VHDL-AMS Quick Reference

Mentor Graphics web site:
www.mentor.com

Mentor Graphics Support:
North America
Phone: 1-800-547-4303
www.mentor.com/supportnet
Worldwide
www.mentor.com/supportnet/support_offices.html

IEEE VHDL-AMS 1076.1 is the premier industry standard mixed-signal high-level description language for electronic and multi-domain systems. This short reference describes the principal features of the language. Complete information is contained in the IEEE publication *IEEE Standard VHDL Analog and Mixed-Signal Extensions*, ISBN 0-7381-1640-8, from which this summary is derived.

The left hand column of each table is a general description and the right hand column contains explanations or examples. The following syntax conventions are used:

entity	reserved words	operand {operation operand}	repeated items
[expression]	optional item	identifier { , . . . }	repeated items (dot notation)
letter | digit	alternative selection	::=	production rule

Design Units

[context_clause]	**library** DISCIPLINES; **use** DISCIPLINES.ELECTRICAL_SYSTEMS.**all**;
entity entity_name is [**generic** (parameter_list);] [port (port_list) ;] {declaration_part} [**begin** {passive_concurrent_statement}] **end [entity]** [entity_name] ;	**entity** comparator is **generic** (level : REAL := 2.5); **port** (**terminal** a, ref : ELECTRICAL; **signal** d : **out** BIT); **begin** **assert** level > 0.0 **report** "Level must be > 0.0" **severity** ERROR; **end entity** comparator;

[context_clause]
architecture architecture_name of
entity_name is
{declaration _part}
begin
{simultaneous_statement |
concurrent_statement} }
end[architecture][architecture_name] ;
	architecture simple **of** comparator **is**
	
	quantity v **across** i **through** a **to** ref;
	begin
	v == 1.0E6*i;
	d <= '1' **when** v'ABOVE(level) **else** '0';
	end architecture simple ;

[context_clause]
package package_name is
declaration { , ...}
 	
end [package] [package_name] ;
 	
[context_clause]
package body package_name is
declaration { ,...}
end [packagebody] [package_name] ;
	package my_functions **is**
	function boolean2bit (b: BOOLEAN)
	return BIT;
	end package my_functions;
	
	package body my_functions **is**
	function boolean2bit (b:BOOLEAN) **return**
	BIT **is**
	begin
	if b= TRUE **then return**'1';
	else return '0';
	endif;
	end function boolean2bit;
	end package my_functions;

context_clause ::= { **library** library_name_list ; } { **use** selected_name {, selected_name}; } selected_name ::= library_name. package_name. item_name | library_name. package_name.**all** | library_name. item_name | library_name.**all**	**library** IEEE; **use** IEEE.STD_LOGIC_1164.**all**; --implicit context clause at the head of any design unit: **library** STD, WORK; **use** STD.STANDARD.**all**;

Simultaneous Statements

[label :] expression == expression [tolerance_aspect];	v == res_value * i ; f == m*x'DOT'DOT + k*x **tolerance** "mechanical_mst";
[label :] **if** condition **use** { simultaneous_statement } { **elsif** condition **use** { simultaneous_statement } } [**else** { simultaneous_statement }] **end use** [label] ;	**if** vmax = REAL'RIGHT **use** verr== vin ; **elsif** vin > vmax **use** verr== vmax ; **elsif** vin < -vmax **use** verr== -vmax ; **else** verr == vin ; **end use** ;
[label :] **case** expression **use** **when** choice { | choice } => { simultaneous_statement } { **when** choice { | choice } => { simultaneous_statement } } **end case** [label] ;	**case** din **use** **when** '0' => v == ron * i + vlo; **when** '1' => v == ron * i + vhi; **when** 'X'' => v == ron * i + vx; **when** 'Z' => v == roff * i + vx; **end case** ;

[label :] **procedural** [**is**]
{ declaration_part }
begin
{ sequential_statement }
end procedural [label] ;
	procedural is
	variable sum := 0.0 ;
	begin
	for i **in** inp'RANGE **loop**
	sum := sum + inp(i);
	end loop;
	outp := sum;
	end procedural ;

Concurrent Statements

[process_label :]
process [(signal_name { , ...})] [**is**]
{ process_declaration }
begin
{ sequential_declaration }
end process [process_label] ;
	p1a : **process** (a, b)
	begin
	s <= a **xor** b;
	end process p1a;
	
	p1b : **process is**
	begin
	s <= a **xor** b;
	wait on a, b;
	end process p1b;

