A Visual Formalism for Real Time Requirement Specifications

Konrad Feyerabend[1*] and Bernhard Josko[2**]

[1] Dept. of Computer Science, Oldenburg University, D-26111 Oldenburg, Germany
[2] OFFIS, Escherweg 2, D-26121 Oldenburg, Germany

Abstract. This paper presents a semantical basis of a graphical specification language, called *real-time symbolic timing diagrams (RTSTD)*, to express real-time requirements of embedded systems. RTSTD allow a concise and unambigous formulation of real-time properties that are intuitively understandable by hardware designers. We give a precise semantical foundation of this graphical language in terms of real-time temporal logic. Due to this interpretation RTSTD can be embedded into existing verification tools to check whether an implementation satiesfies the given specification expressed as RTSTD.

1 Introduction

Nowadays embedded control systems can be found in varying areas such as automotive, production technique, plant control and avionics. Essential for these control systems are their *reactive* nature. Furthermore, these systems are often safety-critical where an error may lead to catastrophic results for people, material asset, and environment. These safety requirements include real time constraints: the control system has to react to asynchronously occurring events within a given period to ensure a fault free behaviour of the system. For example, during a car crash the inflation of the airbag should neither be triggered too early nor too late to protect the driver. If the airbag does not work correctly the injury caused may be worse than without any safety device.

To increase reliability of safety-critical systems formal proofs that requirements are indeed satisfied are highly demanded. To prove properties of control-dominated reactive systems model checking techniques have been developed in the past. Since the introduction of symbolic, BDD-based techniques [5] these methods have been successfully applied in industrial designs. Using model checking, requirements are expressed in temporal logics. But temporal logics formulae are not well accepted by engineers who require a more readable specification language. To meet these needs a user-friendly notation called symbolic timing diagrams (STD) has been developed [6] that resemble the graphical appeal of conventional timing diagrams as widely used in hardware design. STD have a

* Now at Ericsson Eurolab Deutschland GmbH, D-31135 Hildesheim, Germany
** Part of this work has been funded by the Commission of the European Communities under the ESPRIT project 20897, SACRES.

Cadence Design Systems, Inc. - Ex. 1007, Page 00001
IPR2024-00356 (Cadence Design Systems, Inc. v. Semiconductor Design Technologies, LLC)

well-defined semantics given in temporal logics and have been used successfully in several applications within the ESPRIT project FORMAT [14] by industrial partners.

To deal with real time constraints, timed automata and real time temporal logics have been investigated in recent years [2,1], and there are already quite a few model verification environments available (e.g. [7,15,3]) that can handle real time. But adding real time to temporal logics makes the specifications even more difficult to read and makes the need for an intuitive specification style even more pressing.

This paper deals with the extension of STD to real time. We give a formal semantics of these timing diagrams based on real-time temporal logics. While the extension of the graphical notation of STD to include timing constraints is quite obvious, its semantical interpretation is not. In the untimed version a requested event may be arbitrarily long delayed. Including timing information this is no longer possible. Due to this fact, we have to resolve possible conflicts of different types of constraints – whether a violation of a part of the constraints is interpreted as a failure or only as an exit of the diagram. The semantics given in this paper solves this conflict by introducing a priority on the constraint types. We will explain in more details and justify our decision in Sect. 5. Real time STD (RTSTD) will be used within an integrated tool suit for the specification and verification of VHDL [12] and StateMate [10] designs.

Related Work. Combining real time temporal logic with a graphical interface has been studied by Melliar-Smith et al. in [16] where the (purely graphical) real time future interval logic RTFIL is introduced. The graphical notation and the logics used there is quite different from our approach which makes comparison difficult. Dietz [8] uses duration calculus as semantic foundation of her graphical specification language. While duration calculus is extremely expressive it is shown to be undecidable and thus not suitable for automated proof.

Formalizing timing diagrams for interface specifications has been studied by Borriello ([4]). Khordoc and Cerny [13] study realizability of timing diagrams and their use in testing designs. To the best of our knowledge, no formal semantics based on temporal logics is given to timing diagrams that is suitable for model checking.

This paper is structured as follows: Section 2 introduces a running example used throughout the rest of the paper. Section 3 introduces the language RTSTD formally. Before we give a formal semantics to RTSTD in Sect. 5 we introduce the logic TPTL in Sect. 4.

