

POSET Timing and its Application to the Synthesis and Verification of Gate-Level Timed Circuits

Chris J. Myers, Tomas G. Rokicki, Teresa H.-Y. Meng

Abstract— This paper presents a new algorithm for timed state space exploration, *POSET timing*. *POSET timing* improves upon geometric methods by utilizing concurrency and causality information to dramatically reduce the number of geometric regions needed to represent the timed state space. The utility of *POSET timing* is illustrated by its application to the automatic synthesis and verification of *gate-level timed circuits*. Timed circuits are a class of asynchronous circuits that incorporate explicit timing information in the specification which is used throughout the synthesis procedure to optimize the design. Using *POSET timing*, our synthesis procedure derives a timed circuit that is hazard-free. The circuit uses only basic gates to facilitate the mapping to semi-custom components, such as standard-cells and gate-arrays. The resulting gate-level timed circuit implementations are 30 to 40 percent smaller and 30 to 50 percent faster than those produced using other asynchronous design methodologies. This paper also demonstrates that timed designs can be smaller and faster than their synchronous counterparts. The *POSET timing* algorithm can not only efficiently verify our synthesized circuits but also a wide collection of large, highly concurrent timed circuits and systems that could not previously be verified using traditional techniques.

Keywords— Timed asynchronous circuits, *POSET timing*, geometric regions, partial orders, logic synthesis, formal verification.

I. INTRODUCTION

IN recent years, there has been a resurgence of interest in the design of *asynchronous circuits* due to their ability to eliminate clock skew problems, achieve average case performance, adapt to processing and environmental variations, and provide component modularity. Asynchronous circuits can also lower system power requirements by reducing synchronization power, automatically powering down unused components, removing spurious transitions, and easily adjusting to a dynamic power supply. While asynchronous designs have long been used in interface circuits, they are now being considered for the design of low-power embedded controllers and portable devices due to their low-power advantages.

Traditional academic asynchronous design methodologies use unbounded delay assumptions, resulting in circuits that are verifiably correct, but sacrifice timing for simplicity, leading to unnecessarily conservative designs. In industry, however, timing is critical to reduce both chip area and circuit delay. Due to the lack of formal methods to handle timing information correctly, circuits with

timing constraints usually require extensive simulation to gain confidence in the design. Our research bridges this gap by introducing *timed circuits* in which explicit timing information is incorporated into the specification and utilized throughout the design procedure to optimize the implementation. Timed circuits can be significantly smaller and faster than those produced using traditional methods, and they are more reliable than those produced using ad hoc methods. The specification of timing constraints also facilitates a natural interaction between synchronous and asynchronous circuits.

Timing considerations, however, often introduce an additional exponential factor of complexity into the design procedure. As a result, timing analysis has hitherto either been avoided [1], [2], [3], simplified [4], [5], or considered only after synthesis [6]. This paper develops an exact and efficient timing analysis algorithm, *POSET timing*, and applies it to the automatic synthesis and verification of gate-level timed circuits.

In our previous work, an efficient timing analysis algorithm is developed and applied to incorporate timing considerations into the synthesis of timed circuits [7]. We verified that our timed circuit implementations are hazard-free using Burch's timed circuit verifier [8]. This work, however, is not without its limitations. First, since the timing analysis is limited to only choice-free specifications, our synthesis procedure could only be applied to a limited class of circuits. Second, these timed circuit implementations require complex-gates, making it difficult to use semi-custom components, such as standard-cells and gate-arrays, which are becoming increasingly important to improve time-to-market. Third, the discrete-time verification approach employed by Burch's verifier is limited in its applicability since the number of discrete-time states grows exponentially with respect to the number of concurrent events.

This paper presents a new timing analysis algorithm, *POSET timing*, which allows us to develop more general and widely applicable procedures for the synthesis and verification of timed circuits. First, it allows us to extend our synthesis and verification procedures to a very general class of specifications, namely any specification which can be translated into a 1-safe *timed Petri-net*. Second, our new synthesis procedure facilitates the mapping of our implementations to semi-custom components by using additional correctness constraints, thereby producing hazard-free timed circuits using only basic gates such as AND gates, OR gates, and C-elements. Our synthesis procedure has been fully automated in a CAD tool and applied to several examples, resulting in gate-level timed circuit implementations which are 30 to 40 percent smaller and

C. Myers is with the Dept. of Electrical Engineering, University of Utah, Salt Lake City, UT 84112. E-mail: myers@ee.utah.edu.

T. Rokicki is with HP Laboratories, Palo Alto, CA 94303.

T. Meng is with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305.

This research is supported by an NSF Fellowship and a research grant by ARPA.

30 to 50 percent faster than designs using other methodologies. After synthesis, the POSET timing algorithm is used to verify if the synthesized timed circuit implementation back-annotated with delays from a given cell-library satisfies its timed specification. Our verification procedure is shown to be able to rapidly verify larger, more concurrent timed circuits and systems than could previously be verified using traditional techniques.

Section 2 describes the initial specification method, timed Petri-nets, and how they can be translated to ones which can be analyzed using the POSET timing algorithm. Section 3 describes the POSET timing algorithm and how it is used to explore the timed state space. Section 4 briefly explains the application of POSET timing to the synthesis and verification of timed circuits. Section 5 gives our conclusions.

II. TIMED SPECIFICATIONS

Timed Petri nets [9] are a natural way to specify timed circuits and systems. They can, however, be difficult to analyze directly. This work uses timed Petri nets as a specification language, and translates them to a *orbital nets* for analysis. *Orbital nets* are shown later to be efficiently analyzable using the POSET timing algorithm.

A. Timed Petri nets

A 1-safe timed Petri net (TPN) is modeled by the tuple $\langle P, T, F, M_0, \Delta \rangle$ where P is the set of places, T is the set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the set of edges, $M_0 \subseteq P$ is the initial marking, and $\Delta : P \rightarrow \mathcal{N} \times \mathcal{N} \cup \infty$ is an assignment of timing requirements to places. A *marking* is a subset of the places. For a place $p \in P$, the *preset* of p (denoted $\bullet p$) is the set of transitions connected to p (i.e., $\bullet p = \{t \in T \mid (t, p) \in F\}$), and the *postset* of p (denoted $p\bullet$) is the set of transitions to which p is connected (i.e., $p\bullet = \{t \in T \mid (p, t) \in F\}$). For a transition $t \in T$, the presets and postsets are similarly defined (i.e., $\bullet t = \{p \in P \mid (p, t) \in F\}$ and $t\bullet = \{p \in P \mid (t, p) \in F\}$).

Timing is associated with a place as a timing requirement consisting of a lower and upper bound. The lower bound is a nonnegative integer and the upper bound is an integer greater than or equal to the lower bound, or ∞ . Since real values can be expressed as rationals within any required accuracy, restricting the bounds of timing requirements to be integers does not decrease the practical expressiveness of timed Petri nets.

In a timed Petri net, an *untimed state* is a marking of the net (i.e., $M \subseteq P$). A *timed state* is an untimed state with a time-valued clock clk_i associated with each marked place p_i (i.e., $(M, CLK) \subseteq P \times \mathbb{R}$). Each clock advances with time and denotes how long the place has been marked.

The behavior specified by a timed Petri net is defined with an operational semantics composed of two types of operations: firing of transitions and advancement of time. For a given timed state, (M, CLK) , a transition is *untimed enabled* if all places in its preset are marked (i.e., $\bullet t \subseteq M$). The set of untimed enabled transitions is denoted $T_e(M)$. A transition is *timed enabled* when it is untimed enabled

and all places in its preset have a clock that is greater than or equal to the lower bound of the timing requirement on the place (i.e., $\forall p_i \in \bullet t . clk_i \geq l_i$). A transition is *expired* when it is timed enabled and all places in its preset have a clock that is greater than the upper bound of the timing requirement on the place (i.e., $\forall p_i \in \bullet t . clk_i > u_i$). A transition cannot occur until it is timed enabled, and it must occur before it becomes expired. Any timed enabled transition can be fired instantaneously, and any number of transitions can be fired without time advancing. A transition is fired by removing the tokens in the places in its preset and discarding the clocks. The places in the postset of the fired transition are then marked, and all newly marked places are assigned a clock initialized to zero.

Time is advanced by uniformly increasing the clocks associated with the places by an amount δ which is less than or equal to *max-advance* for a given timed state. The function *max-advance* is defined as the maximum amount of time that can be allowed to advance before some transition would become expired. More formally, for a given timed state, (M, CLK) , *max-advance* is defined as follows:

$$\text{max-advance}(M, CLK) = \min_{t \in T_e(M)} \{ \max_{p_i \in \bullet t} \{ u_i - clk_i \} \}.$$

These semantics define the set of *timed firing sequences*, *TFS*, as a sequence of pairs of transition firings and time values. For simplicity, the time value represents a non-negative duration since the previous pair. Executing a timed firing sequence α on a timed Petri net results in the timed state $\text{fire}(\alpha)$. The set *TFS* is defined recursively. The empty sequence ε is in *TFS*. For every firing sequence α in *TFS* and for every value of δ such that $\delta \leq \text{max-advance}(\text{fire}(\alpha))$, then $\alpha; (\phi, \delta)$ is in *TFS*, where ϕ represents an 'empty' firing. In addition, if a transition t is timed enabled in $\text{fire}(\alpha)$, then $\alpha; (t, 0)$ is also in *TFS*. The reachable timed state space, *TS*, is the range of the function *fire* over *TFS*.

B. Orbital nets

In order to use the POSET timing algorithm for timed state space exploration, it is necessary to translate a timed Petri net into an orbital net representation. An orbital net, while similar to a timed Petri net, has several key differences which facilitate the development of efficient timing analysis algorithms. Orbital nets are essentially a labeled 1-safe Petri net extended with automatic net constructions and syntactic shorthands for *composition* and *receptiveness* [10]. The net constructions allow us to have relatively straightforward operational semantics, while the syntactic shorthands allow us to compose the nets without an exponential blowup in net size. These features are described in detail in [10]. Orbital nets also allow *simultaneous* actions (i.e., transitions labeled with sets of actions). These allow us to easily mix behavior and environmental requirements even at the gate model level.

