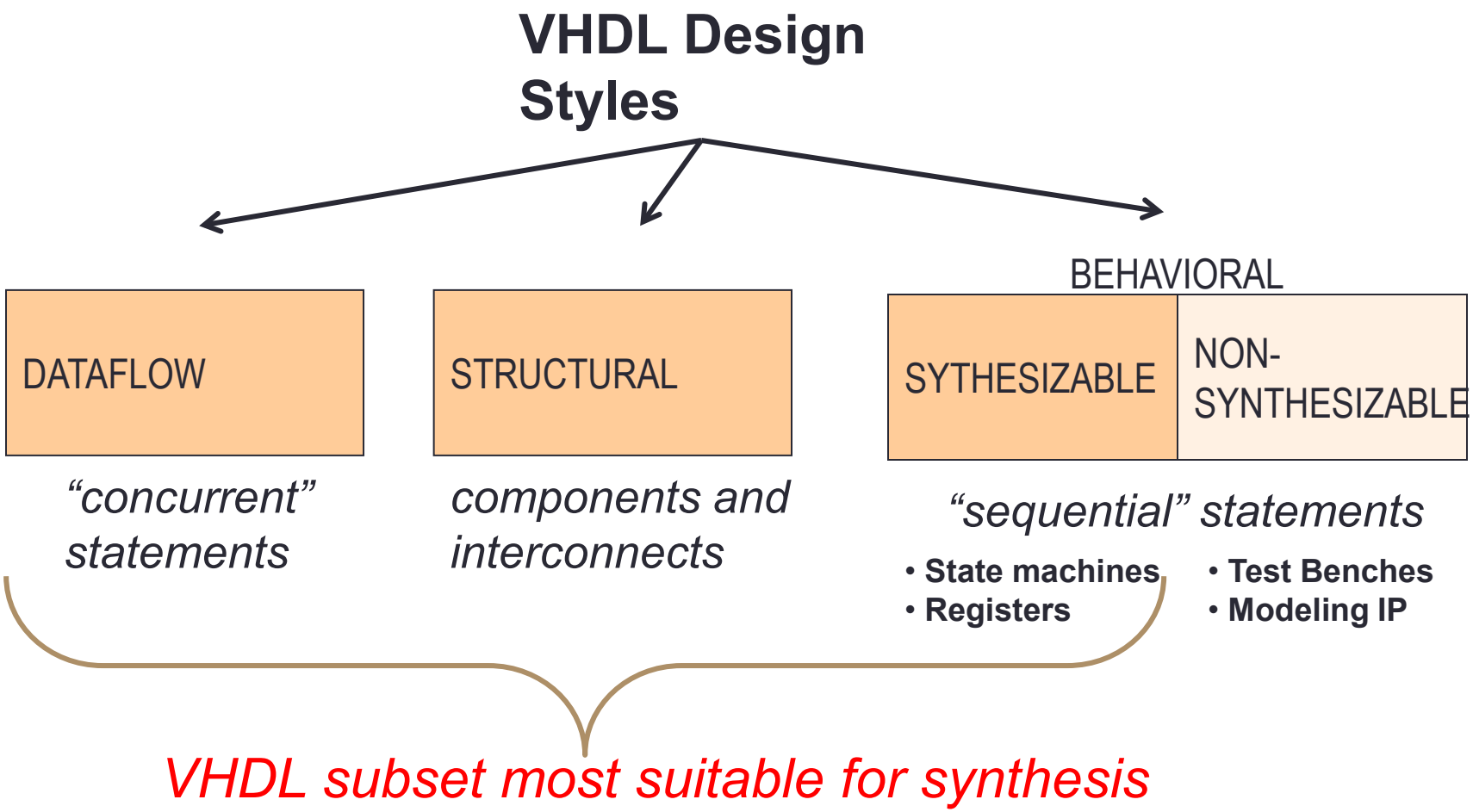
ECE 1195: Advanced Digital Design

VHDL – Part 2

Dr. Amr Mahmoud

VHDL DESIGN STYLES

VHDL Design Styles (Architecture)



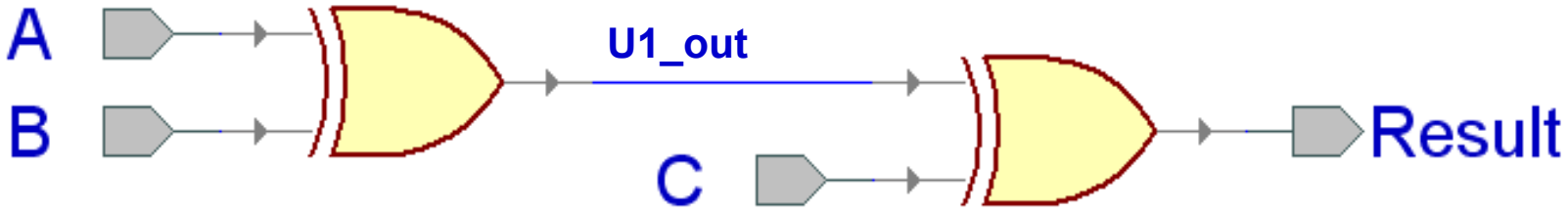
XOR3 Example

Entity XOR3 (same for all architectures)

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

ENTITY xor3 IS
    PORT(
        A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        C : IN STD_LOGIC;
        Result : OUT STD_LOGIC
         );
END xor3;
```

Dataflow Architecture

```
ARCHITECTURE dataflow OF xor3 IS
SIGNAL U1_out: STD_LOGIC;
BEGIN
        Result <= U1_out XOR C;
        U1_out <= A XOR B;
END dataflow;
```



Dataflow Description

- Describes how data moves through the system and the various processing steps.
  - uses series of concurrent statements to realize logic.
  - used to implement simple combinational logic
  - also called “concurrent” code
- Concurrent statements are evaluated at the same time; thus, the order of these statements doesn’t matter
  - This is not true for sequential/behavioral statements

This order…

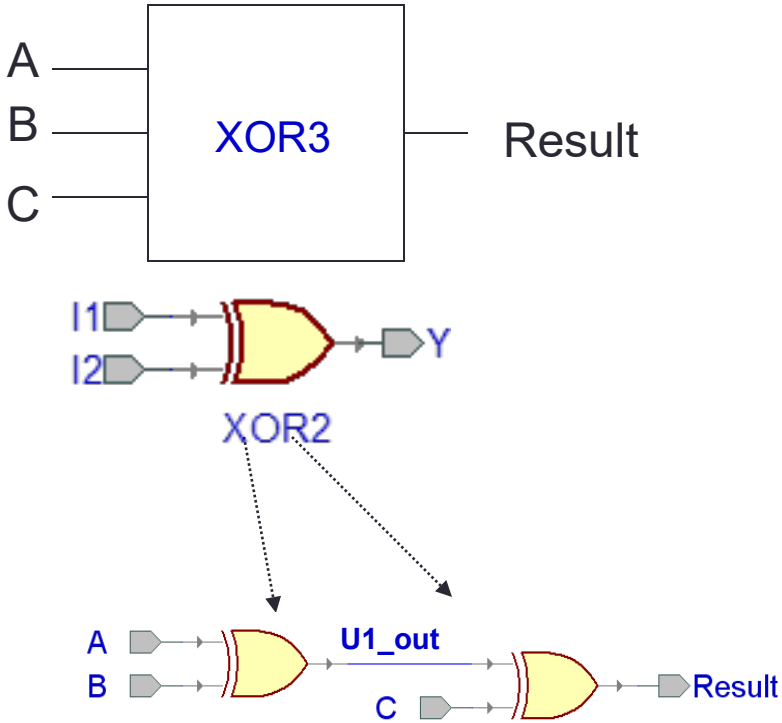
```
U1_out <= A XOR B;
Result <= U1_out XOR C;
```

Is the same as this order…

```
Result <= U1_out XOR C;
U1_out <= A XOR B;
```

Structural Architecture (XOR3 gate)

```
ARCHITECTURE structural OF xor3 IS
SIGNAL U1_OUT: STD_LOGIC;

COMPONENT xor2 IS
    PORT(
         I1 : IN STD_LOGIC;
         I2 : IN STD_LOGIC;
         Y : OUT STD_LOGIC
        );
END COMPONENT;

