

Building Blocks

Entity Declaration

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

Entity Declaration with Generics

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

Architecture Body

Description	Example
<pre>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];</pre>	<pre>architecture archregister8 of register8 is begin process (rst, clk) begin 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; end if; end process end archregister8; architecture archfsm of fsm is type state_type is (st0, st1, st2); signal state: state_type; signal y, z: std_logic; begin</pre>

	<pre> process begin wait until clk = '1'; case state is when st0 => state <= st1;; y <= '1'; when st1 => state <= st2;; z <= '1'; when others => state <= st3;; y <= '0'; z <= '0'; end case; end process; end archfsm; </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> component register8 generic(width: integer :=8); port (clk, rst, en: in std_logic; data: in std_logic_vector(7 downto 0); q: out std_logic_vector(7 downto 0); end component; </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> component register8 generic(width: integer :=8); port (clk, rst, en: in std_logic; data: in std_logic_vector(width-1 downto 0); q: out std_logic_vector(width-1 downto 0); end component; </pre>

Component Instantiation (named association)

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

<pre>{, port_name => signal_name expression variable_name open});</pre>	<pre>clk => clock, rst => reset, en => enable, data => data_in; q=> data_out); end archreg8;</pre>
--	--

Component Instantiation with Generics (named association)

Description	Example
<pre>instruction_label: component_name generic map(generic_name => signal_name expression variable_name open {, generic_name => signal_name expression variable_name open}) port map (port_name => signal_name expression variable_name open {, port_name => signal_name expression variable_name open});</pre>	<pre>architecture arch5 of reg5 is signal clock, reset, enable: std_logic; signal data_in, data_out: std_logic_vector(7 downto 0); begin First_reg5: Registern generic map (width => 5) – no semicolon here port map (clk => clock, rst => reset, en => enable, data => data_in; q=> data_out); end archreg5;</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>architecture arch8 of reg8 is signal clock, reset, enable: std_logic; signal data_in, data_out: std_logic_vector(7 downto 0); begin First_reg8: register8 port map (clock, reset, enable, data_in, data_out); end archreg8;</pre>

Component instantiation with Generics (positional association)

Description	Example
<pre>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});</pre>	<pre>architecture archreg5 of reg5 is signal clock, reset, enable: std_logic; signal data_in , 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 archreg5;</pre>

Concurrent statements

Boolean equations

Description	Example
<pre>relation { and relation } relation { or relation } relation { xor relation } relation { nand relation } relation { nor relation }</pre>	<pre>v<= (a and b and c) or d;-- parenthesis req'd w/ 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;</pre>

When-else conditional signal assignment

Description	Example
<pre>{expression when condition else } expression;</pre>	<pre>x<= '1' when b = c else '0'; y<= j when state = idle else l when state = secon_state else m when others;</pre>

With-select-when Selected Signal Assignment

Description	Example
<pre>with selection_expression select {identifier<= expression when identifier expression discrete_range others, } identifier<= expression when identifier expression discrete_range others;</pre>	<pre>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; --st0</pre>

	<pre>"or" st1 "0010" when st2 st3; y when st4; z when others; end archfsm;</pre>
--	--

Generate scheme for component instantiation or equations

Description	Example
<pre>generate_label: (for identifier in discret_range) (if condition) generate { concurrent_statement } end generate[generate_label];</pre>	<pre>g1: for i in 0 to 7 generate reg1: register8 prot 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;</pre>

Sequential statements

Process statement

Description	Example
<pre>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];</pre>	<pre>my_process; process(rst,clk) constant zilch: std_logic_vector(7 downto 0):= "0000_0000"; begin wait until clk = '1'; if (rst = '1') then q<= zilch; elsif (en = '1') then q<= data; else q<= q; end if; end my_process;</pre>

If-then-else statement

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

Case-when statement

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

For-loop statement

Description	Example
<pre>[loop_label:] for identifier in discrete_range loop { sequence_of_statements } end loop[loop_label];</pre>	<pre>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;</pre>

While-loop statement

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

Describing Synchronous Logic Using Processes

No Reset (Assume clock is of type std_logic)

Description	Example
<pre>[process_label:] process (clock) begin if clock'event and clock = '1' then -- or rising_edge synchronous_signal_assignment_statement; end if; end process [process_label]; -----Or----- [process_label:] process begin wait until clock = '1' ; synchronous_signal_assignment_statement; end process [process_label];</pre>	<pre>reg8_no_reset: process (clk) begin if clk'event and clk = '1' then q <= data; end if; end process Reg8_no_reset; -----Or----- reg8_no_reset: process begin wait until clock = '1'; q <= data; end process Reg8_no_reset;</pre>

Synchronous Reset

Description	Example
<pre>[process_label:] process (clock) begin if clock'event and clock = '1' then if synch_reset_signal = '1' then synchronous_signal_assignment_statement; else synchronous_signal_assignment_statement; end if; end if; end process [process_label];</pre>	<pre>reg8_no_reset: process (clk) begin if clk'event and clk = '1' then if synch_reset = '1' then q <= "0000_0000"; else q <= data; end if; end if; end process;</pre>

