

Information Modelling of VHDL'93

Cristian A Giumale

Department of Computer Science

Technical University of Bucharest

313 Splaiul Independentei, Bucharest 77206, Romania

Hilary J Kahn

Department of Computer Science

University of Manchester

Oxford Road, Manchester M13 9PL, UK

1 Abstract

The paper discusses issues related to the application of information modelling to VHDL'93. It is shown that an information model of VHDL can provide a uniform description of VHDL objects at different levels of design and of the behavioural semantics of the language. This enhances the application range of the model and its coverage of the language.

2 Introduction

VHDL [1] has emerged as a major language for describing hardware with descriptions being suitable for simulation, synthesis, as well as design documentation and interchange. Apart from efficient and reliable implementations, an important focus of VHDL-related research is directed towards the modelling of the language semantics. The main purpose of these research efforts is to capture in a clear and unambiguous notation what is described informally in the VHDL reference manual and to use the resulting descriptions as a basis for the formal proof of properties of VHDL descriptions, the comparison of VHDL with other HDLs and the implementation of related software tools.

Research in the field of VHDL modelling has resulted in formal models [2], data and information models [3], and implementation models [4] which exhibit the necessary operational details for language implementation. Most existing VHDL models, particularly formal models, consider restricted subsets of VHDL and even fewer address the semantic extensions of VHDL'93 [5].

Of the methods mentioned, information modelling provides a suitable basis for controlling the complexity of the model without adopting simplifying assumptions that many formal modelling methods use. However, the relatively restricted circulation and ap-

plication of the existing information models, and the relative novelty of the field has limited the general acceptance of information modelling as a useful tool with a wide range of applicability. The consideration of information modelling as a competitive alternative to other modelling methods has also suffered because, misguidedly, information modelling is seen only as a means of defining the contents of a data base rather than as a conceptual description of the modelled universe of discourse.

The discussion in this paper shows that an information model of VHDL (a) describes uniformly both the structural and the behavioural VHDL'93 semantics, and (b) spans fully the language perspectives a VHDL object can have - source description, analysis, elaboration and execution - and specifies the relationships between these perspectives.

The problems discussed in the paper result from work supported by the UK Defence Research Agency and by the ESIP (ESPRIT 8370) project to produce comprehensive information models of VHDL'87 and of VHDL'93. As the work has progressed it has become clear that a monolithic model of VHDL is not a satisfactory option. Instead, a hierarchy of models, each of which describes the relevant aspects related to a specific application and from a specific perspective of VHDL, is a more realistic solution. The model at the apex of this hierarchy describes the essential objects and the semantics of VHDL at the description, analysis, elaboration and simulation levels of design. Such a model of VHDL'93 [6] is the basis for this paper.

The plan of the paper is as follows. Section 2 is a brief informal introduction to EXPRESS [7], an information modelling language used in the work being described. Section 3 shows the way different language perspectives of a VHDL process are specified in an information model. Section 3 presents the basic idea of modelling the execution of a VHDL design. Section 4 discusses the technique used for describing the VHDL'93 process transition automaton, which can be

considered as the central part of the execution layer of the language semantics. Section 5 highlights the way behavioural relationships between process states and executed wait statements are modelled. Finally, some benefits of VHDL information models are discussed.

3 Modelling with EXPRESS

In contrast to computation models, which aim to show how values which characterise the execution of a VHDL design are obtained, an information model is intended to capture the conceptual structure of the language: the underlying objects, their relationships and constraints from the different language perspectives: source description, elaboration and execution. The main thrust of computation is to solve; the goal of information modelling is to describe.

The main construct of a VHDL information model is the description of a VHDL object. It specifies the **attributes** of the object and the **constraints** which the values of the attributes must satisfy. For example, using EXPRESS [7], the object **process**, corresponding to a **process_statement**, can be described by means of a construct called **ENTITY** as in example 1.

This partial example illustrates the main points of an object description. Each attribute has a name and a type, which is the name of another entity. It specifies that in an entity instance the value of the attribute is normally an instance of the entity which represents the type of the attribute. For example **pulse_rejection_limit** is an attribute the values of which are of type **time_expression**, itself an entity which must be fully specified in the model. The attributes in the example are: mandatory (e.g. **target**), optional (e.g. **pulse_rejection_limit**), computed (e.g. **assignments**) and inverse. In this example the inverse attribute **parent_process** certifies the membership of its containing **signal_assignment** in the **signal_assignments** set of the designated process.

