figure 4: A hierarchical view of a combinational circuit block.

In contrast to other delay models and delay computation methods, the timed automaton framework offers the advantage of having a uniform representation that can be used at all levels of the hierarchy. To the best of our knowledge, current industrial practice is to exploit hierarchy in an ad hoc way by using look-up tables for delay simulation. Our method is distinguished by the facts that (i) it enables formal verification, i.e., coverage of all possible inputs, and (ii) it provides a formal guarantee that the hierarchical models are conservative abstractions of lower level models.

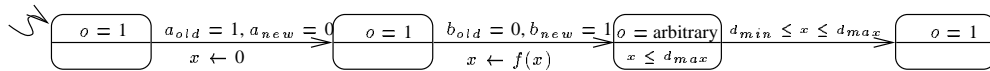


Figure 5: Part of the automaton representing the higher level model for XNOR. The reset function f models the dependency of delay on the relative arrival times of a and b . The rest of the automaton is constructed similarly using a total of two timers.

3 Delay Computation Viewed as Image Computation

We pose the delay computation problem as follows. A combinational circuit and a set of primary input waveforms are given. The goal is to determine for each primary output the latest time at which it becomes stable. The set of input waveforms is represented by a timed automaton as described in section 2.4. The circuit is also described as an interconnection of timed automata, the delay parameters of which have been determined using layout information as described in section 2.3. Each circuit component transforms the set of waveforms at its inputs to a set of waveforms at its output. By propagating waveforms across the circuit in topological order, we arrive at the set of resulting primary output waveforms, again represented by timed automata. From this set, the maximum delay of the circuit can be calculated, as well as other useful information such as whether the set-up and hold times of latches are met. The procedure we propose for propagating input waveforms across the circuit to arrive at output waveforms mimics implicit image computation across a Boolean network. The rest of this section is devoted to making the preceding argument more precise.

Given a timed automaton I representing a set of input waveforms, and a circuit consisting of components represented by G_1, \dots, G_k , the composition $C = I \parallel G_1 \parallel G_2 \parallel \dots \parallel G_k$ is a representation for the waveforms observed at all nodes of the circuit. Smoothing all primary input variables (denoted by \vec{i}) and intermediate node variables (denoted by \vec{n}) from C yields a timed automaton representation

$$F = (\exists \vec{i}, \vec{n})(I \parallel G_1 \parallel G_2 \parallel \dots \parallel G_k)$$

for all primary output waveforms that the input set I can give rise to. One can then determine the longest time that F can take from an initial location of F to the set of locations where the primary outputs have stabilized (applying the the maximum delay algorithm of [CY91], for instance). Note that the uniform timed automaton representation enables a clean mathematical formulation of the delay computation problem.

The analogy with image computation manifests itself in the equation for F . As is the case in taking the image of inputs across a Boolean network, there is freedom in the order that the compositions (\parallel operations) and the smoothing (\exists operations) are performed. Heuristics need to be used for ordering these operations so that intermediate results can be kept small. Otherwise, for instance, if the product of all the G 's is taken first, much more information is represented than is necessary for calculating the maximum delay. We argue that the following two ideas can make the computation of F viable:

- (i) The circuit is viewed hierarchically, as consisting of carefully chosen partitions, which in turn consist of sub-partitions. This corresponds to “parenthesizing” the expression $G_1 \parallel \dots \parallel G_n$. A timed automaton representation of the input-output behavior of a partition is obtained from its sub-partitions using composition, variable hiding and, possibly, some abstraction as discussed earlier. Abstraction corresponds to overapproximating F .
- (ii) Image computation is performed by “propagating wavefronts” across the partitions. This corresponds to performing the composition of the partitions in topological order, and smoothing variables whenever all the G_i 's that depend on those variables have been composed. As depicted in figure 6, if the set of waveforms at a given cut-set has been characterized, all variables to the left of the cut-set can be smoothed. This allows minimization and possibly abstraction of the intermediate results.

