Verification of Timing Properties of VHDL

Costas Courcoubetis[1]*, Werner Damm[2], Bernhard Josko[2]

[1] Department of Computer Science, University of Crete, and
Institute of Computer Science, FORTH, Greek
[2] Department of Computer Science, University of Oldenburg, Germany

Abstract. This paper shows how timing properties of VHDL processes can be verified using timed transition systems. The timing model being adopted is the timed automaton model used in the timing extension of Kurshan's COSPAN system. It demonstrates how a VHDL process can be translated into a timed automaton by describing the construction of the corresponding timed process that handles the scheduled signal assignments of the VHDL specification. Verification is performed in the case in which the complement of the timing properties to be verified are provided in terms of a timed automaton. Interestingly enough, this is the case for a large class of hardware properties expressed in terms of timing diagrams.

1 Introduction

VHDL is a standardized [11] and widely accepted hardware description language. It is a powerful language that allows to specify hardware at gate level as well as at system level. One can express in VHDL various real-time properties such as timeouts and delays after which certain signals will be set to a particular value. Hence verifying properties of VHDL specifications becomes a highly non-trivial task. The goal of this paper is to provide a method for translating a restricted subset of VHDL into timed automata, and hence use the already existing verification machinery for doing verification of VHDL programs.

In VHDL a system is described in terms of entities which can either be structural descriptions or behavioural descriptions. A behavioural description is defined by a set of processes each given as a sequential program. Timing aspects such as output delays or edge triggered events can be expressed in VHDL using signal assignments and wait statements. For instance the signal assignment `out <= in after 5ns;` expresses that the value of the inport *in* is written to the outport *out* with a delay of 5 ns. Each such pending assignment is added to an event queue (a transaction is scheduled) and the process instantaneously proceeds to the next statement. After the delay time has elapsed, the value is written on the corresponding port. Wait statements are used for example to model the change of a signal triggered by the positive edge of a clock: `wait on clk until clk = '1'; out <= in;`. The semantics of VHDL is based on simulation and time only advances when executing a wait statement.

* Partially supported by the BRA ESPRIT project REACT

The only approach existing so far for validating a VHDL design is to add assertions which are checked during a simulation run. Since the complexity of hardware systems is continuously growing and consequently, the simulation of the system for every possible input is not feasible, such an approach becomes obsolete and can not be used for guaranteeing correctness. Especially for control devices in safety critical applications, methods for validating the correctness of hardware systems are needed. Hence formal verification techniques become very important in hardware design. In [18] and [5] it is shown how a gate level description written in VHDL can be checked against an abstract behavioural description, also written in VHDL. The PREVAIL system of [5] is based on the Boyer-Moore theorem prover, whereas [18] uses the higher order logic HOL. In [8] it is shown how a finite state model can be generated out of a VHDL process description. This allows the application of standard model checking procedures to check a VHDL design against a temporal logic specification. However, in that approach every clock tick is represented explicitly by a transition, which may lead to a state explosion. In this paper we translate VHDL processes into timed automata of the type used in RT-COSPAN [6], which are based on the original timed model of Dill [7]. The main advantage is that the resulting finite-state system used by the verification algorithm remains in general reasonably small. The reason is that the timing information of the processes is collapsed into a small finite number of states, each representing an equivalence class of time assignments for the pending events of the system. Unfortunately, there are VHDL descriptions which can not be translated into finite timed automata. The reason is that one can construct VHDL processes for which the size of the list of the pending events can grow arbitrarily large. A sufficient condition for avoiding this is to consider a restricted subset of VHDL in which the time expressions in all statements of the program are constants. We believe that this is not a serious restriction for any practical purpose.

Our approach can be used for constructing a VHDL compiler into RT-COSPAN. The final requirement for performing verification is the provision of the properties to be verified in a certain format: We need a timed automaton accepting precisely the undesirable system behaviours (task complement). Interestingly enough, this is not such an impossible task in the case of hardware components. Fortunately, in [17] it is shown that an interesting class of properties related to hardware verification can be specified in terms of a restricted class of *timing diagrams*. The important property of specifications in this class is that they can be directly complemented, i.e., one can automatically derive the task complement process. This greatly enhances the applicability of our method.

