VietNam National University

University of Engineering and Technology

Modelling Combinational Circuit with VHDL

TS. Nguyễn Kiêm Hùng

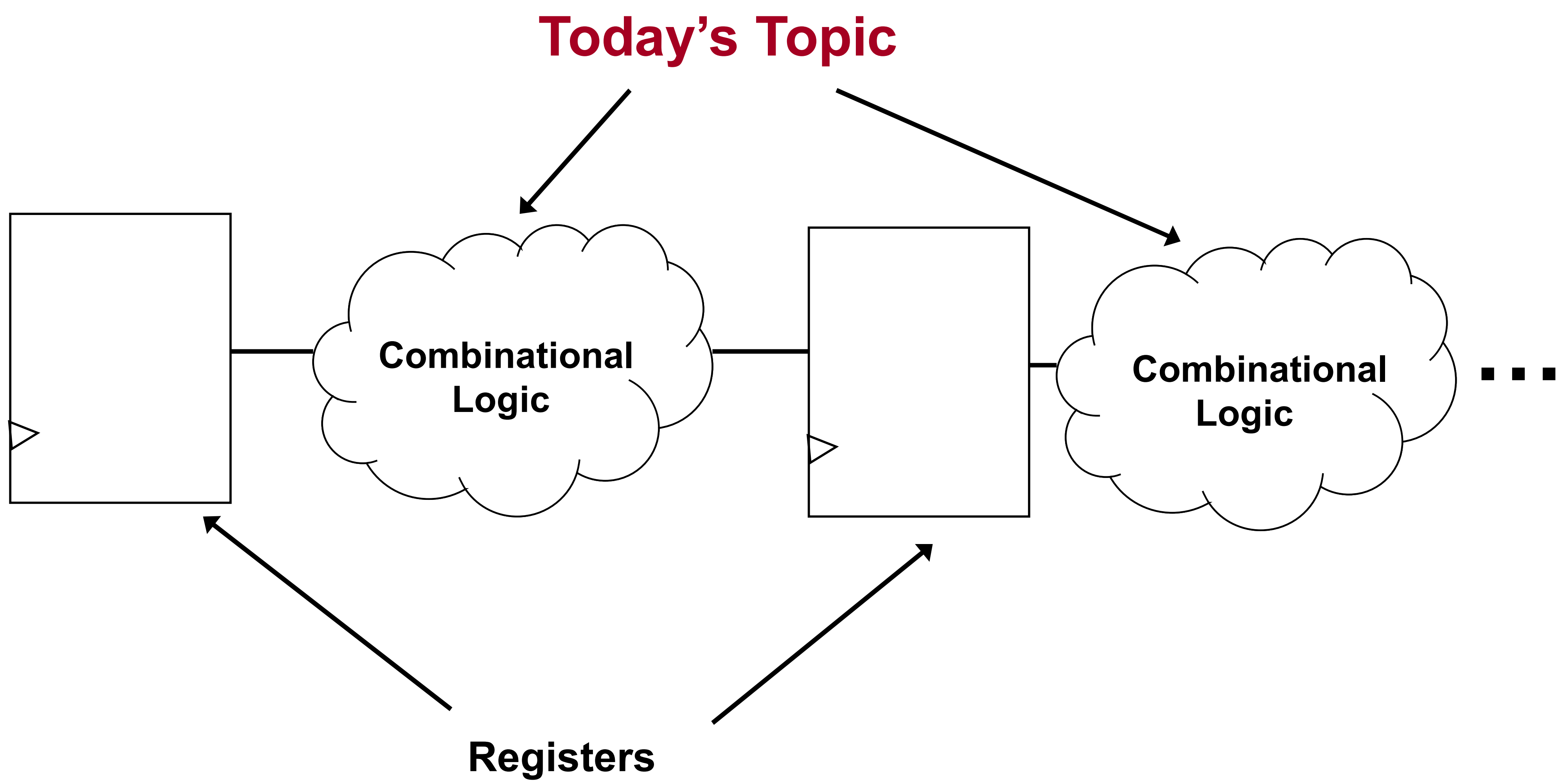
Email: kiemhung@vnu.edu.vn

Laboratory for Smart Integrated Systems

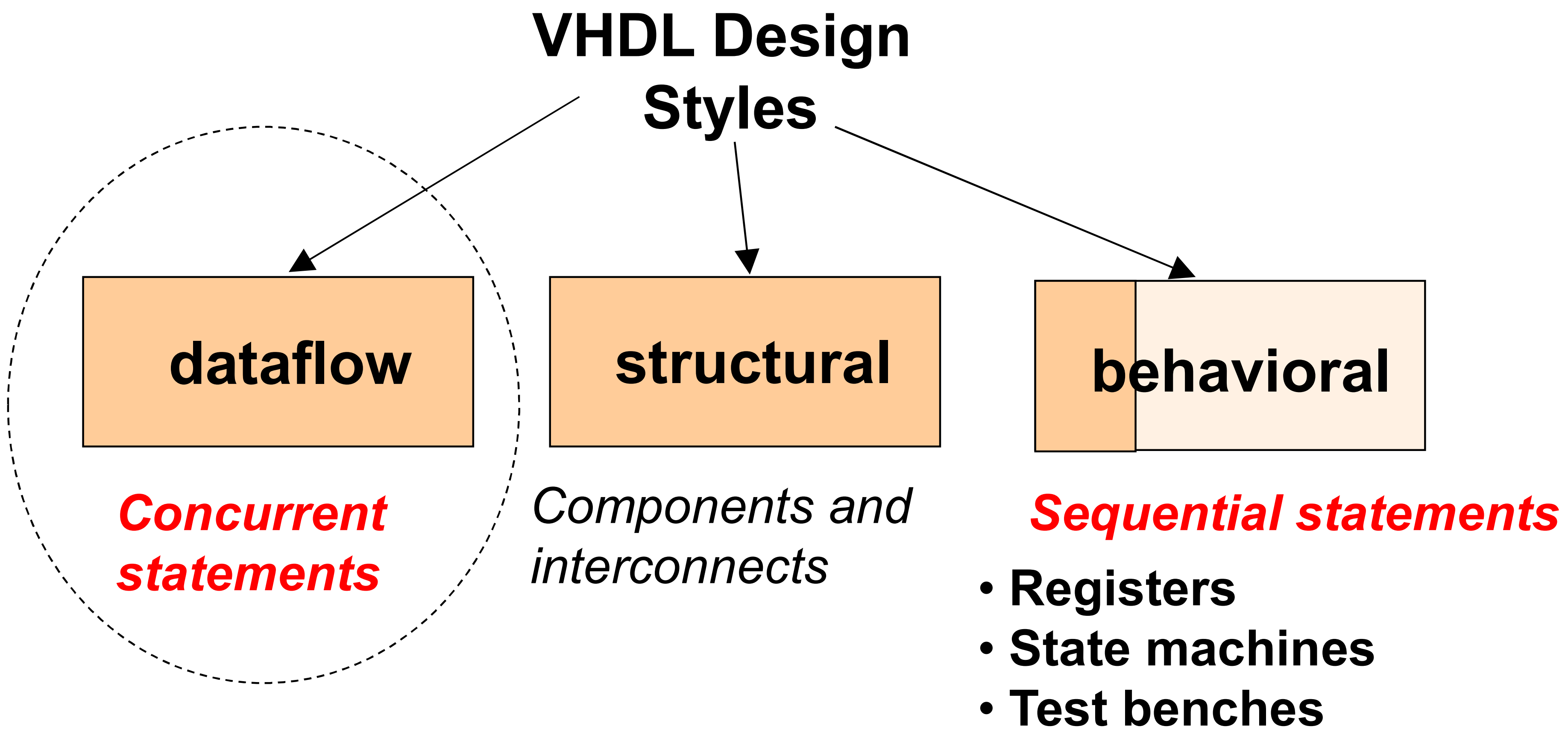
Objectives

- In this lecture you will be introduced to:
  - **Key VHDL constructs used to define combinational circuits**
  - **Commonly used combinational subcircuits**

