

# Διάχυτα και Ενσωματωμένα Συστήματα

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Ακαδημαϊκό έτος 2018-19

VHDL:

Συνδυαστικά κυκλώματα –  
περιγραφή dataflow

# Προσομοίωση λειτουργίας

```
library IEEE;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity mux21 is port  
  (a, b, s: in bit;  
    c: out bit);  
end mux21;  
  
architecture struct1 of mux21 is  
  begin  
    c <= (a and s) or (b and not s);  
end struct1;
```

# Προσομοίωση λειτουργίας

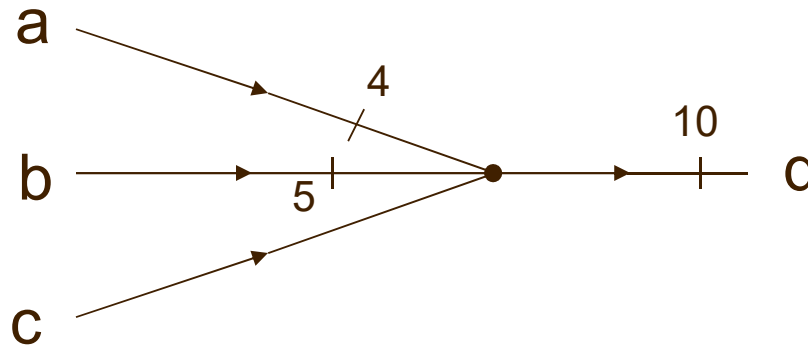
```
entity test_mux is  
end test_mux;  
  
architecture test_b of test_mux is  
signal A1, B1, S1, C1: bit;  
component mux21 port (a, b, s: in bit; c: out bit);  
end component;  
  
begin  
M1: mux21 PORT MAP (a=>A1, b=>B1, s=>s1, c=>c1);  
process  
  begin  
    A1 <= '1'; B1 <= '0'; S1 <= '1'; wait for 20 ns;  
    A1 <= '1'; B1 <= '0'; S1 <= '0'; wait for 20 ns;  
  end process;  
end test_b;
```

# Data-Flow VHDL

## Concurrent Statements

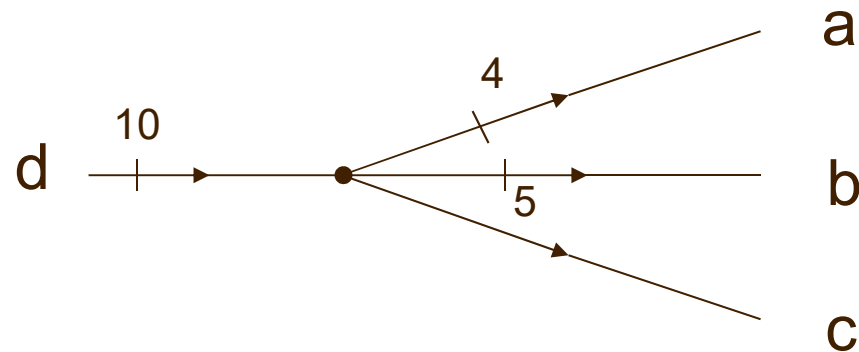
- **concurrent signal assignment**  
**( $\Leftarrow$ )**
- **conditional concurrent signal assignment**  
**(when-else)**
- **selected concurrent signal assignment**  
**(with-select-when)**

# Merging wires and buses



```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);  
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);  
SIGNAL c: STD_LOGIC;  
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);  
  
d <= a & b & c;
```

# Splitting buses



```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);  
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);  
SIGNAL c: STD_LOGIC;  
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);  
  
a <= d(9 downto 6);  
b <= d(5 downto 1);  
c <= d(0);
```

# Operators

- Relational operators

=	/=	<	<=	>	>=
---	----	---	----	---	----

- Logic and relational operators precedence

Highest  
↓  
Lowest

=	/=	<	<=	>	>=
and	or	nand	nor	xor	xnor

# Priority of logic and relational operators

compare  $a = bc$

## Incorrect

... when  $a = b$  **and**  $c$  else ...

