

## VHDL Quick Reference

### 1. Entity + RTL Architecture

A design in VHDL is created with an entity and an architecture. The entity describes the IO of the design. The architecture describes the implementation.

```
-- Library + Use = similar to ".h" in C
library ieee ;
use ieee.std_logic_1164.all ;

-- Entity Declaration = IO
entity MuxReg is
port (
    Clk : In std_logic ;
    Sel : In std_logic ;
    A : In std_logic_vector(7 downto 0) ;
    B : In std_logic_vector(7 downto 0) ;
    Y : Out std_logic_vector(7 downto 0)
);
end MuxReg ;

-- Architecture = Implementation
architecture RTL of MuxReg is
    signal Mux :
        std_logic_vector(7 downto 0);
begin
    Mux <= A when (Sel = '0') else B ;
    RegisterProc : process (Clk)
    begin
        if (Clk = '1' and Clk'event) then
            Y <= Mux ;
        end if ;
    end process ;
end RTL ;
```

A design is created with an entity and architecture pair. The architecture is associated with an entity by specifying the entity name in the architecture declaration. As a result, the architecture does not need to be in the same file as an entity.

The architecture shown above describes hardware. By convention we label these architectures RTL. You will note that this is only a convention. Every entity in a design may have an architecture named RTL.

### 2. Structural Architecture

An architecture that describes interconnections of lower level blocks is called a structural design or a netlist. In order to create a netlist, there must be an entity and architecture compiled into a library that corresponds to each instance (component or subblock) in a design.

A structural design has three pieces: signal declarations which describe connectivity between blocks in a design, component instantiations which map signals, and component declarations which declare the interface to an entity (redundant for many uses, but required for synthesizable designs).

#### architecture Structural of MuxReg is

```
-- Signal Declarations
signal MuxOut :
    std_logic_vector(7 downto 0);

-- Component Declarations
component Mux8x2
port (
    Sel : In std_logic ;
    I0, I1 : In
        std_logic_vector(7 downto 0);
    Y : Out
        std_logic_vector(7 downto 0)
);
end component ;
component Reg8
port (
    Clk : In std_logic ;
    D : In
        std_logic_vector(7 downto 0);
    Q : Out
        std_logic_vector(7 downto 0)
);
end component ;

-- Component Instantiations
U_Mux : Mux8x2
port map (
    Sel => Sel,
    I0 => A,
    I1 => B,
    Y => MuxOut
);

-- Positional Association
U_Reg : Reg8
port map (Clk, MuxOut, Y);

end Structural ;
```

### 3. Common Synthesizable Types

Type / Abbreviation	Value	Origin
std_logic / sl	U X 0 1 Z W L H -	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
boolean / bool	(False, True)	std
integer / int	-(2 <sup>31</sup> - 1) to 2 <sup>31</sup> - 1	std

See VHDL Operators and Types Quick Reference for details on packages, types and operators.

### 4. Assigning Values

A\_sl <= '1' ; -- Character literal  
 B\_slv <= "1111" ; -- string literal  
 C\_slv <= X"F" ; -- X = hex. \*\*VHDL-93 Only  
 D\_int <= 15 ; -- universal integer  
 E\_int <= 16#F# ; -- base literal (16 = base)

### 5. VHDL Operators

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

Precedence increases from logic to misc (except not which is same precedence as Misc operators). Underlined items are VHDL-93.

### 6. Concurrent Statements

At the core of VHDL are its concurrent statements that are in used directly in the architecture. Concurrent statements executed due to signals changing in an expression or input list of a component instantiation.

#### 6.1 Simple Signal Assignment

Creates logic or wires:

```
Z <= Address ;
Sel <= Sel1 and Sel2 ;
Y <= A(6 downto 0) & '0' ; --Shift Rt
```

#### 6.2 Conditional Signal Assignment

Concurrent if statement.

```
Mux2 <=
    A when (Sel1 = '1' and Sel2 = '1')
    else B or C ;
```

Each target can be an expression rather than just a signal name. The conditional expression must be boolean. Use conditional operators as shown above to accomplish this. Also see the if statement.

### 6.3 Selected Signal Assignment

Concurrent case statement. See case statement for rules about MuxSel and use of others.

```
with MuxSel select
Mux41 <=
  A when "00",
  B when "01",
  C when "10",
  D when "11",
  'X' when others ;

6.4 Process
Concurrent container of sequential code. Must have either a sensitivity list or wait statement.

RegProc : process ( Clk, nReset)
begin
  -- Asynchronous Reset
  if (nReset = '0') then
    AReg <= '0' ;
    BReg <= '0' ;
  elsif (Clk = '1' and Clk'event) then
    AReg <= A ;
    BReg <= B ;
  end if ;
end process ;
```

### 7. Sequential Statements

Contained in processes and subprograms.

