ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

# Πρόγραμμα Μεταπτυχιακών Σπουδών «Πληροφορική και Εφαρμογές»

## Αρχές Ψηφιακής Τεχνολογίας

Εισαγωγή στη VHDL

Γιάννης Βογιατζής

2018-2019

# VHDL

- VHDL: γλώσσα περιγραφής υλικού
  - **VHDL** : **V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) **H**ardware **D**escription **L**anguage

# Naming and Labeling (1)

- VHDL is case insensitive

  Example:

  Names or labels

  **`databus`**

  **`Databus`**

  **`DataBus`**

  **`DATABUS`**

  are all equivalent

# Naming and Labeling (2)

1. Τα ονόματα αρχίζουν από (a-z or A-Z)
2. Χρησιμοποιούνται χαρακτήρες (a-z or A-Z) ψηφία (0-9) και underscore (_)
3. Δε χρησιμοποιούνται punctuation ή reserved characters (!, ?, ., &, +, -, etc.)
4. Δε χρησιμοποιούνται δύο ή περισσότερα underscore characters (__)
5. Όλα τα ονόματα σε ένα entity πρέπει να είναι μοναδικά

# Free Format

- Η Vhdl είναι “free format” γλώσσα

  Example:

```
if (a=b) then
```

  *or*

```
if  (a=b)          then
```

  *or*

```
if (a =
b) then
```

  are all equivalent

# Σχόλια

- Με “double dash”, i.e., “--”
  - Μπορούν να μπουν οπουδήποτε στη γραμμή
  - Ότι είναι στην ίδια γραμμή είναι σχόλιο

Examples:

```
-- main subcircuit
Data_in <= Data_bus;     -- reading data from the input FIFO
```

# Example VHDL Code

- 3 sections to a piece of VHDL code
- File extension for a VHDL file is .vhd
- Name of the file should be the same as the entity name (nand_gate.vhd)