BEGIN
   U1: xor2 PORT MAP ( I2 => B,
                       I1 => A,
                       Y  => U1_OUT);
   U2: xor2 PORT MAP ( I1 => U1_OUT,
                       I2 => C,
                       Y  => Result);
END structural;
```



Component and Instantiation

- Named association connectivity (recommended)

```
COMPONENT xor2 IS
    PORT(
       I1 : IN STD_LOGIC;
       I2 : IN STD_LOGIC;
       Y : OUT STD_LOGIC
       );
END COMPONENT;
BEGIN
U1: xor2 PORT MAP (    I1 => A,
                       I2 => B,
                       Y  => U1_OUT);
...
```

COMPONENT PORT NAME

LOCAL WIRE

Component and Instantiation

- Positional association connectivity

(not recommended)

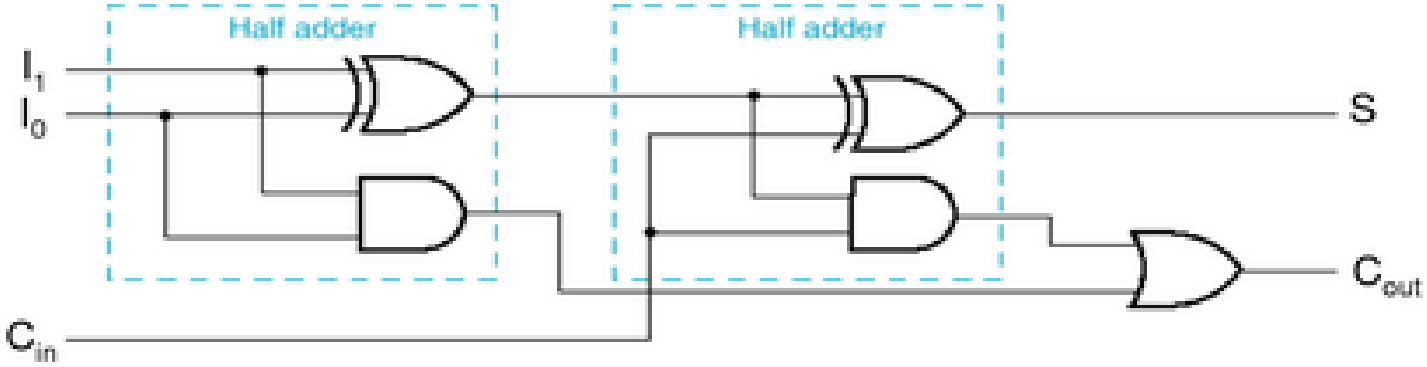
```
COMPONENT xor2 IS
     PORT(
       I1 : IN STD_LOGIC;
       I2 : IN STD_LOGIC;
       Y : OUT STD_LOGIC
         );
END COMPONENT;
BEGIN
U1: xor2 PORT MAP (A, B, U1_OUT);
...
```

Structural Description

- Structural design is the simplest to understand. This style is the closest to schematic diagram and utilizes simple building blocks to compose logic functions.
- Components are interconnected in a hierarchical manner.
- Structural descriptions may connect simple gates or complex, abstract components.
- Structural style is useful when expressing a design that is naturally composed of sub-blocks.

Task #2

- Write the VHDL code that describes **FULL-Adder** using 2 half-adders.



Task #2(solution)

- Write the VHDL code that describes **FULL-Adder** using 2 half-adders.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
----------------------------
entity half_adder is
port( A: in STD_LOGIC;
      B: in STD_LOGIC;
      Sum,Cout: out STD_LOGIC
     );
end half_adder;


architecture One of half_adder is
begin

    Sum <= A XOR B;
    Cout <= A AND B;

end One;
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
----------------------------
entity full_adder is
port( A: in STD_LOGIC;
      B: in STD_LOGIC;
      Cin: in STD_LOGIC;
      Sum,Cout: out STD_LOGIC
     );
end full_adder;


architecture One of full_adder is
---------------------------------
Component half_adder IS
PORT( A: in STD_LOGIC;
      B: in STD_LOGIC;
      Sum,Cout: out STD_LOGIC
    );
end Component;
Signal Temp1,Temp2,Temp3: STD_LOGIC;
----------------------------------
begin

U1: half_adder PORT MAP( A => A,
                         B => B,
                         Sum =>Temp1,
                         Cout => Temp2);
U2: half_adder PORT MAP( A => Temp1,
                         B => Cin,
                         Sum =>Sum,
                         Cout => Temp3);
Cout <= Temp2 OR Temp3;

end One;
```

Behavioral Architecture (XOR3 gate)

```
ARCHITECTURE behavioral OF xor3 IS
BEGIN

   PROCESS (A,B,C)
   BEGIN
       IF ((A XOR B XOR C) = '1') THEN
              Result <= '1';
       ELSE
              Result <= '0';
       END IF;
   END PROCESS;
END behavioral;
```

Behavioral Description

- It accurately models what happens on the inputs and outputs of the black box (no matter what is inside and how it works).
- This style uses PROCESS statements in VHDL.
- Statements are executed in sequence in a process statement → order of code matters!

DESCRIBING COMBINATIONAL LOGIC USING DATAFLOW VHDL

Dataflow VHDL

- **concurrent signal assignment (⇐)**
- conditional concurrent signal assignment (when-else)
- selected concurrent signal assignment (with-select-when)
- generate scheme for equations (for-generate)

Dataflow VHDL: 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

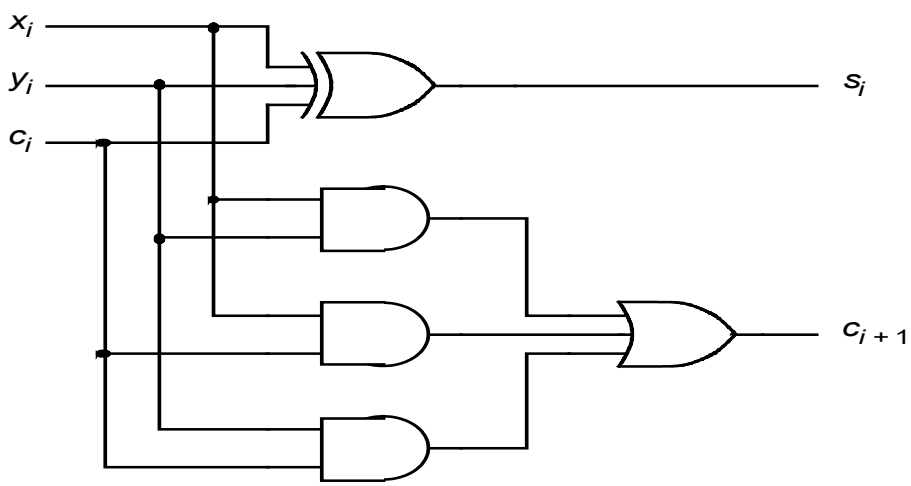
c_i \ x_iy_i	00	01	11	10
0		1		1
1	1		1	

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

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

Full Adder Example

TIP: for architecture name for dataflow circuits use either "dataflow" or "entityname_dataflow"

