On the Semantics of the TREEMOLA Language Version 4.0

Ulrich Bieker
Lehrstuhl Informatik XII
University of Dortmund

Report No. 435

July 1992

On the Semantics of the TREEMOLA Language Version 4.0

Table of Contents

1. Introduction 3
   - 1.1 Motivation 3
   - 1.2 Notation 3
   - 1.3 Basic Definitions 4
2. Primary Design Unit 6
3. Structure 6
4. Behavior 7
5. Reservations 10
6. Initializations 10
7. Macros 10
8. Declarations 11
9. Expressions 12
10. Examples 16
11. Conclusions 18
12. Acknowledgements 18
13. Bibliography 19
14. Index 20

1. Introduction

1.1 Motivation

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 good documentation or a formal semantic definition is of great importance for every user of the language. This report is intended to fill this gap in the context of MIMOLA.

MIMOLA [BMSJ91] is a computer hardware description language (CHDL) which has been influenced by other hardware description languages like VHDL [IEEE88] and DACAPO [DOS87]. TREEMOLA is the language that is used to exchange design data between different CAD-tools in the MIMOLA hardware design system MSS,[Kel87], [Mar90].

Using first order predicate calculus [Scho89] a formal semantic definition is obtained for a subset of the intermediate language TREEMOLA. This report is expected to be read along with the *TREEMOLA Language Reference Manual Version 4.0* [Bec91], since all structures defined there serve as a basis for the semantic definition. Organization of this document is similar to the TREEMOLA Manual and therefore some node definitions may be used before their formal definition appear. The goal of this report is to fix the semantic of module declarations of the type <UniTyp>. The reason for restriction to this subset of all node types is to deal primarily with circuit descriptions, after a synthesis is performed. For this reason not all tree alternatives, which are possible using the node types below, are considered. An extension to deal with all TREEMOLA nodes is possible, but there are still some difficulties that need to be resolved especially with constructs like loops, or treatment of all compound statements. On the other hand, there are no problems in dealing with multiple instances of a module because it is possible to generate unique variable identifiers by substitution of <instance_name>.<identifier> for <identifier>. The derivation of a logic program from a correct logic description can lead to a simulator for hardware description languages such as MIMOLA. A detailed description of the techniques concerning derivation of logic programs is given in [Devi90]. The idea of formulating this semantic was inspired by the book *Verifikation digitaler Systeme* written by Hans Eveking [Eve91].

1.2 Notation

The type of nodes is similar as in the TREEMOLA Manual. Some abbreviations used in the report are shown in the following table.

Table 1: abbreviations

$[T_{min},T_{max}]$	time interval, T_{min}, $T_{max} \in N_0$ and $T_{min} \leq T_{max}$
a	address tree
e	expression tree
t	time, nonnegative integer
K	set of nodes
N	set of nonnegative integers
N_0	set of positive integers
R	set of edges
SQ	table of Signals and States

1.3 Basic Definitions

Definition 1.1:

A tree **bn** (behavior node) out of a TREEMOLA unit module description is a structure of the type <nodeentry>. **BN** is the set of all behavior nodes bn.

The semantic definition is given for the following subset of node types:

<typ> ::= <BhvTyp> | <CCtTyp> | <CatTyp> | <CdsTyp> | <DstTyp> | <FctTyp> | <HllTyp> | <IfcTyp> | <IniTyp> | <IntTyp> | <NumTyp> | <SigTyp> | <SrcTyp> | <UniTyp> | <VarTyp>

Definition 1.2:

A TREEMOLA unit module description is a tree **T(K, R)**, $K \subset BN$ a set of nodes, and $R \subset K \times K$, with the following characteristics:

1. There is exactly one root $w \in K$ of the type <UniTyp>: $\forall f, s \in K: ((f, s) \in R \Rightarrow s \neq w)$
2. $\forall s \in K$ with $s \neq w$, exist exactly one sequence $(f_0, ..., f_n)$, $1 \leq n$, $0 \leq i \leq n-1$, $f_i \in K$: $(f_i, f_{i+1}) \in R$, $w = f_0$, $s = f_n$.

Given a tree T(K, R) and w, $r \in K$; The son s is a subtree defined as a tree $T_r(K_r, R_r)$ with the root $r \in K_r$, $K_r \subset K$, $R_r \subset R$ and $(w, r) \in R$. If s denotes a son, then s is an entire subtree and if T(K, R) is a tree with the root $w \in K$ then $\mathbf{T_r(K_r, R_r)}$ denotes a **subtree** with the root $r \in K$.

Definition 1.3:

Given a tree T(K, R) and $f \in K$; the set **SONS(f)** is defined as:

$$SONS(f) := \{s \mid s \text{ is a son of } f\}$$

Definition 1.4:

Given a tree T(K, R), $f \in K$ and SONS(f) is the set of sons of f , and $c = |SONS(f)|$ is the number of sons of f; **O(s)** is a function, allocating all sons of f a unique ordinal number $i \in N$, with $1 \leq i \leq c$, in the following way:

$$\forall s_i, s_j \in SONS(f), (s_i \neq s_j): O(s_i) \neq O(s_j)$$

To decide whether the behavior of a circuit is possible (TRUE) or not (FALSE), it must be possible to store 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_2 is able to access this table. f_2 needs the parameters <identifier>, <address>, <time> and SQ to return the value of an identifier with a given address at a definite time. Additionally, function f_2 requires a bit number i to select bit i of the considered bitstring. If no address is available (e.g. for registers and signals), the default address is taken as 0.

Definition 1.5.1:

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 $i \in N_0$ the function f_2 is defined as follows:

$$f_2(SQ, \langle identifier \rangle, \langle address \rangle, \langle time \rangle, i) \rightarrow \{0, 1, X, Z\}$$

Expressions are evaluated in a similar manner. If there is an expression tree T(K,R) with the root e ∈ K (usually e is of the type <FctTyp>, i.e. built in function type) the two functions exp_2 and exp_{10} are able to return the value of an expression at time t. Function exp_2 requires a bit number i to select bit i of the considered expression whereas exp_{10} returns a decimal value. A conversion from decimal values to binary values or vice versa may be sometimes needed. This is possible if there are no bits set to X or Z and only for this case exp_{10} is defined.

