Verification of Timed Circuits with Symbolic Delays

Robert Clarisó
Universitat Politècnica de Catalunya
Barcelona, Spain

Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

***Abstract*— When time is incorporated in the specification of discrete systems, the complexity of verification grows exponentially. When the temporal behavior is specified with symbols, the verification problem becomes even more difficult. This paper presents a formal verification technique for timed circuits with symbolic delays. The approach is able to provide a set of sufficient linear constraints on the symbols that guarantee the correctness of the circuit. The applicability of the technique is shown by solving the problem of *automatic discovery of linear constraints on input and gate delays that guarantee the correct behavior of asynchronous circuits*.**

I. Introduction

The correctness of concurrent systems often depends on the temporal characteristics: response times, timeouts, computational delays, etc. Several formalisms have been proposed to model such systems, such as Timed Transition Systems [13], Timed Automata [2] and Hybrid Automata [1].

In these models, a system is specified as an automaton with annotated timing information. Given a property, verification usually gives an answer of this sort: "*the system is correct*" or "*the trace σ leads to a failure*", where σ is a sequence of events annotated with time. However, the previous answer is only valid for the particular timing information provided for that instance of the system. Let us assume, for example, that this information is the set of gate delays of a circuit. The answer would only be valid for a particular technology, and could not be extrapolated to other technologies. Would it be possible to give a characterization of the circuit as a set of timing constraints that could guarantee the correctness of the circuit and that would be independent from the technology? For example, the following answer would be much more meaningful: *The circuit is correct if*

$$\begin{aligned} \delta(G_1) + \delta(G_2) &< \delta(G_3) + \delta(G_4) + \delta(G_5) \qquad \wedge \\ \delta(G_2) &< 3\,\delta(G_6) + 2\,\delta(G_7) \end{aligned}$$

where $\delta(G_i)$ denotes the delay of the gate G_i. The advantage of this type of answer is obvious. However, this requires an analysis with *symbolic* delays, that makes verification much more complex.

This paper presents an algorithmic approach for the *automatic discovery of linear constraints in timed systems that guarantee their correctness*. The technique is based on the the paradigm of *Abstract Interpretation* [8], that was originally devised for the static analysis of programs [9]. One of the main motivations of this work is the characterization of the behavior of asynchronous controllers. The correctness of these circuits often depends on the actual delays of the gates. Under certain gate delays, the circuit may manifest hazardous behavior that can be propagated to some output signal and produce a failure. The purpose of the verification is to derive a set of linear constraints on the gate delays that guarantee a correct behavior. Each constraint usually refers to a pair of structural paths in the circuit whose delays must be related by an inequality (e.g. delay(path_1) $<$ delay(path_2)).

The complexity of the problem restricts the size of the circuits that can be verified with this approach, since explicit representations of the states are required. So far, circuits with up to 20 symbols have been verified. This makes the approach specially suitable for the verification of small circuits whose behavior depends on the timing characteristics of the components, such as asynchronous controllers. Some examples of these controllers are the IPCMOS circuits from IBM [18], or the GasP FIFO control circuits from Sun Microsystems [19] (see Figure 8). The technique is also applicable to any level of granularity. For example, one could verify RTL specifications with delays at the level of functional blocks (ALUs, counters, controllers, etc).

The paper is organized as follows. Section II presents an example of verification with symbolic delays. Section III discusses related work in the area. Section IV introduces timed transitions systems and symbolic delays. The main algorithm for reachability analysis is presented in Section V. Finally, Section VI illustrates the applicability of the approach to some examples.

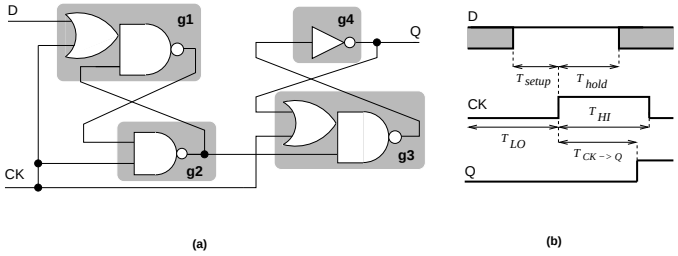
II. Example: Verifying a D flip-flop

We illustrate the power of symbolic analysis with linear contraints by means of an example. Let us take the D flip-flop depicted in Fig. 1(a) [17]. Each gate g_i has a symbolic delay in the interval $[d_i, D_i]$. We call T_{setup}, T_{hold} and $T_{CK\rightarrow Q}$ the setup, hold and clock-to-output times, respectively. T_{LO} and T_{HI} define the behavior of the clock. Our goal is to symbolically characterize the latch behavior in terms of the internal gate delays.