The paper is organized as follows. In Sect. 2 we briefly discuss the timing aspects of VHDL and describe the timing model used in this paper to verify VHDL properties. Section 3 describes the translation of VHDL programs into timed automata. This translation is illustrated by an example given in Sect. 4. Finally, we conclude in Sect. 5 with some directions for further research.

2 VHDL and the Timing Model

In this section we provide a brief and informal introduction to the basic VHDL concepts which involve time, and we discuss the timing model and the verification methodology used in the paper.

A VHDL architecture is given by an *entity declaration* describing the interface and an architecture declaration which can be either a *behavioural body* or a *structural body*. A structural body describes how subarchitectures are composed together and a behavioural body is given in terms of a programming language notation. There are essentially two non-standard statement constructs: *signal assignment* and *wait statement*. There are two types of signal assignment depending whether *inertial delay* or *transport delay* is used: `s <= '1' after 10 ns;` resp. `s <= transport '1' after 10 ns;`. If the keyword `transport` is missing inertial delay is assumed. After the execution of a signal assignment a *transaction* is entered into the *projected (output) waveform* associated with the signal, i.e., all future values for the signal together with their posted time of application are collected in the projected waveform. If there are already old entries from previous assignments on the projected waveform, then some of the old transactions may be cancelled. The way this is done depends whether transport or inertial delay is used. In both cases all entries scheduled at a later time than the new transaction are cancelled. Inertial delay is used to describe that an input pulse is not recognized if it is shorter than the given output delay. Hence transactions scheduled before the new one will be erased as well if they have a different value. The management of these projected waveforms including the cancellation of transactions and updating of signals will be modelled by a special process in the translation.

The second non-standard statement is the wait statement. A wait statement can contain three types of clauses: a *sensitivity clause*, a *condition clause*, and a *timeout clause*: `wait on s1, s2 until s1 = s2 for 10 ns;`

When reaching a wait statement the process is suspended until the wait clause evaluates to true. The meaning of the statement is that if the value of some signal in the sensitivity clause (`on s1, s2`) changes, the condition (`s1 = s2`) is evaluated; if it is true (or the condition is omitted) the process proceeds, otherwise the process resuspends. The timeout clause gives a maximum time for which the process will wait.

The semantics of a VHDL program is based on simulation. All processes are synchronized with the execution of wait statements. The simulation starts with an initial phase and then repeats a two-stage simulation cycle. During the initial phase, all signals are initialized, the simulation time is set to zero, and every behavioural description is executed until its first wait statement. In the first stage of a simulation cycle, the simulation time is advanced to the earliest time at which a transaction has been scheduled and all transactions scheduled for that time are executed. In the second stage all modules evaluate their wait clauses and proceed to the next wait statement or resuspend until the next cycle, if their wait condition is not satisfied. This global synchronization of all processes will be reflected by introducing a scheduler process in the translation.

In order to use automatic verification tools such as RT-COSPAN we need that a VHDL description can be modelled by a finite timed transition system. This restricts us to consider only a subset of VHDL. Clearly we must restrict data types to be finite; in this work we regard variables and signals to be only of type Boolean, Bit. Another important restriction comes from the following observation. Since in general time expressions need not be constants, we can specify such expressions which cause larger and larger delays as the system runs. This implies that the number of pending events can grow as well without bounds. Our remedy for avoiding the arbitrary growth of the projected waveforms is to consider only systems with time expressions which are constant[3]. With this restriction the length of the projected waveforms is bounded and we can compute an upper bound for the lengths of the corresponding event lists. This is crucial since in our translation we need to assign a clock with each pending event, and hence we need to know in advance the maximum number of clocks we will need.

A simple way to compute an upper bound is the following. Consider the *complete* system and let $t_1, \ldots, t_n$ be all time constants occurring in signal assignments or wait statements. Let δ be the maximal time constant such that every t_i can be written as $k_i \cdot \delta$ for some integer k_i. If t_n is the largest such time constant, then k_n is the maximal number of pending events in each projected waveform. All pending events are scheduled for a time instant $k \cdot \delta$ for some integer $k \leq k_n$ (note that the system changes state only at time instants which are multiples of δ, which is deduced from the complete VHDL system).

