An Approach for Extracting RT Timing Information to Annotate Algorithmic VHDL Specifications

Cordula Hansen
Forschungszentrum Informatik (FZI)
at the University of Karlsruhe
hansen@fzi.de

Francisco Nascimento
University of Tübingen
moreira@informatik.uni-tuebingen.de

Wolfgang Rosenstiel
University of Tübingen and FZI
rosen@informatik.uni-tuebingen.de

ABSTRACT *- This paper presents a new approach for extracting timing information defined in a simulation vector set on register transfer level (RTL) and reusing them in the behavioral specification. Using a VHDL RTL simulation vector set and a VHDL behavioral specification as entry, the timing information are extracted and as well as the specification transformed in a Partial Order based Model (POM). The POM expressing the timing information is then mapped on the specification POM. The result contains the behavioral specification and the RTL timing and is retransformed in a corresponding VHDL specification. Additionally, timing information contained in the specification can be checked using the RTL simulation vectors.*

1. Introduction

Up to now, most ASIC or FPGA designers have developed synthesizable register transfer level (RTL) models and the corresponding stimuli usually implemented in VHDL or Verilog. In this context, several approaches have been published to facilitate the development of RTL specifications and RTL stimuli [7] [8]. More and more, industrial VHDL specifications are implemented as behavioral descriptions on the algorithmic level. Also already existing RT specifications are now often integrated in algorithmic VHDL descriptions. Thus, a new question arise: is it possible to reuse information defined in the RTL simulation vector set for a specification now on an algorithmic level? This paper addresses the problem of extracting timing information from the VHDL RTL simulation vectors for the algorithmic specification. Main feature of the presented approach is the integration of the extracted timing information 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 information contained in the RT simulation vector set. This allows an efficient reuse of already existing timing information and, therefore, a reduction of development time.

The 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.. 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) [9] [2]. This model is event based, and each event is defined as a unique instance of an action and can not occur more than once 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 this execution is assumed.

2.1 Basic Definitions

The POM is based on the Chu space formalism [3]. 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: $x_i = \{a | (a \in A) \wedge (R(a, x_i) = 1)\}$

POMs can be represented as a matrix or as a logical formula (Figure 1, 2). In the matrix, each entry (*a*,*x*) contains the value of the occurrence relation. Consequently, the matrix rows correspond to the POM states. Considering the matrix as a truth table, the POM can be represented as a logical formula.

Definition 2: The logical representation f_C of the POM *C* is defined as: $f_C = \vee f_{x_i}$ where $n = |X|$ and f_{x_i} is the logical formula corresponding to $x_i \in X$, defined as follows:

$$f_{x_i} = (\wedge\{a \mid (a \in A) \wedge (R(a, x_i) = 1)\}) \wedge (\wedge\{(\neg a) \mid (a \in A) \wedge (R(a, x_i) = 0)\})$$

In the logical representation, the events are defined as variables, the states are defined as terms of the formula, and the relation *R* determines whether the variable appears complemented or not. The logical formula *f* is true for each state that is permitted in the POM. Due to the fact that a partial order exists between the states, the execution of a POM can be defined using the concept of successor states. It is important to emphasize that if an event *a* has occurred in a state *x*, then this event has also occurred in all successor states of *x* and, consequently, the occurrence relation holds.

2.2 Relations Between Events

Concerning the POM, four basic relations between events are defined: the independence, the precedence, the conflict, and the disjunctive enable relation.

2.2.1 Independence Relation

The independence relation represents the independent execution of two events *a* and *b* ($a \nabla b$). The POM for this relation can be seen in Figure 1 (a). No order is imposed to the occurrence of the events *a* and *b*.

There are two reasons to represent the independence relation as a tautology. One is related to the use of BDDs [1] 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

DAC 99, New Orleans, Louisiana

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.