The method presented in this paper is capable of deriving a set of sufficient linear contraints that guarantee the correctness of the latch's behavior. The verified property is the following:

> *The value of Q after a delay $T_{CK\rightarrow Q}$ from CK's rising edge must be equal to the value of D at CK's rising edge.*

Any behavior not fulfilling this property is considered to be a



$$
\begin{aligned}
T_{CK\to Q} &\leq D_2 + D_3 + D_4 \\
T_{setup} &> D_1 + D_2 - d_2 \\
T_{hold} &> D_2 + D_3 \\
T_{HI} &> D_2 + D_3 + D_4 \\
T_{HI} &> T_{hold} \\
T_{LO} &> T_{setup} \\
d_1 &> D_2
\end{aligned}
$$

(c)

$$
\begin{aligned}
T_{CK\to Q} &\leq 3 + D_3 + D_4 \\
T_{setup} &> 9 \\
T_{hold} &> 0 \\
T_{HI} &> 3 + D_3 + D_4 \\
T_{LO} &> T_{setup}
\end{aligned}
$$

(d)

Fig. 1. (a) Implementation of a D flip-flop [17], (b) description of variables that characterize any D flip-flop and (c) sufficient constraints for correctness for any delay of the gates, (d) sufficient constraints if some delays are known: $g1 = [4, 7]$ and $g2 = [1, 3]$.

failure[1]. Fig. 1(c) reports the set of sufficient timing constraints derived by the algorithm. The most interesting aspect of this characterization is that it is *technology independent*.

As an example, let us focus on two constraints. First, $d_1 > D_2$ is necessary to prevent the cross-coupled gates $g1$ and $g2$ read the wrong value of D or enter metastability. Second, $T_{setup} > D_1 + D_2 - d_2$ defines the setup time that, interestingly, depends on the variability of the delay of g2. In case of no variability on the delays, the constraint is reduced to $T_{setup} > D_1$, which is the time required for $g1$ to capture the value of D.

The degree of parametrization can be chosen at the designer's will. If some delays are known, they can be used during the verification. As an example, let us assume that the delay of g_1 and g_2 are in the intervals $[4, 7]$ and $[1, 3]$, respectively. The sufficient constraints with these assumptions are reported in Fig. 1(d).

III. Related work

Several techniques for computing conservative timing constraints for the correct operation of asynchronous circuits are available in the literature. The main difference between this paper and these approaches is that they are based on analyzing the circuit with known constant min-max delays in gates and wires [4], [5], [7], [15], [16]. The approach presented in this paper can deal with *unknown* delays that are represented as *symbols*. Therefore, the analysis can be performed without making any assumption on the delay of the components of the circuit or the events of the environment.

The kind of timing constraints that can be computed also differs from our approach. The first class of constraints is *metric timing* constraints, i.e. constant min-max bounds for the components of a circuit. In [15], contraints are described as bounded delays called *delay paddings* that have to be introduced in the circuit to guarantee correctness. [5] computes delay paddings, plus the required delay bounds on input events. Another group of constraints is *relative timing* [4], [7], [14], [16], i.e. constraints that describe the relative order among concurrent events. Our approach can compute a wider class of constraints, *linear constraints*. Therefore, our analysis provides less conservative timing constraints, that can yield an increase in performance.

Our approach uses convex polyhedra as the abstraction to represent sets of timed states. In [9] convex polyhedra are used to analyze linear relations among variables, in the context of algorithms for the static analysis of programs. To preserve closedness in set operations, polyhedra can only represent approximations of the state space. For example, the union is not closed for convex polyhedra. As an overapproximation, the convex hull is used instead. This strategy has also been used by other authors for the approximate verification of real-time systems [11], linear hybrid automata and synchronous programs with counters [12]. Linear hybrid automata and synchronous programs differ from timed transition systems in the condition required for an event to happen, i.e. there is no restriction on the time elapsed since an event becomes enabled for firing until it is finally fired, contrary to the lower and upper bound requirements defined in timed transition systems.

[1]An inertial delay model is assumed for the verification

IV. Definitions

The behavior of a timed circuit can be modeled as a timed transition system (TTS). A TTS is a transition system where each event has a lower and upper delay bounds. In the remainder of the paper, the delay of an event will be denoted by $[d_e, D_e]$. For those events where no distinction is made for min and max delays, we will simply use the notation $\delta(e)$. Intuitively, the lower bound restriction states that an event e should be fired at least d_e time units after becoming enabled, and the upper bound restriction states that e is fired at most D_e time units after becoming enabled.

The following definitions present the concepts of transition system and TTS, together with the semantics of these models, i.e. the concept of "run".

Definition 4.1: [3] A *transition system* (TS) is a quadruple $A = \langle S, \Sigma, T, s_{in} \rangle$, where S is a non-empty set of *states*, Σ is a non-empty alphabet of *events*, $T \subseteq S \times \Sigma \times S$ is a *transition relation*, and s_{in} is the *initial state*. Transitions are denoted by $s \xrightarrow{e} s'$. An event e is enabled at state s if $\exists s \xrightarrow{e} s' \in T$. We will denote the set of events enabled at state s by $\mathcal{E}(s)$.

Definition 4.2: Let $A = \langle S, \Sigma, T, s_{in} \rangle$ be a TS. A *run* of A is a sequence $s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} \ldots$ such that $s_1 = s_{in}$ and $s_i \xrightarrow{e_i} s_{i+1} \in T$ for all $i \geq 1$.

Definition 4.3: [13] A *timed transition system* (TTS) is a triple $A = \langle A^-, d, D \rangle$ where $A^- = \langle S, \Sigma, T, s_{in} \rangle$ is a TS called the *underlying transition system*, $d : \Sigma \to \mathbb{R}^+$ and $D : \Sigma \to \mathbb{R}^+ \bigcup \{\infty\}$ respectively associate a *minimal* and a *maximal delay bounds* to each event, such that $\forall e \in \Sigma : d_e \leq D_e$.