By a more detailed analysis of dependencies between events one can obtain better approximations for the maximum number of pending events. First observe that the maximum number of pending events of one process is given by k_r where t_r is the maximum time constant within the considered process. Furthermore, if for example an event is only scheduled in every second simulation cycle (there are at least two wait statements executed between two signal assignment statements) the maximum number for that event can be halved.

There exist several approaches for verifying timing properties of real-time systems [7, 3, 4, 1, 14, 10, 6]. Our approach is based on the framework used in the system RT-COSPAN described in [6], which combines timed transition systems such as the timed automata by Dill [7] and the timed graphs in [1], with a model of coupled finite-state machines (the selection/resolution model) and its underlying verification tool COSPAN [12, 16, 9]. We have chosen this method because of the expressive power of the underlying timing model (timed ω-automata), the nice features of the coupled finite-state machine model underlying COSPAN, and the availability and efficiency of the software for doing verification.

A timed automaton is an automaton with timing constraints on the transitions. The version of timed automata used in RT-COSPAN is the one introduced by Dill [7], where the timing information is added to the automaton in terms of a set of clocks whose values are decreasing with time (they are counting down like alarm clocks). These clocks are being set and expire while the automaton transitions. To describe a timing constraint on the time elapsed between two

[3] It is only necessary that the time expressions are constant at compile time.

transitions, a clock is set with the first transition and its expiration coincides with the second transition. Also when a clock is set (starts counting), its initial value is chosen nondeterministically from an interval whose end points depend on the particular clock.

In the selection/resolution (S/R) model, a system is decomposed into a set of simple components operating concurrently in a synchronous fashion, which are called *processes*. A process is represented by an edge-labelled graph. The vertices of the graph are the states of the process. Each process has a name and a selection variable (possibly a vector of variables), whose name is denoted by the process name followed by #. Each state is associated with a set of possible values for the selection variable of the process, which can be chosen by the process when it is at that state. Each edge of the process graph describes a transition which is possible in one computation step: it is labelled with a predicate on the selection variables of all the processes in the system.

The computations are performed in two phases: first, in the *selection* phase each process chooses a value for its selection variable (from the set of possible values in its current state). Then, in the *resolution* phase each process chooses a transition among the ones whose conditions do not evaluate to false (these are evaluated on the values of the selection variables updated during the selection phase). This synchronous product of the processes corresponds to the global system, and the tool COSPAN can generate its corresponding reachable subgraph. The semantics of the model are trace semantics. A trace is an infinite sequence of global state selection pairs. To include fairness constraints, the notions of *cysets* ("cycling sets") and *recur edges* ("recurring edges") may be used: the system (a system component) should not eventually stay forever in some cyset and it should not infinitely often perform transitions from the set of recur edges. The set of computations of a process is the set of its fair traces.

In [6] it is shown how timing constraints can be incorporated into S/R processes in order to transform them into timed automata such as the ones in [7]. The specification of a timed process is given by two parts: the *time-independent* and the *time-dependent* part. The time-independent part corresponds to an "over-specification" of the system when time constraints are not taken into account and is given in terms of a number of S/R processes corresponding to the components of the system. The time-dependent part captures the real-time constraints on the behaviour of the untimed part, and is given in terms of a set of clocks and their corresponding set and expire conditions (also called a "clock-system"). The set and expiration events of these clocks are associated with specific transitions of the untimed system. The semantics of a timed process are *timed* traces. A timed trace is an infinite sequence of triples (*global state, global selection, timestamp*), where *timestamp* is a real number denoting the value of time at which the system entered the above global state. The set of timed computations of a process is the set of its timed traces which are fair and consistent with the semantics of the clock-system provided in the time-dependent part of the process. Projecting the time information out of the timed language produces the *untimed* language of a timed process. This corresponds to the set of timing-consistent computations

and is a sublanguage of the set of computations of the time-independent part. An important result of this model is that this language is the intersection of the language of the untimed part and the language of a process that can be derived from the time-dependent part. We refer to this process as the *time monitor*.

