Computer-Aided Design

Sequential & Concurrent statements

Mahdi Aminian

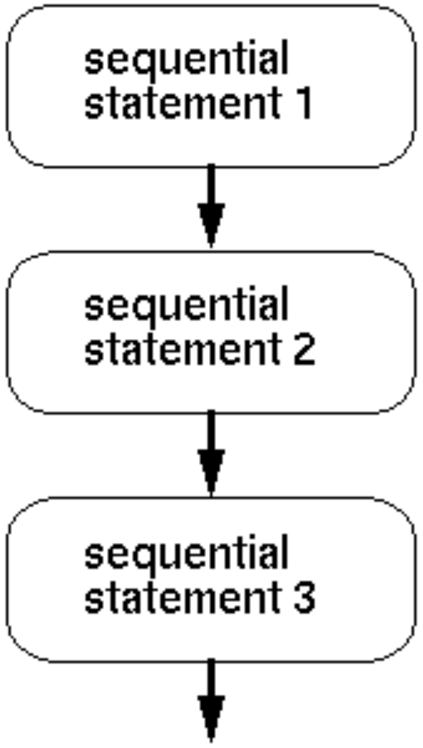
The University Of Guilan
Rasht - Iran

Some slides courtesy of:
- "Hardware Systems Modeling", A. Vachoux, EPFL
- CAD slides from Dr. Saheb Zamani

Sequential Statements

Sequential Statements

- Executed according to the order in which they appear
- Permitted only within processes and subprograms
- Used to describe algorithms



IF Statement

```
if CONDITION then
  -- sequential statements
end if;

 if CONDITION then
  -- sequential statements
else
  -- sequential statements
end if;

if CONDITION then
  -- sequential statements
elsif CONDITION then
  -- sequential statements
. . .
else
  -- sequential statements
end if;
```

- **Condition is a boolean expression**
- **Optional elsif sequence**
  - conditions may overlap
  - priority
- **Optional else path**
  - executed, if all conditions evaluate to false

IF Statement: Example

```
entity IF_STATEMENT is
     port (A, B, C, X : in   bit_vector (3 downto 0);
             Z            : out bit_vector (3 downto 0);
end IF_STATEMENT;
```

```
architecture EXAMPLE1 of IF_STATEMENT is
 begin
   process (A, B, C, X)
   begin
     Z <= A;
      if  (X = "1111")  then
       Z <= B;
      elsif  (X > "1000")  then
       Z <= C;
      end if;
   end process;
 end EXAMPLE1;
```

```
architecture EXAMPLE2 of IF_STATEMENT is
 begin
   process (A, B, C, X)
   begin

      if  (X = "1111")  then
       Z <= B;
      elsif  (X > "1000")  then
       Z <= C;
      else
       Z <= A;
      end if;
   end process;
 end EXAMPLE2;
```

IF Statement: Example

- شرط IF اول داراي بيشترين اولويت است (اگر شرط برقرار باشد بقية دستورات تا آخر skip مي شوند)
- دو شرط if و elsif همپوشاني دارند اما به دليل اولويت اولي ← Z <= B

```
architecture EXAMPLE1 of IF_STATEMENT is
 begin
   process (A, B, C, X)
   begin
     Z <= A;
      if  (X = "1111")  then
        Z <= B;
      elsif  (X > "1000")  then
        Z <= C;
      end if;
   end process;
 end EXAMPLE1;
```

```
architecture EXAMPLE2 of IF_STATEMENT is
  begin
    process (A, B, C, X)
    begin

      if  (X = "1111")  then
        Z <= B;
      elsif  (X > "1000")  then
        Z <= C;
      else
        Z <= A;
      end if;
    end process;
  end EXAMPLE2;
```

CASE Statement

```
case  EXPRESSION  is

    when  VALUE_1  =>
      -- sequential statements

    when  VALUE_2 | VALUE_3   =>
      -- sequential statements

    when  VALUE_4 to VALUE_N  =>
      -- sequential statements

    when   others =>
      -- sequential statements

 end   case ;
```