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

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

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

Logic Operators

- Logic operators

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

- Logic operators precedence

Highest

↓

Lowest

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

No Implied Precedence

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

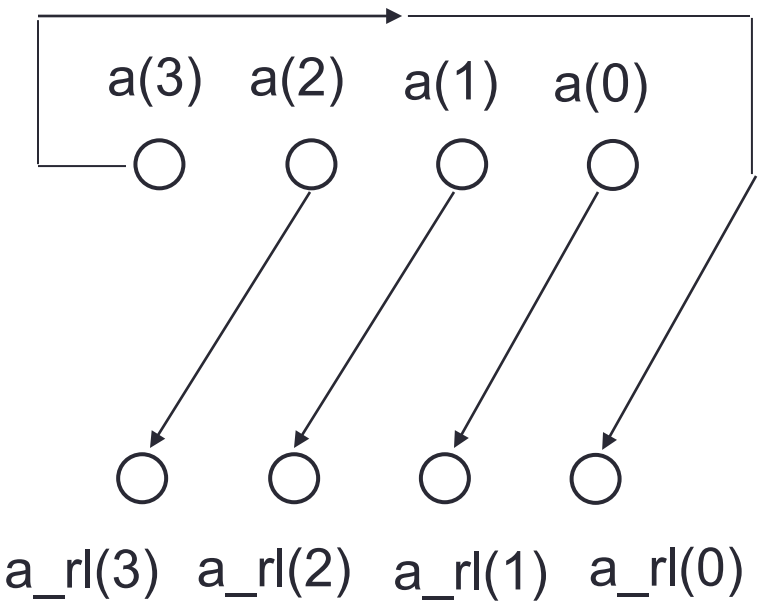
Correct

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

Concatenation

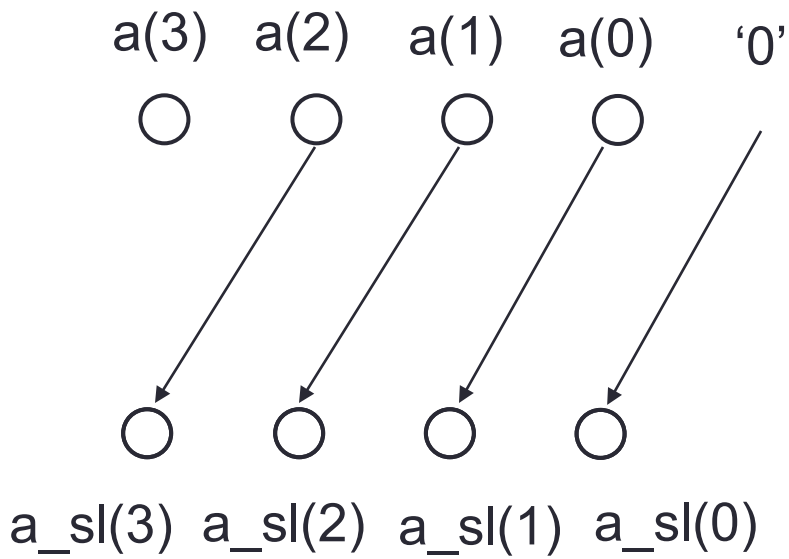
```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

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"
```

Rotations in VHDL




```
a_rl <= a(2 downto 0) & a(3);
```

Shifts in VHDL (zero-stuff)



a_sl <= a(2 downto 0) & '0';

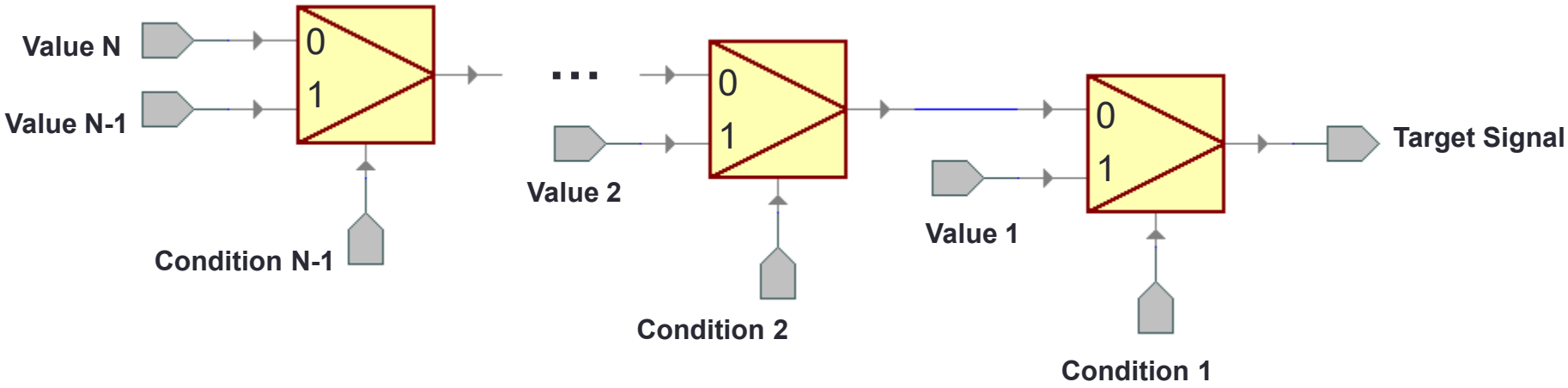
Dataflow VHDL

- concurrent signal assignment ($\Leftarrow$)
- **conditional concurrent signal assignment (when-else)**
- selected concurrent signal assignment (with-select-when)
- generate scheme for equations (for-generate)

Conditional concurrent signal assignment

When - Else

```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                   . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```



Operators

- Relational operators

```
=      /=      <      <=      >      >=
```

- Logic and relational operators precedence

Highest
↓
Lowest

```
                    not
=      /=      <      <=      >      >=
and    or    nand    nor    xor    xnor
```

Priority of Logic and Relational Operators

compare a = bc

Incorrect

… when a = b **and** c else …

equivalent to

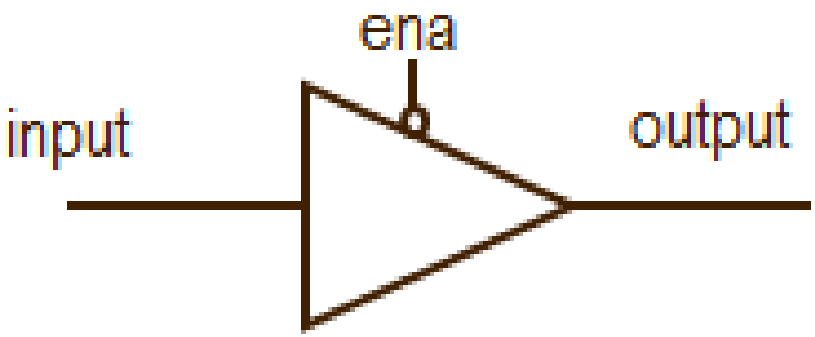
… when (a = b) **and** c else …

Correct

… when a = (b **and** c) else …

Task #3

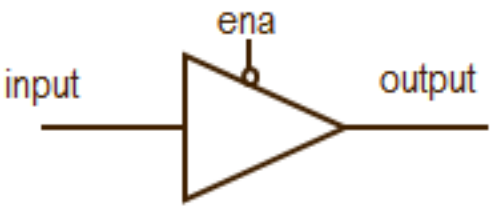
- Write the VHDL code that describes **Tri-state Buffer**.



Task #3(solution)

- Write the VHDL code that describes **Tri-state Buffer**.

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

ENTITY tri_state IS
  PORT ( ena:    IN STD_LOGIC;
      input:  IN STD_LOGIC_VECTOR(7 downto 0);
          output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
              );
END tri_state;
ARCHITECTURE tri_state_dataflow OF tri_state IS
BEGIN
     output <= input WHEN (ena = '0') ELSE
                    (OTHERS => 'Z'); -- Equivalent to "zzzzzzzz"
END tri_state_dataflow;
```



OTHERS means all bits not directly specified, in this case all the bits.