Definition 1.5.2:

Given a 3-dimensional table **SQ**, an expression e, a time <time> ∈ N_0 and a bit number i ∈ N_0 the functions exp_2 and exp_{10} are defined as follows:

$$exp_2\,(SQ, e, \langle time\rangle, i) \rightarrow \{0, 1, X, Z\}$$

$$exp_{10}\,(SQ, e, \langle time\rangle) \rightarrow N_0$$

Here the entire tree T(K, R) with the root e serves as an identifier. If all bits of an expression, given as a bitstring b ∈ $(0, 1)^{n+1}$ with n = high-low, are defined by exp_2, then b can be converted into a decimal value d ∈ N_0 as follows:

$$d = \sum_{i=0}^{n} 2^i \cdot b\,[i]$$

Here b[i] denotes the i.th bit of the bitstring b. Decimal values are usually required for addresses. In most other cases the representation of signal values as bitstrings is prefered.

For a given tree or subtree T(K,R) and its root s ∈ K a predicate I has to be defined. This predicate decides whether a sequence of states in a given time interval and for the circuit described by T(K,R) is a feasible sequence or not.

Later on delay values are deduced from the field <delay_key>. Delays are necessary when assignment trees are considered, i.e. nodes of the type <DstTyp>. It is assumed that delay values are calculated correctly with respect to signals going up or down.

In the following, a tree T(K, R) and its root r ∈ K are denoted as $r(s_1, ..., s_n)$. Every s_i with $1 \le i \le n$ is a subtree

$$T_{r_i}(K_{r_i}, R_{r_i})$$

with the root r_i. Here SONS(r) = $s_1, ..., s_n$ are the sons of r and for all s_i holds $O(s_i) = i$. It is possible that the set of sons of a node is empty. In this case the node is a leaf. This leads to an interpretation predicate I as follows.

Definition 1.6:

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

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

2. Primary Design Unit

According to [Bec91] the node structure as defined in chapter 1.4 of the TREEMOLA Manual is used. Therefore, the type of a concurrent node is abbreviated as <CCtTyp> and identified by the quotation marked character ‘u’ if the identifier is used separately. In association with a tree or another construct, quotation marks are omitted.

Starting point of the interpretation is always a unit module description $U(s_1, ..., s_n)$ with the root ‘U’ of the type <UniTyp>. In this report the subtrees of the sons of ‘U’ are restricted to an interface tree, a behavior tree and an initialization tree.

Definition 2.1:

Given a TREEMOLA unit module description $U(s_1, ..., s_n)$ with the root ‘U’ of the type <UniTyp> ; $\{s_1, ..., s_n\} \subseteq \{$‘i’, ‘o’, ‘e’$\}$ the set of sons of ‘U’; the tree ‘i’ of the type <IfcTyp> (interface type); the tree ‘o’ of the type <BhvTyp> (behavior type); the tree ‘e’ of the type <IniTyp> (initialization type).
$I(U(s_1, ..., s_n), [T_{min},T_{max}], SQ)$ is a true interpretation :⇔

$$\forall\ i,\ 1 \le i \le n: I(s_i, [T_{min}, T_{max}], SQ)$$

Definition 2.2:

Given a TREEMOLA unit module description $i(s_1, ..., s_n)$ with the root ‘i’ of the type <IfcTyp>;
$I(i(s_1, ..., s_n), [T_{min},T_{max}], SQ)$ is a true interpretation :⇔

$$\forall\ i,\ 1 \le i \le n: I(s_i, [T_{min}, T_{max}], SQ)$$

An interface tree consists of one or several ports described by nodes of the type <SigTyp>.

Definition 2.3:

Given a TREEMOLA unit module description ‘S’ with the root ’S’ of the type <SigTyp>, i.e. ‘S’ is a leaf; the number <port_number> of the port ‘S’; the range <bit_range> = (high, low);
$I(S, [T_{min},T_{max}], SQ)$ is a true interpretation :⇔

$$\forall\ i \in N_0, (low \le i \le high)\ \Lambda\ \forall\ t, T_{min} < t \le T_{max}:$$
$$(f_2(SQ, \text{<port\_number>}, 0, t, i) = 1) \vee$$
$$(f_2(SQ, \text{<port\_number>}, 0, t, i) = 0) \vee$$
$$(f_2(SQ, \text{<port\_number>}, 0, t, i) = X) \vee$$
$$(f_2(SQ, \text{<port\_number>}, 0, t, i) = Z) \vee$$
$$(f_2(SQ, \text{<port\_number>}, 0, t, i) = f_2(SQ, \text{<port\_number>}, 0, t\text{-}1, i))$$

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. The field PartId is not used as long as the behavior of just one module instance is considered. The <port_-number>, however, is significant. Logically the port mode (IN, OUT, INOUT, CLK) is unimportant.

3. Structure

In this report the structure tree is not considered.

It is possible to describe a complete finite state machine by one TREEMOLA unit module description $U(s_1, ..., s_n)$. The structure tree is not considered because components of the part tree are just duplicates of the modules defined in the module type tree and therefore the only demand is to create unique identifiers. All connections of the netlist can be considered as equalization of variables. A signal connecting a source part with a destination part is represented by **one** variable. These signals are not allowed to have a delay key and therefore it is possible to use the same variable at the source part as well as at the destination part for all times t.

4. Behavior

A behavior tree $o(s_1, ..., s_n)$ describes the complex behavior of a module and there is an optional declaration subtree and at least one compound statement s_i. Semantics of a variable declaration is given in chapter 8.

Definition 4.1:

Given a TREEMOLA unit module description $o(s_1, ..., s_n)$ with the root 'o' of the type <BhvTyp>; **$I(o(s_1, ..., s_n), [T_{min},T_{max}], SQ)$** is a true interpretation :⇔

$$\forall\ i,\ 1 \le i \le n: I(s_i, [T_{min}, T_{max}], SQ)$$

Only trees of the type <VarTyp> or <CCtTyp> are possible as sons of 'o'. Moreover, at the level considered here, concurrent statements are the only possible compound statements. Trees of the type <SeqTyp>, <SSqTyp> and <ParTyp> are not considered.

Definition 4.2:

Given a TREEMOLA unit module description $u(s_1, ..., s_n)$ with the root 'u' of the type <CCtTyp>; **$I(u(s_1, ..., s_n), [T_{min},T_{max}], SQ)$** is a true interpretation :⇔

$$\forall\ i,\ 1 \le i \le n: I(s_i, [T_{min}, T_{max}], SQ)$$

Sons of the concurrent statement could be a simple, a structured or another compound statement.

