Submission to the Proceedings of DAC ‘99

Title: An Approach for Extracting RT Timing Information

Authors: Cordula Hansen (presenter) Francisco Nascimento, Wolfgang Rosenstiel

Address: FZI Forschungszentrum Informatik at the University of Karlsruhe
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
Phone: ++49-721-9654-454
Fax.: ++49-721-9654-409
E-mail: hansen@fzi.de

University of Tübingen Lehrstuhl für Technische Informatik
Am Sand 13
72076 Tübingen, Germany
Phone: ++49-7071-29-75482
Fax.: ++49-7071-610399
E-mail: {rosen,moreira}@peanuts.informatik.uni-tuebingen.de

DAC topic number: M1.1, M1.2, M1.7

All appropriate organizational approvals for the publication of this paper have been obtained. If accepted the author(s) will prepare the final manuscript in time for inclusion in the Conference Proceedings and will present the paper at the Conference.
An Approach for Extracting RT Timing Information

Abstract

This paper presents a new approach for extracting timing information defined in a register transfer level (RTL) simulation vector set and reusing them in the behavioral specification. Using a VHDL RTL test-bench and a VHDL behavioral specification as entry, the timing constraints are extracted and as well as the specification transformed in a Partial Order based Model (POM). The POM expressing the timing constraints (POM_{tc}) is then mapped onto the specification POM (POM_{spec}). The result contains the behavioral specification and the RTL timing constraints and is retransformed in a corresponding VHDL specification. This specification can then be checked using the RTL simulation vectors.
An Approach For Extracting RTL Timing Information

Abstract

This paper presents a new approach for extracting timing constraints defined in a register transfer level (RTL) simulation vector set and reusing them in the behavioral specification. Using a VHDL RTL testbench and a VHDL behavioral specification as entry, the timing constraints are extracted and as well as the specification transformed in a Partial Order based Model (POM). The POM expressing the timing constraints (POM\textsubscript{TC}) is then mapped onto the specification POM (POM\textsubscript{spec}). The result contains the behavioral specification and the RTL timing constraints and is retransformed in a corresponding VHDL specification. This specification can then be checked using the RTL simulation vectors.

1 Introduction

Up to now, most ASIC or FPGA designers have developed synthesizable register transfer level (RTL) VHDL models and the corresponding simulation vector sets also implemented in VHDL. In this domain, several approaches have been published to facilitate the development of RTL specifications and RTL stimuli [2][?]. Currently, industrial VHDL specifications are increasingly implemented as behavioral descriptions on the algorithmic level. Also new versions of already existing RT specifications are now re-implemented as algorithmic VHDL descriptions. Thus, a new question arise: is it possible to reuse the information defined in the RTL simulation vector set for a specification now on an algorithmic level?

Therefore, this paper addresses the problem of extracting timing constraints from the VHDL RTL simulation vectors for the algorithmic specification. Main feature of the presented approach is the integration of the extracted timing constraints into the algorithmic specification using a Partial Order based Model. Further, already defined timing constraints in the specification can be validated by comparing them with the timing constraints contained in the RT simulation vector set. This allows an efficient reuse of already existing timing information and, therefore, a reduction of development time.

This paper is organized as follows: Section 2 describes the basic definitions and relations of the used Partial Order based Model. In Section 3, the design flow and the basic concepts of our approach are presented. Some examples, including experimental results, are presented in Section 4. Finally, this paper concludes with a summary in Section 5.

2 Partial Order Based Model

One model that can be extended to implement the desired characteristics is the Partial Order based Model (POM) [3][4]. This model is event based, and each event is defined as a unique instance of an action and can not occur more than one time in a computation. Further, the POM is a non-interleaving model, i.e. in the representation of the parallel execution of two events, no information about the order of the event execution is assumed.

2.1 Basic Definitions

The POM is based on the Chu space formalism [5]. A POM consists of a set $A$ of events, where each event represents the unique instance of an action, and a set $X$ of states representing the possible or permitted states. Further, a state is defined in terms of an occurrence relation $R(a,x)$ that is true when the event $a$ has occurred in the state $x$.