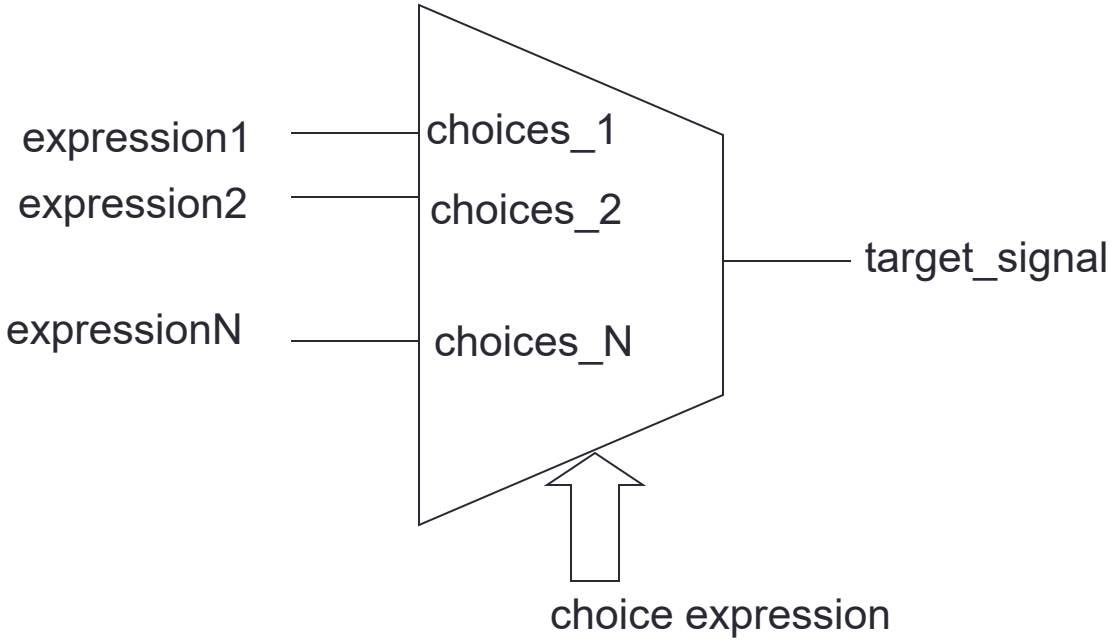
Dataflow VHDL

- concurrent signal assignment ($\Leftarrow$)
- conditional concurrent signal assignment (when-else)
- **selected concurrent signal assignment (with-select-when)**
- generate scheme for equations (for-generate)

Selected concurrent signal assignment

With –Select-When

```
with choice_expression select
    target_signal <= expression1 when choices_1,
                     expression2 when choices_2,
                         . . .
                     expressionN when choices_N;
```



Allowed formats of *choices_k*

```
WHEN value

WHEN value_1 to value_2

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

this means boolean “or”

Allowed formats of *choice_k - example*

```
WITH sel SELECT
     y <= a WHEN "000",
          b WHEN "011" to "110",
          c WHEN "001" | "111",
          d WHEN “010”,
          (others => ‘0’) WHEN others;
```

Task #4

- Write the VHDL code that describe **4-bit Shifter**:

Selection S1 S0	Operation
0 0	Shift Left
0 1	Shift Right
1 0	Rotate Left
1 1	Rotate Right

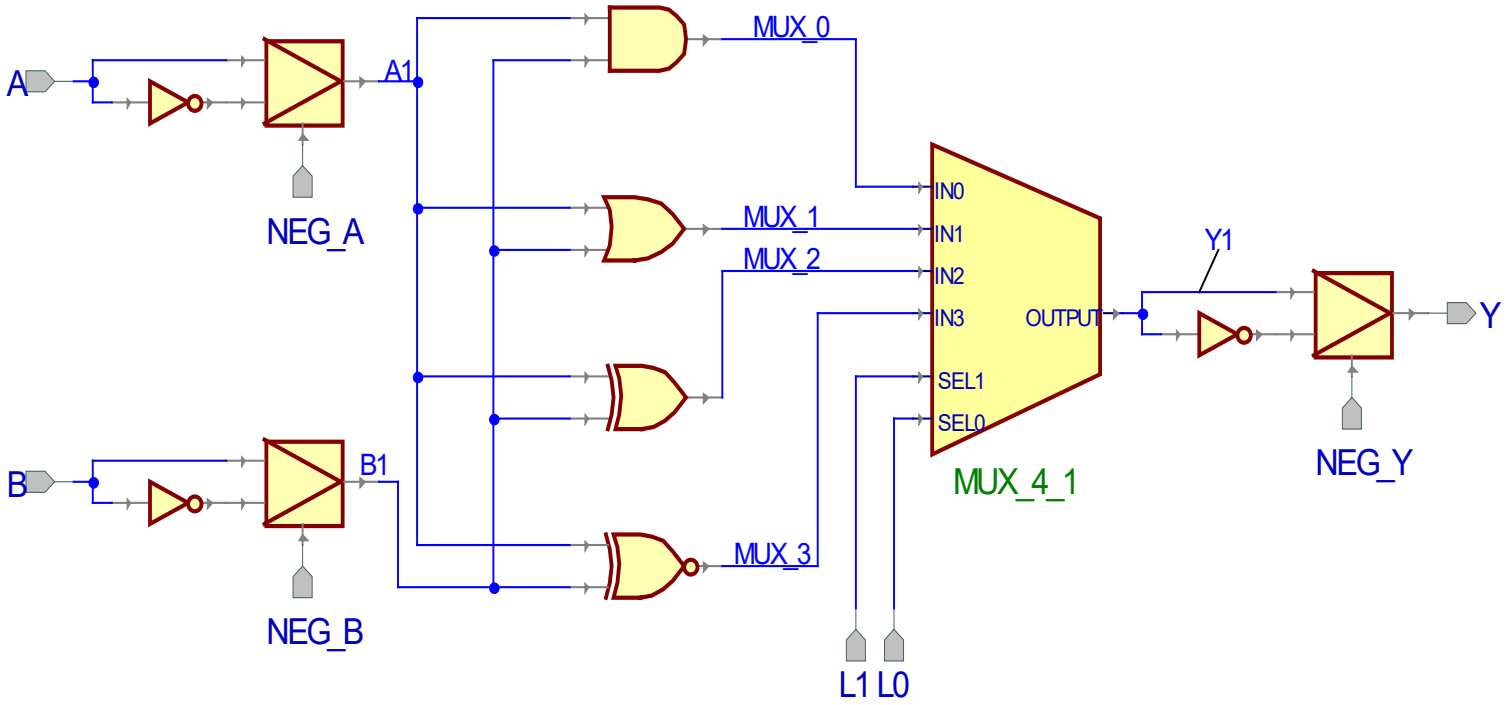
Task #4(solution)

- Write the VHDL code that describe **4-bit Shifter**:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY SHIFTER IS
      PORT(
           A: IN STD_LOGIC_VECTOR(3 downto 0);
           Sel: IN STD_LOGIC_VECTOR(1 downto 0);
           Z: OUT STD_LOGIC_VECTOR(3 downto 0)
          );
END SHIFTER;
----Truth Table
--Sel   -> Operation
--00    ->   SHL
--01    ->   SHR
--10    ->   ROL
--11    ->   ROR

ARCHITECTURE Combinational OF SHIFTER IS
SIGNAL X1,X2,X3,X4: STD_LOGIC_VECTOR(3 downto 0);
BEGIN
    X1<=A(2 downto 0) & '0';
    X2<='0' & A(3 downto 1);
    X3<=A(2 downto 0) & A(3);
    X4<=A(0) & A(3 downto 1);

    WITH Sel SELECT
           Z <= X1 WHEN "00",
                X2 WHEN "01",
                X3 WHEN "10",
                X4 WHEN others;

END Combinational;
```

(LU Example)

- Write the VHDL code that describes **LU**:



MLU: Entity Declaration

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

ENTITY mlu IS
   PORT(
     NEG_A : IN STD_LOGIC;
     NEG_B : IN STD_LOGIC;
     NEG_Y : IN STD_LOGIC;
     A :            IN STD_LOGIC;
     B :            IN STD_LOGIC;
     L1 :          IN STD_LOGIC;
     L0 :          IN STD_LOGIC;
     Y :           OUT STD_LOGIC
       );
END mlu;
```

MLU: Architecture Declarative Section

```
ARCHITECTURE mlu_dataflow OF mlu IS

SIGNAL  A1 :  STD_LOGIC;
SIGNAL  B1 :  STD_LOGIC;
SIGNAL  Y1 : STD_LOGIC;
SIGNAL  MUX_0 : STD_LOGIC;
SIGNAL  MUX_1 : STD_LOGIC;
SIGNAL  MUX_2 : STD_LOGIC;
SIGNAL  MUX_3 : STD_LOGIC;
SIGNAL  L: STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL Abar : STD_LOGIC;
```

MLU - Architecture Body

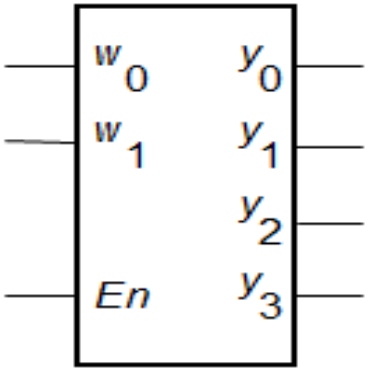
```
BEGIN
  A1<= Abar  WHEN (NEG_A='1') ELSE
    A;
  B1<= NOT B  WHEN (NEG_B='1') ELSE
    B;
  Y <= NOT Y1 WHEN (NEG_Y='1') ELSE
    Y1;
  Abar  <= not A;
  MUX_0 <= A1  AND  B1;
  MUX_1 <= A1   OR  B1;
  MUX_2 <= A1  XOR  B1;
  MUX_3 <= A1  XNOR B1;