Register Transfer Level (RTL) Design



VHDL Design Styles



Operators

Logic Operators

- Logic operators

```
and    or    nand    nor    xor    not    xnor
```

only in VHDL-93

- Logic operators precedence

Highest

$\downarrow$

Lowest

```
                    not
and    or    nand    nor    xor    xnor
```

Concatenation

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e, f: STD_LOGIC_VECTOR(7 DOWNTO 0);

a <= ″0000″;
b <= ″1111″;
c <= a & b;                   -- c = ″00001111″

d <= ‘0’ & ″0001111″;         -- d <= ″00001111″

e <= ‘0’ & ‘0’ & ‘0’ & ‘0’ & ‘1’ & ‘1’ &
     ‘1’ & ‘1’;               -- e <= ″00001111″
f <= (‘0’,‘0’,‘0’,‘0’,‘1’,‘1’,‘1’,‘1’) ;
                              -- f <= ″00001111″
```

Arithmetic Operators (1)

To use basic <u>arithmetic operations</u> involving std_logic_vectors you need to include the following library packages:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
or
USE ieee.std_logic_signed.all;
```

Arithmetic Operators (2)

You can use standard +, - , *, [/] operators to perform addition and subtraction:

```
signal A :  STD_LOGIC_VECTOR(3 downto 0);
signal B :  STD_LOGIC_VECTOR(3 downto 0);
signal C :  STD_LOGIC_VECTOR(3 downto 0);
   ……
C <= A + B;
```

Relational Operators

- Relational operators

=	/=	<	<=	>	>=

- Logic and relational operators precedence

Highest ↓ Lowest						
			not			
	=	/=	<	<=	>	>=
	and	or	nand	nor	xor	xnor

Precedence of logic and relational operators

The desired function: compare a = bc

(a equal to b and c or not?)

Incorrect

… when a = b **and** c else …

equivalent to

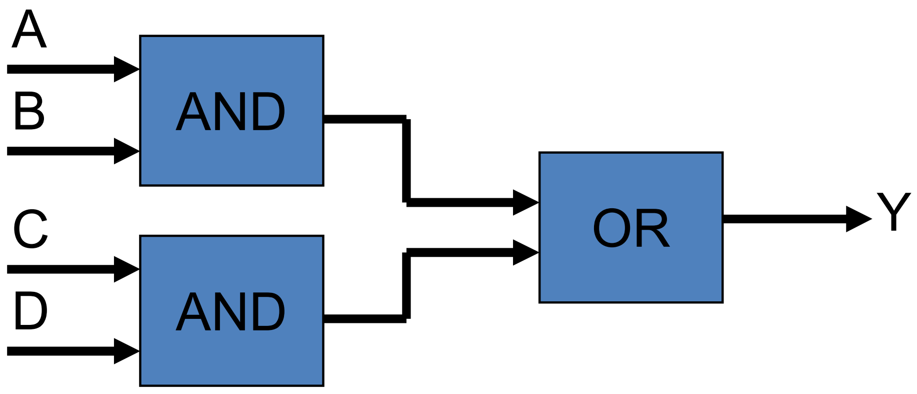
… when (a = b) **and** c else …

Correct

… when a = (b **and** c) else …

No Implied Precedence of Logic operator

Wanted: y **= ab + cd**



Incorrect

y <= a **and** b **or** c **and** d ;

equivalent to

y <= ((a **and** b) **or** c) **and** d ;

equivalent to

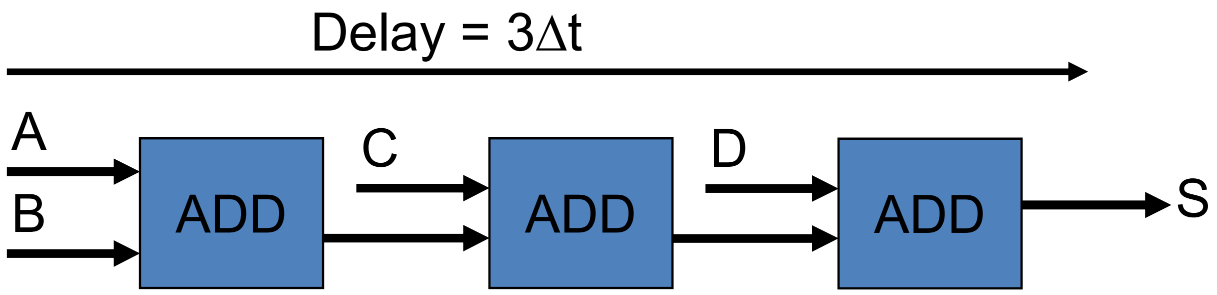
y = (ab + c)d

Correct

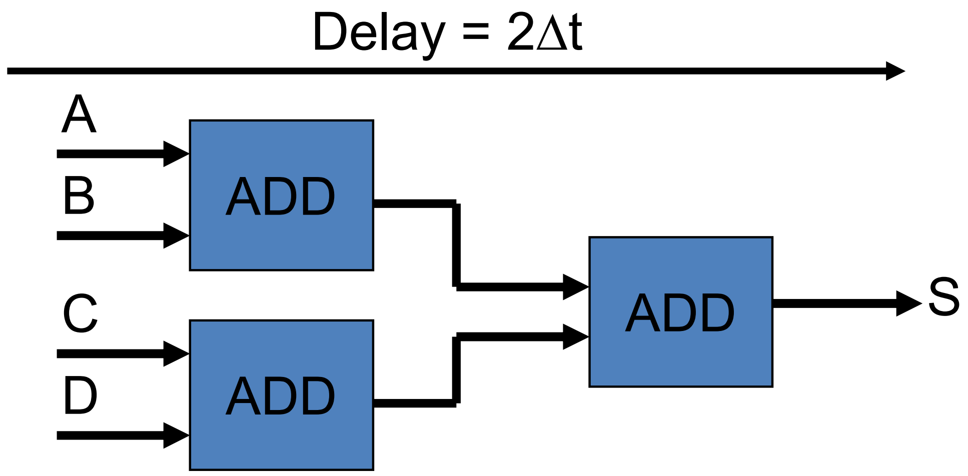
y <= (a **and** b) **or** (c **and** d) ;

Precedence of Arithmetic Operator

```
S <= A + B + C + D;
```



```
S <= (A + B) + (C + D);
```



Describing Combinational Logic Using Dataflow Design Style

Concurrent statements

Concurrent signal assignments are event triggered and executed as soon as an event on one of the signals occurs

- Simple signal assignment (⇐)
- Conditional signal assignment (when-else)
- Selected signal assignment (with-select-when)
- Generate scheme for equations (for-generate)

Simple Signal Assignment

- **Syntax:**

  [Label:] Target_signal <= expression;

- Example:

```
Sum <= (A xor B) xor Cin;
Carry <= (A and B);
Z <= (not X) or Y;
```

