ECE 448
Lecture 5

Combinational-Circuit Building Blocks

Part 2

Reading

Required

- P. Chu, *FPGA Prototyping by VHDL Examples*

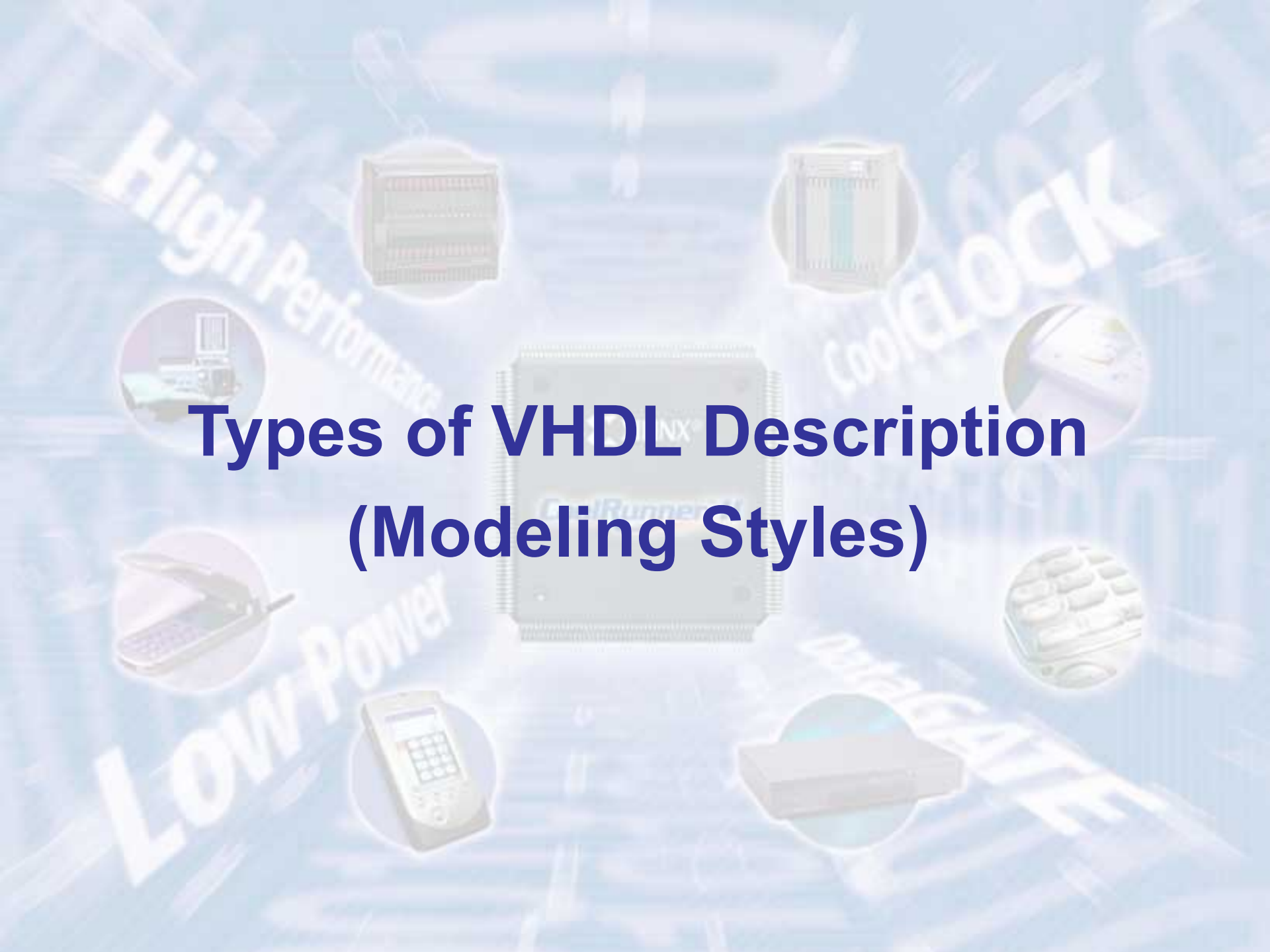
  Chapter 3, RT-level combinational circuit

Recommended

- S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*

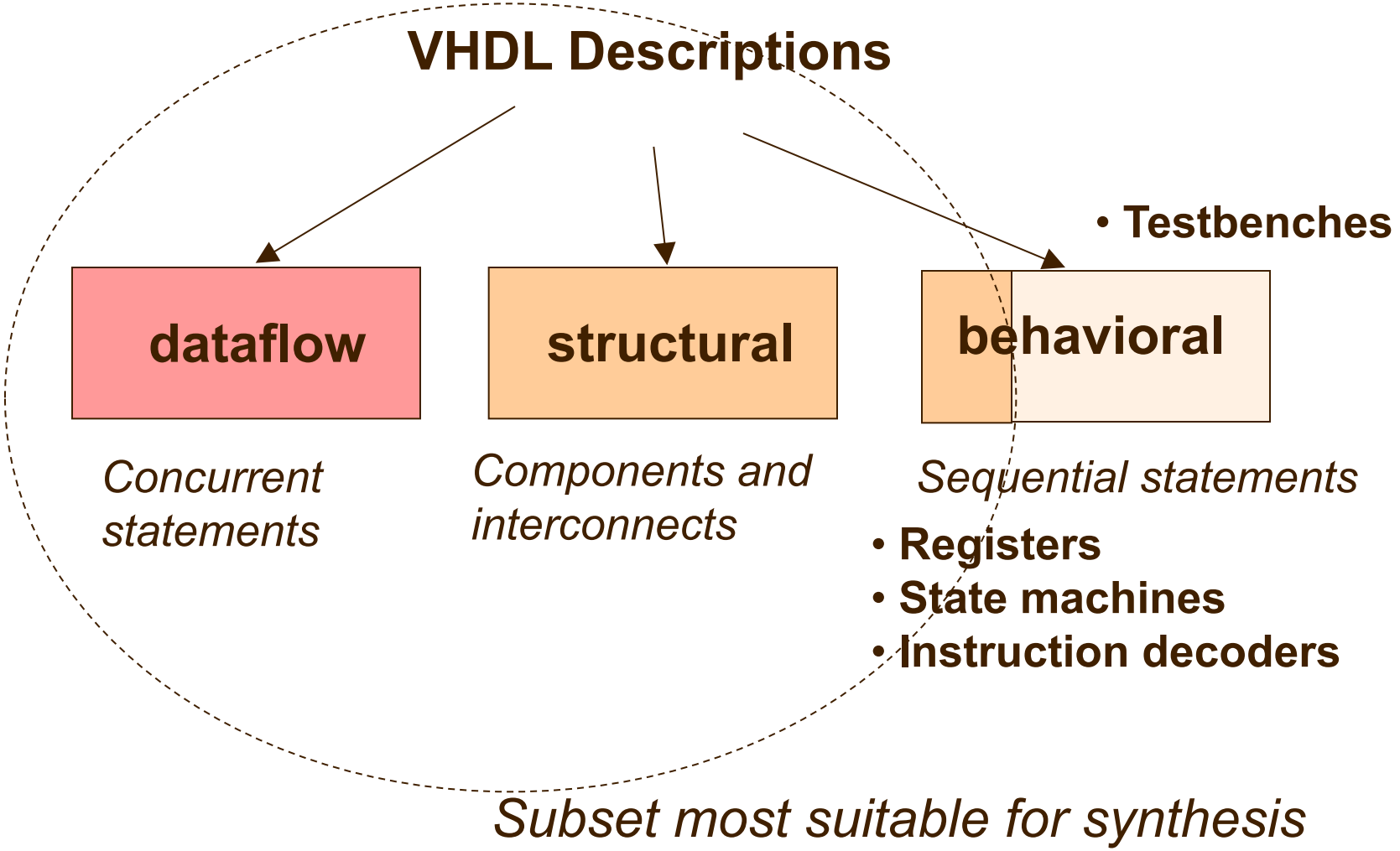
  Chapter 6, Combinational-Circuit Building Blocks