A constraint specifies either a local condition on the values of an entity attribute or a relationship between the values of the attributes belonging to the same entity or to different entities. It is a logical expression which must not evaluate to false for each valid instance of the containing entity. For example, the constraint **valid_pulse_rejection_limit** states that the

pulse rejection limit can be specified only if the delay mode of the signal assignment is inertial. Predefined or user defined functions can be called within constraint expressions. In the example above **EXISTS** is a predefined function which checks if an optional attribute is present in an **ENTITY** instance.

Entities can be structured into hierarchies of supertypes-subtypes which can be used for classification purposes and for attribute and constraint inheritance. Entities are grouped to form a **SCHEMA**, which is a sub-model of a specific part of the universe of discourse. Entities from one schema can be used in other schemas. In this way a complete model can be partitioned into smaller and conceptually consistent parts.

The model in Example 1 can be represented diagrammatically by using a graphical notation called EXPRESS-G, as illustrated in Figure 1. The diagram consists of symbols with different formats for different EXPRESS constructs such as entities and types. For example, an entity is represented by a solid rectangle which is starred if the entity contains constraints. The symbols are connected by lines: thin lines connect attributes to their containing entities; thick lines indicate subtype/supertype relationships. Attribute optionality is represented by dashed lines. The directionality of the relationships is indicated by a circle at the end of a line. Additional information is displayed on thin lines indicating the nature of the attribute, e.g. DER stands for a computed attribute.

4 Modelling perspectives

A comprehensive core model must describe the basic VHDL objects at different levels of the language: source design description (which corresponds to a source VHDL program), design analysis (which focuses on library units and libraries), design elaboration (which transforms an analysed VHDL description into a network of processes controlled by signals), and design simulation (which highlights the time-dependent behaviour of an elaborated VHDL description). There is a sub-model corresponding to each VHDL level.

The **process** entity in Example 1 illustrates object modelling at the level of source design description. However other perspectives of a process address its elaboration and simulation. Whereas the modelling of an elaborated process still focuses on structural

```

SCHEMA design_description_schema; ...
ENTITY process
  ABSTRACT SUPERTYPE OF (ONEOF(postponed_process,
                                non_postponed_process));
  ...
  assign_statements: OPTIONAL SET [1:?] OF signal_assignment;
  wait_statements:    SET [1:?] OF wait_statement;
DERIVE
  assignments: SET OF signal_assignment:=
    NVL(assign_assignments, []);
END_ENTITY;

ENTITY signal_assignment;
  pulse_rejection_limit: OPTIONAL time_expression;
  delay_mode:            assignment_delay;
  waveform_elements:    LIST [1:?] OF waveform_element;
  target:               signal;
INVERSE
  parent_process: process FOR signal_assignments;
WHERE
  valid_pulse_rejection_limit:
  NOT EXISTS(pulse_rejection_limit) OR
  (EXISTS(pulse_rejection_limit) AND (delay_mode=inertial_delay));
  ...
END_ENTITY;
END_SCHEMA;

```

Example 1: The partial model of a process

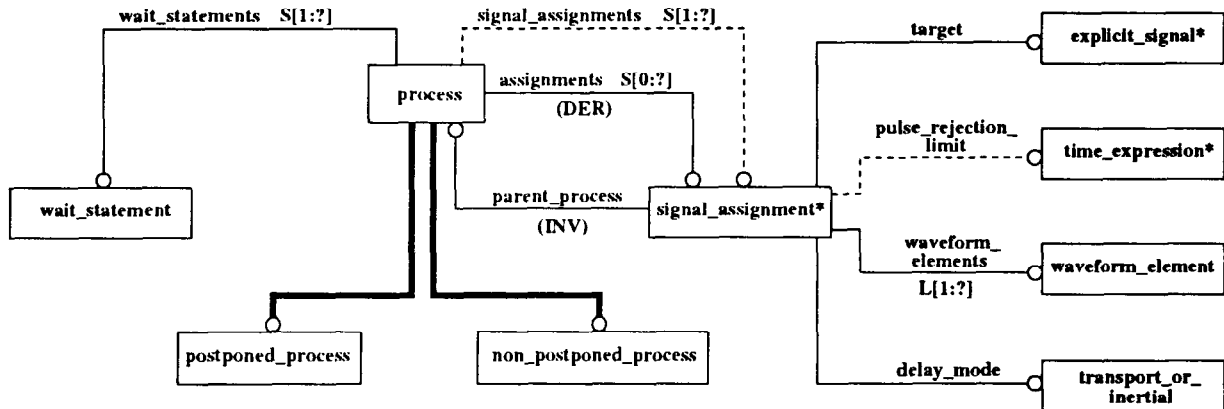


Figure 1: The diagrammatic representation of the process model

properties and on the way these are interrelated with the source description of the process, the model of process execution describes the time dependent behaviour of the process as a member of a simulated design.

