

## Building Blocks


### Entity Declaration

Description	Example
<b>entity</b> entity_name <b>is</b> <b>port</b> ( [ <b>signal</b> ] identifier {, identifier}: [mode] signal_type {; [ <b>signal</b> ] identifier {, identifier}: [mode] signal_type}); <b>end</b> [entity ] [entity_name];	<b>entity</b> register8 <b>is</b> <b>port</b> ( clk, rst, en: <b>in</b> std_logic; data: <b>in</b> std_logic_vector(7 downto 0); q: <b>out</b> std_logic_vector(7 downto 0)); <b>end</b> register8;

### Entity Declaration with Generics

Description	Example
<b>entity</b> entity_name <b>is</b> generic ( [signal] identifier {, identifier}: [mode] signal_type [:=static_expression] {;[signal] identifier {, identifier}: [mode] signal_type [:=static_expression]}); <b>port</b> ( [ <b>signal</b> ] identifier {, identifier}: [mode] signal_type {; [ <b>signal</b> ] identifier {, identifier}: [mode] signal_type}); <b>end</b> [entity ] [entity_name];	<b>entity</b> register_n <b>is</b> generic( width: integer :=8); <b>port</b> ( clk, rst, en: <b>in</b> std_logic; data: <b>in</b> std_logic_vector(width-1 downto 0); q: <b>out</b> std_logic_vector(width-1 downto 0)); <b>end</b> register_n;

### Architecture Body

Description	Example
architecture architecture_name of entity_name is type_declaration   signal_declaration   constant_declaration   component_declaration   alias_declaration   attribute_specification   subprogram_body  begin { process_statement   concurrent_signal_assignment_statement   component_instantiation_statement   generate_statement } end [architecture] [architecture_name];	<b>architecture</b> behavioral <b>of</b> register8 <b>is</b> <b>begin</b> <b>process</b> (rst, clk) <b>begin</b> if (rst='1') then q <= (others => '0'); elseif (clk'event and clk='1') then if (en='1') then q <= data; else q <= q; end if; <b>end if</b> ; <b>end process</b>  <b>end</b> behavioral;  <b>architecture</b> archfsm <b>of</b> fsm <b>is</b> <b>type</b> state_type is (st0, st1, st2); <b>signal</b> state: state_type; <b>signal</b> y, z: std_logic;

	<pre> <b>begin</b>   <b>process begin</b>     <b>wait until</b> clk = '1';     <b>case state is</b>       when st0 =&gt;         state &lt;= st1;         y &lt;= '1';       when st1 =&gt;         state &lt;= st2;         z &lt;= '1';       when others =&gt;         state &lt;= st3;         y &lt;= '0';         z &lt;= '0';     <b>end case;</b>   <b>end process;</b> <b>end archfsm;</b> </pre>
--	--

## Declaring a Component

Description	Example
<pre> component component_name port (   [signal] identifier {, identifier}: [mode] signal_type   {; [signal] identifier {, identifier}: [mode] signal_type}); end component [component_name]; </pre>	<pre> <b>component</b> register8 <b>port</b> (   clk, rst, en: <b>in</b> std_logic;   data: <b>in</b> std_logic_vector(7 downto 0);   q: <b>out</b> std_logic_vector(7 downto 0)); <b>end component;</b> </pre>

## Declaring a Component with Generics

Description	Example
<pre> component component_name generic (   [signal] identifier {, identifier}: [mode] signal_type   [:=static_expression]   {[signal] identifier {, identifier}: [mode] signal_type   [:=static_expression]}); port (   [signal] identifier {, identifier}: [mode] signal_type   {; [signal] identifier {, identifier}: [mode] signal_type}); end [component ] [component_name]; </pre>	<pre> <b>component</b> register8 <b>generic</b>(   width: integer :=8); <b>port</b> (   clk, rst, en: <b>in</b> std_logic;   data: <b>in</b> std_logic_vector(width-1 downto 0);   q: <b>out</b> std_logic_vector(width-1 downto 0)); <b>end component;</b> </pre>