Definition 4.4: [13] A *timed state sequence* is a pair $\rho = \langle \sigma, t \rangle$ such that σ is a sequence of states and t is a sequence of

time stamps in $\mathbb{R}^+$, t_1, t_2, t_3, ... such that $t_1 \leq t_2 \leq t_3 \leq \ldots$ (*monotonic*) and $\forall k \in \mathbb{R}^+ : \exists i\ t_i \geq k$ (*progress*).

Definition 4.5: [13] Let $A = \langle A^-, d, D \rangle$ be a TTS. A *run* of A is a timed state sequence $\rho = \langle \sigma, t \rangle$ such that σ is a run of the underlying transition system A^- and:

- *lower bound*: $\forall e \in \Sigma, i \geq 0, j \geq i : t_j < t_i + d_e : (s_j \xrightarrow{e} s_{j+1} \in \sigma) \rightarrow (e \in \mathcal{E}(s_i))$.
- *upper bound*: $\forall e \in \Sigma, i \geq 0 : \exists j \geq i : t_j \leq t_i + D_e : e \notin \mathcal{E}(s_i) \vee (s_j \xrightarrow{e} s_{j+1} \in \sigma)$.

The definition of TTS can be easily extended to allow symbolic delays in addition to constant delays.

V. Timing reachability algorithm

A. Overview

Events of a TTS can only be fired if their lower and upper bound restrictions are satisfied. Intuitively, each event has an associated event clock that stores the amount of time elapsed since the transition became enabled. Each time an event is fired, event clocks have to be modified accordingly. Analysis of the values of event clocks can reveal whether an event can be fired or not in a given state.

This section presents an algorithm that computes a conservative upper approximation of the event clock values. Approximations will be propagated and combined using fixpoint techniques described in abstract interpretation [8].

The following sections describe the different parts of the algorithm: the *abstract interpretation* techniques used to propagate the approximations are explained in V-B; the basic operations on our approximation, *convex polyhedra*, are defined in V-C; finally, the function that computes how our approximation is changed after firing an event, one of the main contributions of the paper, is defined in V-D.

B. Abstract interpretation

Abstract interpretation [8], [9] is a framework of approximate static analysis techniques which can be applied to many kinds of analysis problems in different types of systems. In order to solve a specific problem, the framework of abstract interpretation has to be adapted to:

- *the properties being studied:* We can define a *state* of a system as the set of values that describe the configuration of the system at any given point. The state may contain information which is not necessary to check a given property. Therefore, in our analysis we can work with an *abstraction*, a simplification of the state that ignores the information of the configuration that is not relevant in the specific problem.
- *the semantics of the system:* The behavior of a system can be defined by identifying a set of *locations* where we require information about the state. The relations among the state of the system in these locations establishes a *system of equations*.

The system of equations is solved iteratively using fixpoint techniques, yielding an abstraction that describes an upper approximation of the state in each location of the system.

```
Algorithm AbstractInterpretation (G, A_In)
Input: A graph G = (N, E) with initial node In and initial constraints A_In.
Output: The abstraction Time for all nodes and edges.

  foreach node n ∈ N do Time(n) := ∅; endfor
  foreach edge e ∈ E do Time(e) := ∅; endfor
  Time(In) := A_In;
  changed := {N};
  do
    n := node in changed with lowest DFS number;
    changed := changed \ n;
    foreach edge n --e--> m ∈ E
      newTime := transfer(Time(n));
      if (newTime ⊆ Time(e)) continue;
      Time(e) := newTime;
      if (Time(e) ⊆ Time(m)) continue;
      if (e is a back edge)
        Time(m) := Time(m) ▽ (Time(e) ⋃ Time(m));
      else
        Time(m) := Time(m) ⋃ Time(e);
      changed := changed ⋃ {m};
  while (changed ≠ ∅);
```

Fig. 2. Abstract interpretation algorithm

For the problem of timing analysis of a TTS, a configuration is a set of valid assignments of constant values to clocks and symbolic delays. We will abstract the set of valid assignments as a convex polyhedron that is an upper approximation of this set, i.e. all valid assignments are included in the polyhedron. The convex polyhedron will describe the linear constraints that are satisfied among clock values and symbolic delays in all these valid assignments.

There will be two kinds of locations of interest of our timing analysis of TTS: states and transitions. We will note the abstraction in a given location x as $\mathsf{Time}(x)$, even though the abstraction has a different meaning for states than for transtions.

- In states, we are interested in the value of clocks when a state is reached, i.e. the *precondition* of the state.
- About transitions, we would like to compute the value of clocks after the transition happens, i.e. after firing an event. This can be considered as computing the *postcondition* of the transition.

In order to define the timing behavior of the system, we have to build a system of equation that defines how time elapses. When an state is reached, several events become enabled while other events that were enabled previously continue to be enabled. These events have to be fired according to its lower and upper delay bound, taking into account that some events have already been enabled for some time. We have defined a symbolic function called *transfer* (explained in detail in section V-D) that advances the clock values while satisfying all upper and lower bounds. Using this function, the abstractions for states and transitions can be defined as the following system of equations:

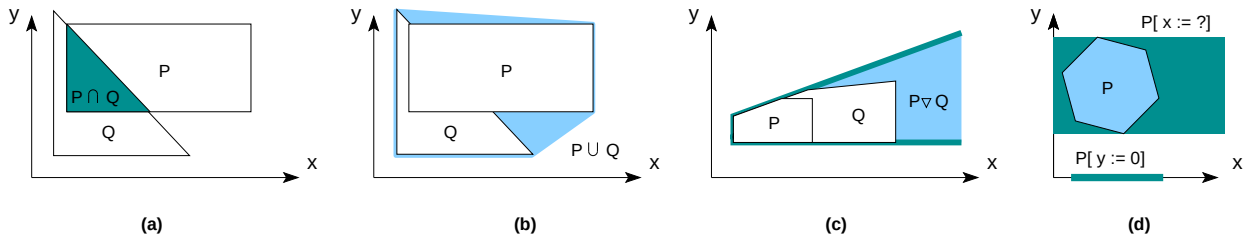


Fig. 3. Several operations on convex polyhedra: (a) intersection of polyhedra , (b) union of polyhedra as the convex hull, (c) widening of polyhedra and (d) assignment of a linear expression or an undefined value.

- $\forall n \xrightarrow{e} m \in T : \mathsf{Time}(e) = transfer(\mathsf{Time}(n))$
- $\forall m \in T, n \xrightarrow{e} m \in T : \mathsf{Time}(m) = \bigcup \mathsf{Time}(e)$

Figure 2 describes an algorithm that computes a solution for this system of equations using a *increasing* fixpoint. Each location starts with an empty set of valid assignments to clocks and values, i.e. and empty abstraction. The algorithm applies the equations iteratively as long as the add new valid assignments. The solution is reached when there is a fixpoint, i.e. applying all equations another time does not yield any new states in any location of the system.

Termination, i.e. convergence of the system of equations, is guaranteed by modifying the computation for loops. A *widening* operator [9] is used in the the equations of those states that are the targets of back-edges. The widening operator ($A \bigtriangledown B$) is defined as a superset of $A \bigcup B$ that reaches convergence after being applied a finite number of times. Using widening in all loops ensures the convergence of the system of equations, as well as reducing the number of iterations required to reach a fixpoint. An in-depth discussion on termination of fixpoints and the necessity of widening can be found in [8], [9].

C. Convex polyhedra

This section will introduce the basic concepts about convex polyhedra required to understand the implementation of its operators. The reader can find an in-depth description of convex polyhedra in [9], [12].

Convex polyhedra can be represented as the set of solutions of a conjunction of *linear inequalities* with rational coefficients. Let P be a polyhedron over $\mathbb{Q}^n$, then it can be represented as the solution to the system of m inequalities $P = \{X | AX \geq B\}$ where $A \in \mathbb{Q}^{m \times n}$ and $B \in \mathbb{Q}^n$. Convex polyhedra can also be represented in a *polar* representation, called the *system of generators*, as a linear combination of a set of vertices V (points) and a set of rays R (vectors).

The fact that there are two representations is important, because there are efficient algorithms [9] that translate one representation to the other, and several of the operations for convex polyhedra are computed very efficiently when the proper representation of polyhedra is available.

The set of operations on convex polyhedra that are required for timing analysis are the following:

- **Test for inclusion** ($P \subseteq Q$): Inclusion is an exact operation. P is included in Q only if the generators of P satisfy the constraints of Q, that is, $\forall v \in V : Av \geq B$ and $\forall r \in R : Ar \geq 0$.
- **Union** ($P \bigcup Q$)): The union of convex polyhedra is not necessarily convex, and therefore an upper approximation is used. This approximation is called *convex hull*, the least convex polyhedron that includes P and Q. $P \bigcup Q$ is defined as the polyhedron with a system of generators that is the union of those in P and Q.
- **Intersection** ($P \bigcap Q$): The intersection of two convex polyhedra is necessarily convex. $P \bigcap Q$ can be defined as the polyhedron with a system of linear inequalities that contains all the inequalities in P and Q.
- **Widening** ($P \bigtriangledown Q$): Widening is the approximate operator used to guarantee termination in loops. Widening operator must ensure that it will reach fixpoint after a finite number of iterations. $P \bigtriangledown Q$ is defined as the system of linear inequalities which are satisfied both by P and Q. As the number of inequalities in P and Q is finite and this operator can only reduce or maintain the number of inequalities, termination in a finite number of steps is ensured.
- **Applying a linear assignment** ($P[d := Cx+D]$): Linear assignments to a dimension of the polyhedron transform the vertices and the edges of the polyhedron as $V' = \{Cv + D | v \in V\}$ and $R' = \{Cr | r \in R\}$.
- **Assigning an undefined value to a dimension** ($P[d := ?]$)): this operation removes all constraints for a given dimension of the polyhedron, while keeping all the implicit constraints about the rest of dimensions intact. This operation is implemented with the Fourier-Motzkin elimination [10] method, i.e. we update the system of inequalities as follows: First, we add all the possible linear combinations of inequalities with non-zero coefficient in d so the coefficient in d becomes zero. For m inequalities, at most $(m/2)^2$ linear combinations will be added to the system of inequalities. Then, inequalities where dimension d has non-zero coefficient are removed.

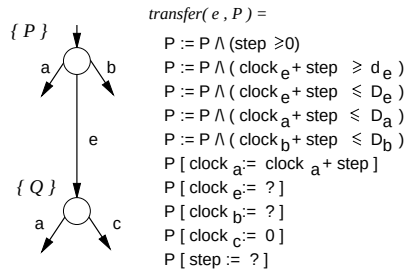
Figure 3 shows some examples of these operations on convex polyhedra. It should be noted that the convex hull and the widening operator are the only operators that loose precision. All other operators are exact.