A simplified model of an elaborated process and signal assignment is illustrated in Figure 2. It shows that the elaboration of signal assignments and processes must preserve the isomorphic relationship between the source design description and the corresponding elaborated design. The model of process execution is described in the next sections of the pa-

per.

In addition to the description of relationships between objects from different design levels, the structuring of the model according to the levels of VHDL makes it possible to trace information. This is essential if a VHDL model is to provide an implementation basis for different applications of the language. The tracking of information links a specific result of the application program to those parts of the source design description which are relevant to the result.

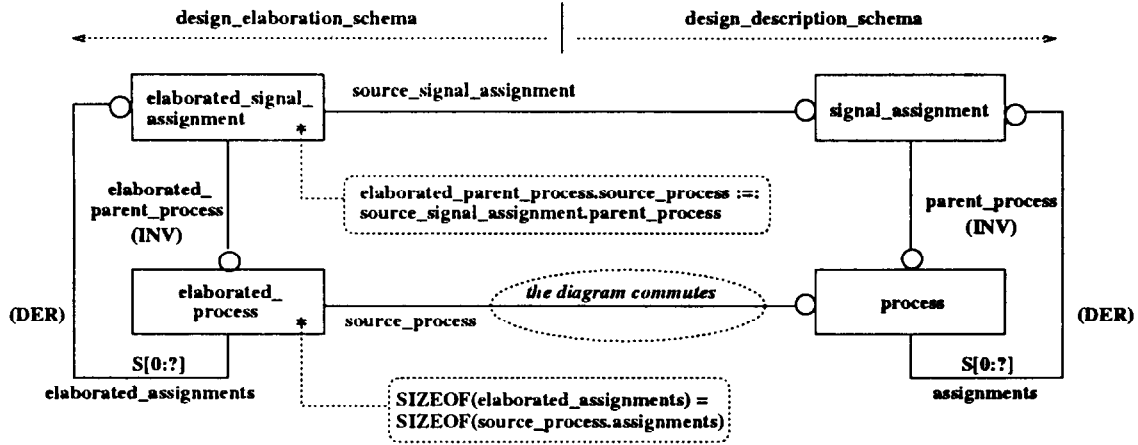


Figure 2: The partial model of an elaborated signal assignment

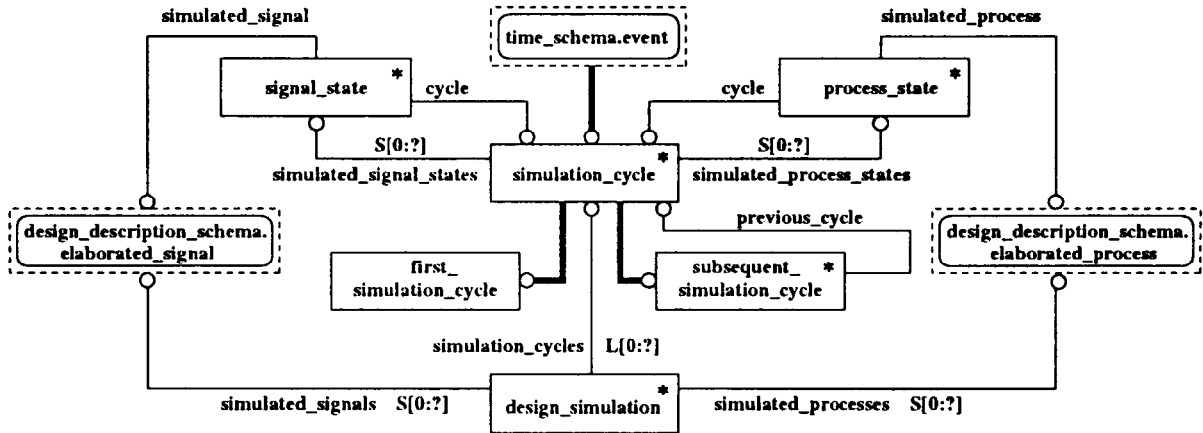


Figure 3: The general model of the simulation process

5 Execution Modelling

The VHDL time dependent semantics are described using an abstract event-driven simulation model which specifies the way the language objects are evaluated while the simulation process unfolds. The general model of the simulation process is illustrated in Figure 3. The attribute `simulation_cycles` of the entity `design_simulation` implicitly specifies the simulation function as a list of time ordered points (`previous simulation cycle`, `current simulation cycle`), where the entity `simulation_cycle` contains as attributes the set of process and signal states corresponding to the cycle. The ordering of `simulation_cycles` relies on a model of time [8] which enables the ordering of simulation cycles separated by zero delays (`delta cycles`), as discussed in [9].

The points of the simulation function must correspond to valid transitions between the process states. Therefore, additional properties are specified to further constrain the points of the simulation function. These 'behavioural' constraints show how process states relate to signal states via a hierarchy of other entities which include: executed wait statements, executed signal assignments, signal drivers and transactions. These represent the result of executing elaborated VHDL statements which correspond to statements from the source description of a VHDL design. The model specifies these relationships directly. Hence it is possible to relate behavioural properties of an executed design to structural properties of the source design description.

It should be noted that the 'behavioural' constraints

do not describe the simulation framework of VHDL, e.g. the kernel process and the different phases of a hypothetical VHDL simulator. Instead they make sure that the functional relationships, with regard to time, between different simulation events are correct. From this point of view the level of abstraction of the information model is higher than the level of abstraction of other VHDL models which describe the behavioural semantics of VHDL indirectly, by considering the semantics of a hypothetical simulation machine [10].

6 Modelling process state transitions