(a) Independence: $a \nabla b$

	a	b
x_0	0	0
x_1	0	1
x_2	1	0
x_3	1	1

$f_\nabla = \bar{a}\,\bar{b} + \bar{a}\,b + a\,\bar{b} + a\,b$
$= 1$

(b) Precedence: $a < b$

	a	b
x_0	0	0
x_1	1	0
x_2	1	1

$f_< = \bar{a}\,\bar{b} + a\,\bar{b} + ab$
$= a + \bar{b}$

Figure 1. Independence and Precedence Relations

2.2.2 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 1 (b). This relation is used to model the sequential execution of events.

2.2.3 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 2 (a).

2.2.4 Disjunctive Enable Relation

The disjunctive enable relation permits two events to be represented, whose executions disjunctively enable a third event, i.e. *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 2 (b) represents the POM and the corresponding logical formula.

(a) Conflict: $a \# b$

	a	b
x_0	0	0
x_1	0	1
x_2	1	0

$f_\# = \bar{a}\,\bar{b} + \bar{a}\,b + a\,\bar{b}$
$= \bar{a} + \bar{b}$

(b) Disjunctive Enable: den(a,b,c)

	a	b	c
x_0	0	0	0
x_1	0	1	0
x_2	0	0	1
x_3	0	1	1
x_4	1	0	1
x_5	1	1	0
x_6	1	1	1

$f_{den} = a \rightarrow (b + c)$

Figure 2. Conflict and Disjunctive Enable Relations

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:

- the extraction of timing constraints from an RT simulation vector set (Figure 3 (a)). The timing information defined there can be used to restrict the timing behavior of the specification.
- the validation of timing constraints defined in the algorithmic specification (Figure 3 (b)-(c)). The timing constraints extracted from the specification can be compared with the extracted timing information of the RT simulation vector set

In the following, the design flow is described containing the CADDY-II synthesis system [4] [5], and the extraction of timing information, as well as the generation of the POMs.

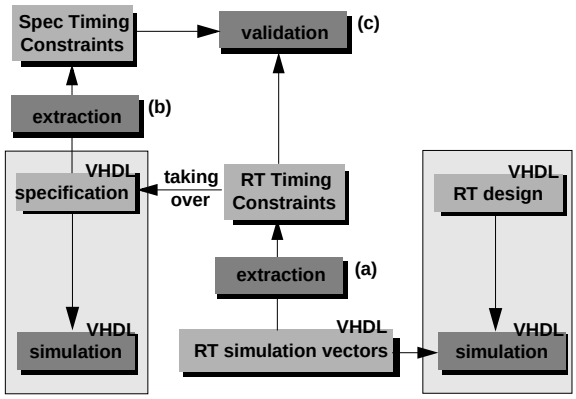

Figure 3. An Overview of the Extraction and Validation of Timing Constraints

In the first step, a VHDL preprocessing is started. The main task throughout this phase is the generation of a flow graph description. 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 (POM_{SPEC}). After the generation of the preprocessor results, the POM_{SPEC} 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 executed. This process transforms the behavioral algorithmic specification into a structural RT description. The resulting RT design is usually represented by a VHDL description. The next step is the generation of the second POM (POM_{TC}) representing the timing information contained in the RT simulation vector set. This task is 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).

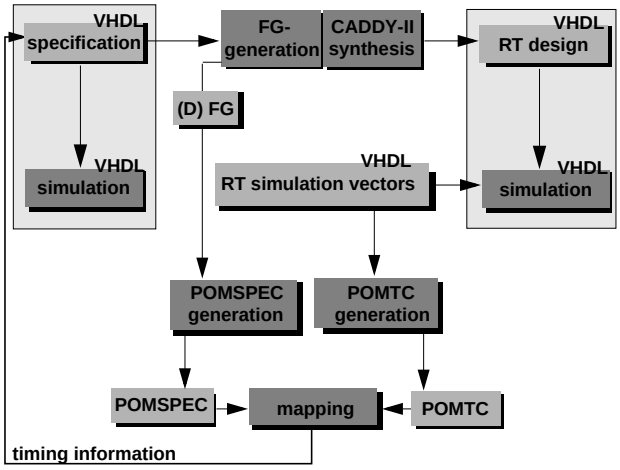

Figure 4. Extraction of Timing Constraints

The second task, the validation of timing constraints defined in the algorithmic specification can be solved using the same methods. Instead of generating a POM_{SPEC}, a POM is generated representing the timing constraints in the algorithmic specification. Finally, this POM is compared with the POM_{TC}.

