Verification of Timed Circuits with Symbolic Delays

Robert Clarisó

Department of Software
Universitat Politècnica de Catalunya
e-mail: rclariso@lsi.upc.es

Jordi Cortadella

Department of Software
Universitat Politècnica de Catalunya
e-mail: jordi.cortadella@upc.es

Abstract— Verifying timed circuits is a complex problem even when the delays of the system are fixed. This paper deals with a more challenging problem, the formal verification of timed circuits with unspecified delays represented as symbols. The approach discovers a set of sufficient linear constraints on the symbols that guarantee the correctness of the circuit. Experimental results from the area of asynchronous circuits show the applicability of the approach.

I. Introduction

The correct operation of a timed circuit often depends on the delays of its gates and wires and the timing behavior of its environment. Timing analysis can check the correctness of the circuit but the result is only valid for the particular timing information provided for that instance of the circuit. The answer could not be extrapolated to the same circuit implemented in other technologies. A more meaningful answer would be a characterization of the circuit as a set of timing constraints that could guarantee the correctness of the circuit and that would be independent of the technology. For example: *The circuit is correct if*

$$\begin{aligned} D_1 + D_2 &< d_3 + d_4 + d_5 \quad \wedge \\ D_2 &< 3d_1 + 2d_7 \end{aligned}$$

where d_i (D_i) denotes the minimum (maximum) delay of a gate i. The advantage of this type of answer is obvious. Apart from the useful feedback provided to the designer, the correctness of a set of delays can be checked automatically and efficiently just by testing if the delays satisfy the constraints. Many technology mappings can be tested efficiently instead of choosing conservative delays to ensure correctness. However, this requires an analysis with *symbolic* delays, that makes verification 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* [7], that was originally devised for the static analysis of programs [8]. One of the main motivations of this work is the characterization of the behavior of asynchronous controllers. Under certain gate delays, these circuits 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.

$$delay(path1) < delay(path2)$$

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, e.g. [19]. But 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 a simple 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) [18]. Each gate g_i has a symbolic delay in the interval $[d_i, D_i]$. We call T_{setup}, T_{hold} and $T_{CK\to Q}$ 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.

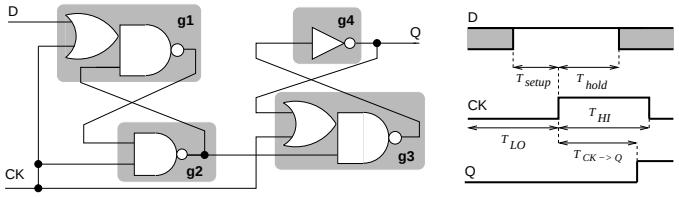
The method presented in this paper is capable of deriving a set of sufficient linear contraints that guarantee the correctness of the latch's behavior. The verified property is the following:

> *The value of Q after a delay $T_{CK\to Q}$ 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 failure. 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



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

(c)

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

(d)

Fig. 1. (a) Implementation of a D flip-flop [18], (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 $g2 = [1,3]$.

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. Many techniques rely on analyzing the circuit with known constant min-max delays in gates and wires [4, 5, 16, 17]. *Time-symbolic simulation* [14] assumes that each gate has an unknown constant delay, but the analysis requires that a lower and upper bound is known. Contrary to these approaches, the technique presented in this paper can deal with completely unknown delays that are represented as *symbols*.

The kind of timing constraints that can be computed also differs from our approach. The first class of constraints is *metric timing* constraints [5,16], i.e. constant min-max bounds for the components of a circuit and its inputs events. Another group of constraints is *relative timing* [4, 15, 17], 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.

Few techniques allow the analysis of asynchronous circuits with symbolic delays. *Parametric difference bound matrices* [2, 13] and *Presburger arithmetic* [1] handle symbolic delays in the verification of timed automata and timing diagrams, respectively. However, these approaches exhibit a high complexity that limits the number of states and symbolic delays of the examples: at most 5 symbols with parametric DBMs, and 12 symbols with Presburger formulas. Another approach avoids this complexity by selecting concrete constant values for the symbolic delays using integer linear programming [20]. The circuit is analyzed with constant delays, and if a reachable failure is found, the constant delays are updated to make it unreachable using integer linear programming. A shortcoming of this method is that the result is a set of metric timing constraints.