Simple Signal Assignment : Example

Full Adder

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a) Truth Table

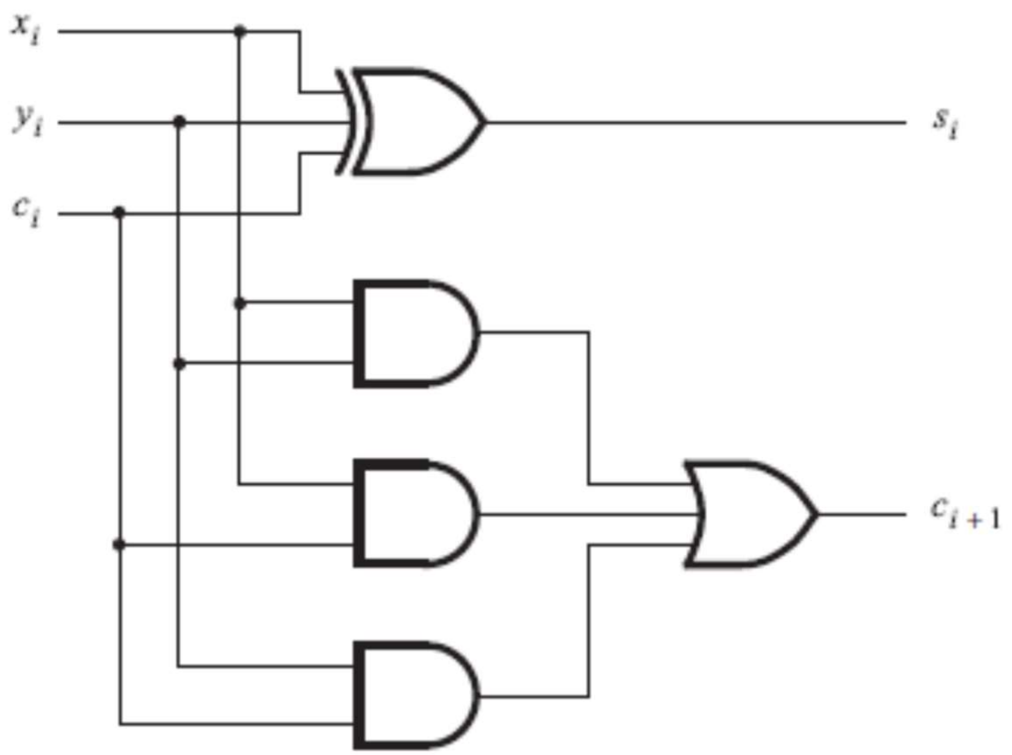
c_i \ x_iy_i	00	01	11	10
0		1		1
1	1		1	

$$s_i = x_i \oplus y_i \oplus c_i$$

c_i \ x_iy_i	00	01	11	10
0			1	
1		1	1	1

$$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$$

(b) Karnaugh maps



(c) Circuit

Simple Signal Assignment : Example (1)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY fulladder IS
        PORT ( x        : IN          STD_LOGIC ;
               y        : IN          STD_LOGIC ;
               cin      : IN          STD_LOGIC ;
               s        : OUT              STD_LOGIC ;
               cout     : OUT         STD_LOGIC ) ;
END fulladder ;
```

Simple Signal Assignment : Example (2)

```
ARCHITECTURE fulladd_dataflow OF fulladder IS
BEGIN
        U1: s <= (x XOR y) XOR cin ;
        U2: cout <= (x AND y) OR (cin AND x) OR (cin AND y) ;
END fulladd_dataflow ;
```

Simple Signal Assignment : Example (3)

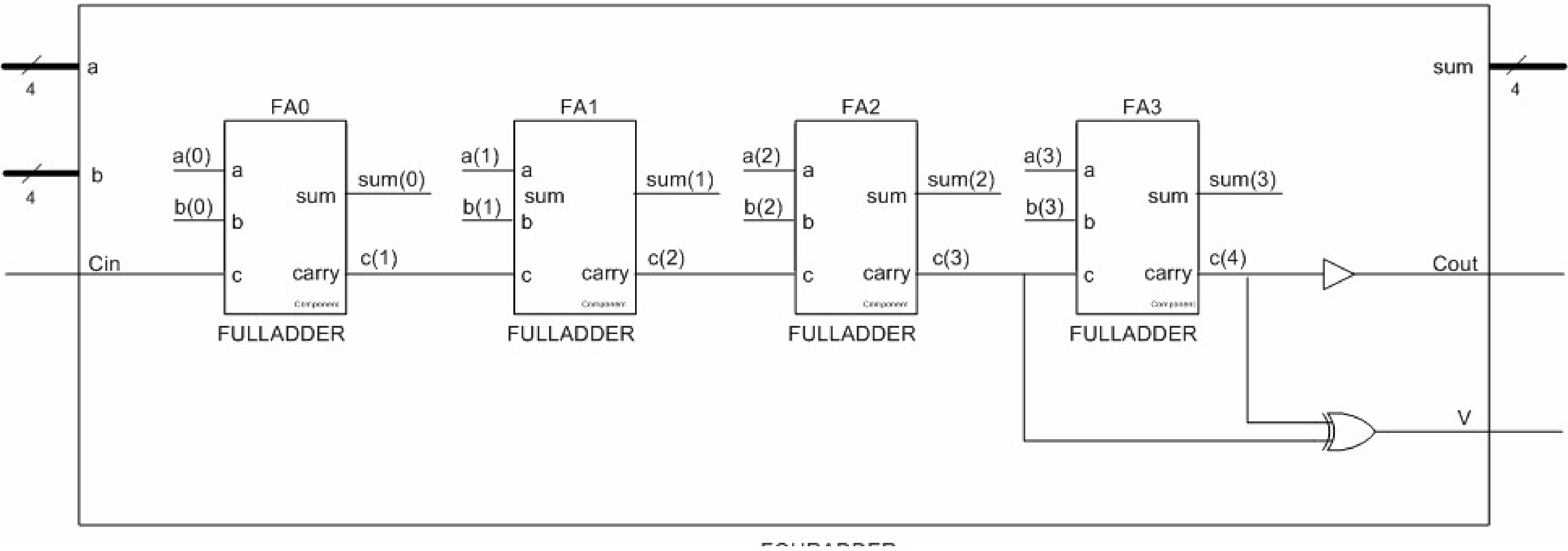
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE IEEE.std_logic_unsigned.all;

…
```

```
ARCHITECTURE fulladd_dataflow OF fulladd IS
-- define internal SUM signal
signal SUM: STD_LOGIC_VECTOR(1 downto 0);
BEGIN
        U0: SUM <= (‘0’ & x) + (‘0’ & y) + (‘0’ & cin);
        U1: COUT <= SUM(1);
        U2: S <= SUM(0);
END fulladd_dataflow ;
```

QUIZ

How to build a structural model of 4-bit adder by using the described Full Adder?