2 An Example Real Time STD

Throughout this paper we will use a single real time symbolic timing diagram (RTSTD) shown in Fig. 1 as a running example. It is a simplified handshaking protocol between a sender and a receiver (cf. Fig. 2). In this timing diagram part of the requirements concerning the sender are given, i.e. we are considering open systems and a modular specification style.

Cadence Design Systems, Inc. - Ex. 1007, Page 00002

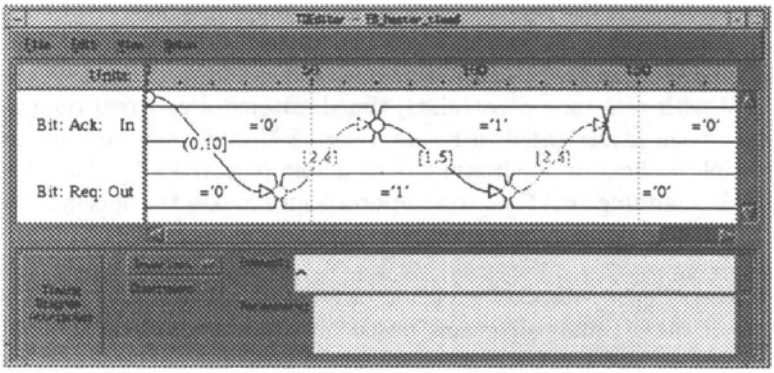

Fig. 1. Sender specification with real time constraints

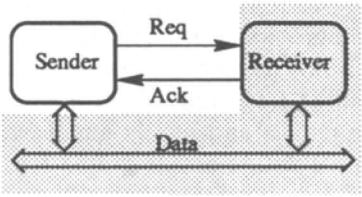

Fig. 2. The sender/receiver protocol. The shaded parts are not considered in the example

A RTSTD consists of four major parts:

- An *entity declaration ed* defining the interface of the component. It specifies the ports (or signals) (**Req**, **Ack**) of the system as well as their data type (Bit) and access mode (In, Out).
- A set of *waveforms*. Each waveform defines a sequence of values (or, more generally, a sequence of expressions) associated to the signals. The rising and falling edges in a waveform are called *(symbolic) events e*. A special *activation event* e_{act} is located to the left over the waveforms.
- A set of constraints between events of (different) waveforms. We distinguish *strong constraints*, given by black solid arcs which must be satisfied by the component under consideration, and *weak constraints*, given by shaded arcs, which denote assumptions on the behaviour of the environment.
- An *activation mode* ("Invariant" or "Initial"). An *initial* diagram describes requirements on the initial system state whereas an *invariant* diagram should be satisfied again and again (see below).

The example diagram describes the following requirements specified by strong constraints (solid arcs):

- Whenever both **Req** and **Ack** are not set, then **Req** must be set within 10 time units (indicated by the first solid arc). Hence, at no time both **Req** and **Ack** may be continuously unset for longer than 10 time units.

Cadence Design Systems, Inc. - Ex. 1007, Page 00003

- If the request is acknowledged (both **Req** and **Ack** are set) the sender must guarantee that **Req** is reset within 1 to 5 time units.

On the other hand, this protocol will only work correctly if the receiver that controls **Ack** meets the protocol definition, specified by weak constraints:

- We assume that **Ack** is not set before **Req**. Furthermore it is assumed that a request will be acknowledged within 2 to 4 time units. If **Ack** does not fulfill these requirements (i.e. **Ack** is set before **Req** or before two time units have elapsed, or it is not set within 4 time units after **Req**), we cannot continue with the normal protocol behaviour and the timing diagram is disabled, i.e. all the following requirements need not to be fulfilled.
- If both **Req** and **Ack** are set, **Req** is assumed to be reset first, and we assume that **Ack** is reset within 2 to 4 time units after request.

The activation mode of the example is "Invariant" indicating that the requirements imposed by the RTSTD must be met (the diagram is *activated*) whenever the leftmost conditions of all waveforms is matched by the actual values of the ports. In our example this happens if **Req=0** and **Ack=0**.

3 Syntax of RTSTD

In this section we give a formal definition of RTSTD, guided by the graphical notation wherever possible. Furthermore we introduce the notion of *phases* and *unwinding* that will be used to define the semantics of the language.