The partitions must be chosen in a way to expedite the image computation. We believe that the following heuristics will work well:

- As much as possible, create partitions with disjoint support. For such partitions, only the set of possible waveforms at the output nodes need to be stored, the input variables can be hidden. This is not possible for arbitrary partitions, since the correspondence between the output waveforms and the input waveforms needs to be remembered.
- Find partitions which have few output signals: the fewer variables there are in the cut-set, the easier to represent the set of waveforms.

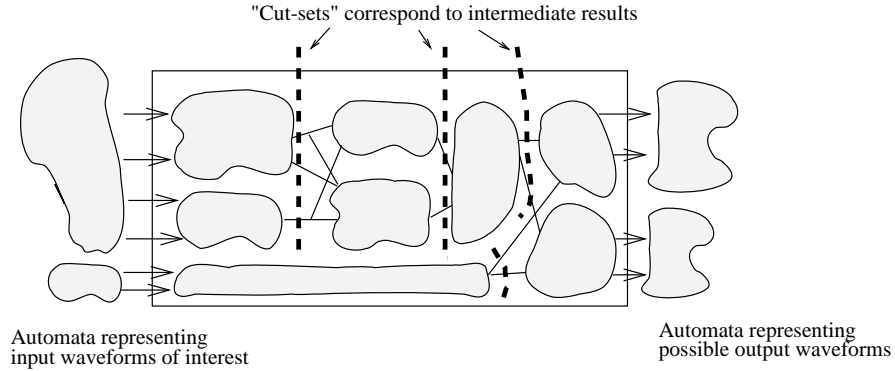


Figure 6: Waveform propagation across partitions.

- Long and narrow partitions are good, because they can essentially be treated separately, they have few input variables in their support, and the cut-sets in such partitions are small.
- The sizes of the partitions can be determined in a way similar to the selection of intermediate points in implicit image computation. When the sizes of timed automata get too big, it is time to smooth out some variables and perform some abstraction.
- It is possible to partition the set of possible waveforms at each point. If the timed automata get too big, one can divide the set of waveforms into separate sets represented by separate automata and perform image computation on each of them individually.

The image computation across a given block can then be performed by recursively partitioning it using the heuristics above, until each sub-partition is small enough to be handled in one pass.

3.1 Compose-Smooth-(Abstract)

For this approach to be practical, the “Compose-Smooth-(Abstract)” operation needs to be implemented efficiently, keeping results manageably small at all times. We propose the following algorithm:

- (i) Take the product of automata and hide intermediate variables.
- (ii) Perform “untimed reachability analysis” on the product automaton.
Delete locations that are unreachable.
- (iii) Apply the timer minimization algorithm of [DY96] to the reachable part of the product.
- (iv) Minimize the result with respect to bisimulation equivalence, modulo transitions that do not correspond to observable events. (*Observable events* are events that involve input or output variables. Events involving internal variables are said to be *unobservable*.)

Note that the algorithm avoids computation on the “timed state space” involving sets of timer valuations. Computations involving this space often have exponential complexities, whereas algorithms that operate on a syntactic description of the timed automaton have their complexities expressed in terms of the number of locations. Therefore, even if the latter have high degree polynomial complexities, they are desirable if they can achieve reductions in the size of the timed state space. We now elaborate on steps (ii)-(iv).

- (ii) The “untimed reachability analysis” ignores the timing information on the edges, and may declare locations reachable whereas analysis using timing information would have concluded that they are not. Untimed reachability analysis provides an overestimate of the set of reachable locations that can be computed quickly.
- (iii) [DY96] presents an algorithm that minimizes the number of timer variables used by a timed automaton. It operates on a textual description of a timed automaton and outputs a timed automaton that is bisimilar to the input automaton. [DY96] achieves further reduction of timer variables by allowing on the automaton edges assignments of the form “ $x \leftarrow y$ ”, where x and y are timers. The added flexibility of assigning one timer’s value to another does not affect the complexity of the verification problem and is incorporated in our timed automaton model also.