Structural Model of 4-bit Adder



Structural Model of 4-bit Adder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY ADDER_4BIT IS
        PORT ( x        : IN     STD_LOGIC _VECTOR (3 downto 0);
               y        : IN      STD_LOGIC _VECTOR (3 downto 0);
               cin      : IN     STD_LOGIC ;
               sum      : OUT   STD_LOGIC _VECTOR (3 downto 0);
               v, cout : OUT            STD_LOGIC ) ;
END ADDER_4BIT ;
```

Structural Model of 4-bit Adder

```
ARCHITECTURE adder_4bit_structure OF adder_4bit IS
   signal c: std_logic_vector (4 downto 0);
   -- component declaration
   component FULLADDER
            port (x, y, cin:  in std_logic;
                   s, cout:  out std_logic);
   end component;
BEGIN
  FA0: FULLADDER
            port map (x(0), y(0), cin, sum(0), c(1));
  FA1: FULLADDER
            port map (x(1), y(1), C(1), sum(1), c(2));
  FA2: FULLADDER
            port map (x(2), y(2), C(2), sum(2), c(3));
  FA3: FULLADDER
            port map (x(3), y(3), C(3), sum(3), c(4));
  Cout <= c(4);
  V <= c(3) xor c(4);
END adder_4bit_structure ;
```

Assigning Signal Values Using OTHERS

- **Syntax:**

  S <= (OTHERS => value) ;

  -- set each bit of the destination operand to Value

- **Example:**

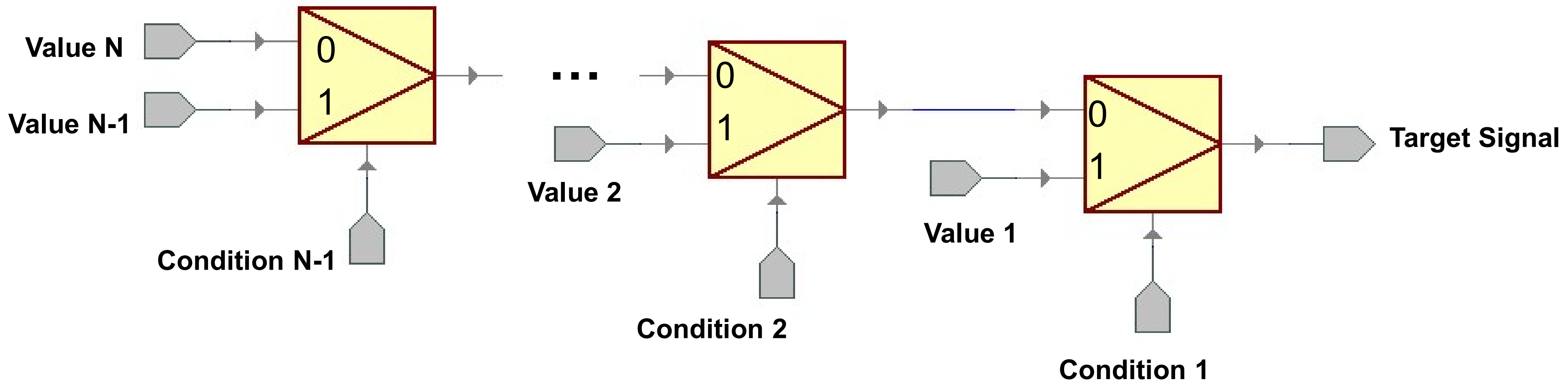
  S <= (OTHERS => ’0’) ;

  S <= (0 => ‘1’, OTHERS => ’0’) ;

Conditional Signal Assignment

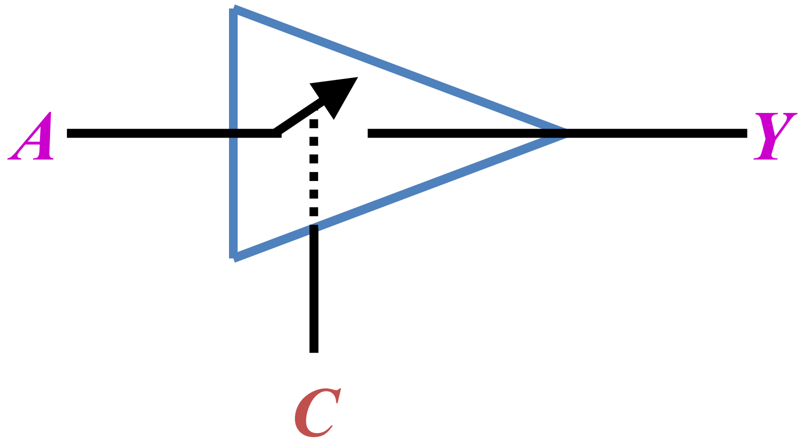
When - Else

```
[label:]
target_signal <= value1 WHEN condition1 ELSE
                 value2 WHEN condition2 ELSE
                    .  .  .
                 valueN-1 WHEN conditionN-1 ELSE
                 valueN;
```

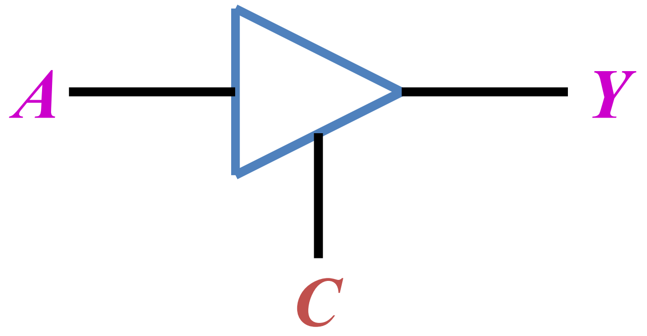


Example: Tri-state Buffer

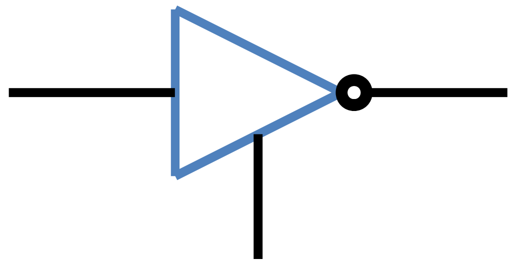
- Tri-State Buffer



C	*A*	*Y*
0	x	Hi-Z
1	0	0
1	1	1



- Tri-State Inverter



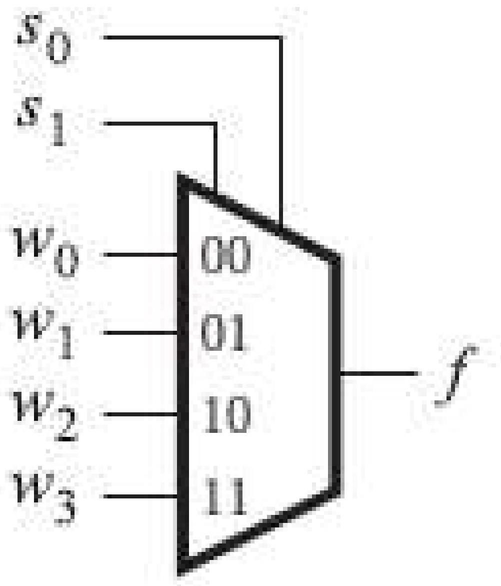
Example: Tri-state Buffer (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY tri_state IS
   PORT (  C  :  IN          STD_LOGIC;
           A  :  IN          STD_LOGIC_VECTOR(7 downto 0);
           Y  : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
         );
END tri_state;
```

Example: Tri-state Buffer (2)

```
ARCHITECTURE tri_state_dataflow OF tri_state IS
BEGIN
    Y <= A WHEN (C = ‘1’) ELSE
                (OTHERS => ‘Z’);
END tri_state_dataflow;
```

Example: 4-to-1 multiplexer (1)



(a) Graphical symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

Example: 4-to-1 multiplexer (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux4to1 IS
   PORT (A, B, C, D: IN        std_logic;
               SEL : IN       std_logic_vector (1 downto 0);
                Z: OUT       std_logic
               );
END mux4to1;
```

4-to-1 Multiplexer– example (2)