Definition 4.3.1:

Given a TREEMOLA unit module description :(e) with the root ':' of the type <DstTyp> and OpId = LOAD, PartId = <part_identifier>, Range = <bit_range> = (high, low), Keys = <delay_keys>; the son 'e', i.e. |SONS(':')| = 1 (register assignment), is an expression; appropriate delay values $\in N_0$ are deduced from <delay_keys>; **$I(:(e), [T_{min},T_{max}], SQ)$** is a true interpretation :⇔

$$\forall\ t, T_{min} < t \le T_{max}\ \Lambda\ \forall\ i \in N_0, (low \le i \le high):$$
$$f_2(SQ, \text{<part\_identifier>}, 0, i, t+delay) = exp_2(SQ, e, t, i)$$

If the register is not loaded at a time t, it is assumed that, with respect to definition 8.2, the predecessor value at time t-1 is valid.

Definition 4.3.2:

Given a TREEMOLA unit module description :(e, c) with the root ':' of the type <DstTyp> and OpId = CONDLOAD, PartId = <part_identifier>, Range = <bit_range> = (high, low), Keys = <delay_keys>; the sons 'e', 'c' of the node ':', with O(e) = 1 and O(c) = 2, i.e. |SONS(':')| = 2 (register assignment), are an expression and a condition; appropriate delay values $\in N_0$ are deduced from <delay_keys>; **$I(:(e, c), [T_{min},T_{max}], SQ)$** is a true interpretation :⇔

$$\forall\ t, T_{min} < t \le T_{max}\ \big(exp_2(SQ, c, t, 0) = 1\ \Lambda\ \forall\ i \in N_0, (low \le i \le high):$$
$$f_2(SQ, \text{<part\_identifier>}, 0, i, t+delay) = exp_2(SQ, e, t, i)\big)$$
$$\vee\ \big(exp_2(SQ, c, t, 0) = 0\big)$$

The range of the condition 'c' is always low = high = 0 and therefore $exp_2(SQ, c, t, 0)$ is evaluated.

Definition 4.4.1:

Given a TREEMOLA unit module description :(e, a) with the root ':' of the type <DstTyp>, OpId = LOAD, PartId = <part_identifier>, Range = <bit_range> = (high, low), Keys = <delay_keys>; the sons 'e', 'a' of the node ':', with O(e) = 1 and O(a) = 2, i.e. |SONS(':')| = 2 (memory assignment), are an expression and an address; appropriate delay values $\in N_0$ are deduced from <delay_keys>; **$I(:(e, a), [T_{min},T_{max}], SQ)$** is a true interpretation :⇔

$$\forall\ t, T_{min} < t \le T_{max}\ \Lambda\ \forall\ i \in N_0, (low \le i \le high):$$
$$f_2(SQ, \text{<part\_identifier>}, exp_{10}(SQ, a, t), i, t+delay) = exp_2(SQ, e, t, i)$$

Definition 4.4.2:

Given a TREEMOLA unit module description :(e,a,c) with the root ':' of the type <DstTyp>, OpId = CONDLOAD, PartId = <part_identifier>, Range = <bit_range> = (high, low), Keys = <delay_keys>; the sons 'e', 'a', 'c' of the node ':', with O(e) = 1, O(a) = 2 and O(c) = 3, i.e. |SONS(':')| =3 (memory assignment), are an expression, an address and a condition; appropriate delay values $\in N_0$ are deduced from <delay_keys>; **I(:(e, a, c), [T_{min},T_{max}], SQ)** is a true interpretation :⇔

$$\forall\, t,\ T_{min} < t \le T_{max}\ \big(exp_2(SQ, c, t, 0) = 1\ \Lambda\ \forall\, i \in N_0,\ (low \le i \le high):$$
$$f_2(SQ, \text{<part\_identifier>}, exp_{10}(SQ, a, t), i, t+delay) = exp_2(SQ, e, t, i)\big)$$
$$\vee\ \big(exp_2(SQ, c, t, 0) = 0\big)$$

Definition 4.5:

Given a TREEMOLA unit module description :(e) with the root ':' of the type <DstTyp>, OpId = OUTPUT, Port = <port_number>, Range = <bit_range> = (high, low), Keys = <delay_keys>; the son 'e' of the node ':', i.e. |SONS(':')| = 1, is an expression; appropriate delay values $\in N_0$ are deduced from <delay_keys>; It follows that **I(:(e), [T_{min},T_{max}], SQ)** is a true interpretation :⇔

$$\forall\, t,\ T_{min} < t \le T_{max}\ \Lambda\ \forall\, i \in N_0,\ (low \le i \le high):$$
$$f_2(SQ, \text{<port\_number>}, 0, i, t+delay) = exp_2(SQ, e, t, i)$$

The signal assignment tree which denotes an assignment to a (local) signal is not considered here, because it is assumed that there remain no local signals after the synthesis is performed. Therefore an interpretation predicate for nodes of the type <DstTyp> with OpId = SIGASSIGN is not defined.

Definition 4.6:

Given a TREEMOLA unit module description ':' with the root ':' of the type <DstTyp>, OpId = NOLOAD and PartId = <part_identifier>; **I(:, [T_{min},T_{max}], SQ)** is always a true interpretation.

Later in definition 8.2 it is guaranteed that, without using the NOLOAD statement, the values of the predecessor time t-1 are held for all registers and memories if there is no explicit assignment at time t. A NOLOAD statement does not lead to an activity but indirectly it is assumed that the values of a register or memory are not changed.

Up to this point, the complete time interval [T_{min}, T_{max}] has been passed through the interpretation predicates. The next definitions concerning the structured statements AT, IF and CASE are slightly different because they invoke their sons only at a determined time t.

