Abstract — Most asynchronous compilers do not take into account timing constraints, although these are crucial for industrial interface applications. In this paper we extend the classical signal transition graph model such that timing constraints can be modeled in a concise way. We also propose synthesis algorithms that transform the initial specification such that the circuit derived from the transformed specification satisfies all timing constraints.

1 Introduction

The classical signal transition graph model (STG-model) [3, 12] has been proven to be an attractive model for the specification and synthesis of asynchronous controllers. Because it is based on Petri nets [7] it is very well suited to model concurrency and conditional behavior. Several synthesis techniques starting from the STG-model have been developed [3, 4, 12, 11].

The classical STG-model is intended for controllers in self-timed systems (also called self-timed controllers). In self-timed systems all system events occur in a particular sequence, but nothing ever has to occur at a particular time. In most interface applications the controllers do not follow the self-timed principles. The firing time of the transitions is constrained. For that reason the STG-model can be extended to represent timed systems, by connecting timing information to the arcs in the STG. This timing information is used to constrain the firing time of the transitions. This timing information can be interpreted in different ways. These different interpretations are called the firing semantics.

Not all arcs in a timed specification are “causal” arcs. There are two different types of relations in a specification, namely specification relations and delay relations. Delay arcs indicate how an existing circuit reacts, and specify its delay characteristics. Specification relations indicate the time range in which the synthesized circuit should react if specified correctly. These different interpretations given to the arcs are called relational semantics.

In Figure 1(a) part of the specification of a RAM-interface is shown. On the input signal WRITE a pulse will be generated (by the environment) of length between 30 ns and 100 ns. In response to this pulse, another pulse should be generated on the output signal CS of length between 80 ns and 200 ns. Moreover, the signal CS should have gone high before WRITE goes down again. How is this expressed by the arcs in Figure 1? In Figure 1, “delay arcs” are labeled with a “d” and “specification arcs” with an “s”. For example, the arc between the up- and down-transition of the signal WRITE is a delay arc. It indicates that the environment will generate a pulse of duration between 30 ns and 100 ns.

A delay relation can also be used to characterize the response delay of an internally generated event. This is for instance the case for the delay arc WRITE\(^+\) \(\rightarrow\) CS\(^+\).

The specification arc CS\(^+\) \(\rightarrow\) CS\(^-\) indicates that on the output signal a pulse should be generated of length between 80 ns and 200 ns. This is a specification constraint in the sense that a circuit should be designed that generates a pulse within the specified interval.

An important synthesis problem is to come up with an implementation where all specification relations are satisfied (Section 5). This will be done by transforming the STG. Signals and/or arcs will be added to the STG in order to satisfy all the specification relations. This corresponds to the following design decisions:

- The decision on which transition of the clock (or possibly transition of another signal) signal transitions should be triggered in order to satisfy the specification relations.
- The addition of delay-lines in order to satisfy certain specification relations.

This means that these design decisions are treated in one consistent framework. All timing details, which are often overlooked by the designer, are taken into account.

The synthesis process is basically a two step process. In the first step arcs and/or signal transitions are added to satisfy the specification relations. In the second step transforma-
tions are performed to make the circuit race-free. Now, hazard-elimination techniques may be used to derive a glitch-free circuit that satisfies the constraints. In the example of Figure 1 we concentrate on the first step.

A possible STG that corresponds to an implementation of the STG of Figure 1(a) is shown in Figure 1(b). The delays are \( WRITE^+ \rightarrow CS^+ \), \( DELAY^+ \rightarrow CS^- \) and \( WRITE^- \rightarrow CS^- \) all represent the delay behavior of the logic implementing the STG. These delay arcs actually "implement" the specification relations. The specification relation \( WRITE^+ \rightarrow CS^+ \) is implemented by the delay arc \( WRITE^- \rightarrow CS^- \). The specification relation \( CS^+ \rightarrow CS^- \) is satisfied in a more complex way. First a delay-line is added to the STG. Its behavior is represented by the signal \( DELAY \) in the STG of Figure 1(b). Then the appropriate delay arcs were added that represent the behavior of the logic. In this case the specification relation \( CS^+ \rightarrow CS^- \) is satisfied by the delay arcs \( CS^+ \rightarrow DELAY^+ \) and \( DELAY^+ \rightarrow CS^- \). The implementation of the time-consistent STG is shown in Figure 2.