The model differentiates between **complete** and **partial** process states. Whereas a complete state fully describes the behaviour of a process in a given simulation cycle, a partial state specifies properties common to several complete states. There are two reasons for expressing the complete states as symbol sequences ('words') $(x_1 \dots x_n)$, $x_i \in P$, over a set P , of partial states.

First, partial states accurately describe the different phases of process behaviour. For example, a postponed process is active in a given simulation cycle if it is resumed and executed in that cycle or if it is executed after its resumption at the end of a sequence of delta cycles. The complete state **active_process** can be seen as a 'word' over the set of partial states **resumed** and **active**: **active_process** ::= (**resumed**, **active**) | (**active**). Second, the process transition automaton can be specified as a simpler constrained hierarchy of partial states.

In a simulation cycle **C** an **elaborated_process** - postponed or non postponed - can be in one of the following partial states.

- **B (blocked_process_state)** corresponds to a suspended process which stays suspended in the given cycle **C**.
Property: the process is blocked by executing a wait statement. This event is identified by the attribute **wait_for** of the entity **blocked_process_state**.
- **R (resumed_process_state)** corresponds to a resumed process which may or may not be immediately executed according to its type, postponed or non-postponed.
Property: if **C** is the first simulation cycle the

process is automatically resumed. If **C** is a subsequent simulation cycle, the resumption is the effect of satisfying a wait statement. This event is identified by the attribute **unblocked_wait** of the entity **resumed_process_state**.

- **A (active_process_state)** corresponds to a process which executes in the given cycle **C**.
Property: the process is suspended by a wait statement. This event is identified by the attribute **wait_for** of the entity **active_process_state**. In addition, the execution of the process may result in a non empty set of executed signal assignments. This effect of process execution is designated by the attribute **assignments** of the entity **active_process_state**.
- **L (in_limbo_process_state)** which corresponds to a process which will be executed in a cycle **C'** which occurs after **C** and which is the last delta cycle of the sequence of simulation delta cycles which contains **C**.
Property: the process is a postponed process and the cycle **C** is a delta cycle.

The complete states of a process are composite states as follows.

- **(B)** - the process is already suspended and remains suspended.
- **(RA)** - the process is resumed and immediately executed.
- **(A)** - the process (postponed) is executed after its resumption, at the end of a sequence of simulation delta cycles.
- **(RL)** - the process (postponed) is resumed and set "in limbo", waiting for a cycle which is not a simulation delta cycle.
- **(L)** - the process (postponed) has been resumed in a former simulation cycle and is still "in limbo".