Definition 4.7.1:

Given a TREEMOLA unit module description !(e, s_2, ...,s_n) with the root '!' of the type <HllTyp>, OpId = AT, PartId = UP; the son 'e' of the node '!' with O(e) = 1 being an expression;
I(!(e, s_2, ..., s_n), [T_{min},T_{max}], SQ) is a true interpretation :⇔

$$\forall\, t,\ T_{min} < t \le T_{max}:$$
$$(exp_{10}(SQ, e, t) = 1\ \Lambda\ exp_{10}(SQ, e, t\text{-}1) = 0\ \Lambda\ (\ \forall\, i,\ 2 \le i \le n:\ I(s_i, t, SQ)))$$
$$\vee\ (exp_{10}(SQ, e, t) = 0) \vee (exp_{10}(SQ, e, t\text{-}1) = 1)$$

Definition 4.7.2:

Given a TREEMOLA unit module description !(e, s_2, ...,s_n) with the root '!' of the type <HllTyp>, OpId = AT, PartId = DOWN; the son 'e' of the node '!' with O(e) = 1 being an expression;
I(!(e, s_2, ..., s_n), [T_{min},T_{max}], SQ) is a true interpretation :⇔

$$\forall\, t,\ T_{min} < t \le T_{max}:$$
$$(exp_{10}(SQ, e, t) = 0\ \Lambda\ exp_{10}(SQ, e, t\text{-}1) = 1\ \Lambda\ (\forall\, i,\ 2 \le i \le n:\ I(s_i, t, SQ)))$$
$$\vee\ (exp_{10}(SQ, e, t) = 1) \vee (exp_{10}(SQ, e, t\text{-}1) = 0)$$

Definition 4.7.3:

Given a TREEMOLA unit module description !(e, s_2, ...,s_n) with the root '!' of the type <HllTyp>, OpId = AT, PartId = HIGH; the son 'e' of the node '!' with O(e) = 1 being an expression;
I(!(e, s_2, ..., s_n), [T_{min},T_{max}], SQ) is a true interpretation :⇔

$$\forall\, t,\ T_{min} < t \le T_{max}:$$
$$(exp_{10}(SQ, e, t) = 1 \Lambda (\forall\, i,\ 2 \le i \le n: I(s_i, t, SQ)))$$
$$\vee\ (exp_{10}(SQ, e, t) = 0)$$

Definition 4.7.4:

Given a TREEMOLA unit module description !(e, s_2, ...,s_n) with the root '!' of the type <HllTyp>, OpId = AT, PartId = LOW; the son 'e' of the node '!' with O(e) = 1 being an expression;
I(!(e, s_2, ..., s_n), [T_{min},T_{max}], SQ) is a true interpretation :⇔

$$\forall\, t,\ T_{min} < t \le T_{max}:$$
$$(exp_{10}(SQ, e, t) = 0 \Lambda (\forall\, i,\ 2 \le i \le n: I(s_i, t, SQ)))$$
$$\vee\ (exp_{10}(SQ, e, t) = 1)$$

Definition 4.8:

Given a TREEMOLA unit module description ?(c, s_1, s_2) with the root '?' of the type <CdsTyp>, OpId = IF; the sons 'c', 's_1', 's_2' of the node '?', with O(c) = 1, O(s_1) = 2 and O(s_2) = 3, i.e. |SONS('?')| =3 are a condition and two statements (then and else); **I(?(c, s_1, s_2), [T_{min},T_{max}], SQ)** is a true interpretation :⇔

$$\forall\, t,\ T_{min} < t \le T_{max}:$$
$$(exp_2(SQ, c, t, 0) = 1 \Lambda I(s_1,\ t, SQ)) \vee (exp_2(SQ, c, t, 0) = 0 \Lambda I(s_2,\ t, SQ))$$

The meaning of an IF statement is that the first statement is executed if the value of the condition 'c' is 1, otherwise the second statement is executed. The range of the condition 'c' is always low = high = 0 and therefore exp_2(SQ, c, t, 0) is evaluated, i.e. the selected bit number is 0.

Definition 4.9:

Given a TREEMOLA unit module description !(e, s_2, ...,s_n) with the root '!' of the type <HllTyp>, OpId = CASE; the son 'e' of the node '!' with O(e) = 1, being an expression; all trees 's_i' are of the type <HllTyp> with OpId = OF or ELSE;
I(!(e, s_2, ..., s_n), [T_{min},T_{max}], SQ) is a true interpretation :⇔

$$\forall\, t\ \exists\ i,\ 2 \le i \le n\ \Lambda\ \Big($$
$$\big(OpId = OF\ \Lambda\ |SONS(s_i)| = n\ \Lambda\ n > 1\ \Lambda\ \exists\, c \in SONS(s_i)\ \Lambda\ O(c) < n\ \Lambda$$
$$exp_{10}(SQ,e, t) = exp_{10}(SQ, c, t)\ \Lambda\ st \in SONS(s_i)\ \Lambda\ O(st) = n\ \Lambda\ I(st, t, SQ)\ \big) \vee$$
$$\big(OpId = ELSE\ \Lambda\ |SONS(s_i)| = 1\ \Lambda\ st \in SONS(s_i)\ \Lambda\ I(st, t, SQ)\big)\Big)$$

The only possibility to fix the semantics of a tree of the type <HllTyp> with OpId = OF or OpId = ELSE is to look at these nodes in association with the CASE structure. In contrast to a CASE-expression (given in definition 9.7) a CASE-statement returns no value. In the TREEMOLA language the OF and ELSE sons are always of the type <HllTyp> (high level language type) and therefore these two cases are only distinguishable by the CASE node. A CASE-expression is of the type <FctTyp> and a CASE-statement of the type <HllTyp> and thereby the conflict is resolved.

5. Reservations

The reservation tree is not considered at this level of circuit representation.

6. Initializations

Definition 6.1

Given a TREEMOLA unit module description $e(s_1, ...,s_n)$ with the root ‘e’ of the type <IniTyp>; the address range of an ‘s_i’ is (min_i, max_i) with $min_i, max_i \in N_0$; $<identifier>_i$ is the OpId of an ‘s_i’; Range = <bit_range> = $(high_i, low_i)$ is the bit range of ‘c_i’ $\in$ SONS(s_i) (constant expression);
$I(e(s_1, ..., s_n), [T_{min},T_{max}], SQ)$ is a true interpretation:$\Leftrightarrow$

$$\forall\ i,\ 1 \le i \le n\ \Lambda\ \forall\ address \in N_0,\ min_i \le address \le max_i\ \Lambda\ \forall\ j \in N_0,\ (low_i \le j \le high_i):$$
$$f_2(SQ, <identifier>_i, address, 0, j) = exp_2(SQ, c_i, 0, j))$$

The meaning of this definition is, that at time 0 a register or a memory is initialized by a constant expression. In case of addressable memory, all cells within the address range are initialized by the same constant.

7. Macros

Macro trees are not considered at this level of circuit representation.

8. Declarations

Only signal declaration trees and variable declaration trees are considered.