The key difference between orbital nets and timed Petri nets is that timing requirements can be either *behavioral* (*b*) or *constraint* (*c*) (i.e., $\langle l, u \rangle$ type), and that only a single *behavioral place* can be in the preset of any transition.

The timing bounds associated with a behavioral place are used to specify guaranteed timing behavior. The timing requirements associated with a *constraint place* are used to specify desired timing behavior, and they do not affect the actual timing behavior. If the timing requirement on a place is omitted, it is assumed to be $\langle 0, \infty \rangle c$. For a given orbital net, the set B is the subset of the places in P which are of type behavioral, and the set C is the set of constraint places.

The POSET timing algorithm described later relies on the fact that each behavioral place represents a single non-deterministic choice of delay that cannot be affected by other behavioral places. In other words, the delay between a transition in the preset of a behavioral place and a transition in the postset of the same place should always fall between the lower and upper bound of the timing requirement on this place. This property simplifies the function *max-advance* to be the minimum difference over all marked behavioral places between the upper bound of the timing requirement on the place and its clock, or ∞ if there are no marked behavioral places. More formally, for a given timed state, (M, CLK) , *max-advance* is defined as follows:

$$\text{max-advance}(M, CLK) = \min_{p_i \in (M \cap B)} \{u_i - clk_i\}$$

With this definition of *max-advance*, a simple all-pairs shortest-path algorithm (Floyd's algorithm) can be at the core of the timed state space exploration algorithm.

When there are multiple behavioral places in the preset of a transition, the timing semantics allow the delay between the transition in the preset and postset of one of the behavioral places to exceed the upper bound on its timing requirement when the transition is being constrained by another behavioral place. To avoid this problem, orbital nets are restricted to include at most a single behavioral place in the preset of each transition.

In an orbital net, a transition is timed enabled when it is untimed enabled and if there is a behavioral place in its preset, this place's clock is greater than or equal to the lower bound of the timing requirement on the place. Before firing a transition, however, the constraint places in the entire net must be checked, and if any constraint place p_i is marked with $clk_i > u_i$, this firing is marked as a failure. Also, the clocks corresponding to a marking that is removed from a constraint place p_i must be checked, and if $clk_i < l_i$, this firing is also marked as a failure. Finally, after the firing of a transition, every marked behavioral place must have a transition in its postset that is untimed enabled in the new state; if this condition is not satisfied, this firing is a failure. This requirement ensures that every marked behavioral place can fire in all states in which its timing conditions are met, and thus the value of its clock when it fires cannot be controlled by external state. If a failure is detected during synthesis, the specification is inconsistent and must be modified before an implementation can be obtained. If a failure is detected during verification, the timed circuit violates its specification.

C. Translation from timed Petri nets to orbital nets

If timed Petri nets are translated to orbital nets in the obvious way of marking all places as behavioral, then most specifications of interest would naturally have multiple behavioral places in the preset of some transitions. Fortunately, the original timed Petri net specification can always be transformed into an orbital net which satisfies the *single behavioral place requirement*. This transformed net can then be analyzed to find the reachable states using our efficient timing analysis algorithm. Therefore, the POSET timing algorithm can be applied to any 1-safe timed Petri net.

The transformation from an arbitrary timed Petri net into an orbital net which satisfies the single behavioral place requirement is completed in two steps. The first step in the transformation labels all places of the original timed Petri net as behavioral places (including those that have $\langle 0, \infty \rangle$ timing requirements). After step one, the net satisfies the single behavioral place requirement if and only if for every transition in the original timed Petri net there is only one place in its preset. If at least one transition has two places in its preset, then the second step must be performed.

To illustrate the second step, consider a fragment of a timed Petri net that has two places in the preset of a transition shown in Figure 1(a). The desired timing behavior can be depicted graphically as shown in Figure 1(b). This net can be transformed to the orbital net shown in Figure 2(a) which satisfies the single behavioral place requirement. The idea behind this transformation is that a path through the net is created for each possible ordering of the transitions in the preset. This has the effect that each transition in the preset is given the chance to be the one controlling the firing time of the transition in the postset. For illustration purposes, additional events c_0 and c_1 are added to the net to occur simultaneously with the two transitions associated with c . Note that every place from the original net is behavioral while the newly added places are constraint (i.e., $\langle 0, \infty \rangle c$). The timing behavior of c_0 and c_1 are shown graphically in Figure 2(b) and (c), respectively. The behavior of these two together is exactly the desired timing behavior of c . For n behavioral places, the net is transformed to model the $n!$ possible orderings of the n enabling events. While this transformation can lead to a substantial blowup in the net size ($O(2^n)$ times in the worst-case) making the timing analysis more complex, we have found that the value of n tends to be quite small in practical examples.

The transformation is a bit more complicated in the case that one of the behavioral places in the preset has multiple transitions in its postset. In this case, the transformation described above is applied to each of the transitions in the postset individually first. Then, the common transitions and places are stitched together as shown in Figure 3.

It is important to note that an orbital net has both a static and dynamic restriction. The static restriction is the single behavioral place requirement. The dynamic restriction is that if at any time during state space exploration,

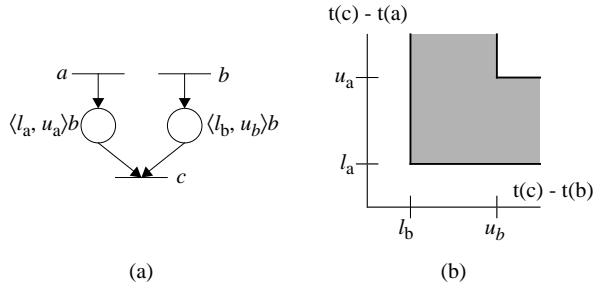


Fig. 1. (a) Fragment of the orbital net that violates the single behavioral place requirement; (b) graphical representation of the desired timing behavior.

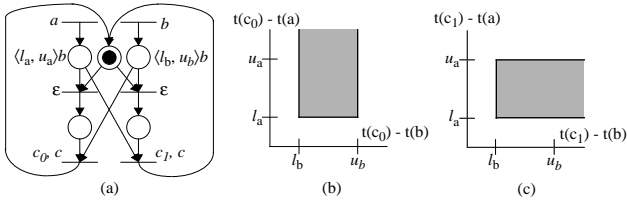


Fig. 2. (a) Orbital net that satisfies the single behavioral place requirement; graphical representation of the timing behavior of c_0 (b) and c_1 (c).

the behavioral place is marked, it must be the case that a sufficient number of constraint places must also be marked such that a transition in the postset of the behavioral place is enabled. Fortunately, the net transformation procedure described above restricts the use of constraint places such that this is always the case. If, however, a designer adds additional constraint places to check various timing properties, then it is possible that the dynamic restriction may be violated.

D. Reachability graphs and state graphs

The goal of timed state space exploration is to find the set of reachable states for the system being analyzed. The reachable untimed state space for a TPN can be represented as a *reachability graph* (RG). A RG is a graph in which its vertices are untimed states (i.e., markings) and its edges are possible *state transitions*. A RG is modeled by the tuple $\langle \Phi, \Gamma \rangle$ where Φ is the set of states and $\Gamma \subseteq \Phi \times T \times \Phi$ is the set of edges.

A *state graph* (SG) is a RG in which the states have been labeled with bitvectors corresponding to the binary values of each of the systems signals in that state, and the

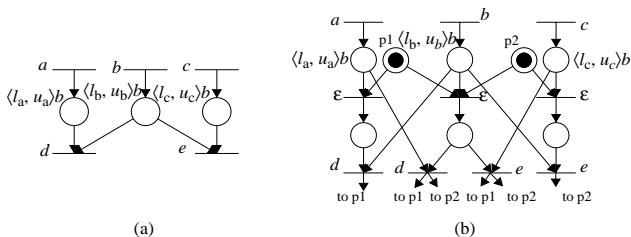


Fig. 3. (a) A choice place; (b) an orbital net that satisfies the single behavioral place requirement.

transitions are labeled with the signal which transitions to move between the two states. A SG is modeled by the tuple $\langle S, \Phi, \Gamma, \lambda_\Phi, \lambda_T \rangle$ where S is the set of signals, $\lambda_\Phi : \Phi \rightarrow (S \rightarrow \{0, 1\})$ is the state labelling function, and $\lambda_T : T \rightarrow S$ is the transition labelling function. S can be further partitioned into the set of input signals I and output signals O . There exists a SG for a corresponding RG if and only if there is a *consistent state assignment*. A consistent state assignment exists if each state and transition can be labeled such that between two states in a state transition the only signal to change value is the one which experienced the transition. In other words, for a RG, $\langle \Phi, \Gamma \rangle$, and a set of signals, S , a SG exists if there exists a λ_Φ and λ_T such that

$$\forall (M, t, M') \in \Gamma. \forall u \in S. (\lambda_T(t) \neq u \wedge \lambda_\Phi(M)(u) = \lambda_\Phi(M')(u)) \vee (\lambda_T(t) = u \wedge \lambda_\Phi(M)(u) \neq \lambda_\Phi(M')(u))$$

For synthesis, it is useful to be able to determine in which states a signal is untimed enabled to rise or fall. The sets *rise*(u) and *fall*(u) provide this information and are defined as follows:

$$\begin{aligned} \text{rise}(u) &= \{M \in \Phi \mid \lambda_\Phi(M)(u) = 0 \wedge \\ &\quad \exists t \in T_e(M). \lambda_T(t) = u\} \\ \text{fall}(u) &= \{M \in \Phi \mid \lambda_\Phi(M)(u) = 1 \wedge \\ &\quad \exists t \in T_e(M). \lambda_T(t) = u\} \end{aligned}$$

An *excitation region* for signal u is defined as a maximally connected subset of either *rise*(u) or *fall*(u). If it is a subset of *rise*(u), it is called a *set region*, and it is denoted $ER(u\uparrow, k)$ where k indicates that it is the k^{th} set region. Similarly, a *reset region* can be denoted $ER(u\downarrow, k)$. For each signal u , there are two sets of stable, or *quiescent states*. There is the set of states where the signal u is stable high denoted $QS(u\uparrow)$ (i.e., $QS(u\uparrow) = \{M \in \Phi \mid \lambda_\Phi(M)(u) = 1 \wedge M \notin \text{fall}(u)\}$), and the set where it is stable low denoted $QS(u\downarrow)$ (i.e., $QS(u\downarrow) = \{M \in \Phi \mid \lambda_\Phi(M)(u) = 0 \wedge M \notin \text{rise}(u)\}$).