As introduced above a RTSTD over an entity declaration *ed* consists of an activation mode $A \in \{Initial, Invariant\}$, an activation event e_{act}, a set of waveforms W and a set of timing constraints RT:

$$TD \stackrel{\text{def}}{=} (A, W, e_{act}, RT)$$

Each waveform $\rho \in W$ consists of a set of *symbolic events* $\rho_1, \rho_2, \ldots$ which are linearly ordered from left to right. A topmost symbolic event $\top_\rho$ that is not depicted in the graphical notation is used to identify that a waveform has reached its end. Thus, the waveform identified with the **Ack** signal consists of the linear ordered symbolic events

$$Ack_1 <_{Ack} Ack_2 <_{Ack} \top_{Ack}$$

(e.g. Ack_1 is the rising edge from **Ack=0** to **Ack=1**). With every symbolic event we identify two propositions: An *activation condition actc* given by the condition to the left of the event, e.g.

$$actc(Ack_1) \stackrel{\text{def}}{=} \texttt{Ack=0} \ ,$$

and a *continuation condition contc* depicted to the right:[1]

$$contc(Ack_1) \stackrel{\text{def}}{=} \texttt{Ack=1}$$

We require that $contc(\rho_i)$ is equal to the activation condition of the successor event $actc(\rho_{i+1})$ if $\rho_{i+1} \neq \top_\rho$ which explains why there is only one expression depicted between adjoining events:

$$contc(Ack_1) \stackrel{\text{def}}{=} actc(Ack_2) \stackrel{\text{def}}{=} \texttt{Ack=1}$$

Furthermore we define

$$actc(\top_\rho) = contc(\top_\rho) = true \ .$$

Constraint Arcs. The constraint arcs RT are used to demand an ordering on the symbolic events of different waveforms and to impose timing requirements on events. There are four basic classes of constraints (cf. Fig. 3).

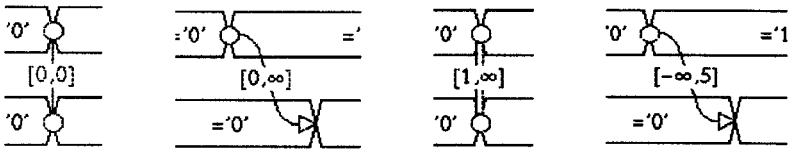

Fig. 3. Simultaneous, precedence, conflict$_1$ and leads-to$_5$ constraints

Two of them impose an ordering on the events: *Simultaneous*, $RT^{=}$, requiring that two symbolic events must be taken in the same unwinding step, and *Precedence*, $RT^{\leq}$, stating that the target event must not happen before the source event. The other two carry timing annotations and impose timing requirements: *Conflict*, $RT_t^{\neq}$, requires that the two events connected may not happen within the same step and that furthermore their occurrence times must be at least t time units apart, and *Leads-to*, $RT_t^{\rightsquigarrow}$, requiring that the target event may not happen later than t time units after the source event[2]. Each of these constraint classes come as weak or strong constraints indicating an assumed resp. required behaviour. Arcs in the graphical notation may be combined constraints. For example, the second solid arc in Fig. 1 is formed by conjunction of strong precedence, conflict$_1$ and leads-to$_5$ constraints, i.e. $(Ack_1, Req_2) \in RT^{\leq}, RT_1^{\neq}, RT_5^{\rightsquigarrow}$.

[1] In addition, the full language allows additional *exit conditions* that may be used to specify conditions that allow premature deactivation of the diagram. They are not used in the example and not considered in the rest of this paper. Refer to [9] for details.

[2] Note that that target event of a leads-to may well happen before the source event. No ordering is induced by a leads-to constraint.