Definition 8.1:

Given a TREEMOLA unit module description $S(s_1, ...,s_n)$ with the root 'S' of the type <SigTyp>; the OpId = <identifier> of a 's_j'; the Range <bit_range> = (high, low) of a 's_j';
$I(S(s_1, ..., s_n), [T_{min},T_{max}], SQ)$ is a true interpretation :⇔

$$\forall\ j,\ 1 \le j \le n\ \Lambda\ \forall\ i \in N_0,\ (low \le i \le high)\ \Lambda\ \forall\ t,\ T_{min} < t \le T_{max}:$$
$$(f_2(SQ, \text{<identifier>}, 0, t, i) = 1) \vee$$
$$(f_2(SQ, \text{<identifier>}, 0, t, i) = 0) \vee$$
$$(f_2(SQ, \text{<identifier>}, 0, t, i) = X) \vee$$
$$(f_2(SQ, \text{<identifier>}, 0, t, i) = Z) \vee$$
$$(f_2(SQ, \text{<identifier>}, 0, t, i) = f_2(SQ, \text{<identifier>}, 0, t\text{-}1, i))$$

Again 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 of the predecessor time t-1 is assumed to be holding. It is possible that there are several sons 's_j' of the root 'S'. Every tree 's_j' has got its own unique identifier and range.

Definition 8.2:

Given a TREEMOLA unit module description $V(\__1(s_1, s_2, s_3, s_4, s_5), ...,\__n(s_1, s_2, s_3, s_4, s_5))$ with the root 'V' of the type <VarTyp>, '$\__k$' and 's_1' of the type <SrcTyp>; 's_2', 's_3', 's_4', 's_5' of the type <NumTyp>; OpId = <identifier> of a '$\__k$'; the range <bit_range> = (high, low) of a '$\__k$'; the constant expression $exp_{10}(SQ, s_3, t)$ = <size_number> ∈ N is the number of cells required for the variable;
$I(V(\__1(s_1, s_2, s_3, s_4, s_5), ...,\__n(s_1, s_2, s_3, s_4, s_5)), [T_{min},T_{max}], SQ)$ is a true interpretation :⇔

$$\forall\ k,\ 1 \le k \le n\ \Lambda\ \forall\ i, j \in N_0,\ (1 \le j \le \text{<size\_number>},\ low \le i \le high)\ \Lambda\ \forall\ t,\ T_{min} < t \le T_{max}:$$
$$(f_2(SQ, \text{<identifier>}, j, t, i) = 1) \vee$$
$$(f_2(SQ, \text{<identifier>}, j, t, i) = 0) \vee$$
$$(f_2(SQ, \text{<identifier>}, j, t, i) = X) \vee$$
$$(f_2(SQ, \text{<identifier>}, j, t, i) = Z) \vee$$
$$(f_2(SQ, \text{<identifier>}, j, t, i) = f_2(SQ, \text{<identifier>}, j, t\text{-}1, i))$$

Again the meaning of the last alternative is a storing property of variables, i.e. memories or registers. Every tree '$\__k$' has got its own unique identifier and range.

The other type declaration trees (e.g. referenced type, range type, bit type, field type, array type, record type, subprogram type) are not considered here.

9. Expressions

The treatment of expressions is done in the same order as in the TREEMOLA Manual. From amongst several standard operators available in MIMOLA [BMSJ91] we shall demonstrate the definition for the following operators to illustrate the basic methodology of defining semantics. The operators are TOGGLE, ABS, NOT, =, AND and OR. In case of standard operators the node is of the type <FctTyp> and OpId = <operation_identifier>.

Expressions return the result of a computation. The roots of the expression trees considered here are of the type <CatTyp>, <FctTyp>, <IntTyp> or <NumTyp>. In chapter 1.3 the two functions exp_2 and exp_{10} for evaluating expressions have been introduced. The result of an expression is always valid in the closed time interval $[T_{min}, T_{max}]$, because the arguments are used by the operator at every time t.

TREEMOLA stores the 4 logic values 0, 1, X and Z using two strings V and X . To decode the TREEMOLA field <bitstring> = (V, X), it is necessary to define a function decode((V, X), i) as follows:

Definition 9.0:

Given the two words V, X $\in (0,1)^n$ (i.e. <bitstring> = (V, X)) and the bit position i with $1 \le i \le n$; decode((V, X), i) -> {0, 1, Z, X} is defined as follows:

$$\text{decode}((V, X), i) = 1 :\Leftrightarrow V[i] = 1 \ \Lambda\ X[i] = 0$$
$$\text{decode}((V, X), i) = 0 :\Leftrightarrow V[i] = 0 \ \Lambda\ X[i] = 0$$
$$\text{decode}((V, X), i) = X:\Leftrightarrow V[i] = 0 \ \Lambda\ X[i] = 1$$
$$\text{decode}((V, X), i) = Z :\Leftrightarrow V[i] = 1 \ \Lambda\ X[i] = 1$$

This function is needed to calculate the value returned by an expression of the type <IntTyp>. An expression is defined either by exp_2 or exp_{10}. Both can be converted into each other as discussed earlier.

Definition 9.1.1:

Given a TREEMOLA unit module description '=' with the root '=' of the type <IntTyp>; Integ = <bitstring> = (V, X); Range <bit_range> = (high, low). The value returned by exp_2(SQ, '=', t, i) is determined as follows:

$$\forall\ i \in N_0,\ (low \le i \le high) :$$
$$exp_2(SQ, '=', t, i) = \text{decode}((V, X), i)$$

This means, that the value returned by a node of the type <IntTyp> is a constant for all time t.

Definition 9.1.2:

Given a TREEMOLA unit module description '%' with the root '%' of the type <NumTyp> and NumValue = <integer> $\in N_0$; the value returned by exp_{10}(SQ, '%', t) is determined as follows:

$$exp_{10}(SQ, '\%', t) = \text{<integer>}$$

This is exactly the same meaning, that the returned value is a constant for all time t.