A portion of the timed Petri net and SG for a controller for a port selector (SEL) is shown in Figure 4. The SEL example accepts data and a port selection and forwards the data out the selected port. In the initial state, all signals are low and $xfer_i$ is enabled to rise. After $xfer_i$ rises, sel_o and $data_o$ become enabled to rise. Assuming that $data_o$ rises first, in the next state $data_i$ and sel_o are both enabled to rise. However, the maximum delay for sel_o to rise is 20 while the minimum delay for $data_i$ to rise is 40. Therefore, the only possible next transition is for sel_o to rise, and the timing information has removed a possible state transition. The final state graph obtained for the SEL contains 53 states. A state graph generated ignoring all the timing information contains 256 states. The size of the SG and the complexity of the circuitry are strongly correlated. For the SEL example, our synthesis procedure derives a gate-level timed circuit implementation with 27 literals shown in Figure 5(a). If all the timing information is ignored, synthesis produces a gate-level speed-independent circuit implementation with 44 literals shown in Figure 5(b). Besides being nearly 40 percent smaller, the timed circuit has reduced

nonzero fractional component; the diagonal line segments are the cases where both clocks have the same nonzero fractional component, and the interior triangles are the cases where each clock has a distinct fractional component.

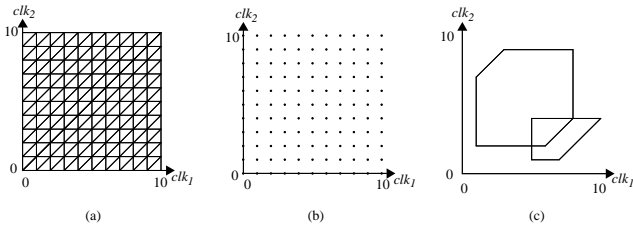


Fig. 7. (a) Unit-cube, (b) discrete, and (c) geometric representations of the timed state space.

Let us assume the number of distinct untimed states in an orbital net is $|S|$. If the maximum value of any non-infinite timing requirement is k , and there are at most n marked places in the net in any state (this value is trivially bounded by the size of the safe net), the worst-case size of the state space for Alur's method is asymptotically [10],

$$|S| \frac{n!}{\ln 2} \left(\frac{k}{\ln 2} \right)^n 4^{1/k}.$$

While it is sufficient to store only those equivalence classes that occur after the firing of a transition, the number of such equivalence classes still explodes in practice. For our example, there are a total of 2,463 unit-cube equivalence classes in the total timed state space. Of course, if the timing values are increased, this number rises significantly.

It has been proven, however, that the general unit-cube technique is unnecessary for orbital nets since considering only integer event times gives a full characterization of the continuous-time behavior [10] (this proof is similar to one given by Henzinger, et. al. in [13] for timed transition systems). In other words, only timed states associated with each discrete-time instance, represented as a point for the two-dimensional case in Figure 7(b), need be considered. This technique is used by Burch for verifying timed circuits [8], and has a worst-case state space size of $|S| (k+1)^n$ which is better than the unit-cube method by more than $n!$. The essential restriction in orbital nets that allows this optimization is that clocks are only compared with constants using ' \leq ' or ' \geq ', rather than ' $<$ ' or ' $>$ '; this can be justified for real systems with the imprecise nature of physical time. State space exploration in discrete time is a simplification of that in Alur's unit-cube method in that the ordering relation of the fractional components of the clock is dropped, as is the possibility of time advancing in non-integral units. Our example yields a total of 487 discrete timed states; again, this number rises significantly as the timing requirements increase.

Both unit-cube and discrete-time methods, however, are of little more than theoretical interest because the size of the state space increases exponentially with the concurrency in the net. For a circuit with timing values accurate to two significant digits, with up to six independent concurrent pending events, the state space is easily in excess

of 10^{12} states—well beyond the capabilities of most finite-state synthesis and verification techniques.

Other efficient techniques have been developed [14], [15], [16], to find a single exact time separation between two events in various types of specification methods. However, each time a separation is needed during state space exploration, these techniques reanalyze the complete graph, so using them to compute all of the possible separations in a graph is slow making them impractical for timed state space exploration.

Another approach to represent timed states is to use convex *geometric regions* (or zones) as shown in Figure 7(c). Even though the worst-case performance is much worse than either the unit-cube or the discrete-time approaches, this approach usually performs well in practice. Dill [17], Lewis [18], and Berthomieu and Diaz [19] originated geometric timing, and it has become an active area of research [20], [21], [22]. Unfortunately, as illustrated later, for highly concurrent systems, geometric methods alone can result in a substantial state explosion.

A number of techniques have been proposed to deal with state explosion in highly concurrent systems. For untimed systems, stubborn sets [23], partial orders [24], and unfoldings [25] have been shown to perform well. These techniques reduce the number of states explored by considering only a subset of the possible interleavings between events. These approaches have been extended to timed systems in [26]. This algorithm, however, reduces verification time by exploring only part of the timed state space. This may limit the timing properties that can be verified. Furthermore, the entire timed state space is needed for timed circuit synthesis. While reducing the number of interleavings is useful, one region is still required for every firing sequence explored to reach a state. If most interleavings need to be explored, this technique could still result in state explosion.

This section describes POSET timing which improves upon the geometric methods by making use of concurrency and causality information. This is accomplished through the exploration of *partially ordered sets of events* as opposed to linear sequences.

A. Geometric regions

Rather than consider at each step a single discrete-time state or a minimum equivalence class of timed states, the geometric timing method considers a larger set of timed states in parallel. Specifically, convex geometric regions of timed states are used to represent the timed state space. These regions are described by a set of constraints which are either lower and upper bounds on specific clock values or differences between pairs of specific clock values. These constraints are usually encapsulated in a matrix (sometimes known as a *difference bound matrix*), where the constraints on clocks $\{clk_1, \dots, clk_n\}$ are of the form $clk_i - clk_j \leq a_{ji}$. A fictitious clock, clk_0 , that is always exactly zero is introduced so that upper and lower limits on a particular clock can be represented in the same form [17].

For any convex region that can be represented by such

a matrix, there are many matrices that represent the same convex region. Fortunately, a canonical matrix can be obtained using Floyd’s all-pairs shortest-path algorithm to maximally tighten all the inequalities [17]. While in general Floyd’s algorithm runs in time $O(n^3)$, since only incremental changes are made to the matrix during analysis, specializations of Floyd’s algorithm that run in time $O(n^2)$ suffice [10]. For large systems, this optimization can speed execution substantially.

B. State space exploration with geometric timing

Each geometric region can be considered as an infinite set of timed states which are operated on in parallel. In order to perform state space exploration using geometric timing, this section redefines the operational semantics of orbital nets in terms of these geometric regions as opposed to individual timed states. The aspects of state space exploration that do not consider time are not discussed, since they are the same in both cases. This section describes how these operations work for a single step in a timed sequence, assuming it works for the predecessor sequence; the trivial base case and structural induction on sequences completes the proof that these operations work for all sequences.

Figure 8 shows how geometric timing works on our example. The first column shows the untimed state, the second shows the geometric region before advancing time, and the third shows the region after advancing time. The regions are shown both as constraint matrices and regions in space. The 0th row and column of each matrix represent the time relationships between the clocks on the places and the fictitious clock, clk_0 , which is always 0. In other words, the 0th row is the maximum value of the corresponding clock, and the 0th column is the negative of the minimum value. The other entries represent time relationships between the clocks. For example, in row 1, the third column and second row for the first matrix represents the relationship $clk_a - clk_s \leq 8$.

The initial state of our example has tokens in $(P_a, P_b, P_{f_i}, P_{s_i})$, with clock values of 0 for the tokens in P_a and P_b . Before advancing time, the region is represented as a single point at the origin, as shown in the leftmost region for the initial state in Figure 8. In our original operational semantics, advancing time involves adding some number t to all clocks. For geometric regions, advancing time involves extruding the geometric region in the $clk_1 = clk_2 = \dots = clk_n$ direction, subject to *max-advance*, which itself is a convex region. To perform this operation, the algorithm simply sets the maximum bounds on each clock (the first row of the constraint matrix) to their maximum values (if the places are behavioral places) or infinity (if the places are constraint places). The algorithm then recanonicalizes the matrix in time $O(n^2)$. The resulting matrix and geometric region are shown in the rightmost columns of Figure 8. In row 0, the upper bound on P_a is 8 and P_b is 10. Since both of their clocks are initially zero, *max-advance* returns 8.

Determining whether a particular transition is timed-enabled in our original operational semantics entails comparing the clocks with the timing requirements. With ge-

ometric regions, the timing analysis algorithm determines the subset of the timed states in the region for which the particular transition is enabled; this is called the *enabling region*. This can be calculated by introducing the enabling conditions on the transition selected for firing as additional constraints on the region, and recanonicalizing. For orbital nets, these conditions are always a single new minimum value on that behavioral place, and the algorithm can recanonicalize in time $O(n^2)$. The dashed lines on the geometric diagrams in Figure 8 show, for each behavioral place, the minimum firing time constraint added by that place. Thus, for example, in the initial region, any transition in the postset of either P_a or P_b can fire, since introducing the minimum firing times in either case produces a non-empty region.