- (iv) Smoothing internal variables deletes many edge predicates. Therefore, the resultant automaton has many edges that are associated with unobservable events only (“ τ - transitions”). An automaton that is bisimilar to this automaton in terms of observable events can be computed by collapsing locations that have the same outputs and that are connected with τ transitions only. Further reduction in the number of locations can be achieved by computing the bisimilarity quotient³ of this automaton, treating timer predicates and reset functions merely as symbols labeling the edges. This method results in less minimization than computing the timed bisimulation quotient, but is less costly.

3.2 Why is it advantageous to perform delay computation in this way?

The main advantage of our approach is the fact that it provides many “knobs”, so that one can trade-off between efficiency and accuracy. Below we list some of these knobs and other important advantages of this method.

- Timed automata are expressive enough to incorporate many delay models, and a different delay model can be incorporated without having to change the delay computation algorithm. It should be noted that for simpler delay models, the timed automaton representations are also small. For instance, an XBD0 delay model ([MSBS93]) with a single upper-bound on delay per gate can be represented by a timed automaton with four locations and one timer.
- The timed automata formalism provides a sound way to obtain a delay model for a group of gates, using abstractions if needed to obtain a smaller and more abstract description that is guaranteed to be correct.
- In addition to computing delay, other timing aspects of the circuit can be verified as well. One can determine whether spikes occur at a given node, whether there is a channel-connected path from source to ground, etc.
- Sets of similar waveforms can be represented concisely with timed automata. To obtain a smaller representation, sets of waveforms can be over-approximated. The timed automaton formalism provides a formal guarantee that the analysis is indeed conservative.
- Much flexibility is allowed in choosing clusters and choosing when to perform the “Compose” and “Smooth” operations. Many heuristics used in image computation can be mimicked.
- The automaton representing the set of waveforms at a given cut-set can be partitioned into separate sets or simplified by over-approximating this set.
- Suppose it becomes too expensive to store the correlation between two signals x and y . We can then over-approximate, and say if α is a possible waveform at x and β is a possible waveform at y , then the combination (α, β) is a possible waveform at (x, y) . By ignoring some correlations, we obtain a conservative but possibly simpler analysis.

The most distinct conceptual advantage of the timed automaton framework is the fact that it provides a precise formulation for delay computation, with delay models that can be made as accurate as desired. The exact solution can then be approximated in a formally correct way.

4 Conclusions and Future Work

We proposed a timed-automaton based method for delay computation. The key features of this method are hierarchical modeling and analysis, and the use of heuristics that mimic image computation on Boolean networks. An important side benefit of this approach is the decoupling of modeling issues from the algorithmic issues of analysis and verification by employing timed automata as a clean interface between the two. In this way, the algorithms developed remain applicable for different delay models and at different levels of the hierarchy. Moreover, more efficient techniques developed for analyzing timed automata can be incorporated immediately.

³*Bisimilarity* in the sense we are using is an equivalence relation between the locations of an automaton. The *bisimilarity quotient* is an automaton that has one location representing each equivalence class. See [CLM89], for instance, for a more detailed treatment of bisimulation.

The timed automaton network is conceptually appealing, and provides a lot of flexibility in implementing delay computation algorithms. The theoretical aspect of analysis with timed automata has been investigated extensively. Many algorithmic aspects of the problem have been addressed in the literature as well, as discussed in previous sections. Some important practical issues specific to our method remain to be addressed. Efficient implementations of algorithms are needed for

- minimizing the number of timers and locations of a timed automaton,
- computing the (untimed) bisimilarity quotient
- (recursively) partitioning a combinational network to facilitate image computation.

The practical success of this approach depends most critically on the partitioning algorithm. Hierarchical verification works well when the high level models encapsulate just the right amount of information needed to carry out the analysis. Without proper partitioning, high level models may have to represent information about many intermediate signals, which would make analysis costly. Most other key ingredients of the approach have been studied in different contexts. Currently we are implementing and integrating these algorithms. Once the framework is built, we will experiment with heuristics for partitioning and abstraction.