## Component Instantiation (named association)

Description	Example
<pre> instantiation_label: component_name port map (   port_name =&gt; signal_name     expression     variable_name     open   {, port_name =&gt; signal_name </pre>	<pre> <b>architecture</b> arch8 <b>of</b> reg8 <b>is</b>   <b>signal</b> clock, reset, enable: std_logic;   <b>signal</b> data_in, data_out: std_logic_vector(7 downto 0); <b>begin</b>   First_reg8: register8   <b>port map</b> (     clk =&gt; clock,     rst =&gt; reset, </pre>

expression variable_name open});	en => enable, data => data_in, q=> data_out); <b>end</b> arch8;
--	--

### Component Instantiation with Generics (named association)

Description	Example
<pre> instruction_label: component_name generic map(   generic_name =&gt; signal_name       expression       variable_name       open   {, generic_name =&gt; signal_name       expression       variable_name       open}) port map (   port_name =&gt; signal_name       expression       variable_name       open   {, port_name =&gt; signal_name       expression       variable_name       open}); </pre>	<pre> <b>architecture</b> structural <b>of</b> reg5 <b>is</b>   <b>signal</b> clock, reset, enable: std_logic;   <b>signal</b> data_in, data_out: std_logic_vector(7 downto 0); <b>begin</b>   First_reg5: Registern     <b>generic map</b> (width =&gt; 5) – no semicolon here     <b>port map</b> (       clk =&gt; clock,       rst =&gt; reset,       en =&gt; enable,       data =&gt; data_in,       q=&gt; data_out); <b>end</b> structural; </pre>

### Component Instantiation (positional association)

Description	Example
<pre> instantiation_label: component_name port map ( signal_name   expression     variable_name   open   {, signal_name   expression     variable_name   open}); </pre>	<pre> <b>architecture</b> structural <b>of</b> reg8 <b>is</b>   <b>signal</b> clock, reset, enable: std_logic;   <b>signal</b> data_in, data_out: std_logic_vector(7 downto 0); <b>begin</b>   First_reg8: register8     <b>port map</b> (clock, reset, enable, data_in, data_out); <b>end</b> structural; </pre>

## Component instantiation with Generics (positional association)

Description	Example
instantiation_label: component_name generic map( signal_name   expression   variable_name   open {, signal_name   expression   variable_name   open}) port map (signal_name   expression   variable_name   open {, signal_name   expression   variable_name   open});	architecture structural of reg5 is signal clock, reset, enable: std_logic; signal data_in: std_logic_vector(7 downto 0); signal data_out: std_logic_vector(7 downto 0);  begin first_reg5: register_n generic map (5) port map (clock, reset, enable, data_in, data_out); end structural;

## Concurrent statements

### Boolean equations

Description	Example
relation { and relation}   relation {or relation}   relation {xor relation}   relation {nand relation}   relation {nor relation}	v <= (a and b and c) or d; -- parenthesis required with 2-level logic w <= a or b or c; x <= a xor b xor c; y <= a nand b nand c; z <= a nor b nor c;

## When-else conditional signal assignment

Description	Example
{expression when condition else} expression;	x <= '1' when b = c else '0'; y <= j when state = idle else k when state = second_state else m when others;

## With-select-when Selected Signal Assignment

Description	Example
with selection_expression select {identifier <= expression when identifier   expression   discrete_range   others,} identifier <= expression when identifier   expression   discrete_range   others;	architecture archfsm of fsm is type state_type is (st0, st1, st2, st3, st4, st5, st6, st7, st8); signal state: state_type; signal y, z: std_logic_vector(3 downto 0); begin with state select x <= "0000" when st0   st1,

	"0010" when st2   st3, y when st4, z when others; end archfsm;
--	---

## Generate scheme for component instantiation or equations

Description	Example
generate_label: (for identifier in discret_range)   ( if condition) generate { concurrent_statement } end generate [generate_label];	g1: for i in 0 to 7 generate reg1: register8 port map (clock, reset, enable, data_in(i), data_out(i)); end generate g1;  g2: for j in 0 to 2 generate a(j) <= b(j) xor c(j); end generate g2;

## Sequential statements

### Process statement

Description	Example
[process_label:] process ( sensitivity_list) { type_declaration   constant_declaration   variable_declaration   alias_declaration } begin { wait_statement   signal_assignment_statement   variable_assignment_statement   if_statement   case_statement   loop_statement } end process [process_label];	my_process: process (rst, clk) constant zilch: std_logic_vector(7 downto 0) := "00000000";  begin wait until clk = '1'; if (rst = '1') then q <= zilch; elsif (en = '1') then q <= data; else q <= q; end if; end process my_process;

