

DIGITAL LOGIC DESIGN

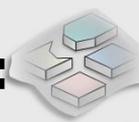
VHDL Coding for FPGAs

Unit 2

- ✓ *CONCURRENT DESCRIPTION*
 - *‘with-select’, ‘when-else’ statements*
 - *Arithmetic expressions, integer type, type conversions.*
 - *Examples: multiplexor, LUT, decoder, tri-state buffer*

✓ CONCURRENT DESCRIPTION

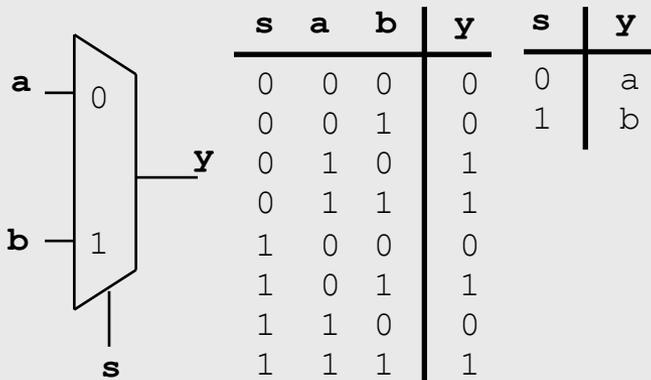
- In this description, the order of the statements is irrelevant: all the statements represent circuits that are working at the same time. This type of description is well-suited for combinatorial circuits.
- The use of sentences with ‘and’, ‘or’, ‘xor’, ‘nand’, ‘nor’, ‘xnor’, and ‘not’ is a basic instance of the concurrent description. It is sometimes called ‘horizontal description’.
- In Unit 4, the so-called ‘structural description’ is just a generalization of the concurrent description.
- Since we already know how to build circuits based on logic gates, we now present two concurrent assignment statements (*with - select*, *when - else*), which are far more powerful than the statements with logic gates when it comes to describe complex circuits.



■ CONCURRENT ASSIGNMENT STATEMENTS:

- **Selected Signal Assignment: WITH-SELECT:** This statement allows assigning values to a signal based on certain criterion.

■ MUX 2-to-1:



$$y = \bar{s}(a \bar{b} + a b) + s(\bar{a} b + a b)$$

$$y = \bar{s}a + sb$$

with s select: 's' specifies the selection criterion
 when specifies the value assigned to 'y' for each value of 's'
 when others: we need to include all the possible values of 's', that are 9 according to *std_logic* type.

```

library ieee;
use ieee.std_logic_1164.all;

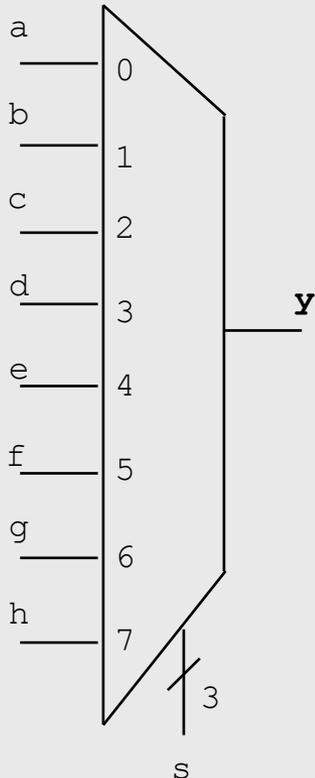
entity my_mux21 is
  port ( a, b, s: in std_logic;
         y: out std_logic);
end my_mux21;

architecture st of my_mux21 is
begin
  with s select
    y <= a when '0',
        b when others;
end st;

```

Selected Signal Assignment (WITH – SELECT):