Our approach uses convex polyhedra as the abstraction to represent sets of timed states in timed transition systems. Convex polyhedra have also been used in the analysis of systems with different semantics: sequential programs [8], real-time systems [10], linear hybrid automata and synchronous programs with counters [11]. All these methods are approximate, as some operations on polyhedra are approximate to preserve closedness. For example, the union of convex polyhedra is not necessarily convex, and as an overapproximation, the convex hull is used instead.

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 e will be denoted by $[d_e, D_e]$. Intuitively, these delay bounds indicate that if an event e is fired t time units after becoming enabled, then $d_e \leq t \leq D_e$. The following definitions present the concepts of transition system and TTS, together with the semantics of these models, i.e. the concept of "run". These definitions can be easily extended to allow symbolic delays in addition to constant delays.

Definition IV.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 IV.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 IV.3 *[12] 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 IV.4 *[12] 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 $\mathbb{R}^+$, $t_1, t_2, t_3, \ldots$ 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 IV.5 *[12] 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*: $\forall e \in \Sigma, i \geq 0, j \geq i : t_j < t_i + d_e : (s_j \xrightarrow{e} s_{j+1} \in \sigma) \to (e \in \mathcal{E}(s_i))$.*
- upper bound*: $\forall e \in \Sigma, i \geq 0 : \exists j \geq i : t_j \leq t_i + D_e : e \notin \mathcal{E}(s_i) \vee (s_j \xrightarrow{e} s_{j+1} \in \sigma)$.*

```
Algorithm AbstractInterpretation (R, Inv)
Input: A timed transition system R = ⟨⟨S, Σ, T, s_in⟩, d, D⟩
with an invariant Inv defining constraints on symbolic delays.
Output: The abstraction Time for all states.

  foreach state s ∈ S do Time(s) := ∅; endfor
  Time(s_in) := Inv;
  changed := {s_in};
  do
    n := state in changed with lowest DFS number;
    changed := changed \ {n};
    foreach transition n --e--> m ∈ T
      newTime := transfer(n, e, m);
      if (newTime ⊆ Time(m)) continue;
      newTime := newTime ∪ Time(m);
      if (e is a back edge)
        Time(m) := (Time(m) ∇ newTime) ∩ Inv;
      else
        Time(m) := newTime ∩ Inv;
      changed := changed ∪ {m};
  while (changed ≠ ∅);
```

Fig. 2. Abstract interpretation algorithm

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 are updated accordingly. Analysis of the values of event clocks can reveal whether an event can be fired or not in a given state.

We present an algorithm that computes a conservative upper approximation of the event clock values, *convex polyhedra*. Approximations will be propagated and combined using fixpoint techniques described in abstract interpretation [7]. The following sections describe the different parts of the algorithm: the abstract interpretation techniques (B); the operations on convex polyhedra (C); and finally, the function that updates the clock values after firing an event (D).

B. Abstract interpretation

Abstract interpretation [7, 8] 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:* The state of a system 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 irrelevant information.
- *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.

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. The locations of interest of our timing analysis of will be the states of the TTS. We will note the abstraction in a given state s as $\mathsf{Time}(s)$. This abstraction describes the values of clocks when a state is reached, i.e. the *precondition* of the state.

In order to define the timing behavior of the system, we have to build a system of equations 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* (see section D) that advances the clock values while satisfying all upper and lower bounds. The output of this function is the value of clocks after firing an event, i.e. the *postcondition* of the transition being taken. Using this function, the abstractions for states can be defined as the following system of equations:

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

Fig.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. an empty abstraction. The algorithm applies the equations iteratively as long as they add new valid assignments. The solution is reached when there is a fixpoint.

Termination, i.e. convergence of the system of equations, is guaranteed by modifying the computation for loops. A *widening* operator [8] is used in the the equations of those states that are the targets of back-edges, i.e edges closing loops. Intuitively, widening extrapolates the effect of iterating a loop an unknown number of times. An in-depth discussion on termination of fixpoints and the necessity of widening can be found in [7, 8].