#### 7.1 Simple Signal Assignment

Expression is evaluated immediately. Value is assigned one delta cycle later. Note, conditional signal assignment and selected signal assignment are not supported in sequential statements.

```
Z <= AddReg ;
S1 <= Sel1 and Sel2 ;
```

#### 7.2 Variable Assignment

Expression is evaluated and assigned immediately.

```
MuxSel := S1 & S0 ;
```

#### 7.3 IF Statement

```
if (in1 = '1') then
  NextState <= S1 ;
  Out1 <= '1' ;
elsif (in2 = '1' and in3 = '1') then
  NextState <= S2 ;
elsif (in4 and in5) = '1' then
  NextState <= S3 ;
else
  NextState <= S4 ;
end if ;
```

An IF statement can have one or more signal assignments per branch. The conditional expression <http://www.SynthWorks.com> jim@SynthWorks.com

must be boolean. Use conditional operators as shown above to accomplish this.

#### 7.4 Case Statement

```
Mux : process (MuxSel, A, B, C, D)
begin
  case MuxSel is
  when "00" => Y <= A ;
  when "01" => Y <= B ;
  when "10" => Y <= C ;
  when "11" => Y <= D ;
  when others => Y <= 'X' ;
  end case ;
end process ;
```

A case statement can have zero or more assignments per target. Others statement must be last and is required if all conditions are not covered. For std\_logic with 9 values, others is almost always required.

The case expression must be either a signal name or a slice name. To satisfy this requirement, a variable is commonly used to from expressions.

```
Mux : process (S1, S0, A, B, C, D)
  variable MuxSel :
  std_logic_vector(1 downto 0) ;
begin
  MuxSel := S1 & S0 ;
  case MuxSel is
  when "00" => Y <= A ;
  when "01" => Y <= B ;
  when "10" => Y <= C ;
  when "11" => Y <= D ;
  when others => Y <= 'X' ;
  end case ;
end process ;
```

#### 7.5 For Loop

```
RevAProc : process(A)
begin
  for i in 0 to 7 loop
    RevA(7 - i) <= A(i) ;
  end loop ;
end process ;
```

Loop index can be any identifier and does not need to be declared. Synthesizable when loop indices are integers.

### 7.6 Wait Until and Clocks

Using wait until to create a register:

```
RegProc : process
begin
  wait until Clk = '1' ;
  AReg <= A ;
  BReg <= B ;
end process ;
```

### 7.7 Wait Until, Wait For, and Assert

Using wait until and wait for as used in a testbench:

```
TestProc : process begin
  wait until Clk = '1' ;
  Addr <= "000" after tpd_Clk_Addr ;
  wait until Clk = '1' ;
  Addr <= "001" after tpd_Clk_Addr ;
  -- and so on ...
  wait for tperiod_clk * 5 ;
  assert false
  report " = Normal Completion"
  severity failure ;
end process ;
```

### 7.8 Synchronous Reset Register

Synchronous reset is specified after the clock.

Asynchronous reset is specified before the clock (see example in process section).

```
RegProc : process (clk)
begin
  if (Clk = '1' and Clk'event) then
    if (nReset = '0') then
      AReg <= '0' ;
    else
      AReg <= A ;
    end if ;
  end if ;
end process ;
```

The wait until form of a register only supports synchronous resets.

© 2000, 2002 by SynthWorks Design Inc. All rights reserved.

**SynthWorks Design Inc.**

VHDL Training with an emphasis on  
Hardware Design and Test

11898 SW 128<sup>th</sup> Ave. Tigard OR 97223 1-(503)-590-4787  
<http://www.SynthWorks.com> jim@synthworks.com

# SynthWorks

VHDL Training Solutions for hardware design and test.