      L <= L1 & L0;

  with (L) select
      Y1 <=  MUX_0  WHEN "00",
             MUX_1  WHEN "01",
         MUX_2  WHEN  "10",
            MUX_3  WHEN OTHERS;

END mlu_dataflow;
```

Do it yourself

- Write the VHDL code that describes **2-to-4 Decoder**:

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

Data-flow VHDL

- concurrent signal assignment ($\Leftarrow$)
- conditional concurrent signal assignment (when-else)
- selected concurrent signal assignment (with-select-when)
- **generate scheme for equations (for-generate)**

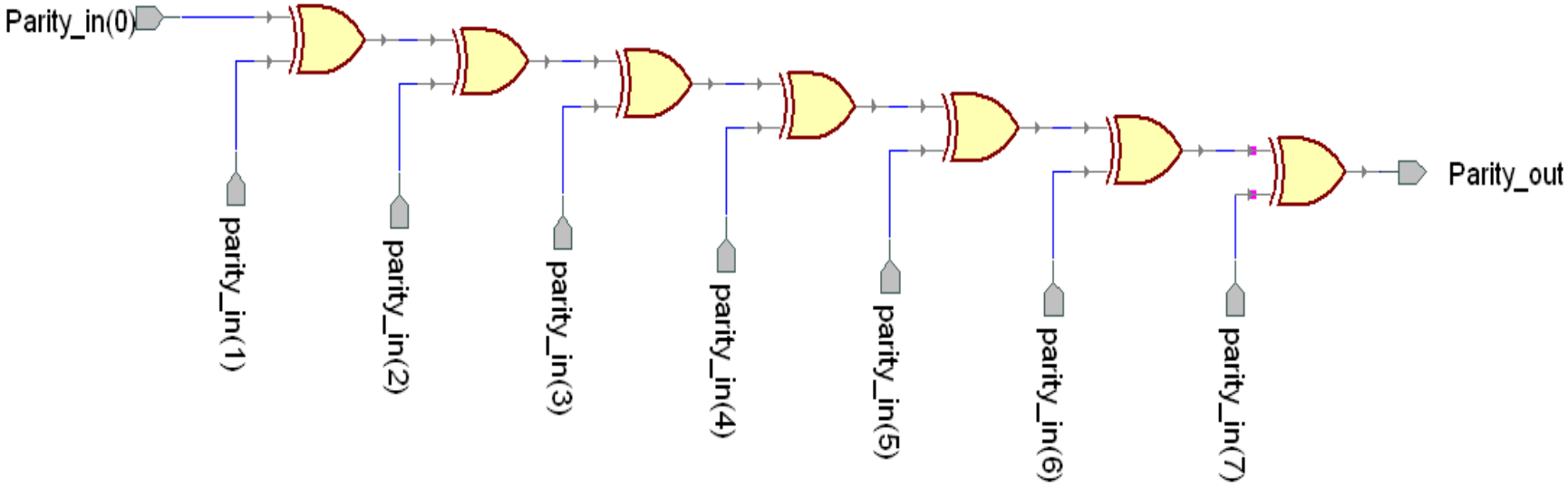
For Generate Statement

For - Generate

```
label: FOR identifier IN range GENERATE
          BEGIN
           {Concurrent Statements}
          END GENERATE label;
```

Generate Statement must has a label

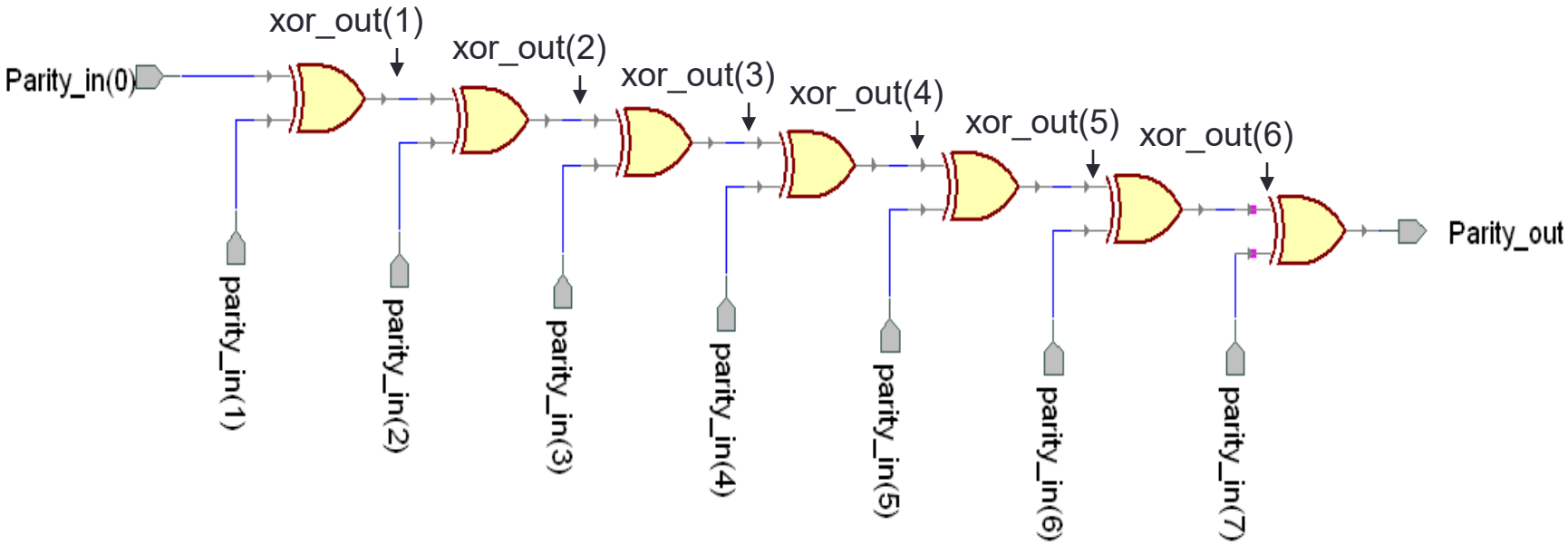
PARITY: Block Diagram



PARITY: Entity Declaration

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

ENTITY parity IS
    PORT(
        parity_in    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        parity_out  : OUT STD_LOGIC
        );
END parity;
```

PARITY: Block Diagram



PARITY: Architecture

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: std_logic_vector (6 downto 1);

BEGIN

    xor_out(1) <= parity_in(0) XOR parity_in(1);
    xor_out(2) <= xor_out(1) XOR parity_in(2);
    xor_out(3) <= xor_out(2) XOR parity_in(3);
    xor_out(4) <= xor_out(3) XOR parity_in(4);
    xor_out(5) <= xor_out(4) XOR parity_in(5);
    xor_out(6) <= xor_out(5) XOR parity_in(6);
    parity_out <= xor_out(6) XOR parity_in(7);

END parity_dataflow;
```

PARITY: Architecture with for generate

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

ENTITY parity IS
   PORT(
     parity_in     : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
     parity_out   : OUT STD_LOGIC
       );
END parity;

ARCHITECTURE parity_overflow OF parity IS
----
Signal Xor_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    Xor_out(0) <= parity_in(0);
    L1:FOR i IN 1 TO 7 GENERATE
        Xor_out(i) <= Xor_out(i-1) XOR parity_in(i);
    END GENERATE;
    parity_out <= Xor_out(7);
END parity_overflow;
```

COMBINATIONAL LOGIC SYNTHESIS

Simple Rules

- For combinational logic,
- Use
  - concurrent signal assignment ($\Leftarrow$)