The existence of the time monitor process is key for doing verification. Assume that the timing properties that need to be verified are given in terms of their complement (the "task complement"), i.e., we are given a timed process whose timed language corresponds precisely to the forbidden computations of the system. Then the system is correct if and only if it does not contain an undesired computation, i.e., the timed language of the product process (system process and complement of desired properties process) is empty[4]. Now, since the timed language of a timed process is empty if and and only if its untimed language is empty, the above emptiness problem is reduced from the timed domain to the untimed domain: we need to check if the product of the time monitor process capturing the timing constraints with the untimed part process is empty. COSPAN does this by automatically generating (on-the-fly) the monitor process and checking for emptiness of the product with the untimed part.

In general, the construction of the task complement can be very difficult or even impossible, see [3]. Fortunately, in [17] it is shown that an interesting class of properties related to hardware verification can be specified in terms of a restricted class of *timing diagrams*. The important property of specifications in this class is that they can be directly complemented, i.e., one can automatically derive the task complement process. This makes our approach very promising in the area of hardware verification.

3 Translating VHDL to Timed Automata

In this section we provide the basic ideas for the translation of VHDL specifications to timed automata. To simplify the exposition we found that it is easier to use a different notation for timed automata which can be directly translated to the one mentioned in the previous section. This corresponds to a restricted version of timed graphs [1], where each clock x is counting up and the only predicate on this clock labelling the transitions of the system is the comparison with a certain constant d_x whose value depends on the particular clock.

A VHDL module will be modelled by two processes. The first, which does not depend on real time, is called the *control graph* reflecting the control flow of the VHDL program, and is an untimed S/R process. The second, which captures the timing information, is called the *monitor process* or *event graph*. This process manages the projected waveforms by scheduling new events and setting the signals at their scheduled time with the appropriate value. This process is a timed S/R process and uses clocks to model the delays between different transitions. There is also a global scheduler process reflecting the two-stage simulation cycle.

[4] The product of two timed processes is a timed process whose untimed part is the usual product of the untimed parts and the timed part the union of the two corresponding clock-systems.

The construction of the control graph is straightforward as it reflects the control flow within the VHDL program. The transitions associated to signal assignments and wait statements will be labelled with additional information which will guarantee the correct synchronization. The execution of a wait statement has to be synchronized with the scheduler, which allows such a synchronization step if every VHDL process is ready to execute a wait statement. Hence every state before a wait statement displays the selection *at_wait*[5], and the edge reflecting the wait transition is labelled with the conjunction of the condition that the scheduler allows the step to occur (this is modelled by the scheduler selecting the value *synch*) and the conditional clause of the wait statement. The negation of the above label must appear on the self-loop transition out of that state.

The sensitivity clause of a wait statement (`wait on s1, s2;`) will be translated into a condition $changed(s1) \wedge changed(s2)$. Hence, for every signal we need a process which keeps track whether its value has changed since the last simulation cycle or not (the above condition should be expressed in terms of the selections of these processes). Every assignment `s <= transport v after t ns;` (we refer to this as $event(s, v, t)$) is modelled by the selection (s, v, t) which will be read by the event graph and will cause the updating of the projected waveform.

We proceed now with the construction of the second process, the event graph. This process is responsible for keeping track of time in order to realize the timely execution of the set signal events. We have already mentioned that in a VHDL system time elapses only during a synchronization step, i.e., when every VHDL process of the system executes its current wait statement. All other statements do not consume time. These timing constraints are incorporated in the global scheduler process as we will show in the next section.

The event graph uses several clocks, one for each pending signal assignment. Whenever a signal assignment is executed a new clock is used and is being reset. When the time provided in the signal assignment has elapsed, the corresponding signal is updated. Generating new clocks during the execution of the VHDL program can lead to systems with infinitely many clocks. Note also that we cannot associate a fixed clock $x^{s,v,t}$ with a specific statement `s <= transport v after t ns;` since consecutive executions of the statement may generate several entries in the projected waveform for the signal s. In order to use finitely many clocks we have to keep track of the use of the clocks and a clock which has expired should be released. This is done by associating to every signal s, every value v, and every time constant t a set of clocks $X^{s,v,t} = \{x_0^{s,v,t}, \ldots, x_{k-1}^{s,v,t}\}$, where k is the maximal number of pending events of the particular type. (See comments in Sect. 2 for how to compute k.)