```
ARCHITECTURE mux4to1_dataflow OF mux4to1 IS
BEGIN




END mux4to1_dataflow;
```

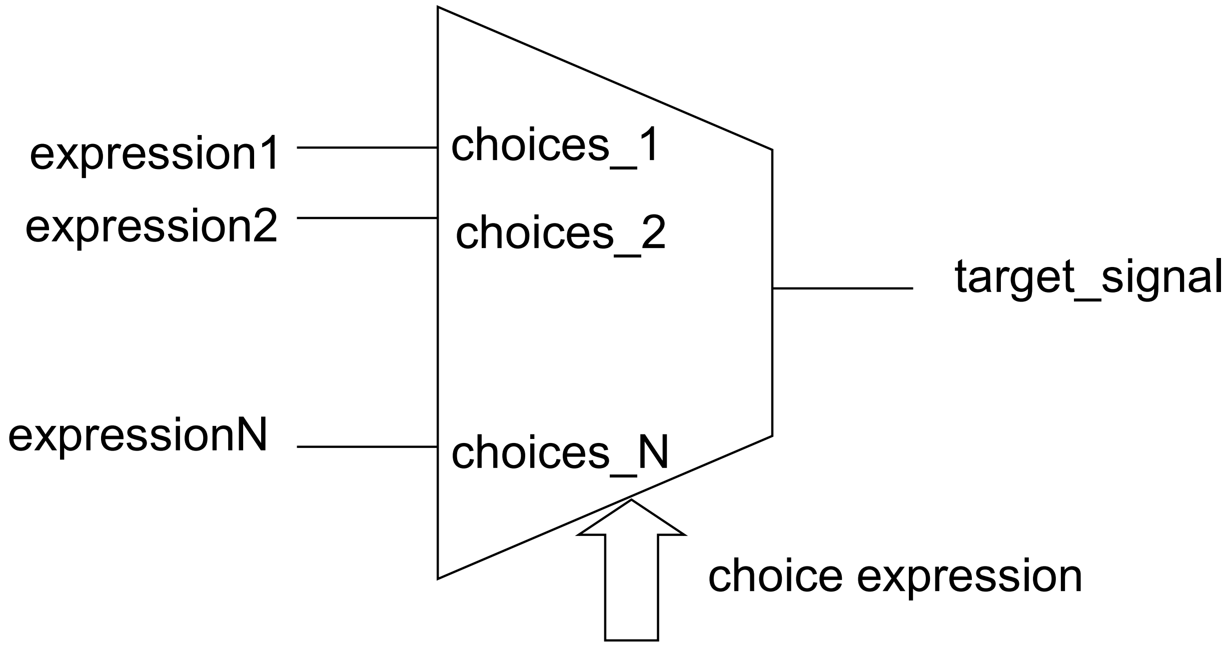
4-to-1 Multiplexer– example (2)

```
ARCHITECTURE mux4to1_dataflow OF mux4to1 IS
BEGIN
      Z <= A when SEL=“00” else
           B when SEL=“01” else
           C when SEL=“10” else
           D;

END mux4to1_dataflow;
```

Selected Signal assignment

With – Select - When

```
[label:]
WITH choice_expression SELECT
   target_signal <= expression1 WHEN choices_1,
                    expression2 WHEN choices_2,
                          . . .
                    expressionN WHEN choices_N;
```



Rules

- ✓ **No two choices can overlap**
- ✓ **All possible values of choice_expression must be covered by the set of choices, unless an OTHERS choice is present.**
- ✓ **Allowed formats of *choices_k:***

WHEN value

WHEN value_1 to value_2

WHEN value_1 | value_2 | .... | value N

Allowed formats of *choice_k - example*

```
SIGNAL a,b,c,d,y: STD_LOGIC;
SIGNAL sel: STD_LOGIC_VECTOR(2 downto 0)
WITH sel SELECT
        y <= a WHEN "000",
             b WHEN "011" to "110",
             c WHEN "001" | "111",
             d WHEN OTHERS;
```

Example: 4-to-1 multiplexer (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux4to1 IS
   PORT (A, B, C, D: IN        std_logic;
              SEL : IN        std_logic_vector (1 downto 0);
               Z:      OUT  std_logic
              );
END mux4to1;
```

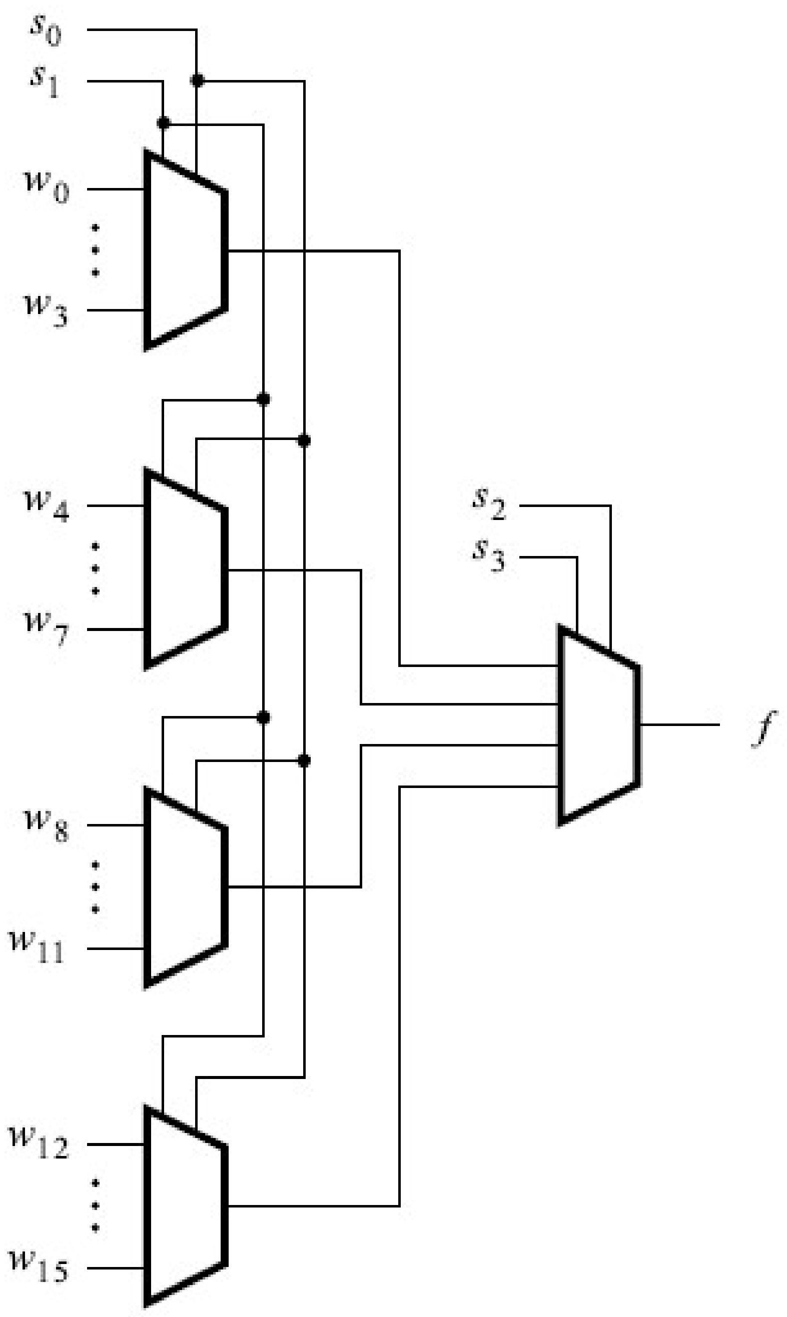
Example: 4-to-1 multiplexer (2)

```
ARCHITECTURE mux4to1_dataflow OF mux4to1 IS
BEGIN
   WITH SEL SELECT
         Z <=     A when “00”,
                  B when “01”,
                  C when “10”,
                  D when OTHERS;

END mux4to1_dataflow;
```

QUIZ

How to build a structural model of 16-to-1 multiplexer by using the described 4-to-1 multiplexer ?