**Definition 1:** A POM is a Chu space $C$ given by the tuple $(A, X, R)$, where

- $A = \{a_0, a_1, ..., a_n\}$ is a set of events
- $X = \{x_0, x_1, ..., x_m\}$ is a set of states and
- $R : A \times X \rightarrow \{0, 1\}$ represents the occurrence relation, i.e. $R(a, x) = 1$ if the event $a$ has occurred in the state $x$ otherwise $R(a, x) = 0$.

Each state $x_i \in 2^A$ is defined in terms of $R$ as:

$$x_i = \{a | (a \in A) \land (R(a, x_i) = 1)\}$$
The occurrence of the events relation can be seen in Figure 2 (a). No order is imposed for the implementation of the symbolic representation. The independence definition makes it possible to handle the state space explosion problem, since a high degree of parallelism in the specification leads to more independence relations and, consequently, to smaller BDDs. The second point is, that, if necessary, two events can be turned sequentially using one logical operation. A simple conjunction of the logical formula of the independence relation with the logical formula of the precedence relation implements this serialization.

**Precedence Relation**

The precedence relation \((a < b)\) represents the occurrence of the event \(a\) followed by the occurrence of the event \(b\). The POM representation for this relation can be seen in Figure 2 (b). This relation is used to model the sequential execution of events.

\[
\begin{array}{c|c}
\text{a} & \text{b} \\
\hline
0 & 0 \\
1 & 1 \\
0 & 1 \\
1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{f}_\text{den} & \text{f}_\text{POM} \\
\hline
\text{a} & \text{a} \\
\text{b} & \text{b} \\
\text{c} & \text{c} \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{x}_0 & 0 \\
\text{x}_1 & 1 \\
\text{x}_2 & 1 \\
\text{x}_4 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
f_a = \text{a} \text{b} + \text{a} \text{b} + \text{a} \text{b} \\
f_d = \text{a} \rightarrow \text{b} + \text{b} \\
\end{array}
\]

**Conflict Relation**

The conflict relation \((a \# b)\) represents either the occurrence of \(a\) or the occurrence of \(b\). The correspondent POM and logical formula are shown in Figure 3 (a).

**Disjunctive Enable Relation**

The disjunctive enable relation permits two events to be represented, whose executions disjunctively enable a third event, i.e. \(\text{den}(a,b,c)\) means the execution of \(b\) OR \(c\) enables the occurrence of \(a\). This relation, together with the above explained conflict relation, is necessary to enable the events that follow an "if then else"-statement. Figure 3 (b) presents the POM and the corresponding logical formula.

\[
\begin{array}{c|c}
\text{a} & \text{b} \\
\hline
0 & 0 \\
1 & 1 \\
0 & 1 \\
1 & 1 \\
\text{x}_0 & \text{x}_1 \\
\text{x}_2 & \text{x}_4 \\
\text{x}_5 & \text{x}_6 \\
\text{f}_\text{den} & \text{f}_\text{a} \\
\text{f}_d & \text{f}_a \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{x}_0 & \text{x}_1 & \text{x}_2 & \text{x}_4 & \text{x}_5 & \text{x}_6 & \text{f}_\text{a} & \text{f}_\text{den} \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]
3 Extracting and Validating Timing Information

In this section, the design flow and the basic concepts are explained in more detail. First, the design flow is presented. Then, the requirements and boundary conditions are described. Next, an outline of the used POM semantics is given. Finally, the main topic of this paper, the extraction and validation concepts are presented.

3.1 Design Flow

The approach presented in this paper facilitates two different tasks:

1. the extraction of timing constraints from a RT simulation vector set (Figure 4 (a)). The timing information defined there can then be used to constraint the timing behavior of the algorithmic specification.

2. the validation of timing constraints defined in the algorithmic specification (Figure 4 (b)-(c)). The timing constraints extracted from the specification can be compared with the extracted timing constraint of the RT simulation vector set.

In the following, the design flow is described containing the CADDY-II synthesis system [6] [7] and the extraction of timing constraints as well as the generation of the POMs (Figure 5).

In the first step, the VHDL preprocessing is started. The main task during preprocessing is the generation of a flow graph description. This graph serves as an interchange format for the CADDY-II system. Moreover, this description is used to identify the data dependencies of the I/O signals necessary for the generation of the POM representing the algorithmic specification (POMSPEC). After the generation of the preprocessor results, the POMSPEC generation is started. This task is described in more detail in Section 3.2.2. In the third step, the high-level synthesis process is started. This process transforms the behavioral algorithmic specification into a structural RT description. This resulting RT design is usually represented by a VHDL description. The next step is the generation of the second POM (POMTC) representing the timing information contained in the RT simulation vector set. This task is also described in more detail in Section 3.2.3. The last step is the mapping of the different event semantics and the generation of a corresponding VHDL specification (Section 3.2.4).