After selecting an enabled transition, firing that transition involves removing some set of clocks and introducing new clocks initialized to zero. With geometric regions, removing these clocks involves projection of the system of constraints to eliminate a particular set of variables, and introducing new clocks is done by adding a new set of variables equal to zero. For example, to obtain the geometric region in row 1 (before advancing time), the algorithm introduces the minimum firing constraint for P_a then projects the region onto the y axis (representing the elimination of the token from P_b). The algorithm then introduces a new clock for the token in place P_s , and initializes it to zero. At this point, time is again advanced by setting the maximum bounds on each clock to their maximum values. In this region, $af\uparrow$ is enabled because there is some subset of the region above the dotted line representing the token in place P_a , but $bs\downarrow$ is not enabled because the introduction of the minimum firing time of place P_s yields an empty region.

While unit-cube and discrete-time methods operate on timed firing sequences, geometric timing operates over untimed firing sequences. For each untimed firing sequence that the orbital net might execute, geometric timing computes directly the full set of reachable states of all possible timed sequences that have the same underlying untimed sequence.

There are many different options for storing the geometric regions and deciding when to backtrack. In general, for every explored untimed state, there are one or more geometric regions that have been seen for that state. For instance, the untimed firing sequence $[bs\uparrow]$ leads to the same untimed state as $[bs\uparrow, af\uparrow, af\downarrow]$ (compare rows 1 and 3 in Figure 8), yet have different geometric regions. One option, perhaps the simplest, is to simply hash each geometric region into the state table and only backtrack when an identical geometric region is seen. For our example, the total number of geometric regions for the 7 reachable untimed states using this technique is 234, for an average of 33 regions per untimed state. The number of transitions fired in our depth-first exploration was 518.

Another option for state space exploration is to maintain a list of seen geometric regions for each untimed state, and compare each new geometric region against the entire list using the subset operation. It is possible to check if one ge-

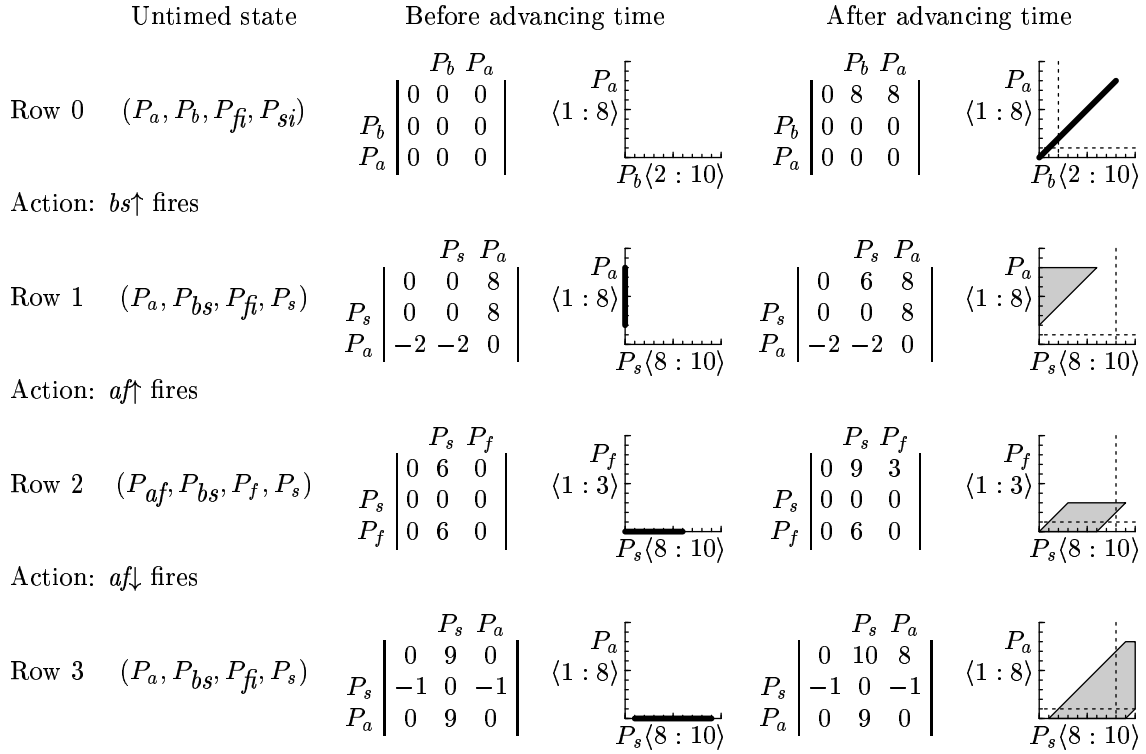


Fig. 8. Example geometric region state space exploration for the sequence $[bs\uparrow, af\uparrow, af\downarrow]$.

ometric region is a subset of another in time $O(n^2)$. Then, the search can backtrack when a newly found region is a subset of a previously seen region. Conversely, if the newly found region is a superset of a previously seen region, then any pending state exploration from the previously seen region can be canceled, and the previously seen region can be removed from the list. In our example, using this technique, 20 different geometric regions are found for the 7 untimed states in 78 transition firings.

Many other optimizations are possible and have been explored by the authors; some of them yield reasonable savings in runtime, but usually the savings in runtime is less than a factor of two; in the face of the combinatorial explosion of timed state space exploration in general, these improvements are typically too small to be worth detailing in this paper.

C. Drawback of geometric timing

While geometric timing can be very efficient, with more concurrent examples, such as the adverse example, *adv4x40*, shown in Figure 9, the number of geometric regions can rise astronomically. While only having a single untimed state, standard geometric timing techniques generate an incredible 219,977,777 distinct geometric regions. This is more than the number of discrete-time states!

The main difficulty with all of the geometric timing techniques is the large number of geometric regions that can exist for each untimed state. The POSET timing technique introduced next tends to dramatically reduce the number of geometric regions for each untimed state, typically down

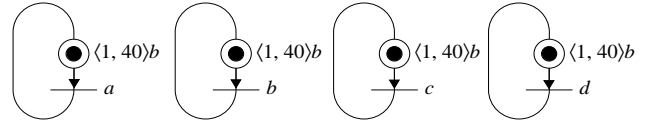


Fig. 9. The adverse example *adv4x40* with $n = 4$ and $k = 40$.

to an average of one or two.

D. Concurrency, causality, and posets

The major source of blowup in the adverse example is the way the standard geometric timing algorithm calculates the set of timed states reachable from a sequence of transition firings; the transition firings are linearly ordered, even if they are concurrent in the system being evaluated. That is, if two concurrent transitions start clocks, the constraints between the two clocks reflect the linear order that the transitions are fired in the original sequence. For example, when the geometric timing algorithm analyzes the untimed firing sequence $[a, b]$, it obtains the upper geometric region shown in Figure 10, and when the algorithm considers the sequence $[b, a]$, it obtains the lower geometric region. In general, if there are n concurrent transitions that reset clocks visible in the resulting timed state, there are $n!$ different sequences that need to be considered, each of which leads to a distinct geometric region. For this reason, it is important to distinguish the causal ordering of transitions from the non-causal ordering that comes about from the selection of a particular firing sequence.

To solve this problem, our POSET timing algorithm con-

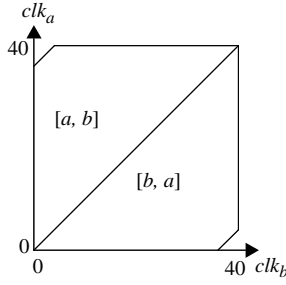


Fig. 10. Geometric regions from the adverse example.

constructs a partially ordered set, or *poset*, for each untimed firing sequence which is represented with an acyclic, choice-free unfolding of the original orbital net. The poset reflects the concurrency and causality inherent in the firing sequence. Initially, the unfolded net representing the poset contains a single transition with places in its postset corresponding to each initially marked place. Transitions are added in the same order as they occur in the firing sequence. For each transition in the firing sequence, a correspondingly labeled transition is added to the unfolded net. A set of arcs into the transition are connected from the most recently added places in the unfolded net corresponding to places in the preset of the transition in the original orbital net. Finally, a new set of places corresponding to the places in the postset of the transition in the original net are added, and these places are connected to the new transition. Every place and every transition in the unfolded net, except the first, correspond to some place and some transition in the original net. Every place and every transition in the original net correspond to zero or more places and transitions in the unfolded net.

A poset explicitly represents the concurrency in a particular firing sequence. That is, a particular poset corresponds to many different firing sequences that differ only in the interleavings of concurrent transitions; every such firing sequence fires the same set of transitions and leads to the same final untimed state. For example, the poset represented with the unfolded net shown in Figure 11 corresponds both to the sequence $[a, b]$ and to the sequence $[b, a]$.

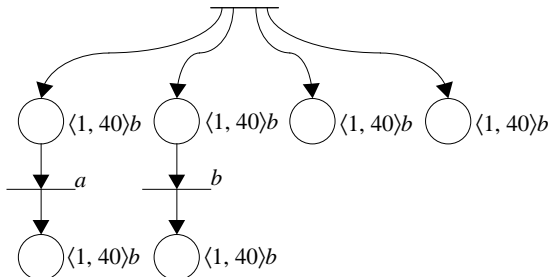


Fig. 11. One poset from the adverse example.

