SynthWorks

VHDL Training Experts

VHDL Types and Operators Quick Reference

1. Packages & Libraries

Usage	Abbr.	Source
Library **ieee** ;		
use std.**standard**.all ; -- *	std	IEEE
ieee.**std_logic_1164**.all ;	1164	IEEE
use ieee.**numeric_std**.all ;	ns	IEEE
use ieee.**numeric_std_unsigned**.all;	nsu	IEEE
use ieee.**std_logic_arith**.all ;	sla	Shareware
use ieee.**std_logic_unsigned**.all ;	slu	Shareware
use std.**textio**.all ;	textio	IEEE
use ieee.**std_logic_textio**.all ;	-	Shareware

VHDL-2008 adds packages for fixed and floating point.

libraries work and std are implicitly referenced
* package std.standard is implicitly referenced

Packages are augmented with tool specific directives, hence, only use packages provided by your tool vendor.

2. Values of std_ulogic, std_logic

'U'	Uninitialized, default value at elaboration
'X'	Unknown / Synthesis also Don't Care
'0'	Logic 0 / Driven
'1'	Logic 1 / Driven
'Z'	Tristate / High Impedance
'W'	Resistive Unknown
'L'	Pull Down / Weak 0
'H'	Pull-up / Weak 1
'-'	Don't care

3. Common Types

Type / Abbreviation	Value	Origin
std_ulogic / sul	Base Type	1164
std_ulogic_vector / sulv	array of std_ulogic	1164
std_logic / sl	resolved std_ulogic	1164
std_logic_vector / slv	array of std_logic	1164
signed / sv	array of std_logic	ns, sla
unsigned / uv	array of std_logic	ns, sla
integer / int	$-(2^{31} - 1)$ to $2^{31} - 1$	std
natural / int0+	0 to $2^{31} - 1$	std
line	access string	textio

Enumerated   **type** StateType **is** (S0, S1, S2, S3) ;

4. Assigning Values

```
A_sl    <= '1' ;       -- Character literal
B_slv  <= "1111" ; -- string literal
C_slv  <= X"F";        -- hex.  4 bits per character
D_slv5 <= '1' & X"F"; -- 5 bit object
E_slv5 <= 5X"1F" ; -- VHDL-2008, 5 bit object
F_slv  <= (others => '1') ; -- aggregate
G_slv(0) <= '1' ; -- index
H_slv(0 to 1) <= "01" ; -- slice
L_int   <= 15 ;        -- universal integer
M_int   <= 16#F# ;    -- base literal (16 = base)
N_bool <= TRUE ;      -- boolean only true or false
```

5. Common Attributes

signal A : unsigned(7 downto 0) ;

Array Attribute	Result	Value / Meaning
A'**length**	value	8
A'**range**	range	7 downto 0
A'**reverse_range**	range	0 to 7
A'**left**	value	7
A'**right**	value	0

Signal Attribute	Result	Value / Meaning
S'**event**	bool	True if signal changed

6. VHDL Operators

Logic	and, or, nand, nor, xor, xnor
Comp	=, /=, <, <=, >, >=
Shift	*sll, srl, sla, sra, rol, ror*
Add	+, -
Sign	+, -
Mult	*, /, mod, rem
Misc	**, abs, not, *and, or, nand, nor, xor, xnor*

Precedence increases from logic to misc. Underlined items are VHDL-2008.

7. Strong Typing and Assignments

Size and type of target (right) must equal size and type of expression (left). For the array-based types, each operation has a specific sized result. VHDL operators allow multiple implementations for different types (overloading).

8. Logic operators = Logic Gates

Separate operators with parentheses when using different operators (except not) or non-associative operators (nand, nor).

Legal:	Illegal:
Z <= A and B and C ;	
Z <= (A and B) or C ;	**Z <= A and B or C ;**
Z <= (A nand B) nand C ;	**Z <= A nand B nand C ;**
Z <= not A and B ;	
Z <= (not A) and B ;	

Array must be the same length. Arrays are handled bit-wise from left to right, independent of indices.