MUX 8-to-1



```

library ieee;
use ieee.std_logic_1164.all;

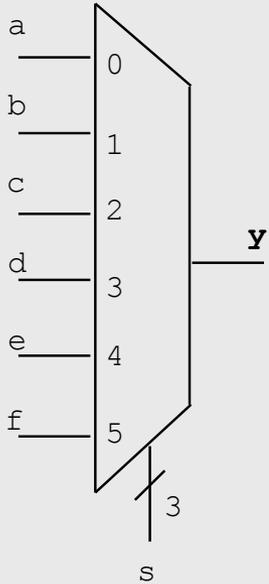
entity my_mux81 is
  port ( a,b,c,d,e,f,g,h: in std_logic;
         s: in std_logic_vector (2 downto 0);
         y: out std_logic);
end my_mux81;

architecture struct of my_mux81 is
begin
  with s select
    y <= a when "000", -- note ',' instead of ';'
         b when "001",
         c when "010",
         d when "011",
         e when "100",
         f when "101",
         g when "110",
         h when others;
end struct;

```



- Selected Signal Assignment (WITH – SELECT):
- MUX 6-to-1



```

library ieee;
use ieee.std_logic_1164.all;

entity my_mux61 is
  port ( a,b,c,d,e,f: in std_logic;
         s: in std_logic_vector (2 downto 0);
         y: out std_logic);
end my_mux61;

architecture struct of my_mux61 is
begin
  with s select
    y <= a when "000",
         b when "001",
         c when "010",
         d when "011",
         e when "100",
         f when "101",
         '- ' when others;
end struct;

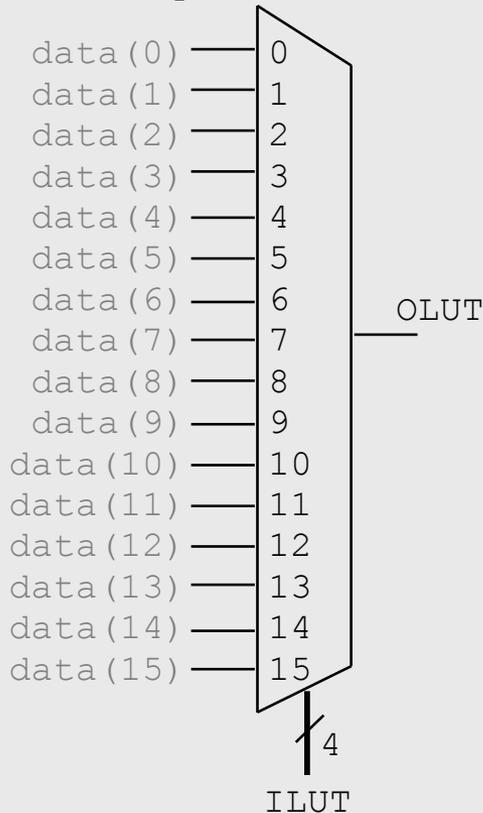
```

Value '-': Don't care output
 It helps the synthesizer to optimize the circuit



Selected Signal Statement

Example: 4-to-1 LUT



```

library ieee;
use ieee.std_logic_1164.all;

entity my4to1LUT is
  generic (data: std_logic_vector (15 downto 0) :=x"FEAB");
  port ( ILUT: in std_logic_vector (3 downto 0);
        OLUt: out std_logic);
end my4to1LUT;

architecture struct of my4to1LUT is
begin
  with ILUT select
    OLUt <= data(0) when "0000",
           data(1) when "0001",
           data(2) when "0010",
           data(3) when "0011",
           data(4) when "0100",
           data(5) when "0101",
           data(6) when "0110",
           data(7) when "0111",
           data(8) when "1000",
           data(9) when "1001",
           data(10) when "1010",
           data(11) when "1011",
           data(12) when "1100",
           data(13) when "1101",
           data(14) when "1110",
           data(15) when "1111",
           '0' when others;
end struct;