Due to the fact, that the POMs internally are represented as BDDs, the implementation of a comparison of two POMs can be implemented using a simple conjunction [6].

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 defined 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 synchronous timing behavior. Timing information 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: *WAIT FOR <time>;* or
  WAIT UNTIL <clock_name>'event AND <clock_name> = <condition>;
- the VHDL specification can consist of assignments or loops with a known number of loop passes. For loops with a unknown number of loop passes, timing information for a single loop pass can be extracted.

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

1. A POM representing the algorithmic specification (POM_{SPEC}) is generated using an event execution semantics.
2. A POM representing the timing information of the RT stimuli set (POM_{TC}) is created using an event timing semantics.
3. The event execution semantics of the POM_{TC} is mapped on the event timing semantics of the POM_{SPEC}.
4. The RT timing constraints are assigned to the POM_{SPEC} states. As a result, the POM_{SPEC} is restricted by the RT timing information. 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_{SPEC} and an event timing semantics for the POM_{TC}.

Definition 3: 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, Example 1 is used. In this example, 9 different events can be detected. These events are summed up to *Event Execution Collections*. Every collection consists of one output signal and the corresponding input signals. The resulting events and event collections are listed in Table 1, e.g. in column 1, an event collection *SP0123* has been generated consisting of the events *sp0* (read e0), *sp1* (read e1), *sp2* (read e2), and *sp3* (write x).

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

Example 1. A simple VHDL program

It is important to note that an *Event Execution 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. The dataflow analysis is used to detect the input signals from which the actual output signal is dependent.

Event Execution Collection	Event	Description
SP0123	sp0, sp1,sp2 sp3	read e0, e1, e2 write x
SP45	sp4, sp5	read e3 write y
SP678	sp6, sp7, sp8	read e1, e3 write z

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. Hence, the event timing semantics in Definition 4 can be used.

Definition 4: (first event timing semantics)
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.

```
PROCESS
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;
```

Example 2. Simulation vector set with ASSERT statement

Event	Description
tc_a0	write e0
tc_a1	write e1
tc_a2	write e2
tc_a3	WAIT FOR 5 ns
tc_a4	ASSERT x
tc_a5	write e3
tc_a6	WAIT FOR 2 ns
tc_a7	ASSERT y
tc_a8	ASSERT z

Table 2: First event timing list

Sometimes the expected design results are not described in the VHDL simulation vector set but in separate data files. Therefore, another event timing semantics have to be used (Definition 5).

Definition 5: (second event timing semantics)
A POM e*vent* e_i *(i = 0,..,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 statements. The resulting event timing list

consists of events listed in Table 3. After the production of the different event lists, the generation of the POM_{SPEC} and the POM_{TC} can be started.

Event	Description
tc_r0	write e0
tc_r1	write e1
tc_r2	write e2
tc_r3	WAIT FOR 5 ns
tc_r4	write e3
tc_r5	WAIT FOR 1,5 ns

Table 3: Second event timing list

3.2.2 Generation of the POM_{SPEC}

The generated event execution list is used for the generation of the POM_{SPEC}. The generation of the POM_{SPEC} starts with a first state (*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: *SP0123* ∇ *SP45*, *SP0123* ∇ *SP67*, *SP45* ∇ *SP67*, and the resulting states are generated.

state	SP0123	SP45	SP67
state0	0	0	0
state1	0	1	0
state2	1	0	0
state3	1	1	0
state4	1	0	1
state5	1	1	1
state6	0	0	1
state7	0	1	1