E. State space exploration with POSET timing

State space exploration proceeds just as it does for the geometric method, except that, for each sequence, the algorithm constructs the corresponding unfolded net. With depth-first search, this is done incrementally. The algorithm also incrementally calculates a poset matrix that stores the firing time relationship among the transitions. For each place p in the poset, there is a unique occurrence of a transition in its preset, $(\bullet p, i)$, and its postset, $(p\bullet, j)$. Note that i and j are the occurrence indices for these transitions. For each constraint place p , the constraint $\tau(\bullet p, i) \leq \tau(p\bullet, j)$ is introduced where the function $\tau()$ is the time of the occurrence of the given transition. For each behavioral place p in the resulting unfolded net with a timing requirement of $\langle l, u \rangle b$, two constraints are introduced. The first reflects the minimum separation, $\tau(\bullet p, i) - \tau(p\bullet, j) \leq -l$. The second reflects the maximum separation, $\tau(\bullet p, i) - \tau(p\bullet, j) \leq u$. All constraints introduced in this fashion for a given unfolded net must be satisfied. This poset matrix can then be used to produce a geometric region which after canonicalizing represents the full set of reachable states for the poset corresponding to the unfolded net. Applying this procedure to the unfolded net shown in Figure 11, the POSET timing algorithm obtains at once the geometric region which encloses both regions shown in Figure 10.

Figure 12 shows a poset from our example, and Figure 13 shows the geometric regions found by the evaluation of this poset. In the initial state, only the initial *reset* transition has fired, leading to the initial marking. The poset matrix is a singleton 0 which states that the maximum time separation between *reset* and itself is 0. In general, the poset matrix is extended by adding the new constraints from the firing transition and recanonicalizing. The constraint matrix has three components: minimum values for each marked timed place, maximum values for each marked timed place, and constraints on the differences between two marked places. To compute the constraint on the difference between two marked places, the algorithm copies the constraint on the difference between the firing times of the two transitions in their presets (which may be the same transition) from the poset matrix. The minimum values for newly marked timed places are set to zero; the minimum values for timed places that retain their token are copied from the enabling region of the previous state. The maximum values are computed as before; the maximum delay for behavioral places and infinity for constraint places. The algorithm can then canonicalize the resulting matrix with a specialization of Floyd's algorithm in time $O(n^2)$; this is the time-extruded set of reachable states for all transition sequences that share this partial order.

Returning to our example, to fire $bs\uparrow$ from the initial state, the algorithm extends the poset with the additional transition. Then, the algorithm computes the enabling region as before; this provides the minimum values for the timed places that retain their token. The algorithm then extends the poset matrix; the empty matrix is extended with the bounds on the separation between the *reset* tran-

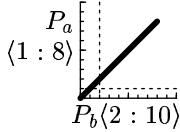
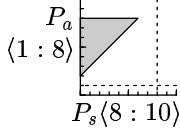
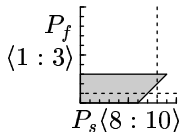
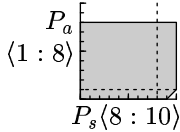
	Untimed state	POSET matrix	Before canonicalization	After canonicalization	
Row 0	$(P_a, P_b, P_{fi}, P_{si})$	$\begin{array}{c c} & \text{reset} \\ \text{reset} & 0 \end{array}$	$\begin{array}{c c} P_b & P_a \\ \hline 0 & 10 & 8 \\ P_b & 0 & 0 & 0 \\ P_a & 0 & 0 & 0 \end{array}$	$\begin{array}{c c} P_b & P_a \\ \hline 0 & 8 & 8 \\ P_b & 0 & 0 & 0 \\ P_a & 0 & 0 & 0 \end{array}$	
	Action: $bs\uparrow$ fires				
Row 1	$(P_a, P_{bs}, P_{fi}, P_s)$	$\begin{array}{c c} & \text{reset } bs\uparrow \\ \text{reset} & 0 & 10 \\ bs\uparrow & -2 & 0 \end{array}$	$\begin{array}{c c} P_s & P_a \\ \hline 0 & 10 & 8 \\ P_s & 0 & 0 & 10 \\ P_a & -2 & -2 & 0 \end{array}$	$\begin{array}{c c} P_s & P_a \\ \hline 0 & 6 & 8 \\ P_s & 0 & 0 & 8 \\ P_a & -2 & -2 & 0 \end{array}$	
	Action: $af\uparrow$ fires				
Row 2	$(P_{af}, P_{bs}, P_f, P_s)$	$\begin{array}{c c} & \text{reset } bs\uparrow af\uparrow \\ \text{reset} & 0 & 10 & 8 \\ bs\uparrow & -2 & 0 & -6 \\ af\uparrow & -1 & 9 & 0 \end{array}$	$\begin{array}{c c} P_s & P_f \\ \hline 0 & 10 & 3 \\ P_s & 0 & 0 & 9 \\ P_f & 0 & 6 & 0 \end{array}$	$\begin{array}{c c} P_s & P_f \\ \hline 0 & 9 & 3 \\ P_s & 0 & 0 & 3 \\ P_f & 0 & 6 & 0 \end{array}$	
	Action: $af\downarrow$ fires				
Row 3	$(P_a, P_{bs}, P_{fi}, P_s)$	$\begin{array}{c c} & bs\uparrow af\uparrow af\downarrow \\ bs\uparrow & 0 & 6 & 9 \\ af\uparrow & 9 & 0 & 3 \\ af\downarrow & 8 & -1 & 0 \end{array}$	$\begin{array}{c c} P_s & P_a \\ \hline 0 & 10 & 8 \\ P_s & 0 & 0 & 8 \\ P_a & 0 & 9 & 0 \end{array}$	$\begin{array}{c c} P_s & P_a \\ \hline 0 & 10 & 8 \\ P_s & 0 & 0 & 8 \\ P_a & 0 & 9 & 0 \end{array}$	

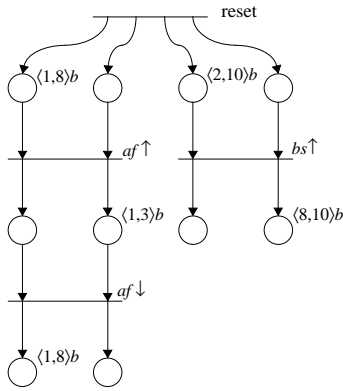
Fig. 13. Example poset state space exploration for the sequence $[bs\uparrow, af\uparrow, af\downarrow]$.

Fig. 12. A poset for the consumer/resource example.

sition and the transition $bs\uparrow$, derived from the behavioral timing place $[2, 10]$. The resulting matrix is shown in row 1 of Figure 13.

Compare row 2 between Figure 8 and Figure 13 to see that already, after two firings, the reached states are larger for the POSET method; after three firings, the difference is even more dramatic. Unlike in the geometric region method, the geometric region found in row 3 is a superset of that found in row 1, so all further execution from row 1 can be canceled in the POSET timing method. For our example, the POSET timing method can represent the full set of reachable timed states with only 9 geometric regions after only 26 transition firings.

The POSET timing algorithm is shown in Figure 14.

The algorithm begins with an initial state composed of the initial marking, geometric region, and poset matrix. Next, it calculates the set of timed enabled transitions, and it selects one to fire. The algorithm then restricts the region to the subset where the transition is enabled to fire, canonicalizes this new region, and checks if any constraint places for this transition have not met their lower bounds. Next, the algorithm updates the marking and poset, and computes the new geometric region. The clocks in this new region are then allowed to advance subject to *max-advance*, and the region is canonicalized and normalized. Normalization is the step which keeps the state space finite by taking care of infinite upper bounds. If in the new region any constraint place is able to exceed its upper bound, a failure is reported. The new timed state is then checked to see if it has been seen before. If it has not, then it is added to the state table, and a new set of timed enabled transitions are computed. If it has been seen before, then a timed state is popped off the stack. For the next timed state, the set of untimed enabled transitions are computed, and if any marked behavioral does not have an untimed enabled transition in its postset, a failure is reported. If the stack is empty, then the state space exploration is complete.

Note that POSET timing still explores every reachable untimed state and every reachable timed state. This allows us to verify arbitrary timing properties, and to use the resulting state graph to synthesize a timed circuit.

Our examples enable only two clocks at a time so the regions can be drawn in 2-dimensions. The improvements

Algorithm III.1 (POSET timing)

```

RG POSET_timing(orbital net  $N = \langle P, T, F, M_0, \Delta \rangle$ ) {
   $TS = M_0 \times R_0 \times Q_0$ ;
   $M = M_0$ ;  $R = R_0$ ;  $Q = Q_0$ ;
   $\Phi = \{TS\}$ ;
   $T_t = \text{find\_timed\_enabled}(N, TS)$ ;
  done=false;
  while ( $\neg$ done){
     $t = \text{head}(T_t)$ ;
    push( $TS, \text{tail}(T_t)$ );
     $R = R \cup \{clk_i \geq l_i\}$  where  $p_i \in (\bullet t \cap B)$ 
     $R = \text{canonicalize}(R)$ ;
    if ( $\exists p_i \in \bullet t \cap C . clk_i < l_i$ ) then return failure;
     $M = \text{update\_marking}(N, M, t)$ ;
     $Q = \text{update\_poset}(N, Q, t)$ ;
     $R = \text{compute\_region}(N, Q)$ ;
     $R = \text{advance\_time}(N, R)$ ;
     $R = \text{canonicalize}(R)$ ;
     $R = \text{normalize}(R)$ ;
    if ( $\exists p_i \in (M \cap C) . clk_i > u_i$ ) then
      return failure;
     $TS_{new} = M \times R \times Q$ ;
    if ( $TS_{new} \notin \Phi$ ) then
       $\Phi = \Phi \cup \{TS_{new}\}$ ;
       $\Gamma = \Gamma \cup \{(TS, t, TS_{new})\}$ ;
       $TS = TS_{new}$ ;
       $T_t = \text{find\_timed\_enabled}(N, TS)$ ;
    else
      if (stack is not empty) then ( $TS, T_t$ ) = pop();
      else done = true;
    if ( $\exists p \in B \cap M . p \bullet \cap T_e(M) = \emptyset$ ) then
      return failure;
  }
  return  $\langle \Phi, \Gamma \rangle$ ;
}

```

Fig. 14. POSET timing algorithm.

due to POSET timing are much more dramatic with many more simultaneous clocks due to the $n!$ potential orders in which those clocks can be enabled. In fact, the POSET method typically reduces the average number of timed regions for each untimed state to a value close to one. For the adverse example in Figure 9, POSET timing obtains exactly one geometric region corresponding to the one untimed state.