3.2 Extraction of Timing Constraints by Generating Partial Order Based Models

When implementing the extraction of timing constraints from a RT simulation vector set, the following requirements have to be fulfilled:

- in general, timing information extracted from simulation vector sets for one simulation cycle are sufficient. However, an extraction from simulation vectors defined for several simulation cycles are also possible.
- at least, the simulation vector set has partially to implement a synchron timing behavior. Timing constraints can only be extracted from simulation vectors that are not implementing a complete handshake protocol.
• the timing behavior in the VHDL simulation vector set has to be defined using one of the following WAIT constructs: \( \text{WAIT FOR } \langle \text{time} \rangle; \) or \( \text{WAIT UNTIL } \langle \text{clock_name} \rangle \text{’event AND } \langle \text{clock_name} \rangle = \langle \text{condition} \rangle; \)

• a high-level synthesis process should be used allowing a change in the cycle-by-cycle behavior without any limitations (e.g. free-float I/O scheduling mode).

• the VHDL specification can consist of assignments or loops with a known number of loop passes. For loops with an unknown number of loop passes, timing information for the single loop pass can be extracted.

Following these requirements the extraction of the timing constraints can be executed in 4 steps:

1. A POM representing the algorithmic specification (POM\(_{\text{SPEC}}\)) is generated using an event execution semantics.

2. A POM representing the timing information of the RT stimuli set (POM\(_{\text{TC}}\)) is created using an event timing semantics.

3. The event execution semantics of the POM\(_{\text{TC}}\) is mapped onto the event timing semantics of the POM\(_{\text{SPEC}}\).

4. The RT timing constraints are assigned to the POM\(_{\text{SPEC}}\) states. As a result, the POM\(_{\text{SPEC}}\) is restricted by the RT timing constraints. The modified VHDL specification is produced.

3.2.1 POM Semantics

The basis of our approach is the POM described in Section 2. Now, event semantics have to be defined that consider the data dependencies between the I/O signals in the specification and the timing information contained in the simulation vector set. Therefore, two different event semantics are necessary: an event execution semantics (Definition 3) for the POM\(_{\text{SPEC}}\) and an event timing semantics for the POM\(_{\text{TC}}\).

**Definition 3:** event execution semantics

A POM event \( e_i (i = 0,..,n) \) is defined as a read operation from an interface signal or as a write operation to an interface signal.

For illustration purposes, the example in Program 1 is used. In this example, 9 different events can be detected. These events can be summed up corresponding to their occurrence in the VHDL specification (Event Execution Collection).

```vhdl
PROCESS (e0,e1,e2,e3)
BEGIN
  x <= e0 + e1 + e2;
  y <= e3;
  z <= e1 + e3;
END PROCESS;
```

**Program 1: A simple VHDL example**

The resulting events and event collections are listed in Table 1. It is important to note that a collection can easily be created when input signals are directly assigned to output signals. Using variables a collection can be created achieving a dataflow analysis, i.e. it has to be clarified from which input signals an output signal is dependent.

<table>
<thead>
<tr>
<th>Event Execution Collection</th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP0123</td>
<td>sp0, sp1, sp2</td>
<td>read e0, e1, e2</td>
</tr>
<tr>
<td></td>
<td>sp3</td>
<td>write x</td>
</tr>
<tr>
<td>SP45</td>
<td>sp4, sp5</td>
<td>read e3</td>
</tr>
<tr>
<td></td>
<td>sp6</td>
<td>write y</td>
</tr>
</tbody>
</table>

**Table 1: Event execution list**

Concerning the event timing semantics a further distinction has to be made. Simulation vector sets can be implemented in several ways. Most important is the specification of the expected design results corresponding to the defined stimuli. In VHDL, these results can be specified by using ASSERT statements. Thus, the event timing semantics in Definition 4 can be used.

**Definition 4:** event timing semantics (ASSERT)

A POM event \( e_i (i = 0,..,n) \) is defined as a write operation to an interface signal, as a VHDL WAIT construct or as a VHDL ASSERT construct.