- **Choice options must not overlap**
- **All choice options have to be covered**
  - single values
  - selection of values ("|" means "or")
  - value range
  - "when others" covers all remaining choice options

- **type عبارت case header باید مطابق type مقادیر باشد.**

CASE Statement Example

```
entity CASE_STATEMENT is
   port (A, B, C, X : in   integer range 0 to 15;
            Z              : out integer range 0 to 15;
 end CASE_STATEMENT;

 architecture EXAMPLE of CASE_STATEMENT is
 begin
   process (A, B, C, X)
   begin
       case  X  is
         when  0  =>
           Z <= A;
         when  7  I  9  =>
           Z <= B;
         when  1 to 5  =>
           Z <= C;
         when others =>
           Z <= 0;
       end case;
   end process;
 end EXAMPLE;
```

Defining Ranges

```
entity RANGE_1 is
 port (A, B, C, X : in   integer range 0 to 15;
        Z           : out integer range 0 to 15;
 end RANGE_1;

 architecture EXAMPLE of RANGE_1 is
 begin
   process (A, B, C, X)
   begin
     case X is
       when 0 =>
         Z <= A;
       when 7 | 9 =>
         Z <= B;
        when 1 to 5 =>
         Z <= C;
       when others =>
         Z <= 0;
     end case;
   end process;
 end EXAMPLE;
```

```
entity RANGE_2 is
 port (A, B, C, X : in    std_logic_vector(3 downto 0);
        Z           : out  std_logic_vector(3 downto 0);
 end RANGE_2;

 architecture EXAMPLE of RANGE_2 is
 begin
   process (A, B, C, X)
   begin
     case X is
       when "0000" =>
         Z <= A;
       when "0111" | "1001" =>
         Z <= B;
        when "0001" to "0101" =>   -- wrong
         Z <= C;
       when others =>
         Z <= 0;
     end case;
   end process;
 end EXAMPLE;
```

Signed / Unsigned Numbers

Bit-Based Numerical Data Representation

- **Type unsigned and signed**

```
-- unsigned integer value
type unsigned is array (natural range <>) of std_logic;
-- signed integer value
type signed is array (natural range <>) of std_logic;
```

  - Examples:

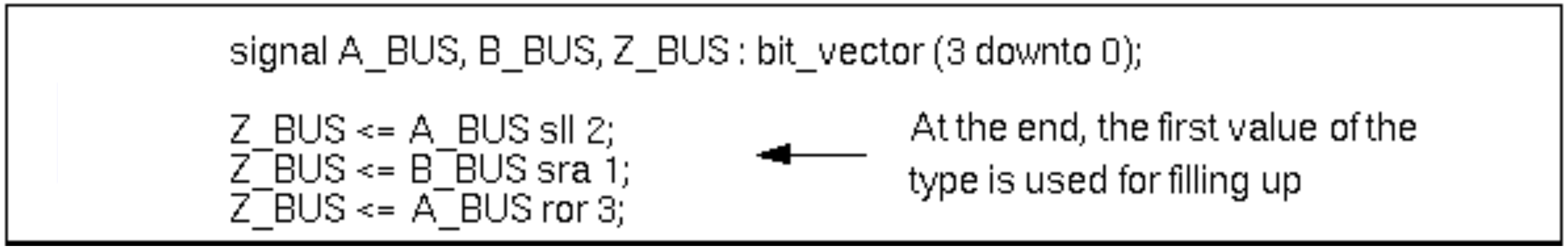
```
signal au : unsigned(7 downto 0);   -- values 0 to 255
signal as : signed(7 downto 0);     -- values -128 to 127 (2's-complement)
```

- **Supported operators:**

```
Relational:        =     /=    <     <=     >     >=
Logical:           and   or    nand  nor    not   xor   xnor
Arithmetic:        abs(1)  sign -(1)  +   -   *   /   rem   mod
Shift and rotate:  sll   srl   sla(2)  sra(2)  rol   ror
Concatenation:     &
(1) type signed only    (2) VHDL-2008 only
```

```
-- required context clause
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

Shift Operators

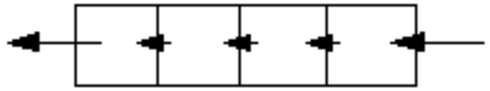