  Chapter 5.5, Design of Arithmetic Circuits Using CAD Tools

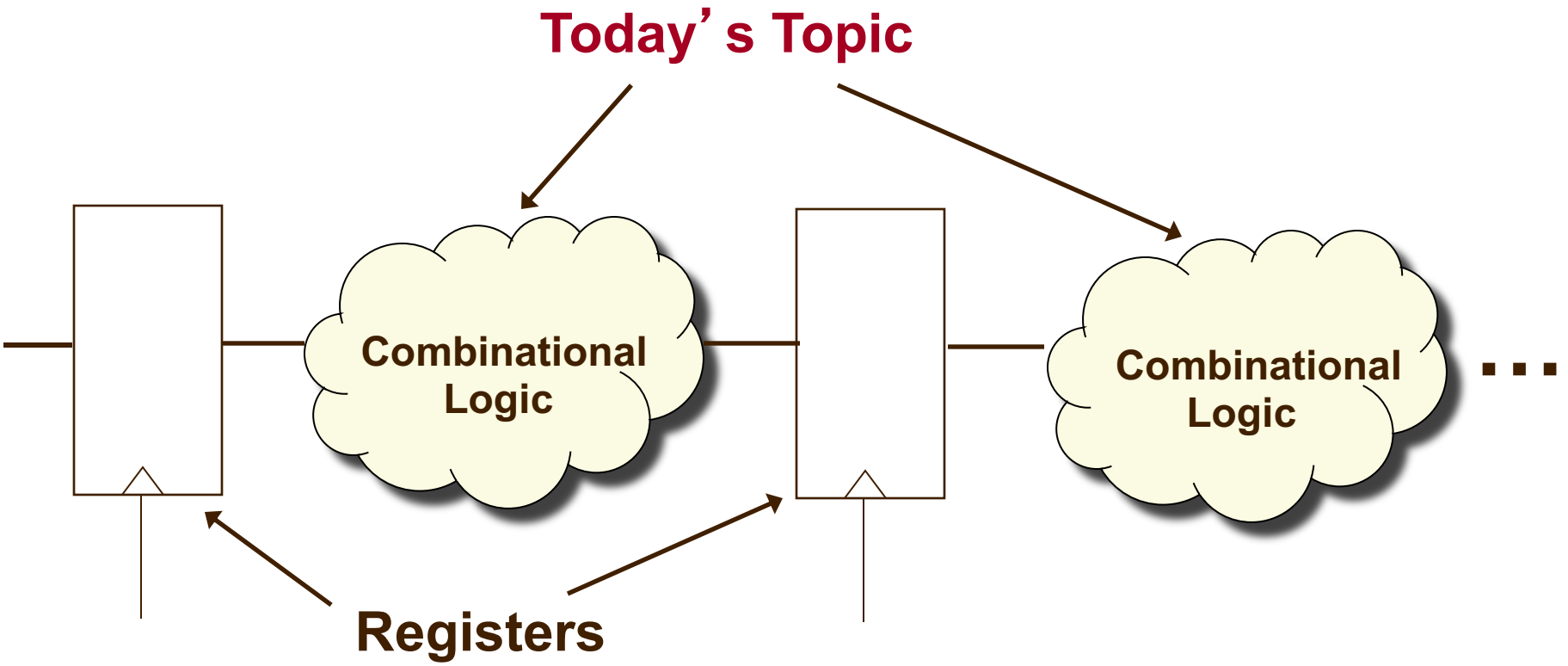


Types of VHDL Description (Modeling Styles)

Types of VHDL Description



Register Transfer Level (RTL) Design Description



- Use of medium scale-components (adders, multipliers, MUXes, ROMs)
- The designer needs to specify what happens in the circuit in every clock cycle

Data-Flow VHDL

Concurrent Statements

- **simple concurrent signal assignment**
  (⇐)
- **conditional concurrent signal assignment**
  (when-else)
- **selected concurrent signal assignment**
  (with-select-when)

Concurrent signal assignment

<=

```
target_signal <= expression;
```

Conditional concurrent signal assignment

When - Else

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

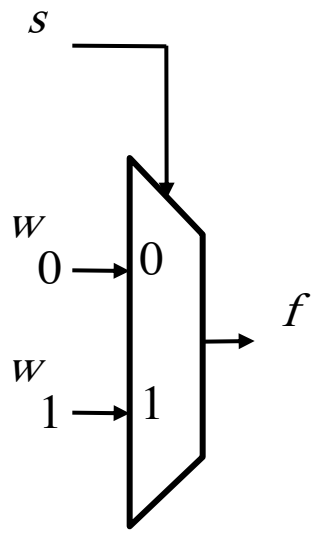
Selected concurrent signal assignment

With –Select-When

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

Multiplexers

2-to-1 Multiplexer



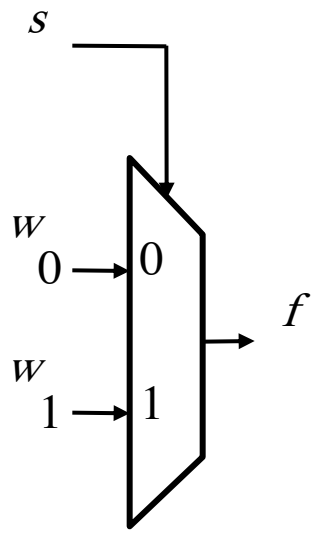
(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table

VHDL:

2-to-1 Multiplexer



(a) Graphical symbol

s	f
0	w_0
1	w_1

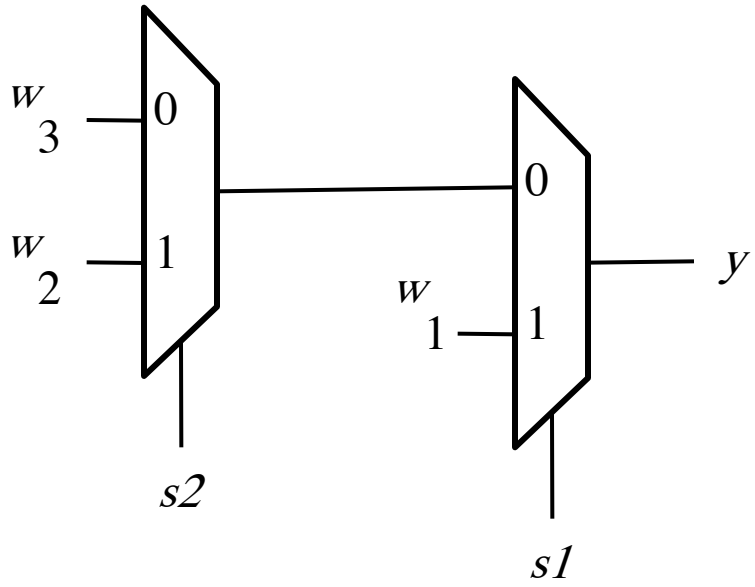
(b) Truth table

VHDL: **f <= w1 WHEN s = '1' ELSE w0 ;**

or

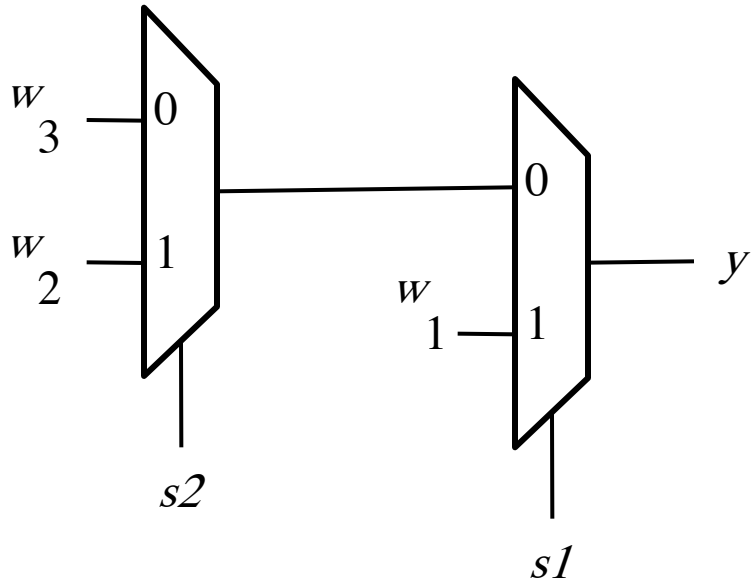
f <= w0 WHEN s = '0' ELSE w1 ;

Cascade of two multiplexers



VHDL:

Cascade of two multiplexers



VHDL:

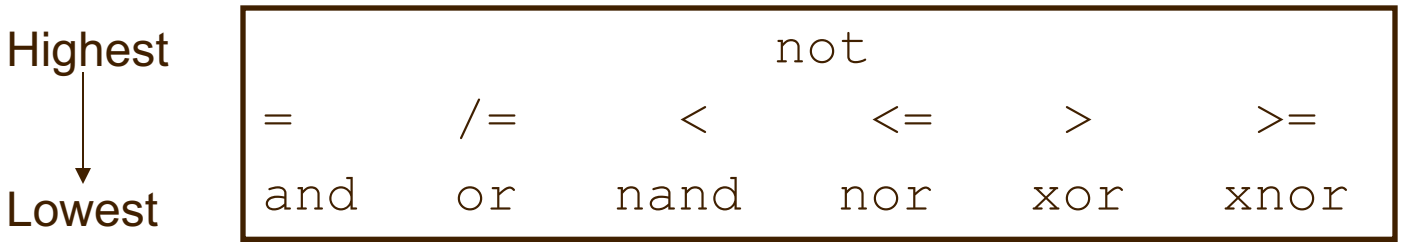
```
y <= w1  WHEN s1 =  '1' ELSE
     w2  WHEN s2 =  '1' ELSE
     w3 ;
```