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

ENTITY nand_gate IS
     PORT(
          a    : IN BIT;
          b    : IN BIT;
          z    : OUT BIT);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
BEGIN
     z <= a NAND b;
END model;
```

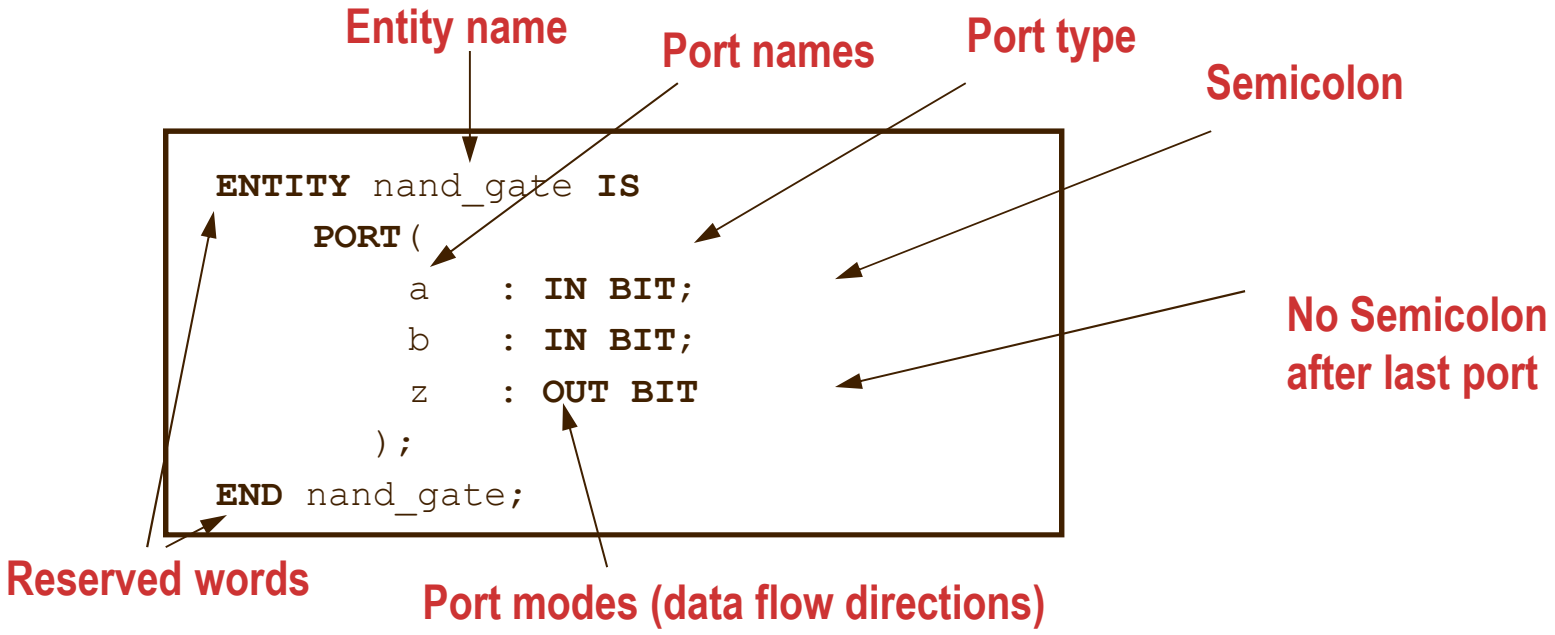
LIBRARY DECLARATION

ENTITY DECLARATION

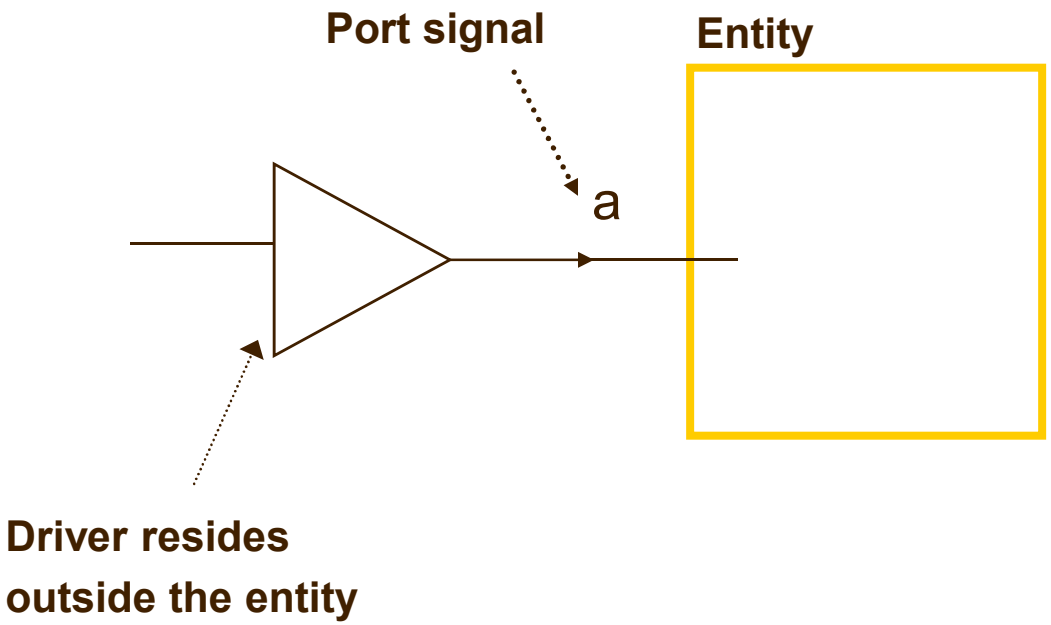
ARCHITECTURE BODY

# Entity Declaration

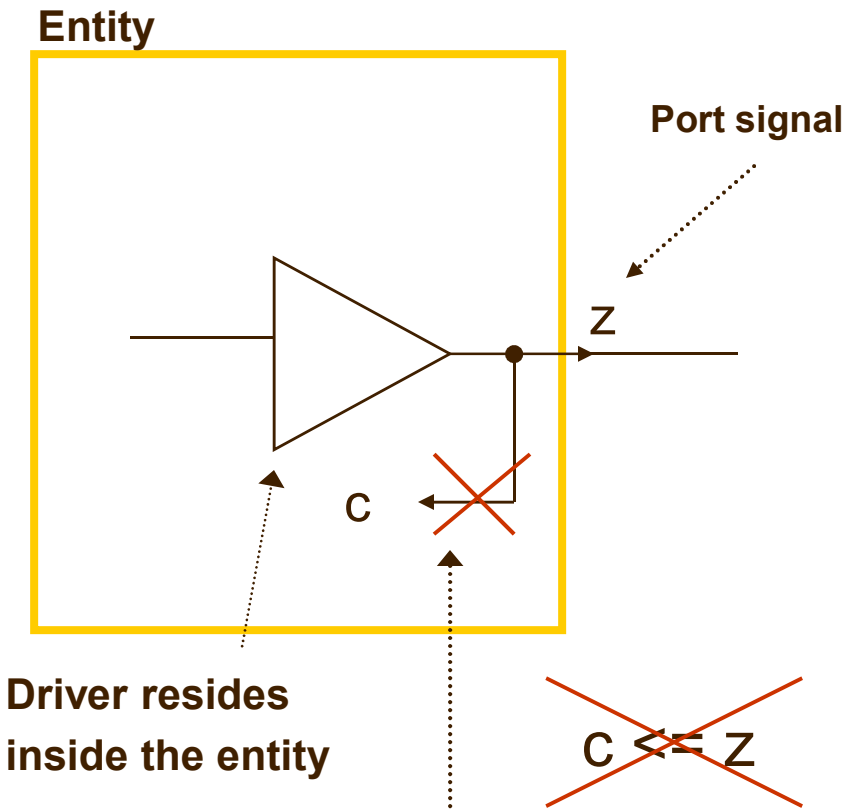