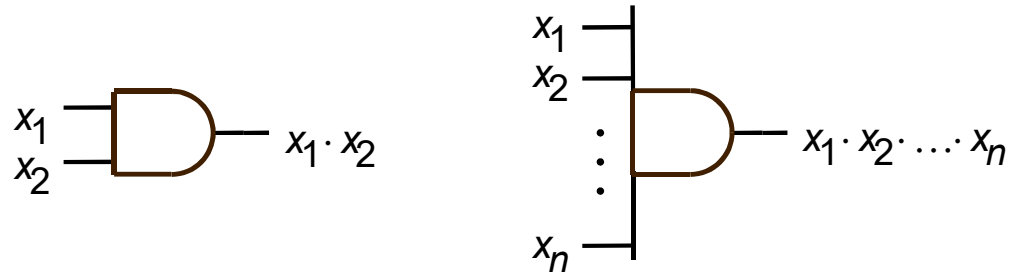
equivalent to

... when  $(a = b)$  **and**  $c$  else ...

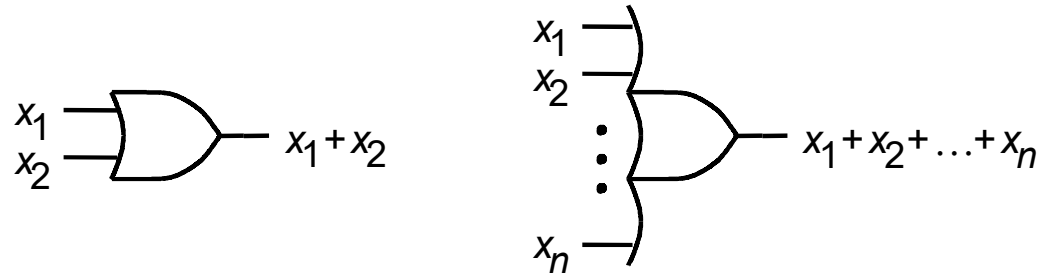
## Correct

... when  $a = (b$  **and**  $c)$  else ...