QUIZ



Package Definition: *My_package*

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
PACKAGE My_package IS
COMPONENT mux4to1
PORT ( A, B, C, D : IN          STD LOGIC ;
         SEL : IN         STD LOGIC VECTOR (1 DOWNTO 0) ;
         f  : OUT                   STD LOGIC ) ;
END COMPONENT ;
END My_package ;
```

Example: 16-to-1 multiplexer (1)

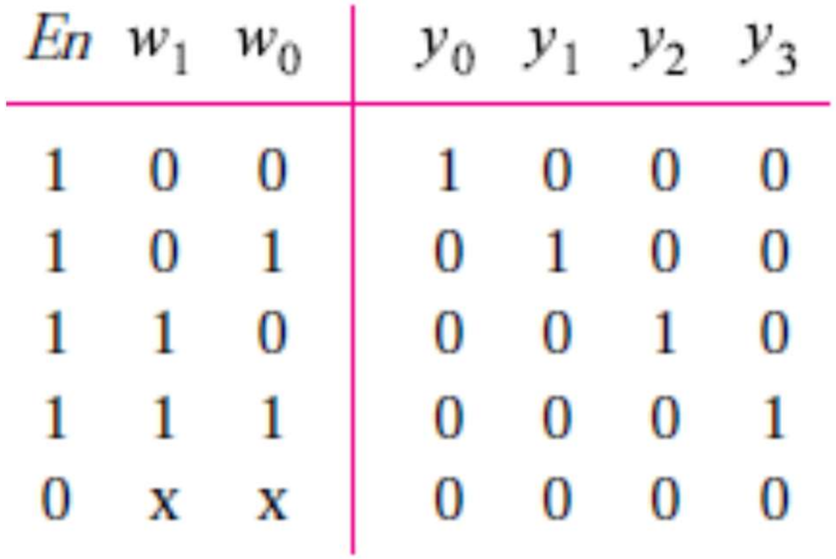
```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
LIBRARY work ;
USE work.My_package.all ;

ENTITY mux16to1 IS
   PORT ( w : IN STD LOGIC VECTOR(0 TO 15) ;
                s: IN STD LOGIC VECTOR(3 DOWNTO 0) ;
                f : OUT STD LOGIC ) ;
END mux16to1 ;
```

Example: 16-to-1 multiplexer (2)

```
ARCHITECTURE Structure OF mux16to1 IS
  SIGNAL m : STD LOGIC VECTOR(0 TO 3) ;
  BEGIN
    Mux1: mux4to1 PORT MAP
            ( w(0), w(1), w(2), w(3), s(1 DOWNTO 0), m(0) ) ;
   Mux2: mux4to1 PORT MAP
            ( w(4), w(5), w(6), w(7), s(1 DOWNTO 0), m(1) ) ;
   Mux3: mux4to1 PORT MAP
            ( w(8), w(9), w(10), w(11), s(1 DOWNTO 0), m(2) ) ;
  Mux4: mux4to1 PORT MAP
            ( w(12), w(13), w(14), w(15), s(1 DOWNTO 0), m(3) ) ;
  Mux5: mux4to1 PORT MAP
            ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;
END Structure ;
```

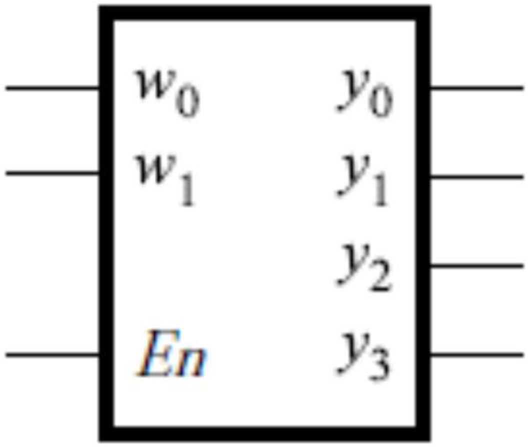
QUIZ

Write VHDL code for a 2-to-4 binary decoder by using selected signal assignment statements?



En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table



(b) Graphical symbol

Example: 2-to-4 Decoder (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec2to4 IS
   PORT (E: IN          std_logic;
          w : IN          std_logic_vector (1 downto 0);
           y:  OUT       std_logic_vector (3 downto 0)
          );
END dec2to4;
```

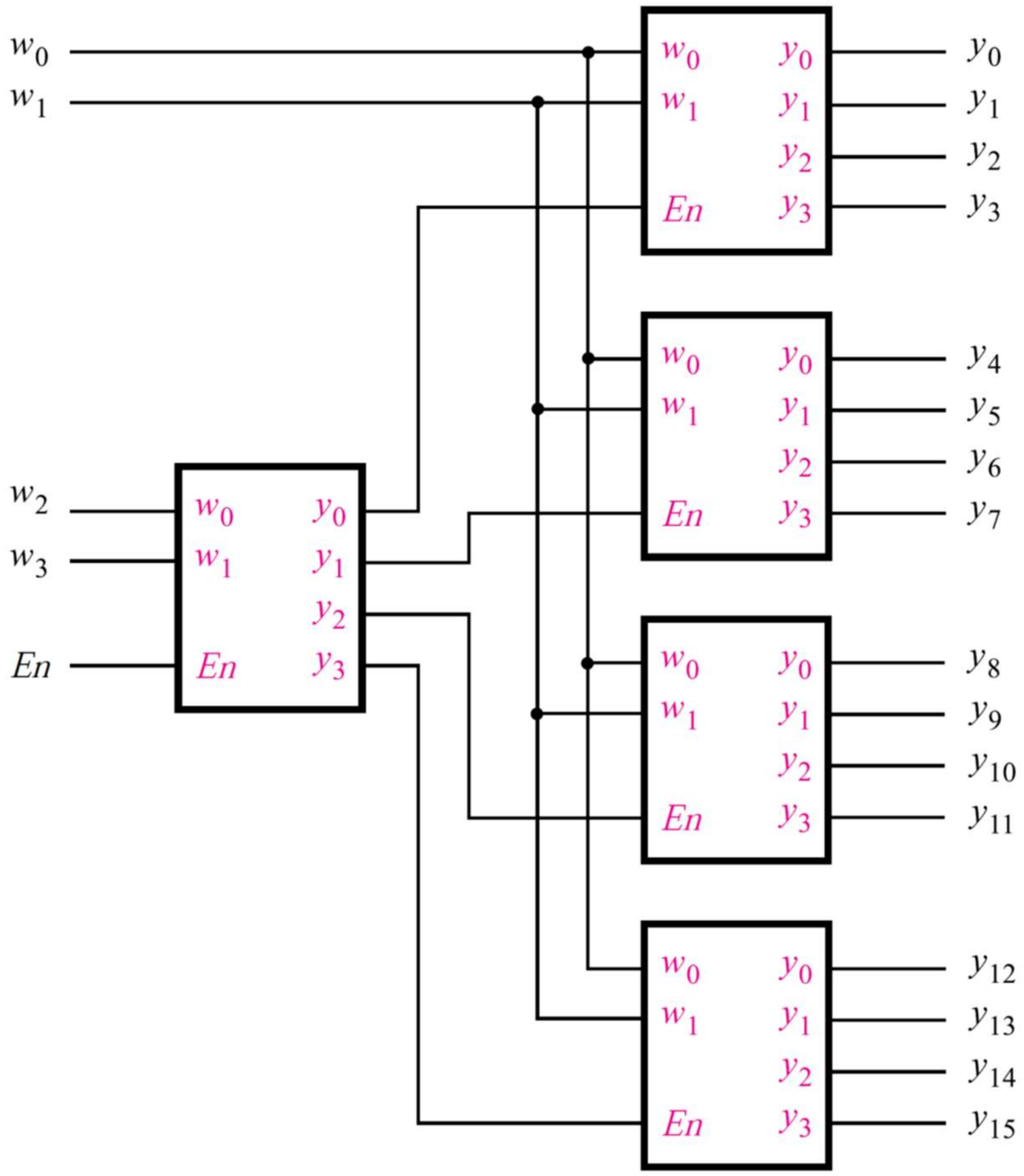
Example: 2-to-4 Decoder (2)

```
ARCHITECTURE dec2to4_dataflow OF dec2to4 IS
 Signal t : std_logic_vector(2 downto 0);
BEGIN
    t <= E & w;
    WITH t SELECT
          y <=      “0001” when “100”,
                    “0010” when “101”,
                    “0100” when “110”,
                    “1000” when “111”,
                    “0000” when  OTHERS;

END dec2to4_dataflow;
```