To every set $X^{s,v,t}$ we associate three variables: a lower and an upper bound, $lb_{s,v,t}$ and $ub_{s,v,t}$, describing the interval of the clocks in use (active clocks) and a boolean variable, $e_{s,v,t}$, indicating if a clock of $X^{s,v,t}$ is used at all, i.e. $e_{s,v,t}$ is used to indicate whether an interval with $lb_{s,v,t} = ub_{s,v,t}$ is the singleton set $\{x_{lb_{s,v,t}}^{s,v,t}\}$ or the empty set.

[5] The set of possible selections of a process is enclosed between curly brackets next to the corresponding state, see Fig. 3.

The algorithm for allocating clocks is the following. Whenever a signal assignment $event(s, v, t)$ is executed, the next free clock of the corresponding set $X^{s,v,t}$ is used and the upper bound $ub_{s,v,t}$ is increased by one (we count modulo k). If a clock expires the index of that clock must be equal to the lower bound of the active clocks, and hence we will increase $lb_{s,v,t}$ and perform the signal update. If $lb_{s,v,t} = ub_{s,v,t}$ we do not increase $lb_{s,v,t}$, but we set $e_{s,v,t}$ to 0 instead (empty interval). If now a new clock is used we only increase the value of $e_{s,v,t}$ to 1. In the full paper we will prove that there are no conflicts in the allocation of clocks. Whenever a new clock is required it is guaranteed that there is a free clock of the appropriate type. Furthermore, active clock indices will always form an interval.

Another important concept necessary to implement the VHDL semantics is the possibility for *cancelling* events from a projected waveform and to release the corresponding clocks. An event of a projected waveform should be cancelled if a new event with an earlier time of occurrence is scheduled. When an event is cancelled all other pending events following this event must be cancelled as well.

From the above discussion it follows that the event graph process must encode for every signal s a projected waveform of the form $waveform(s) = [(v_1, x_1), \ldots (v_n, x_n)]$, where the clock x_k counts the time for the application of the value v_k to the signal s; for example if $x_k \in X^{s,u,t}$ then when x_k will reach the value t, s will be set to v_k. The clock cancellation works as follows. If an event $event(s, v, t)$ occurs, then all clocks in the projected waveform for the signal s are released for which $t_k - x_k \geq t$ holds (t_k is the expiration time of the clock x_k when it was set). It is easy to show that if a clock $x_k^{s,v,t}$ is released, then also all clocks $x_r^{s,v,t}$ with $k \leq r \leq ub_{s,v,t}$ must be released as well.

In general a VHDL system consists of a large number of components. In this case the control graph of the system is given by the product of the control graphs of its components. Similarly, the event graph of the system will be the product of the timed processes corresponding to the event graphs of the components.

4 An Example

We illustrate our construction by a simplified version of a bus protocol. Figure 1 shows the structure of an architecture consisting of a *Master* component and a *Slave* component. The exchange of data between these components has to follow the request/acknowledge protocol as shown in the timing diagram of Fig. 2. For simplicity we have omitted the data line and we have considered only the control lines. The VHDL description of the Slave is given by:

```
entity Slave is
    port  ( Req : in Bit; Ack : out Bit := '0');
end;
architecture behavioural_body of Slave is  begin
    process begin
        wait until Req = '1';   Ack <= transport '1' after 2 ns;
        wait until Req = '0';   Ack <= transport '0' after 2 ns;
    end process;
end;
```

