Building Blocks

Entity Declaration

Description	Example

entity entity_name **is**
  port (
  [**signal**] identifier {, identifier}: [mode] signal_type
  {; [**signal**] identifier {, identifier}: [mode] signal_type});
end [**entity**] [entity_name];
	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;

Entity Declaration with Generics

Description	Example

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];
	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;

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];
	architecture behavioral **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 behavioral;
	
	architecture archfsm **of** fsm **is**
	type state_type is (st0, st1, st2);
	signal state: state_type;
	signal y, z: std_logic;

	begin
	  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;

Declaring a Component

Description	Example

component component _name
  port (
  [signal] identifier {, identifier}: [mode] signal_type
  {; [signal] identifier {, identifier}: [mode] signal_type});
end component [component _name];
	component register8
	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;

Declaring a Component with Generics

Description	Example

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];
	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;

Component Instantiation (named association)

Description	Example

instantiation_label:
  component_name
  port map (
  port_name => signal_name
  | expression
  | variable_name
  | open
  {, port_name => signal_name
	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 (
	  clk => clock,
	  rst => reset,

	| expression | variable_name | open});	en => enable, data =>data_in, q=> data_out); **end** arch8;

Component Instantiation with Generics (named association)

Description	Example

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});
	architecture structural **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 structural;

Component Instantiation (positional association)

Description	Example
instantiation_label: component_name port map (signal_name | expression | variable_name | open {, signal_name | expression | variable_name | open});	**architecture** structural **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** structural;

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_lable:]
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:]
process (clock)
begin
if clock'event **and** clock = '1' **then**
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];
	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 clk = '1';
	q <= data;
	end process reg8_no_reset;

Synchronous Reset

Description	Example

[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];
	reg8_sync_reset:
	process (clk)
	begin
	  if clk'event **and** clk = '1' **then**
	  if synch_reset = '1' **then**
	  q <= "00000000";
	  else
	  q <= data;
	  end if;
	  end if;
	end process;

Asynchronous Reset or Preset

Description	Example

[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];
	reg8_async_reset:
	process (async_reset, clk)
	begin
	  if async_reset = '1' **then**
	  q <= (**others** => '0');
	  elsif clk'event and clk = '1' then
	  q <= data;
	  end if;
	end process reg8_async_reset **;**

Asynchronous Reset and Preset

Description	Example

[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];
	reg8_async:
	process (async_reset, async_preset, clk)
	begin
	  if async_reset = '1' **then**
	  q <= (others => '0');
	  elsif async_preset = '1' **then**
	  q <= (others => '1');
	  elsif clk'event and clk = '1' then
	  q <= data;
	  end if;
	end process reg8_async **;**

Conditional Synchronous Assignment (enables)

Description	Example

[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
	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

synchronous_signal_assignment_statement;
else
synchronous_signal_assignment_statement;
end if;
end if;
end process [process_label];
	q <= data;
	else
	q <= q;
	end if;
	end if;
	end process reg8_sync_assign**;**

bit and bit_vector

Description	Example

• 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.
	signal x: bit;
	. . .
	if x = '1' **then**
	state <= idle;
	else
	state <= start;
	end if;

Boolean

Description	Example

• Values are true and false
• Often used as return values of function.
	signal a: Boolean;
	. . .
	if a **then**
	state <= idle;
	else
	state <= start;
	end if;

Integer

Description	Example

• 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.
	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, rst)
	begin
	if (rst = '1') **then**
	count <= (**others** => '0');
	elsif (clk'event and clk = '1') **then**
	count <= count + 1;
	end if;
	end process;

Enumeration Types

Description	Example

• Values are user-defined.
• Commonly used to define states for a state machine.
	architecture archfsm **of** fsm **is**
	  type state_type **is** (st0, st1, st2);
	  signal state: state_type;
	  signal y, z: std_logic;
	begin
	  process
	  begin
	  wait until clk'event;
	  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;

Variables

Description	Example

• 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 of 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.
	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'high **downto** 0 **loop**
	  tmp := tmp and data(i);
	  end loop;
	  result <= tmp;
	end process;
	end archloopstuff;

Data Types and Subtypes

Std_logic

Description	Example
• Values are: '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 multivalue logic system for VHDL model interoperability. • A resolved type (i.e., a resolution function is used to determine the outcome in case of multiple drivers). • To use must include the following two lines: **library** ieee; **use** ieee.std_logic_1164.all;	**signal** x, data, enable: std_logic; . . . x <= data **when** enable = '1' **else** 'Z';

Std_ulogic

Description	Example
• Values are: '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_ilogic_1164.all;	**signal** x, data, enable: std_ulogic; . . . x <= data **when** enable = '1' else 'Z';

Std_logic_vector and std_ulogic_vector

Description	Example

• 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;
	signal mux: std_logic_vector(7 **downto** 0);
	. . .
	if state = address or (state = ras) **then**
	mux <= dram_a;
	else
	mux <= (**others** => 'Z');
	end if;

In, Out, Buffer, Inout

Description	Example

• 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 reason, some designers standardize on mode buffer as well.
• 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.
	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;
	-- enter your statements here --
	
	end behavior;

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_logc_1164, numeric_bit, and nmeric_std packages.

Miscellaneous Operators

Description	Example
• Operator: **, abs, not . • The not operator is used frequently, the other two are rarely used for designs to be synthesized.	**variable** a, b: integer **range** 0 **to** 255; … a <= b**2;

• Predefined for any integer type (*, /, mod, rem), and any floating point type (*, /).	

Sign

Description	Example
• Operators: +, -. • Rarely used for synthesis. • Predefined for any numeric type (floating point or integer).	**variable** a, b, c: integer **range** 0 **to** 255; … b <= - (a + 5);

Adding Operators

Description	Example
• Operators: +,- • Used frequently to describe incrementers, decrementers, adders, and subtractors. • Predefined for any numeric type.	**signal** count: integer **range** 0 to 255; … count <= count -5;

Shift Operators

Description	Example
• Operators: sll, srl, sla, sra, rol, ror. • Used occasionally. • Predefined for any one-dimensional array with elements of type bit or Boolean. Overloaded for std_logic arrays.	**signal** a, b: bit_vector(4 downto 0); **signal** c: integer **range** 0 **to** 4; … a <= b ror c;

Relational Operators

Description	Example

• Operators: =, /=, <, <=, > >=.
• Used frequently for comparisons.
• Predefined for any type (both operands must be of same type).
	signal a, b: integer **range** 0 **to** 255;
	signal c: std_logic;
	…
	if a >= b **then**
	c <= '1';
	else
	c <= '0';
	end if;

Logical Operators

Description	Example

• 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.
	signal a, b, c, d: std_logic;
	…
	a <= b nand c;
	d <= a xor b;