F. Efficiency considerations

The number of transitions in the unfolded net is equal to the length of the firing sequence plus one, and it increases with the depth of our search. Calculating the minimum separations between the occurrence times in the unfolded net, even with our incremental $O(n^2)$ approach, becomes prohibitively expensive as the firing sequence lengthens. In addition, the algorithm needs a poset matrix for each step; this would require a tremendous amount of storage during depth-first search.

To keep n bounded as the depth of our search increases,

the algorithm determines what prefix, if any, of the unfolded net can safely be ignored. The algorithm can eliminate any transitions that no longer affect future calculations. In general, the algorithm can eliminate a variable from any set of equations or inequalities whenever it has produced the full set of equations or inequalities that use that variable. Since all constraints introduced through the firing of a transition are associated with places connecting the new transition to the old, once a transition in the unfolded net no longer has any marked places in its postset, it is eliminated from the poset matrix. Thus, our n is—at most—the number of marked places in the original net at any given time, plus one for the current transition.

As with geometric timing, POSET timing backtracks whenever a new geometric region is a subset of a previously seen geometric region. POSET timing also takes advantage of the fact that if the newly found region is a superset of a previously seen region, then any pending state space exploration from the previously seen region can be canceled, and the previously seen region can be removed from the list.

In addition, a hash table of canonicalized sequences of posets that have been explored can be kept; by comparing each potential extension of the poset against the previously seen ones, we can avoid visiting the same poset multiple times. Performing this check is usually significantly faster than performing all the timing calculations, so this can save execution time at the expense of memory.

IV. APPLICATIONS

This section describes the application of POSET timing to the synthesis and verification of timed circuits.

A. Synthesis

Synthesis is the process of transforming a specification into a circuit implementation. Our synthesis procedure begins with a specification in high-level language from which a hazard-free timed circuit implementation is generated using only basic gates such as AND gates, OR gates, and C-elements. After the specification is translated to an orbital net representation, the POSET timing algorithm is used to find the set of reachable states. Our synthesis procedure is briefly described here. For a more complete description, please see [27], [28].

From the resulting SG, there are several different approaches that could be used to obtain a gate-level timed circuit implementation. The first approach is to use a traditional boolean minimization technique directly. Unfortunately, if the logic is mapped to basic gates and the delays of these gates are considered individually, the implementation may be hazardous. Another approach is to split the design of the rising and falling transitions to obtain a *generalized C-element* implementation [1] and decompose it to basic gates. The basic structure is depicted in Figure 15(a) in which the upper sum-of-products represents the logic for the set, the lower sum-of-products represents the logic for the reset, and the result is merged with a C-element. This can be implemented directly in CMOS as a single compact

gate with weak-feedback as shown in Figure 15(b) or as a fully-static gate as shown in Figure 15(c). This technique alleviates some of the hazard problems, but it may still be hazardous when mapped to basic gates. To address this problem, after a generalized C-element implementation is produced and decomposed to basic gates, the design could be back-annotated with delays from the gate library, and the circuit could be verified. While this may often work, it is not clear what to do in the cases in which a hazard does exist. Also, a hazard is a spurious transition which wastes power and does no useful work. In a power efficient implementation, it is desirable to have logic which is hazard-free both internally and externally.

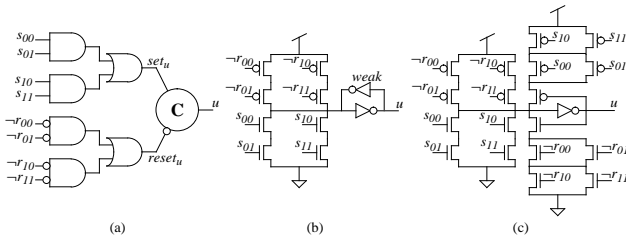


Fig. 15. (a) The generalized C-element configuration with (b) weak-feedback and (c) fully-static CMOS implementations.

To avoid the hazard concerns discussed above, we take a *standard C-implementation* approach in which each rising and falling region for each output signal is implemented using an atomic gate (often a single *cube* or AND gate), which must satisfy certain correctness constraints. While the general structure of the standard C-implementation is similar to the generalized C-element structure shown in Figure 15(a), each set or reset cube is implemented with an atomic gate that must satisfy certain constraints to guarantee that the merged implementation is a gate-level hazard-free circuit. The approach is conservative in that timing analysis may show that the decomposed generalized C-element implementation is sufficient, but the overhead required tends to be small to get a safe implementation that is free of internal hazards. It has been observed that the atomic gate is often only a single cube. In [28], 23 of the 27 speed-independent benchmarks had a single cube implementation. In at least one of the four that does not have a single cube implementation, one exists when realistic timing numbers are incorporated into the design of a timed circuit. In [28], a single-cube algorithm is described that is typically over an order of magnitude faster than the general algorithm used for multi-cube atomic gates. Both algorithms have been implemented for timed circuits. For descriptions of the algorithms, please see [28]. This subsection describes the theory and justifies its correctness for timed circuits.

The cover of a set region $C(u\uparrow, k)$ (or a reset region $C(u\downarrow, k)$) is a set of states for which the corresponding atomic gate in the implementation evaluates to one. In order for a cover to lead to a hazard-free implementation, it must satisfy certain *correctness constraints* [29], [28]. These constraints guarantee that any gate in the implementation only changes when it is actively driving the output signal

to change. This ensures that the transition of the gate is *acknowledged*.

First, a correct cover needs to satisfy a *covering constraint* which says that the reachable states in the cover must include the entire excitation region but must not include any states outside the union of the excitation region and associated quiescent states, i.e.,

$$ER(u*, k) \subseteq [C(u*, k) \cap \Phi] \subseteq [ER(u*, k) \cup QS(u*)]$$

where “*” indicates either “ \uparrow ” for set regions or “ \downarrow ” for reset regions.

Second, the covers of each excitation region must also satisfy an *entrance constraint* to ensure hazard-freedom. This constraint says that the cover must only be entered through excitation region states, i.e.,

$$\begin{aligned} [(s, t, s') \in \Gamma \wedge s \notin C(u*, k) \wedge s' \in C(u*, k)] \\ \Rightarrow s' \in ER(u*, k) \end{aligned}$$

The definition of correct covers is based on the definition given for speed-independent circuits in [29], [28]. Since then, other researchers have come up with similar correctness constraints for their speed-independent design methodologies [30], [31]. Our definition of correct covers differs slightly from the one in [29], [28] in that $QS(u*)$ does not need to be a maximal connected set of states. It is proven in the appendix that this condition is made redundant by the entrance constraint. A concern may also be raised that a quiescent state may be reachable from multiple excitation regions; this state would be eliminated from any correct cover by the entrance constraint which is another implication of the proof in the appendix.

The correctness of our timed circuits is a direct result from the proof for the speed-independent case given in [28]. In [28], it is proven that a standard C-implementation that satisfies these correctness constraints operates correctly regardless of the delay of the gates in the implementation and the environment. In other words, all delays are unknown and fall in the range from $\langle 0, \infty \rangle$. In our case, the delays of the gates and environment are known. As for the gate delays, it has already been proven that these correctness constraints guarantee correctness regardless of the delay of the gates, so knowing the delays of the gates does not change that. This delay information, however, has also been used by the POSET timing algorithm to find all possible interleavings between both the input and output signals (i.e., the SG). In other words, the timing information has limited what signal orderings are possible. As long as the states in the reduced state graph *are* the only reachable states, the correctness constraints guarantee correct circuit operation. Therefore, the only concern would be that the delays of the actual circuit implementation allow additional states. As long as the minimum and maximum delay of the standard C-implementation falls in the given range, there is no problem. Checking this is the subject of the next subsection on timed circuit verification.

The synthesis procedure using the POSET timing algorithm to find the reachable state space has been fully

TABLE I
EXPERIMENTAL RESULTS.

Ex.	$ \Phi $	Timed				Other Design Methodologies						
		gC Lit	Lit	Area	Del	$ \Phi $	Area	Del	SIS Area	Del	3D Lit	Del
SEL	53	25	27	104	5	256	160	7	158	11	n/a	n/a
SEL2	36	19	21	76	5	128	108	6.5	130	11.5	n/a	n/a
MMU	187	56	62	210	4.5	23,296	412	10	out of memory		n/a	n/a
DRAM	79	38	38	110	5.5	n/a	n/a	n/a	n/a	n/a	46	7
TSBM	113	32	33	140	4.5	n/a	n/a	n/a	n/a	n/a	58	7.5

the sequencing of the states. For example, if a sequence of states in which the counter is counting 00-11-01-10 is possible, this circuit would generate the correct next state given the current state. This extra logic, however, is unnecessary since this counter always goes through the states in the same order: 00-01-10-11-00, etc.

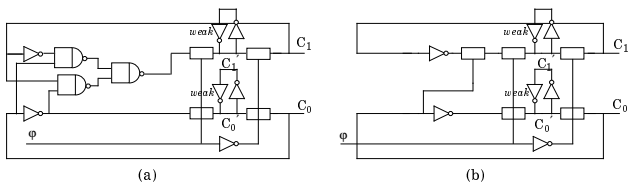


Fig. 17. Implementation of a two-bit synchronous counter derived using (a) SIS and (b) ATACS.

B. Verification

Verification is the process of checking if the synthesized circuit satisfies its specification. Our verification procedure requires both a specification and circuit implementation either given in or translated to an orbital net representation. The orbital net for the specification is *mirrored* (i.e., inputs and outputs are swapped) [37] and composed with the orbital net for the implementation. The state space is then explored using the POSET timing analysis algorithm described earlier. If a failure is detected in the process of exploring the state space, an error trace is returned, otherwise the timed circuit is found to implement its timed specification. In order to verify timed circuits, our verification procedure adopts trace theory as defined by Dill [37] as its behavioral semantics, as well as Burch [8] extensions to trace theory semantics for timed circuits. Our verification procedure provides structural constructions and syntactic shorthands for labeled safe Petri nets that correspond to the behavioral semantics operations.