Simple Rules

- For circuits composed of:
  - simple logic operations (logic gates)
  - simple arithmetic operations (addition, subtraction, multiplication)
  - shifts/rotations by a constant
- Use
  - concurrent signal assignment ($\Leftarrow$)

Simple Rules

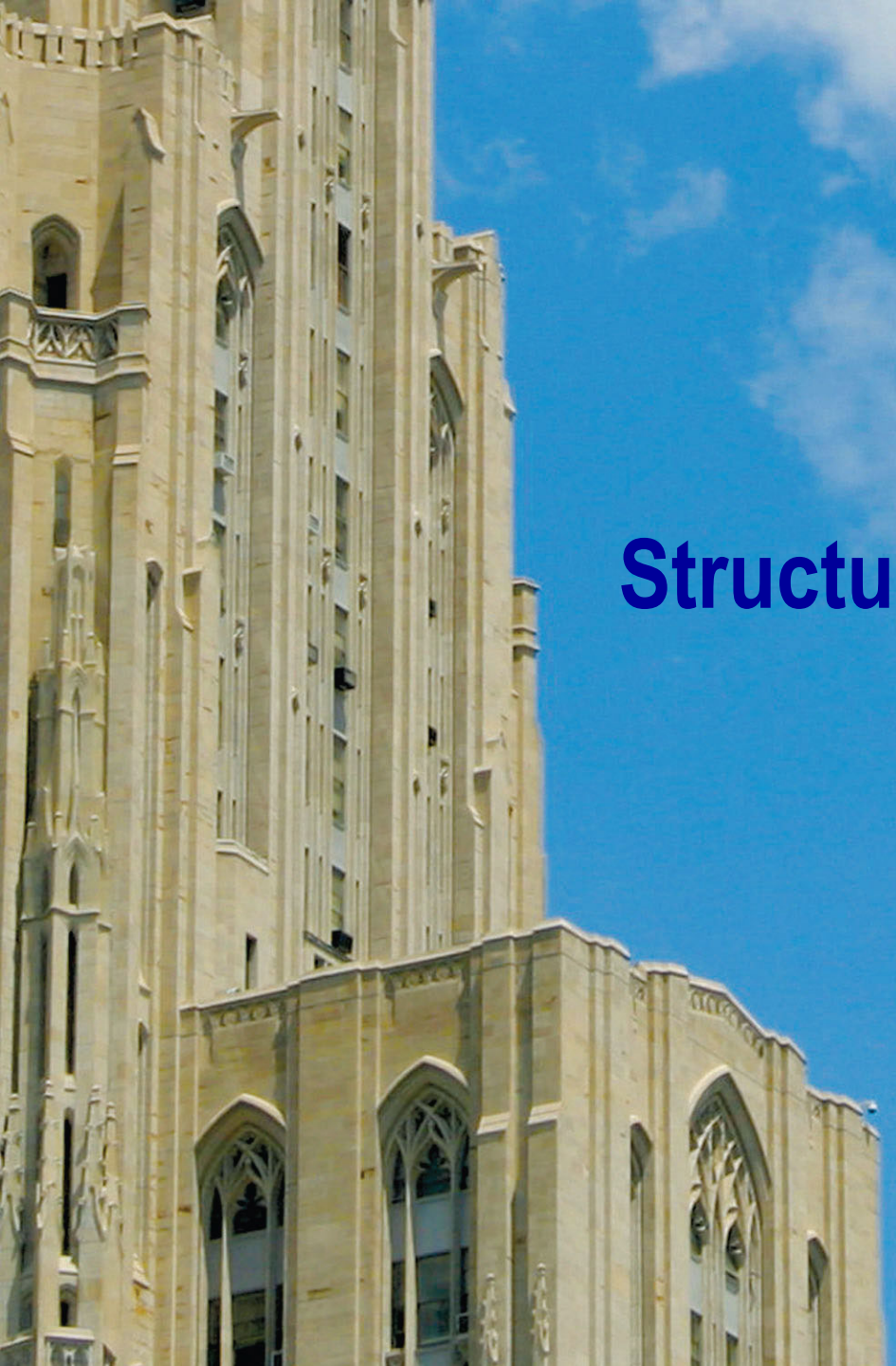
- For circuits composed of
  - Multiplexers
  - Demultiplexers
  - Decoders

- Use
  - selected concurrent signal assignment (with-select-when)

Simple Rules

- For circuits composed of
  - priority encoders
  - tri-state buffers
- Use:
  - conditional concurrent signal assignment (when-else)

Structural VHDL Coding



VHDL Design Styles (Architecture)

VHDL Design Styles

DATAFLOW

"concurrent" statements

- **Gates**
- **Simple Comb. Logic**

STRUCTURAL

components and interconnects

BEHAVIORAL

SYNTHESIZABLE

"sequential" statements

- **Sequential Logic (registers, counters, etc.)**
- **State machines**
- **Complex Comb. Logic**

NON-SYNTHESIZABLE

- **Test Benches**
- **Modeling IP**

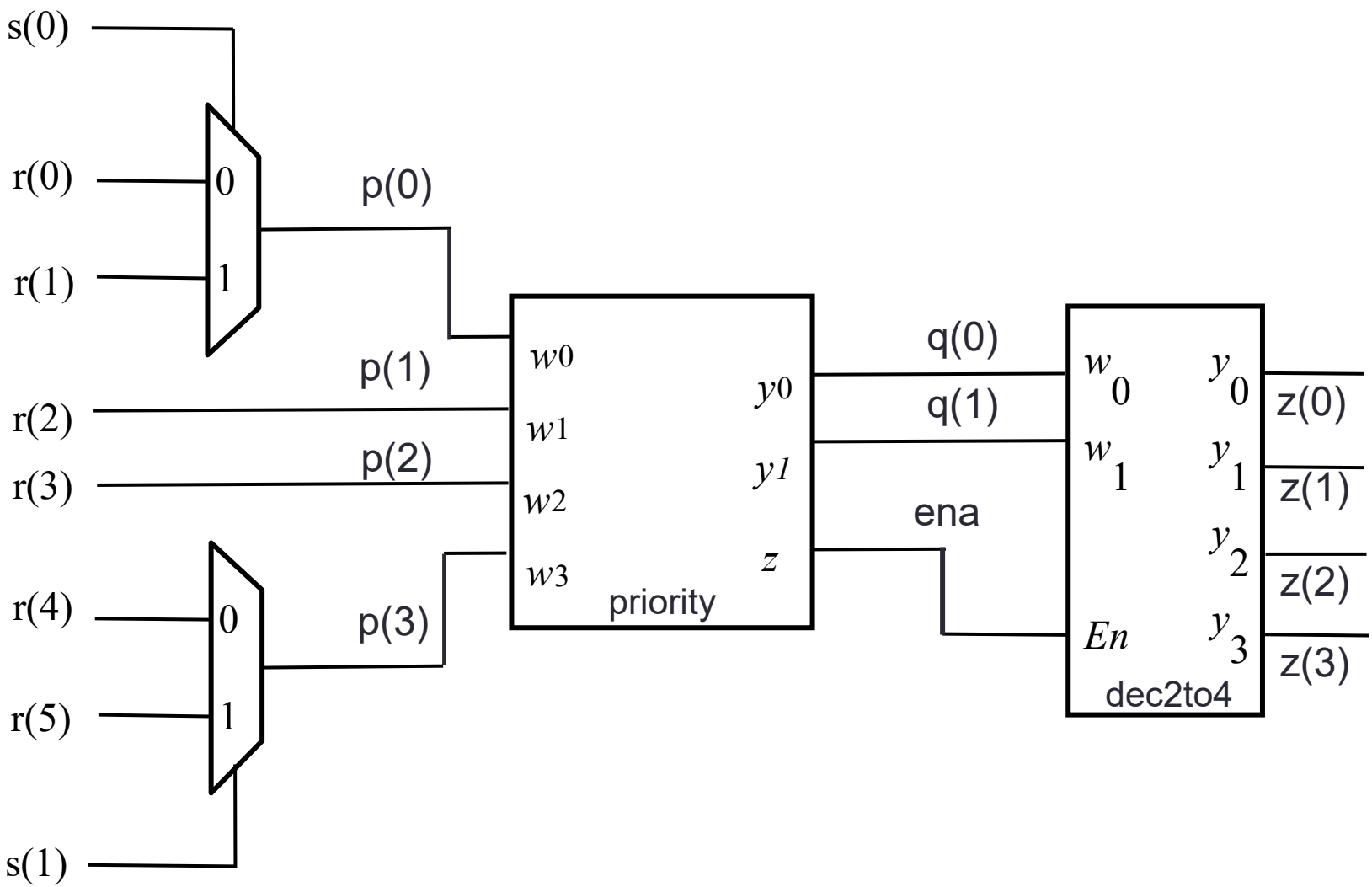
Structural VHDL

- **component instantiation (port map)**
- **component instantiation with generic (generic map, port map)**
- generate scheme for component instantiations (for-generate)