References

- [AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- [ACHDW92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill and H. Wong-Toi. Minimization of Timed Transition Systems In *Proceedings of the Third Conference on Concurrency Theory*, 1992
- [CLM89] E. M. Clarke, D. E. Long, and K. L. McMillan “Compositional Model Checking” In *Proc. 4th Annual Symposium on Logic in Computer Science*, June 1989.
- [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
- [CY91] C. Courcoubetis and M. Yannakakis Minimum and maximum delay problems in real-time systems In *Proceedings of the Third Workshop on Computer-Aided Verification*, LNCS 575, pages 399–409, 1991.
- [DY96] C. Daws and S. Yovine Reducing the Number of Clock Variables of Timed Automata In *Proceedings of the 1996 Real-Time Systems Symposium, RTSS '96*, Washington, DC, USA, December 1996.
- [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.
- [KB97] Y. Kukimoto and R. K. Brayton Hierarchical Timing Analysis under the XBD0 Model In *Proceedings of the Intl. Workshop on Logic Synthesis, IWLS '97*, May 1997.
- [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.
- [LLPY97] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi Efficient Verification of Real-Time Systems: Compact Data Structure and State Space Reduction In *Proceedings of 8th IEEE Real-Time Systems Symposium*, December 1997.
- [LPY95] K. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, 1995.
- [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.
- [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.
- [SN96] K. Shepard and V. Narayanan Noise in Deep Submicron Digital Design In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD '96*, pages 524–31, 1996
- [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.
- [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.
- [YH95] H. Yalcin and John P. Hayes Hierarchical Timing Analysis Using Conditional Delays. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD '95*, pages 371–377, 1995

Appendix

Definitions

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 positive 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 $\langle \delta, \mathbf{f}, \mathbf{f}' \rangle$ consisting of a positive real-valued increment δ and the observation-event $\langle \mathbf{f}, \mathbf{f}' \rangle$. Such an event means that the automaton can wait for the time period δ and then update its output variables from $\mathbf{f}(O)$ to $\mathbf{f}'(O)$ while the environment is updating the input variables from $\mathbf{f}(I)$ to $\mathbf{f}'(I)$. 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 $\gamma = \langle \delta, \mathbf{f}, \mathbf{f}' \rangle$ in Γ_A , define $\sigma \xrightarrow{\gamma} \tau$ iff $\mathbf{f}(O) = \mu(s)$, $\mathbf{f}'(O) = \mu(t)$, $wait(\sigma, \delta)$, and there exists an edge $\langle s, t, \varphi, \chi, R \rangle$ such that $(\nu + \delta) \models \varphi$, $\langle \mathbf{f}, \mathbf{f}' \rangle \models \chi$, and μ is obtained from ν by applying R as explained above.

Reset functions

Reset functions $f : \mathbb{N} \mapsto \mathbb{N} \times \mathbb{N}$ that appear in assignments of the form $x \leftarrow f(\lfloor x \rfloor)$ (type (iii) “look-up table” functions of section 2.2) must have the property that for each f there exists some $k_f \in \mathbb{N}$ such that for all $k \geq k_f$, $f(k) = f(k_f)$. In words, the value of f is constant after k_f , which enables f to be represented in the form of a look-up table.

Let a timed automaton $A = \langle S, S_0, O, I, X, \alpha, \mu, E \rangle$ be given. For each timer $x \in X$, let $C_x \in \mathbb{N}$ be the maximum among (i) constants that x gets compared with in the X -predicates on the edges of A and invariants on the locations of A , (ii) constants k_f such that the reset $x \leftarrow f(\lfloor x \rfloor)$ occurs on some edge of A . Following [AD94], we define two timer valuations ν and μ to be *region equivalent* iff for all $x \in X$, either both $\nu(x)$ and $\mu(x)$ are larger than C_x , or all of the following hold

