On the Formal Semantics of a CHDL - A Case Study

Ulrich Bieker
University of Dortmund, Department of Computer Science
Postfach 500500, W-4600 Dortmund 50
e-mail: bieker@ls12.informatik.uni-dortmund.de

1. Abstract

The semantics of HDL descriptions influences all facets of VLSI design such as synthesis, test, verification, logic simulation and fault simulation. In this paper formal semantics of the intermediate language TREEMOLA, used in the MIMOLA hardware design system MSS, is presented. In particular semantics of module declarations, described at the Register Transfer Level by the CHDL MIMOLA, is defined.

2. Introduction

Usually the semantics of a hardware description language is either implicitly given by a simulator or is in the mind of the designer of the language. Therefore, a formal semantics definition is of great importance for every user of the language. The work presented here is intended to fill this gap in the context of MIMOLA. Principles of defining formal semantics are commonly known, but it is interesting how this is done for an already existing HDL in detail.

MIMOLA (**m**achine **i**ndependent **m**icr**o**programming **la**nguage) [BMSJ91] is a computer hardware description language (CHDL) which has been influenced by other hardware description languages like VHDL [IEEE88] and DACAPO [DOS87]. TREEMOLA (**tree** **m**icro **o**peration **la**nguage) [Bec91] is the language that is used to exchange design data between different CAD-tools in the MSS [Kel87], [Mar90].

Using first order predicate calculus [Scho89] a formal semantics definition is obtained for a subset of the intermediate language TREEMOLA. With this subset it is possible to describe a hardware at the RT - level, i.e. to describe the behavior of certain modules such as registers and gates up to ALU’s and their interconnections. Based on the principles of the semantics described below, a simulator for this particular level has been implemented in Prolog [ClMe81]. Derivation of a logic program from a correct logic description has led to the simulator for the hardware description language MIMOLA. A detailed description of the techniques concerning derivation of logic programs is given in [Devi90]. The idea of formulating this semantics was inspired by the book *Verifikation digitaler Systeme* written by Hans Eveking [Eve91].

In what follows we first show the basic methodology of the semantics and due to the lack of space we discuss a very small but illustrating example. The complete semantics definition is given in [Biek92].

3. Methodology of the Semantics

A TREEMOLA RTL - description is a netlist of modules. Modules, ranging in size from gates and ALU’s up to complete finite state machines, are described in terms of their interface and their behavior. Such a module is called a *TREEMOLA unit module description* (Fig. 1). A unit module description is a tree $r(s_1,...,s_n)$, with root r and n subtrees (sons) $s_1,...,s_n$. $\{s_1,...,s_n\}$ = **SONS(r)** is the set of sons of r. Each s_i can be e.g. a statement, an expression or a port description. The semantics described here is defined by a predicate **I** which determines whether a sequence of states in a given time interval T_{min} to T_{max} and for a given unit module description is a feasible sequence or not. Therefore, timing behavior has to be considered. To decide whether the behavior of a circuit is a feasible sequence (**I** = TRUE) or not (**I** = FALSE), it must be possible to access values of signals, memories and registers. Therefore it is assumed that there is a large 3-dimensional table **SQ** (Signals and States), in which an arbitrary number of values of signals, registers and memories can be stored as **bitstrings**. The function f is able to access this table. f needs parameters <identifier>, <address>, <time> and SQ to return the value of an identifier with a given address at a definite time. Additionally, function f requires a bit number i to select bit i of the considered bitstring. Possible values of one bit of a bitstring are {0, 1, X, Z}. X denotes an undefined value and Z a high impedance (tristate) value. If no address is available (e.g. for registers and signals), the default address is taken as 0. **SQ** may be partial predefined and **I** decides whether a given circuit is able to realize this sequence of signals and states.

Given a 3-dimensional table **SQ**, a string <identifier> of arbitrary length, an address <address> $\in N_0$, a time <time> $\in N_0$ and a bit number i $\in N_0$; functionality of f is defined as:

$$f(SQ, \langle identifier \rangle, \langle address \rangle, \langle time \rangle, i) \rightarrow \{0, 1, X, Z\}$$

Expressions, also represented as trees, are evaluated in a similar manner. An expression a+b is represented as a tree +(a,b) with root (operator) + and two sons (arguments) a, b. For an expression tree $e(s_1,...,s_n)$ with the root e and the n arguments $s_1,..., s_n$, the function exp is able to return the value of the expression at time t. It is possible that every s_i is an entire subtree or that the set of sons of s_i is empty, i.e. s_i is a leaf. Function exp requires a bit number i to select bit i of the considered expression.