### If-then-else statement

Description	Example
if condition then sequence_of_statements { elsif condition then sequence_of_statements } [ else sequence_of_statements ] end if;	if (count = "00") then a <= b; elsif (count = "10") then a <= c; else a <= d; end if;

## Case-when statement

Description	Example
case expression is {when identifier   expression   discrete_range others => sequence_of_statements} end case;	case count is when "00" => a<=b; when "10" => a<=c; when others => a<=d; end case;

## For-loop statement

Description	Example
[loop_label:] for identifier in discrete_range loop {sequence_of_statements} end loop [loop_label];	my_for_loop: for i in 3 downto 0 loop if reset(i) = '1' then data_out(i)<= '0'; end if; end loop my_for_loop;

## While-loop statement

Description	Example
[loop_label:] while condition loop {sequence_of_statements} end loop [loop_label];	count := 16; while (count > 0) loop count := count - 1; result <= data_in; end loop;

## Describing Synchronous Logic Using Processes

### No Reset (Assume clock is of type std\_logic)

Description	Example
[process_label:] <b>process</b> (clock) <b>begin</b> if clock'event <b>and</b> clock = '1' <b>then</b> synchronous_signal_assignment_statement; <b>end if</b> ; <b>end process</b> [process_label]; -----or----- [process_label:] <b>process</b> <b>begin</b> <b>wait until</b> clock = '1' ; synchronous_signal_assignment_statement; <b>end process</b> [process_label];	reg8_no_reset: <b>process</b> (clk) <b>begin</b> if clk'event <b>and</b> clk = '1' <b>then</b> q <= data; <b>end if</b> ; <b>end process</b> reg8_no_reset; -----or----- reg8_no_reset: <b>process</b> <b>begin</b> <b>wait until</b> clk = '1'; q <= data; <b>end process</b> reg8_no_reset;

## Synchronous Reset

Description	Example
<pre>[process_label:] <b>process</b> (clock) <b>begin</b>   <b>if</b> clock'event <b>and</b> clock = '1' <b>then</b>     <b>if</b> synch_reset_signal = '1' <b>then</b>       synchronous_signal_assignment_statement;     <b>else</b>       synchronous_signal_assignment_statement;     <b>end if</b>;   <b>end if</b>; <b>end process</b> [process_label];</pre>	<pre>reg8_sync_reset: <b>process</b> (clk) <b>begin</b>   <b>if</b> clk'event <b>and</b> clk = '1' <b>then</b>     <b>if</b> synch_reset = '1' <b>then</b>       q &lt;= "00000000";     <b>else</b>       q &lt;= data;     <b>end if</b>;   <b>end if</b>; <b>end process</b>;</pre>

## Asynchronous Reset or Preset

Description	Example
<pre>[process_label:] <b>process</b> (reset, clock) <b>begin</b>   <b>if</b> reset = '1' <b>then</b>     asynchronous_signal_assignment_statement;   <b>elsif</b> clock'event <b>and</b> clock = '1' <b>then</b>     synchronous_signal_assignment_statement;   <b>end if</b>; <b>end process</b> [process_label];</pre>	<pre>reg8_async_reset: <b>process</b> (async_reset, clk) <b>begin</b>   <b>if</b> async_reset = '1' <b>then</b>     q &lt;= (<b>others</b> =&gt; '0');   <b>elsif</b> clk'event <b>and</b> clk = '1' <b>then</b>     q &lt;= data;   <b>end if</b>; <b>end process</b> reg8_async_reset ;</pre>