To verify that our synthesized timed circuits implement their timed specifications, our verification procedure begins with the specification in a high-level language and the implementation given as a netlist of basic gates. The specification is translated to an orbital net representation in which the timing requirement for each behavioral place in the preset of an output transition is changed to a constraint place with timing requirement $\langle 0, \infty \rangle_c$. These constraints must be satisfied by the timed circuit implementa-

tion. Note that a $\langle 0, \infty \rangle_c$ timing requirement is used rather than the delays given in the original behavioral place, since circuits with delays outside this range may function correctly. For $\langle 0, \infty \rangle_c$ constraint place to be satisfied, it is necessary for this place to be marked whenever a transition in its postset has its behavioral place marked. In other words, if the circuit produces an output transition, the specification must be in a state in which it is willing to accept that transition. If a performance constraint is desired, additional constraint places with finite delay ranges can be added as well.

For each gate in the implementation, an orbital net is constructed corresponding to an instantaneous function block such as the one given for the AND gate in Figure 18(b). This net is composed with a delay element such as the one in Figure 18(c) with the behavioral timing requirement set by the delay given in the gate library. The behavioral place labeled $\langle 2, 4 \rangle$ indicates that an output occurs between 2 and 4 time units after the preceding input occurs; no behavior violating this requirement are generated by the net. The constraint places do not constrain the behavior of the net, but if another input event occurs before the preceding output event then the environment violates the specification. Composition of these nets gives an AND gate operating under the output delay model. In a similar manner, an AND gate operating under the *input delay model* could also be obtained. The delay model shown in Figure 18(c) is relatively simple, and it suffices for many types of circuits. More complex delay models can and have been constructed, modeling more accurately the behavior of a gate under hazard conditions (for example, one which models inertial delay); for these, the separation of gate models into combinational function and delay behavior is essential [10]. Each orbital net in the implementation is composed with the other orbital nets as dictated by the connections in the netlist.

To determine if a timed circuit implements its timed specification, the reachable state space is found using the POSET timing algorithm for the orbital net obtained by composing the implementation with its mirrored specification. If while exploring the state space a failure is detected, a sequence of transitions found using a depth-first search is reported that demonstrates the failure. This sequence, however, may be quite long, so after reporting the fail-

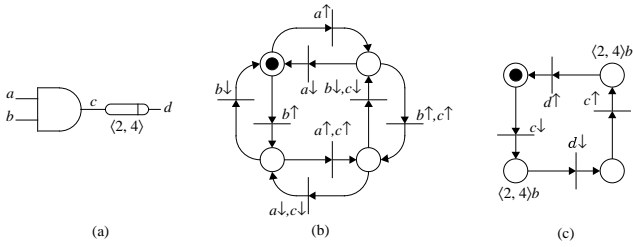


Fig. 18. (a) AND gate with inputs a and b , and output d ; (b) orbital net for its functional behavior; (c) delay buffer with input c , output d , and delay of $(2, 4)$.

ure the procedure finds a possibly shorter sequence using a breadth-first search.

The verification procedure described in the previous section has been automated in the tool `Orbits` written by Tom Rokicki. This tool has been incorporated into the design system for timed circuits ATACS. Experimental results are given in Table II which were run on an HP9000/735 with 144 megabytes of memory using CScheme 7.3. The left four columns indicate values that are the same for geometric and POSET timing. The startup time is the time required to parse the input and construct the appropriate orbital net. The number of net nodes is the sum of the places and transitions in the resulting orbital net. The third column gives an estimate of the number of untimed states. The fourth column gives the number of discrete states, after all timing parameters are divided by their greatest common divisor. The next four columns give the number of geometric regions and the runtime in seconds for verification using standard geometric timing and POSET timing, respectively.

The first half of Table II consists of the automatically synthesized gate-level timed circuits described above. First, we find that the number of discrete states can be quite large making discrete-time verification difficult, if not impossible. Verification of these examples using POSET timing is also more efficient than the geometric timing approach, especially in the case of the DRAM controller where the verification time is improved by over an order of magnitude.

The second half of the table consists of other timed circuits and systems that exhibit a high degree of concurrency. For example, the `seitz` queue element is from [38], and `seitz2` is two connected copies of this circuit. The `ky` examples [35] have thirty-seven gates and timing parameters given to three significant digits. Where the examples ran out of time or space using the geometric method, often the verification is far from done. For the `seitz2` example, after one hour of CPU time, only 1,404 of the 4,572 untimed states have been seen, yet 473,202 distinct geometric regions have been encountered. One particular untimed state has 13,275 distinct geometric regions at this point. POSET timing for this example finds the entire state space as 5,820 geometric regions in one half minute of CPU time.

One more thing to consider from Table II is the ratio of the number of regions found using POSET timing to the

number of untimed states. We find that POSET timing often finds on average very close to one, and in all of our examples, no more than two geometric regions for every untimed state. This means that the POSET timing approach is achieving a near optimal representation of the timed state space.

V. CONCLUSION

This paper describes POSET timing and its application to the automatic synthesis and verification of gate-level timed circuits. The POSET timing algorithm operates on specifications represented as orbital nets. These nets must satisfy the single behavioral place restriction. We describe a net transformation method that can be applied to any 1-safe timed Petri-net to always produce an orbital net that satisfies the single behavioral place restriction. The POSET timing algorithm extends geometric methods using concurrency and causality information represented in a poset. By considering posets of events instead of linear sequences, the POSET timing algorithm is capable of substantially reducing the number of geometric regions necessary to represent the reachable timed state space. Our original synthesis method for timed circuits is restricted to choice-free systems. POSET timing allows us to extend this synthesis method to a very general class of systems, namely any system that can be specified as a timed Petri-net. We also demonstrate the effectiveness of this synthesis procedure on several practical examples, and our results indicate that our timed circuit implementations are significantly smaller and faster than those produced by other asynchronous and synchronous design methodologies. Our verification results show that POSET timing verification can handle some larger, more concurrent examples than the standard discrete or geometric methods. POSET timing also often finds on average very close to one and no more than two geometric regions for every untimed state which means this approach is achieving a near optimal representation of the timed state space. The efficiency of the POSET timing algorithm has allowed us to incorporate timing into asynchronous circuit design. Our procedure produces both efficient and reliable implementations opening the door to the use of asynchronous circuits in domains previously dominated by synchronous circuits.

ACKNOWLEDGMENTS

We are especially thankful for invaluable comments on this work from Professor Peter Beerel of the University of Southern California. We would also like to thank Professor Steve Burns, Professor Gaetano Borriello, and Henrik Hulgaard of the University of Washington for providing many stimulating discussions about timing analysis. We greatly appreciate the comments on this work which we received from Professor Alain Martin and his graduate students at Caltech. We would like to thank Ludmila Cherkasova and Vadim Kotov of Hewlett-Packard Laboratories for their discussions and comments during the development of `Orbits`. We are grateful to Wendy Belluomini and Robert Thacker of the University of Utah for their

TABLE II

VERIFICATION RESULTS. TIME VALUES ARE GIVEN IN SECONDS. AN ENTRY OF *out of time* INDICATES THAT THE VERIFICATION DID NOT COMPLETE WITHIN TWO HOURS, AND AN ENTRY OF *out of memory* INDICATES THAT THE VERIFICATION RAN OUT OF MEMORY BEFORE COMPLETING.

Examples	Startup time	Net nodes	Untimed states	Discrete states	Geometric		POSET	
					regions	time	regions	time
SEL	2.59	770	271	6.16e5	582	1.91	358	1.76
SEL2	2.26	616	96	2033	130	0.33	102	0.29
MMU	5.94	2248	547	2.21e7	1163	5.22	583	2.03
DRAM	3.83	1326	8093	1.17e6	70611	1492.97	8899	98.13
TSBM	3.57	1464	305	49936	510	3.36	305	2.07
adv3x40	0.05	6	1	68921	1.52e5	164.99	1	0.01
adv4x40	0.03	8	1	2.83e6	out of memory		1	0.01
adv50x40	0.27	100	1	4.36e80	out of memory		1	60.21
phil3	0.19	149	144	27806	758	0.77	188	0.36
phil4	0.22	197	1152	9.82e5	out of time		1541	6.98
phil5	0.25	245	9840	3.47e7	out of time		14039	159.40
seitz	0.41	355	344	2.92e13	3234	5.48	416	1.22
seitz2	0.55	624	4572	5.48e19	out of memory		5820	29.79
kyy5	2.46	1484	5266	>1e20	out of memory		6083	56.74
kyy15	1.97	1484	18357	>1e20	out of memory		20250	321.47

comments on this manuscript. Finally, we are very grateful to Professor David Dill of Stanford University for his steady support and guidance.