A simple example of a VHDL simulation vector set is given in Program 2. The resulting event timing list is given in Table 2.

```vhdl
PROCESS (e0,e1,e2)
BEGIN
  e0 <= 5; e1 <= 7; e2 <= 13;
  WAIT FOR 5 ns;
  ASSERT (x = 25);
  e3 <= 9;
  WAIT FOR 2 ns;
  ASSERT (y = 9);
  ASSERT (z = 16);
END PROCESS;
```

**Program 2: Simulation vector set with ASSERT statement**
In some cases, the expected design results are not described in VHDL. Therefore, another event timing semantics have to be used (Definition 5).

**Definition 5:** event timing semantics (READ)
A POM event \( e_i \) \((i = 0, \ldots, n)\) is defined as a VHDL WAIT construct or as a write operation to an interface signal.

As stimuli, the vector set described in Program 2 is used but without any ASSERT statement. As a result, the event timing list consists of the following events:

After the production of the different event lists, the generation of the POM SPEC and the POM TC can be started.

### 3.2.2 Generation of the POM SPEC

The generated event execution list can now be used for the generation of the POM SPEC. The generation of the POM SPEC starts with a first state \( \text{state0} \), where „no event has occurred“. The three assignments in the VHDL specification represented as Event Execution Collections are independent. Consequently, the independent relation are used and the remaining states are generated. E.g. SP67 has currently not occurred and SP0123 is independent from SP45, thus \( \text{state1} \) and \( \text{state2} \) and \( \text{state3} \) have to be generated. To summarize the following relations are used:

\[
\begin{align*}
\text{SP0123} & \land \text{SP45} \land \text{SP67} \\
\text{SP0123} & \land \text{SP45} \land \neg \text{SP67} \\
\neg \text{SP0123} & \land \text{SP45} \land \neg \text{SP67} \\
\neg \text{SP0123} & \land \neg \text{SP45} \land \text{SP67} \\
\neg \text{SP0123} & \land \neg \text{SP45} \land \neg \text{SP67} \\
\end{align*}
\]

The POM SPEC can also be described as logical representation:

\[
f_{\text{POM-SPEC}} = (\neg \text{SP0123} \land \neg \text{SP45} \land \neg \text{SP67}) \lor
\neg \text{SP0123} \land \text{SP45} \land \neg \text{SP67} \lor
\neg \text{SP0123} \land \text{SP45} \land \text{SP67} \lor
\neg \text{SP0123} \land \neg \text{SP45} \land \text{SP67} \lor
\neg \text{SP0123} \land \neg \text{SP45} \land \neg \text{SP67}
\]

### 3.2.3 Generation of the POM TC

Using one of the „event timing semantics“ definitions, the generated event timing list can now be applied for the generation of the POM TC.

Using Definition 4 „event timing semantics (ASSERT)“, a POM TC is generated with the relations:

\[
t_c_{a0} \land \text{write e0} \land t_c_{a1} \land \text{write e1} \land t_c_{a2} \land \text{write e2} \land t_c_{a3} \land \text{WAIT FOR 5 ns} \land t_c_{a4} \land \text{ASSERT x} \land t_c_{a5} \land \text{write e3} \land t_c_{a6} \land \text{WAIT FOR 2 ns} \land t_c_{a7} \land \text{ASSERT y} \land t_c_{a8} \land \text{ASSERT z}
\]

These relations express the following order: the three first write operations \( e0, e1 \) and \( e2 \) are independent, but they are all predecessors from the first WAIT construct (\( \text{WAIT FOR 5 ns} \)). This WAIT construct has again to be executed before the \( \text{ASSERT x} \) statement. The write operation \( e3 \) and the \( \text{ASSERT x} \) statement are executed before the second WAIT (\( \text{WAIT FOR 1,5 ns} \)) construct. Further, the second WAIT construct has to be executed before the \( \text{ASSERT y} \) and the \( \text{ASSERT z} \) statement. These last two \( \text{ASSERT} \) statements are independent from each other.

Corresponding to the generation of the POM SPEC, these relations are used to produce the POM TC states and the logical representation that internally is also represented as BDD.

