APPENDIX A - VHDL-AMS KEYWORDS

Keyword	Syntax	Function
AND	*A* and *B*	Logical and
ARCHITECTURE	*ARCHITECTURE behav OF example IS* || *declaration part* || *BEGIN* || *statements section* || *END ARCHITECTURE behav;*	Assigning the behaviour description *behav* to the object *example* (see 1.2)
BIT	*Name: BIT*	Declare a variable in the binary format (permitted values: '0' or '1') Value Assignment: Name <= '0'
BREAK ON	*Interruption of a process*	See Process
CASE ... IS	*CASE* examining_variable *IS* *{ WHEN* expression > = || instructions || } [*WHEN OTHERS* = > || instructions ||] *END CASE;*	Procedural (sequential) *CASE* instruction. The same restrictions are valid as in the sequential *IF* instruction (see *IF… THEN*)
CASE ... USE	*CASE* examining_variable *USE* *{WHEN* expression > = || instructions || } [*WHEN OTHERS* = > || instructions ||] *END CASE;*	Simultaneous *CASE* instruction. (see If USE ...)
DOT	*quantityname 'DOT*	Gives back the 1st derivation of time of the *QUANTITY quantityname* as a floating point number (*REAL)*
ENTITY	*ENTITY example [* (*behav*) *]* *GENERIC MAP* (|| *parameter assignments* ||) *PORT MAP* (|| *interface assignments* ||);	Creates instance of the component Example under use of the behaviour description *behav* for this component
ENTITY ... IS	*ENTITY example* IS *GENERIC*(|| *PARAMETER_DECLARATION* ||); *PORT*(|| *INTERFACE_DECLARATION* ||); *BEGIN* || *instructions* || *END ENTITY example*	Laying out a new object with the name Example (see 1.2.1). Definition of the interfaces (lay out a prototype)

FOR ... TO ... LOOP	*[loop_label:] FOR* range *LOOP* *||sequential_instructions||* *END LOOP [loop_label];*	'Normal' *FOR* loop (see) only usable in processes, because sequential execution, is required (see *PROCESS*)
GENERIC	*GENERIC* (|| *parameterdecl.* ||);	Declaration part defining parameters which can be adjust at instantiation of the component (see 1.2.1, *ENTITY*)
IF ... THEN	*IF* (*condition*) *THEN* || *statement_section* || *{ELSIF* (*condition*) || *statements_section* ||*}* *[ELSE* || *statement_section* ||*]* *END IF;*	Procedural *IF* instruction. This *IF* instruction is permitted only within processes. Therefore the instructions are executed sequentially and the condition is checked only if the *PROCESS* is active.
IF ... USE	*IF* (*condition*) *USE* || *statements_section* || *{ELSIF* (*condition*) || *statements_section* ||*}* *[ELSE* || *statement_ section* ||*]* *END USE;*	Simultaneous *IF*, instruction. With this instruction it is possible to check the conditions permanently and to execute the instructions simultaneously. This isn't possible with the *IF-THEN* instruction since these must always be in a *PROCESS*.
IN	*SIGNAL name: IN type*	*SIGNAL name* can only be read.
INOUT	*SIGNAL name: INOUT type*	*SIGNAL name* can be read and written
INTEG	*quantityname ' INTEG*	Result is the integral of *quantityname* from the simulation beginning to the current simulation time as a floating point number (*REAL*)
INTEGER	*Name: INTEGER;*	Declaration of variable in the *INTEGER* format
LIBRARY	*LIBRARY libraryname*	Including a *LIBRARY*
MAP	*PORT MAP* (*Node1=>Port1,* *Node2=> Port2);*	Assign the interfaces of a component (*Portname1*, *Portname2*) to the external signals at instantiation. The order must

