A Core Information Model of VHDL

Cristian A. Giumale

Department of Computer Science
Bucharest Technical University
Bucharest 77206, Romania

Hilary J. Kahn

Department of Computer Science
University of Manchester
Manchester M13 9PL, UK

Abstract

The paper discusses issues related to the application of information modelling to VHDL. It is shown that an information model of VHDL can provide a coherent and uniform description of: the VHDL objects at different levels of design, the interrelationships between the design levels, and the behavioural semantics of the language. A hierarchy of VHDL information models can exist which encompasses the range from abstraction to detail. The 'core' model at the apex of this hierarchy is the base for the paper.

1 Introduction

An important domain of research in the field of electronic design (ECAD) is that of modelling. The aim is to build clear and unambiguous descriptions of the semantics of various languages and artifacts. Of the many possible modelling methods, information modelling has received particular attention in recent years and there are ECAD languages accompanied by information models [3] [2]. However, the relatively restricted circulation and application of these early 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 (**UoD**).

This paper tries to show that (a) an information model of VHDL [10] can integrate different perspectives of VHDL objects, both static and time dependent, directly and uniformly; (b) the model can describe the mappings between these perspectives; (c) there can be several VHDL information models, depending on the goal of the model, which can be organised into hierarchies for better support of applications. These features, taken together, can seldom be found in other kinds of model such as formal models [9] which mainly describe the functional aspects of VHDL, including its behavioural semantics, or implementation models [11] which concentrate on the representation of the language objects as data structures.

The problems discussed relate to work supported by the UK Defence Research Agency and by the ESIP (ESPRIT 8370) project which aim to produce a comprehensive information model of VHDL'87 and of VHDL'93. The VHDL'87 model [6] which is the basis for this paper, identified here as the "core model", describes the essential objects and the semantics of VHDL at the design description, analysis, elaboration and simulation levels of the language. The problems discussed apply as well to the core model of VHDL'93.

2 Information modelling

There are several levels of using computers to tackle a problem from a given **UoD**. At the conventional level the underlying concepts of the **UoD**, their relationships and constraints, are implicitly used in order to build a description, in particular a program, which, when executed, solves the problem. The aim is to show how values which characterise the **UoD** according to the given problem are obtained. At a higher level, the conceptual structure of the **UoD** is explicitly described and then used to assist the problem solving process. The first level is characteristic of computation whereas the second relates to the field of conceptual modelling, in particular to information modelling.

For example, a program written in VHDL is a computational model of the electronic circuit it describes. When executed, the model behaves as the described circuit. Alternatively, an information model of VHDL itself, written in a specific information modelling language, can be used to explain the way any given VHDL model behaves. A model of VHDL can be considered as the model of a generic electronic circuit seen through VHDL glasses.

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 the modelling language EXPRESS [4], the object **explicit_signal**, which designates an explicitly declared signal, can be partially de-

```
SCHEMA design_description_schema;
(*
   Submodel of VHDL which groups models of VHDL objects
   used at the level of source design description.
*)
REFERENCE FROM vhdl_type_schema;
...
ENTITY signal
       ABSTRACT SUPERTYPE OF(ONEOF(explicit_signal,...);
   signal_type: VHDL_type;
END_ENTITY;

ENTITY explicit_signal
       ABSTRACT SUPERTYPE OF
              (ONEOF(port,internal_signal)           AND
               ONEOF(scalar_signal,composite_signal) AND
               ONEOF(signal_subelement,complete_signal))
       SUBTYPE OF(signal);

   -- attributes
   signal_kind:         OPTIONAL bus_or_register;
   disconnection_delay: OPTIONAL time_expression;
   resolved_by:         OPTIONAL resolution_function;

   -- constraints
WHERE
   valid_disconnection_delay:
   -- Only guarded signals can have disconnection delays
   EXISTS(signal_kind)OR NOT EXISTS(disconnection_delay);
   ...
END_ENTITY;
END_SCHEMA;
```

Example 1: The partial model of a signal

scribed as shown in example 1. Each attribute has a name and a type, which is the name of another object. It specifies that, in an object instance, the value of the attribute is normally an instance of the object which represents the type of the attribute. For example **disconnection_delay** is an attribute the values of which are of type **time_expression**, itself an object which must be fully specified in the model. The attributes in the example are: mandatory (e.g. **signal_type**, inherited from the **signal** object) and optional (e.g. **disconnection_delay**).