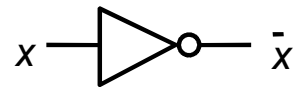
# Basic Gates – AND, OR, NOT



(a) AND gates

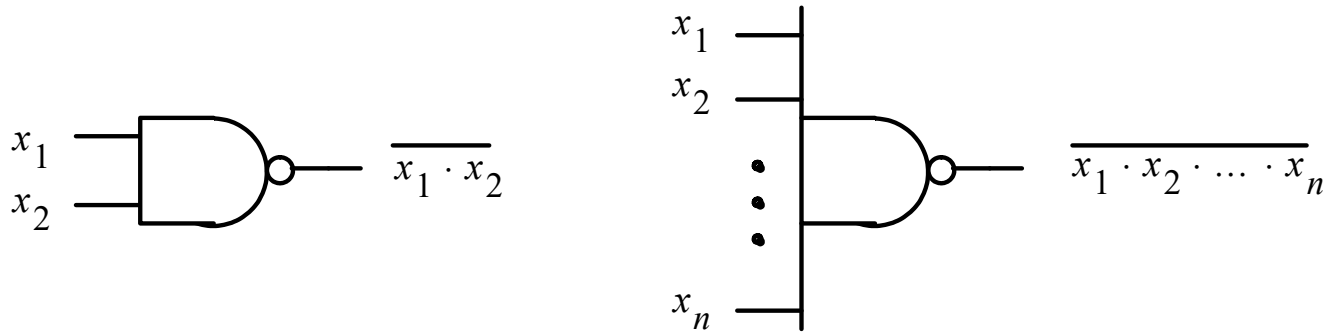


(b) OR gates

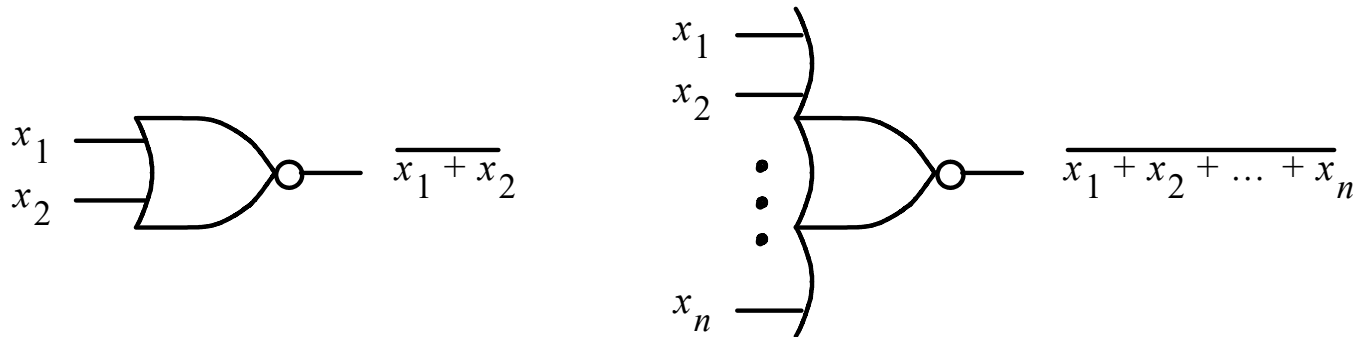


(c) NOT gate

# Basic Gates – NAND, NOR

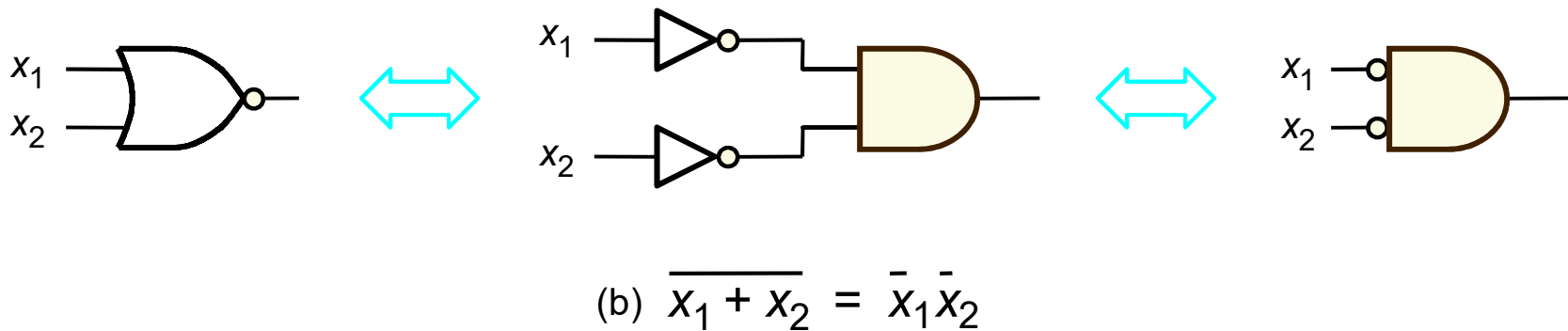
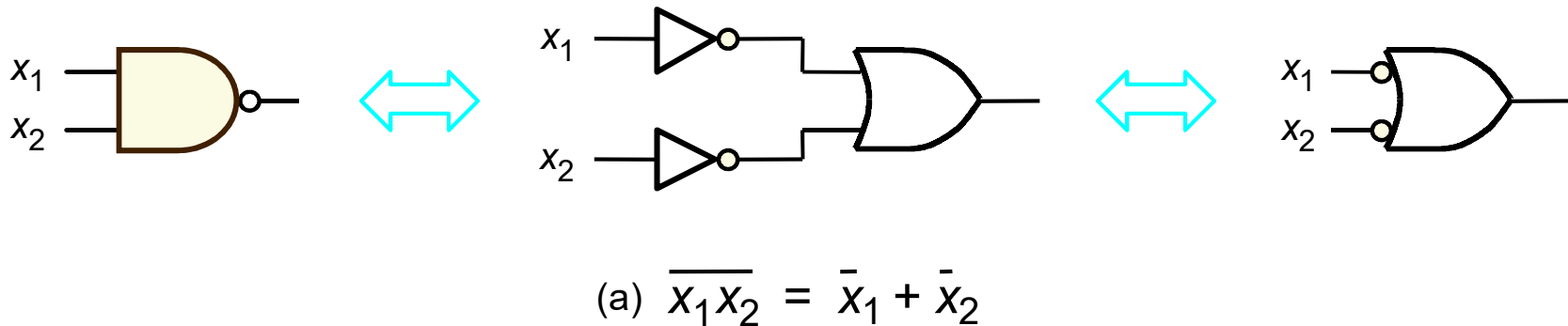