## Asynchronous Reset and Preset

Description	Example
<pre>[process_label:] <b>process</b> (reset, preset, clock) <b>begin</b>   <b>if</b> reset = '1' <b>then</b>     asynchronous_signal_assignment_statement;   <b>elsif</b> preset = '1' <b>then</b>     synchronous_signal_assignment_statement;   <b>elsif</b> clock'event <b>and</b> clock = '1' <b>then</b>     synchronous_signal_assignment_statement;   <b>end if</b>; <b>end process</b> [process_label];</pre>	<pre>reg8_async: <b>process</b> (async_reset, async_preset, clk) <b>begin</b>   <b>if</b> async_reset = '1' <b>then</b>     q &lt;= (<b>others</b> =&gt; '0');   <b>elsif</b> async_preset = '1' <b>then</b>     q &lt;= (<b>others</b> =&gt; '1');   <b>elsif</b> clk'event <b>and</b> clk = '1' <b>then</b>     q &lt;= data;   <b>end if</b>; <b>end process</b> reg8_async ;</pre>

## Conditional Synchronous Assignment (enables)

Description	Example
<pre>[process_label:] <b>process</b> (reset, clock) <b>begin</b>   <b>if</b> reset = '1' <b>then</b>     asynchronous_signal_assignment_statement;   <b>elsif</b> clock'event <b>and</b> clock = '1' <b>then</b>     <b>if</b> enable = '1' <b>then</b></pre>	<pre>reg8_sync_assign: <b>process</b> (rst, clk) <b>begin</b>   <b>if</b> rst = '1' <b>then</b>     q &lt;= (<b>others</b> =&gt; '0');   <b>elsif</b> clk'event <b>and</b> clk = '1' <b>then</b>     <b>if</b> enable = '1' <b>then</b></pre>

<pre> synchronous_signal_assignment_statement; <b>else</b> synchronous_signal_assignment_statement; <b>end if;</b> <b>end if;</b> <b>end process</b> [process_label]; </pre>	<pre> q &lt;= data; <b>else</b> q &lt;= q; <b>end if;</b> <b>end if;</b> <b>end process</b> reg8_sync_assign; </pre>
--	--

## bit and bit\_vector

Description	Example
<ul style="list-style-type: none"> <li>Bit values are: '0' and '1'.</li> <li>Bit vector is an array of bits.</li> <li>Pre-defined by the IEEE 1076 standard.</li> <li>This type was used extensively prior to the introduction and synthesis-tool vendor support of std_logic_1064.</li> <li>Useful when metalogic values not required.</li> </ul>	<pre> <b>signal</b> x: bit; ... <b>if</b> x = '1' <b>then</b> state &lt;= idle; <b>else</b> state &lt;= start; <b>end if;</b> </pre>

## Boolean

Description	Example
<ul style="list-style-type: none"> <li>Values are true and false</li> <li>Often used as return values of function.</li> </ul>	<pre> <b>signal</b> a: Boolean; ... <b>if</b> a <b>then</b> state &lt;= idle; <b>else</b> state &lt;= start; <b>end if;</b> </pre>

## Integer

Description	Example
<ul style="list-style-type: none"> <li>Values are the set of integers.</li> <li>Data objects of this type are often used for defining widths of signals or as an operand in an addition or subtraction.</li> <li>The types std_logic_vector work better than integer for components such as counters because the use of integers may cause “out of range” run-time simulation errors when the counter reaches its maximum value.</li> </ul>	<pre> <b>entity</b> counter_n <b>is</b> <b>Generic</b> ( width: integer := 8); <b>port</b> ( clk, rst: in std_logic; count: out std_logic_vector(width-1 downto 0)); <b>end</b> counter_n; ... <b>process</b>(clk, rst) <b>begin</b> <b>if</b> (rst = '1') <b>then</b> count &lt;= (<b>others</b> =&gt; '0'); <b>elsif</b> (clk'event and clk = '1') <b>then</b> count &lt;= count + 1; <b>end if;</b> <b>end process;</b> </pre>



## Enumeration Types

Description	Example
<ul style="list-style-type: none"> <li>Values are user-defined.</li> <li>Commonly used to define states for a state machine.</li> </ul>	<pre> <b>architecture</b> archfsm <b>of</b> fsm <b>is</b>   <b>type</b> state_type <b>is</b> (st0, st1, st2);   <b>signal</b> state: state_type;   <b>signal</b> y, z: std_logic; <b>begin</b>   <b>process</b>   <b>begin</b>     <b>wait</b> until clk'event;     <b>case</b> state <b>is</b>       <b>when</b> st0 =&gt;         state &lt;= st2;         y &lt;= '1';         z &lt;= '0';       <b>when</b> st1 =&gt;         state &lt;= st3;         y &lt;= '1';         z &lt;= '1';       <b>when</b> others =&gt;         state &lt;= st0;         y &lt;= '0';         z &lt;= '0';       <b>end case</b>;     <b>end process</b>;   <b>end archfsm</b>; </pre>