QUIZ

How to build a structural model of 4-to-16 decoder by using the described 2-to-4 decoder ?

QUIZ



w_0
w_1
w_2
w_3
En
y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12} y_{13} y_{14} y_{15}

Generate Statements: Motivation

```
ARCHITECTURE adder_4bit_structure OF adder_4bit IS
   signal c: std_logic_vector (4 downto 0);
   component FULLADDER
           port (x, y, cin:  in std_logic;
                 s, cout:  out std_logic);
   end component;
BEGIN
  FA0: FULLADDER
           port map (x(0), y(0), cin, sum(0), c(1));
  FA1: FULLADDER
           port map (x(1), y(1), c(1), sum(1), c(2));
  FA2: FULLADDER
           port map (x(2), y(2), c(2), sum(2), c(3));
  FA3: FULLADDER
           port map (x(3), y(3), c(3), sum(3), c(4));
  Cout <= c(4);
END adder_4bit_structure ;
```

Generate Statements: Syntax

For - Generate

```
Generate_Label: FOR i IN 0 TO N GENERATE
    Logic_Expression;
    Component_Label: Componet_name
        PORT MAP (Port1, Port2,…, Portn ) ;
END GENERATE ;
```

Note:

✓The generate statement must have a label

✓The variable i is not explicitly declared in the code; it is automatically defined as a local variable whose scope is limited to the FOR-GENERATE statement

Generate Statements: Example 1

```
ARCHITECTURE adder_4bit_structure OF adder_4bit IS
   signal c: std_logic_vector (4 downto 0);
   component FULLADDER
            port (x, y, cin:  in std_logic;
                  s, cout:  out std_logic);
   end component;
BEGIN
      C(0) <= cin;
      G1: FOR i IN 0 TO 3 GENERATE
        FA: FULLADDER
          port map (x(i), y(i), c(i), sum(i), c(i+1));
      END GENERATE ;

      Cout <= c(4);

END adder_4bit_structure ;
```

Generate Statements: Example 2

Rewrite code for a 16-to-1 multiplexer using a FOR GENERATE statement!

Generate Statements: Example 2

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.My_package.all ;
ENTITY mux16to1 IS
 PORT ( w : IN STD_LOGIC_VECTOR(0 TO 15) ;
            s : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            f : OUT STD_LOGIC ) ;
END mux16to1 ;
ARCHITECTURE Structure OF mux16to1 IS
   SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
   G1: FOR i IN 0 TO 3 GENERATE
   Muxes: mux4to1 PORT MAP (
      w(4*i), w(4*i+1), w(4*i+2), w(4*i+3), s(1 DOWNTO 0), m(i) ) ;
END GENERATE ;
   Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;
END Structure ;
```

Generate Statements: Syntax

IF - Generate

```
Generate_Label: IF expression GENERATE
     Logic_Expression;
     Component_Label: Componet_name
          PORT MAP (Port1, Port2,…, Portn ) ;
END GENERATE ;
```

Note:

✓The generate statement must have a label

Generate Statements: Example 3

Rewrite code for a 4-to-16 decoder using FOR-GENERATE and IF-GENERATE statements!

Generate Statements: Example 3

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.My_package.all ;
ENTITY dec4to16 IS
PORT ( w : IN STD LOGIC VECTOR(3 DOWNTO 0) ;
  En : IN STD LOGIC ;
  y : OUT STD_LOGIC_VECTOR (0 TO 15) ) ;
END dec4to16 ;
ARCHITECTURE Structure OF dec4to16 IS
SIGNAL m : STD LOGIC VECTOR(0 TO 3) ;
BEGIN
   G1: FOR i IN 0 TO 3 GENERATE
      Dec ri: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(i), y(4*i TO 4*i+3) );
     G2: IF i=3 GENERATE
         Dec left: dec2to4 PORT MAP ( w(i DOWNTO i-1), En, m ) ;
     END GENERATE ;
  END GENERATE ;
END Structure ;
```

Sequential Assignment Statements

✓ The ordering of the *sequential assignment statements* may affect the meaning of the code.

✓ The sequential assignment statements be placed inside another type of statement, called a *process* statement

✓ VHDL provides two types of sequential assignment statements:

- ***if-then-else*** statement
- ***case*** statement

Process Statement

```
[process_label:] PROCESS [ (sensitivity_list) ] [IS]
            [[VARIABLE or CONSANT declarations]] -- No Signal
BEGIN
    --list of sequential statements such as:
            --simple signal assignments
            --variable assignments
            -- case statements
            -- if-then-else statements
            -- For statements
            -- WAIT statments
END PROCESS [process_label];
```

IF-THEN-ELSE Statement

Syntax:

```
IF condition1 THEN
          --sequential statements
ELSIF condition2 THEN
          --sequential statements
ELSE
          --sequential statements
END IF;
```

Note:

The **if-then-else** statement must **ALWAYS** be inside a process construct!

IF-THEN-ELSE Statement: Example

```
-- 2-to-1 Multiplexer
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY mux2to1 IS
PORT ( w0, w1, s : IN STD LOGIC ;
              f : OUT STD LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
PROCESS ( w0, w1, s )
  BEGIN
    IF s = ’0’ THEN
              f <= w0 ;
    ELSE
              f <= w1 ;
    END IF ;
  END PROCESS ;
END Behavior ;
```

IF-THEN-ELSE Statement: QUIZ

```
-- 2-to-1 Multiplexer
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY mux2to1 IS
PORT ( w0, w1, s : IN STD LOGIC ;
              f : OUT STD LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
PROCESS ( w0, w1, s )
  BEGIN
              f <= w0 ;
     IF s = '1' THEN
              f <= w1 ;
    END IF ;
  END PROCESS ;
END Behavior ;
```

QUIZ

nand_gate.vhd: various implementations of architecture body

```
ARCHITECTURE model OF nand_gate IS
    -- signal declaration (of internal signals X)
    signal X: std_logic;
BEGIN
    x <= a AND b;
    z <= NOT X;
END model;
```

Vs.

```
ARCHITECTURE model OF nand_gate IS
    -- signal declaration (of internal signals X)
    signal X: std_logic;
BEGIN
    z <= NOT x;
    x <= a AND b;
    END model;
```

IF-THEN-ELSE Statement: QUIZ

```
-- 2-to-1 Multiplexer
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY mux2to1 IS
PORT ( w0, w1, s : IN STD LOGIC ;
              f : OUT STD LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
PROCESS ( w0, w1, s )
  BEGIN
      IF s = '1' THEN f <= w1 ;    END IF ;
              f <= w0 ;
  END PROCESS ;
END Behavior ;
```

QUIZ

```
ARCHITECTURE model OF nand_gate IS
    -- signal declaration (of internal signals X)
    signal X: std_logic;
BEGIN
    z <= '0';
    z <= NOT x;
    x <= a AND b;
    END model;
```

QUIZ

How to write VHDL code of 4-to-1 multiplexer by using the if-then-else and process statements ?