(a) NAND gates

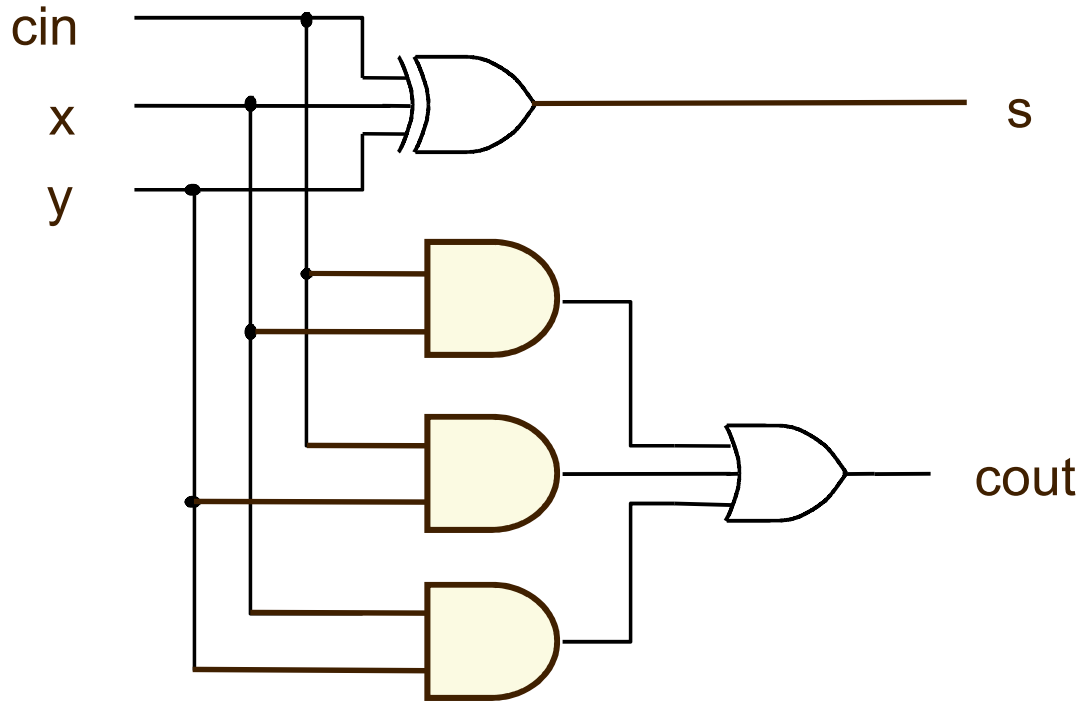


(b) NOR gates

# DeMorgan's Theorem and other symbols for NAND, NOR



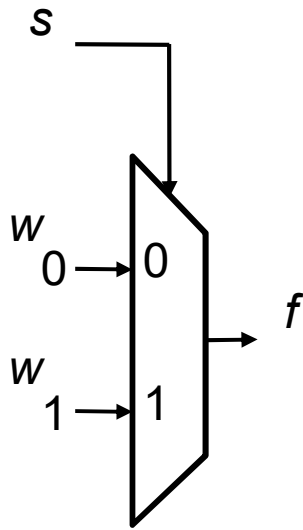
# Data-flow VHDL Example 1: Full adder



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

# Χρήση της δομής when: 2-to-1 Multiplexer



Graphical symbol

s	f
0	w <sub>0</sub>
1	w <sub>1</sub>

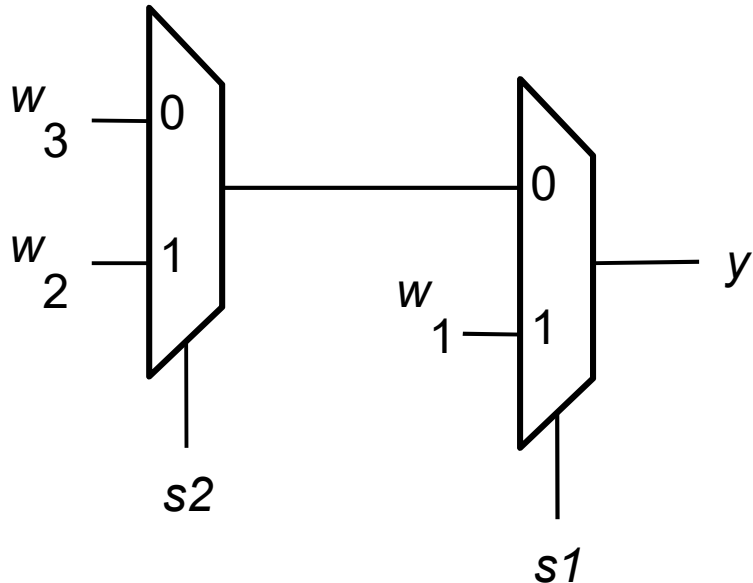
Truth table

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

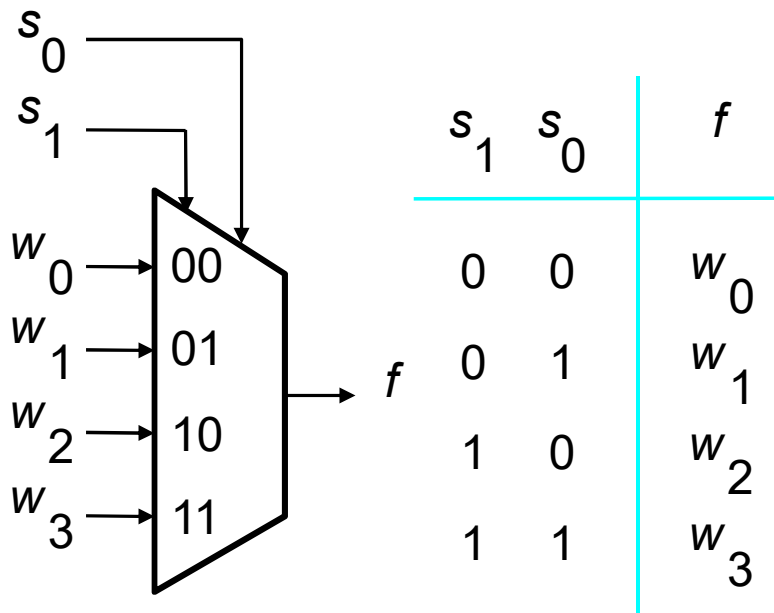
# Χρήση της δομής when: Cascade of two multiplexers



```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
  
ENTITY mux_cascade IS PORT (  
    w1, w2, w3: IN STD_LOGIC ;  
    s1, s2: IN STD_LOGIC ;  
    f: OUT STD_LOGIC ) ;  
END mux_cascade;
```

```
ARCHITECTURE dataflow OF mux2to1 IS  
BEGIN  
    f <= w1 WHEN s1 = '1' ELSE  