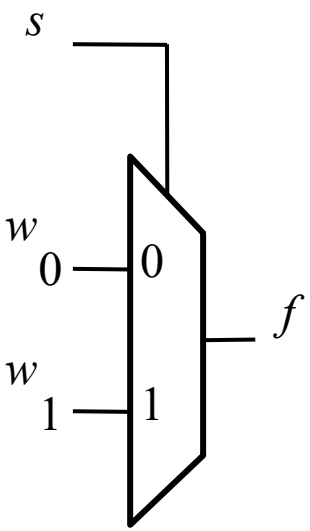
Components

- A component is an **instantiation of an entity** (plus an architecture)
- Allows a designer to re-use common pieces of code (i.e. registers, counters, etc.)
- Two ways of declaring components
  - METHOD #1: Explicit component declaration
    - Components declared in main code
  - METHOD #2: Package component declaration
    - Components declared in a package
- **Component declaration tells the compiler the ports of the components about to be instantiated**

Circuit built of medium scale components



2-to-1 Multiplexer



(a) Graphical symbol

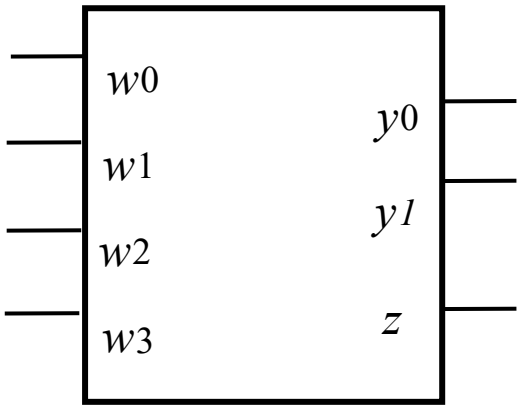
s	f
0	w_0
1	w_1

(b) Truth table

VHDL code for a 2-to-1 Multiplexer

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

ENTITY mux2to1 IS
     PORT (     w0, w1          : IN       STD_LOGIC ;
                 s              : IN       STD_LOGIC ;
                f               : OUT      STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
     f <= w0 WHEN s = '0' ELSE w1 ;
END dataflow ;
```

Priority Encoder



w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

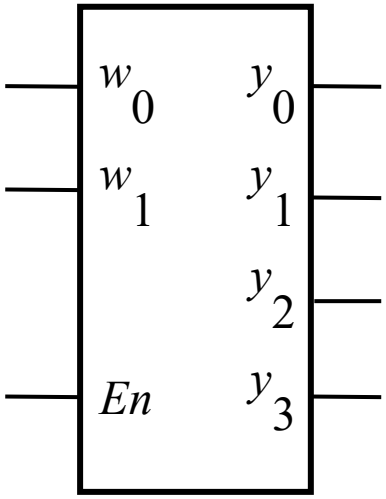
VHDL code for a Priority Encoder

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

ENTITY priority IS
     PORT ( w  : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               y    : OUT     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
               z    : OUT     STD_LOGIC ) ;
END priority ;

ARCHITECTURE dataflow OF priority 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' ;
END dataflow ;
```

2-to-4 Decoder

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



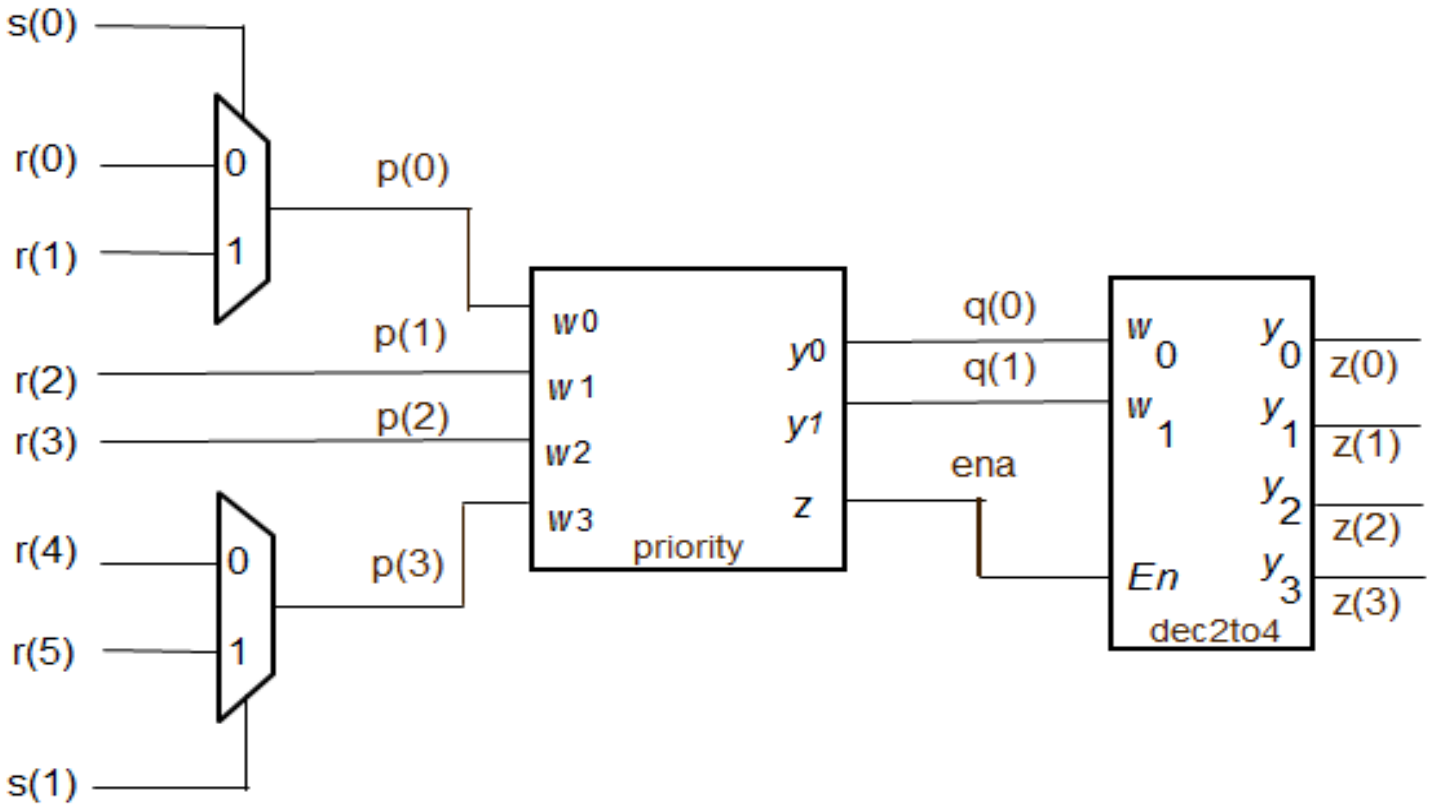
(b) Graphical symbol

VHDL code for a 2-to-4 Decoder

```
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(3 DOWNTO 0) ) ;
END dec2to4 ;

ARCHITECTURE dataflow OF dec2to4 IS
     SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
BEGIN
     Enw <= En & w ;
     WITH Enw SELECT
               y <= "0001" WHEN "100",
                    "0010" WHEN "101",
                    "0100" WHEN "110",
                    "1000" WHEN "111",
                    "0000" WHEN OTHERS ;
END dataflow ;
```

Task #6

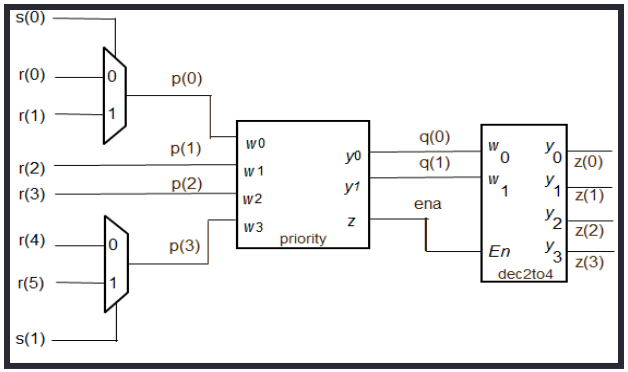
- Write the VHDL code that describes this circuit using the given codes of the building blocks.