[label :] procedure_name [(actual_parameter_list)] ;	check_up(clk=>phi, data=>enable);
[label :] **assert** boolean_expression [**report** string_expression] [**severity** severity_level] ;	**assert** not (res = '1' and eoc = '1') **report** "res and eoc can't be 1 at the same time" **severity** WARNING ;
[label :] signal_name < = waveform_element [, waveform_element];	y1 <= **not** a ; y2 <= a **after** 5 ns ; y3 <= **0.0, 1.0 after** 250 ns ;

with expression **select** signal_name < = [delay_mechanism] { waveform **when** choice { | choice} , } waveform **when** choice { | choice} ;	**with** sel **select** z>=d0 **when** "00" | "11", d1 **when others**;
label : **entity** entity_name [(architecture_name)] [**generic map** (generic_association_list)] [**port map** (port_association_list)] ;	res1 : **entity** resistor (simple) **generic map** (res_value=> 1.0E3) **port map** (p => t1, m => t2);
label : [**component**] component_name [**generic map** (generic_association_list)] [**port map** (port_association_list)] ;	C : flipflop **generic map** (20 ns) **port map** (in_1, in_2, out_1, out_2) ;
[label :] **break** [break_list] [sensitivity_list] [**when** condition] ;	**break on** s **when** Q > 2.0 ;

Sequential Statements

[label :] **wait** [**on** signal_list] [**until** condition] [**for** time_expression] ;	**wait until** clk = '1' **for** 1.2 ns;
[label :] **assert** boolean_expression [**report** string_expression] [**severity** severity_level] ;	**assert** res_value >= 100.0 **report** "res_value is too small" **severity** ERROR ;
[label :] **report** string_expression [**severity** severity_level] ;	**report** "Entering function f1" ;
[label :] signal_name <= waveform_element {, waveform_element};	y1 <= **not** a ; y2 <= '1', '0' **after** ions ;
[label :] variable_name := expression ;	count := count + 1 ;
[label :] procedure_name [(actual_parameter_list)] ;	report_max_and_sum (samples) ;
[label :] **if** condition **then** { sequential_statement } { **elsif** condition **then** { sequential_statement } } [**else** { sequential_statement }] **end if** [label] ;	max_ab : **if** a > b **then** vmax := a ; **else** vmax := b ; **end if** max_ab ;
[label :] **case** expression is **when** choice { | choice } => { sequential_statement } { **when** choice { | choice } => { sequential_statement } } **end case** [label] ;	**case** int **is** **when** 0 => **null**; **when** 1 | 2 | 7 => v := 6; **when** 3 to 6 => v := 8; **when others** => v := 0; **end case**;
[label :] **loop** { sequential_statement } **end loop** [label] ;	L : **loop** **wait until** clk = '1'; q <= d **after** 5 ns ; **exit** L **when** NOW > 100 ms ; **end loop** L;
[label :] while condition loop { sequential_statement } end loop [label] ;	**while** mpier > 0.0 **loop** prod : = prod * mpcand ; mpier : = mpier - 1.0 ; **end loop**;

[label :] **for** identifier **in** range **loop** { sequential_statement } **end loop** [label] ;	**for** i **in** 15 **downto** 0 **loop** vector(i) := i * 2.0; **end loop** ;
[label :] **break** [break_list] [**when** condition] ;	**break**;

Object Declarations

constant name_list : type_or_subtype_name := expression ;	**constant** clk_period : TIME := 20 ns; **constant** data : BIT_VECTOR:= B"1010_0010" ; **constant** coeff : REAL_VECTOR := (2.0, 3.1);
[shared] **variable** name_list : type_or_subtype_name [:= expression] ;	**variable** sum : REAL := 0.0 ; **variable** state_a, state_b : BOOLEAN ;
signal name_list : type_or_subtype_name [:= expression] ;	signal s : BIT_VECTOR (15 **downto** 0) ; signal clk : BIT := '1'; signal reset, strobe, enable: **BOOLEAN**;
file name_list : file_type [[open open_kind] is file_logical_name];	**file** file4 :TEXT **open**WRITE_MODE **is** "intdata";
terminal name_list: nature_name;	**terminal** t1, t2 : ELECTRICAL ;
quantity name_list : real_type_name [:= expression] ;	**quantity** q1, q2 : REAL ; **quantity** qv : REAL_VECTOR (1 **to** 4) := (2=>1.0, others => 0.0);
quantity [across_aspect] [through_aspect] terminal_aspect ; across_aspect ::= name_list [:= expression] **across** through_aspect ::= name_list [:= expression] **through** terminal_aspect ::= plus_terminal_name [**to** minus_terminal_name]	**quantity** v **across** i **through** a to b; **quantity** vin **across** inp **to** ref; **quantity** vctrl **across** inp_ctrl; **quantity** vd **across** id, ic **through** anode **to** cathode;
quantity name_list : real_type_name **spectrum** magnitude_expr, phase_expr ;	**quantity** ac: REAL **spectrum** 1.0, 90.0; **quantity** ac_in: REAL **spectrum** 1.0, 0.0;
quantity name_list : real_type_name **noise** power_expr ;	**quantity** fl_noise_pow : REAL **noise** kf*id**af/frequency ;
limit quantity_list : quantity_type **with** real_expression;	**limit** v : VOLTAGE **with** 0.05/cf;