Table 4: POM_{SPEC}

The POM_{SPEC} can also be described as logical representation:

$$
\begin{aligned}
f_{POM\text{-}SPEC} = \quad & (\neg SP0123 \wedge \neg SP45 \wedge \neg SP67) \vee \\
& (\neg SP0123 \wedge SP45 \wedge \neg SP67) \vee \\
& (SP0123 \wedge \neg SP45 \wedge \neg SP67) \vee \\
& (SP0123 \wedge SP45 \wedge \neg SP67) \vee \\
& (SP0123 \wedge \neg SP45 \wedge SP67) \vee \\
& (SP0123 \wedge SP45 \wedge SP67) \vee \\
& (\neg SP0123 \wedge \neg SP45 \wedge SP67) \vee \\
& (\neg SP0123 \wedge SP45 \wedge SP67)
\end{aligned}
$$

3.2.3 Generation of the POM_{TC}

Using one of the event timing semantic definitions, the generated event timing list is now applied for the generation of the POM_{TC}. Using Definition 4, a POM_{TC} is generated with the relations:

$$
\begin{aligned}
& tc\_a0 \nabla tc\_a1,\ tc\_a0 \nabla tc\_a2,\ tc\_a1 \nabla tc\_a2,\ tc\_a0 < tc\_a3, \\
& tc\_a1 < tc\_a3,\ tc\_a2 < tc\_a3,\ tc\_a3 < tc\_a4, \\
& tc\_a4 \nabla tc\_a5,\ tc\_a4 < tc\_a6,\ tc\_a5 < tc\_a6, \\
& tc\_a6 < tc\_a7.,\ tc\_a6 < tc\_a8,\ tc\_a7 \nabla tc\_a8
\end{aligned}
$$

These relations express the following order: the first three write operations *e0*, *e1* and *e2* are independent, but they are all predecessors from the first WAIT construct (*WAIT FOR 5 ns*). This WAIT construct has again to be executed before the *ASSERT x* statement. The write operation *e3* and the *ASSERT x* statement are executed before the second WAIT (*WAIT FOR 1,5 ns*) construct. Further, the second WAIT construct has to be executed before the *ASSERT y* and the *ASSERT z* statement. These last two 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 formula that internally is represented as BDD. The generation of the POM_{TC} using Definition 5 is similar to the already described strategy. The last step is the mapping of the POM_{TC} on the POM_{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_{TC} and the POM_{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.

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 5. In this figure, Definition 4 is used as event timing semantics. It can easily be stated that every read operation respectively write operation in the specification correspond to an write operation respectively assert statement in the stimuli set.

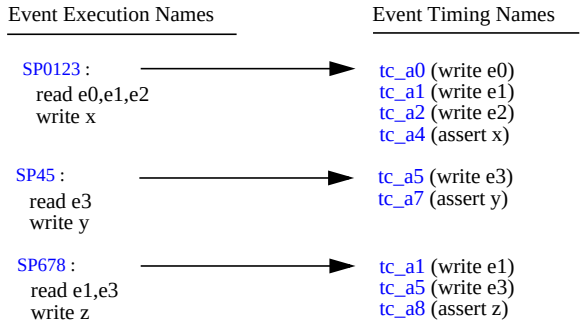

Figure 5. Generation of the Event Name Mapping File

After the creation of the mapping file, for every Event Execution Collection (EEC) an Event Timing Collection (ETC) is created (Figure 6).