- $\lfloor \nu(x) \rfloor = \lfloor \mu(x) \rfloor$, and
- For all $x' \in X$ such that $\nu(x') \leq C_{x'}$, $Fr(\nu(x)) \geq Fr(\nu(x'))$ iff $Fr(\mu(x)) \geq Fr(\mu(x'))$, and
- $Fr(\nu(x)) = 0 \Leftrightarrow Fr(\mu(x)) = 0$.

where for $w \in \mathbb{R}$, $Fr(w)$ denotes the fractional part of w . We write $\langle s, \nu \rangle \equiv \langle t, \mu \rangle$ iff $s = t$, and ν and μ are region equivalent. Observe that if two states are region equivalent, then they satisfy the same set of X -predicates. Let \mathcal{R}_A be the finite set of equivalence classes of \equiv .

Lemma 1 *Let $\langle s, \nu \rangle \equiv \langle s, \mu \rangle$. If, for a timed event $\gamma = \langle \delta, \mathbf{f}, \mathbf{f}' \rangle$, $\langle s, \nu \rangle \xrightarrow{\gamma} \langle s', \nu' \rangle$, then there exist $\gamma' = \langle \delta', \mathbf{f}, \mathbf{f}' \rangle$ and $\langle s', \mu' \rangle$ such that $\langle s, \mu \rangle \xrightarrow{\gamma'} \langle s', \mu' \rangle$ and $\langle s', \nu' \rangle \equiv \langle s', \mu' \rangle$.*

Proof: Let $e = \langle s, s', \varphi, \chi, R \rangle$ be the edge of the automaton associated with the transition $\langle s, \nu \rangle \xrightarrow{\gamma} \langle s', \nu' \rangle$. From [AD94] we know that there exists δ' such that $wait(\langle s, \mu \rangle, \delta')$ and $\langle s, \mu + \delta' \rangle \equiv \langle s, \nu + \delta \rangle$. $\mu + \delta' \models \varphi$, therefore, the edge e can be taken on the timed event $\gamma' = \langle \delta', \mathbf{f}, \mathbf{f}' \rangle$. We argue that the resets in R can be applied to $\mu + \delta'$ in such a way that the final result is region equivalent to ν' . Type (i) and (ii) resets pose no difficulty: given two equivalent X -valuations, applying resets of these types leaves them in the same equivalence class. Suppose that on e , a type (iii) reset is applied to clock x , which assigns to it non-deterministically a value in the range $[n_{min}, n_{max}]$. Since $\lfloor (\nu + \delta)(x) \rfloor = \lfloor (\mu + \delta')(x) \rfloor$, for the γ transition, as well as the γ' transition, the new value of x is selected from $[n_{min}, n_{max}]$. $\mu'(x)$ must be picked to have the same integer part as $\nu'(x)$, and its fractional part must be picked such that the ordering of timers according to their fractional parts is the same for ν' and μ' . This is possible, since we have complete freedom in choosing the fractional part of $\mu'(x)$ and guarantees that $\langle s', \nu' \rangle \equiv \langle s', \mu' \rangle$. ■

Theorem 2 *Suppose that starting from $\langle s_0, \nu_0 \rangle$, the following sequence of transitions are possible: $\langle s_0, \nu_0 \rangle \xrightarrow{\gamma_0} \langle s_1, \nu_1 \rangle \xrightarrow{\gamma_1} \dots \xrightarrow{\gamma_{k-1}} \langle s_k, \nu_k \rangle$. Let μ_0 be such that $\langle s_0, \nu_0 \rangle \equiv \langle s_0, \mu_0 \rangle$. Then there exists a sequence of timed events $\gamma'_0, \dots, \gamma'_{k-1}$ such that $\langle s_0, \mu_0 \rangle \xrightarrow{\gamma'_0} \langle s_1, \mu_1 \rangle \xrightarrow{\gamma'_1} \dots \xrightarrow{\gamma'_{k-1}} \langle s_k, \mu_k \rangle$ and for each i , γ'_i differs from γ_i only in the time increment, and not in the assignments to the input and output variables.*

Proof: Follows by induction on k from Lemma 1. ■