Operators

- Relational operators

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

- Logic and relational operators precedence



Priority of logic and relational operators

compare a = bc

Incorrect

… when a = b **and** c else …

equivalent to

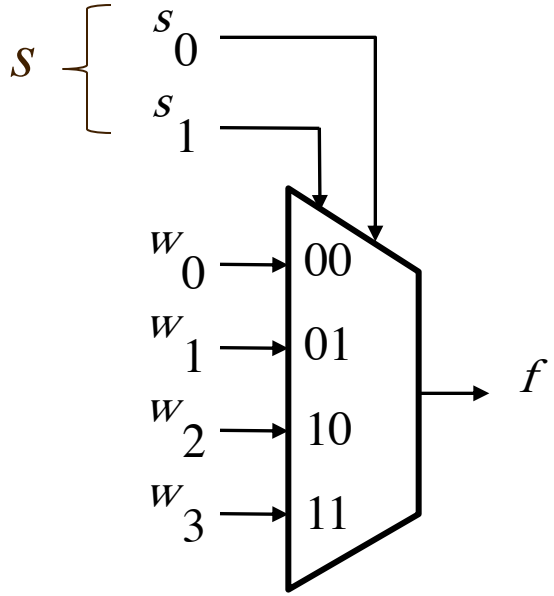
… when (a = b) **and** c else …

Correct

… when a = (b **and** c) else …

4-to-1 Multiplexer

(a) Graphic symbol

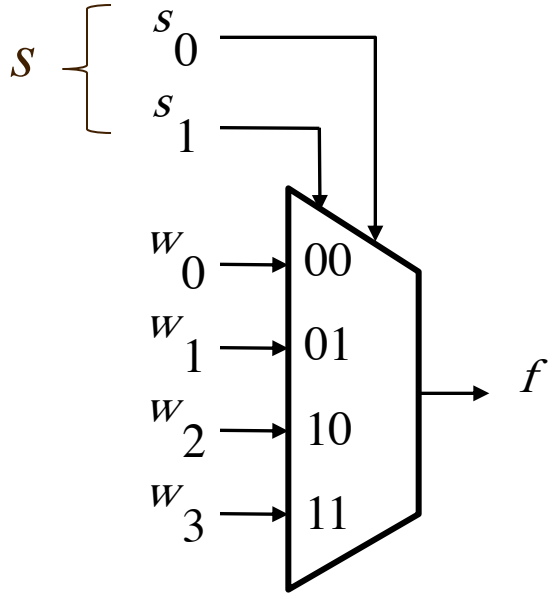


(b) Truth table

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

4-to-1 Multiplexer

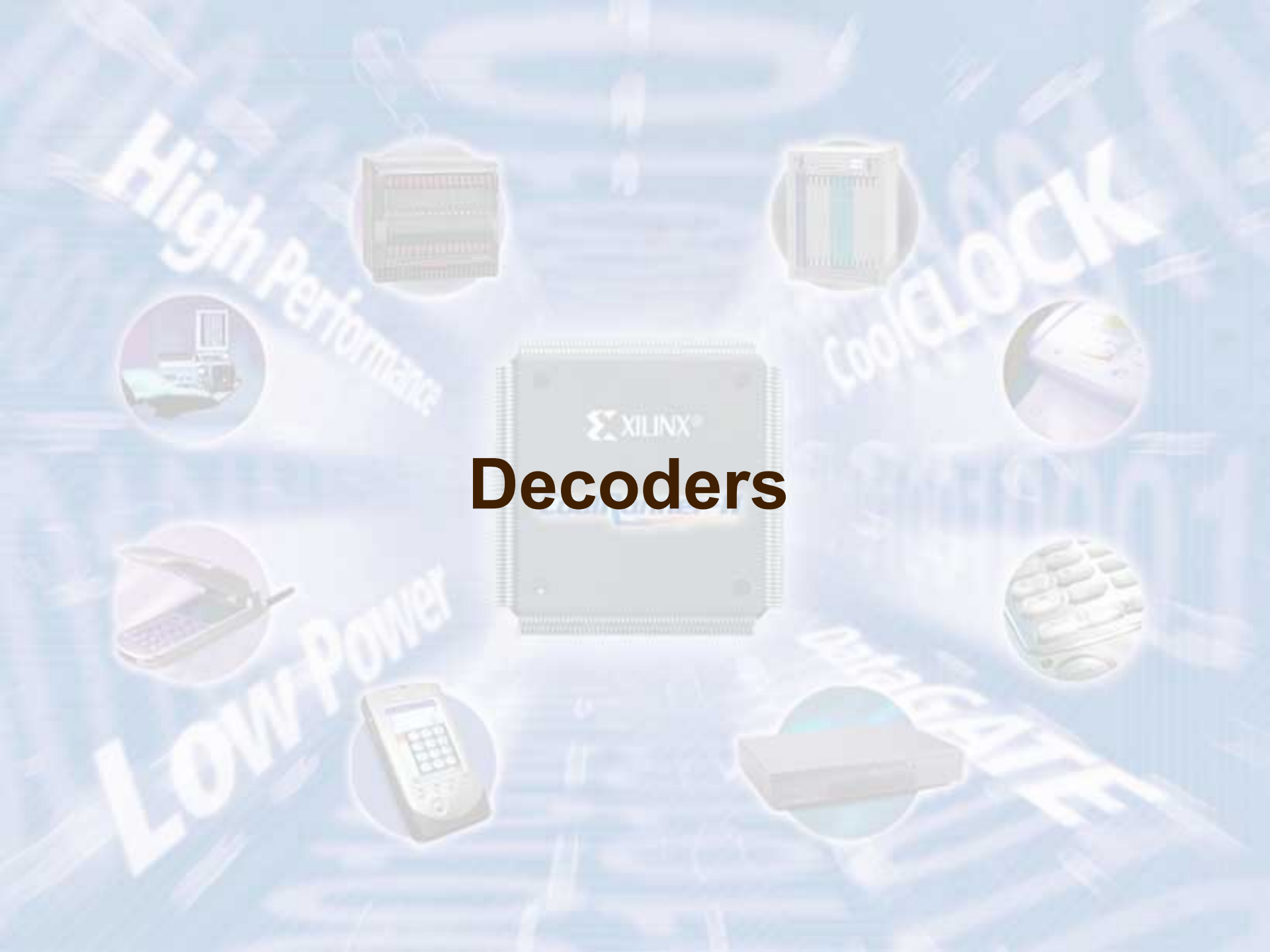
(a) Graphic symbol



(b) Truth table

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

```
WITH s SELECT
        f <= w0 WHEN "00",
             w1 WHEN "01",
             w2 WHEN "10",
             w3 WHEN OTHERS ;
```



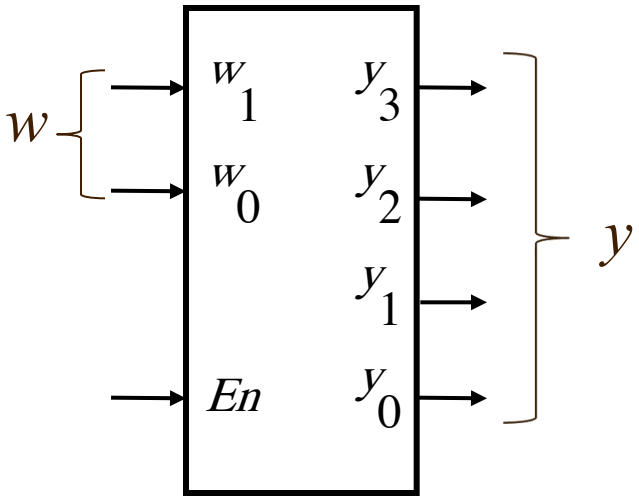
Decoders

2-to-4 Decoder

(a) Truth table

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

(b) Graphical symbol

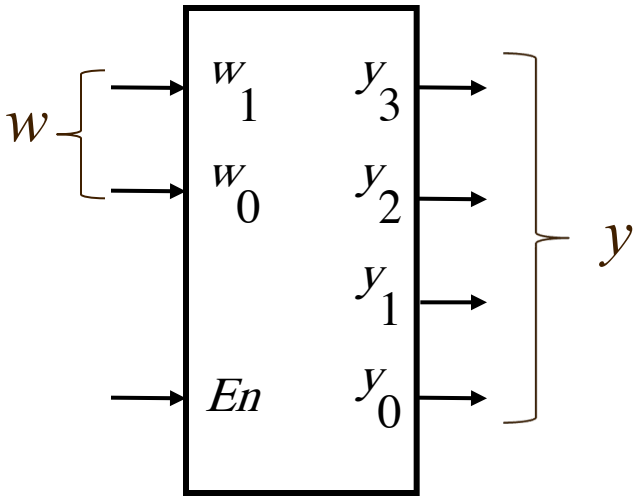


2-to-4 Decoder

(a) Truth table

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

(b) Graphical symbol

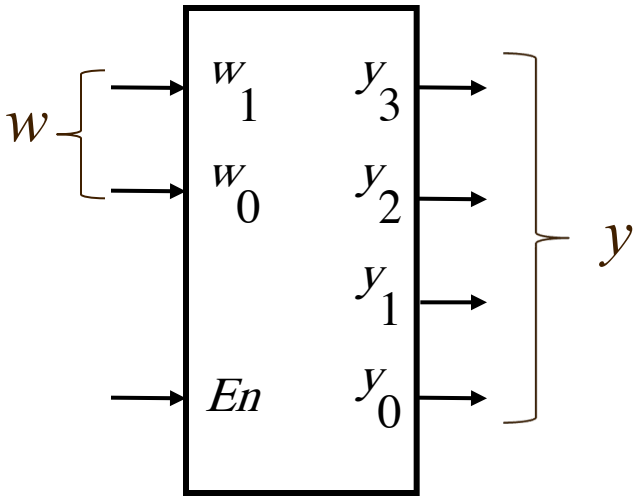


2-to-4 Decoder

(a) Truth table

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

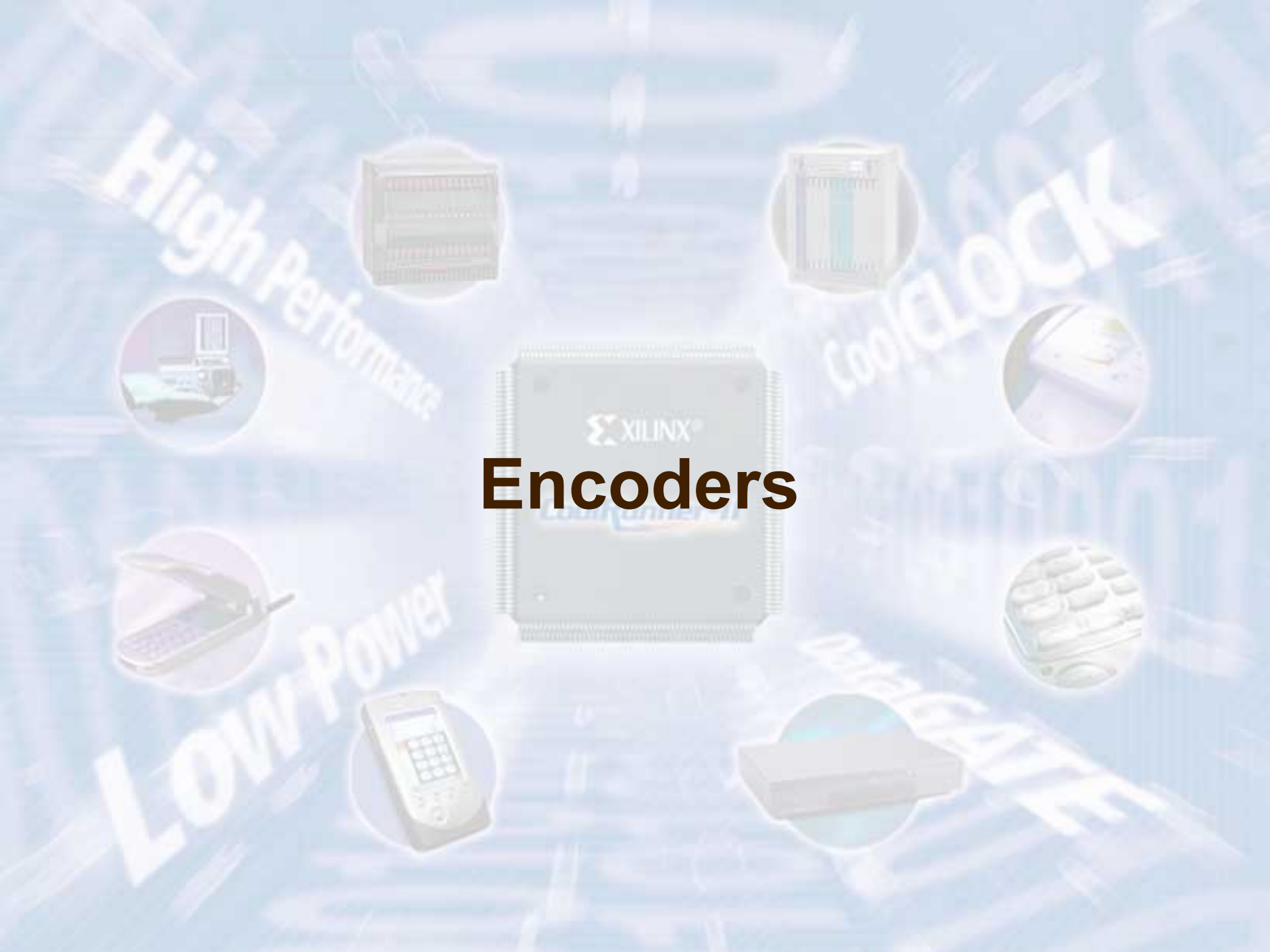
(b) Graphical symbol




```
Enw <= En & w ;
WITH Enw SELECT
    y <= "0001" WHEN "100",
         "0010" WHEN "101",
         "0100" WHEN "110",
         "1000" WHEN "111",
         "0000" WHEN OTHERS ;
```