```

generic clause: Note how 'data' is not an input, but a constant parameter

➤ **my4to1LUT.zip:**
 my4to1LUT.vhd,
 tb_my4to1LUT.vhd

7-segment DECODER (outputs \geq inputs)

```
library ieee;
use ieee.std_logic_1164.all;

entity sevensseg is
    port ( bcd: in std_logic_vector (3 downto 0);
          sevensseg: out std_logic_vector (6 downto 0);
          EN: in std_logic_vector (3 downto 0));
end sevensseg;
```

```
architecture struct of sevensseg is
    signal leds: std_logic_vector (6 downto 0);
begin
```

```
-- | a | b | c | d | e | f | g |
-- |leds6|leds5|leds4|leds3|leds2|leds1|leds0|
```

```
with bcd select
```

```
    leds <= "1111110" when "0000",
           "0110000" when "0001",
           "1101101" when "0010",
           "1111001" when "0011",
           "0110011" when "0100",
           "1011011" when "0101",
           "1011111" when "0110",
           "1110000" when "0111",
           "1111111" when "1000",
           "1111011" when "1001",
           "-----" when others;
```

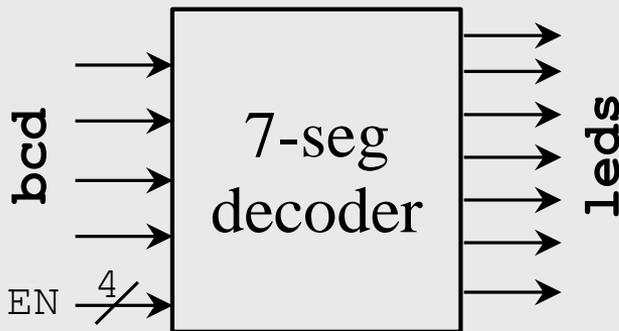
```
-- Nexys4: LEDs are active low.
```

```
-- Each 7-seg display has an active-low enable
```

```
EN <= "1110";
```

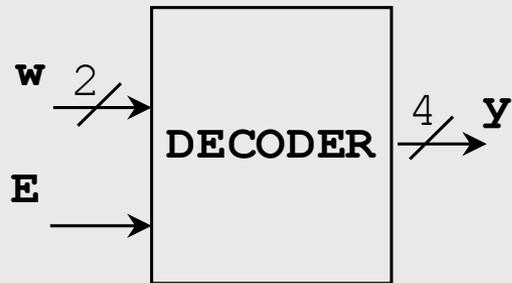
```
sevensseg <= not(leds);
```

```
end struct;
```



- **sevensseg.zip:** sevensseg.vhd,
tb_sevensseg.vhd, sevensseg.ucf

▪ **Decoder 2-to-4 with enable:**



```

library ieee;
use ieee.std_logic_1164.all;

entity dec2to4 is
    port (w: in std_logic_vector (1 downto 0);
          E: in std_logic;
          y: out std_logic_vector (3 downto 0));
end dec2to4;

architecture struct of dec2to4 is
    signal Ew: std_logic_vector (2 downto 0);
begin
    Ew <= E & w; -- concatenation
    with Ew select
        y <= "0001" when "100",
            "0010" when "101",
            "0100" when "110",
            "1000" when "111",
            "0000" when others;
end struct;

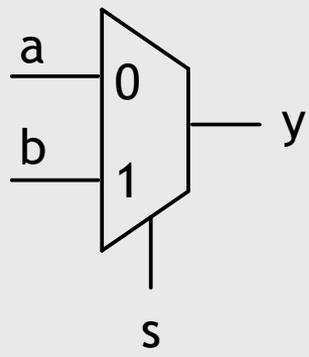
```

▪ **CONCURRENT ASSIGNMENT STATEMENTS :**

▪ **Conditional signal assignment: WHEN - ELSE:**

Similarly to the selected signal assignment, this statement allows a signal to take one out of many values based on a certain condition. The syntax however is different and it allows to describe circuits in a more compact fashion.

▪ **Example: MUX 2-to-1**



```

library ieee;
use ieee.std_logic_1164.all;

entity mux21_cond is
  port ( a, b, s: in std_logic;
         y: out std_logic);
end mux21_cond;

```

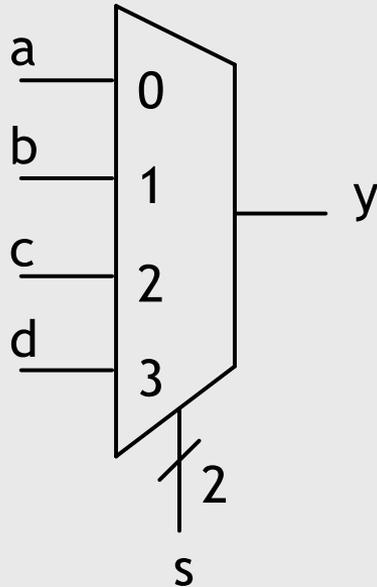
```

architecture est of mux21_cond is
begin
  y <= a when s = '0' else b;
end est;