        w2 WHEN s2 = '1' ELSE  
        w3;  
END dataflow;
```

# Χρήση της δομής with select: 4-to-1 Multiplexer



Graphic symbol

operation table

```
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 ;
```

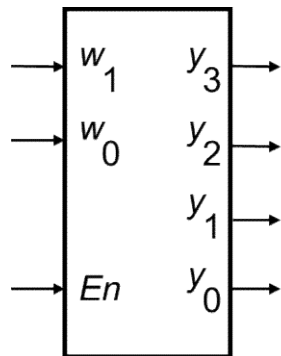
```
ARCHITECTURE dataflow OF mux4to1 IS
BEGIN
  WITH s SELECT
    f <= w0 WHEN "00",
        w1 WHEN "01",
        w2 WHEN "10",
        w3 WHEN OTHERS;
END dataflow ;
```

Ποια είναι τα others?

# 2-to-4 Decoder with enable

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

(a) Truth table



(b) Graphical symbol

```

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 ;
    
```

# Arithmetic Operators

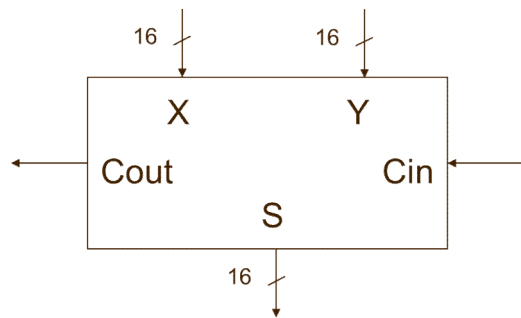
**To use basic arithmetic operations involving `std_logic_vectors` you need to include the following library packages:**