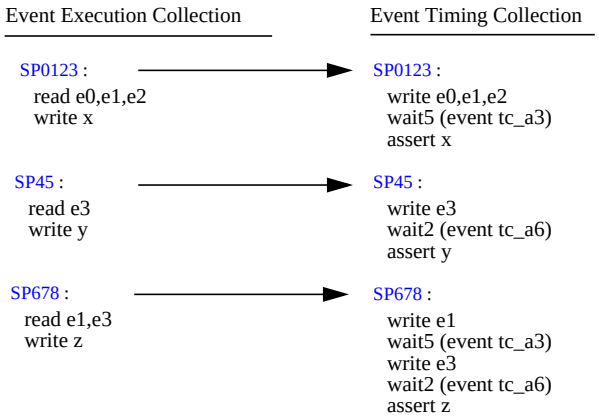

Figure 6. Generation of the Event Timing Collection

The read and write operations summed up in the EEC are assigned to the corresponding write and assert events in the event timing list (ETL). These events create the first part of the ETC. The second part are the timing information contained in the WAIT events.

Using the ETL and the POMs, every event of the ETC are examined whether a timing event belongs to it or not. The principal algorithm is given in the following:

algorithm Find_EventTimingCollection(M_{EEC})
  $M_{ETC} := \emptyset$;
  repeat
    take_EEC_and_evaluate_events(M_{EEC}, *ENM*, $M_{ETCpart1}$);
    take_ETC_and_insert_timing($M_{ETCpart1}$, *ETL*, M_{ETCnew});
    $M_{ETC} := M_{ETC} \cup M_{ETCnew}$;
  until examined_all_EEC;
end Find_EventTimingCollection;

Figure 7. Finding Event Timing Collection

So far, the event timing semantics has been used for this method. Using the second event timing semantics 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_{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_{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 been 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_{TC},M_{EEC},M_{ETC})
  for all $EEC \in M_{EEC}$ **do**
    repeat
      take_EEC_output_and_search_inputs(*EEC*, M_{inputs});
      search_ETC_outputs(M_{inputs}, *ENM*, $M_{outputs}$);
      traverse_POM_{TC}_relation(POM_{TC}, $M_{outputs}$, M_{TC});
      $ETC := ETC \cup M_{TC}$;
    until examined_all_outputs;
  end do;
end Insert_TimingInformation;

Algorithm 1. Inserting Timing Information in the ETC

3.2.5 Assignment of the Timing Information 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 of the events: *read e0, e1, e2*; *wait5*; *write x*. These timing information has 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 second event timing semantics .

However, for every read or write operation from or to an interface signal the corresponding Event Execution $Collection_{TC}$ has 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.

```
PROCESS
BEGIN
    -- t1 : time range 0 ns to 5 ns;
    WAIT FOR t1;
    x <= e0 + e1 + e2;
    -- t2 : time range 0 ns to 2 ns;
    WAIT FOR t2;
    y <= e3;
    z <= e1 + e3;
END PROCESS;
```

Algorithm 2. 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 2.

Throughout the whole 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* is read. In this case, the assignment order implemented in the specification has to be refined. This can be executed by comparing the two POMs (POM_{SPEC} and POM_{TC}) with a simple conjunction of the logical formula. Due to space limitations, a more detailed description of that task is beyond the scope of this paper. For further information the reader is referred to [6]. However, it is important to note, that already defined WAIT statements as well as a sensitivity list in the specification are deleted.

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 are transformed in absolute timing values, and are then 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 an Ultra Sparc Station 1 was less than 1 millisecond.

In columns 2 and 3, the number of specification (spec) and stimuli events are given. In example (3), (4) and (5), the simulation vector sets consist of several simulation cycles. In columns 4 and 5, the number of generated POM states and BDD nodes for the POM_{SPEC} and the POM_{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. The number of events increase with the number of accesses to the interface signals. Therefore, for this

approach a complex example is a design with a large number of read and write operations from and to the interface signals.

Design	Events		POM_{SPEC}		POM_{TC}	
	spec	stimuli (assert)	state	node	state	node
$assignment_1$ $assignment_2$	9 (3 EEC)	9	135	11	24	14
communica-tion	P1: 4 (3 EEC) P2: 6 (4 EEC)	12	6 13	7 14	15	23
differential equation	6 (6 EEC)	40	39	15	85	80
ggt	4 (2 EEC)	30	6	7	37	59
FFT	3 (2 EEC)	21	4	6	7	12
simulated annealing processor	12 (14 EEC)	13	231	37	42	21

Table 5: POM Generation Results

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 reduce the number of BDD nodes (Table 5: 5. and 7. column). For the examples in Table 5, no BDD optimization options have been used.