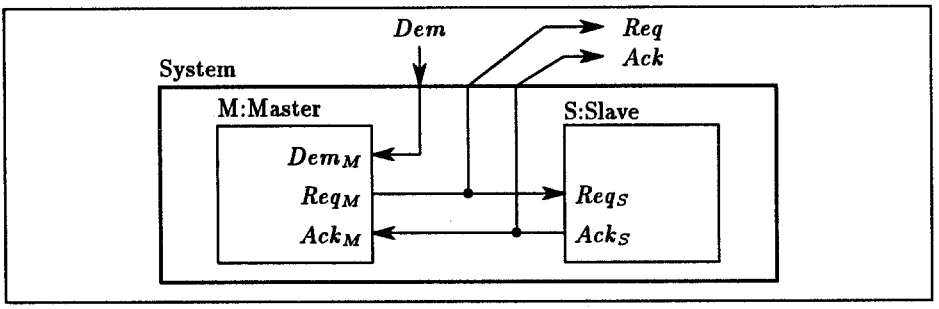

Fig. 1. The master/slave system

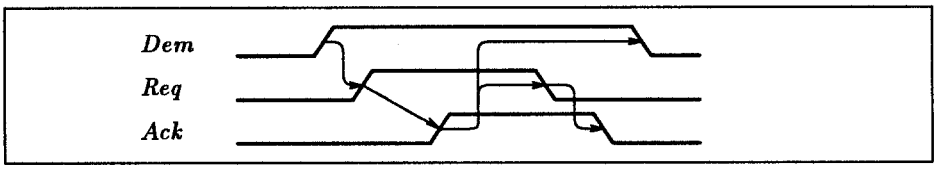

Fig. 2. Behaviour specification, timing diagram

In the following we give the construction of the timed process graphs for the Slave module. For simplicity of the notation our graphs are more general than the ones required by the S/R model, and we allow transition predicates to depend on the value of signal variables and clock variables besides the traditional selection variables. We also associate reset actions with transitions. Translating this into the original model used in COSPAN and RT-COSPAN is a trivial task.

Figure 3 shows the control graph and the global synchronization monitor. The synchronization monitor guarantees that only during a wait statement time advances. This is modelled by using a single clock x which is reset every time a synchronized wait statement is executed. Then an arbitrary number of "non-wait" transitions can take place in zero time since this clock must remain at zero. The next wait transition can occur only if non-zero time has elapsed, i.e., $x > 0$. Note that the structure of the scheduler is such that the only accepted computations are the ones which always stay in state 0. These computations have the property that all processes (in our case just the slave process) execute their wait statements concurrently with the scheduler selection *synch*.

The signal assignment statements generate events which are processed in the event graph shown in Fig. 4. The event graph handles the projected waveforms for every signal of the process. If an $event(Ack, v, t)$ occurs - selection generated by the control graph - this event is added to the projected waveform and a new clock is initialized. If a clock is expired the corresponding signal is updated. In our example we have two different events: $event(Ack, 0, 2ns)$ and $event(Ack, 1, 2ns)$. Therefore we need two sets of clocks: $X = X^{Ack,0,2}$ and $Y = X^{Ack,1,2}$.

Assuming that 1 ns is the minimal delay in our global system including the Master component, we have at most two pending events of each type for the signal *Ack*. (The maximal delay given in module Slave is 2 ns). Now observe that between two signal assignments of the form `Ack <= transport '0' after 2ns;`