## VHDL Types and Operators Quick Reference

### 1. Packages & Libraries

Usage	Abbr.	Source
Library IEEE;		
use IEEE.std_logic_1164.all;	1164	IEEE
use IEEE.numeric_std.all;	ns	IEEE
use IEEE.std_logic_arith.all;	sla	Synopsys
use IEEE.std_logic_unsigned.all;	slu	Synopsys
use IEEE.std_logic_signed.all;	sls	Synopsys

Packages are typically optimized and augmented with meta-comments (comments with meaning) or with attributes by each tool vendor. 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. Synthesizable 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
boolean / bool	array of std_logic	std
integer / int	(False, True) <sub>2</sub> to 2 <sup>31</sup> - 1	std
natural / int0+	0 to 2 <sup>31</sup> - 1	std

### 4. Assigning Constants

A\_slv <= "1111"; -- string literal  
 B\_bitv <= X"F"; -- bit string literal (X = hex)  
 C\_slv <= X"F"; -- illegal in VHDL-87 (synthesis)  
 D\_slv <= to\_stdlogicvector(bit\_vector("X"F"));  
 E\_int <= 15; -- universal integer  
 F\_int <= 16#F#; -- base literal (16 = base)

VHDL-93 supports bit string literals for all arrays; VHDL-87 only with type bit\_vector. Use integer literals with Comp and Add operators (see overloading).

### 5. VHDL Operators

Logic not, and, or, nand, nor, xor, xnor

Comp =, /=, <, <=, >, >=

Shift sll, srl, sla, sra, rol, ror

Add +, -

Sign +, -

Mult \*, /, mod, rem

Misc \*\*, abs

Precedence increases from logic to misc. For precedence consideration, "not" is considered to be a Misc operator. Synthesis syntax is based on VHDL-87. Underlined items were added in VHDL-93 revision and are not supported in Level 1 RTL synthesis.

### 6. Strong Typing

Two data objects may be used in an expression together if and only if there is an operator symbol or function that has matching type as a formal parameter and the result type matches the assignment or expression type.

### 7. Logic operators

Logic operators require parentheses between non-associative operators and different logic operators (except not).

Legal:

Z <= A and B and C;

Z <= (A and B) or C;

Z <= not A and B;

Z <= (not A) and B;

Z <= (A nand B) nand C;

Z <= A nand B nand C;

Illegal:

Z <= A and B or C;

Z <= (A and B) or C;

Z <= not A and B;

Z <= (not A) and B;

Z <= (A nand B) nand C;

### 7.1 Overloaded Logic Operators

Left	Right	Return	Package
bool	bool	bool	std
sl*	sl	sl	1164
slv*	slv	slv	1164
svl*	svl	svl	1164
uv*	uv	uv	ns**
sv*	sv	sv	ns**

\* Not operator only has a right value.

\*\* Provided by numeric\_std but not std\_logic\_arith.

### 8. Comparison

Comparison operators return BOOLEAN. Use either a conditional signal assignment (concurrent code) or if-then statement (sequential code) to change the type:

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

Comparison operators are implicitly defined by VHDL for all enumerated types. 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 std\_logic\_1164 strength strippers (To\_X01, ...) in behavioral models which receive data from a bus.

Comparisons only match identical characters. Hence, the following is invalid:

Dec1 <= '1' when Addr = "1-00" else '0';

Instead split up the expression as follows:

Dec1 <= '1' when (Addr(5) = '1' and  
 Addr(1 downto 0) = "00")  
 else '0';

### 8.1 Overloaded Comparison Operators

Left	Right	Return	Package	Notes
sl	sl	bool	implicit	
svlv	svlv	bool	implicit	
slv	slv	bool	slu, sls, implicit	1
slv	int	bool	slu, sls	1,2
int	slv	bool	slu, sls	1,2
uv	uv	bool	ns, sla	
uv	int0+	bool	ns, sla	2
int0+	uv	bool	ns, sla	2
sv	sv	bool	ns, sla	
sv	int	bool	ns, sla	2
int	sv	bool	ns, sla	2
sv	uv	bool	sla	5
uv	sv	bool	sla	5

Input arrays are extended to be the same length.

### 9. Shift

All shift operators were defined in the VHDL-93 language revision and are not supported in Level 1 RTL synthesis. It is recommended to do shifting with the & operator. Shift right by two for both unsigned and signed numbers (respectively) is shown below:

Z <= "00" & B(7 downto 2); -- unsigned

Z <= B(7) & B(7) & B(7 downto 2); -- signed

numeric\_std provides additional shift functions.

## 10. Addition and Multiplication

Addition and multiplication are only defined by the language for the types integer and real. The arithmetic packages overload the addition and multiplication operators to provide additional support.