Let $S = [(A), (RA), (L), (RL), (B)]$ designate the set of all complete states of a process and assume that the following supertypes are defined:

R supertype_of:	(RA) OR (RL)	- process is resumed
A supertype_of:	(RA) OR (A)	- process is active
L supertype_of:	(RL) OR (L)	- process is in limbo
B supertype_of:	(B)	- process is blocked

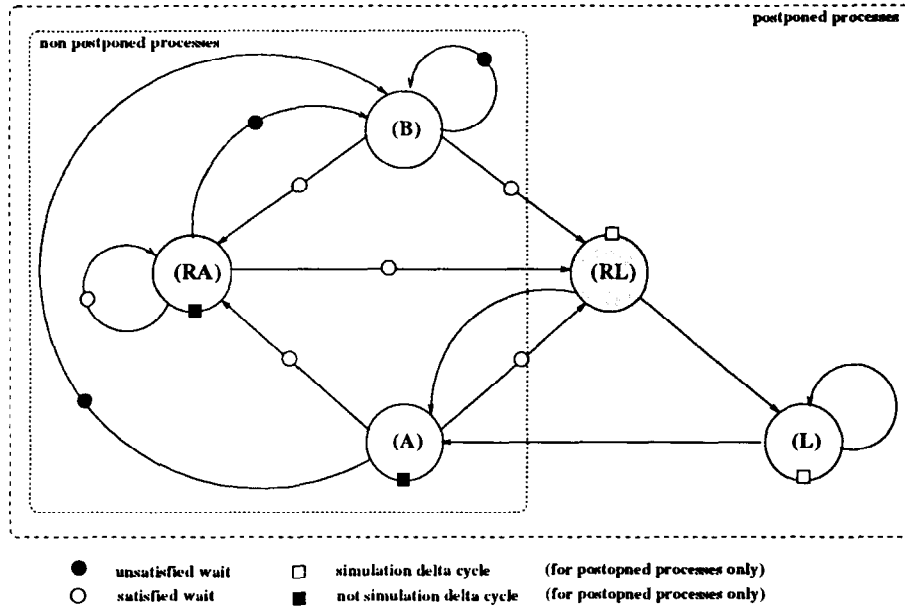


Figure 5: Process state transitions

```

ENTITY subsequent_active_process_state
  SUBTYPE OF(active_process_state);
DERIVE
  previous_state: process_state:=
    find_process_state(simulated_process,cycle.previous_cycle);
WHERE
  valid_simulation_cycle:
    NOT is_start_cycle(cycle);
  valid_process:
    is_type(simulated_process.source_process,'postponed_process');
  valid_process_state:
    NOT is_type(SELF\process_state,'resumed_process_state');
  valid_transition:
    is_type(previous_state,'in_limbo_process_state');
  valid_postponed_execution:
    NOT is_delta_cycle(cycle);
END_ENTITY;

```

Example 3: Constraints of a process state

that the elaborated wait statement which blocks or resumes a process in a given subsequent simulation cycle is identical to the elaborated wait statement which suspended or kept blocked the process in the previous cycle. This constraint is associated with the entities which model the states (R) and (B).

7 Modelling behavioural relationships

The B, A, R process states contain attributes such as **wait_for** or **unblocked_wait** which describe outcomes of executing an elaborated wait statement:

satisfying or not satisfying the statement. The process execution model is completed by the explicit description of these outcomes.

The conditions for satisfying or not satisfying an elaborated wait statement **W** depend on the structure of **W**, which can contain signal and/or condition and/or timeout wait clauses, and on conditions of satisfying or not satisfying these clauses. For example, the entity **satisfied_wait** designates the result of executing a satisfied elaborated wait statement **W**. It shows that **W** is satisfied if at least one of its executed clauses is satisfied.