C. Convex polyhedra

Convex polyhedra [8, 11] 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 $B \in \mathbb{Q}^m$. 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 set of operations on convex polyhedra that are required for timing analysis are the following:

- **Test for inclusion** ($P \subseteq Q$): 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 $\forall r \in R : Ar \geq 0$.

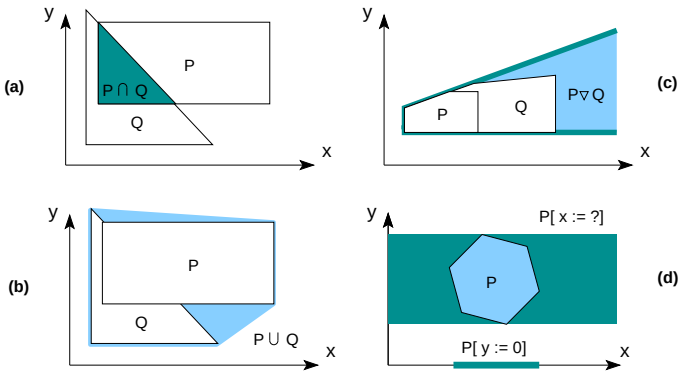


Fig. 3. Several operations on convex polyhedra: (a) intersection of polyhedra , (b) union of polyhedra as the convex hull, (c) widening of polyhedra and (d) assignment of a linear expression or an undefined value.

- **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** ($P \bigcap Q$): $P \bigcap Q$ is defined as the polyhedron with a system of linear inequalities that contains all the inequalities in P and Q.
- **Widening** ($P \nabla Q$): Widening is the extrapolation operator used to guarantee termination in loops. $P \nabla Q$ is defined as the system of linear inequalities which are satisfied both by P and Q.
- **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 is equivalent to an existential quantification: it 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 using the Fourier-Motzkin elimination method [9].

Fig.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 lose precision. All other operators are exact.

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.

```
Function transfer(src, e, dst)
Input: An event src --e--> dst.
Output: The postcondition of src --e--> dst.

  P := Time(src);
  P := P ∧ (step ≥ 0);
  P := P ∧ (clock_e + step ≥ d_e);
  P := P ∧ (clock_e + step ≤ D_e);
  foreach event e' ≠ e: e' ∈ E(src)
    P := P ∧ (clock_e' + step ≤ D_e');
  foreach event e' ≠ e: e' ∈ {E(src) ∩ E(dst)}
    P[clock_e' := clock_e' + step];
  foreach event e' ≠ e: e' ∈ E(dst) ∧ e' ∉ E(src)
    P[clock_e' := 0];
  foreach event e' ≠ e: e' ∈ E(src) ∧ e' ∉ E(dst)
    P[clock_e' :=?];
  if (e ∈ E(dst)) P[clock_e := 0];
  else            P[clock_e :=?];
  P[step :=?];
  return P;
```

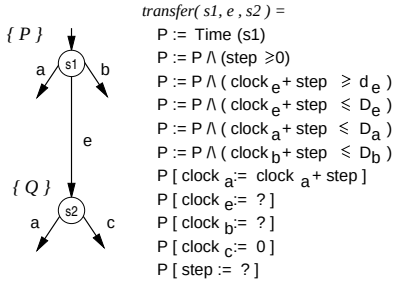
Fig. 4. Clock transfer function

Event clocks for enabled events store the amount of time elapsed since the event became enabled, while disabled clocks are undefined. After firing an event, event clocks should be updated to reflect the time elapsed between 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.

When an event e is fired, the clocks of other events have to be updated. The change in their clocks depends on whether they are enabled or disabled before and after firing e. Events that become newly enabled have their clock reset to zero, while events that become disabled have their clock undefined. If an event remains enabled before and after e, its clock is increased by the clock step. Finally, if an event remains disabled, its clock does not change.

Fig.4 describes the algorithm that computes the transfer function using convex polyhedra operators. Fig.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. This restriction is implied by the



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

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

constraints $clock_a + step \leq D_a, clock_e + step \geq d_e$, $clock_a = 0$, $clock_e = 0$.

E. Main algorithm

Timing analysis provides the required constraints for the reachability of the states of the TTS. However, we are looking for the complementary conditions, i.e. those 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 Fig.6.