Overloading

Left*	Right	Return	Package	Notes
bool	bool	bool	std	
sul	sul	sul	1164	1 also sl
slv	slv	slv	1164	
sulv	sulv	sulv	1164	
uv	uv	uv	ns	not in sla
sv	sv	sv	ns	not in sla

* Not operator only has a right value.

9. General Numeric Overloading

The general sense of overloading for Comparison, Addition, and Multiplication is summarized below. The arrays must be the same type (either uv, sv, or slv).

Left	Right	Return	Package	Notes
Array	Array	Array	ns, sla, slu	2
Array	int	Array	ns, sla, slu	2, 3
int	Array	Array	ns, sla, slu	2, 3

10. Comparison

10.1 Comparisons Return Boolean

Use a conditional to create an appropriate value:

   Z <= '1' when (A > B) else '0' ;

10.2 Comparisons only match identical values

Hence, comparing to '-' is **invalid**:

-- Dec1 <= '1' when Addr = "11---0" else '0'; --illegal

Instead split up the expression as follows:

**Dec1 <=   '1' when (Addr(5 downto 4) = "11" and
                  Addr(0) = '0') else '0';**

10.3 Overloaded Comparison Operators

When using packages ns. sla, and slu, array comparisons are treated numerically.

10.4 * Implicit Definitions

In addition to the general numeric overloading for sv, uv, and slv, the following overloading also exists:

Left	Right	Return	Package	Notes
sul	sul	bool	implicit	1, *
sulv	sulv	bool	implicit	*
slv	slv	bool	implicit	*

For the ordering operators **(<, <=, >, >=),** the left most type values are smaller than right most values. Hence, implicitly for std_logic, **'0' < '1' < 'L' < 'H'**

Use type unsigned or signed, package slu, or functions TO_X01 to avoid this issue.

10.5 Std_Match = comparison with don't care

Std_match does comparisons and understands '-'. It is in package NS and is supported for sul, slv, sulv, uv, sv.

Dec1 <= '1' when std_match(Addr,"11---0") else '0';

11. Shift Operators

With VHDL-2008, shift operators are implemented for std_logic family of types.

```
Y_uv  <= A_uv sll 1 ;
Y_slv <= A(6 downto 0) & '0'; --SLL 1
Z_uv  <= A_uv srl 1 ;
Z_slv <= '0' & A(7 downto 1); --SRL 1
SSR   <= A(7) & A(7 downto 1); --SRA 1
SIR   <= SI_sl & A(7 downto 1); -- Shift In
```

12. Addition

For addition and subtraction, always use the + and - operators (never gate logic). The size of the result equals the size of the largest array input. A short input array is extended appropriately for the type.

Overloading, beyond general numeric overloading

Left	Right	Return	Package	Notes
array	sul	array	sla, **_ns, nsu_**	1
sul	array	array	sla, **_ns, nsu_**	1

Facilitates Add with Carry or counter with count enable:

```
Y_uv    <= A_uv + B_uv + Cin_sl ;
IncVal  <= IncReg + CountEn ;
```

13. Multiplication

NS supports the general numeric overloading. SLA and slu only support array with array. For array with array, the size of result equals the sum of the size of the two array inputs. Array with integer is not useful since the size of the result is 2X the array input.

14. Misc Operators: - abs

Function	In	Rtn	Pkg	Notes
-	sv	sv	ns , sla	2
abs	sv	sv	ns , sla	2

15. Summary of Result Sizes

Operation	Result Size
"10101010"	number of bits in array
X"AA"	4 x number of characters
A_slv8	A_slv8'length
C5 & D3	C5'length + D3'length
A8 and B8	D8'length (= E8'length)
A8 > B8	Boolean
A8 > 10	Boolean
A8 + B8	maximum(A8'length, B8'length)
A8 + 10	A8'length
A8 * B8	A8'length + B8'length
A8 * 10	2 * A8'length

16. Notes for Overloaded Operators

16.1 std_ulogic

Since std_logic automatically converts to std_ulogic, there is no overloading for std_logic.

16.2 numeric_std and std_logic_arith

Do not use together in the same entity / package.