	shorter: PORT MAP (*Nodename1, Nodename2*);	match the interface description if using the short variant. See *ENTITY*
NAND	*a NAND b*	Logical *NAND*
NOR	*a NOR b*	Logical *NOR*
NOT	*a NOT b*	Logical negation
NOW	*NOW*	Gives back the current simulation time as a floating point number (*REAL*)
OR	*a OR*	Logical *OR*
OUT	*SIGNAL name: OUT type;*	*SIGNAL* name can't be read, can only be written
PORT	*PORT* (|| variables-/ signals decl. ||);	Declaration part of the interfaces in an interface description (see 1.2.1, *ENTITY*)
PROCESS	*PROCESS* (|| *signal_list* ||) *BEGIN* || *statements section* || *END PROCESS*; *BREAK ON [signal_name;]*	Defines statements section, where instructions are executed sequentially. The *PROCESS* only starts if a *SIGNAL* given in the signal list changes and then runs exactly once. It is possible to stop the process earlier by assigning signal in the *BREAK ON* statement. If this signal changes the process will be ended.
QUANTITY	*QUANTITY potential*, *ACROSS* *River [THROUGH*] *terminal1 [*TO *terminal2;]*	Declare the variables potential and flux in the declaration part of a behaviour description: Potential is the node voltage between *terminal1* and *terminal2*. Flux means the current that flows into the *terminal1* (see 1.2.2)
REAL	*Name: REAL;*	Declaration of variables in the floating point format

SIGNAL	*SIGNAL name: Type;*	Declaration of a signal. Signals can represent both interfaces and internal signals. They can be unidirectional or bidirectional (see *OUT*, *INOUT)*.
TERMINAL	*TERMINAL Name1,Name2, ...: ELECTRICAL* ;	Declaration of an analogous interface to the outside in the declaration part (see *PORT*) of the type *ELECTRICAL.*
TRANSPORTATION	< = [*TRANSPORTATION*] sig_name *FOR value1 [AFTER time_1] { valuen [AFTER time_N,] };*	Switch into the "*TRANSPORTATION*" delay model. (see B.1)
USE	*USE Libraryname.Packagename.Elementname;*	Specify the used library elements
XOR	*a XOR b*	Logical XOR

APPENDIX B - VHDL-AMS OPERATORS

B.1 ASSIGNMENT OPERATORS

Signal assignments of the type signal:

sig_name <= [*TRANSPORT*] *value1*
[AFTER time_1]
{valuen [AFTER time_n]};

Assigning the values *value1..valuen* to the signal *sig_name* one by one. The so-called *'INERTIAL'* delay model is used per default. This means that the value1 is assigned to the signal if time_1 has passed after simulation start. value2 is assigned at time_2 after simulation start etc.
If keyword *TRANSPORT* is used then the simulation tool switches in the 'transport' delay model. In this model the time expressions refer to the respectively previous assignment. So value1 is assigned at *time_1*, *value2* is assigned if *time_2* has passed after *value1* was assigned and so on. For the *value1* to *valuen* you may also use signals of the same type instead of constant values.

Signal assignments of the type quantity:
q_name == expression;
The *QUANTITY q_name* assigns the value which expression returns.

B.2 ARITHMETICAL OPERATORS

+	Additon
-	Subtraction
*	Multiplication
/	Division
**	Exponent
MOD	Modulo operator

Usage:

sig_name < = *a* +*b*;

q_name == *a*b*;

Value must have the same number format like the right numerical value with the highest precision. This means if *a* has the type *INTEGER* and *b* is of the type *REAL* then left value must be of the type real!

B.3 LOGICAL OPERATORS

See APPENDIX A under *and, or, xor, nor, not, nand*.
Using only in connection with truth values or signals of the type *BIT*.

B.4 RELATIONAL OPERATORS

<	Greater than
>	Smaller than
=	Equality
>=	Greater or equal
<=	Smaller or equal
/=	Inequality

All operators require both operands to be in the same numerical format.

APPENDIX C BIBLIOGRAPHY

[1] Kreß, Dieter; Irmer, Ralf: Angewandte Systemtheorie – kontinuierliche und zeitdiskrete Signalverarbeitung; 1. Auflage; Verlag Technik, Berlin, 1989

[2] IEEE: IEEE standard VHDL Analog and Mixed signal extension; New York, 1999