## Variables

Description	Example
<ul style="list-style-type: none"> <li>Values can be used in processes and subprograms – that is, in sequential areas only.</li> <li>The scope of a variable is the process or subprogram.</li> <li>A variable in a subprogram.</li> <li>Variables are most commonly used as the indices of loops or for the calculation of intermediate values, or immediate assignment.</li> <li>To use the value of a variable outside of the process or subprogram in which it was declared, the value of the variable must be assigned to a signal.</li> <li>Variable assignment is immediate, not scheduled.</li> </ul>	<pre> <b>architecture</b> archloopstuff <b>of</b> loopstuff <b>is</b>   <b>signal</b> data: std_logic_vector(3 downto 0);   <b>signal</b> result: std_logic; <b>begin</b>   <b>process</b> (data)     variable tmp: std_logic;   <b>begin</b>     tmp := '1';     <b>for</b> i <b>in</b> a'high <b>downto</b> 0 <b>loop</b>       tmp := tmp and data(i);     <b>end loop</b>;     result &lt;= tmp;   <b>end process</b>; <b>end archloopstuff</b>; </pre>

## Data Types and Subtypes

### Std\_logic

Description	Example
<ul style="list-style-type: none"><li>Values are:<ul style="list-style-type: none"><li>'U', -- Uninitialized</li><li>'X', -- Forcing unknown</li><li>'0', -- Forcing 0</li><li>'1', -- Forcing 1</li><li>'Z', -- High impedance</li><li>'W', -- Weak unknown</li><li>'L', -- Weak 0</li><li>'H', -- Weak 1</li><li>'-', -- Don't care</li></ul></li><li>The standard multivalued logic system for VHDL model interoperability.</li><li>A resolved type (i.e., a resolution function is used to determine the outcome in case of multiple drivers).</li><li>To use must include the following two lines: <b>library</b> ieee; <b>use</b> ieee.std_logic_1164.all;</li></ul>	<pre><b>signal</b> x, data, enable: std_logic; ... x &lt;= data <b>when</b> enable = '1' <b>else</b> 'Z';</pre>

### Std\_ulogic

Description	Example
<ul style="list-style-type: none"><li>Values are:<ul style="list-style-type: none"><li>'U', -- Uninitialized</li><li>'X', -- Forcing unknown</li><li>'0', -- Forcing 0</li><li>'1', -- Forcing 1</li><li>'Z', -- High impedance</li><li>'W', -- Weak unknown</li><li>'L', -- Weak 0</li><li>'H', -- Weak 1</li><li>'-', -- Don't care</li></ul></li><li>An unresolved type (i.e., a signal of this type may have only one driver).</li><li>Along with its subtypes, std_ulogic should be used over user-defined types to ensure interoperability of VHDL models among synthesis and simulation tools.</li><li>To use must include the following two lines: <b>library</b> ieee; <b>use</b> ieee.std_ulogic_1164.all;</li></ul>	<pre><b>signal</b> x, data, enable: std_ulogic; ... x &lt;= data <b>when</b> enable = '1' <b>else</b> 'Z';</pre>

## Std\_logic\_vector and std\_ulogic\_vector

Description	Example
<ul style="list-style-type: none"> <li>Are arrays of type std_logic and std_ulogic.</li> <li>Along with its subtypes, std_logic_vector should be used over user defined types to ensure interoperability of VHDL models among synthesis and simulation tools.</li> <li>To use must include the following two lines:  <b>library ieee;</b>  <b>use ieee.std_logic_1164.all;</b> </li> </ul>	<pre> <b>signal</b> mux: std_logic_vector(7 <b>downto</b> 0); ... <b>if</b> state = address <b>or</b> (state = ras) <b>then</b>     mux &lt;= dram_a; <b>else</b>     mux &lt;= (<b>others</b> =&gt; 'Z'); <b>end if</b>; </pre>