- Entity Declaration describes the interface of the component, i.e. input and output ports.



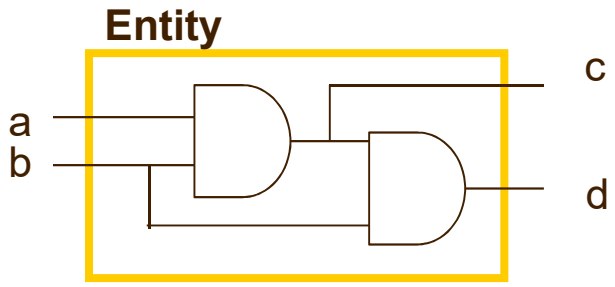
# Port Mode IN



# Port Mode OUT

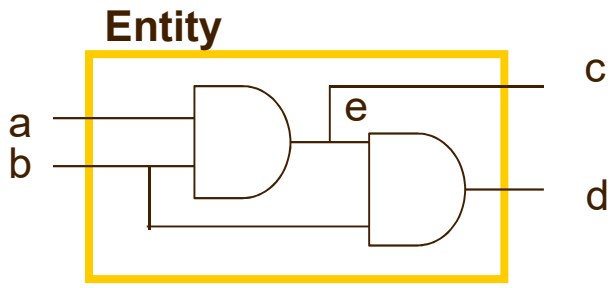


# Port Mode OUT




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY nand_gate IS PORT(
        a,b : IN BIT;
        c,d : OUT BIT);
END nand_gate;

ARCHITECTURE model OF nand_gate IS


BEGIN
  c <= a AND b;
  d <= c and b;
END model;
```