### 10.1 Overloaded Addition Operators

Left	Right	Return	Package	Notes
slv	slv	slv	slu, sls	1
slv	int	slv	slu, sls	1,2
int	slv	slv	slu, sls	1,2
slv	sl	slv	slu, sls	1,3
sl	slv	slv	slu, sls	1,3
uv	uv	uv, slv*	ns, sla	4
uv	int0+	uv, slv*	ns, sla	2,4
int0+	uv	uv, slv*	ns, sla	2,4
uv	sl	uv, slv*	sla	3,4
sl	uv	uv, slv*	sla	3,4
sv	sv	sv, slv*	ns, sla	4
sv	int	sv, slv*	ns, sla	2,4
int	sv	sv, slv*	ns, sla	2,4
sv	sl	sv, slv*	sla	3,4
sl	sv	sv, slv*	sla	3,4
sv	uv	sv, slv*	sla	5
uv	sv	sv, slv*	sla	5

Note the size of the return value is the size of the largest array input. Input arrays are extended to be the same length.

### 10.2 Overloaded Multiplication Operators

Left	Right	Return	Package	Notes
slv	slv	slv	slu, sls	1
uv	uv	uv, slv*	ns, sla	4
uv	int0+	uv	ns	2
int0+	uv	uv	ns	2
sv	sv	sv, slv*	ns, sla	4
sv	int	sv	ns	2
int	sv	sv	ns	2
sv	uv	sv, slv*	sla	5
uv	sv	sv, slv*	sla	5

Note std\_logic\_arith, std\_logic\_unsigned,

std\_logic\_signed only provide \*\*\*.

The size of the result equals the sum of the size of the two array inputs (or 2 x size of the one array input).

## 11. 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 12.4 and 12.5 for examples.

## 12. Notes for Overloaded Operators

**12.1 std\_logic\_unsigned, std\_logic\_signed**  
May not be used together in the same entity / package. May be used with std\_logic\_arith, but not numeric\_std. Unsigned or signed interpretation of std\_logic\_vector determined by package in use.

### 12.2 Use Integer Constants

Integer constants are overloaded to use in signed and unsigned addition. See also note 5.

Z\_uv <= A\_uv + 16#5F#;

### 12.3 Counters with Count Enables

Addition with std\_logic / std\_ulogic facilitates implementation of counters with count enable functions.

IncVal <= IncReg + CountEn;

### 12.4 Std\_logic\_vector Return Values

Std\_logic\_arith overloads its functions to return std\_logic\_vector as well as unsigned or signed.

Facilitates mixing signed/unsigned and std\_logic\_vector  
Z\_slv <= signed(A\_slv) + signed(B\_slv);

Complicates multiple operator expressions:

Z\_sv <= signed(A\_sv + B\_sv) + C\_sv;

Does not apply to numeric\_std.

### 12.5 Signed & Unsigned Arguments

Std\_logic\_arith overloads addition and comparison to allow signed and unsigned operands in the same expression. Complicates usage of constants.

Z\_uv <= A\_uv + unsigned("0101");

Does not apply to numeric\_std.

## 13. Converting between types

### 13.1 Std\_logic <=> Signed(i), Unsigned(i)

Objects with a common base type do not need a conversion function:

A\_sl <= B\_sv(1);

C\_uv(1) <= D\_sl;

E\_sl <= F\_sl;

G\_sv(1) <= H\_uv(1);

### 13.2 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);

### 13.3 Unsigned, Signed <=> Integer

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

### Function

Function	In	Rtn	Pkg
conv_integer(val)	slv	int	slu, sls
conv_integer(val)	sv	int	sla
conv_integer(val)	uv	int	sla
conv_unsigned(val, len)	int, int	uv	sla
conv_signed(val, len)	int, int	sv	sla
conv_std_logic_vector	uv	sv	see note *
to_integer(val)	uv	int0+	ns
to_integer(val)	sv	int	ns
to_unsigned(val, len)	int0+, int0+	uv	ns
to_signed(val, len)	int, int0+	sv	ns

\* Avoid conv\_std\_logic\_vector(int, int) as it returns signed (which is not obvious from the call). Use: std\_logic\_vector(conv\_unsigned(A, int, 8))

## 14. 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
slv	slv	1164
svlv	svlv	1164

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')	uv, sl	uv	ns
to_01(s, xmap := '0')	sv, sl	sv	ns

## 15. X detection

Is\_X detects 'X' on inputs of models.

In	Return	Package
sl	bool	1164
slv	bool	1164
svlv	bool	1164

## 16. Edge detection

Functions rising\_edge and falling\_edge.

In	Return	Package
sl	bool	1164

Not supported by some synthesis tools.

© 1999 by SynthWorks Design Inc. Reproduction of entire document in whole permitted. All other rights reserved.

## SynthWorks Design Inc.

VHDL Hardware Synthesis and Verification Training

11898 SW Ave. Tigard OR 97223 1-(503)-590-4787

<http://www.SynthWorks.com> [jim@synthworks.com](mailto:jim@synthworks.com)