Given a 3-dimensional table SQ, an expression $e(s_1, ..., s_n)$ with $n \in N_0$, a time <time> $\in N_0$ and a bit number i $\in N_0$; functionality of exp is defined as:

$$exp\ (SQ, e(s_1, \ldots, s_n), \langle time \rangle, i) \rightarrow \{0, 1, X, Z\}$$

Given a circuit description as tree $r(s_1, ..., s_n)$ and a time interval $[T_{min}, T_{max}]$ ($T_{min}, T_{max} \in N_0$ and $T_{min} \leq T_{max}$) and the 3-dimensional table SQ described above; functionality of the interpretation predicate **I** is defined as:

$$I(r(s_1, \ldots, s_n),\ [T_{min}, T_{max}], SQ) \rightarrow \{TRUE, FALSE\}$$

$T_{min} = T_{max}$ is possible only if the considered tree has a delay of 0, otherwise the interpretation predicate **I** fails. Starting with root r, the entire tree is traversed by the interpretation predicate **I**. The complete semantics, including constructs like IF, CASE, LOAD, READ and AT, is defined by such interpretation predicates. As case study an example module which is basically a clock, is discussed.

4. Detailed Semantics of a Simple Component

```
MODULE clck(OUT OC : (0));
  BEHAVIOR timer IS
  CONBEGIN
     OC <- TOGGLE UP AFTER 10 DOWN AFTER 1
  CONEND;
```

```
UCLCK
  iCLCK
     SOUT,OC@1(0)
  oTIMER,CLCK
     u
        :OUTPUT,OC@1(0)"u,l=10,10""d,l=1,1"
           .TOGGLE(0)
```

Fig. 1: *A MIMOLA module description and the corresponding TREEMOLA structure*

Fig. 1 shows on the left side the MIMOLA description of a clock with an output port OC of width 1. Different pull-up and pull-down delay times are specified in the behavior assignment. Because a start delay and an initialization are missing, the signal OC starts by default with a low value at time 0.

Every line of the TREEMOLA structure on the right side is a node of a tree. Nodes starting in the same column are brothers (e.g. *iCLCK* and *oTIMER,CLCK*). Indented lines are sons of the previous lines. Now, a short explanation for every line of the TREEMOLA structure is given. The first character of every line represents the type of the node (e.g. *U* of the root *UCLCK* is an abbreviation for *unit type*, which specifies the beginning of a definition of a primary design unit with the identifier *CLCK*). In the second line *iCLCK*, character *i* denotes the interface of the module *CLCK*, which can consist of several ports. The *S* of the line *SOUT,OC@1(0)* is an abbreviation for *signal type,* which specifies the OUTput port *OC* with a width of one bit. Line 4 is the beginning of the behavior description (called *TIMER*) of the module *CLCK*. Character *u* denotes the beginning of a concurrent statement. Line 6 describes an interface assignment. *OUTPUT OC* is the destination of the expression son *.TOGGLE(0)* in the last line. The remaining part of the line *:OUTPUT,OC@1(0) ”u,I=10,10””d,I=1,1”* denotes a pull-up delay of 10 and a pull-down delay of 1. A *TOGGLE* is an operator which yields alternating 0s and 1s.

To define semantics we show how the interpretation predicate **I** defined above, is evaluated in the context of the clock example. In the following, an interpretation predicate is defined for every subtree of the TREEMOLA structure. Using the first character of each line as identifier for the nodes, the complete tree in a short form is **U(i(S), o(u(:(.))))**. In what follows, we consider how the tree **U(i(S), o(u(:(.))))** is traversed by the interpretation predicate **I**.