Component Declaration

component identifier [**is**] [**generic** (generic_list) ;] [**port** (port_list) ;] **end component** [identifier] ;	**component** flipflop **generic** (JtoQ_delay : TIME) ; **port** (j, k : **in** BIT ;q, q_bar : **out** BIT); **end component** ;

Interface Declarations (Ports)

Interface_signal_declaration ::= [**signal**] name_list : [**in** | **out** | **inout** | **buffer**] type_name [:= static_expression]	**signal** clk, d: **in** BIT;
Interface_terminal_declaration ::= **terminal** name_list : nature_name	**terminal** p, m : ELECTRICAL;
Interface_quantity_declaration ::= **quantity** name_list : [**in** | **out**] real_subtype_name [:= static_expression]	**quantity** qi : in REAL;
port (interface_declaration { ; interface_declaration })	**port** (**signal** clk, d: **in** BIT ; **terminal** p, m : ELECTRICAL ; **quantity** qi : **in** REAL)

Subprograms

procedure procedure_name
[(formal_parameter_list)] is
{ declaration_part }
begin
{ sequential_part }
end [procedure] [procedure_name] ;
	procedure check_setup (**signal** clk, data : **in** BIT) is
	begin
	if clk = '1' **then**
	assert data' LAST_EVENT >= 3 ns
	report "Setup time violation" **severity** NOTE;
	end if;
	end procedure check_setup ;

[**pure** | **impure**] **function** function_name **return** type | subtype **is** { declaration_part } **begin** { sequential_part } **end** [**function**] [function_name] ;	**function** f1(x1, x2, x3 : BIT) **return** BIT **is** **variable** y : BIT; **begin** y := ((**not** x1) **or** (x2 **and** x3)) ; **return** y ; **end function** f1 ;

Type and Subtype Declarations

type type_name **is** type_definition ; type_definition ::= scalar_type_definition |	**type** INTEGER **is range** -2147483648 **to** 2147483647; **type** REAL **is range** -1.0E38 **to** 1.0E38; **type** word_index **is range** 31 **downto** 0 ;
	type mstate **is** (initial, active, reset) ;
array_type_definition |	**type** BIT_VECTOR **is array** (NATURAL **range** <>) **of** BIT; **type** word **is** array (31 **downto** 0) **of** BIT; **type** truth_table **is** array (BIT, BIT) **of** BIT;
record_type_definition |	**type** COMPLEX **is** record RE, IM: REAL ; **end record** COMPLEX;
access_type_definition | file_type_definition	**type** word_index_ptr **isaccess** word_index; **type** TEXT **is file of** STRING ;
subtype subtype_name **is** type_name [constraint] [tolerance_aspect] ;	**subtype** byte_index **is** INTEGER **range** 0 **to** 7 ; **subtype** VOLTAGE **is** REAL **tolerance** "default_voltage" ;

Nature Declarations

nature scalar_nature_name **is** type_name **across** type_name **through** reference_node_name **reference** ; **nature** array_nature_name **is** **array** (index_range) **of** nature_name ; subnature_declaration : := subnature identifier **is** nature_mark [index_constraint];	**nature** ELECTRICAL **is** VOLTAGE **across** CURRENT **through** ELECTRICAL_REF **reference**; **nature** ELECTRICAL_VECTOR **is** **array** (NATURAL **range**<>) **of** ELECTRICAL ; **subnature** ev12 **is** electrical_vector (11 **downto** 0);

Operators

+ - * / mod rem abs ** = /= < <= > >=	arithmetic operations for integer valued objects relational operations for integer valued objects
+ - * / abs ** = /= < <= > >=	arithmetic operations for real valued objects relational operations for real valued objects
+ - * / abs = /= < <= > >=	arithmetic operations for objects of type TIME relational operations for objects of type TIME
= /= < <= > >= **and or nand nor xor xnor not**	relational operations for objects of an enumerated type logical operations for objects of types BIT and BOOLEAN
sll srl sla sra rol ror **&**	shift and rotate operations for one dimensional arrays of base types BIT and BOOLEAN concatenation for one dimensional arrays