### Tables:

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tc_r0</td>
<td>write e0</td>
</tr>
<tr>
<td>tc_r1</td>
<td>write e1</td>
</tr>
<tr>
<td>tc_r2</td>
<td>write e2</td>
</tr>
<tr>
<td>tc_r3</td>
<td>WAIT FOR 5 ns</td>
</tr>
<tr>
<td>tc_r4</td>
<td>write e3</td>
</tr>
<tr>
<td>tc_r5</td>
<td>WAIT FOR 1.5 ns</td>
</tr>
</tbody>
</table>

Table 3: Event timing list (READ)

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tc_a0</td>
<td>write e0</td>
</tr>
<tr>
<td>tc_a1</td>
<td>write e1</td>
</tr>
<tr>
<td>tc_a2</td>
<td>write e2</td>
</tr>
<tr>
<td>tc_a3</td>
<td>WAIT FOR 5 ns</td>
</tr>
<tr>
<td>tc_a4</td>
<td>ASSERT x</td>
</tr>
<tr>
<td>tc_a5</td>
<td>write e3</td>
</tr>
<tr>
<td>tc_a6</td>
<td>WAIT FOR 2 ns</td>
</tr>
<tr>
<td>tc_a7</td>
<td>ASSERT y</td>
</tr>
<tr>
<td>tc_a8</td>
<td>ASSERT z</td>
</tr>
</tbody>
</table>

Table 2: Event timing list (ASSERT)

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>SP0123</td>
</tr>
<tr>
<td>state0</td>
<td>0</td>
</tr>
<tr>
<td>state1</td>
<td>0</td>
</tr>
<tr>
<td>state2</td>
<td>1</td>
</tr>
<tr>
<td>state3</td>
<td>1</td>
</tr>
<tr>
<td>state4</td>
<td>1</td>
</tr>
<tr>
<td>state5</td>
<td>1</td>
</tr>
<tr>
<td>state6</td>
<td>0</td>
</tr>
<tr>
<td>state7</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: POM SPEC
ping of the POM\textsubscript{TC} onto the POM\textsubscript{SPEC} and the final generation of the algorithmic VHDL specification.

### 3.2.4 Mapping of the different Event Semantics

The main problem during the mapping process is the different semantics of the single events in the POM\textsubscript{TC} and the POM\textsubscript{SPEC}. Due to this fact, for certain events additional information have been collected, e.g. the Event Execution Collection sums up read and write operations in the specification. In summary, there are two steps to be executed:

1. A mapping file of the event names has automatically to be produced that indicates which event from the event execution list corresponds to which event from the event simulation list.

2. The timing events have to be summed up in Event Timing Collections.

#### Figure 6: Generation of the Event Name Mapping File

The first step is the generation of the event name mapping (ENM) file. For illustration purposes, the mapping of our simple assignment example is shown in Figure 6. In this figure, Definition 4 is used as event timing semantics. After the creation of the mapping file, for every Event Execution Collection (EEC) an Event Timing Collection (ETC) can be created. The read and write operations summed up in the EEC are assigned to the corresponding write and assert events in the ETL. These events create the first part of the ETC. The second part are the timing information contained in the WAIT events. Using the event timing list (ETL) and the POMs, the present events of the ETC are examined whether a timing event belongs to them or not. The principal algorithm is given in the following:

```
Algorithm 1: Finding Event Timing Collection

\begin{algorithm}
\caption{Finding Event Timing Collection ($M_{EEC}$)}
\begin{algorithmic}
\Procedure{Find_EventTimingCollection ($M_{EEC}$)}{$M_{EEC}$ := $\emptyset$; \Repeat
\State take EEC and evaluate events($M_{EEC}$, ENM, $M_{ETC_{\text{new}}}$); \State take ETC and insert timing($M_{ETC_{\text{new}}}$, ETL, $M_{ETC_{\text{new}}}$); \State $M_{ETC}$ := $M_{ETC}$ \cup $M_{ETC_{\text{new}}}$; \Until{examined all EEC; \EndProcedure
\end{algorithmic}
\end{algorithm}
```

The resulting Event Timing Collections are listed in Figure 7.

#### Figure 7: Generation of the Event Timing Collection

So far, the event timing semantics (ASSERT) has been used for this method. Using the event timing semantics (READ) it is more difficult to insert the timing information into the Event Timing Collection. Therefore, the following definition is necessary:

**Definition 6:** A WAIT event is valid for every output signal that are reachable from the input signal updated previously.