Starting point of the interpretation predicate **I** is the root *U* with its two sons *i(S)* and *o(u(:(.)))*. In general it is possible that there are n sons s_1, ..., s_n of *U*. To achieve a true interpretation of the complete unit module description it is necessary that all sons s_j lead to a true interpretation in the given time interval T_{min} to T_{max}. This leads to an interpretation predicate **I** for the root U(s_1, ..., s_n) as follows:

$$\mathbf{I(U(s_1, ..., s_n), [T_{min},T_{max}], SQ)} :\Leftrightarrow \forall\, j,\ 1 \le j \le n\text{: } I(s_j, [T_{min}, T_{max}], SQ)$$

Because root *i* of *i(S)* is just the beginning of the interface description tree and root *o* of *o(u(:(.)))* is just the beginning of the behavior description tree, both interpretation predicates for these subtrees are equal to the interpretation predicate above:

$$\mathbf{I(i(s_1, ..., s_n), [T_{min},T_{max}], SQ)} :\Leftrightarrow \forall\, j,\ 1 \le j \le n\text{: } I(s_j, [T_{min}, T_{max}], SQ)$$
$$\mathbf{I(o(s_1, ..., s_n), [T_{min},T_{max}], SQ)} :\Leftrightarrow \forall\, j,\ 1 \le j \le n\text{: } I(s_j, [T_{min}, T_{max}], SQ)$$

Note that the complete time interval [T_{min}, T_{max}] has been passed through the interpretation predicates. In addition to the constructs considered in this case study, there are structured statements like AT, IF and CASE which are slightly different because they invoke their sons only at a determined time t.

An interface tree consists of one or several ports described by signal type nodes starting with the character *S*. Semantics of such port specifications is the definition of signals, representing a value 0, 1, X or Z. Usually a range, specifying the width of a port, is given as a tuple (high, low). In the clock example high = low = 0 and the identifier *OC* for the output port is given. The interpretation predicate of each port description is as follows:

$$\begin{gathered}\mathbf{I(S, [T_{min},T_{max}], SQ)} :\Leftrightarrow \forall\, i \in N_0,\ (low \le i \le high)\ \Lambda\ \forall\, t,\ (T_{min} < t \le T_{max})\text{:}\\ (f(SQ, \mathit{OC}, 0, t, i) = 1) \vee\\ (f(SQ, \mathit{OC}, 0, t, i) = 0) \vee\\ (f(SQ, \mathit{OC}, 0, t, i) = X) \vee\\ (f(SQ, \mathit{OC}, 0, t, i) = Z) \vee\\ (f(SQ, \mathit{OC}, 0, t, i) = f(SQ, \mathit{OC}, 0, t-1, i))\end{gathered}$$

The output port has only 1 bit, so i is always 0. The meaning of the last alternative is a storing property of signals. That means, if there is no explicit assignment to a signal at time t, the value at the predecessor time t-1 is assumed to be holding. As mentioned above, the default address is taken as 0. Note that the set of sons of *S* is empty. Here, the interpretation predicate do not identify that a clock output is described and value Z is unpossible. Clock output value Z is prevented by interpretation of the interface assignment tree described below.

Next, the concurrent statement *u* has to be considered. In general it is possible that there are n sons s_1, ..., s_n of a concurrent statement, whereas only one son *:(.)* is given in the clock example. All interpretations of all n subtrees have to lead to a true interpretation and therefore we have an interpretation predicate for the concurrent statement *u*, equal to some predicates above:

$$\mathbf{I(u(s_1, ..., s_n), [T_{min},T_{max}], SQ)} :\Leftrightarrow \forall\, j,\ 1 \le j \le n{:}\ I(s_j, [T_{min}, T_{max}], SQ)$$