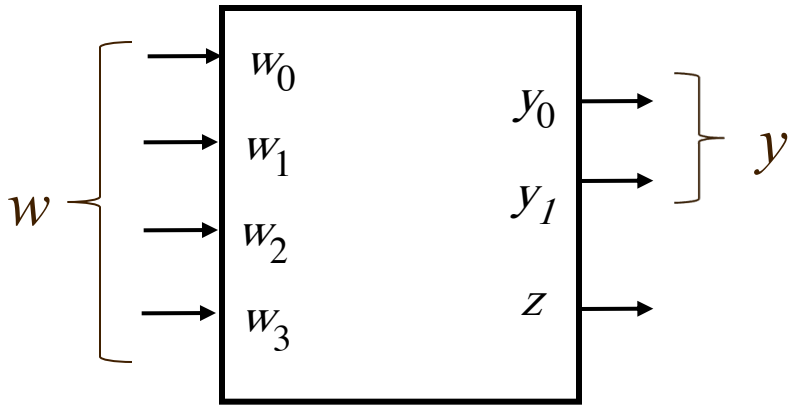
VHDL code for a 2-to-4 Decoder entity

```
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 ;
```



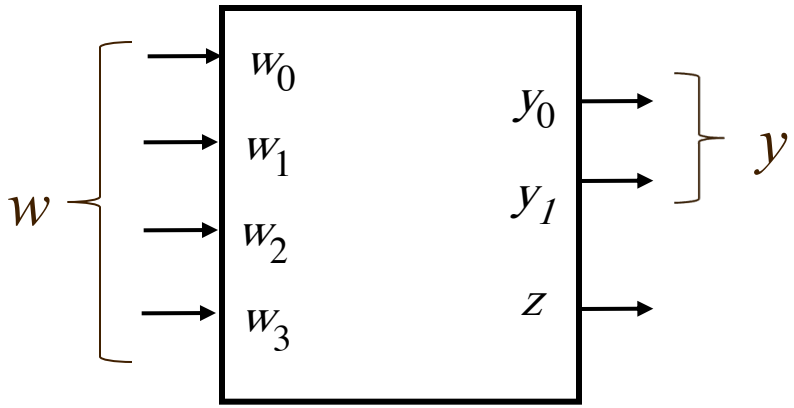
Encoders

Priority Encoder



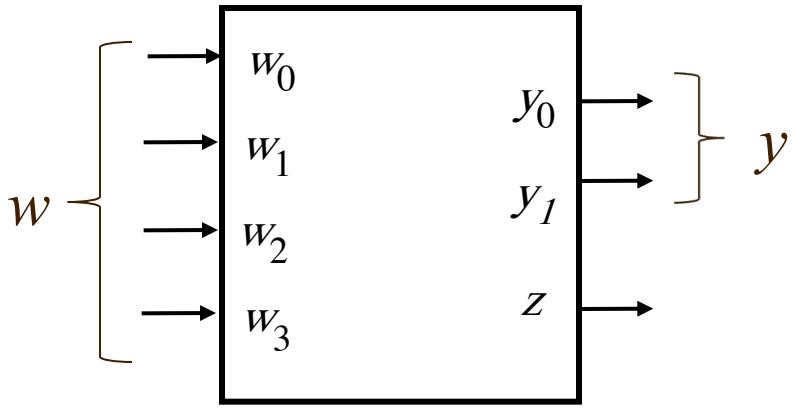
w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0			
0	0	0	1			
0	0	1	–			
0	1	–	–			
1	–	–	–			

Priority Encoder



w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	–	–	0
0	0	0	1	0	0	1
0	0	1	–	0	1	1
0	1	–	–	1	0	1
1	–	–	–	1	1	1

Priority Encoder



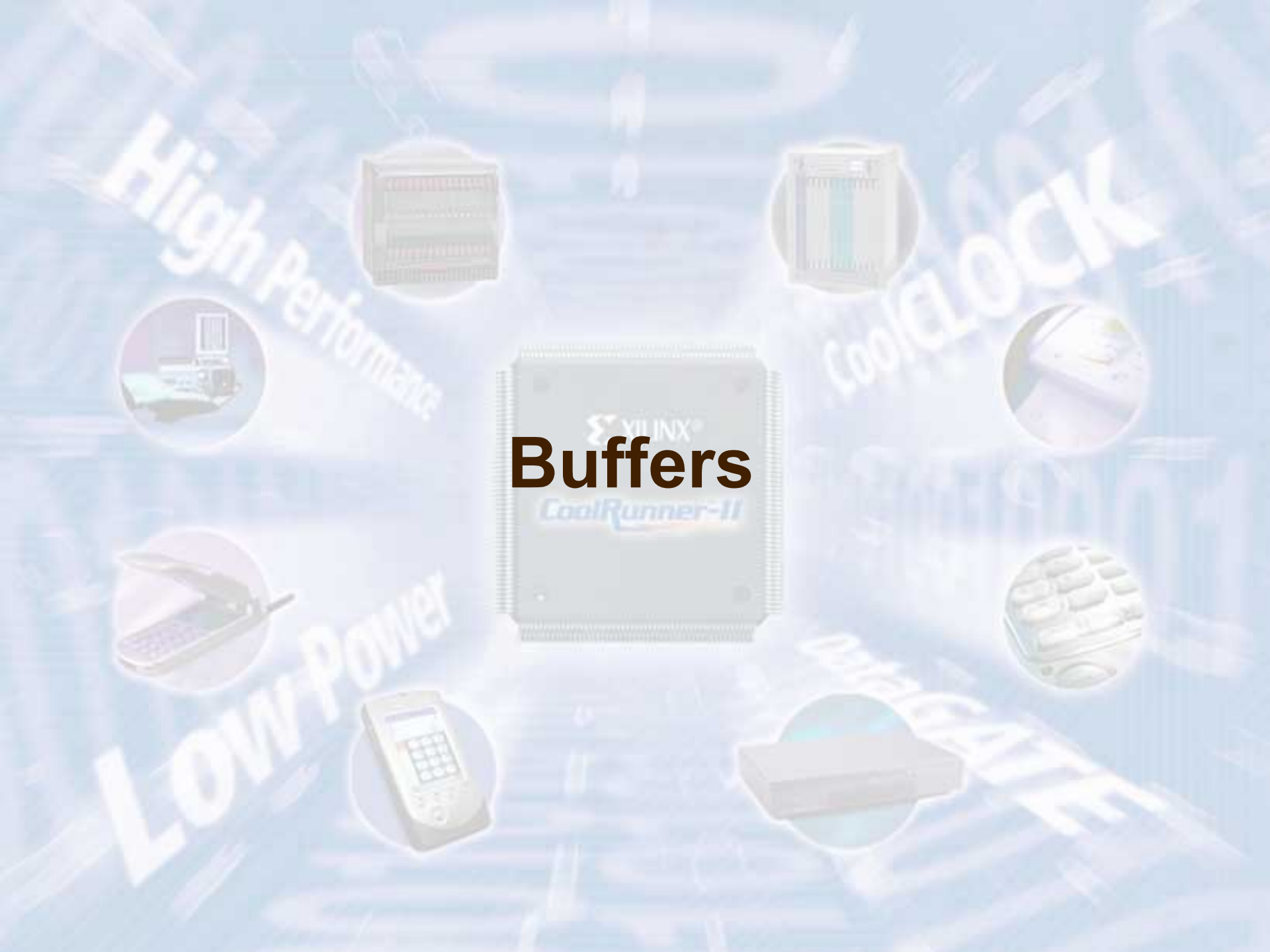
w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	–	–	0
0	0	0	1	0	0	1
0	0	1	–	0	1	1
0	1	–	–	1	0	1
1	–	–	–	1	1	1

```
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' ;
```