Convex polyhedra have been chosen only because of its good trade-off between efficiency and expresiveness. However, any other abstraction of a subset of $\mathbb{Q}^n$ providing the necessary operators could have been used instead of convex polyhedra. An example of an alternative abstraction is finite unions of convex polyhedra, which are more precise but less efficient.

Algorithm *transfer*(src, dst, e, P)
Input: An event $src \xrightarrow{e} dst$ with precondition P.
Output: The postcondition of $src \xrightarrow{e} dst$.

$P := P \wedge (step \geq 0)$;
$P := P \wedge (clock_e + step \geq d_e)$;
$P := P \wedge (clock_e + step \leq D_e)$;
foreach event $e' \neq e$: $e' \in \mathcal{E}(src)$
  $P := P \wedge (clock_{e'} + step \leq D_{e'})$;
foreach event $e' \neq e$: $e' \in \{\mathcal{E}(src) \cap \mathcal{E}(dst)\}$
  $P[clock_{e'} := clock_{e'} + step]$;
foreach event $e' \neq e$: $e' \in \mathcal{E}(dst) \wedge e' \notin \mathcal{E}(src)$
  $P[clock_{e'} := 0]$;
foreach event $e' \neq e$: $e' \in \mathcal{E}(src) \wedge e' \notin \mathcal{E}(dst)$
  $P[clock_{e'} :=?]$;
if $(e \in \mathcal{E}(dst))$ $P[clock_e := 0]$;
else $P[clock_e :=?]$;
$P[step :=?]$;
return P;

Fig. 4. Clock transfer function



$$\{P\} = \{(clock_e = 0) \wedge (clock_a = 0) \wedge (0 \leq clock_b \leq 1)\}$$
$$\{Q\} = \{(clock_c = 0) \wedge (D_e \geq clock_a \geq d_e) \wedge (D_a \geq clock_a \geq d_e) \wedge (d_e + 1 \leq D_b)\}$$

Fig. 5. Example of the transfer function for an event e, with the postcondition Q obtained from a precondition P.

D. The clock transfer function

The core of the analysis is the *clock transfer* function that computes *symbolically* the changes in clock values after firing an event. Clock values are represented by a convex polyhedron, with one dimension per event clock and one dimension per symbolic delay. The restrictions of this polyhedron represent the restrictions on the clock values in a given state. Intuitively, the purpose of the transfer function is to make sure that whenever an event e is fired, its delay bounds d_e and D_e are taken into account and added to the restrictions on the clock values.

Event clocks for enabled events store the amount of time elapsed since the event became enabled, while disabled clocks are undefined, i.e. there is no restriction on their value. After firing an event, event clocks should be updated, because some time has elapsed since the firing of the last event to the firing of current event. This time spent in the state is called clock *step*, and it should satisfy the following properties:

- Step should be ≥ 0, i.e. no negative time increments.
- Step should be long enough to ensure that the firing of e happens at least d_e time units after e was enabled. At the same time, it should be short enough to ensure that e is fired at most D_e time units after becoming enabled.
- Step should be short enough to ensure that any transition that is enabled before firing e is not forced to fire due to its upper bound constraint.

Once the clock step has been defined, the update in event clocks caused by the firing of an event e can be defined as:

- events that are disabled before and after firing e keep their clocks unchanged.
- events that are enabled before and after firing e have their event clocks increased by the clock step.
- events that become enabled by the firing of e have their clock set to 0.
- events that become disabled by the firing of e have their clock undefined.

Figure 4 describes the algorithm that computes the transfer function using convex polyhedra operators. Figure 5 shows an example of the computation that would be performed by the algorithm. Events that are enabled before and after firing event e have been increased by an amount in the interval $[d_e, D_e]$, i.e. the unknown clock step. Also, notice that some constraints among the symbolic delays of different events have been discovered. These constraints were imposed over the clock step during the transfer, and *implied* several restrictions on the delays that are made explicit when variable step is undefined. For example, the restriction $D_a \geq d_e$ means that event e can be fired only if a is not faster than e. Otherwise, the postcondition of this transition would be empty, i.e. no assignment to clock and symbolic delays is consistent with the firing of the event. This restriction is implied by the constraints $clock_a + step \leq D_a, clock_e + step \geq d_e, clock_a = 0, clock_e = 0$.

The clock transfer function described in this section can be easily modified to deal with symbolic timed automata instead of TTS. Checking location invariants and enabling conditions for transitions can be modeled as adding linear constraints to the polyhedron, and resetting clocks can be done with linear assignments, both of which are available operations on polyhedra. The transfer function for timed automata would be defined as (1) increase clocks by step, (2) check the source location invariant, (3) check the enabling condition of the transition, (4) reset clocks, (5) check the target location invariant and (6) undefine step.

E. Main algorithm

Timing analysis provides the required constraints for the reachability of the states and transitions of the TTS. However, we are looking for the complementary conditions, i.e. the conditions that render failures unreachable. Therefore, an algorithm is needed on top of timing analysis to extract selected constraints from those provided by abstract interpretation. This algorithm is presented in Figure 6.

The input is the specification of a TTS: a set of discrete variables, a transition relation, an initial state, and the delays

`Algorithm` *Verification* (S, F, I)
`Input:` A specification of a TTS S, a predicate F describing failure states and transitions, and a predicate I describing known restrictions on the symbolic delays.
`Output:` A set of constraints on the symbolic delays that is sufficient to avoid the failures defined by F.