Definition 9.2.1:

Given a TREEMOLA unit module description :(.) with the root ':' of the type <DstTyp>; Keys = <delay_keys>; the son '.' of the type <FctTyp> with OpId = TOGGLE; appropriate delay values $\in N_0$ (updelay, downdelay) and a value <init_delay> are deduced from <delay_keys>. The value returned by exp_{10}(SQ, :(.), t) is determined as follows:

$$(t = \text{<init\_delay>} \ \Lambda\ exp_{10}(SQ, :(.), t) = 0) \vee$$
$$(t \neq \text{<init\_delay>} \ \Lambda\ (exp_{10}(SQ, :(.), t) = 1 \ \Lambda\ exp_{10}(SQ, :(.), t+\text{<downdelay>}) = 0) \vee$$
$$(exp_{10}(SQ, :(.), t) = 0 \ \Lambda\ exp_{10}(SQ, :(.), t+\text{<updelay>}) = 1))$$

In this definition it is required, that a TOGGLE operator is preceded by a node of the type <DstTyp>. An initialization of a TOGGLE is always done by 0. According to the TREEMOLA Manual the delay values are supposed to be part of the node ':' of the type <DstTyp>.

Definition 9.2.2:

Given a TREEMOLA unit module description $._1(._2)$ with the root '$._1$' of the type <FctTyp>; OpId = ABS; the son '$._2$' of '$._1$' is an expression, i.e. |SONS('$._1$')| = 1.
The value returned by $exp_{10}(SQ, ._1(._2), t)$ is determined as follows:

$$\left(exp_{10}(SQ, ._2, t) \geq 0 \wedge exp_{10}(SQ, ._1(._2), t) = exp_{10}(SQ, ._2, t)\right) \vee$$
$$\left(exp_{10}(SQ, ._2, t) < 0 \wedge exp_{10}(SQ, ._1(._2), t) = -exp_{10}(SQ, ._2, t)\right)$$

Definition 9.2.3:

Given a TREEMOLA unit module description $._1(._2)$ with the root '$._1$' of the type <FctTyp>; OpId = NOT; Range <bit_range> = (high, low); the son '$._2$' of '$._1$' is an expression, i.e. |SONS('$._1$')| = 1.
The value returned by $exp_2(SQ, ._1(._2), t, i)$ is determined as follows:

$$\forall\, i \in N_0, (low \leq i \leq high):$$
$$\left(exp_2(SQ, ._2, t, i) = 0 \wedge exp_2(SQ, ._1(._2), t, i) = 1\right) \vee$$
$$\left(exp_2(SQ, ._2, t, i) = 1 \wedge exp_2(SQ, ._1(._2), t, i) = 0\right)$$

Definition 9.2.4:

Given a TREEMOLA unit module description $._=(._1, ._2)$ with the root '$._=$' of the type <FctTyp>; OpId = '='; Range <bit_range> = (high, low); the sons '$._1$' and '$._2$' are expressions, i.e. |SONS('$._=$')| = 2.
The value returned by $exp_2(SQ, ._=(._1, ._2), t, 0)$ is determined as follows:

$$\left(\forall\, i \in N_0, (low \leq i \leq high): exp_2(SQ, ._1, t, i) = exp_2(SQ, ._2, t, i) \wedge exp_2(SQ, ._=(._1, ._2), t, 0) = 1\right) \vee$$
$$\left(\exists\, i \in N_0, (low \leq i \leq high): exp_2(SQ, ._1, t, i) \neq exp_2(SQ, ._2, t, i) \neq X \wedge exp_2(SQ, ._=(._1, ._2), t, 0) = 0\right) \vee$$
$$\left(\exists\, i \in N_0, (low \leq i \leq high): exp_2(SQ, ._1, t, i) = X \wedge exp_2(SQ, ._=(._1, ._2), t, 0) = X\right) \vee$$
$$\left(\exists\, i \in N_0, (low \leq i \leq high): exp_2(SQ, ._2, t, i) = X \wedge exp_2(SQ, ._=(._1, ._2), t, 0) = X\right)$$

A test on equality yields the value 1 if the expressions of both sons are equal. The value X is returned if at least one bit of an expression is X and the remaining bits are equal. Unequal bits which are not X yields the value 0.

Definition 9.2.5:

Given a TREEMOLA unit module description $.(e_1, ..., e_n)$ with the root '.' of the type <FctTyp>; Range <bit_range> = (high, low); OpId = AND; the sons 'e_1', ..., 'e_n' are expressions, i.e. |SONS('.')| > 2. The value returned by $exp_2(SQ, .(e_1, ..., e_n), t, j)$ is determined as follows:

$$\left(\forall\, i, (1 \leq i \leq n): exp_2(SQ, e_i, t, j) = 1 \wedge exp_2(SQ, .(e_1, ..., e_n), t, j) = 1\right) \vee$$
$$\left(\exists\ i\ (1 \leq i \leq n): exp_2(SQ, e_i, t, j) = 0 \wedge exp_2(SQ, .(e_1, ..., e_n), t, j) = 0\right)$$

Definition 9.2.6:

Given a TREEMOLA unit module description $.(e_1, ..., e_n)$ with the root '.' of the type <FctTyp>; Range <bit_range> = (high, low); OpId = OR; the sons 'e_1', ..., 'e_n' are expressions, i.e. |SONS('.')| > 2. The value returned by $exp_2(SQ, .(e_1, ..., e_n), t, j)$ is determined as follows:

$$\left(\forall\, i, (1 \leq i \leq n): exp_2(SQ, e_i, t, j) = 0 \wedge exp_2(SQ, .(e_1, ..., e_n), t, j) = 0\right) \vee$$
$$\left(\exists\ i\ (1 \leq i \leq n): exp_2(SQ, e_i, t, j) = 1 \wedge exp_2(SQ, .(e_1, ..., e_n), t, j) = 1\right)$$

All available MIMOLA standard operators could be defined in the same way. The allowed number of arguments of an operator is identical to the number of sons of this operator. Sons of an expression are evaluated and used by the operator until the trees are leaves.