```
signal A_BUS, B_BUS, Z_BUS : bit_vector (3 downto 0);

Z_BUS <= A_BUS sll 2;
Z_BUS <= B_BUS sra 1;
Z_BUS <= A_BUS ror 3;
```

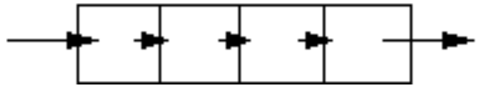
At the end, the first value of the type is used for filling up

- **Logical Shift**

sll

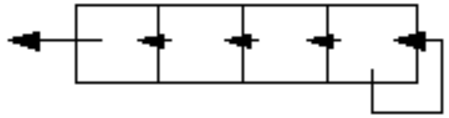


srl

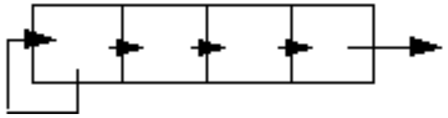


- **Arithmetic Shift**

sla

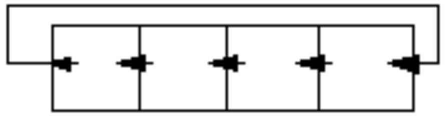


sra

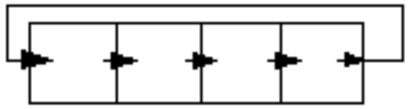


- **Rotation**

rol



ror



Working with std_(u)logic(_vector) and (un)signed

- Given the following declarations (also valid for variables)

```
signal A_sul  : std_ulogic;
signal B_sl   : std_logic;
signal C_sulv : std_ulogic_vector(7 downto 0);
signal D_slv  : std_logic_vector(7 downto 0);
signal E_uv   : unsigned(7 downto 0);
signal F_sv   : signed(7 downto 0);
```

- Compatible types

```
A_sul     <= B_sl;
B_sl      <= A_sul;
A_sul     <= C_sulv(7);
D_slv(2) <= C_sulv(0);
```

```
-- VHDL-2008 only
C_sulv <= D_slv;
D_slv  <= C_sulv;
```

- Type casting

```
C_sulv <= std_ulogic_vector(D_slv);
D_slv  <= std_logic_vector(C_sulv);
E_uv   <= unsigned(D_slv);
F_sv   <= signed(D_slv);
```

```
signal D2_slv, D3_slv : std_logic_vector(D_slv'range);
D_slv <= std_logic_vector(unsigned(D2_slv) + unsigned(D3_slv));
```

```
signal C2_sulv, C3_sulv : std_ulogic_vector(7 downto 0);
signal F2_sv : signed(8 downto 0);
F2_sv  <= signed('0' & C2_sulv) - signed('0' & C3_sulv);
```

Working with (un)signed and integer

- Given the following declarations (also valid for variables)

```
signal E_uv  : unsigned(7 downto 0);
signal F_sv  : signed(7 downto 0);
signal G_int : integer;
signal H_nat : natural;
```

- Conversion functions

```
G_int <= to_integer(E_uv);
H_nat <= to_integer(E_uv);

G_int <= to_integer(F_sv);
```

```
E_uv <= to_unsigned(G_int, E_uv'length);  -- to_unsigned(arg, size)
F_sv <= to_signed(G_int, F_sv'length);    -- to_signed(arg, size)
```

```
signal G2_int, G3_int : integer;
E_uv <= to_unsigned((G2_int + G3_int), E_uv'length);
F_sv <= to_signed((G2_int - G3_int), F_sv'length);
```

```
signal F2_sv, F3_sv : signed(7 downto 0);
G_int <= to_integer(F2_sv + F3_sv);
```

Variable

1-Bit Full Adder: Algorithmic Architecture

- **Variable objects can only be declared and used in a process**
  - It is forbidden to declare signals in a process
- **Sequential statements**
  - Variable assignment statement uses the delimiter ":="
- **Variables get their new values immediately**
- **Variables retain their state between process activations**