The organization of this paper is as follows. In Section 4, the timed STG-model is introduced in a formal way. In this section the firing semantics and relational semantics will be explained. In Section 5, the basic synthesis problems due to the introduction of timing constraints are explained, and solutions are proposed. A technique is proposed to transform a time-inconsistent STG into a time-consistent STG. The relation between the timed STG and the transformations to eliminate state assignment problems will be briefly discussed. In Section 6 a synthesis example is discussed.

2 Previous Work

The synthesis methods proposed by Myers and Burns are aimed at self-timed circuits but take into account timing to optimize or to size logic [6, 2].

Myers [6] proposes an event-rule based system for the synthesis of timed asynchronous circuits. This work only treats the problem of transforming the initial specification to satisfy the timing constraints in a restrictive way. It does not add delay-lines and does not change the edges on which signals are triggered.

Rokicki [8] has proposed a timed Petri net model with relational semantics for general Petri nets. This work concentrates on an efficient method to find the states that are unreachable due to timing constraints and a way to check whether the Petri net is time-consistent. It does not treat the problem of transforming the specification to satisfy all timing constraints.

Borriello proposes an asynchronous compiler that takes into account timing constraints during the synthesis process [1]. The transformations proposed in [1] are based on ad-hoc rules, which are difficult to verify formally. The major advantage of event graphs in comparison with our approach is that it can handle "synchronous” timing constraints (based on the number of clock cycles). Currently, this is not considered in the approach presented in this paper.

3 Classical Signal Transition Graphs

The STG-model [5] is commonly used to specify the behavior of asynchronous control circuits. A Signal Transition Graph (STG) is an interpreted Petri net, \( (T, P, F, m_0) \). \( T \) is a set of transitions, \( s^+ \) denotes the up-transition of the signal \( s \), \( s^- \) denotes the down-transition of the signal \( s \). \( P \) is a set of places which can be used to specify conflict or choice. \( F \) represents the flow relation between transitions and places; namely, \( F \subseteq (T \times P) \cup (P \times T) \). A marking is a non-negative integer labeling of the places \( m : P \rightarrow Z^+ \), denoting the number of tokens in a place \( p \). \( s^+ \) also denotes any transition of the signal \( s \).

A marked graph (MG) is a Petri net [7] where each place \( p \) has exactly one fanin and one fanout transition. It will be represented by graph \( G \), where the vertices correspond to transitions and the arcs to places.

Given a marking \( m \), a transition is enabled and may be fired from this marking if all its fanin places have at least one token in \( m \).

An STG with no state assignment problem is said to satisfy the CSC-requirement [3, 9].

4 Timed Signal Transition Graphs

In this section timed signal transition graphs will be formally introduced. In Subsection 4.1 the firing semantics will be briefly introduced. In Subsection 4.2 the relational semantics will be presented. Based on these definitions the notions of a time-consistent and time-inconsistent STG will be explained. In this paper only acyclic marked graphs are considered.

4.1 Firing semantics

In this section we will give a very brief introduction on the firing time assignment. For a more elaborate discussion the reader is referred to [9].

The numbers indicated next to the transitions in the net of Figure 3 correspond to a specific execution of the net. Each number indicates the time at which the transition fires (occurs) for that specific execution of the net. The assignment of firing times to transitions is represented by the function \( \vartheta : T \rightarrow R^+ \).

The min-max interval connected to each arc in this figure is used to constrain the firing times of the transitions. \( \min f(p) \) corresponds to the minimum value assigned to arc \( p \) while \( \max f(p) \) corresponds to the max-value assigned to arc
The are different ways in which the min- and max-values can constrain the firing times \[0\]. In this paper we will use the following equations:

\[
\begin{align*}
\max_{p \in \pi \cap \pi^p} \left( \min f(p) + \vartheta(s(p)) \right) & \leq \vartheta(t) \\
\vartheta(t) & \leq \max_{p \in \pi \cap \pi^p} \left( \max f(p) + \vartheta(s(p)) \right)
\end{align*}
\]

\(t\) refers to the set of all input places of \(t\), \(s(p)\) refers to the (unique) input transition of place \(p\).

It is clear that there usually is more than one valid firing time assignment. In the sequel it is necessary to take into account all possible firing time assignments. So the complete set of firing time assignments that satisfy the constraints imposed by the places in the set \(P\) is represented by \(\Theta(\Sigma, P)\). Note that we are considering a subset \(P'\) of the set of all places \(P\). \(P'\) will be associated with the delay places or with the specification places in the next section.