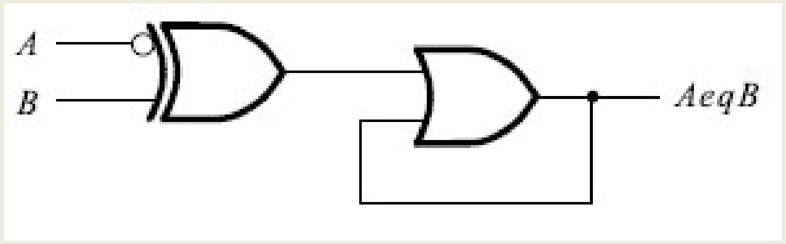
IF-THEN-ELSE Statement: QUIZ

```
-- one-bit equality comparator.
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY compare1 IS
PORT ( A, B : IN STD LOGIC ;
          AeqB : OUT STD LOGIC ) ;
END compare1 ;
ARCHITECTURE Behavior OF compare1 IS
BEGIN
PROCESS ( A, B )
BEGIN
  AeqB <= ’0’ ;
  IF A = B THEN
          AeqB <=’1’ ;
  END IF ;
END PROCESS ;
END Behavior ;
```

IF-THEN-ELSE Statement: QUIZ

```
-- one-bit equality comparator.
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY compare1 IS
PORT ( A, B : IN STD LOGIC ;
          AeqB : OUT STD LOGIC ) ;
END compare1 ;
ARCHITECTURE Behavior OF compare1 IS
BEGIN
PROCESS ( A, B )
BEGIN
  IF A = B THEN
          AeqB <='1' ;
  END IF ;
END PROCESS ;
END Behavior ;
```

What is the circuit synthesized from the code?



implied memory

CASE Statement

Syntax:

```
CASE expression IS
          WHEN choice_1 =>
                    --sequential statements
           WHEN choice_2 =>
                    --sequential statements
                    -- branches are allowed
          [WHEN OTHERS => --sequential statements ]
END CASE;
```

Note:

The **Case** statement must ALWAYS be inside a process construct!

CASE Statement

Rules:

✓ *expression* must evaluate to an integer, an enumerated type of a one-dimensional array
✓ no two choices can overlap (i.e. each choice can be covered only once)
✓ if the “when others" choice is not present, all possible values of the expression must be covered by the set of choices.
✓ **Allowed formats of *choices_k:***

WHEN value

WHEN value_1 to value_2

WHEN value_1 | value_2 | .... | value N

CASE Statement: Example 1

```
-- 2-to-1 multiplexer
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY mux2to1 IS
PORT ( w0, w1, s : IN STD LOGIC ;
              f : OUT STD LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
PROCESS ( w0, w1, s )
 BEGIN


…




END PROCESS ;
END Behavior ;
```

CASE Statement: Example 1

```
-- 2-to-1 multiplexer
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY mux2to1 IS
PORT ( w0, w1, s : IN STD LOGIC ;
             f : OUT STD LOGIC ) ;
END mux2to1 ;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
PROCESS ( w0, w1, s )
 BEGIN
   CASE s IS
             WHEN ’0’ =>
                          f <= w0 ;
             WHEN OTHERS =>
                          f <= w1 ;
  END CASE ;
END PROCESS ;
END Behavior ;
```

CASE Statement: Example 2

```
-- 2-to-4 dcoder
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY dec2to4 IS
PORT ( w : IN STD LOGIC VECTOR(1 DOWNTO 0) ;
            En : IN STD LOGIC ;
            y : OUT STD LOGIC VECTOR(0 TO 3) ) ;
END dec2to4 ;
```

CASE Statement: Example 2

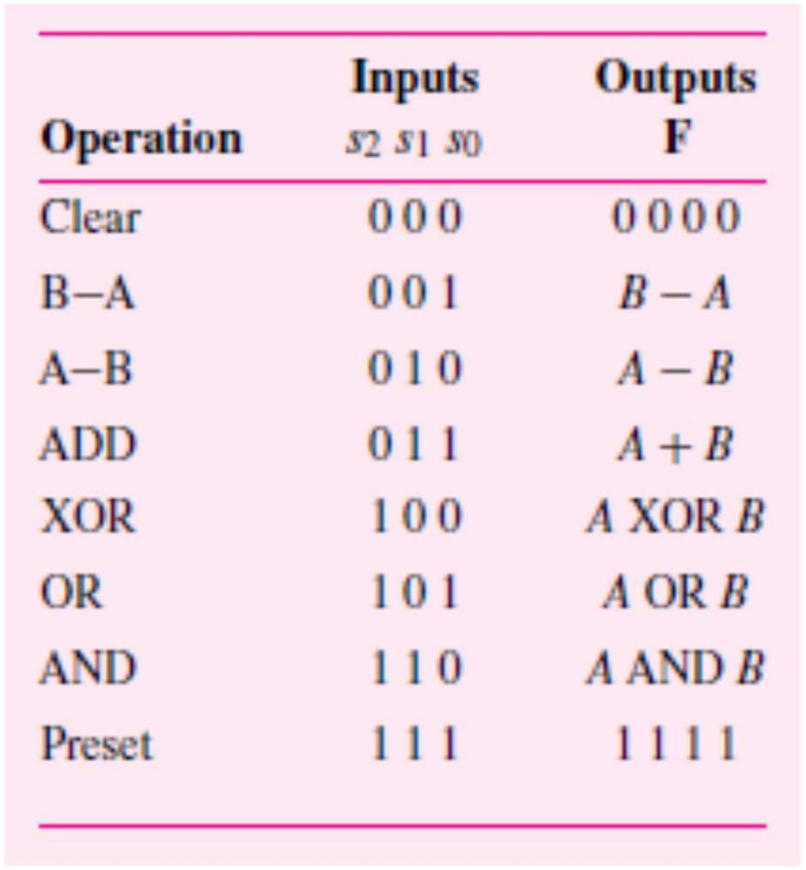
```
ARCHITECTURE Behavior OF dec2to4 IS
BEGIN

END Behavior ;
```

CASE Statement: Example 2

```
ARCHITECTURE Behavior OF dec2to4 IS
BEGIN
   PROCESS ( w, En )
   BEGIN
    IF En = ’1’ THEN
      CASE w IS
              WHEN ”00” =>
                             y <= ”1000” ;
              WHEN ”01” =>
                             y <= ”0100” ;
              WHEN ”10” =>
                             y <= ”0010” ;
              WHEN OTHERS =>
                             y <= ”0001” ;
      END CASE ;
    ELSE
              y <=”0000” ;
    END IF ;
  END PROCESS ;
END Behavior ;
```

QUIZ

How to write VHDL code of a ALU unit by using the CASE and process statements ?



Operation	Inputs $s2\ s1\ s0$	Outputs F
Clear	0 0 0	0 0 0 0
B–A	0 0 1	$B - A$
A–B	0 1 0	$A - B$
ADD	0 1 1	$A + B$
XOR	1 0 0	A XOR B
OR	1 0 1	A OR B
AND	1 1 0	A AND B
Preset	1 1 1	1 1 1 1