```

- If the condition on **when** is FALSE, we assign a value to 'y' after **else**. This assignment can also be conditioned by another **when-else** clause (see next example)

- **Conditional signal assignment (WHEN - ELSE):**
- **Example:** MUX 4-to-1



- Note that the assignment on 'b' is conditioned by another **when-else** clause. Same for 'c'. Only the assignment of 'd' is not conditioned. There is no limit to the number of nested conditions.

```

library ieee;
use ieee.std_logic_1164.all;

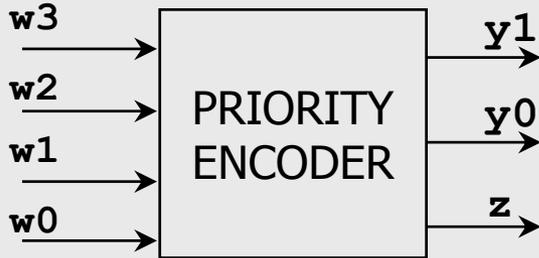
entity mux41_cond is
  port (a, b, c, d: in std_logic;
        s: in std_logic_vector (1 downto 0);
        y: out std_logic);
end mux41_cond;

architecture est of mux41_cond is
begin
  y <= a when s = "00" else
      b when s = "01" else
      c when s = "10" else
      d;
end est;

```

Conditional signal assignment (WHEN - ELSE):

Example: Priority Encoder 4-to-2



w ₃	w ₂	w ₁	w ₀	y ₁	y ₀	z
0	0	0	0	0	0	0
1	x	x	x	1	1	1
0	1	x	x	1	0	1
0	0	1	x	0	1	1
0	0	0	1	0	0	1

when-else has a priority level, \Rightarrow it is easy to describe a priority encoder. With **with-select**, this circuit description would be very tedious. \triangleright

```

library ieee;
use ieee.std_logic_1164.all;

entity my_prienc is
  port ( w: in std_logic_vector (3 downto 0);
        y: out std_logic_vector (1 downto 0);
        z: out std_logic);
end my_prienc;

architecture struct of my_prienc is
begin
  y <= "11" when w(3) = '1' else
      "10" when w(2) = '1' else
      "01" when w(1) = '1' else
      "00";
  z <= '0' when w = "0000" else '1';
  -- If no input is '1', z is '0'
end struct;

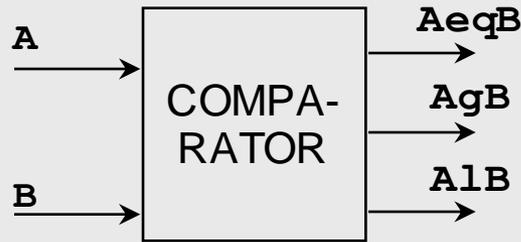
```

Example: Priority encoder 8 to 3:

\triangleright **my_prienc8to3.zip**: my_prienc8to3.vhd, tb_my_prienc8to3.vhd, my_prienc8to3.ucf

- **Conditional signal assignment (WHEN - ELSE):**

- **Example:** 4-bit comparator



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; -- unsigned #s
  
```

```

entity my_comp is
  port ( A,B: in std_logic_vector (3 downto 0);
         AeqB, AgB, AlB: out std_logic);
end my_comp;
  
```

```

architecture struct of my_comp is
begin
  
```

```

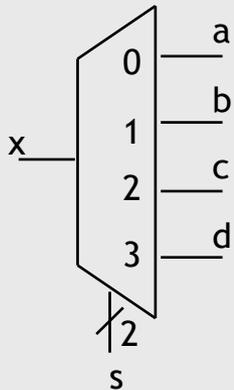
    AeqB <= '1' when A = B else '0';
    AgB  <= '1' when A > B else '0';
    AlB  <= '1' when A < B else '0';
  
```