### 4.2 Relational semantics

A timed STG is obtained by assigning a min-max timing interval to each place and by partitioning the set of places \(P\) in a set of places corresponding to the delay relations \(P_d\) and a set of places corresponding to the specification relations \(P_s\).

An STG is called time-consistent if all its specification relations are automatically satisfied, when only the delay relations are taken into account. In other words, all possible executions of the net that are imposed by the delay relations should automatically satisfy the constraints imposed by the specification relations.

An STG \(\Sigma\) is time-consistent if

\[\Theta(\Sigma, P_d) \subseteq \Theta(\Sigma, P_s)\]

So the set of all firing time assignments, which are valid for the delay relations, should be a subset of the set of all firing time assignments, which are valid for the specification relations.

In the algorithms of Section 5.2 a function \(\beta\) will be used. This function has the following meaning. Let \(p\) be a place corresponding to a specification relation between two transitions \(t_1\) and \(t_2\). \(\min f(p)\) and \(\max f(p)\) are assigned to place \(p\). The minimum and maximum time distance between \(t_1\) and \(t_2\) based only on the delay relations should reside in the interval indicated by \(\min f(p)\) and \(\max f(p)\). This is denoted by \(\beta(\Sigma, s) = 1\) (An algorithm that performs this timing analysis is presented in \([5, 9]\).)

### 5 Synthesis of Timed STGs

After having defined the firing and relational semantics of a timed STG, it will be discussed how this reflects on the synthesis of a timed asynchronous controller.

An overview of the synthesis process is given in Figure 4. In the initial STG specification only input-delay relations and specification relations are present. No output-delay relations are present yet. Output-delay relations are delay-relations pointing to transitions of output signals, input-delay relations are delay-relations pointing to transitions of input signals.

- The input-delay relations represent the (known) behavior of the signals produced by the environment. These signals are input signals for the synthesized circuit.
- The specification relations specify in which range in the time domain the controller should react.

During synthesis the controller decides where and how to add new circuitry. This is performed at STG level by adding delay and specification relations and possibly by adding new signals with new transitions. This synthesis process consists of two steps (see Figure 4).

- In the first step relations are added to make the STG time-consistent.
- In the second step relations are added to make the STG satisfy the CSC requirement \([9]\).

This section is organized as follows. The specific reasons why the three types of relations are added to the STG, are discussed in Subsection 5.1. In Subsection 5.2 an algorithm that transforms a time-inconsistent STG into a time-consistent STG is derived. In Subsection 5.3 the relation between the timed STG and the transformations that satisfy the CSC requirement are presented.

#### 5.1 Addition of new relations to an STG

##### 5.1.1 Adding output-delay arcs

The exact delay characteristics of the logic are only known when the logic has been generated. Unfortunately, no logic can be generated before the CSC requirement is satisfied. Output-delay relations are added to satisfy the CSC requirement. So this is a classical chicken-egg problem. To break this vicious circle the following simplification is proposed. The user should give an estimate of the delay of the logic, and the compiler uses this estimate during the synthesis of the STG. All the transformations on the STG are performed based on this estimate. Once the logic is technology mapped to a certain library a timing analysis tool \([8]\) checks whether the assumptions made by the user are correct. If not, the user is forced to adapt the assumption made for the output-delay relations or he may even be forced to change the original specification. So there is a loop over the synthesis process. This is schematically presented in Figure 4. An example of this process is discussed in Section 6.
It is assumed that all output-delay relations are always assigned a minimum value and a maximum value that is set by the user. These values will be referred to as $\text{mindelay}$ and $\text{maxdelay}$. The user is discouraged to use a non-zero $\text{mindelay}$ value.

### 5.1.2 Adding specification relations

Specification relations may be added by the compiler. This means that the compiler is tightening the specification set by the original specification relations. Intuitively this is done to specify additional precedences that have to be satisfied. These additional precedences are useful to satisfy the CSC requirement. An example is presented in Figure 5(a). Assume that the synthesis algorithm, which transforms the STG to satisfy the CSC requirement, requires putting $u^-$ between $a^+$ and $b^+$. So a specification relation has been placed between $u^-$ and $b^+$ (Figure 5(b)). This has to be a specification relation. An input-delay arc would change the behavior of the signal $b^+$, which is an input signal. It is not allowed to change the behavior of the environment. One could argue that it is not necessary to add the specification relation, because a simple timing analysis on the STG of Figure 5(a) shows that $u^+$ always precedes $b^+$. But in the STG of Figure 5(a) it is still allowed to add a delay relation from $b^+$ to $u^+$ which results in the STG of Figure 5(c). This STG is also time-consistent. In this case $u^+$ does not fire any more between $a^+$ and $b^+$. The addition of such a delay relation is actually prevented by the addition of the specification relation in Figure 5(b). If the system is time-consistent after the addition of a new specification relation the original system before the addition of the specification relation is also time-consistent. So this corresponds to a valid transformation.