With this definition and the POM\textsubscript{TC} relations, it is possible to insert the timing information in the Event Timing Collection. E.g. the write operation write $x$ contained in the EEC SP0123 can be executed as soon as the read operation read $e0$, $e1$, $e2$ are executed. These read operations correspond to the write operations write $e0$, $e1$, $e2$ in the ETC. Using the POM\textsubscript{TC} relations the three read operations ($tc_{r0}$, $tc_{r1}$, $tc_{r2}$) are predecessors of the $tc_{r3}$ event (WAIT FOR 5 $ns$). Concerning our definition, the output signal $x$ is reachable from the input signals $e0$, $e1$, and $e2$ which have previously updated. Therefore, the event $tc_{r3}$ is inserted in the ETC. The following algorithm determines the timing information to be included.

```
algorithm Insert_TimingInformation(POM\textsubscript{TC}, $M_{EEC}$, $M_{ETC}$)
for all EEC $\in M_{EEC}$ do
\Repeat
\State take EEC output and search inputs(EEC, $M_{input}$); \State search ETC outputs($M_{input}$, ENM, $M_{output}$); \State traverse POM\textsubscript{TC} relations($M_{ETC}$, $M_{output}$, $M_{TC}$); \State $ETC := ETC \cup M_{TC}$; \Until{examined all outputs; \Enddo
end Insert_TimingInformation;
```

```
Algorithm 2: Inserting Timing Information in the ETC
```
3.2.5 Assignment of the Timing Constraints to the POM$_{SPEC}$ states

The last steps are the assignment of the RT timing information to the POM$_{SPEC}$ states, and the production of the modified VHDL specification. The first step has nearly solved in the last section. The Event Execution Collection has only to be extended by the timing information in the Event Timing Collection, e.g. SP0123 consists now of the events: read e0, e1, e2; wait5; write x. These timing information has now to be inserted in the algorithmic VHDL specification. In the following, the timing semantics concerning the ASSERT interpretation is used. Indeed, this method works similar using the semantics concerning the READ interpretation.

However, for every read or write operation from or to an interface signal the corresponding Event Execution Collection must have to be examined. During this examination, timing conflicts eventually occurred have to be solved. For illustration purposes, the simple assignment specification is used again. The first statement concerning interface signals is: $x <= e0 + e1 + e2;$. The output signal $x$ is contained in the EEC$_{TC}$ SP0123. In this collection, one wait event has been inserted. This wait event indicates that the signal $x$ is read from the simulation vector set after 5 ns. As a result, the signal $x$ has to be written after 5 ns at the latest. A first WAIT statement is therefore included in the VHDL specification with a time range from 0 ns to 5 ns.

The next interesting statement is $y <= e3;$. The output signal $y$ is contained in the EEC$_{TC}$ SP45. In this collection, also a wait event has been inserted. Due to the fact that this is the second statement examined the previous timing information have to be considered. This can be handled using the relations of the POM$_{TC}$. There, the event tc_a7 (assert y) is a predecessor of tc_a6 (WAIT FOR 2 ns) which is finally a predecessor of tc_a4 (WAIT FOR 5 ns). Consequently, the signal $y$ is read after 7 ns (5 ns + 2 ns), and has to be written to the same time at the latest. A second WAIT statement with a time range from 0 ns to 2 ns has to be inserted.

Program 3: VHDL specification with timing information

The last statement in our example is: $z <= e1 + e3;$. In the corresponding collection two wait events have been inserted. These waits are again the events tc_a4 and tc_a7. This means, the signal $z$ is read after 7 ns, and has to be written at this time at the latest. Due to the already inserted WAITs, this time is just reached and so, no further WAIT statements have to be inserted. The resulting VHDL specification can be seen in Program 3.

During this insertion process it is possible that timing conflicts occur, e.g. in the simulation vector set the signals e1 and e3 are written first, and then the signals y and z are read. Finally, the signals e0 and e2 are written and the signal x are read. In this case, the assignment order implemented in the simulation vector set has to be mapped onto the specification. This can be executed by comparing the two POMs with a simple conjunction of the logical formula [8]. However, the original events and not the event collection have to be used for this comparison. Further, it is important to note, that already defined WAIT statements as well as a sensitivity list in the specification are cancelled.