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

**You can use standard + and - 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;
```

# VHDL code for a 16-bit Unsigned Adder



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

```
ENTITY adder16 IS PORT (  
  Cin  : IN  STD_LOGIC ;  
  X, Y : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);  
  S    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);  
  Cout : OUT STD_LOGIC);  
END adder16 ;
```

```
ARCHITECTURE dataflow OF adder16 IS  
  SIGNAL Sum : STD_LOGIC_VECTOR(16 DOWNTO 0);  
BEGIN  
  Sum <= ('0' & X) + Y + Cin;  
  S <= Sum(15 DOWNTO 0);  
  Cout <= Sum(16);  
END dataflow;
```

# 4-bit Unsigned Number Comparator



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

ENTITY compare IS PORT (
  A, B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
  AeqB, AgtB, AltB: OUT STD_LOGIC);
END compare;

ARCHITECTURE dataflow OF compare IS
BEGIN
  AeqB <= '1' WHEN A = B ELSE '0';
  AgtB <= '1' WHEN A > B ELSE '0';
  AltB <= '1' WHEN A < B ELSE '0';
END dataflow;
```

# 4-bit Signed Number Comparator



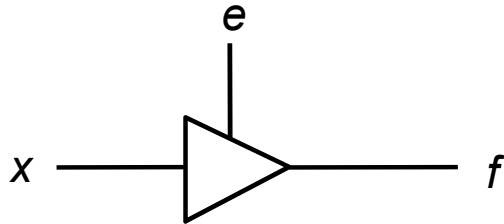
```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
USE ieee.std_logic_signed.all ;
```

```
ENTITY compare IS PORT (  
  A, B: IN STD_LOGIC_VECTOR(3 DOWNT0 0);  
  AeqB, AgtB, AltB: OUT STD_LOGIC);  
END compare;
```

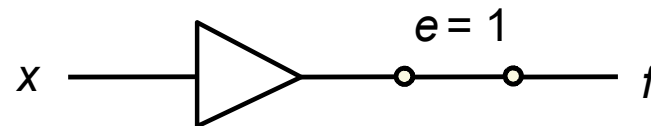
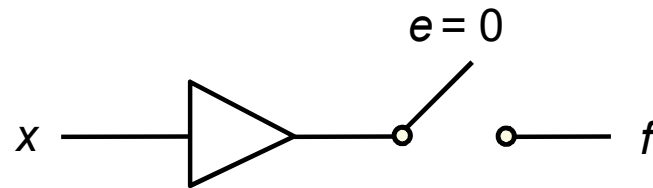
```
ARCHITECTURE dataflow OF compare IS
```

```
BEGIN  
  AeqB <= '1' WHEN A = B ELSE '0';  
  AgtB <= '1' WHEN A > B ELSE '0';  
  AltB <= '1' WHEN A < B ELSE '0';  
END dataflow;
```

# Tri-state Buffer



(a) A tri-state buffer

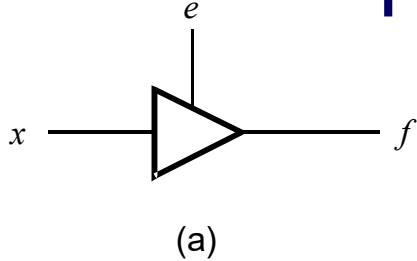


(b) Equivalent circuit

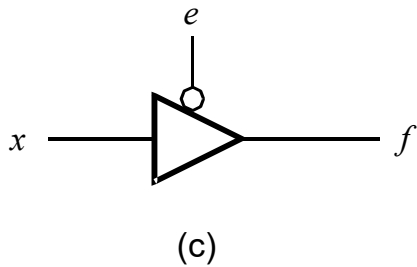
$e$	$x$	$f$
0	0	Z
0	1	Z
1	0	0
1	1	1

(c) Truth table

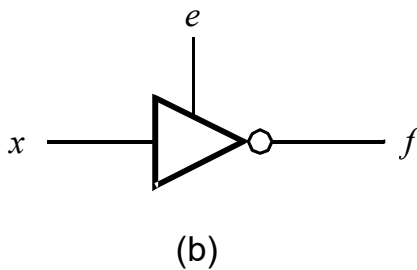
# Four types of Tri-state Buffers



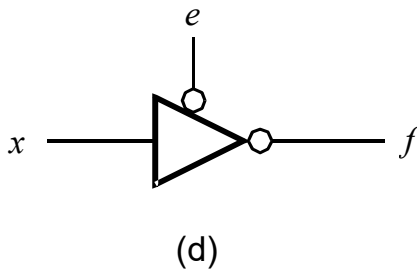
```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;
```



```
ENTITY tri_state IS PORT (  
    e: IN STD_LOGIC;  
    input: IN STD_LOGIC;  
    output: OUT STD_LOGIC);  
END tri_state;
```