```

end struct;
  
```

- Note the use of the operators '=', '>', '<' to compare integer numbers
- Always indicate what type of numbers we are working with. In the example we are using unsigned numbers. For 2's complement, use 'signed'.

- **Conditional Signal Assignment (WHEN - ELSE):**
- **Example:** Demultiplexor



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity my_demux is
  port ( s: in std_logic_vector (1 downto 0);
        x: in std_logic;
        a,b,c,d: out std_logic);
end my_demux;

```

```

architecture struct of my_demux is
begin

```

```

  a <= x when s = "00" else '0';
  b <= x when s = "01" else '0';
  c <= x when s = "10" else '0';
  d <= x when s = "11" else '0';

```

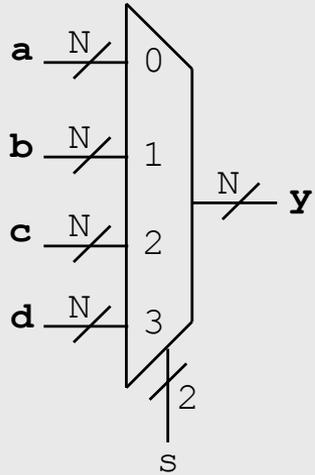
```

end struct;

```

- Concurrent
Description: Note that the order of the statements is not relevant.

- **Example:** 4-to-1 Bus Mux using with-select and when-else.



```

library ieee;
use ieee.std_logic_1164.all;

entity my_busmux4to1 is
  generic (N: INTEGER:= 8);
  port (a,b,c,d: in std_logic_vector (N-1 downto 0);
        s: in std_logic_vector (1 downto 0);
        y_r, y_t: out std_logic_vector (N-1 downto 0));
end my_busmux4to1;

architecture structure of my_busmux4to1 is
begin
  with s select
    y_r <= a when "00",
           b when "01",
           c when "10",
           d when others;

    y_t <= a when s = "00" else
           b when s = "01" else
           c when s = "10" else
           d;
end structure;

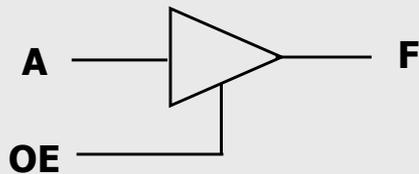
```

'generic' clause: It allows the definition of signals with customizable widths.

➤ **my_busmux4to1.zip:** my_busmux4to1.vhd, tb_my_busmux4to1.vhd

- **Conditional Signal Assignment (WHEN - ELSE):**

- **Example:** Tri-State Buffer



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity my_tristate is
  port ( A, OE: in std_logic;
         F: out std_logic);
end my_tristate;
```

```
architecture struct of my_tristate is
begin
```

```
  F <= A when OE = '1' else 'Z';
```

```
end struct;
```

➤ **my_tristate.zip:** my_tristate.vhd, tb_my_tristate.vhd,
my_tristate.ucf

- **Conditional Signal Assignment (WHEN - ELSE):**

- **Example:** Bidirectional Port

```
library ieee;
use ieee.std_logic_1164.all;
```

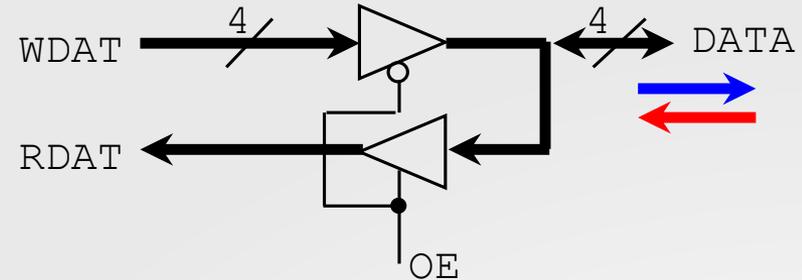
```
entity my_bidport is
  port ( WDAT: in std_logic_vector (3 downto 0);
        RDAT: out std_logic_vector (3 downto 0);
        OE: in std_logic;
        DATA: inout std_logic_vector (3 downto 0));
end my_bidport;
```