A constraint specifies either a local condition on the values of an object attribute or a relationship between the values of the attributes belonging to the same object or to different objects. It is a logical expression which must not evaluate to false for each valid instance of the containing object. For example, the constraint **valid_disconnection_delay** states that each signal (in reality each instance of the **explicit_signal**) can be associated with a disconnection delay only if it is a guarded signal, i.e. if the attribute **signal_kind** is explicitly specified.

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

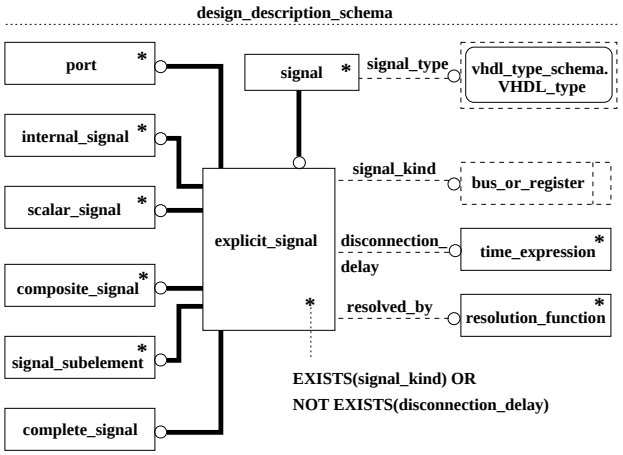

Figure 1: The diagrammatic representation of the signal model

The signal model can be represented diagrammatically by using a graphical notation. The diagram in figure 1 consists of symbols with different formats for different EXPRESS constructs. For example, an object description is represented by a solid rectangle which is starred if the description contains constraints. The symbols are connected by lines: thin lines connect attributes to their containing objects; 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.

3 The modelling focus

Building a model which focusses on the essential objects and semantics of the language makes it necessary to classify VHDL objects into **essential** and **derivable**. Intuitively, an essential object cannot be described in terms of other objects whereas a derivable object can. This task can identify syntactic constructs which are derivable from basic objects and, in contrast, can uncover objects which, conceptually, exist in the language although they have no syntactic equivalent.

For example, the concurrent signal assignment is not an essential concept. It can be described using more basic concepts: process, (sequential) signal assignment, and wait statement [7]. The conversion of a concurrent signal assignment is as shown in figure 2. A **core** model of VHDL need not contain an explicit sub-model of the concurrent signal assignment. Instead, the boxed code sections which occur in the descriptions above must be modelled. The transformation shows that the object **signal_ assign-**

```
guarded_concurrent_assignment: process
begin
  if GUARD then  target <= waveform;
  else           target <= null after disconnection_delay;
                 -- only if the target is a guarded signal
  end_if;                                                        (a)
   wait on (GUARD + signals-from-waveform);
end process;

non_guarded_concurrent_assignment: process
begin
  target <= waveform;                                            (b)
   wait on (signals-from-waveform);
end process;
```

Figure 2: Basic signal assignments

ment, which describes a sequential signal assignment, acts as a supertype of two more specific and conceptually essential signal assignments:

- **non_guarded_signal_assignment**, corresponding to a sequential signal assignment;
- **guarded_signal_assignment**, corresponding to the boxed control structure (a) and which can be taken as a basis for describing the behavioural relationships which interrelate the assignment target signal, its disconnection delay and the corresponding GUARD signal.

Only the **non_guarded_signal_assignment** has a syntactic equivalent in the language, although both assignments are conceptually essential.