# Port Mode OUT (with extra signal)




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY nand_gate IS PORT(
         a,b : IN BIT;
         c,d : OUT BIT);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
signal e: BIT;

BEGIN
  e <= a AND b;
  c <= e;
  d <= e and b;
END model;
```

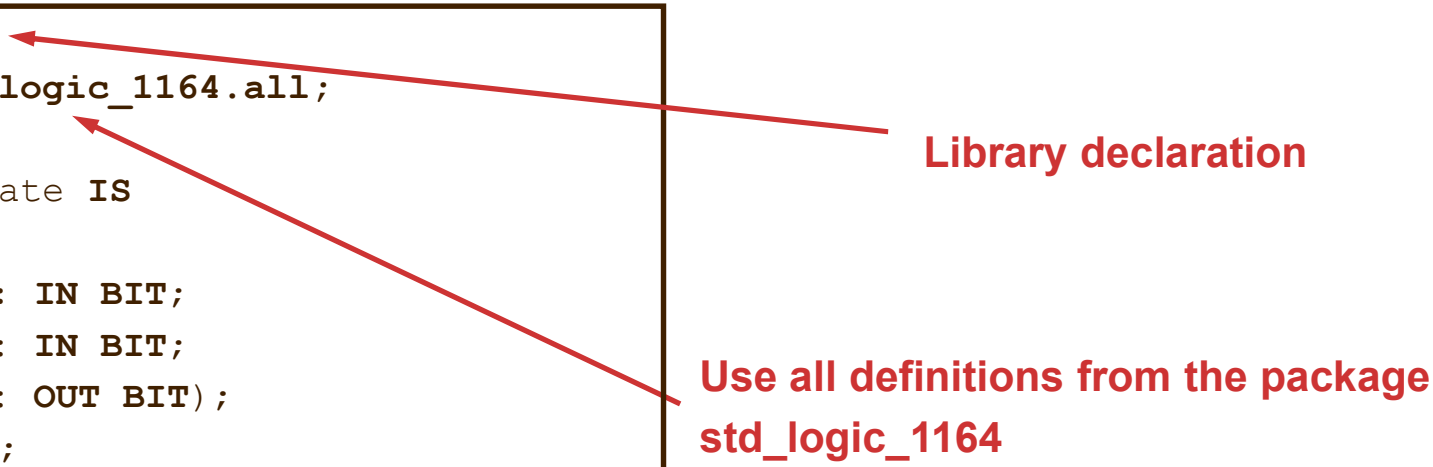
# *Architecture (Architecture body)*

- Describes an implementation of a design entity
- Architecture example:

```
ARCHITECTURE model OF nand_gate IS
BEGIN
     z <= a NAND b;
END model;
```

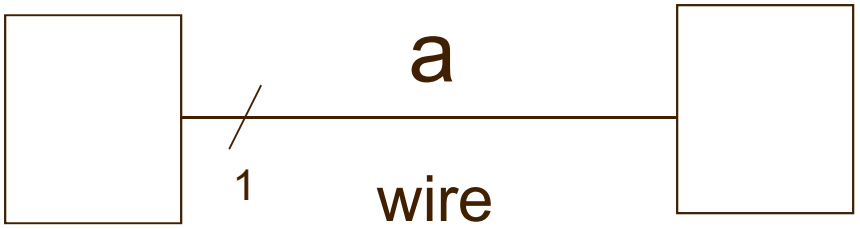
# Library Declarations

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

ENTITY nand_gate IS
     PORT(
          a    : IN BIT;
          b    : IN BIT;
          z    : OUT BIT);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
BEGIN
     z <= a NAND b;
END model;
```

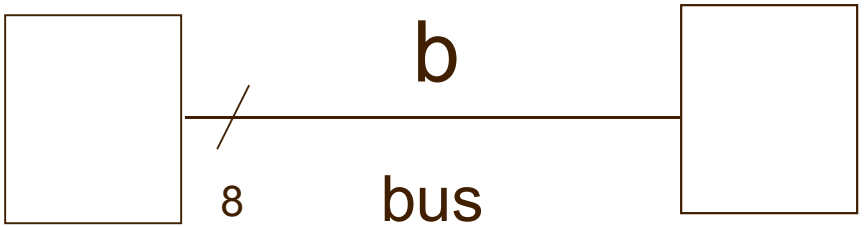


# Signals

SIGNAL a : BIT;




SIGNAL b : BIT_VECTOR(7 DOWNTO 0);