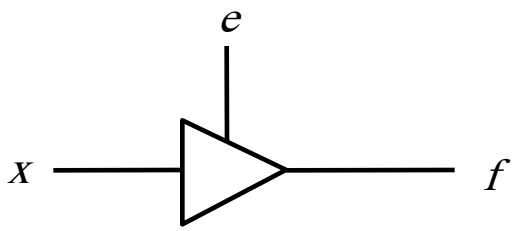
VHDL code for a Priority Encoder entity

```
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 ;
```

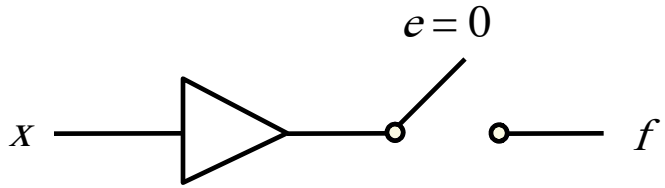


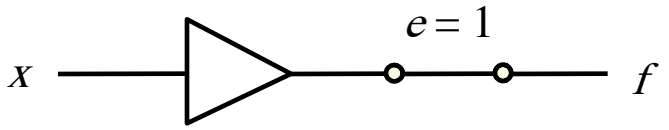
Buffers

Tri-state Buffer



(a) A tri-state buffer



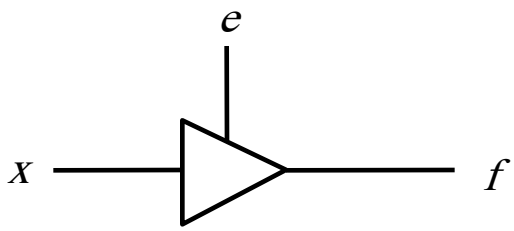


(b) Equivalent circuit

e	x	f
0	0	
0	1	
1	0	
1	1	

(c) Truth table

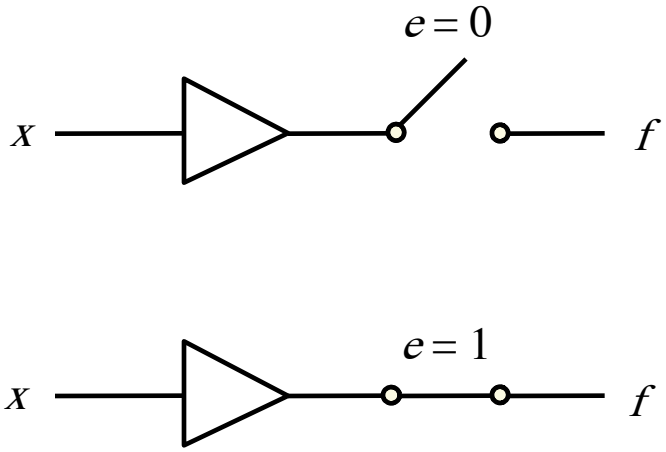
Tri-state Buffer



(a) A tri-state buffer

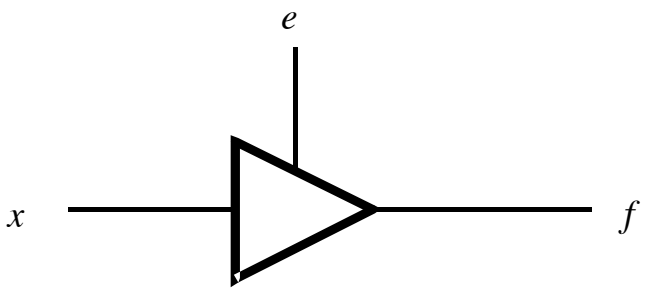
e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

(c) Truth table

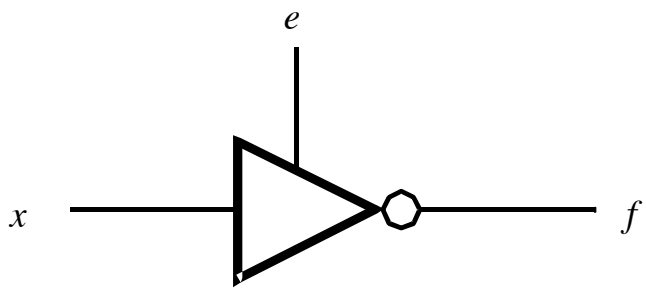


(b) Equivalent circuit

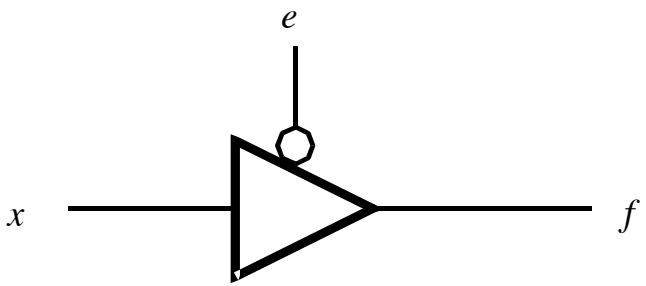
Four types of Tri-state Buffers



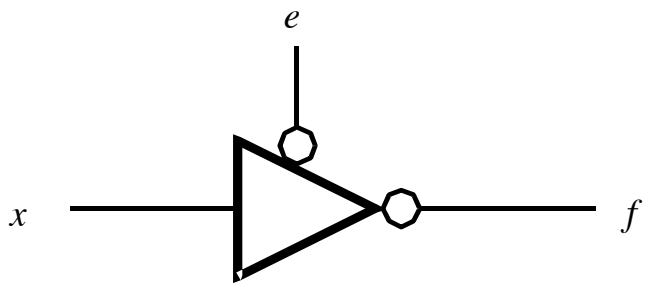
(a)



(b)