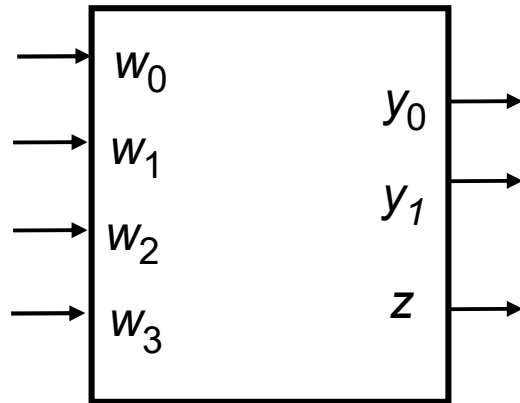


```
ARCHITECTURE dataflow OF tri_state IS  
BEGIN
```



```
    output <= input WHEN (e = '1')  
        ELSE 'Z';  
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

```

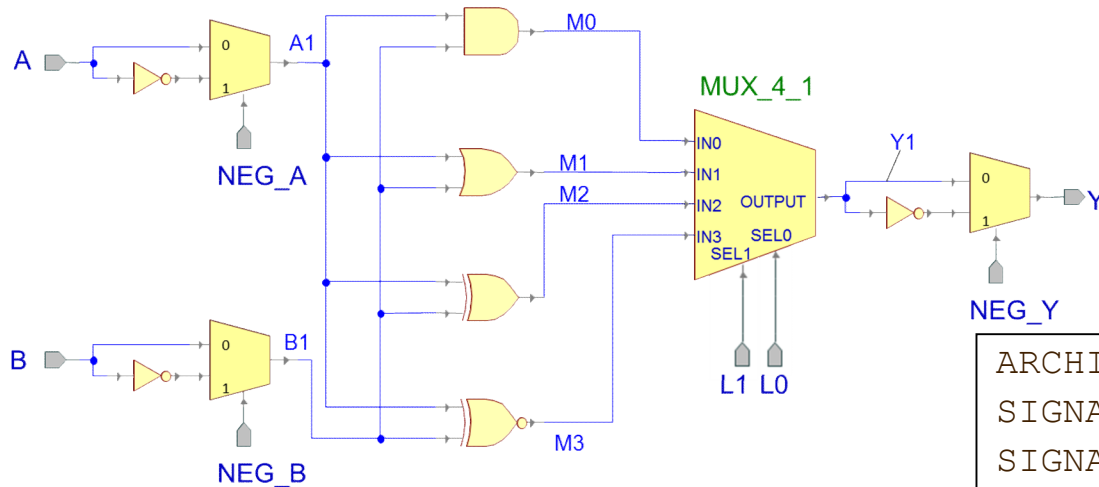
LIBRARY ieee ;
USE ieee.std_logic_1164.all;