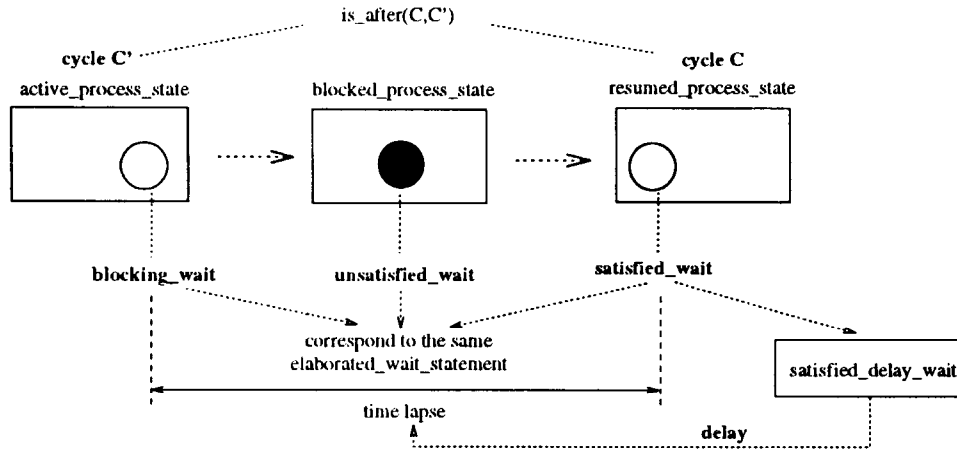


Figure 6: Executing a satisfied timeout wait clause

Hence, the condition of satisfying W is reduced to the condition of satisfying a clause of W . Assuming that W has a timeout clause, and that it is executed in the cycle C by the process P , the timeout clause is satisfied (active) if:

- There is an active process state $A(P,C',W)$ of the process P which is suspended in cycle C' by executing the elaborated wait statement W .
- The simulation cycle C occurs after C' .
- All process states corresponding to process P and which occur after the simulation cycle C' and before C are blocked.
- The time lapse between C and C' equals the waiting delay of the timeout clause of W .

The above conditions form the basis of constraints specified in the entity `satisfied_delay_wait`, a subtype of `satisfied_wait` which is an attribute of a `resumed_process_state`. They are schematically illustrated in Figure 6.

8 Conclusions

A major advantage of a VHDL information model is its stratification according to the level of abstraction and to the roles of the different objects and concepts of the language. The paper has also shown that apart from static, structural properties, the dynamic aspects of the language can be appropriately modelled.

These properties enable information tracking from the simulation level to the design description level of the model. This makes it possible to use an information model for not only for the verification of an abstract design, as many formal models do, but for a real design. In addition, the kind of verification which can be performed is varied ranging from structural properties of a source design description to the design simulation.

Several models can exist for VHDL according to the underlying purpose of the model. In particular, a core model, seen as the root of a hierarchy of specialised models, can be used to enhance language standardisation by clarifying ambiguities and, in addition, as the basis for the conceptual comparison of existing or proposed versions of VHDL.

9 Acknowledgements

The authors wish to acknowledge the support of the Defence Research Agency and the Commission of the European Communities through the ESIP (ESPRIT 8370) project. They are also grateful for the reviews of the VHDL information models by Cleland Newton, Andy Carpenter, Serafin Olcoz and Juana Lopez, as well as other members of the VHDL community.

References

- [1] VHDL Language Reference Manual (IEEE Std 1076-1987/1993). IEEE, Inc., 345 East 47th Street,

New York, NY 10017, USA, 1988/1993

- [2] Schaefer L., Mueller W. and Wilkes W. Examination of Concepts in Existing Modelling Languages / Methodologies to Model Behavioural Semantics. Deliverable Report ECIP2/HU/008-1, 1992
- [3] Giumale C. A. An Information Model of VHDL'87 (Draft Version 10). University of Manchester, October 1994
- [4] VIFASG 1076 VHDL Procedural Interface (Draft Version), VHDL Schema Definition (Draft Version). VIFASG Subgroup on Intermediate Form Definition, November 23, December 11, 1990
- [5] Berge J.M., Fonkoua A., Maginot S. and Rouillard J. VHDL: VHDL'92. Kluwer Academic Publishers, 1993
- [6] Giumale C. A. An Information Model of VHDL'93 (Draft Version 01). University of Manchester, December 1994
- [7] EXPRESS Language Reference Manual. ISO 10303: Part 11 Version N14, April 1991
- [8] Giumale C. A. and Kahn H.J. An Information Model of Time. Proc. IEEE/ACM Design Automation Conference, Dallas 14-18 June 1993, pp 668-672
- [9] Olcoz S. and Colom J.M. The Discrete Event Simulation Semantics of VHDL. Proceedings of the International Conference of Simulation and Hardware Description Languages, Tempe, Arizona, January 1994, pp 128-134
- [10] Borger E., Glasser U. and Wolfgang M. The Semantics of Behavioural VHDL'93 Descriptions. Proceedings EURO-DAC'94/EURO-VHDL'94, Grenoble, September 19-23, IEEE Press 1994, pp 500-505