Phases and Unwinding. When activated, a timing diagram restricts the behaviour of the specified system. These restrictions can be defined by sequences of *phases* derived by *unwinding* the timing diagram. Each phase ζ is characterized by the set of symbolic events that may occur next. I.e. a phase ζ contains exactly one symbolic event ρ_i of every waveform $\rho \in W$. ζ describes the phase before one (or some) of the events $E \subseteq \zeta$ occurs, hence the activation conditions $actc(e), e \in \zeta$ hold throughout the phase. For example, if the RTSTD of Fig. 1 is activated the first phase is denoted by $\{Req_1, Act_1\}$ and Req=0 $\wedge$ Ack=0 holds during this phase. If the continuation conditions *contc* of some events $E \subseteq \zeta_1$ become valid we may change to a successor phase by replacing the events E that took place by their waveform successor events E'. We call this an *unwinding step* from ζ_1 to $\zeta_2 = (\zeta_1 \setminus E) \cup E'$ of the set of waveforms W and say that ζ_2 is an E-successor of ζ_1. If we forget about the constraint arcs for a moment (that restrict the allowed unwinding steps) the phase $\{Ack_1, Req_1\}$ has three successor phases: We may unwind to $\{Ack_2, Req_1\}$ ($\{Ack_1, Req_2\}$, $\{Ack_2, Req_2\}$) if Ack (resp. Req, or both) changes from 0 to 1. Note that this successor relation on phases induces a partial order on the phases ζ_i. The waveform delimiters $\top_\rho$ must never be taken since they have no successor. Thus the very last phase of W is reached if ζ consists only of top elements $\zeta = \{\top_{\rho_i} \mid \rho_i \in W\}$.

4 Real Time Model

The formal semantics of RTSTD will be given by a translation to a timed propositional logic (TPTL, [2]) dialect that uses clocks and *reset quantification* as used e.g. in [11]. Timed automata [1] serve as an intermediate representation. We characterize a system by the sequence of observable signal changes over the time at its interface called *timed state sequence* $tss = (\bar{\sigma}, \bar{\tau})$ [2]. A timed state sequence consists of an infinite sequence of states $\bar{\sigma} = \sigma_0\sigma_1 \ldots$ where each σ_i maps each port of the component to a type correct value, and an infinite time sequence $\bar{\tau} = \tau_0\tau_1 \ldots$ with $\tau_i \in \mathbb{N}$ that maps each state in the sequence to a discrete time stamp. The time sequence $\bar{\tau}$ must be monotonous ($\forall i : \tau_i \leq \tau_{i+1}$) and unbounded ($\forall n \in \mathbb{N} : \exists \tau_i : \tau_i > n$).

TPTL. Timed PTL uses the operators of the untimed logic PTL such as $\Box\phi$ (always) ϕ_1 **until** ϕ_2 (ϕ_2 will hold somewhere in the future and ϕ_1 holds before) and ϕ_1 **unless** ϕ_2 (ϕ_1 holds as long as ϕ_2 does not hold). Arbitrary boolean expressions $Expr_{ed}$ over the ports of the entity declaration may be used as basic propositions of the logic. On top of PTL our logic uses clocks $z_i \in C$ that may be constrained within the formulae by clock constraints $\Phi(C) \stackrel{\text{def}}{=} z_1 \leq t_1 \mid z_1 + t_1 \leq z_2 + t_2$ ($t_i \in \mathbb{N}$ time constants) and reset to 0 by the *reset quantifier* $z.\phi$. (For the full definition and formal semantics see [2] for TPTL with freeze quantification and [9] for TPTL as used here.) For example, in

$$z.(\texttt{Req=0} \wedge \texttt{Ack=0}\ \textbf{until}\ \texttt{Req=1} \wedge \texttt{Ack=0} \wedge z \leq 10)$$

the clock variable z is reset to 0 at the beginning of the formula. While we wait for request to change its value the clock increases and we require that $z \leq 10$ when the change on request takes place (cf. Fig. 4).

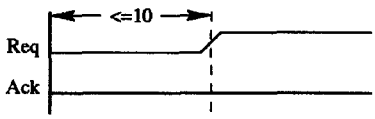

Fig. 4. Request must be set within 10 time units

5 Semantics of RTSTD

We define the formal semantics of RTSTD in three steps. First we translate the set of waveforms with the constraints imposed to an acyclic fragment of a timed Büchi automaton called ***unwinding structure.*** This unwinding structure is then translated to TPTL yielding the raw formula of the RTSTD body. In the last step we add the translation of the activation mode. The formula resulting is the characterization of the diagram in TPTL and may be used in formal proofs.

5.1 Unwinding Structure

The unwinding structure of a RTSTD *td* is a fragment of a timed Büchi automaton $(S, s_0, C, \mathcal{T}, F)$.