Asynchronous Reset or Preset

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

Asynchronous Reset and Preset

Description	Example
<pre>[process_label:] process (reset, preset, clock) begin if reset = '1' then asynchronous_signal_assignment_statement; elsif preset = '1' then synchronous_signal_assignment_statement; elsif clock'event and clock = '1' then synchronous_signal_assignment_statement; end if; end process [process_label];</pre>	<pre>reg8_async: process (asyn_reset,asyn_preset, clk) begin if asynch_reset = '1' then q <= (others => '0'); elsif asynch_preset = '1' then q <= (others => '1'); elsif clk'event and clk = '1' then q <= data; end if; end process reg8_async ;</pre>

Conditional Synchronous Assignment (enables)

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

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

bit and bit_vector

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

Boolean

Description	Example
<ul style="list-style-type: none"> • Values are true and false • Often used as return values of function. 	<pre>signal a: boolean; . . . if a = '1' then state <= idle; else state <= start; end if;</pre>

Integer

Description	Example
<ul style="list-style-type: none"> • Values are the set of integers. • Data objects of this type are often used for defining widths of signals or as an operand in an addition or subtraction. • 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. 	<pre>Entity counter_n is Generic (Width: integer := 8); port (clk, rst: in std_logic; count: out std_logic_vector(width-1 downto 0)); end counter_n; . . . Process(clk) Begin If (rst = '1') then Count ,+ 0; Elsif (clk'event and clk = '1') then Count <= count + 1; End if; End process;</pre>

Enumeration Types

Description	Example
<ul style="list-style-type: none"> • Values are user-defined. • Commonly used to define states for a state machine. 	<pre>architecture archfsm of fsm is type state_type is (st0, st1, st2); signal state: state_type; signal y, z: std_logic; begin process begin</pre>

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

Variables

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

Data Types and Subtypes

Std_logic

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

Std_ulogic

Description	Example
<ul style="list-style-type: none"> Values are: <ul style="list-style-type: none"> 'U', -- Uninitialized 'X', -- Forcing unknown '0', -- Forcing 0 '1', -- Forcing 1 'Z', -- High impedance 'W', -- Weak unknown 'L', -- Weak 0 'H', -- Weak 1 '-', -- Don't care An unresolved type (i.e., a signal of this type may have only one driver). Along with its subtypes, std_ulogic should be used over user-defined types to ensure interoperability of VHDL models among synthesis and simulation tools. To use must include the following two lines: Library ieee; Use ieee.std_logic_1164.all; 	<pre> signal x, data, enable: std_ulogic; . . . x <= data when enable = '1' else 'Z'; </pre>

Std_logic_vector and std_ulogic_vector

Description	Example
<ul style="list-style-type: none"> Are arrays of type std_logic and std_ulogic. 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. To use must include the following two lines: Library ieee; Use ieee.std_logic_1164.all; 	<pre> signal mux: std_logic_vector(7 downto 0); . . . if state = address or state = ras then mux <= dram_a; else mux <= (others => 'Z'); end if; </pre>

In, Out, Buffer, Inout

Description	Example
<ul style="list-style-type: none"> In: Used for signals (ports) that are inputs-only to an entity. Out: Used for signals that are outputs-only and for which the values are not required internal to the entity. Buffer: 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 	<pre> entity dff_extra is port(D : in STD_LOGIC_vector(3 downto 0); clock : in STD_LOGIC; E : in STD_LOGIC; Q : out STD_LOGIC_vector(3 downto 0)); end dff_extra; architecture behavior of dff_extra is begin process(clock) begin if(clock = '1' and clock'EVENT) then if(E = '1') then Q <= D; end if; end if; end process; end architecture; </pre>

<p>reason, some designers standardize on mode buffer as well.</p> <ul style="list-style-type: none"> • Inout: 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. 	<pre> end if; end if; end process; -- enter your statements here -- end 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"> • Operator: <code>**</code>, <code>abs</code>, <code>not</code>. • The <code>not</code> operator is used frequently, the other two are rarely used for designs to be synthesized. • Predefined for any integer type (<code>*</code>, <code>/</code>, <code>mod</code>, <code>rem</code>), and any floating point type (<code>*</code>, <code>/</code>). 	<pre> variable a, b: integer range 0 to 255; ... a <= b *2; </pre>

Sign

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

Adding Operators

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

Shift Operators

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

Relational Operators

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

Logical Operators

Description	Example
<ul style="list-style-type: none">• Operators: and, or, nand, nor, xor, xnor.• Used frequently to generate Boolean equations.• Predefined for types bit and Boolean. Std_logic_1164 overloads these operators for std_ulogic and its subtypes.	<pre>signal a, b, c, d: std_logic; ... a <= b nand c; d <= a xor b;</pre>