```
architecture algo of fadd1 is
   use ieee.numeric_std.all;
begin
   process (opa, opb, cin)
      variable tmp : unsigned(1 downto 0);
   begin
      tmp := (others => '0');
      if opa = '1' then tmp := tmp + 1; end if;
      if opb = '1' then tmp := tmp + 1; end if;
      if cin = '1' then tmp := tmp + 1; end if;
      if tmp > 1 then
         cout <= '1';
      else
         cout <= '0';
      end if;
      if tmp mod 2 = 0 then
         sum <= '0';
      else
         sum <= '1';
      end if;
   end process;
end architecture algo;
```

Signals or Variables?

Signals	Variables
♦ Represent hardware signals	♦ Represent (temporary) storage
♦ Global scope (design entity)	♦ Local scope (process)
♦ Complex data structure (driver)	♦ Memory location only
♦ Assignment with "<="	♦ Assignment with ":="
♦ Updated in one step after all processes have finished executing	♦ Updated immediately after the assignment
♦ Most appropriate for expressing concurrent hardware behavior	♦ Most appropriate for expressing algorithmic (procedural) behavior

- Signals(variables) can be assigned to variables(signals) of compatible types

For … Loop

FOR Loops

```
entity FOR_LOOP is
   port (A : in    integer range 0 to 3;
         Z  : out bit_vector (3 downto 0));
 end FOR_LOOP;

 architecture EXAMPLE of FOR_LOOP is
 begin
   process (A)
   begin
     Z <= "0000";
      for  I  in  0 to 3  loop
        if (A = I) then
          Z(I) <= `1`;
        end if;
      end loop;
   end process;
 end EXAMPLE;
```

- **Loop parameter is implicitly declared**
  - cannot be declared externally
  - read only access
  - not visible outside the loop
- **The loop parameter adopts all values from the range definition**
  - integer ranges
  - enumerated types

```
type T_STATE is (START, EXECUTE, FINISH);
…
signal CURRENT_STATE, NEXT_STATE : T_STATE ;
```

Loop Syntax

```
[LOOP_LABEL :]
 for IDENTIFIER in DISCRETE_RANGE loop
   -- sequential statements
 end loop [LOOP_LABEL] ;

 [LOOP_LABEL :]
 while CONDITION loop
   -- sequential statements
 end loop [LOOP_LABEL] ;
```

- **Optional label**
- **Use especially for nested loops**
- **Range attributes**
  - ‘low
  - ‘high
  - ‘range

Synthesis requirements:
- Loops must have a fixed range
- 'while' constructs usually cannot be synthesized

Loop Examples

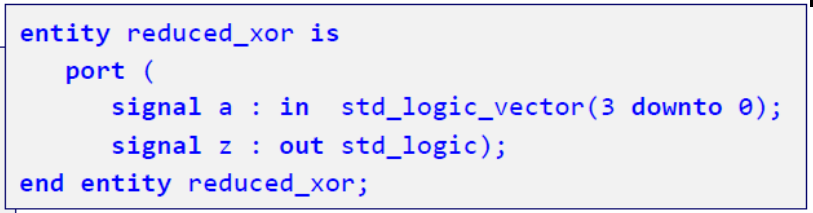
```
entity CONV_INT is
    port (VECTOR: in   std_logic_vector(7 downto 0);
          RESULT:  out integer);
end CONV_INT;
```

```
architecture A of CONV_INT is
 begin
   process(VECTOR)
     variable TMP: integer;

   begin
     TMP := 0;

     for I in 7 downto 0 loop
       if (VECTOR(I)='1') then
         TMP := TMP + 2**I;
       end if;
     end loop;