- The states S of the automaton correspond to the phases ζ_i of *td* plus a unique exit state ζ_{exit}.
- The start state s_0 is the first phase of the diagram, e.g. $\{Ack_1, Req_1\}$ in our example.
- The set of clocks C comprises one clock z_e for each symbolic event e in the waveforms of *td* as well as one clock z_{act} for the activation event e_{act}. If we do an unwinding step in the diagram all the clocks corresponding to the events taken in this step are reset.
- The set of accepting states $F \subseteq S$ is derived from the set of strong leads-to constraints. If there is some event e_1 that already took place connected by a strong leads-to constraint with an event e_2 that has not took place yet this indicates that e_2 must eventually happen. Thus we need to leave the actual phase — or in the translation the actual state — and thus need to exclude this state from F. Furthermore, the exit state ζ_{exit} that indicates that the diagram was deactivated is in F.
- Transitions between phases correspond to unwinding steps. (ζ_1, ζ_2, *cond*, *reset*, *timing*) is in the transition relation if ζ_2 is a E-successor phase of ζ_1, and all ordering constraint imposed by the simultaneous, precedence and conflict constraints are met.

cond states that for all symbolic events $e \in E$ that happened in this unwinding $contc(e)$ is met, and that the remaining events $e' \in \zeta_1 \setminus E$ are still activated ($actc(e')$). *reset* contains all the clocks z_e associated with the events $e \in E$ taken in this unwinding. These clocks are reset by the transitions, hence all constraints whose source is an event $e \in E$ can be checked in the future. Finally, the timing condition *timing* must ensure that all real time constraints imposed by conflict and leads-to constraints are met.

The left half of the automaton in Fig. 5 corresponds to the unwinding steps of the diagram. Since the precedence constraint arcs impose a linear order on the symbolic events

$$Req_1 \rightarrow Ack_1 \rightarrow Req_2 \rightarrow Ack_2$$

each phase has exactly one valid successor phase.

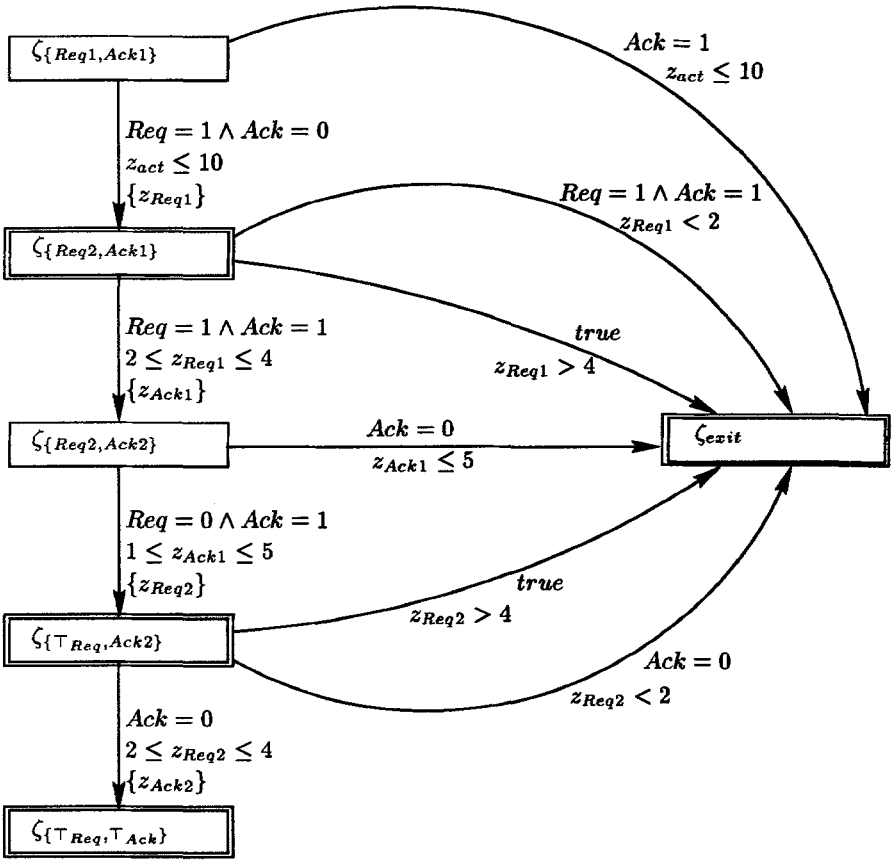

Fig. 5. The unwinding structure of the RTSTD in Fig. 1

The transition steps to ζ_{exit} in the automaton correspond to premature deactivation of the diagram due to the violation of some weak constraint. Recall that weak constraints are used to model assumption on how the environment will behave, and strong constraints to define how the component specified must behave. Thus, if both a strong and a weak constraint are violated at the same instance there is a malfunction in both the component and the environment. The faulty behaviour of the component must be detected, i.e. we disallow an exit step in this case. This policy is achieved by the *timing* and *cond* predicates associated with the transitions:

- The first weak constraint in Fig. 1 is violated if **Ack** changes to 1 before or simultaneous with **Req**. Thus a transition from $\zeta_{\{Req1,Ack1\}}$ to ζ_{exit} is added in the unwinding structure. But due to the first strong constraint a failure occurs if the diagram has been activated longer than 10 time units. Thus we have to ensure $z_{act} \leq 10$.
- If we change to phase $\{Req_2, Ack_1\}$ and **Ack** changes to 1 before two time units have elapsed, we may exit the diagram — but only if **Req** does not change to 0 at the same moment which causes a violation of a strong constraint.
- Timing violation of weak leads-to constraints happens if we stay in the phase longer than four time units.

This general rule that a violation of a strong constraint causes a failure despite the violation of a weak constraint has one exception. If both assumed and required upper timing bounds were not met (that is, both a strong and a weak leads-to constraint are violated) we have to take a closer look at the timing before deciding on exit or failure. Consider the example diagrams of Fig. 6 and the system behaviour of Fig. 7 which will result in a timed state sequence *tss*:

$$((x = 0, y = 0), \tau = 0),\ ((x = 1, y = 1), \tau = 1),\ ((x = 0, y = 0), \tau = 11)\ \ldots$$

At global time 11 where we observe the unsetting of the signals both the weak and the strong constraint are violated. But in the upper diagram the weak constraint was violated earlier in time, in the lower diagram the strong constraint. The translation procedure ensures that the corresponding transitions in the unwinding structure allow a transition to ζ_{exit} for the upper diagram but disallows it for the lower one (cf. Fig. 8) in the given situation. This is done by comparing the time elapsed since the timers z_{x1} and z_{y1} have reached their allowed upper bound.

5.2 Translation to TPTL

In a second translation step the unwinding structure described above is translated to PTL. Finally, the activation mode of the timing diagram is added. Translation of the unwinding structure is accomplished by the translation function TL on the states of the unwinding structure, starting in the first phase. We may stay in a phase as long as its activation condition ***actcond*** is valid, and leave

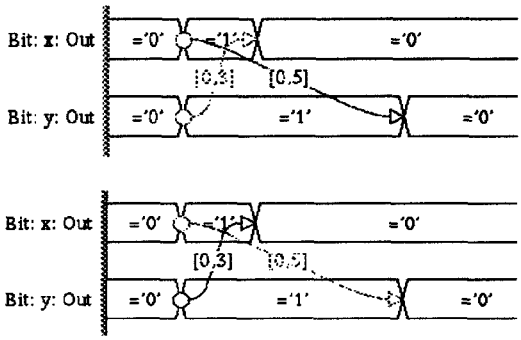

Fig. 6. Two RTSTD: The upper one assumes that **x** is reset within 3 time units after the setting of **y** and requires that **y** is reset within 5 time units after **x** is set. In the lower diagram the assumptions and requirements are exchanged.