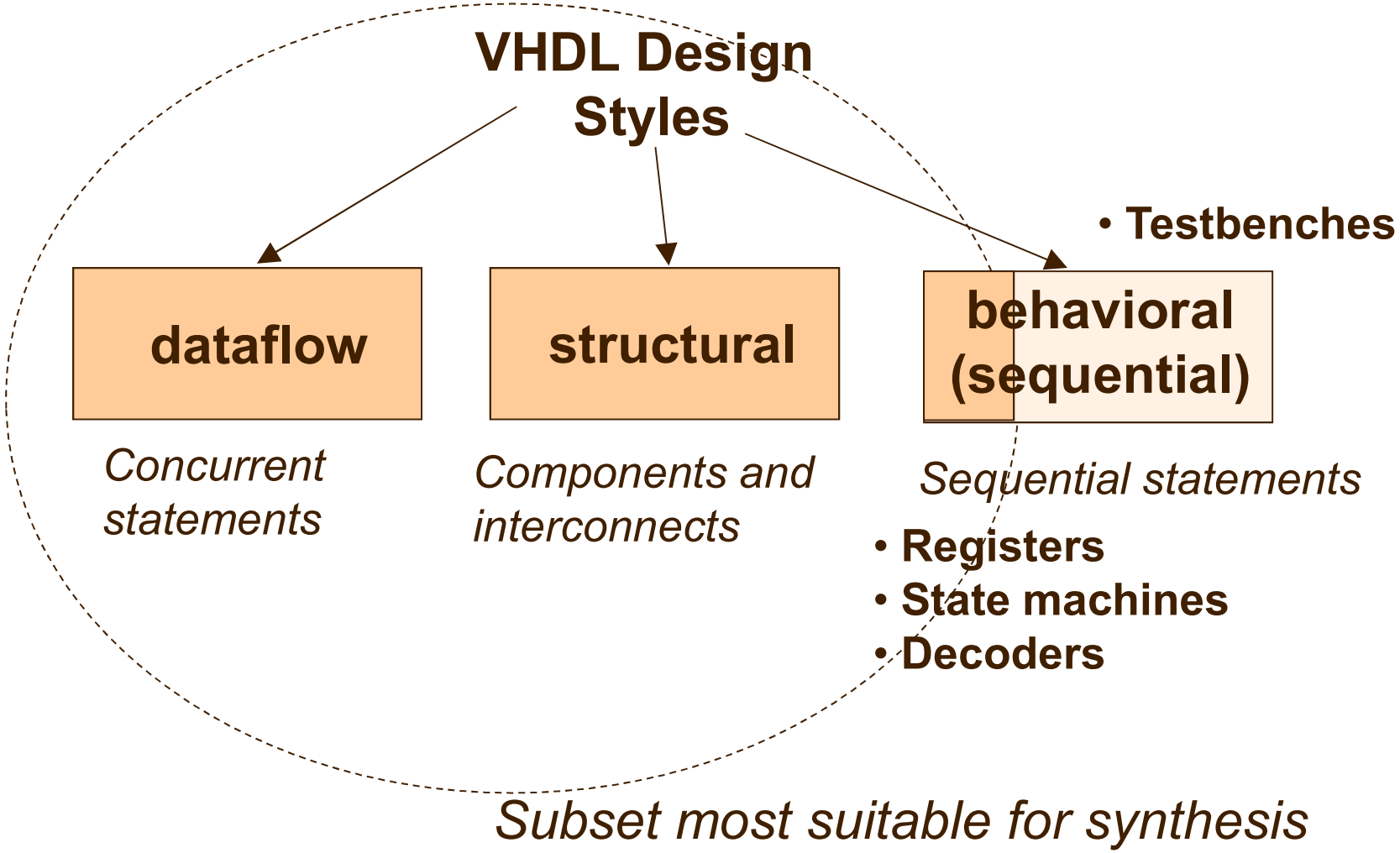
# Standard Logic Vector examples

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(3 DOWNTO 0);
SIGNAL c: BIT_VECTOR(3 DOWNTO 0);
SIGNAL d: BIT_VECTOR(15 DOWNTO 0);
SIGNAL e: BIT_VECTOR(8 DOWNTO 0);
                                                    …….
a <= '1';
b <= "0000";      -- Binary base assumed by default
c <= B"0000";     -- Binary base explicitly specified
d <= X"AF67";     -- Hexadecimal base
e <= O"723";      -- Octal base
```

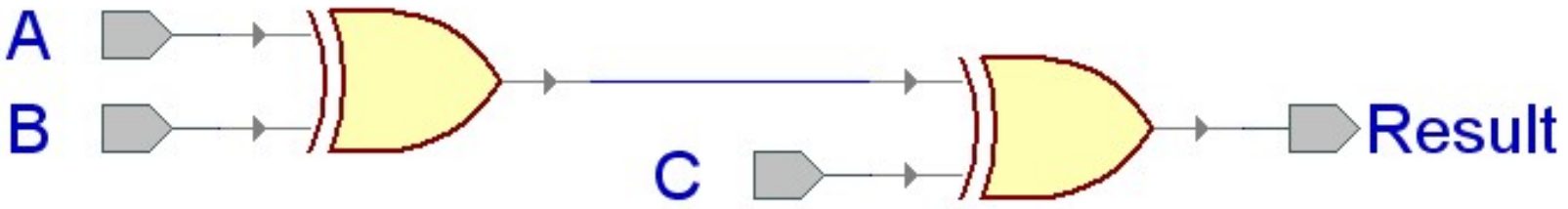
# Vectors and Concatenation

```
SIGNAL a: BIT_VECTOR(3 DOWNTO 0);
SIGNAL b: BIT_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: BIT_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"
```

# VHDL Design Styles



# xor3 Example

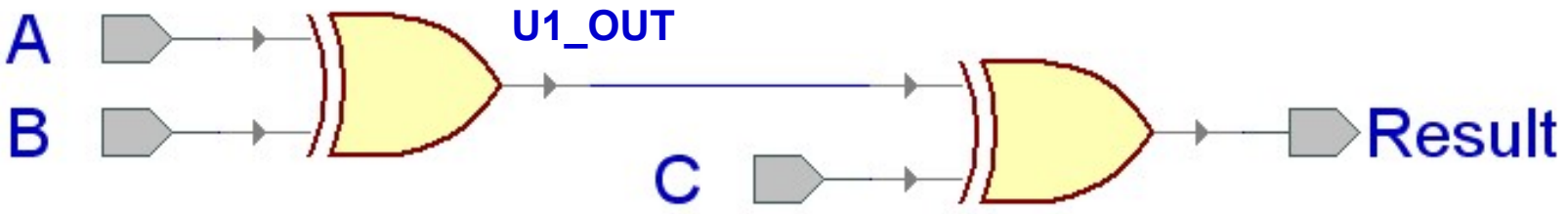


# Entity xor3_gate

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

ENTITY xor3_gate IS PORT(
      A : IN BIT;
      B : IN BIT;
      C : IN BIT;
      Result : OUT BIT);
end xor3_gate;
```

# Dataflow Architecture (xor3_gate)

```
ARCHITECTURE dataflow OF xor3_gate IS
SIGNAL U1_OUT: BIT;
BEGIN
     U1_OUT <= A XOR B;
     Result  <= U1_OUT XOR C;
END dataflow;
```



# Dataflow Description

- Περιγράφει πώς μετακινούνται τα data στο σύστημα.
  - Σειρά από concurrent statements
  - Χρήσιμο όταν μια σειρά από Boolean συναρτήσεις αναπαριστούν τη λογική →
    - χρησιμοποιείται για απλή συνδυαστική λογική
  - Ο Dataflow κώδικας ονομάζεται και “concurrent” κώδικας
- Concurrent statements: αξιολογούνται ταυτόχρονα
- Έτσι, η σειρά δεν παίζει ρόλο

This …

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

Is the same as this …

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

# Structural Description

- Structural design είναι η πιο κοντινή περιγραφή στο schematic capture.
- Τα Components διασυνδέονται με ιεραρχικό τρόπο.
- Μπορεί να διασυνδέει απλές πύλες ή πολύπλοκες μονάδες.
- Χρήσιμο όταν περιγράφουμε μονάδες που αποτελούνται από υπομονάδες.