16.3 Use Integer Literals

```
Z_uv <= A_uv + 16#5F# ;
```

16.4 SLA: Signed & Unsigned Arguments

Std_logic_arith overloads addition, comparison, and multiplication to allow signed and unsigned operands in the same expression. Requires type qualifiers when using string literals. Does not apply to NS.

```
Z_uv <= A_uv + unsigned'("0101") ;
```

16.5 SLA: Std_logic_vector Return Values

sla overloads functions to return slv as well as unsigned or signed. Requires type qualifiers with sla and slu in the case shown below. Does not apply to NS and slu.

```
Z_slv  <= signed'(A_sv + B_sv) +
          signed'(C_sv + D_sv);
```

17. Ambiguous Expressions

A statement is ambiguous if more than one operator symbol or function can match its arguments. VHDL **type qualifiers** (**TypeName'**) are a mechanism that specifies the type of an argument or return value of a subprogram. See notes 15.4 and 15.5.

18. Conversions

18.1 Std_ulogic <=> Signed(i), Unsigned(i)

Two objects that are subtypes of the same type covert automatically and do not need a conversion function:

```
A_sul      <= B_sl ;
C_sl       <= D_sv(1) ;
E_uv(1)    <= F_sl ;
G_sv(1)    <= H_uv(1) ;
```

18.2 Signed, Unsigned <=> Std_logic_vector

Arrays with a common base type and indices that have a common base type can be converted by type casting:

```
A_slv      <= std_logic_vector( B_uv ) ;
C_slv      <= std_logic_vector( D_sv ) ;
G_uv       <= unsigned( H_slv ) ;
J_sv       <= signed( K_slv ) ;
```

18.3 Unsigned, Signed <=> Integer

Converting between either signed or unsigned and integer requires a conversion function:

Function	In	Rtn	Pkg
to_integer(val)	uv	int0+	ns
to_integer(val)	sv	int	ns
to_integer(val)	slv	int	_nsu_
to_unsigned(val, len)	int0+, int0+	uv	ns
to_signed(val, len)	int, int0+	sv	ns
to_slv(val, len)	int0+, int0+	slv	_nsu_
conv_integer(val)	uv	int	sla
conv_integer(val)	sv	int	sla
conv_unsigned(val, len)	int, int	uv	sla
conv_signed(val, len)	int, int	sv	sla

18.4 std_logic_vector <=> Integer

```
A_slv <= std_logic_vector( to_unsigned(B_int, 8) ) ;
C_int <= to_integer(unsigned(D_slv)) ;
```

Alternately use conversions in Numeric_std_unsigned.

18.5 Resizing Arrays

Resizing can be done manually, with operator overloading (when only one operand is smaller), or with the following functions:

Function	In	Rtn	Pkg
resize(val, len)	uv, int0+	uv	ns
resize(val, len)	sv, int0+	sv	ns
conv_unsigned(val, len)	uv, int	uv	sla
conv_signed(val, len)	uv, int	sv	sla

19. Strength Strippers

Strength strippers are used in behavioral models and IO pads to map values of a type to a simpler set (X, 0, 1).

Strength strippers **TO_X01**, **TO_X01Z**, **TO_UX01** convert the input values to X01, X01Z, and UX01 respectively. Maps: 'H' to '1', 'L' to '0' and others to 'X'.

In	Return	Package
sl	sl	1164
array	array	1164, ns

With VHDL-2008, array is slv, sulv, sv, uv, …

Strength stripper TO_01 maps: 'H' to '1', 'L' to '0' and others to XMAP value.

Function	In	Rtn	Pkg
to_01 (s, xmap := '0')	array, sl	array	ns, 1164

With VHDL-2008, array is slv, sulv, sv, uv, …

20. X detection

The function **is_X** detects 'X' on inputs of models.

In	Return	Package
sl	bool	1164
array	bool	1164, ns

With VHDL-2008, array is slv, sulv, sv, uv, …

21. Edge detection

Functions **rising_edge** and **falling_edge**.

In	Return	Package
sl	bool	1164

SynthWorks Design Inc.
VHDL Hardware Synthesis and Verification Training
11898 SW 128th Ave. Tigard OR 97223   (800)-505-8435
http://www.SynthWorks.com   jim@synthworks.com