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;

```

# Data-flow VHDL Example 2: MLU module



Ποιες εισόδους;  
 Ποιες εξόδους;  
 Ποια signals;

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

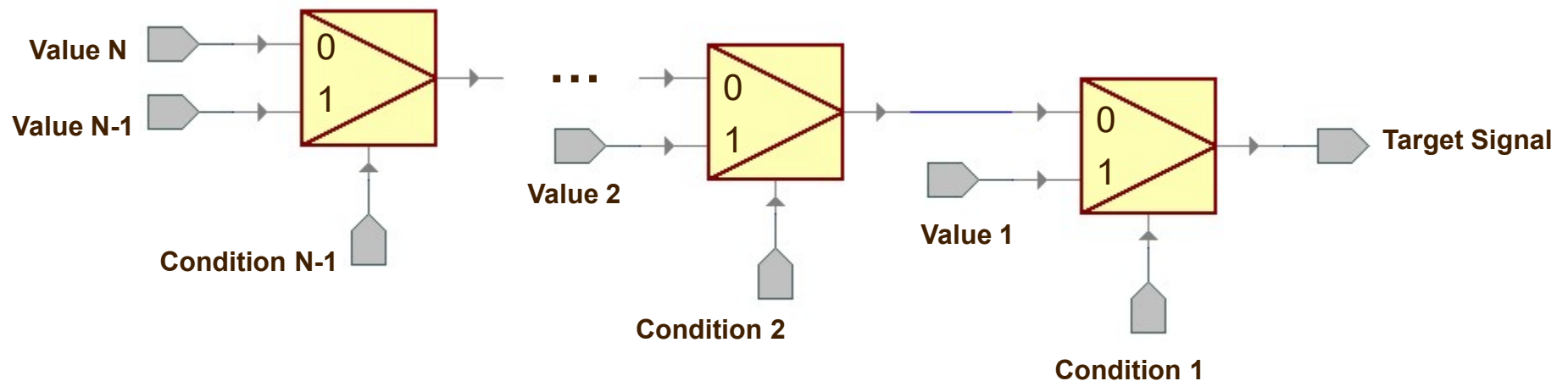
```

ARCHITECTURE mlu_dataflow OF mlu IS
    SIGNAL A1, B1, Y1: STD_LOGIC;
    SIGNAL M0, M1, M2, M3: STD_LOGIC;
    SIGNAL L: STD_LOGIC_VECTOR(1 DOWNTO 0);
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;
    M0 <= A1 AND  B1;
    M1 <= A1 OR   B1;
    M2 <= A1 XOR  B1;
    M3 <= 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 by Conditional and Selected Concurrent Signal Assignments

## *When - Else*

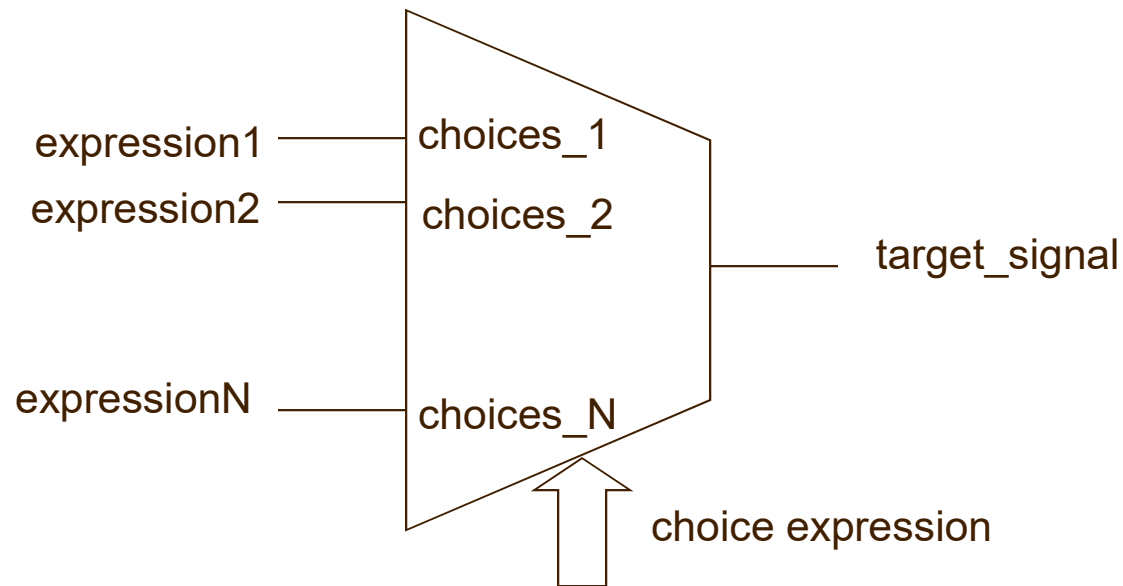
```
target_signal <= value1  when condition1 else  
                    value2  when condition2 else  
                    . . .  
                    valueN-1 when conditionN-1 else  
                    valueN;
```



# Logic Implied by Conditional and Selected Concurrent Signal Assignments

*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
```

## Example

```
WITH sel SELECT
```

```
y <= a WHEN "000",
```

```
    c WHEN "001" | "111",
```

```
    d WHEN OTHERS;
```