# Structural architecture - xor2

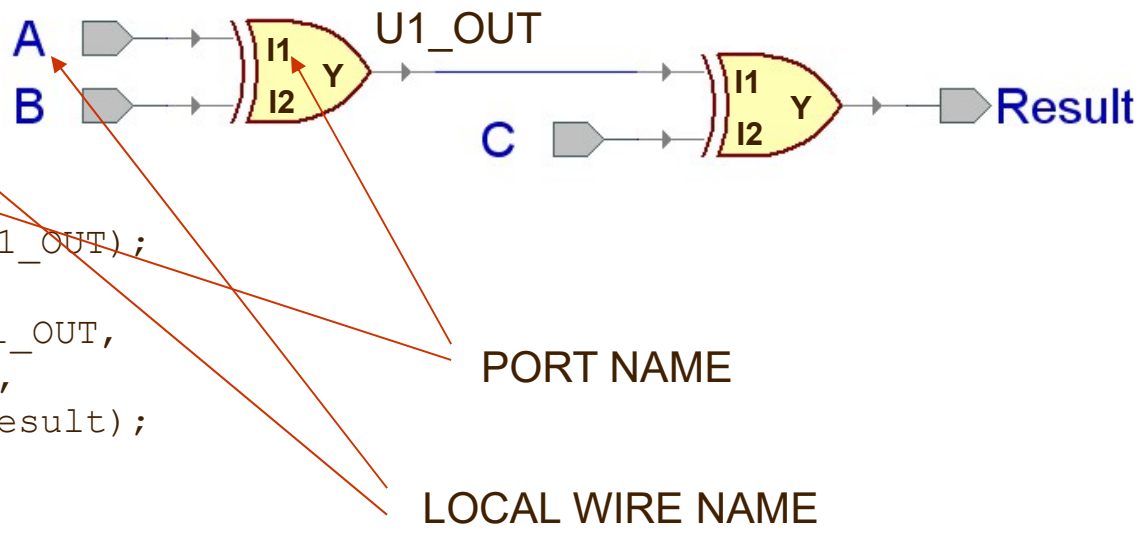
xor2.vhd

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

ENTITY xor2 IS
     PORT(
          I1    : IN BIT;
          I2    : IN BIT;
          Y     : OUT BIT);
END xor2;

ARCHITECTURE dataflow OF xor2 IS
BEGIN
     Y <= I1 xor I2;
END dataflow;
```

# Structural Architecture

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY xor2 IS PORT(
          I1    : IN BIT;
          I2    : IN BIT;
          Y     : OUT BIT);
END xor2;
ARCHITECTURE dataflow OF xor2 IS
BEGIN
     Y <= I1 xor I2;
END dataflow;
```

```
ARCHITECTURE structural OF xor3_gate IS
SIGNAL U1_OUT: BIT;

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

BEGIN
   U1: xor2 PORT MAP (I1 => A,
                      I2 => B,
                      Y  => U1_OUT);

   U2: xor2 PORT MAP (I1 => U1_OUT,
                      I2 => C,
                      Y  => Result);

END structural;
```



# Behavioral Architecture (xor3 gate)

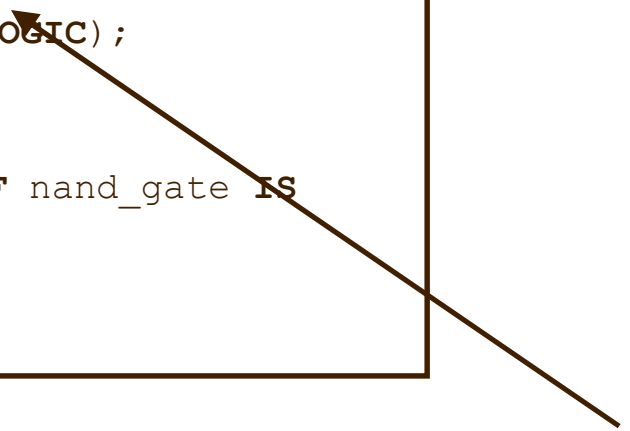
```
ARCHITECTURE behavioral OF xor3 IS
BEGIN
xor3_behave: PROCESS (A, B, C)
BEGIN
  IF ((A XOR B XOR C) = '1') THEN
     Result <= '1';
  ELSE
     Result <= '0';
  END IF;
END PROCESS xor3_behave;
END behavioral;
```

# Behavioral Description

- Περιγράφει τι συμβαίνει μεταξύ των εισόδων και εξόδων σα να είναι μαύρο κουτί
- Χρησιμοποιεί PROCESS statements.