Deciding what is essential in a **UoD** and what is not essential also relies on the role played by the objects of the given **UoD**. There can be objects which play syntactic roles, concepts which are merely programming commodities, etc. For instance, in VHDL, a disconnection delay expression is associated with a signal by means of a disconnection specification. The association mechanism is not important for the purpose of the core model. What is fundamental is the fact that a guarded signal can have a disconnection delay. Therefore, it is sufficient to add the disconnection delay as an attribute of the object **explicit_signal**.

In the discussion above, objects were considered essential and derivable based on the purpose of the model which is to describe fully the semantics of the VHDL language, ignoring syntactic and computational details. However, if the purpose of the model changes, for example if the model is to be used to implement VHDL, then objects such as concurrent signal assignment or subprogram have to be considered essential. This suggests that there can be several models, more or less complete and with different levels of abstraction, for the same version of the language but aiming to satisfy different goals. These models can be organised to form a hierarchy. The model at the apex of this hierarchy can be seen as a **core** model covering only the basic concepts of VHDL. A model in the hierarchy is a specialisation of the core model, covering additional, non basic, concepts of the language, ignoring non-relevant concepts, and possibly containing transformations of the contents of the core model.

4 Modelling objectives

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 focusses 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 **explicit_signal** object in figure 1 illustrates object modelling at the level of source design description. The model shows that a signal can be classified according to: (a) its role :- port or internal_signal (declared in an entity declaration, package declaration, block or architecture), (b) its structure :- scalar or composite, and (c) to signal membership :- subelement of another signal or stand alone signal. Therefore, an **explicit_signal** is an aggregate of these three perspectives.

A major goal of the core model is to describe the relationships between objects from different design levels. A typical case is that of describing properties of the algorithmic processes which are used to generate or transform objects from one design level into objects of another design level.

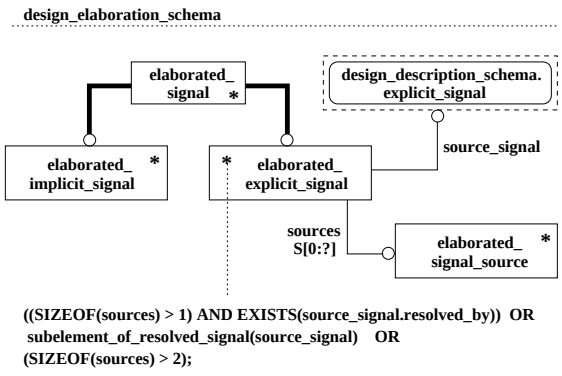

Figure 3: The partial model of an elaborated signal

The model **elaborated_explicit_signal** from figure 3 describes the result of elaborating a source **explicit_signal**. The **elaborated_explicit_signal** is contained in the **design elaboration schema**, which is the sub-model corresponding to the VHDL design elaboration level. The model shows that the nature of the signal, e.g. whether it is a **port** or an **internal_signal**, is no longer important at the elaboration level where a design is seen as a flat structure containing processes connected by a network of signals [8]. Instead, an **elaborated_explicit_signal** is characterised by a set of sources which are used to compute the values of the elaborated signal while the simulation process unfolds.

In addition, the model shows that an elaborated signal which has multiple sources and is not a sub-element of a resolved signal of a composite type must be elaborated from a **source_signal** which is associated with a resolution function (Std 1076-1987 page 2-7,4-6).

5 Modelling of time-dependent behaviour

An important part of VHDL semantics corresponds to the behaviour of a given design during simulation and, therefore, the VHDL model must also focus on relationships between signals and processes as a function of time. The general model illustrated in figure 4 suggests how VHDL behavioural relationships can be modelled.