Design	Extractable RT Timing Information	Inserted Timing Constructs	Refinement
$assignment_1$	2 WAIT UNTIL	2 WAIT FOR	no
$assignment_2$	2 WAIT UNTIL	2 WAIT FOR	yes
communication	2 * 1 WAIT FOR 2 * 1 WAIT UNTIL	2 * 2 WAIT FOR	no
differential equation	2 WAIT FOR 2 WAIT UNTIL	2 WAIT FOR 2 WAIT UNTIL	yes
ggt	2 WAIT FOR	2 WAIT UNTIL	no
FFT	2 WAIT FOR	2 WAIT UNTIL	no
simulated annealing processor	3 WAIT FOR 2 WAIT UNTIL	5 WAIT FOR	no

Table 6: Extraction Results

In Table 6, for every design the number of extracted timing information are given. All RT stimuli sets have been specified with WAIT FOR and WAIT UNTIL constructs (column 2). During the insertion process the WAIT UNTIL constructs have been automatically transformed into WAIT FOR statements (column 3). It is also possible to maintain the WAIT UNTIL constructs (column 3). This is suitable if the algorithmic description has been specified using a clock signal. With some designs, a re-ordering of the specification (refinement) due to the timing information was necessary. This information is listed in column 3. Above all, this refinement is relevant for designs with a large number of signal assignments, e.g. filter functions.

In summary, the presented results show several advantages of our extraction strategy. The system automatically identify the extractable RT timing information. WAIT UNTIL statements used in the RT stimuli set are automatically transformed in WAIT FOR constructs if necessary. Furthermore, an automated refinement of the specification are executed.

5. Conclusion

In this paper, a new method 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 BDDs 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 BDDs. Further, an automated extraction and mapping process as well as the generation of a VHDL specification containing RT timing information have been developed. The RT timing constructs are automatically adapted and the specification is refined if necessary. Results from several designs demonstrated the effectiveness of the approach.

6. References

[1] Bryant, R.E.: "*Symbolic boolean manipulation with ordered binary-decision diagrams*", ACM Computing surveys 24, 3 (September 1993), pp. 293-318.

[2] Garcez, E.; Nascimento, F.: "*A Model Checker for a Partial Order based Model of Concurrency*", Proceedings of Workshop of Beschreibungs-sprachen und Modellierungsparadigmen, March 1998.

[3] Gupta, V.: "*Chu Spaces: A Model of Concurrency*", PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, USA, 1994.

[4] Gutberlet, P.; Krämer, H.; Rosenstiel, W.: „*CASCH - a Scheduling Algorithm for High Level -Synthesis*“, Proceedings of the EDAC, pp. 311-315, February 1991.

[5] Gutberlet, P.; Rosenstiel, W.: "*Timing Preserving Interface Transformations for the Synthesis of Behavioural VHDL*", Proceedings of EURO-DAC, September 1994.

[6] Hansen, C.; Nascimento, F.; Rosenstiel, W.: „*Verifying High-Level Synthesis Results Using a Partial Order based Model*“, HLDVT‘98, La Jolla (CA), November 1998.

[7] Heinkel, U.; Glauert, W.: „*An Approach for a Dynamic Generation/Validation System for the Functional Simulation Considering Timing Constraints*“, Proceedings of ED & TC, Paris 1996.

[8] Mayer, C.; Sahm, H.; Pleickhardt, J.: "*A Graphical Data Management System for HDL-Based ASIC Design Projects*", Proceedings of EURO-DAC, September 1996.

[9] Nascimento, F.; Rosenstiel, W.: "*Partial Order Based Modeling of Concurrency at the System Level*", Proceedings of CONSYSE, September 1997.