Increasing Precedence of Operators

and or nand nor xor xnor = /= < <= > >= **sll srl sla rol ror** **+ - &** **+ -*** **/ mod rem** **** abs not**	logical operations relational operations shift and rotate operations adding sign (unary) multiplying miscellaneous

Attributes of Types

type_name'LOW type_name'HIGH type_name'LEFT type_name'RIGHT type_name'POS(x) type_name'PRED(x) type_name'SUCC(x) type_name'VAL(n) type_name'VALUE(s) type_name'IMAGE(x) real_type_name'TOLERANCE	least value greatest value leftmost value rightmost value position of x in type value at the postion one less than x value at the postion one greater than x value at postion n in type value in type represented by the string s string representation of x in type tolerance group of real_ type_name (string)

Attributes of Arrays

array_name'LOW [(n)]	least value of [n th] index
array_name'HIGH [(n)]	greatest value of [n th] index
array_name'LEFT [(n)]	leftmost value of [n th] index
array_name'RIGHT [(n)]	rightmost value of [n th] index
array_name'RANGE [(n)]	index range of array ([n th] index)
array_name'REVERSE_RANGE [(n)]	index range of array ([n th] index) in reversed direction
array_name'ASCENDING [(n)]	TRUE, when index range ([n th] index) is ascending
array_name'LENGTH [(n)]	number of elements in index range of array ([n th] index)

Attributes of Signals

signal_name'DELAYED [(T)]	implicit signal delayed by time T
signal_name'STABLE [(T)]	implicit boolean signal (TRUE, when no event for time T)
signal_name'EVENT	boolean value (TRUE if event in current simulation cycle)
signal_name'LAST_EVENT	time value equal to the time since the last event occurred
signal_name'LAST_VALUE	previous value of the signal before the last change
real_signal_name'RAMP (trise, tfall)	implicit quantity follows the signal with given times
real_signal_name'SLEW (up, down)	implicit quantity follows the signal with given slopes

Attributes of Quantities

quantity_name'DOT	implicit quantity; derivative w.r.t. of quantity_name
quantity_name'INTEG	implicit quantity; integral of quantity_name
quantity_name'DELAYED [(T)]	implicit quantity; quantity_name delayed by T
quantity_name'SLEW [(maxrise [,maxfall])]	implicit quantity following quantity_name with slope
quantity_name'LTF (num , den)	implicit quantity that denotes Laplace transfer function
quantity_name'ZOH (Tsampl [, init_delay])	implicit quantity that implements basic sample-and-hold
quantity_name'ZTF (num, den, t [,init_del])	implicit quantity that denotes z-domain transferfunction
quantity_name'ABOVE (level)	boolean signal; TRUE, when quantity_name > level
quantity_name'TOLERANCE	string with tolerance group of quantity_name

Attributes of Natures and Terminal

nature_name'across	type of the across aspect of the nature
nature_name'through	type of the through aspect of the nature
terminal_name'reference	implicit reference across branch quantity of the terminal
terminal_name'across	implicit contribution through branch quantity

Miscellaneous

now	returns current simulation time (TIME or REAL)
frequency	returns current frequency in frequency analysis
DOMAIN possible values: QUIESCENT_DOMAIN, TIME_DOMAIN, FREQUENCY_DOMAIN	signal returns current analysis algorithm
[label:] **null** ;	statement without an action

label: **for** parameter_specification **generate** [{block_declarative_item} **begin**] { simultaneous_statement | concurrent_statement } **end generate** [label] ;	c : **for** i **in** 1 **to** 4 **generate** q(i)'DOT + a(i)*q(i)==q(i-1); **end generate** ;
- - comment	comment starts with "--" and goes until end of line
identifier ::= letter {[underline] alphanumeric }	base_type_1
BOOLEAN, BIT, CHARACTER, SEVERITY_LEVEL, INTEGER, NATURAL, POSITIVE, REAL, TIME, DELAY_LENGTH, STRING, BIT_VECTOR, REAL_VECTOR	predefined types in package STANDARD of library STD
LINE, TEXT	predefined types in package TEXTIO of library STD
STD_LOGIC	defined in package STD_LOGIC_1164 of library IEEE
mathematical functions	defined in packages MATH_REAL and MATH_COMPLEX of library IEEE
predefined natures	in packages of library DISCIPLINES and IEEE