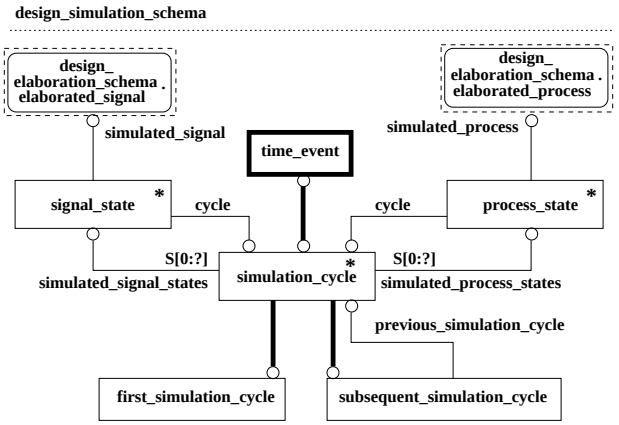

Figure 4: The general model of the simulation process

The attribute **simulation_cycles** of the abstract object **design_simulation** implicitly specifies the simulation function as a set of points (previous simulation cycle, current simulation cycle),where the object **simulation_cycle** contains as attributes the set of process and signal states corresponding to the cycle. 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 objects which include: executed wait statements, executed signal assignments, signal drivers and transactions.

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 [1].

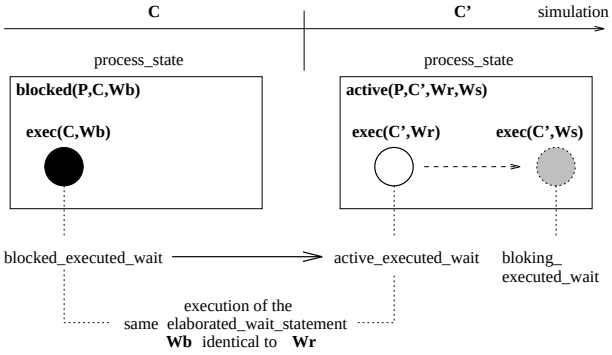

Figure 5: The process state transition blocked-active

For example consider the transition of a process from a suspended (blocked) state to an active state, as illustrated in figure 5. Let **blocked(P,C,Wb)** be the process state corresponding to the elaborated process **P** which is kept suspended by executing the elaborated wait statement **Wb** in the simulation cycle **C**. Also, let **active(P,C',Wr,Ws)** designate an active state of the process **P** which is resumed in cycle **C'** as the result of satisfying the wait statement **Wr** and then suspended by **Ws**. In addition, assume that **exec(C,W)** is the result of executing the elaborated wait statement **W** in the simulation cycle **C**. By convention, **exec(C,W)** is said to be satisfied if the waiting conditions of **W** are satisfied. The transition **blocked(P,C,Wb) -¿ active(P,C',Wr,Ws)** is correct if:

- **C** is not the first simulation cycle;
- **C'** is the immediate successor of **C**;
- The elaborated wait statement **Wb** is identical to **Wr**;
- **exec(C,Wb)** is not satisfied;
- **exec(C',Wr)** is satisfied.

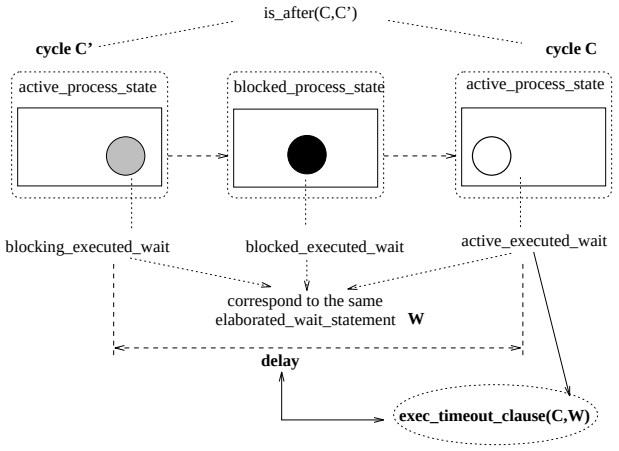

Figure 6: Executing an active timeout clause of a wait statement

The above constraints are part of the **process_state** object and rely on structural information available from the elaboration and design description levels of the design. For instance, the conditions for satisfying or not satisfying (activating or blocking) the **exec(C,W)** depend on the structure of **W**, which can contain signal and/or condition and/or timeout wait clauses. These conditions can be described in the model by hierarchies of concepts and constraints.