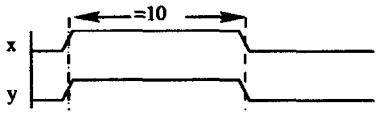

Fig. 7. Both x and y change their values from 1 to 0 after 10 time units

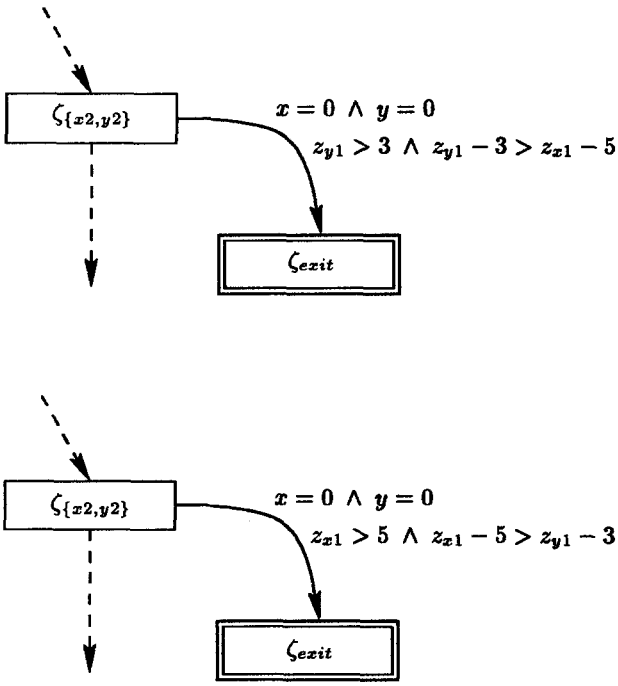

Fig. 8. Extract from the unwinding structures of the RTSTD in Fig. 6

the phase by any outgoing transition if the appropriate valuation of the ports and the imposed timing conditions hold. Hence, translation of the first state is straightforward. We may stay in phase $\zeta_{\{Req1,Ack1\}}$ as long as **Req** and **Ack** are 0, but must finally leave the phase (since $\zeta_{\{Req1,Ack1\}}$ is not an accepting state) by either taking the transition to ζ_{exit} (where the STD no longer constraints the ports, $TL(\zeta_{exit}) = true$) or by an unwinding step to the successor phase $\zeta_{\{Req2,Ack1\}}$.

$$
\begin{aligned}
&TL(\zeta_{\{Req1,Ack1\}}) = \\
&\quad (\mathsf{Req{=}0} \wedge \mathsf{Ack{=}0}\ \textbf{until}\ (\mathsf{Ack{=}1} \wedge z_{act} \leq 10) \\
&\qquad\qquad \vee\ \mathsf{Req{=}1} \wedge \mathsf{Ack{=}0} \\
&\qquad\qquad\quad \wedge\ z_{act} \leq 10 \wedge z_{Req1}.TL(\zeta_{\{Req2,Ack1\}}))
\end{aligned}
$$

If a state is accepting we use the **unless** operator instead of **until** since we may stay in the corresponding phase forever. Thus the translation of the remaining phases is as follows:

$$
\begin{aligned}
&TL(\zeta_{\{Req2,Ack1\}}) = \\
&\quad (\mathsf{Req{=}1} \wedge \mathsf{Ack{=}0}\ \textbf{unless}\ (\mathsf{Req{=}1} \wedge \mathsf{Ack{=}1} \wedge z_{Req1} < 2) \\
&\qquad\qquad \vee\ (z_{Req1} > 4) \\
&\qquad\qquad \vee\ (\mathsf{Req{=}1} \wedge \mathsf{Ack{=}1} \\
&\qquad\qquad\quad \wedge\ 2 \leq z_{Req1} \leq 4 \wedge z_{Ack1}.TL(\zeta_{\{Req2,Ack2\}})) \\
&TL(\zeta_{\{Req2,Ack2\}}) = \\
&\quad (\mathsf{Req{=}1} \wedge \mathsf{Ack{=}1}\ \textbf{until}\ (\mathsf{Ack{=}0} \wedge z_{Ack1} \leq 5) \\
&\qquad\qquad \vee\ (\mathsf{Req{=}0} \wedge \mathsf{Ack{=}1} \wedge \\
&\qquad\qquad\quad \wedge\ 1 \leq z_{Ack1} \leq 5 \wedge z_{Req2}.TL(\zeta_{\{\top_{Req},Ack2\}})) \\
&TL(\zeta_{\{\top_{Req},Ack2\}}) = \\
&\quad (\mathsf{Ack{=}1}\ \textbf{unless}\ (\mathsf{Ack{=}0} \wedge z_{Req2} < 2) \\
&\qquad\qquad \vee\ (z_{Req2} > 4) \\
&\qquad\qquad\quad \vee\ (Ack = 0 \wedge 2 \leq z_{Req2} \leq 4))
\end{aligned}
$$

Finally, we must add the activation mode to the formula. It resets the activation clock z_{act} and forces activation of the diagram

- either — if the activation mode is "Invariant" — whenever the leftmost conditions of the waveforms are matched by the actual values at the ports:

  $$\Box(\mathsf{Req{=}0} \wedge \mathsf{Ack{=}0} \Rightarrow z_{act}.TL(\zeta_{\{Req1,Ack1\}}))$$

  Observe that there is no straightforward way to express this invariance closure ($\Box$) on the level of automata as there may be several invocations of the same timing diagrams activated if during unwinding the activation condition is met again.