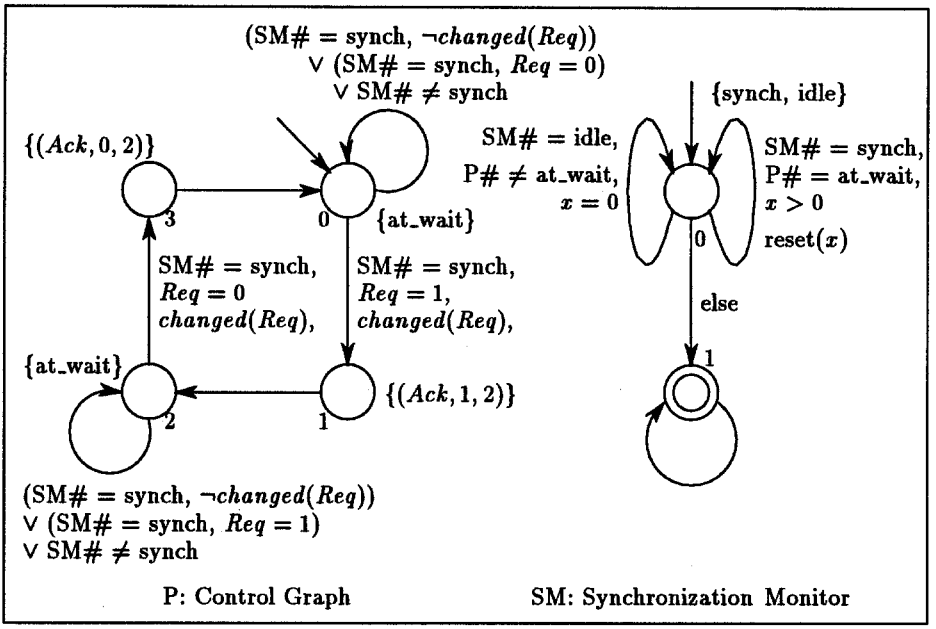

Fig. 3. Control graph of slave and the main synchronization loop

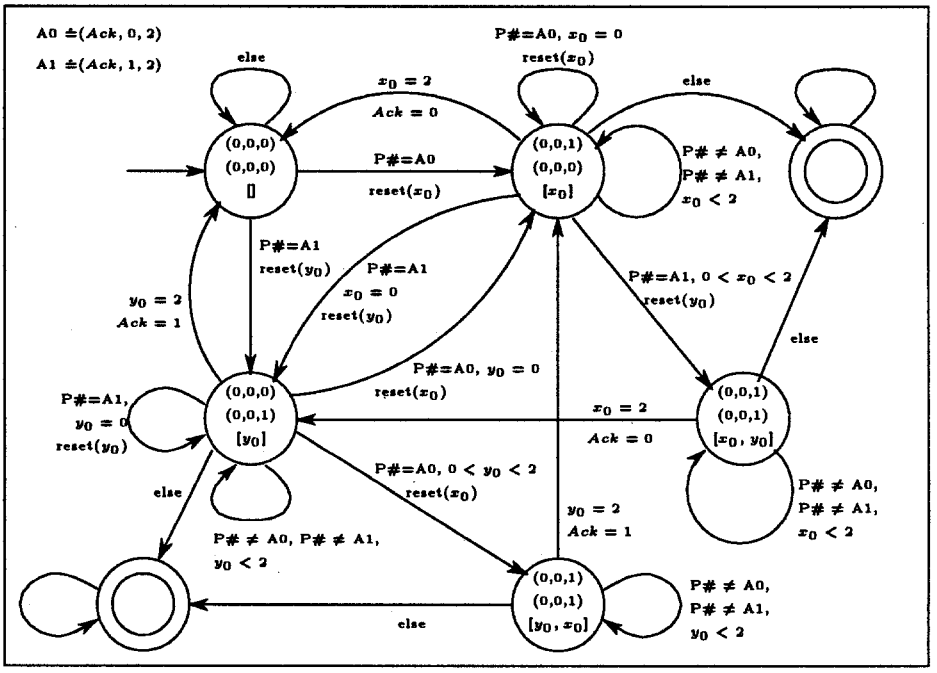

Fig. 4. Event graph of component slave

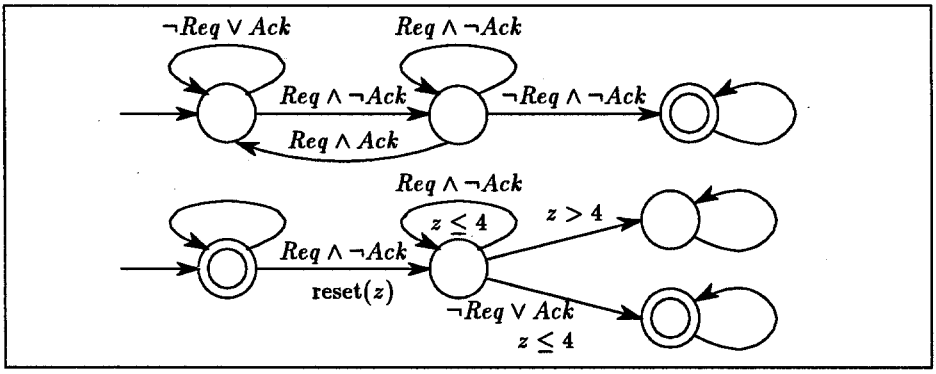

Fig. 5. The negated specification

there are two wait statements. The same is true for signal assignments assigning `'1'` to the signal *Ack*. Therefore the maximum number of pending events of each type is one and hence we need only one clock for each set. Hence $X = \{x_0\}$ and $Y = \{y_0\}$. The states of the event graph encode the values of *lb*, *ub*, and *e* for the sets X and Y, and the pending events for the outport *Ack*. As X is a singleton set the lower and upper bound indices do not change, only the value of *e* changes depending whether the clock x_0 is active or not. When a new event is scheduled by the Slave process, the event graph proceeds accordingly. It resets a new clock for this event and performs the transition which assigns the value of the event when the clock expires.