Definition 9.3:

Given a TREEMOLA unit module description *(e_1, ..., e_n) with the root '*' of the type <CatTyp>; Range <bit_range> = ($high_*$, low_*); the Range ($high_i$, low_i) for every expression 'e_i', $1 \le i \le n$ and l_i = ($high_i$ - low_i + 1). The value returned by exp_2(SQ, *(e_1, ..., e_n), t, j) with $high_* \le j \le low_*$ is determined as follows:

$$(exp_2(SQ, *(e_1, ..., e_n), t, j) = exp_2(SQ, e_i, t, k)) \Lambda$$
$$(k = low_i + relA) \text{ with}$$

$$i = max\left\{ i \mid \sum_{\iota = i}^{n} l_\iota > j \right\}$$

$$relA = j - \sum_{\iota = i+1}^{n} l_\iota$$

The highest bits of the catenated value are given by the first son of a catenation tree. The lowest bits are given by the last son. The returned value of the catenation tree is a part (high, low) of the complete catenated value. Definition 9.3 is illustrated by the following example:

Table 2: catenation example

j	12	11	10	9	8	7	6	5	4	3	2	1	0
k	8	7	6	5	4	4	3	2	1	0	3	2	1
i	1	1	1	1	1	2	2	2	2	2	3	3	3

Table 2 shows an example with three expressions e_1 with the range (8, 4), e_2 with the range (4, 0) and e_3 with the range (3, 1).

$$\Rightarrow l_1 = 5, l_2 = 5, l_3 = 3, n = 3$$

To calculate exp_2(SQ, *(e_1, e_2, e_3), t, j) with j = 11 it is necessary to know i and k in exp_2(SQ, e_i, t, k).

$$\Rightarrow$$

$$\left(i = max\left\{ i \mid \sum_{\iota = i}^{n} l_\iota > 11 \right\} \right), \sum_{\iota = 1}^{3} l_\iota = 13$$

$$\Rightarrow \text{the maximum i is } i = 1$$

That means that the considered bit is part of 'e_1'. To calculate the relevant k in 'e_1' the relative address relA is determined as follows:

$$low_i = low_1 = 4, k = 4 + relA, relA = 11 - 8 = 3 \Rightarrow k = 7$$
$$\Rightarrow exp_2(SQ, *(e_1, ..., e_n), t, 11) = exp_2(SQ, e_1, t, 7)$$

Definition 9.4.1:

Given a TREEMOLA unit module description '.' with the root '.' of the type <FctTyp>; OpId = READ; PartId = <part_identifier>; Range = <bit_range> = (high, low); the set SONS('.') is empty, i.e |SONS('.')| = 0 (register access). The value returned by exp_2(SQ, '.', t, i) is determined as follows:

$$\forall\, i \in N_0, (low \le i \le high): exp_2(SQ, '.', t, i,) = f_2(SQ, <part\_identifier>, 0, t, i)$$

Definition 9.4.2:

Given a TREEMOLA unit module description .(a) with the root '.' of the type <FctTyp>; OpId = READ; PartId = <part_identifier>; Range = <bit_range> = (high, low); the son 'a' is an expression, i.e. the address, and |SONS('.')| = 1(memory access). The value returned by exp_2(SQ, .(a), t, i) is determined as follows:

$$\forall\, i \in N_0, (low \le i \le high):$$

$$exp_2(SQ, .(a), t, i) = f_2(SQ, <part\_identifier>, exp_{10}(SQ, a, t), t, i)$$

The READ operator returns the value of a register or memory by looking into the table SQ with the help of the function f_2. If the address is not given it is by default set to 0. The address is evaluated by the expression exp_{10}(SQ, a, t).

Definition 9.5:

Given a TREEMOLA unit module description '.' with the root '.' of the type <FctTyp>; OpId = INPUT; Port = <port_number>; Range = <bit_range> = (high, low) and the set SONS('.') is empty, i.e '.' is a leaf. The value returned by exp_2(SQ, '.', t, i) is determined as follows:

$$\forall\, i \in N_0, (low \le i \le high): exp_2(SQ, '.', t, i) = f_2(SQ, <port\_number>, 0, t, i)$$

In this case <port_number> serves as an identifier.

The signal access tree which denotes an access to a (local) signal is not considered here, because it is assumed that there remain no local signals after the synthesis tool is performed. Therefore an interpretation predicate for nodes of the type <FctTyp> with OpId = SIGASSIGN is not defined (see also chapter 4).

Definition 9.6:

Given a TREEMOLA unit module description .(c, e_1, e_2) with the root '.' of the type <FctTyp>; OpId = SELECT2; Range = <bit_range> = (high, low); the sons 'c', 'e_1', 'e_2' with O(c) = 1, O(e_1) = 2 and O(e_2) = 3, i.e. |SONS('.')| =3 , are a condition and two expressions.
The value returned by exp_2(SQ, .(c, e_1, e_2), t, i), with low ≤ i ≤ high, is determined as follows:

$$\begin{aligned}
&\big(exp_2(SQ, c, t, 0) = 1 \,\Lambda\, exp_2(SQ, .(c, e_1, e_2), t, i) = exp_2(SQ, e_1, t, i)\big) \vee \\
&\big(exp_2(SQ, c, t, 0) = 0 \,\Lambda\, exp_2(SQ, .(c, e_1, e_2), t, i) = exp_2(SQ, e_2, t, i)\big) \vee \\
&\big(exp_2(SQ, c, t, 0) = X \,\Lambda\, exp_2(SQ, e_2, t, i) = exp_2(SQ, e_1, t, i) = exp_2(SQ, .(c, e_1, e_2), t, i)\big) \vee \\
&\big(exp_2(SQ, c, t, 0) = X \,\Lambda\, exp_2(SQ, e_2, t, i) \neq exp_2(SQ, e_1, t, i) \,\Lambda\, exp_2(SQ, .(c, e_1, e_2), t, i) = X\big)
\end{aligned}$$

Definition 9.7:

Given a TREEMOLA unit module description .(e, e_2, ...,e_n) with the root '.' of the type <FctTyp>; OpId = CASE; Range = <bit_range> = (high, low); the son 'e' of the node '.' with O(e) = 1 is an expression; all trees 'e_i' are of the type <HllTyp> with OpId = OF or ELSE. The value returned by exp_2(SQ, .(e, e_2, ...,e_n), t, j), with low ≤ j ≤ high, is determined as follows:

$$\begin{aligned}
&\exists\ i, 2 \le i \le n \,\Lambda \\
&\big(OpId = OF \,\Lambda\, |SONS(s_i)| = n \,\Lambda\, n > 1 \,\Lambda\, \exists\, c \in SONS(s_i) \,\Lambda\, O(c) < n \,\Lambda \\
&exp_{10}(SQ,e, t) = exp_{10}(SQ, c, t) \,\Lambda\, st \in SONS(s_i) \,\Lambda\, O(st) = n \,\Lambda \\
&exp_2(SQ, .(e, e_2, ...,e_n), t, j) = exp_2(SQ, st, t, j)\big) \vee \\
&\big(OpId = ELSE \,\Lambda\, |SONS(s_i)| = 1 \,\Lambda\, st \in SONS(s_i) \,\Lambda\, exp_2(SQ, .(e, e_2, ...,e_n), t, j) = exp_2(SQ, st, t, j)\big)
\end{aligned}$$

10. Examples

In what follows we discuss two examples to illustrate the semantics. The first one refers to an example module which is basically a clock.

```
UINTCLOCK
  iINTCLOCK
    SOUT,INTERNALCL@1(0)
  oRTLEVEL,INTCLOCK
    uL0018
     :OUTPUT,INTERNALCL@1(0)"u,I=2,2""d,I=1,1""i,I=0,0"
        .TOGGLE(0)
```

The complete tree in a short form is: U(i(S), o(u(:(.))))