One point that has still to be clarified is the usage of WAIT UNTIL constructs in the simulation vector set. It is possible to use these constructs if the clock rate is known. In this case, the clock statements can be transformed in absolute timing values, and can then be handled like a WAIT FOR construct.

4 Results

The proposed technique has been implemented in the POM and the CADDY-II system, and the timing constraints have been extracted successfully from several examples. Some of these examples are listed in Table 5. They are: (1) our simple assignment example (2) communication example with two processes (3) differential equation (4) ggt (5) FFT (6) simulated annealing processor. The CPU time for the POM computation for all examples on a Pentium II (233 MHZ, 80 MB RAM) was less then 1 second.

<table>
<thead>
<tr>
<th>Design</th>
<th>Events</th>
<th>POM$_{SPEC}$</th>
<th>POM$_{TC}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>spec</td>
<td>stimuli</td>
<td>state</td>
</tr>
<tr>
<td>assignment1</td>
<td>9 (3 EEC)</td>
<td>9</td>
<td>135</td>
</tr>
<tr>
<td>assignment2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>communication</td>
<td>P1: 4 (3 EEC)</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>P2: 6 (4 EEC)</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>differential</td>
<td>6 (6 EEC)</td>
<td>40</td>
<td>39</td>
</tr>
<tr>
<td>equation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ggt</td>
<td>4 (2 EEC)</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>FFT</td>
<td>3 (2 EEC)</td>
<td>21</td>
<td>4</td>
</tr>
<tr>
<td>simulated</td>
<td>12 (14 EEC)</td>
<td>13</td>
<td>231</td>
</tr>
<tr>
<td>annealing</td>
<td>processor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: POM Generation Results

In the 2. and the 3. column, the number of specification and stimuli events are given. In some cases, the simulation...
vector sets consist of several simulation cycles. In the 4. and the 5. column, the number of generated POM states and BDD nodes for the POM \text{SPEC} and the POM \text{TC} are listed. Above all, the number of specification and stimuli events as well as the used relations are important, because they determine the number of POM states that has to be generated. However, the maximum number of POM states, that can be handled, depends on the implementation of the BDD library. Usually, in a BDD library several features are contained to handle huge BDDs, e.g. partitioning, cache. Moreover, special procedures are available to change dynamically the variable order, and this can also lead to smaller BDDs. Besides, most of the I/O signals will be independent, and due to the definition of the independent relation as a tautology, this reduces the number of BDD nodes (Table 5: 5. and 7. column). For the examples in Table 5, no BDD optimization options have been used.

<table>
<thead>
<tr>
<th>Design</th>
<th>Extracted Timing Information</th>
<th>Refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignment\text{\textsubscript{1}}</td>
<td>2 \text{ WAIT FOR}</td>
<td>no</td>
</tr>
<tr>
<td>assignment\text{\textsubscript{2}}</td>
<td>2 \text{ WAIT FOR}</td>
<td>yes</td>
</tr>
<tr>
<td>communication</td>
<td>2’2 \text{ WAIT FOR}</td>
<td>no</td>
</tr>
<tr>
<td>differential equation</td>
<td>4 \text{ WAIT FOR}</td>
<td>yes</td>
</tr>
<tr>
<td>ggf</td>
<td>2 \text{ WAIT FOR}</td>
<td>no</td>
</tr>
<tr>
<td>FFT</td>
<td>2 \text{ WAIT FOR}</td>
<td>no</td>
</tr>
<tr>
<td>simulated annealing processor</td>
<td>1 \text{ WAIT FOR}</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 6: Extraction Results

In Table 6, the number of the extracted timing information are given. With some designs, a re-ordering of the specification (refinement) due to the timing information was necessary. This information is listed in the 3. column.

5 Conclusion

In this paper, a new verification technique has been presented for extracting timing information from RT simulation vector sets through a Partial Order based Model. Moreover, timing information defined in the algorithmic specification can be validated using this extracting method. A key feature of this approach is the usage of a Partial Order based model and the definition of event semantics for the specification and the simulation vector set. The Partial Order based Models can be represented as logical formula. These logical representations are implemented as BDD's and, due to the chosen event semantics, a high degree of parallel read and write operations to and from the interface signals lead to smaller BDD's. Further, an automated extraction and mapping process has been developed and, besides, the generation of a VHDL specification containing RT timing information. Results from several designs demonstrated the effectiveness of the approach.

6 References