```
architecture struct of my_bidport is
```

```
begin
```

```
  DATA <= WDAT when OE = '0' else (others => 'Z');
  RDAT <= DATA when OE = '1' else (others => 'Z');
```

```
end struct;
```



▪ Example: Bidirectional Port (test bench)

```
library ieee;
use ieee.std_logic_1164.all;
```

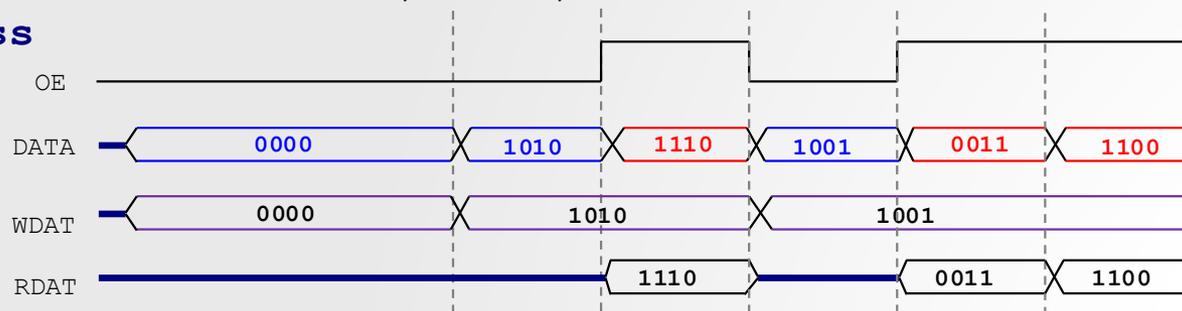
```
entity my_bidport is
end my_bidport;
```

```
architecture struct of my_bidport is
```

```
...
begin
  uut: my_bidport port map (WDAT, RDAT, OE, DATA);
```

```
  process
  begin
    DATA <= "ZZZZ"; wait for 100 ns;
    OE <= '0'; WDAT <= x"A"; DATA <= "ZZZZ"; wait for 20 ns;
    OE <= '1'; DATA <= x"E"; wait for 20 ns;
    OE <= '0'; WDAT <= x"9"; DATA <= "ZZZZ"; wait for 20 ns;
    OE <= '1'; DATA <= x"3"; wait for 20 ns;
    DATA <= x"C"; wait;
```

```
  end process
end;
```



To avoid data contention, make sure that **DATA = Z** when **DATA** is to be an output.

➤ **my_bidport.zip:** my_bidport.vhd, tb_my_bidport.vhd

✓ INTEGER DATA TYPE

- A signal of type `'integer'` represents a binary number. But we do not specify the number of bits for the signal, only the range of decimal values (this can be very convenient).
- Example:** 7-segment decoder. The BCD input is an integer from 0 to 9, requiring 4 bits (computed by the synthesizer).
- Drawback: the datatype `'integer'` does not allow access to the individual bits (unlike `'std_logic_vector'`)

```

library ieee;
use ieee.std_logic_1164.all;

entity sevenseg is
  port ( bcd: in integer range 0 to 9;
        -- bcd: 0000 to 1001 -> 4 bits required
        sevenseg: out std_logic_vector (6 downto 0);
        EN: in std_logic_vector (3 downto 0));
end sevenseg;

architecture struct of sevenseg is
  signal leds: std_logic_vector (6 downto 0);
begin
  -- | a | b | c | d | e | f | g |
  -- |leds6|leds5|leds4|leds3|leds2|leds1|leds0|
  with bcd select
    leds <= "1111110" when 0,
           "0110000" when 1,
           "1101101" when 2,
           "1111001" when 3,
           "0110011" when 4,
           "1011011" when 5,
           "1011111" when 6,
           "1110000" when 7,
           "1111111" when 8,
           "1111011" when 9;
  -- Nexys3: LEDs are active low.
  -- Each 7-seg display has an active-low enable
  EN <= "0111";
  sevenseg <= not(leds);
end struct;

```

■ ARITHMETIC STATEMENTS:

- We can use the operators $+$, $-$, and $*$ for rapid specification of arithmetic operations for integer numbers. We can also use the comparison statements: $>$, $<$, $=$, \neq , \geq , \leq .
- $+/-$: The operands and the result must have the same bit width.
- **Example:** Adder/subtractor implemented by multiplexing the adder and subtractor outputs:

- You must specify the representation for a,b (signed/unsigned):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

- `'_unsigned'` library: a-b: unsigned subtraction.