Sons of the concurrent statement could be a simple, a structured (e.g. AT, IF, CASE) or another concurrent statement. In this case a simple statement, an interface assignment, has to be considered. Obviously, the value returned by the expression *.TOGGLE(0)* has to be assigned to the output signal *OC*, which serves as an identifier. The clock starts with a low initialization value at time 0 $(exp(SQ, .TOGGLE(0), 0, 0) = 0)$. After a pull-up delay of 10 the signal is going up and then, after a pull-down delay of 1, signal *OC* is going down. At last the complete interpretation predicate for the interface assignment together with the TOGGLE operator is given:

$$\begin{gathered}\mathbf{I(:(.), [T_{min},T_{max}], SQ)} :\Leftrightarrow \\ \forall\, t, (T_{min} < t \le T_{max}){:}\ f(SQ, OC, 0, t, 0) = exp(SQ, .TOGGLE(0), t, 0) \wedge \\ \big(exp(SQ, .TOGGLE(0), 0, 0) = 0\big) \wedge \\ \big((exp(SQ, .TOGGLE(0), t, 0) = 1 \wedge exp(SQ, .TOGGLE(0), t+1, 0) = 0) \vee \\ (exp(SQ, .TOGGLE(0), t, 0) = 0 \wedge exp(SQ, .TOGGLE(0), t+10, 0) = 1)\big)\end{gathered}$$

An informal example of an unfeasible sequence (**I** = FALSE) can be a constant sequence of the value 1 as output of the clock instead of expected alternating values of 0 and 1.

Such interpretation predicates have been defined [Biek92] in a general form within the context of MIMOLA. Transformation of circuit descriptions into the TREEMOLA format is done automatically and the given predicates are used as a specification of the derived simulator.

As a practical example a processor, computing prime factors for a given 16 bit number, has been simulated. The processor consists of 16 modules like a 64 KB-RAM, 3 ALU's, an instruction memory, a program counter, 3 multiplexer, 2 register and a clock.

5. Conclusions

It is necessary to formalize what is meant by a HDL description and provide a calculus for working with such descriptions. Such a calculus makes the task of test generation, synthesis, logic simulation and fault simulation easier and more reliable. It also has significant impact on verification.

The functional semantics for a subset of the TREEMOLA language has been defined. Expressions, memories, registers, declarations, initializations of states, interface and behavior descriptions and compound statements are considered and a simulator based on this semantics has been implemented. This semantics can serve as a basis for verification and supports designers of other tools in the MSS. Translation to and from other intermediate languages as well as other hardware description languages, e.g. VHDL or DACAPO, becomes easier with this semantics. Converters from TREEMOLA to VHDL and vice versa are currently under development. As an important future work, directly executable specifications of circuits using Prolog or ML will be generated automatically from hardware specifications in MIMOLA.

6. References

[BMSJ91] R. Beckmann, P. Marwedel, W. Schenk, and R. Jöhnk. The MIMOLA Language Reference Manual - Version 4.0. Research Report 401, Fachbereich Informatik, University of Dortmund, February 1991.

[Bec91] R. Beckmann, W. Schenk, D. Pusch, and R. Jöhnk. The TREEMOLA Language Reference Manual – Version 4.0. Research Report 391, Fachbereich Informatik, University of Dortmund, July 1991.

[Biek92] U. Bieker. On the Semantics of the TREEMOLA Language Version 4.0. Research Report 435, Fachbereich Informatik, University of Dortmund, July 1992.

[ClMe81] W. F. Clocksin, C. S. Mellish. Programming in Prolog. Springer Verlag, Berlin Heidelberg New York, 1981.

[Devi90] Yves Deville. *Logic Programming: Systematic Program Development*. Addison-Wesley, 1990.

[DOS87] DACAPO II, User Manual, Version 3.0. *DOSIS GmbH, Dortmund*, 1987.

[Eve91] Hans Eveking. *Verifikation digitaler Systeme*. B.G. Teubner Stuttgart, 1991.

[IEEE88] Design Automation Standards Subcommittee of the IEEE. IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076). *IEEE Inc., New York*, 1988.

[Kel87] K. Kelle, G. Krüger, P. Marwedel, L. Nowak, L. Terasa, and F. Wosnitza. Werkzeuge des MIMOLA-Hardware-Entwurfssystems. Bericht 8707, Inst. f. Informatik und prakt. Mathematik, Universität Kiel, June 1987.

[Mar90] P. Marwedel. Matching system and component behaviour in MIMOLA synthesis tools. *Proc. EDAC 1990*, 1990.

[Scho89] Uwe Schöning, U. Kulisch, and H. Maurer. *Logik für Informatiker*. Reihe Informatik. BI Wissenschaftsverlag, Mannheim/Wien/Zürich, 1989.