     RESULT <= TMP;
   end process;
 end A;
```

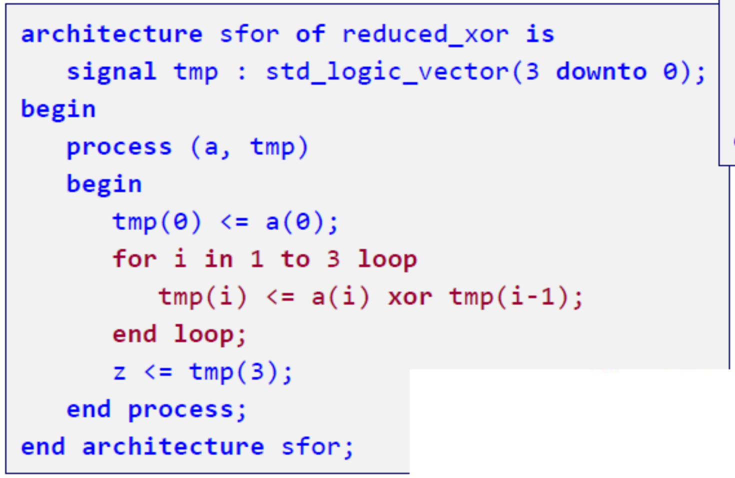
```
architecture B of CONV_INT
 is
 begin
   process(VECTOR)
     variable TMP: integer;

   begin
     TMP := 0;

     for I in VECTOR'range
 loop
       if (VECTOR(I)='1') then
         TMP := TMP + 2**I;
       end if;
     end loop;

     RESULT <= TMP;
   end process;
 end B;
```

```
architecture C of CONV_INT is
 begin
   process(VECTOR)
     variable TMP: integer;
     variable I      : integer;
   begin
     TMP := 0;
     I := VECTOR'high;
     while (I >= VECTOR'low)
 loop
       if (VECTOR(I)='1') then
         TMP := TMP + 2**I;
       end if;
       I := I - 1;
     end loop;
     RESULT <= TMP;
   end process;
 end C;
```

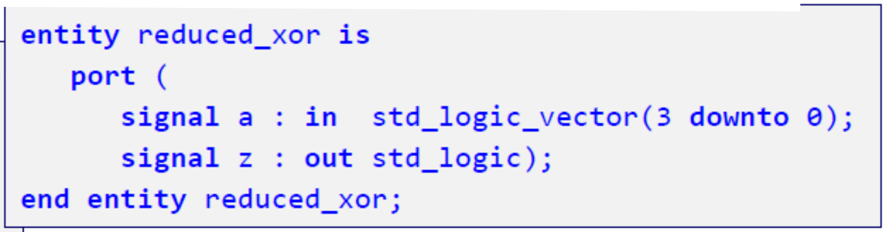
Loop Example



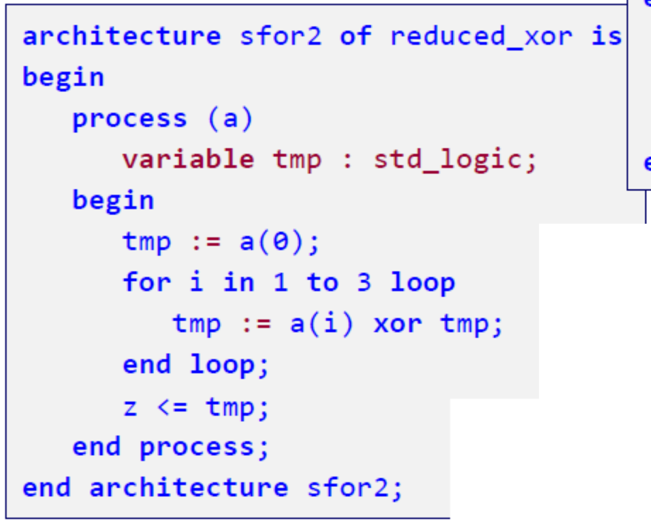
```
entity reduced_xor is
   port (
      signal a : in  std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity reduced_xor;
```