Therefore the starting point for the interpretation of this tree is I(U(i, o), [T_{min}, T_{max}], SQ). Using definition 2.1 it is noted that I(U(i, o), [T_{min}, T_{max}], SQ) is true, if I(i(S), [T_{min}, T_{max}], SQ) and I(o(u), [T_{min}, T_{max}], SQ) is true. The interpretation of the interface tree i is true if the interpretation of all sons becomes true. In this case there is only one port S described. If there is no initialization of the signal INTERNALCL the only thing which could be said at this moment is, that the signal has the value 1, 0, X or Z or holds the value if there is no explicit assignment at time t.

The interpretation of the behavior tree o as well as of the concurrent tree u is true, if the interpretation of all sons is true. This leads directly to the interpretation I(:(.), [T_{min}, T_{max}], SQ). In definition 9.2.1 it is said that the value of the signal INTERNALCL is 0 at time t+downdelay = t+1 if the value of INTERNALCL has been 1 at time t. The value of INTERNALCL is 1 at time t+updelay = t+2 if the value has been 0 at time t. The clock signal is initialized by 0 at time 0. After this interpretation e.g. exp_2(SQ, '.TOGGLE', 0, 0) returns 0.

The second example module shown below is a 4-bit register. The interface tree contains one output STATE with the range (3, 0). Two control inputs LADE and RESET are of the range (0, 0) and the input SUM contains a 4-bit data. An additional clock input has got the port number 5. Looking at the behavior node o the interpretation I(o(V, u), [T_{min}, T_{max}], SQ) has to be executed. In the declaration part a variable REGSTATE of the type halfbyte with the range (3, 0) is declared. The third grandson of V is %1 and that means, that there is just one duplicate of the variable. Every bit of the variable REGSTATE has the value 1, 0, X or Z or holds the value if there is no explicit assignment at time t. Using I(u(:, !), [T_{min}, T_{max}], SQ) the concurrent tree u is interpreted. This leads to two interpretation calls: I(:(.), [T_{min}, T_{max}], SQ) and I(!(., ?), [T_{min}, T_{max}], SQ) . Using definition 4.5 the node ‘:’ of the type <DstTyp> is interpreted. An assignment of the register value REGSTATE to the output signal STATE with a default delay of 1 is performed. The high level statement AT needs the conditional node ‘.’ with OpId = INPUT and PartId = CLOCK to decide whether the first son u(:(=)) or the second son u(?(., u(:(.)), u(:))) has to be interpreted at time t.

```
UREGISTER
  iREGISTER
    SOUT,STATE@1(3:0)
    SIN,LADE@2(0)
    SIN,RESET@3(0)
    SIN,SUM@4(3:0)
    SCLK,CLOCK@5(0)
  oRTLEVEL,REGISTER
    V
      _REGSTATE,HALFBYTE(3:0)
        _Mlocation
        %0
        %1
        %1
        %0
    uL0046
      :OUTPUT,STATE@1(3:0)
        .READ,REGSTATE(3:0)
      !AT,UP
        .INPUT,CLOCK@5(0)
        ?IF
          .INPUT,RESET@3(0)
          uL0048
            :LOAD,REGSTATE(3:0)
              =%0000(3:0)
          uL0049
            ?IF
              .INPUT,LADE@2(0)
              uL0049
                :LOAD,REGSTATE(3:0)
                  .INPUT(3:0)
              uL0050
                :NOLOAD.REGSTATE
```

11. 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, logic simulation and fault simulation easier. It also can have significant impact on verification.

A functional semantics for a subset of the TREEMOLA language has been defined. This semantics can serve as a basis for verification as well as for simulation and supports designers of other tools in the MIMOLA hardware design system. 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 developement.

12. Acknowledgements

I gratefully acknowledge the time spent by Prof. Bhatt and Prof. Padawitz during several fruitful discussions. Finally I wish to thank Prof. Marwedel for his comments and final revision of the text.

13. Bibliography

[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.

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

[DOS87] Fa. DOSIS. 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.

14. Index

A
ABS 12, 13
address 3, 8, 10
AND 12, 13
AT 8, 9, 17

B
behavior node 4, 16, 17
BhvTyp 4, 6, 7
bitstring 4, 5, 12

C
CASE 9, 15
catenation tree 14
CatTyp 4, 12, 14
CCtTyp 4, 7
CdsTyp 4, 9
CLK 6
CLOCK 17
clock 16
compound statement 3, 7
concurrent 7, 16, 17
condition 7, 8, 9, 15
CONDLOAD 7, 8
constant expression 10, 11

D
decode 12
DOWN 8
DstTyp 4, 7, 8, 12, 13, 17

E
ELSE 9, 15
else 9
expression 3, 5, 8, 9, 12, 13

F
FctTyp 4, 5, 12, 13, 15

H
HIGH 9
HllTyp 4, 8, 9, 15

I
IF 9
IfcTyp 4, 6
IN 6
IniTyp 4, 6, 10
INOUT 6
INPUT 15, 17
IntTyp 4, 12

L
LOAD 7, 17
loops 3
LOW 9

M
memory assignment 7, 8

N
NOLOAD 8, 17
NOT 12, 13
NumTyp 4, 11, 12

O
O(s) 4
OF 9, 15
OR 12, 13
OUT 6
OUTPUT 8, 16

P
ParTyp 7
port 6, 16

R
READ 15, 17
register assignment 7

S
SELECT2 15
SeqTyp 7
SIGASSIGN 8, 15
signal access tree 15
signal assignment 8
SigTyp 4, 11
SONS 4
SQ 4, 5
SrcTyp 4, 11
SSqTyp 7
storing property 6, 11
subtree 4, 5

T
then 9
TOGGLE 12, 13, 16

U
UniTyp 3, 4, 6
UP 8, 17

V
VarTyp 4, 7, 11