### 5.1.3 Adding input-delay relations

The compiler may not add input-delay arcs to already existing signals, because it is not allowed to change the behavior of the environment. In particular situations however it may be necessary to add circuitry that does not consist of pure logic. Classical examples are delay-lines and arbitration circuits. In most practical cases the terminal behavior of these circuits can be modeled by an STG. So the behavior of the complete circuit (= logic circuit and special circuits) can be modeled in one STG. The special circuit, once it is added, is considered to be part of the environment. So its output signals are considered to be input signals for the synthesized logic circuit. Hence if the compiler wants to add special circuitry, input-delay arcs have to be added to the STG.

![Figure 5: (a) A time-consistent STG. (b) The same STG with a valid specification arc added. (c) The transformation that is prevented by the arc added in (b).](image)

### 5.2 Transformations to satisfy the specification relations

A procedure is presented that tries to satisfy the specification relations. Assume that there is a specification relation (place) $p$ between $t'$ and $t$ that is not satisfied. The procedure tries to satisfy this relation by adding delay arcs pointing to $t'$. The added delay arc is assigned a minimum value $\text{mindelay}$ and a maximum value $\text{maxdelay}$ determined by the user. Now
every $t^n$, that can be used as a starting node for the delay arc, should satisfy the following relation:

$$\forall \theta \in \Theta(P_d, \Sigma_S) : \quad \hat{\theta}(t) + \min f(p) \leq \hat{\theta}(t^n) + \min delay$$

$$\hat{\theta}(t^n) + \max delay \leq \hat{\theta}(t) + \max f(p) \quad (2)$$

This can also be rewritten as

$$\min f(p) \leq \min (\hat{\theta}(t^n) - \hat{\theta}(t)) + \min delay$$

$$\max (\hat{\theta}(t^n) - \hat{\theta}(t)) + \max delay \leq \max f(p) \quad (3)$$

The above condition is a necessary and sufficient condition to satisfy that particular specification relation. Other specification relations are not taken into account however. So after the transformation is performed, a timing analysis algorithm should check whether no specification relations that were initially satisfied, are now violated. If this is the case the transformation should be rejected. If there are more alternatives, it is a good strategy to take the transition $t^n$ which has the lowest $\max (\hat{\theta}(t^n) - \hat{\theta}(t))$. This gives more freedom to the compiler to perform transformations afterwards.

It is possible that no event $t^n$ can be found that satisfies relation 3. As explained above, such an event should be created by adding a delay-line. Assume that the delay-line has a $(\min_{in}, \max_{in}]$ characteristic. For the example of Figure 6(b) this is $[50, 70]$. Signal $s$ can serve as an input for the delay-line if this signal has a transition $s^+$ that satisfies the following condition.

$$\min f(p) \leq \min (\hat{\theta}(s^+) - \hat{\theta}(t)) + \min delay + \min_{in}$$

$$\max (\hat{\theta}(s^+) - \hat{\theta}(t)) + \max delay + \max_{in} \leq \max f(p) \quad (4)$$

If a number of alternatives are available, it is a good approach to choose the solution with the smallest delay-line. Small delay-lines are usually more reliable than the bigger ones and consume less area.