METHOD #1: Explicit component declaration

- Components declared in main code
- Actual instantiations and port maps always in main code

Structural description – example (1)




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

ENTITY priority_resolver IS
    PORT (r          : IN       STD_LOGIC_VECTOR(5 DOWNTO 0);
          s          : IN       STD_LOGIC_VECTOR(1 DOWNTO 0);
          Z          : OUT      STD_LOGIC_VECTOR(3 DOWNTO 0) );
END priority_resolver;

ARCHITECTURE structural OF priority_resolver IS

SIGNAL  p : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL  q : STD_LOGIC_VECTOR (1 DOWNTO 0) ;
SIGNAL  ena : STD_LOGIC ;

COMPONENT mux2to1
    PORT (w0, w1, s   : IN       STD_LOGIC ;
          f           : OUT      STD_LOGIC ) ;
END COMPONENT ;

COMPONENT priority
    PORT (w           : IN       STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          y           : OUT      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
          z           : OUT      STD_LOGIC ) ;
END COMPONENT ;

COMPONENT dec2to4
    PORT (w           : IN       STD_LOGIC_VECTOR(1 DOWNTO 0) ;
          En          : IN       STD_LOGIC ;
          y           : OUT      STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END COMPONENT ;
```

Structural description – example (2)

```
BEGIN
   u1: mux2to1 PORT MAP (     w0 => r(0) ,
                              w1 => r(1),
                              s => s(0),
                              f => p(0));
   p(1) <= r(2);
   p(2) <= r(3);

   u2: mux2to1 PORT MAP (     w0 => r(4) ,
                              w1 => r(5),
                              s => s(1),
                              f => p(3));

   u3: priority PORT MAP (    w => p,
                              y => q,
                              z => ena);

   u4: dec2to4 PORT MAP (     w => q,
                              En => ena,
                              y => z);

END structural;
```

Local Wire/Port

Entity Port Name

Generic Statement

- The **generic** statement allows components to be parameterized, mostly with size values like the size of an adder (8-bit, 16-bit, 32-bit, etc.).
- The **generic** statement is part of the entity description of a module/component.

Generic Statement: Declaration

- Example: n-bit Nand gate

Parameter name

Parameter type

Default value

```
ENTITY nand_gate IS
    GENERIC(
        n    :Integer   :=4
        );
    PORT(
        a    : IN STD_LOGIC_VECTOR(n-1 downto 0);
        b    : IN STD_LOGIC_VECTOR(n-1 downto 0);
        z    : OUT STD_LOGIC_VECTOR(n-1 downto 0)
      );
END nand_gate;
```

Modify the input width

Generic Statement: Instantiation

- Example: n-bit Nand gate

```
U1: nand_gate
    GENERIC MAP (n => 8)
    PORT MAP(
        a => In1,
        b => In2,
        z => Out
      );
```

Neither Semicolon nor comma

Task #7

- Write the VHDL code that describe **N-bit Shifter**:

Selection S1 S0	Operation
0 0	Shift Left
0 1	Shift Right
1 0	Rotate Left
1 1	Rotate Right

Task #7(solution)

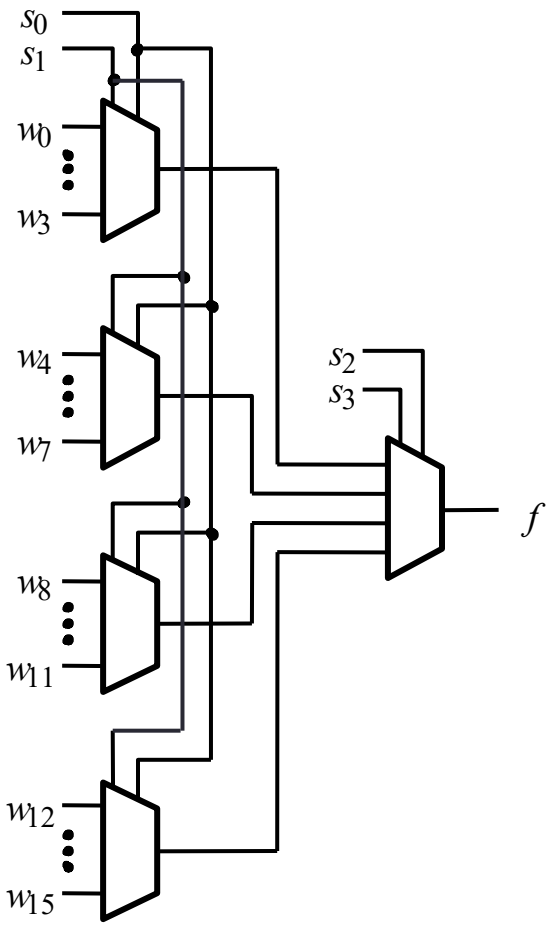
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY N_SHIFTER IS
     GENERIC(
           N: Integer:=4);
     PORT(
           A: IN STD_LOGIC_VECTOR(N-1 downto 0);
           Sel: IN STD_LOGIC_VECTOR(1 downto 0);
           Z: OUT STD_LOGIC_VECTOR(N-1 downto 0)
          );
END N_SHIFTER;

ARCHITECTURE Combinational OF N_SHIFTER IS
SIGNAL X1,X2,X3,X4: STD_LOGIC_VECTOR(N-1 downto 0);
BEGIN
    X1<=A(N-2 downto 0) & '0';
    X2<='0' & A(N-1 downto 1);
    X3<=A(N-2 downto 0) & A(N-1);
    X4<=A(0) & A(N-1 downto 1);

    WITH Sel SELECT
           Z <= X1 WHEN "00",
                X2 WHEN "01",
                X3 WHEN "10",
                X4 WHEN others;
END Combinational;
```

Structural VHDL

- component instantiation (port map)
- component instantiation with generic (generic map, port map)
- **generate scheme for component instantiations (for-generate)**

A 16-to-1 Multiplexer

Write the VHDL code that describe 16-to-1 Multiplexer out of 4-to-1 multiplexers



A 4-to-1 Multiplexer

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

ENTITY mux4to1 IS
    PORT (w0, w1, w2, w3      : IN      STD_LOGIC ;
                   s          : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                   f          : OUT     STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE Dataflow OF mux4to1 IS
BEGIN
    WITH s SELECT
          f <= w0 WHEN "00",
                   w1 WHEN "01",
                   w2 WHEN "10",
                   w3 WHEN OTHERS ;
END Dataflow ;
```

Straightforward code for A 16-to-1 Multiplexer

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

ENTITY Mux16to1 IS
    PORT (     w     : IN      STD_LOGIC_VECTOR(15 DOWNTO 0) ;
               s     : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               f     : OUT     STD_LOGIC ) ;
END Mux16to1 ;
ARCHITECTURE Structure OF Mux16to1 IS

    COMPONENT mux4to1
          PORT (    w0, w1, w2, w3      : IN      STD_LOGIC ;
                    s                   : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                    f                   : OUT     STD_LOGIC ) ;
    END COMPONENT ;

    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 ;
```

A 16-to-1 Multiplexer using For-Generate

```
LIBRARY ieee ;
USE ieee.std_logic_1164.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

COMPONENT mux4to1
    PORT (  w0, w1, w2, w3  : IN  STD_LOGIC ;
      s   : IN  STD_LOGIC_VECTOR(1 DOWNTO 0) ;
      f   : OUT   STD_LOGIC ) ;
END COMPONENT ;

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 ;
```