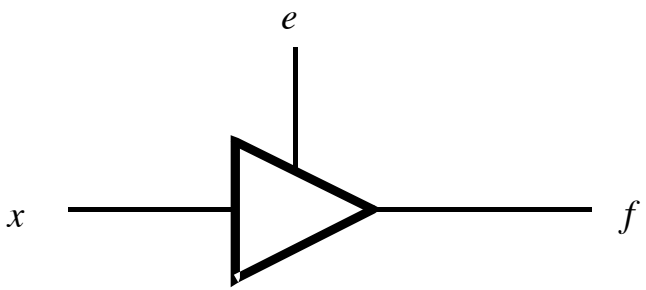


(c)

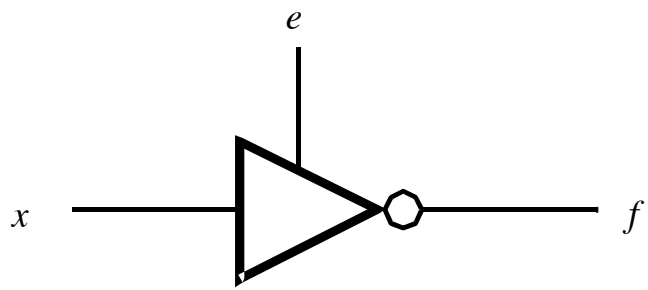


(d)

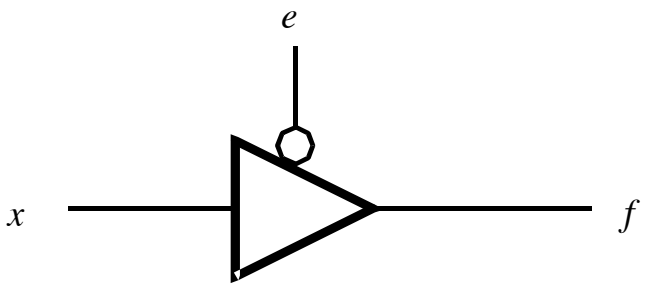
Four types of Tri-state Buffers



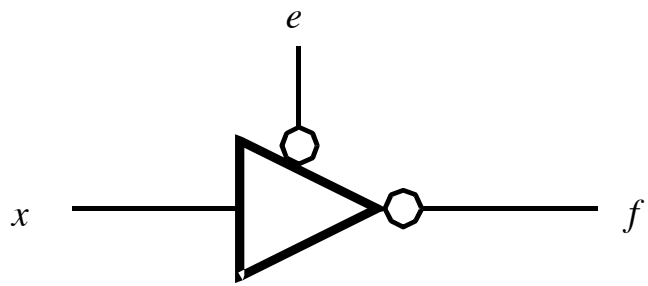
f <= x WHEN (e = '1') ELSE 'Z';



f <= not x WHEN (e = '1') ELSE 'Z';



f <= x WHEN (e = '0') ELSE 'Z';



f <= not x WHEN (e = '0') ELSE 'Z';

Tri-state Buffer entity (1)

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

ENTITY tri_state IS
   PORT ( e:    IN STD_LOGIC;
                x:  IN STD_LOGIC;
                f: OUT STD_LOGIC
               );
END tri_state;
```

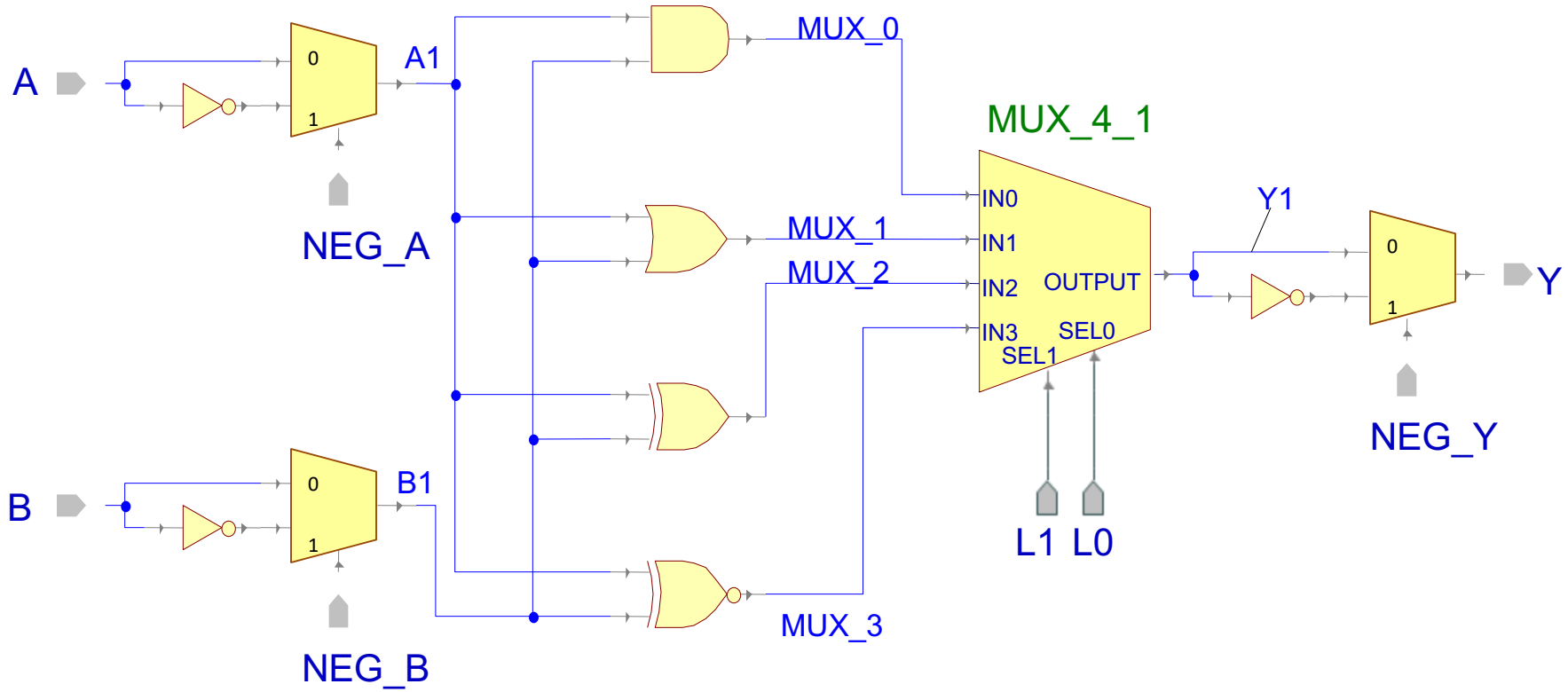
Tri-state Buffer entity (2)

```
ARCHITECTURE dataflow OF tri_state IS
BEGIN
    f <= x WHEN (e = '1') ELSE   'Z';
END dataflow;
```

Describing Combinational Logic Using Dataflow Design Style

MLU Example

MLU Block Diagram



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);
```

MLU - Architecture Body

```
BEGIN
    A1<= NOT A  WHEN (NEG_A='1') ELSE
         A;
    B1<= NOT B  WHEN (NEG_B='1') ELSE
         B;
    Y <= NOT Y1 WHEN (NEG_Y='1') ELSE
         Y1;

    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;
```

Logic Implied Most Often by Conditional and Selected Concurrent Signal Assignments

Data-Flow VHDL

Concurrent Statements

- **simple concurrent signal assignment**
  (⇐)
- **conditional concurrent signal assignment**
  (when-else)
- **selected concurrent signal assignment**
  (with-select-when)