The global procedure then becomes as follows:

**Algorithm 1 (Making an STG time-consistent)**

Make Time Consistent ($\Sigma_S$)

```
{ pset = \emptyset /* the set of specification rel. already satisfied */
  FOREACH (p \in P_s)
    if \theta(\Sigma_S, p) == t
      pset = pset \cup \{p\};
  FOREACH (p \in (P_s \setminus pset)) /* p is not satisfied yet */
    if (newt == NULL)
      { FOREACH (delay-line in LIBRARY from small to big)
        if (newt == NULL)
          { newt = FindTriggerTransition (\Sigma_S, p);
            if (newt == NULL)
              { newt = NULL; /* no solution */
                return \Sigma_S;}
          }
        if (newt == NULL)
          { newt = NULL; /* no solution */
            return \Sigma_S;
          } else
            { \Sigma_S = \Sigma_S + \text{delay arc added from newt to t'};
              pset = pset \cup \{p\};
              FOREACH (p \in pset)
                if (\theta(\Sigma_S, p) == t')
                  newt = NULL; /* no solution */
                  return \Sigma_S;
                else
                  \Sigma_S = \Sigma_S + \text{delay arc added from newt to t'};
                  newt = NULL; /* no solution */
                  return \Sigma_S;
              }
            }
        } break;
      }
  } } return \Sigma_S;
```

The following algorithm tries to find a signal transition for adding delay arc. FindTriggerTransition returns a signal transition newt such that if a delay arc is added between newt and $t$ the specification relation $p$ may be satisfied.

**Algorithm 2**

FindTriggerTransition ($\Sigma_S, p$) { 

```
p = a place between $t$ and $t'$; FOR EACH ($t'' \in T$)
  { if ($t''$ satisfies relations 3)
    { if $\max (\hat{\theta}(t'') - \hat{\theta}(t)) \leq \max$
      { max = $\max (\hat{\theta}(t'') - \hat{\theta}(t))$;
        newt = $t''$;
        return newt;
      }
    }
  }
```

For a discussion of the running time the reader is referred to [9].

### 5.3 Transformations to satisfy the CSC-requirement

For these transformations it is assumed that the initial STG from which the transformation process starts, is time-consistent. So during the transformation process the compiler has to check after each transformation whether the STG is still time-consistent. If it is not time-consistent it means that the transformation, which the compiler has performed, is not a valid one. The compiler then searches for another transformation that results in a time-consistent STG. If no such transformation can be found, the compiler cannot find a solution within the constraints and within the user estimate for the maxdelay and mindelay values. The user therefore has to adapt his estimation of maxdelay and mindelay values or he may even be forced to change the initial specification. This process is schematically represented in Figure 4.

### 6 Case study

The initial timed STG for an up-down counter is shown in Figure 8. There are two conditional paths: $a$ goes high or $b$ goes high first. $d_1$ and $d_2$ represent the behavior of two delay-lines. Although this is a cyclic graph, the analysis and synthesis can be performed on the two acyclic parts which constitute this STG.

This STG does not satisfy the CSC requirement. Therefore a state signal has to be added using techniques presented in [9]. With this example we want to demonstrate that different solutions will be found dependent on the choice of the delay-max parameter.

Assume that the user sets delay-max = 10 ns. Part of the resulting STG is presented in Figure 9(a). The implementation is shown in Figure 10(a). A new transition $H^*$ of a state signal $H$ has been added to resolve the state assignment...
conflicts. For more information on satisfying state assignment conflicts the reader is referred to [9]. Note that the new transition $H^+$ has been added in a very “strict” way. It is for instance specified that down should go low and next $H$ should go high before $a$ goes high. Dependent on the library this implementation might not satisfy these requirements. This can be verified with a timing verifier like the one presented in [8]. The user might therefore adapt his estimation for delay-max. Assume that the user now sets delay-max = 15 ns. Part of the resulting STG is presented in Figure 9(b). The implementation is shown in Figure 10(b). The transition $H^+$ now has been added in a less “strict” way. It can be verified now that the resultant logic becomes slightly more complex. If this implementation still does not satisfy all the requirements, the user can set delay-max = 24 ns. Part of the resulting STG is presented in Figure 9(c). The implementation is shown in Figure 10(c). Again the resultant logic becomes slightly more complex. If this implementation still does not satisfy the assumptions, the user is forced to change the specification or to take another library of gates.

Figure 8: (a) Initial timed STG for an up-down counter.

Figure 9: (a) Part of the transformed STG with delay-max = 10. (b) Part of the transformed STG with delay-max = 15. (c) Part of the transformed STG with delay-max = 24.

Figure 10: (a) Implementation of the STG in Figure 9 (a). (b) Implementation of the STG in Figure 9 (b). (c) Implementation of the STG in Figure 9 (c).

7 Conclusions

In this paper, a number of extensions have been discussed to model time in the STG-model. Based on these new semantics a transformation technique is discussed to transform an STG in such a way that the circuit derived from the STG satisfies all the specified timing constraints.

References