G := $ReachabilityAnalysis(S, F)$;
$constraints$:= I;
`do`
  $AbstractInterpretation(G, constraints)$;
  C := set of linear constraints required to reach a failure that are not implied by $constraints$;
  choose a linear constraint c from C;
  $constraints$:= $constraints \wedge \neg c$;
`while` (any failure is reachable $\wedge$ $constraints \neq false$);
$\{constraints = false \rightarrow$ `unavoidable failure`$\}$
`return` $constraints$;

Fig. 6. Main algorithm for verification

of each event. Additionally, a predicate describing the failure states and an invariant of known delay constraints are also provided. The output is a set of sufficient constraints that ensure the absence of failures.

The first step is the calculation of the reachable state space using untimed depth-first reachability analysis. During this stage, failure edges and states will be identified. Also during this traversal, all back-edges of loops are identified and nodes are numbered in quasi-topological order; this order will be used to speed up convergence of the abstract interpretation analysis.

Timing analysis can then be performed on the TTS. The result of this step will be a polyhedron attached to each state and transition of the TTS, including the edges that lead to a failure. The polyhedron attached to each of these edges describes constraints that are required to reach a failure. If any of these constraints is false, the failure will be unreachable. For example, if one polyhedron has the constraints,

$$(a \leq b + c) \wedge (e \geq f)$$

then, the constraint that makes sure that the failure is unreachable is the following disjunction

$$(a > b + c) \vee (e < f)$$

The algorithm proceeds by choosing one of these linear constraints at a time and adding it to the invariant. Currently, this choice is performed interactively, even though we have plans to automate this procedure. The verification continues until all failures have become unreachable or the invariant is *false*. A false invariant means "cannot find a constraint that makes the system correct". It can happen if the system has an unavoidable failure or the algorithm cannot find sufficient constraints due to approximation. On the other hand, the algorithm might return the initial invariant, which means that no additional constraints are required for the correctness of the system.

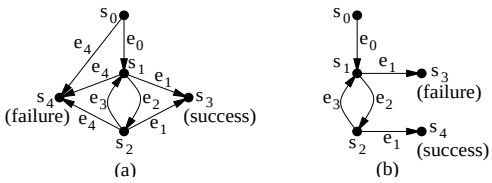


Fig. 7. Cyclic behavior to illustrate the widening operator.

F. Approximate analysis

The problem of computing the set of feasible clock and delay values is computationally expensive. This is the reason why we are using an approximate analysis technique, such as abstract interpretation, instead of trying to compute an exact solution. In the context of our problem, we calculate an *upper* approximation of the state space that guarantees no false positives in the verification of safety properties.

The source of approximation comes from the union of reconvergent paths. In case of acyclic reconvergence, the union is approximated by the convex hull. In case of cyclic reconvergence, the widening operator must also be used to guarantee the convergence of the algorithm. This technique is crucial when symbols are used to represent delays, as the number of iterations of a loop may depend on the actual delays of the components. Since the delays are symbolic, this number may be unknown.

Figure 7 depicts two cyclic behaviors defined by the back edge $s_2 \xrightarrow{e_3} s_1$. Let as assume that each event e_i has a fixed delay $\delta(e_i)$. In Fig. 7(a), the correctness of the system is independent from the delays of e_2 and e_3. However, the absence of the widening operator would produce the following sequence of polyhedra in s_1:

iter.	Time(s_1)
0	$ck_{e_1} = ck_{e_2} = 0 \ \wedge \ ck_{e_4} = \delta(e_0) \leq \delta(e_4)$
1	$ck_{e_1} \leq \delta(e_2) + \delta(e_3) \leq \delta(e_1) \ \wedge \ ck_{e_2} = 0 \ \wedge$ $ck_{e_4} = ck_{e_1} + \delta(e_0) \ \wedge \ ck_{e_4} \leq \delta(e_4)$
...	...
i	$ck_{e_1} \leq i \cdot \delta(e_2) + i \cdot \delta(e_3) \leq \delta(e_1) \ \wedge \ ck_{e_2} = 0 \ \wedge$ $ck_{e_4} = ck_{e_1} + \delta(e_0) \ \wedge \ ck_{e_4} \leq \delta(e_4)$
...	...

where the predicate

$$ck_{e_1} \leq i \cdot \delta(e_2) + i \cdot \delta(e_3)$$

results from the convex union of the same predicate with equality instead of $\leq$, for all $0 \leq k \leq i$. With the widening operator applied after the first iteration, the polyhedron representing Time(s_1) would be reduced to

$$ck_{e_1} \leq \delta(e_1) \ \wedge \ ck_{e_2} = 0 \ \wedge \ ck_{e_4} = ck_{e_1} + \delta(e_0) \leq \delta(e_4)$$

This polyhedron would become invariant in the following iterations. After verification, the condition for absence of failure would be the following:

$$\delta(e_4) > \delta(e_0) + \delta(e_1)$$

Figure 7(b) depicts a situation of a *non-convex* condition for the avoidance of failures. It is easy to prove that the system is correct if the following predicate holds:

TABLE I
EXPERIMENTAL RESULTS

Example	Circuit		STG		TTS		# of	# of	CPU Time
	Signals	Gates	Places	Trans	States	Trans	symbols	constraints	(seconds)
nowick	10	7	19	14	60	119	10	2	0.5
gasp-fifo	9	7	10	8	66	209	12	10	8.1
sbuf-read-ctl	13	10	19	16	74	157	14	4	1.2
rcv-setup	9	6	14	15	72	187	12	8	2.1
alloc-outbound	15	11	21	22	82	161	19	3	1.3
ebergen	11	9	16	14	83	188	13	5	1.3
mp-forward-pkt	13	10	24	16	194	574	12	6	1.9
chu133	12	9	17	14	288	1082	7	3	1.3
converta	14	12	16	14	396	1341	14	13	20.4

$$\exists i > 0 : i \cdot \delta(e_2) + (i-1)\delta(e_3) < \delta(e_1) < i(\delta(e_2) + \delta(e_3))$$

Unfortunately, the existential quantifier represents a disjunction that cannot be expressed as a convex polyhedron. In this case, the predicate for $\mathsf{Time}(s_1)$ would be:

$$ck_{e_1} \geq 0 \ \wedge \ ck_{e_1} \leq \delta(e_1) \ \wedge \ ck_{e_2} = 0$$

This abstraction does not show dependencies between symbolic delays. Therefore, the verification would not be able to provide any set of linear constraints to avoid the failure, even though there are values for delays that make the circuit correct.

VI. EXPERIMENTAL RESULTS

We have implemented the algorithm presented in this paper in a verification tool. In this section, we show some examples that have been verified using this tool.

A. GasP FIFO controller

We have formally verified a GasP FIFO controller from Sun Microsystems [19]. This circuit handles the flow of data between stages of a pipeline: whenever the previous stage is FULL and the next stage is EMPTY, the control circuit (a) produces a pulse to the data latch in order to make it transparent, (b) declares that the next stage is FULL and (c) declares that the previous stage is EMPTY. The state of a stage is encoded in a single wire, where EMPTY (FULL) is encoded as HI (LO). Figure 8 shows the controller of one stage of a pipeline. The environment of this controller corresponds to the previous and next stages of the pipeline. Notice that wire le corresponds to the wire re in the previous stage of the pipeline. The behavior of the environment is modeled with Signal Transition Graphs (STG) [6]. Environment events such as $x+$ or $y-$ describe the rising or falling of signals, and its delay models the time required to fire an event since it becomes enabled in the STG.

This asynchronous controller is designed to achieve a very high throughput, so it depends on timing constraints for its correct operation. In [14], this circuit is verified and sufficient relative timing constraints to ensure correctness are derived. However, it is hard to translate relative timing constraints into constraints on the delays of the components of the circuit.

The correctness of the circuit has been verified with respect to three criteria: *absence of short-circuits*; *absence of hazards*, i.e. once an event becomes enabled, it does not become disabled before being fired; and *conformance*, i.e. all output events produced by the circuit are expected by the environment. These criteria can be satisfied with the timing constraints that appear in Figure 8.

B. Asynchronous pipeline

We have also verified an asynchronous pipeline with different number of stages and an environment running at a fixed frequency. The processing time required by each stage has different min and max symbolic delays. The safety property being verified in this case was *"the environment will never have to wait before sending new data to the pipeline"*. Figure 9 shows the pipeline, with an example of a correct and incorrect behavior. The tool discovered that correct behavior can be ensured if the following holds:

$$d_{IN} > D_1 \wedge \ldots \wedge d_{IN} > D_N \wedge d_{IN} > D_{OUT}$$

where D_i is the delay of stage i, and d_{IN} and D_{OUT} refer to environment delays. This property is equivalent to:

$$d_{IN} > max(D_1, \ldots, D_N, D_{OUT})$$

Therefore, the pipeline is correct if the environment is slower than the slowest stage of the pipeline. CPU time for the different lengths of pipeline can be found in Figure 9.

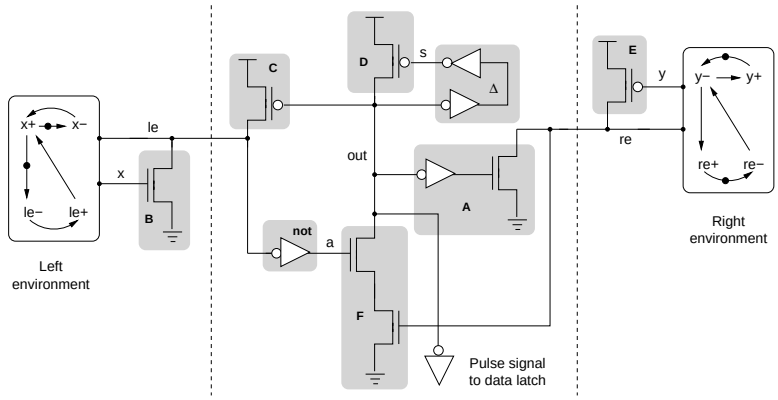
C. Other examples

We have also verified a set of asynchronous circuits available in the literature, defined as a network of simple gates plus a STG modeling the behavior of the environment. In these circuits, correctness has been defined as *absence of hazards* and *conformance* with the STG. Table I shows the size of the circuits, STGs and the computed TTSs, the number of symbolic delays, the number of constraints required for correctness, and the CPU time used for the verification.

