If statement ::= if condition then

sequence of sequential statements

{elsif condition then

sequence of sequential statements }

[else

sequence of sequential statements ]

end if;
entity IFSTMT is
  port (          RSTn, CLK, EN, PL : in  bit;
                DATA        : in  integer range 0 to 31;
                COUNT     : out integer range 0 to 31);
end IFSTMT;

architecture RTL of IFSTMT is
  signal COUNT_VALUE : integer range 0 to 31;
begin
  p0 : process (RSTn, CLK)   begin
    if (RSTn = '0') then
      COUNT_VALUE <= 0;
    elsif (CLK'event and CLK = '1') then
      if (PL = '1') then
        COUNT_VALUE <= DATA;
      elsif (EN = '1') then
        if (COUNT_VALUE = 31) then
          COUNT_VALUE <= 0;
        else
          COUNT_VALUE <= COUNT_VALUE + 1;
        end if;
      end if;
    end if;
  end if;   end process;
  COUNT <= COUNT_VALUE;
end RTL;

The signal COUNT is of the **OUT mode** so it cannot be read.

A **temporary signal** COUNT_VALUE is used to calculate the COUNT value.

Then COUNT_VALUE is assigned to the output port COUNT **outside the process**.
REMEMBER THIS TYPICAL CONSTRUCTION:

- The signal **COUNT** is of the **OUT mode** so it cannot be read.
- A **temporary signal** **COUNT_VALUE** is used to calculate the COUNT value.
- Then **COUNT_VALUE** is assigned to the output port **COUNT** **outside the process**.
If statement

if (RSTn = '0') then
    COUNT_VALUE <= 0;
elsif (CLK'event and CLK = '1')
then
    if (PL = '1') then
        COUNT_VALUE <= DATA;
    elsif (EN = '1') then
        if (COUNT_VALUE = 31)
        then
            COUNT_VALUE <= 0;
        else
            COUNT_VALUE <= COUNT_VALUE + 1;
        end if;
    end if;
end if;

Let us observe this counter simulation
If statement

FIGURE 4.5 5-bit counter simulation waveform.
Case statement ::= case expression is
  when choice(s)=>
  sequence of sequential statements
  [when choices(s)=>
  sequence of sequential statements]
end case;
package PACK is
  type month_type is (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC);
end PACK;

use work.PACK.all;
entity CASESTMT is
  port (MONTH : in month_type; LEAP : in boolean; DAYS : out integer);
end CASESTMT;
architecture RTL of CASESTMT is
begin
  p0 : process (LEAP, MONTH)
case MONTH is
  when FEB =>
    if LEAP then
      DAYS <= 29;
    else
      DAYS <= 28;
    end if;
  when APR | JUN | SEP | NOV =>
    DAYS <= 30;
  when JUL to AUG =>
    DAYS <= 31;
  when others =>
    DAYS <= 31;
end case;
end process;
end RTL;
Case statement

FIGURE 4.7 CASESTMT simulation waveform.
Loop statement ::= 
[loop_label:] [while condition | for identifier in discrete_range] loop 
sequence of sequential statements 
end loop [loop_label];
entity LOOPSTMT is
end LOOPSTMT;
architecture RTL of LOOPSTMT is   type arytype is array (0 to 9) of integer;
   signal A : arytype := (1, 2, 3, 4, 11, 6, 7, 23, 9, 10);
   signal TOTAL : integer := 0;
beginsignal A : arytype := (1, 2, 3, 4, 11, 6, 7, 23, 9, 10);
   signal TOTAL : integer := 0;
beginsignal A : arytype := (1, 2, 3, 4, 11, 6, 7, 23, 9, 10);
   signal TOTAL : integer := 0;
beginsignal A : arytype := (1, 2, 3, 4, 11, 6, 7, 23, 9, 10);
   signal TOTAL : integer := 0;
beginp0 : process (A)
   variable sum : integer := 0;
   variable i : integer := 20;
beginsum := 0;
end process;loop1 : for i in 0 to 9 loop
-- notice that i is local in loop1
exit loop1 when A(i) > 20;
next when A(i) > 10;
sum := sum + A(i);
end loop loop1;
if i = 20 then
   TOTAL <= -33;
else
   TOTAL <= sum;
end if;
end process;
end RTL;
FIGURE 4.8 Simulation waveform for the LOOPSTMT.
begin
    sum := 0;
    loop1 : for i in 0 to 9 loop
        exit loop1 when A(i) > 20;
        next when A(i) > 10;
        sum := sum + A(i);
    end loop loop1;
    if i = 20 then
        TOTAL <= -33;
    else
        TOTAL <= sum;
    end if;
end process;

Note:
The looping identifier i is not visible outside the loop statement, and the local variable i is not the same as the looping identifier. Variable i is used to assign the signal TOTAL.
Next statement

- **Next_statement** ::= next [loop_label][**when** condition ];

- Must be enclosed by a loop statement with the same loop label, and the next statement applies to that loop statement.

- If the loop label is not specified, it always applies to the innermost level of the loop statements.
Exit statement

`Exit_statement ::= exit [loop_label][when condition];`

- Must be enclosed by a loop statement with the same loop label, and the exit applies to that loop statement.
- If the loop label is not specified, the exit always applies to the innermost level of the loop statements.
entity EXITSTMT is
end EXITSTMT;

architecture BEH of EXITSTMT is
  type matrix is array (1 to 5, 1 to 4) of integer;
  constant TABLE : matrix :=
    ( (1, 2, 3, 4),
      (2, 8, 1, 0),
      (8, 5, 3, 7),
      (3, 0, 2, 1),
      (1, 1, 0, 2) );
begin
  p0 : process
    variable NUMROW, ROWSUM : integer := 0;
    variable ROWDONE, ALLDONE : bit;
    begin
      ALLDONE := '0';
      outloop : for i in matrix'range(1) loop
        ROWSUM := 0;
        ROWDONE := '0';
        ... matrix'range(2) loop
        ROWSUM := ROWSUM + TABLE (i, j);
        if (ROWSUM > 10) then
          NUMROW := NUMROW + 1;
          exit outloop when NUMROW = 2;
          exit; -- get out of inloop
        end if;
        wait for 20 ns;
      end loop inloop;
ROWDONE := '1';
wait for 20 ns;
end loop outloop;
ALLDONE := '1';
wait for 60 ns;
end process;
end BEH;
Sources

- VLSI, Ohio University, Starzyk