

## APPENDIX A - VHDL-AMS KEYWORDS

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

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

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

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

## APPENDIX B - VHDL-AMS OPERATORS

### B.1 ASSIGNMENT OPERATORS

Signal assignments of the type signal:

```
sig_name <= [TRANSPORT] value1  
[AFTER time_1]  
{value_n [AFTER time_n]};
```

Assigning the values *value1..value\_n* 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 *value\_n* 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

+     Addition  
-     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