VII. CONCLUSIONS

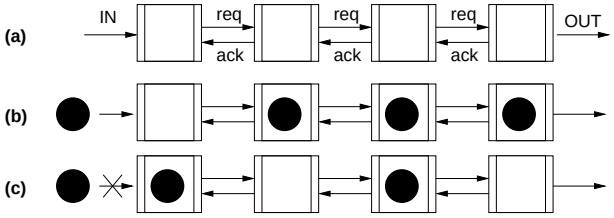
An algorithm for symbolic timing analysis of concurrent systems has been presented. The output of the algorithm is a conservative approximation of the values of clocks and symbolic delays in the reachable states of the system. An application has been shown by computing the constraints of gate and input delays in an asynchronous circuit that guarantee correct behavior. Remarkably, the approach works for more than 15 symbolic delays within a reasonable time.

The technique is well suited for analyzing small-sized timed circuits such as asynchronous controllers. These circuits often operate at very high throughputs, and they heavily rely



$$
\begin{aligned}
\delta(y-)+\delta(A) &> \delta(D)+\delta(\Delta)\\
\delta(x+)+\delta(B)+\delta(F) &> \delta(y-)+\delta(y+)+\delta(A)\\
\delta(x+) &> \delta(\Delta)+\delta(D)+\delta(not)\\
\delta(not)+\delta(B)+\delta(F) &> \delta(x-)\\
\delta(D)+2\delta(\Delta) &> \delta(y-)+\delta(A)+\delta(E)\\
\delta(\Delta) &> \delta(not)+\delta(C)\\
\delta(D)+\delta(\Delta) &> \delta(A)\\
\delta(x+)+\delta(B) &> \delta(D)+2\delta(\Delta)\\
\delta(y+) &> \delta(E)\\
\delta(x-) &> \delta(B)
\end{aligned}
$$

Fig. 8. GasP FIFO controller. Each shaded area has been modeled with a different symbolic delay. On the right, the discovered timing constraints that are sufficient to guarantee the correct operation of the circuit.



# of stages	TTS States	TTS Trans	# of symbols	CPU Time (seconds)
2	36	88	8	0.6
3	108	312	10	2
4	324	1080	12	13.5
5	972	3672	14	259.2

Fig. 9. (a) Asynchronous pipeline with N=4 stages, (b) correct behavior of the pipeline and (c) incorrect behavior. Dots represent data elements. On the right, the CPU times required to verify pipelines with different number of stages.

on stringent timing constraints to ensure a correct behavior. However, more complex circuits can also be verified if (a) they are analyzed at a higher level of abstraction or (b) part of the delays are defined as ranges of known integer delays instead of symbols. Future work will try to broaden the area of application of this technique, in order to handle bigger circuits with more symbolic delays. We plan to use representations based on Binary Decision Diagrams to represent sets of states and timing constraints symbolically.

Acknowledgments

This work has been partially supported by CICYT TIC2001-2476 and ACiD-WG (IST-1999-29119).

REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, pp. 3–34, 1995.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] A. Arnold. *Finite Transition Systems.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[4] W. J. Belluomini and C. J. Myers. Timed circuit verification using TEL structures. *IEEE Transactions on CAD*, 20(1):129–146, 2001.

[5] S. Chakraborty, D. L. Dill, and K. Y. Yun. Min-max timing analysis and an application to asynchronous circuits. *Proceedings of the IEEE*, 87(2):332–346, 1999.

[6] T.-A. Chu. *Synthesis of self-timed VLSI circuits from graph-theoretic specifications.* PhD thesis, MIT, June 1987.

[7] J. Cortadella, M. Kishinevsky, S. M. Burns, A. Kondratyev, L. Lavagno, K. S. Stevens, A. Taubin, and A. Yakovlev. Lazy transition systems and asynchronous circuit synthesis with relative timing assumptions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(2):109–130, 2002.

[8] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symp. on Principles of Programming Languages*, pp. 238–252. ACM Press, New York, 1977.

[9] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *ACM Symp. on Principles of Programming Languages*, pp. 84–97. ACM Press, New York, 1978.

[10] G. Dantzig and B. Eaves. Fourier-motzkin elimination and its dual. *Journal of combinatorial theory*, 14:288–297, 1973.

[11] D. Dill and H. Wong-Toi. Verification of real-time systems by successive over and under approximation. In *Proc. of the Conf. on Computer Aided Verification*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[12] N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.

[13] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In *Proc. REX Workshop Real-Time: Theory in Practice*, volume 600, pp. 226–251. LNCS, New York, 1992.

[14] H. Kim, P. Beerel, and K. Stevens. Relative timing based verification of timed circuits and systems. In *Proc. 8th Int. Symp. on Asynchronous Circuits and Systems*, 2002.

[15] L. Lavagno, K. Keutzer, and A. L. Sangiovanni-Vincentelli. Synthesis of hazard-free asynchronous circuits with bounded wire delays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1), 1995.

[16] M. A. Peña, J. Cortadella, A. Kondratyev, and E. Pastor. Formal verification of safety properties in timed circuits. In *Proc. Int. Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 2–11, 2000.

[17] C. Piguet et al. Memory element of the master-slave latch type, constructed by CMOS technology. US Patent 5,748,522, 1998.

[18] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins. Asynchronous Interlocked Pipelined CMOS Circuits Operating at $3.3-4.5$GHz. In *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, pp. 292–293, 2000.

[19] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Proc. Int. Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 46–53, 2001.