After computing the untimed state space using depth-first reachability analysis, timing analysis can be performed on the TTS. Interestingly, the polyhedra computed for failure states describe the required costraints to reach the failures. If any of these constraints is false, the failure state will be unreachable. For example, if one polyhedron has the constraints $(a \leq b + c) \wedge (e \geq f)$, then the constraint $(a > b + c) \vee (e < f)$ ensures that the failure is unreachable.

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. Several simple heuristics can be used, such as "choose the constraint that removes the highest number of failures". Backtracking might be required to reconsider bad choices, as in [20]. 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.

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. Asynchronous pipeline

We have 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*". Fig.7 shows the pipeline, with an example of a correct and incorrect behavior. The tool discovered that correct behavior can be ensured if:

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

where D_i is the delay of stage i, and d_{IN} and D_{OUT} refer to environment delays. 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 Fig.7.

```
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.

  R := ReachabilityAnalysis(S,F);
  constraints := I;
  do
    AbstractInterpretation(R, 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 ∧ ¬c;
  while (any failure is reachable ∧ constraints ≠ false);
  {constraints = false → unavoidable failure}
  return constraints;
```

Fig. 6. Main algorithm for verification

B. Other examples

We have also verified a set of asynchronous circuits available in the literature, defined as a network of simple gates plus a Signal Transition Graph (STG) [6] modeling the behavior of the environment. In these circuits, correctness has been defined as *absence of hazards*, i.e once an event becomes enabled, it does not become disabled before being fired; an *conformance*, i.e. all output events produced by the circuit are expected by the environment. 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.

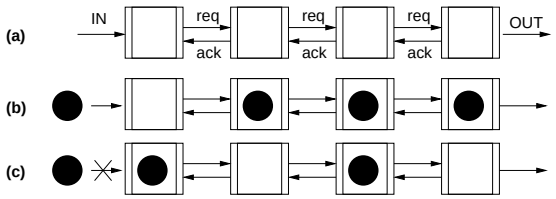
VII. Conclusions

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 asynchronous circuits 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 on stringent timing constraints to ensure a correct behavior. Future work will try to handle bigger circuits with more symbolic

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



# 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. 7. (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.

delays. We plan to use representations based on Binary Decision Diagrams to represent sets of states and timing constraints symbolically.

ACKNOWLEDGEMENTS

This work has been partially funded by CICYT TIC2001-2476, ACID-WG (IST-1999-29119), a distinction by the Generalitat de Catalunya and a travel grant from the DATE community and the EDAA association.

REFERENCES

[1] T. Amon, G. Borriello, T. Hu and J. Liu. "Symbolic timing verification of timing diagrams using presburger formulas." In *Proc. of Design Automation Conference*, pp. 226-231, 1997.

[2] A. Annichini, E. Asarin and A. Bouajjani. "Symbolic techniques for parametric reasoning about counter and clock systems". In *Proc. of Computer Aided Verification*, pp. 419–434, 2000.

[3] A. Arnold. *Finite transition systems*. Prentice Hall, 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] 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, 1977.

[8] 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, 1978.

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

[10] D. Dill and H. Wong-Toi. "Verification of real-time systems by successive over and under approximation." In *Proc. of Computer Aided Verification*, LNCS. Springer-Verlag, 1995.

[11] 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.

[12] 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, 1992.

[13] T. Hune, J. Romijn, M. Stoelinga and F. W. Vaandrager. "Linear parametric model checking of timed automata." In *Tools and Algorithms for Construction and Analysis of Systems*, pp. 189–203, 2001.

[14] N. Ishiura, Y. Deguchi and S. Yajima. "Coded time-symbolic simulation using shared binary decision diagram." In *Proc. of Design Automation Conference*, pp. 130–135, 1990.

[15] 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.

[16] 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.

[17] M. A. Peña, J. Cortadella, A. Kondratyev, and E. Pastor. "Formal verification of safety properties in timed circuits." In *Proc. Int. Symp. on Advan-ced Research in Asynchronous Circuits and Systems*, pp. 2–11, 2000.

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

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

[20] T. Yoneda, T. Kitai and C. Myers. "Automatic derivation of timing constraints by failure analysis." In *Proc. of Computer Aided Verification*, pp. 195–208, 2002.