## In, Out, Buffer, Inout

Description	Example
<ul style="list-style-type: none"> <li><b>In:</b> Used for signals (ports) that are inputs-only to an entity.</li> <li><b>Out:</b> Used for signals that are outputs-only and for which the values are not required internal to the entity.</li> <li><b>Buffer:</b> Used for signals that are outputs, but for which the values are required internal to the given entity. Caveat with usage: If the local port of an instantiated component is of mode buffer, then if the actual is also a port, it must be of mode buffer as well. For this reason, some designers standardize on mode buffer as well.</li> <li><b>Inout:</b> Used for signals that are truly bidirectional. May also be used for signals that are inputs-only or outputs, at the expense of code readability.</li> </ul>	<pre> <b>entity</b> dff_extra <b>is</b>     <b>port</b>(         D : in STD_LOGIC_vector(3 <b>downto</b> 0);         clock : in STD_LOGIC;         E : in STD_LOGIC;         Q : out STD_LOGIC_vector(3 <b>downto</b> 0)     ); <b>end</b> dff_extra;  <b>architecture</b> behavior <b>of</b> dff_extra <b>is</b>     <b>begin</b>         <b>process</b>(clock)             <b>begin</b>                 <b>if</b> (clock = '1' <b>and</b> clock'EVENT) <b>then</b>                     <b>if</b> (E = '1') <b>then</b>                         Q &lt;= D;                     <b>end if</b>;                 <b>end if</b>;             <b>end process</b>;             -- enter your statements here --         <b>end</b> behavior; </pre>

## Operator

All operators of the class have the same level of precedence. The classes of operators are listed here in order of the decreasing precedence. Many of the operators are overloaded in the std\_logic\_1164, numeric\_bit, and numeric\_std packages.

### Miscellaneous Operators

Description	Example
<ul style="list-style-type: none"> <li>Operator: **, abs, not .</li> <li>The not operator is used frequently, the other two are rarely used for designs to be synthesized.</li> </ul>	<pre> <b>variable</b> a, b: integer <b>range</b> 0 <b>to</b> 255; ... a &lt;= b**2; </pre>

<ul style="list-style-type: none"> <li>Predefined for any integer type (*, /, mod, rem), and any floating point type (*, /).</li> </ul>	
---	--

## Sign

Description	Example
<ul style="list-style-type: none"> <li>Operators: +, -.</li> <li>Rarely used for synthesis.</li> <li>Predefined for any numeric type (floating point or integer).</li> </ul>	<b>variable</b> a, b, c: integer <b>range</b> 0 to 255; ... b <= - (a + 5);

## Adding Operators

Description	Example
<ul style="list-style-type: none"> <li>Operators: +, -</li> <li>Used frequently to describe incrementers, decrementers, adders, and subtractors.</li> <li>Predefined for any numeric type.</li> </ul>	<b>signal</b> count: integer <b>range</b> 0 to 255; ... count <= count -5;

## Shift Operators

Description	Example
<ul style="list-style-type: none"> <li>Operators: sll, srl, sla, sra, rol, ror.</li> <li>Used occasionally.</li> <li>Predefined for any one-dimensional array with elements of type bit or Boolean. Overloaded for std_logic arrays.</li> </ul>	<b>signal</b> a, b: bit_vector(4 downto 0); <b>signal</b> c: integer <b>range</b> 0 to 4; ... a <= b ror c;

## Relational Operators

Description	Example
<ul style="list-style-type: none"> <li>Operators: =, /=, &lt;, &lt;=, &gt;, &gt;=.</li> <li>Used frequently for comparisons.</li> <li>Predefined for any type (both operands must be of same type).</li> </ul>	<b>signal</b> a, b: integer <b>range</b> 0 to 255; <b>signal</b> c: std_logic; ... <b>if</b> a >= b <b>then</b> c <= '1'; <b>else</b> c <= '0'; <b>end if</b> ;

## Logical Operators

Description	Example
<ul style="list-style-type: none"> <li>Operators: and, or, nand, nor, xor, xnor.</li> <li>Used frequently to generate Boolean equations.</li> <li>Predefined for types bit and Boolean. std_logic_1164 overloads these operators for std_ulogic and its subtypes.</li> </ul>	<b>signal</b> a, b, c, d: std_logic; ... a <= b nand c; d <= a xor b;