VHDL Looping

Prof. James L. Frankel
Harvard University

Version of 7:46 PM 4-Dec-2021

Infinite **Loop** Statement

- Repeat statements forever

  ⟦ *loop*_label : ⟧
  loop
  ⦃ statement ⦄
  end loop ⟦ *loop*_label ⟧ ;

  - ⦃ <contents> ⦄ means zero or more repetitions of <contents>
  - ⟦ <contents> ⟧ means that <contents> is optional

- Use an **exit** statement to terminate the loop

- It is very unusual to use this infinite **loop** statement

Exit Statement

- Use to terminate the innermost loop, possibly conditionally

  exit ⟦ **when** condition ⟧ ;

  - ⟦ <contents> ⟧ means that <contents> is optional

- Use to terminate a labelled loop, possibly conditionally

  exit *loop*_label ⟦ **when** condition ⟧ ;

  - ⟦ <contents> ⟧ means that <contents> is optional

- If the infinite **loop** statement is used, an **exit** statement is almost always used to terminate the loop

Loop Statement Using **while**

- Repeat statements zero or more times

⟦ *loop*_label : ⟧
while condition **loop**
⦃ statement ⦄
end loop ⟦ *loop*_label ⟧ ;

  - ⦃ <contents> ⦄ means zero or more repetitions of <contents>
  - ⟦ <contents> ⟧ means that <contents> is optional

Loop Statement Using **for**

- Repeat statements zero or more times

  ⟦ *loop*_label : ⟧
  for loop_parameter **in** range **loop**
  ⦃ statement ⦄
  end loop ⟦ *loop*_label ⟧ ;

  - ⦃ <contents> ⦄ means zero or more repetitions of <contents>
  - ⟦ <contents> ⟧ means that <contents> is optional

- See the architecture **behavioral_loop** of **comparator4BitStdSeveralConfig** in example **comparator4bitstdseveralconfig.vhd** on the class web site

Generate Statement Using **for**

- Iterative replication of *concurrent* statements

*generate*_label :
for generate_parameter **in** range **generate**
⟦ { declarations }
begin ⟧
{ concurrent_statement }
end generate ⟦ *generate*_label ⟧ ;

  - { <contents> } means zero or more repetitions of <contents>
  - ⟦ <contents> ⟧ means that <contents> is optional

Generate Statement Using **if**

- Conditional instantiation of *concurrent* statements

  *generate*_label :
  if condition **generate**
  [{ declarations }
  begin]
  { concurrent_statement }
  end generate [*generate*_label] ;

  - { <contents> } means zero or more repetitions of <contents>
  - [<contents>] means that <contents> is optional

- Use the **generate** statement using **if** when there is an exceptional case to the **generate** statement using **for**

Example of **Generate** Statement Using **for**

- See the second to the last example in comparator4bitstdseveral.vhd

```
architecture structural_generate of comparator4BitStdSeveral is
  attribute chip_pin: string;
  attribute chip_pin of a: signal is "Y23, Y24, AA22, AA23";
  attribute chip_pin of b: signal is "AA24, AB23, AB24, AC24";
  attribute chip_pin of equal: signal is "H15";
  signal x: std_logic_vector(0 to 3);
begin
  generate_xnors: for i in 0 to 3 generate
    u0to3: xnor02 port map (
      a => a(i),
      b => b(i),
      q => x(i));
  end generate generate_xnors;
  u4: and04 port map (
    a => x(0),
    b => x(1),
    c => x(2),
    d => x(3),
    q => equal);
end architecture structural_generate;
```

Another Example of **Generate** Statement Using **for**

- See the last example in comparator4bitstdseveral.vhd

```
architecture structural_generate_alt of comparator4BitStdSeveral is
  attribute chip_pin: string;
  attribute chip_pin of a: signal is "Y23, Y24, AA22, AA23";
  attribute chip_pin of b: signal is "AA24, AB23, AB24, AC24";
  attribute chip_pin of equal: signal is "H15";
  signal x: std_logic_vector(0 to 3);
begin
  generate_xnors: for index in 0 to 3 generate
    u0: xnor02 port map (
      a => a(index),
      b => b(index),
      q => x(index));
  end generate generate_xnors;

  and_results: process(x)
    variable equalVar: std_logic;

  begin
    equalVar := '1';
    for index in 0 to 3 loop
      equalVar := equalVar and x(index);
    end loop;
    equal <= equalVar;
  end process and_results;
end architecture structural_generate_alt;
```

Using **Generic** in the Declaration of an Entity

```
entity register is
  generic (width: integer);
  port (clk, en, clr: in std_ulogic;
        d: in std_ulogic_vector(width-1 downto 0);
        q: out std_ulogic_vector(width-1 downto 0));
end entity register;
```

- The generic clause must precede the port clause
- Within the architecture for the entity, the generic can be used as a constant
- Allows a parameterized entity to be designed

Instantiating a **Generic** Entity

- Precede the **port map** with a **generic map** clause, as shown here:

```
u0: regStdUlogicWithSyncClearGeneric
    generic map (width => 8)
    port map (
      clk => clock_50,
      en => enableReg,
      clear => resetReg,
      d => sw17 & sw16 & sw15 & sw14 & sw13 & sw12 & sw11 & sw10,
      q(7) => ledr17,
      q(6) => ledr16,
      q(5) => ledr15,
      q(4) => ledr14,
      q(3) => ledr13,
      q(2) => ledr12,
      q(1) => ledr11,
      q(0) => ledr10);
```