Conditional concurrent signal assignment

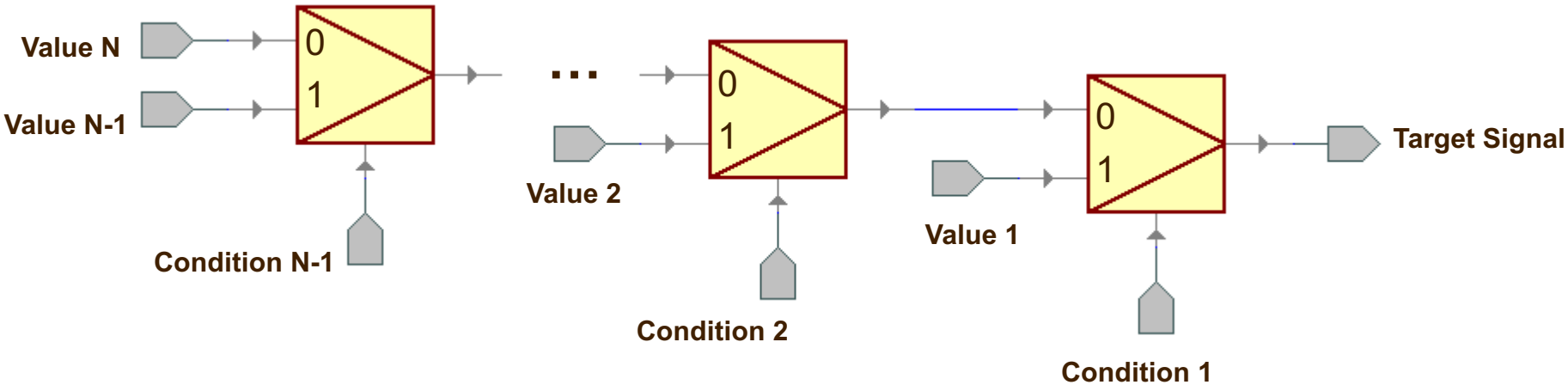
When - Else

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

Most often implied structure

When - Else

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



Data-Flow VHDL

Concurrent Statements

- **simple concurrent signal assignment**
  (⇐)
- **conditional concurrent signal assignment**
  (when-else)
- **selected concurrent signal assignment**
  (with-select-when)

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;
```

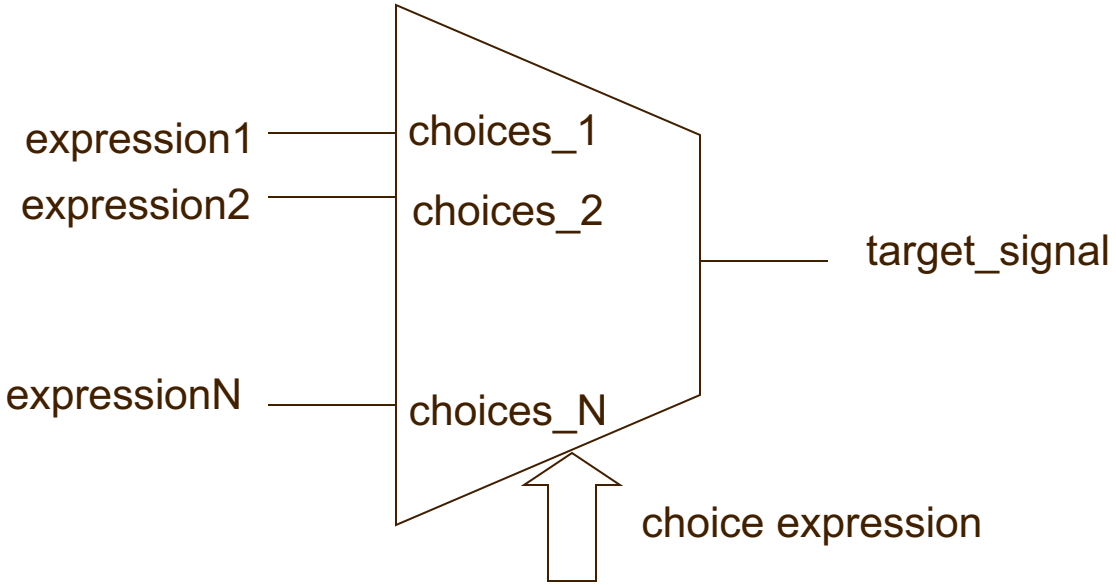
Selected concurrent signal assignment

- choice_expression
  - Discrete type or 1-D array
  - With finite possible values
- choices_k
  - A value of the data type used for choice_expression
- Choices must be
  - Mutually exclusive
  - All inclusive
  - OTHERS can be used as a last choice
- target_signal and expressions(1..N) must have the same type

Most Often Implied Structure

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 | value_2 | .... | value N

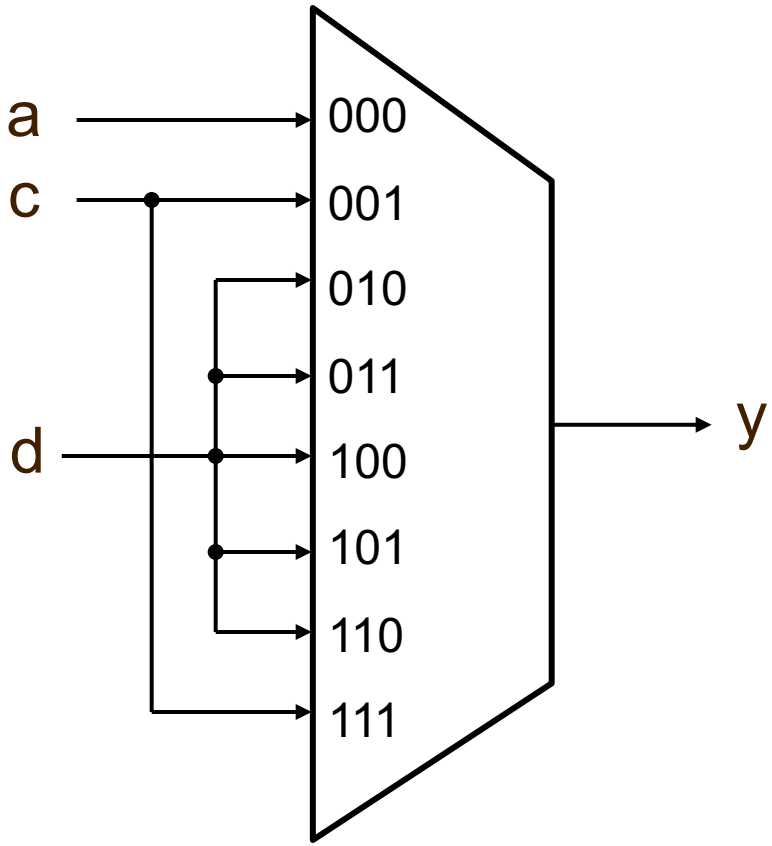
WHEN OTHERS

Allowed formats of *choices_k - example*

```
WITH sel SELECT
     y <= a WHEN "000",
           c WHEN "001" | "111",
           d WHEN OTHERS;
```

Corresponding circuit

```
WITH sel SELECT
     y <= a WHEN "000",
          c WHEN "001" | "111",
          d WHEN OTHERS;
```



when-else vs. with-select-when (1)

"when-else" should be used when:

1) there is only one condition (and thus, only one else), as in the 2-to-1 MUX

2) conditions are independent of each other (e.g., they test values of different signals)

3) conditions reflect priority (as in priority encoder); one with the highest priority need to be tested first.

when-else vs. with-select-when (2)

"with-select-when" should be used when there is

1) more than one condition
2) conditions are closely related to each other (e.g., represent different ranges of values of the same signal)
3) all conditions have the same priority (as in the 4-to-1 MUX).