```
architecture sfor of reduced_xor is
   signal tmp : std_logic_vector(3 downto 0);
begin
   process (a, tmp)
   begin
      tmp(0) <= a(0);
      for i in 1 to 3 loop
         tmp(i) <= a(i) xor tmp(i-1);
      end loop;
      z <= tmp(3);
   end process;
end architecture sfor;
```



```
entity reduced_xor is
   port (
      signal a : in  std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity reduced_xor;
```



```
architecture sfor2 of reduced_xor is
begin
   process (a)
      variable tmp : std_logic;
   begin
      tmp := a(0);
      for i in 1 to 3 loop
         tmp := a(i) xor tmp;
      end loop;
      z <= tmp;
   end process;
end architecture sfor2;
```

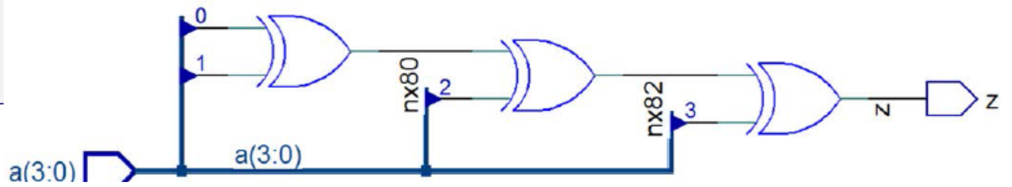
a(3:0)

Loop Example

```
architecture sfor of reduced_xor is
   signal tmp : std_logic_vector(3 downto 0);
begin
   process (a, tmp)
   begin
      tmp(0) <= a(0);
      for i in 1 to 3 loop
         tmp(i) <= a(i) xor tmp(i-1);
      end loop;
      z <= tmp(3);
   end process;
end architecture sfor;
```

```
entity reduced_xor is
   port (
      signal a : in  std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity reduced_xor;
```

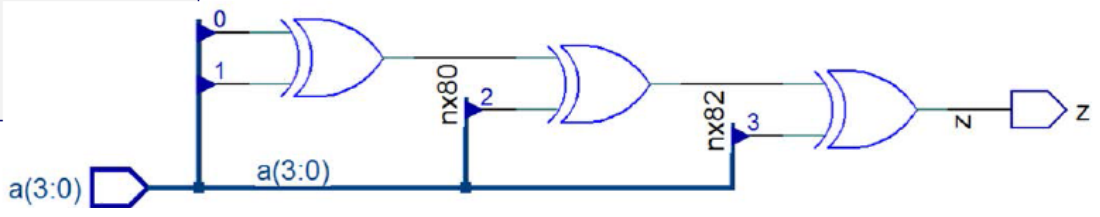
$$z = a(3) \oplus a(2) \oplus a(1) \oplus a(0)$$



```
architecture sfor2 of reduced_xor is
begin
   process (a)
      variable tmp : std_logic;
   begin
      tmp := a(0);
      for i in 1 to 3 loop
         tmp := a(i) xor tmp;
      end loop;
      z <= tmp;
   end process;
end architecture sfor2;
```

```
entity reduced_xor is
   port (
      signal a : in  std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity reduced_xor;
```

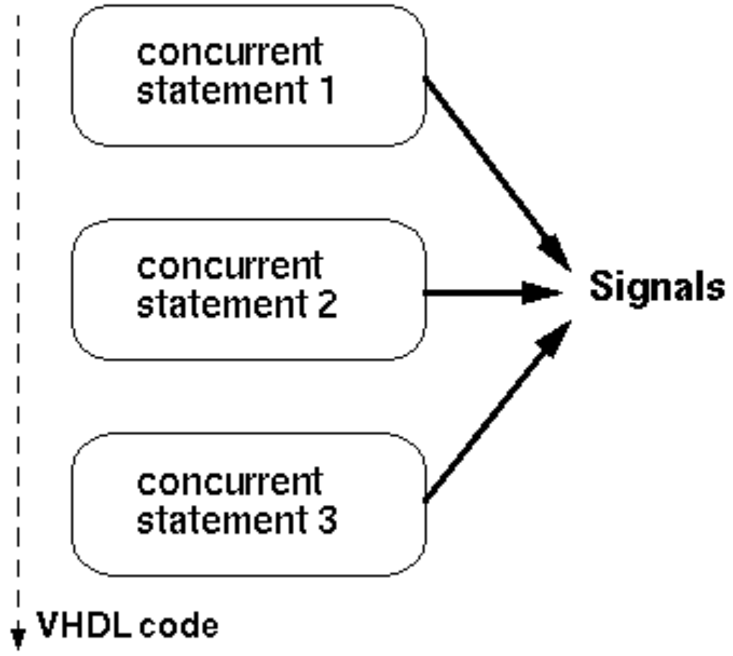
$$z = a(3) \oplus a(2) \oplus a(1) \oplus a(0)$$



Concurrent Statements

Concurrent Statements

- Concurrent statements are executed at the same time, independent of the order in which they appear



Conditional Signal Assignment