REFERENCES

- [1] A. J. Martin, "Programming in VLSI: from communicating processes to delay-insensitive VLSI circuits," in *UT Year of Programming Institute on Concurrent Programming*, C.A.R. Hoare, Ed. Addison-Wesley, 1990.
- [2] T.-A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications*, Ph.D. thesis, Massachusetts Institute of Technology, 1987.
- [3] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messersmith, "Automatic synthesis of asynchronous circuits from high-level specifications," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 11, pp. 1185–1205, November 1989.
- [4] G. Borriello, *A New Specification Methodology and its Applications to Transducer Synthesis*, Ph.D. thesis, University of California, Berkeley, 1988.
- [5] P. Vanbekbergen, *Synthesis of Asynchronous Controllers from Graph-Theoretic Specifications*, Ph.D. thesis, Katholieke Universiteit Leuven, September 1993.
- [6] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli, "Algorithms for synthesis of hazard-free asynchronous circuits," in *Proceedings of the 28th ACM/IEEE Design Automation Conference*, 1991.
- [7] C. J. Myers and T. H.-Y. Meng, "Synthesis of timed asynchronous circuits," *IEEE Transactions on VLSI Systems*, vol. 1, no. 2, pp. 106–119, June 1993.
- [8] J. R. Burch, *Trace Algebra for Automatic Verification of Real-Time Concurrent Systems*, Ph.D. thesis, Carnegie Mellon University, 1992.
- [9] P. Merlin and D. J. Faber, "Recoverability of communication protocols," *IEEE Transactions on Communications*, vol. 24, no. 9, 1976.
- [10] T. G. Rokicki, *Representing and Modeling Circuits*, Ph.D. thesis, Stanford University, 1993.
- [11] R. Alur, *Techniques for Automatic Verification of Real-Time Systems*, Ph.D. thesis, Stanford University, August 1991.
- [12] T. A. Henzinger, *The Temporal Specification and Verification of Real-Time Systems*, Ph.D. thesis, Stanford University, 1991.
- [13] T. A. Henzinger, Z. Manna, and A. Pnueli, "What good are digital clocks?," in *ICALP 92: Automata, Languages, and Programming*, 1992, pp. 545–547, Springer-Verlag.
- [14] H. Hulgaard and S.M. Burns, "Bounded delay timing analysis of a class of CSP programs with choice," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, November 1994, pp. 2–11.
- [15] E. A. Walkup and G. Borriello, "Interface timing verification with combined max and linear constraints," in *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, September 1993.
- [16] J. Gunawardena, "Timing analysis of digital circuits of min-max functions," in *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, September 1993.
- [17] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Proceedings of the Workshop on Automatic Verification Methods for Finite-State Systems*, 1989.
- [18] H. R. Lewis, "Finite-state analysis of asynchronous circuits with bounded temporal uncertainty," Tech. Rep., Harvard University, July 1989.
- [19] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time petri nets," *IEEE Transactions on Software Engineering*, vol. 17, no. 3, March 1991.
- [20] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi, "An implementation of three algorithms for timing verification based on automata emptiness," in *Proceedings of the Real-Time Systems Symposium*, 1992, pp. 157–166, IEEE Computer Society Press.
- [21] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model-checking for real-time systems," in *Proceedings of the 7th Symposium Logics in Computers Science*, 1992, IEEE Computer Society Press.
- [22] N. Halbwachs, "Delay analysis in synchronous programs," in *Computer Aided Verification*, Costas Courcoubetis, Ed. 1993, pp. 333–346, Springer-Verlag.
- [23] A. Valmari, "A stubborn attack on state explosion," in *International Conference on Computer-Aided Verification*, June 1990, pp. 176–185.
- [24] P. Godefroid, "Using partial orders to improve automatic verification methods," in *International Conference on Computer-Aided Verification*, June 1990, pp. 176–185.
- [25] K. McMillan, "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits," in *Proc. International Workshop on Computer Aided Verification*,

- G. v. Bochman and D. K. Probst, Eds. 1992, vol. 663 of *Lecture Notes in Computer Science*, pp. 164–177, Springer-Verlag.
- [26] T. Yoneda, A. Shibayama, B. Schlingloff, and E. M. Clarke, “Efficient verification of parallel real-time systems,” in *Computer Aided Verification*, Costas Courcoubetis, Ed. 1993, pp. 321–332, Springer-Verlag.
- [27] C. J. Myers, *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*, Ph.D. thesis, Stanford University, 1995.
- [28] P. A. Beerel, C. J. Myers, and T. H.-Y. Meng, “Covering conditions and algorithms for the synthesis of speed-independent circuits,” *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 3, March 1998.
- [29] P. Beerel and T.H.-Y. Meng, “Automatic gate-level synthesis of speed-independent circuits,” in *Proc. International Conf. Computer-Aided Design (ICCAD)*. Nov. 1992, pp. 581–587, IEEE Computer Society Press.
- [30] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev, “Basic gate implementation of speed-independent circuits,” in *Proc. ACM/IEEE Design Automation Conference*, June 1994, pp. 56–62.
- [31] Kuan-Jen Lin, Jih-Wen Kuo, and Chen-Shang Lin, “Direct synthesis of hazard-free asynchronous circuits from STGs based on lock relation and MG-decomposition approach,” in *Proc. European Design and Test Conference*. 1994, pp. 178–183, IEEE Computer Society Press.
- [32] K. Y. Yun, *Synthesis of Asynchronous Controllers for Heterogeneous Systems*, Ph.D. thesis, Stanford University, 1994.
- [33] C. J. Myers and A. J. Martin, “The design of an asynchronous memory management,” Tech. Rep. CS-TR-93-30, California Institute of Technology, 1993.
- [34] S. M. Nowick, K. Y. Yun, and D. L. Dill, “Practical asynchronous controller design,” in *International Conference on Computer Design, ICCD-1992*. 1992, IEEE Computer Society Press.
- [35] K. Y. Yun, “Private communication,” 1993.
- [36] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli, “SIS: A system for sequential circuit synthesis,” Tech. Rep. UCB/ERL M92/41, University of California, Berkeley, May 1992.
- [37] D. L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*, MIT Press, 1989.
- [38] C. L. Seitz, “System timing,” in *Introduction to VLSI Systems*, Carver A. Mead and Lynn A. Conway, Eds., chapter 7. Addison-Wesley, 1980.
- [39] T. G. Rokicki and C. J. Myers, “Automatic verification of timed circuits,” in *International Conference on Computer-Aided Verification*. 1994, pp. 468–480, Springer-Verlag.
- [40] C. J. Myers, T. G. Rokicki, and T. H.-Y. Meng, “Automatic synthesis of gate-level timed circuits with choice,” in *16th Conference on Advanced Research in VLSI*. 1995, pp. 42–58, IEEE Computer Society Press.

APPENDIX

Our definition of the covering constraint for correct standard C-element covers differs from the one given in [29], [28] in that it allows the inclusion of any quiescent states in the cover and does not restrict it to quiescent region states. This restriction is redundant when the entrance constraint is considered. Furthermore, if a quiescent state s is reachable by different paths from two distinct excitation regions, the entrance constraint also excludes this state. This appendix proves that our correctness constraints are identical to ones which only allow their covering constraint to extend into states in non-shared quiescent regions.

Definition V.1: (Quiescent Path)

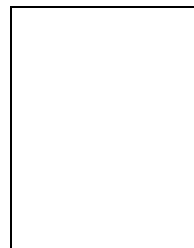
A stable path for signal u is defined to be a set of states $\{s_0, s_1, s_2, \dots, s_n\}$ such that for all i less than n , $(s_i, t_i, s_{i+1}) \in \Gamma$ and for all i greater than 0, signal u is stable (i.e., $s_0 = F$ and $s_1(u) = s_2(u) = \dots = s_n(u) = 0$ or $s_0 = R$ and $s_1(u) = s_2(u) = \dots = s_n(u) = 1$).

Theorem V.1: A correct cover for $ER(u^*, k)$ as defined in Section IV-A cannot include a quiescent state s which

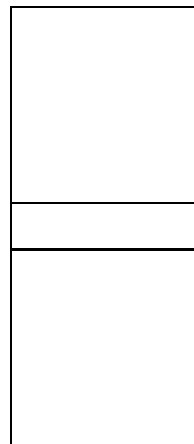
can be reached through a quiescent path that begins with a state s' which is not in $ER(u^*, k)$.

Proof: (by contradiction) Assume that state s is in $C(u^*, k)$ and there exists a quiescent path that begins in state $s_0 = s'$ which is not in $ER(u^*, k)$ (i.e., it is in $ER(u^*, k')$) and ends in state $s_n = s$. From the covering constraint, s' cannot be in $C(u^*, k)$ since it is in neither $ER(u^*, k)$ or $QS(u^*)$. Since s' is not in $C(u^*, k)$ and s_1 is not in $ER(u^*, k)$ (the second state in the quiescent path is in $QS(u^*)$), s_1 cannot be in $C(u^*, k)$ due to the entrance constraint. Since s_1 is not in $C(u^*, k)$ and s_2 is not in $ER(u^*, k)$, s_2 also cannot be in $C(u^*, k)$ due to the entrance constraint. This argument can be carried through to state s (i.e., state s_n) to show that state s cannot be in $C(u^*, k)$. ■

Note that this proof shows that the entrance constraint eliminates from covers any quiescent states reachable from other excitation regions whether they are reachable by the excitation region being covered or not.



Chris J. Myers received the B.S. degree in electrical engineering and Chinese history in 1991 from the California Institute of Technology, Pasadena, CA, and the M.S.E.E. and Ph.D. degrees from Stanford University, Stanford, CA, in 1993 and 1995, respectively. He has been an Assistant Professor in the Department of Electrical Engineering, University of Utah, Salt Lake City, UT, since 1995. His current research interests are innovative architectures for high performance and low power, algorithms for the computer-aided analysis and design of real-time concurrent systems, formal verification, and asynchronous circuit design. Dr. Myers received an NSF Fellowship in 1991 and an NSF CAREER award in 1996. He was recently awarded a Center for Asynchronous Circuit and System Design by the State of Utah, for which he serves as Director.



Tomas G. Rokicki received his Ph.D. in Computer Science from Stanford University, Stanford, CA in 1993. Since 1993, he has been working at HP Labs in Palo Alto, CA.

Teresa H.-Y. Meng joined the faculty of the Electrical Engineering Department at Stanford University in 1988, where she is Associate Professor. Her current research activities include low-power circuit design, wireless communications, and portable DSP systems. Among the awards she has received are the IEEE Signal Processing Society's Paper Award in 1989, the 1989 NSF Presidential Young Investigator Award, the 1989 ONR Young Investigator Award, a 1989 IBM Faculty Development Award, and the 1988 Eli Jury Award from U.C. Berkeley for recognition of excellence in systems research. She received a B.S. degree from National Taiwan University in 1983 and an M.S. and Ph.D. from U.C. Berkeley in 1984 and 1988 respectively. Dr. Meng was elected Fellow of the IEEE in 1998.