One requirement on the component *Slave* is that is has to send an acknowledge within e.g. 4 ns after the rising edge of the request signal *Req*, provided the environment guarantees that the signal *Req* remains high unless the acknowledgement occurs. The complement of this specification is the following: The environment guarantees the given assumption, but there is some time instant where *Req* is high and *Ack* does not occur within 4 ns. The automata for this task complement is shown in Fig. 5. The first automaton describes the assumption on the environment and the second one describes that there is a time interval of 4 ns with *Req* set to high but with *Ack* being low during this interval. The start state is marked as a cyset to force the process to step forwards. Verification can now be performed by checking for emptiness of the timed automaton corresponding to the product of the task complement and the VHDL program (see [6]).

5 Conclusion

In this paper we have demonstrated how timing properties of VHDL processes can be verified using RT-COSPAN. This approach can easily be extended to a system of several processes. Doing this we have to add a process handling the wiring of signals and the resolution functions for multiple driven signals. An important issue which must be pursued further is the case in which the

construction of the task complement is not feasible. In this case we should investigate how to apply the model checking algorithm as described in [1] and [2]. An important issue is the construction of small region graphs by exploiting the particular structure of VHDL programs. The final part that needs to be performed in order to validate the practical importance of our approach is to implement a compiler from VHDL to the RT-COSPAN language. Although in this paper we have solved the theoretical issues related to such a translation, this remains a non trivial task since efficiency is extremely important.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Symp. on Logic in Computer Science*, pp. 414-425, 1990.
2. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *CONCUR 92: Theories of Concurrency*, LNCS 630, pp. 340-354, 1992.
3. R. Alur and D. Dill. Automata for modelling real-time systems. In *ICALP 90*, LNCS 443, pages 322–225, 1990.
4. R. Alur. *Techniques for Automatic Verification of Real-Time Systems.* Technical report stan-cs-91-1378, Stanford University, 1991.
5. D. Borrione, L. Pierre, and A. Salem. PREVAIL: A proof environment for VHDL descriptions. In *Proc. Advanced Research Workshop on Correct Hardware Design Methodologies*, pp. 145-168, 1991.
6. C. Courcoubetis, D. Dill, M. Chatzaki, and P. Tzounakis. Verification with real-time COSPAN. In *Proceedings CAV 92*, 1992.
7. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, LNCS 407, 1989.
8. W. Damm, B. Josko, and R. Schlör. Linking VHDL with formal verification tools: How to generate finite state models out of VHDL designs. Technical report, 1992.
9. Z. Har'El and R. Kurshan. Automatic verification of coordinating systems. In *Proc. Workshop on Automatic Verification Methods for Finite-State Systems*, 1989.
10. T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proceedings of the 18th ACM Symposium on Principles of Programming Languages*, pp. 353-366, 1991.
11. IEEE standard 1076-1987, VHDL language reference manual, 1987.
12. R. Kurshan and B. Gopinath. The selection/resolution model for coordinating concurrent processes. Technical report, AT&T Bell Laboratories, 1980.
13. R. Kurshan. Analysis of discrete event coordination. LNCS 480, 1990.
14. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In [15].
15. J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors. *Real-Time: Theory in Practice. REX Workshop 1991.* LNCS 600, 1992.
16. K.K. Sabnani, S. Aggarwal, and R.P. Kurshan. A calculus for protocol specification and validation. In *Protocol Specification, Testing and Verification, III*, 1983.
17. R. Schlör and W. Damm. Specification and verification of system-level hardware designs using timing diagrams. EDAC 93.
18. J. van Tassel and D. Hemmendinger. Toward formal verification of VHDL specifications. In L. Claesen, editor, *Proc. Workshop on Applied Formal Methods For Correct VLSI Design*, pp. 261-270, 1989.