```
TARGET <= VALUE_1 when CONDITION_1 else
          VALUE_2 when CONDITION_2 else
          . . .
          VALUE_n;
```

- **Condition is a Boolean expression**
- **Mandatory else path, unless unconditional assignment**
  - conditions may overlap
  - priority
- **Similar to if ..., elsif ..., else constructs**

- هرگاه تغييري روي سيگنالهاي سمت راست رخ دهد اين انتساب بار ديگر ارزيابي مي شود (هميشه فعال است)

Conditional Signal Assignment (Example)

```
entity CONDITIONAL_ASSIGNMENT is
  port (A, B, C, X : in   bit_vector (3 downto 0);
          Z_CONC : out bit_vector (3 downto 0);
          Z_SEQ    : out bit_vector (3 downto 0));
end CONDITIONAL_ASSIGNMENT;

architecture EXAMPLE of CONDITIONAL_ASSIGNMENT i
s
begin
  -- Concurrent version of conditional signal assignment
  Z_CONC <= B when X = "1111" else
                  C when X > "1000" else
                  A;
 -- Equivalent sequential statements
  process (A, B, C, X)
  begin
     if  (X = "1111")  then
       Z_SEQ <= B
     elsif  (X > "1000")  then
       Z_SEQ <= C;
    else
       Z_SEQ <= A;
     end if;
  end process;
end EXAMPLE;
```

- توجه: در پروسس، همة سيگنالهاي سمت راست انتساب در ليست حساسيت آمده اند.

Selected Signal Assignment

```
with EXPRESSION select

   TARGET <= VALUE_1 when CHOICE_1,

                     VALUE_2 when CHOICE_2 | CHOICE_3,

                     VALUE_3 when CHOICE_4 to CHOICE_5,
                     . . .
                     VALUE_n when others;
```

- **Choice options must not overlap**
- **All choice options have to be covered**
  - single values
  - value range
  - selection of values
    ("|" means "or")
  - "when others" covers all remaining choice options
- **Similar to case ..., when ... constructs**

Selected Signal Assignment : Example

```
entity SELECTED_ASSIGNMENT is
    port (A, B, C, X : in   integer range 0 to 15;
           Z_CONC : out integer range 0 to 15;
           Z_SEQ    : out integer range 0 to 15);
  end SELECTED_ASSIGNMENT;

  architecture EXAMPLE of SELECTED_ASSIGNMENT is
  begin
    -- Concurrent version of selected signal assignment
    with X select
      Z_CONC <= A when 0,
                      B when 7 I 9,
                      C when 1 to 5,
                      0 when others;

    -- Equivalent sequential statements
    process (A, B, C, X)
    begin
      case X is
        when 0         => Z_SEQ <= A;
        when 7 I 9     => Z_SEQ <= B;
        when 1 to 5  => Z_SEQ <= C;
        when others => Z_SEQ <= 0;
    end process;
  end EXAMPLE;
```

Concurrent Statements: Summary

- **Modeling of multiplexers**
  - conditional signal assignment: decision based upon several signals
  - selected signal assignment: decision based upon values of a single signal