For example, the constraints of an active or blocked **exec(C,W)** can be modelled by taking into account that:

- The **exec(C,W)** is satisfied if at least one of its executed clauses is satisfied.
- The **exec(C,W)** is blocked if all of its executed clauses are unsatisfied.

Hence, the condition of satisfying a generic **exec(C,W)** is reduced to the condition of satisfying a generic clause of **W**. Assuming that **W** has a timeout clause, this is satisfied (active) if:

- There is a process state **active(P,C',Wr,W)** which is suspended due to 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 by the same elaborated wait statement **W**.
- 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 object **active_delay _wait**, a subtype of **active_executed_wait** which is an attribute of a **process_state**. They are schematically illustrated in figure 6.

The task of modelling the VHDL behavioural semantics also relies on a model of time [5] which enables the ordering of simulation cycles separated by zero delays, as discussed in [8].

6 Conclusions

The view of information modelling as discussed in this paper is somewhat different from the usual perspective which considers an information model as the contents of a data base. The stress on modelling has been on conceptual issues. There are important consequences of this idea.

A core model, seen as the root of a hierarchy of specialised models, can be used to enhance the the VHDL language standard by clarifying ambiguous aspects and, in addition, as a conceptual comparison base between the already existing versions of VHDL. For instance, a core information model of VHDL could be used to help decide on language alterations and the extent to which these conceptually affect the language. In this respect a core model makes it easier to judge to what extent the proposed extensions maintain upward compatibility of design descriptions. In the limit, language extensions could be performed on the model which could then be used to generate the modified language.

The analysis of the compliance of different implementations, including for different versions of the language, can also rely on information models. Moreover, a core model can be used as a basis for easier inter-operability comparison between VHDL and other HDLs or languages for electronic information interchange. From this point of view designing specific parts of ECAD application frameworks can also rely on information models. For example, the models of VHDL and EDIF can be compared to determine the set of equivalent entities and relationships which are relevant for designing tools which can operate on both representations.

The observations above show that a hierarchy of information models of VHDL is a worthwhile alternative to other kinds of models. It can span from the abstract level to the implementation level of VHDL as required for a given application.

Acknowledgement

The authors wish to acknowledge the support of the Defence Research Agency (DRA), UK 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 of DRA, Andy Carpenter of The University of Manchester, and Serafin Olcoz and Juana Lopez of TGI, as well as other members of the VHDL community.

References

[1] 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

[2] *Standards for Electronic Design Automation, Release 1.0*, CAD Framework Initiative, Inc., 4030 W.Braker Lane, Suite 550, Austin, Texas 78759 U.S.A., 1992

[3] *Electronic Design Interchange Format Version 300*, Electronic Industries Association / EDIF Division, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006, U.S.A., EIA-618, 1993

[4] *EXPRESS Language Reference Manual*, ISO 10303: Part 11 Version N14, April 1991

[5] Giumale C. A. and Kahn H.J. "An Information Model of Time", *Proceedings IEEE/ACM Design Automation Conference*, Dallas 14-18 June 1993, pp 668-672

[6] Giumale C. A. *An Information Model of VHDL'87 (Draft Version 5) / VHDL'93 (Draft Version 1)*, University of Manchester, August/November 1994

[7] Lipsett R., Schaefer C. and Ussery C. *VHDL: Hardware Description and Design*, Kluwer Academic Publishers, 1989

[8] 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

[9] Schaefers 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

[10] *VHDL Language Reference Manual (IEEE Std 1076-1987/1993)*, IEEE, Inc., 345 East 47th Street, New York, NY 10017, USA, 1988/1993

[11] *VIFASG 1076 VHDL Procedural Interface (Draft Version), VHDL Schema Definition (Draft Version)*, VIFASG Subgroup on Intermediate Form Definition, November 23, December 11, 1990