– or, if the activation mode is "Initial" — at system startup. In this case the activation of our example would have been

$$\mathsf{Req{=}0} \wedge \mathsf{Ack{=}0} \wedge z_{act}.TL(\zeta_{\{Req1,Ack1\}}) \ .$$

6 Discussion

In this paper we have extended the graphical specification language STD to allow specification of real time properties and given an introduction to the formal semantics of the extended language. Since the semantics of "pure" STD is not altered the constraints used in STD can be treated as special cases of RTSTD constraints which can be handled more efficient. Thus we only need to use real time requirements where really necessary and stick to qualitative timing wherever possible which helps decreasing complexity of the proofs.

A major obstacle of the semantics presented in this paper is the complexity of TPTL model checking. There are two answers to tackle this problem: on the one hand many subformulas produced during translation can be shared between different pathes through the timing diagram which reduces the proof amount drastically. Furthermore, the number of clocks needed in the formula may be limited by static analysis of the diagram. For example, in the diagram presented a single clock would suffice in the translation.

On the other hand, when model checking synchronous (clocked) systems we will use PTL without clocks to give semantics to RTSTD. In this case, we identify each time unit in the specification with a step of the system and use PTL's nexttime operator $\bigcirc$ to specify time intervals.

Thus, future research will focus on extending model checking techniques based on RTSTD and investigating subclasses of RTSTD with respect to efficient proof methods.

References

1. R. Alur and D. Dill. The theory of timed automata. In J.W. de Bakker, T. Henzinger, and W.P. de Roever, editors, *Real Time: Theorie in Practice, REX Workshop 1991*, LNCS 600, pages 45–73. Springer-Verlag, 1992.
2. R. Alur and T. Henzinger. A really temporal logic. *Journal of the ACM*, 41:181–204, 1994.
3. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL – a tool suite for automatic verification of real-time systems. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, 1995.
4. G. Boriello. Formalized timing diagrams. In *Proceedings, The European Conference on Design Automation*, pages 372–377, Brussels, Belgium, 1992.
5. J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Proceedings, 27th ACM/IEEE Design Automation Conference*, pages 46–51, June 1990.
6. W. Damm, B. Josko, and R. Schlör. Specification and verification of VHDL-based system-level hardware designs. In E. Börger, editor, *Specification and Validation Methods*, pages 331–410. Oxford University Press, 1995.

Cadence Design Systems, Inc. - Ex. 1007, Page 00012

7. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III, Verification and Control*, LNCS 1066. Springer-Verlag, 1996.
8. C. Dietz. Graphical formalization of real-time requirements. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'96)*, LNCS 1135, pages 366–384. Springer-Verlag, 1996.
9. K. Feyerabend. Real time symbolic timing diagrams. Technical report, Department of Computer Science, Oldenburg University, September 1996. http://ca.informatik.uni-oldenburg.de.
10. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16:403–414, 1990.
11. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
12. IEEE Standard VHDL Reference Manual IEEE Std 1076-1987, 1987.
13. P.K. Khordoc, M. Dufresne, and E. Czerny. A stimulus/response system based on hierarchical timing diagrams. Technical Report 770, Universite de Montreal, 1991.
14. C. Delgado Kloos and W. Damm, editors. *Formal Methods for Hardware Verification*. Springer-Verlag, 1996.
15. L. E. Moser, P. M. Melliar-Smith, Y.S. Ramakrishna, G. Kutty, and L. K. Dillon. The real-time graphical interval logic toolset. In *Proceedings of the 8th Int. Conference on Computer Aided Verification CAV'96*, LNCS 1102. Springer-Verlag, 1996.
16. Y.S. Ramakrishna, P.M. Melliar-Smith, L.E. Moser, L.K. Dillon, and G. Kutty. Really visual temporal reasoning. In *Proceedings of 14th Real-Time Systems Symposium*, pages 262–273, December 1993.