```
entity ex_addsub is
    port (a,b: in std_logic_vector (3 downto 0);
          s: in std_logic;
          f: out std_logic_vector (3 downto 0));
end ex_addsub;
```

- `'_signed'` library: a-b: signed (2C) subtraction

- Overflow not accounted for. Here, you might need to zero/sign-extend the input operands.

```
architecture structure of ex_addsub is
begin
    with s select
        f <= a+b when '0',
          a-b when others;
```

- This implements a fully combinational circuit.

```
end structure;
➤ ex_addsub.zip: ex_addsub.vhd,
  tb_ex_addsub.vhd, ex_addsub.xdc
```

■ ARITHMETIC STATEMENTS:

- **Example:** Multiplier (fully combinational):
- You can use the operator `*`. Make sure the output size is correct.
 - ✓ Input Operators: a (N_A bits), b (N_B) bits.
 - ✓ Output result: f (N_A+N_B bits)
- Operations:
 - ✓ `use ieee.std_logic_unsigned.all;` → A*B treats A, B as unsigned
 - ✓ `use ieee.std_logic_signed.all;` → A*B treats A, B as signed.

Important:

- You must select the representation (unsigned/signed) for the operands.
- `signed`: multiplication in 2C arithmetic.
- `unsigned`: unsigned multiplication

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
-- use ieee.std_logic_signed.all;

entity mult4x4 is
    port (a,b: in std_logic_vector (3 downto 0);
          f: out std_logic_vector (7 downto 0));
end mult4x4;

architecture structure of mult4x4 is
begin
    f <= a*b;
end structure;

```

■ ARITHMETIC STATEMENTS

- Arithmetic operators and comparison statements: You must indicate the representation: signed or unsigned.
- **First Approach:** we can specify the representation for all operations in the .vhd file using non-standard libraries (supported by Xilinx®):

✓ `use ieee.std_logic_unsigned.all;` → all operations will treat data as unsigned. Here $A < B$, $A - B$ treats A , B as unsigned.

✓ `use ieee.std_logic_signed.all;` → all operations will treat data as signed. Here $A < B$, $A - B$ treats A , B as signed.

```
signal A, B, S: std_logic_vector (3 downto 0);
```

```
f <= '1' when A < B else '0';
```

```
S <= A - B;
```

- **Second Approach:** using the following non-standard library, we can specify a representation for specific statements within the .vhd file:

✓ `use ieee.std_logic_arith.all;`

```
f <= '1' when unsigned(A) < unsigned(B) else '0';
```

```
S <= unsigned(A) - unsigned(B); → unsigned subtraction!
```

- We can also use 'signed':

```
S <= signed(A) - signed(B); → signed subtraction
```

▪ ARITHMETIC STATEMENTS

- The library `ieee.std_logic_arith.all` lets us perform conversions from bit vector to binary and viceversa:
 - `signal A: std_logic_vector (3 downto 0);`
 - `signal F: integer range 0 to 15;`
- `conv_integer (A)`: It converts the bit vector A into an integer.
 - `F <= conv_integer (A)`; It converts A into an integer. We need to indicate the representation of A, i.e., we also must use either the `ieee.std_logic_unsigned` (or `signed`) library.
 - `F <= conv_integer (unsigned (A))`; This omits the use of the `ieee.std_logic_unsigned` (or `signed`) library.
- `conv_std_logic_vector (F, N)`: It converts an integer data into a vector with N bits. The representation is given by the range of N.
 - `F <= 5;`
 - `A <= conv_std_logic_vector (F, 4)`; $\rightarrow A=0101$.