# STD_LOGIC

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

ENTITY nand_gate IS
     PORT(
          a    : IN STD_LOGIC;
          b    : IN STD_LOGIC;
          z    : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE dataflow OF nand_gate IS
BEGIN
     z <= a NAND b;
END dataflow;
```

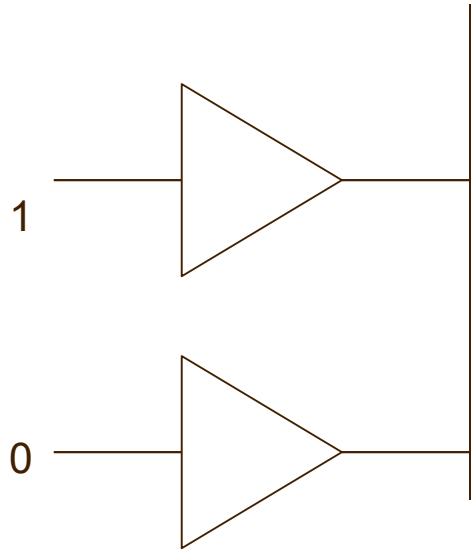


What is **STD_LOGIC**?

# BIT versus STD_LOGIC

- BIT type can only have a value of ‘0’ or ‘1’
- STD_LOGIC can have eight values
  - ‘0’,’1’,’X’,’Z’,’W’,’L’,’H’,’-’
  - Useful mainly for simulation
  - ‘0’,’1’, and ‘Z’ are synthesizable
- What is ‘X’, ‘Z’;

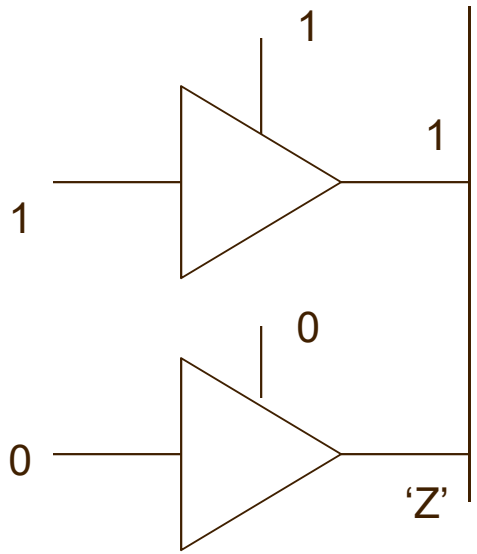
# STD_LOGIC: ‘X’




**‘X’**

**Contention on the bus**

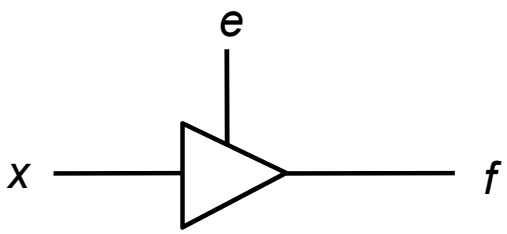
# STD_LOGIC: ‘Z’




Tristate buffer

Πώς φτιάχνω ένα Tristate buffer??

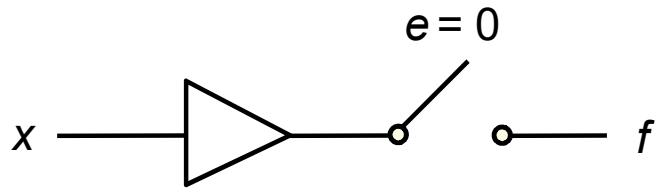
# Tri-state Buffer

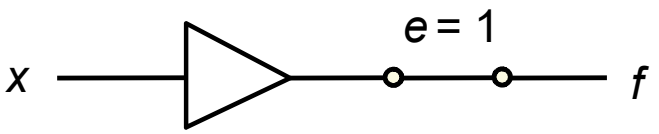



(a) A tri-state buffer

| e | x | f |
|---|---|---|
| 0 | 0 | Z |
| 0 | 1 | Z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

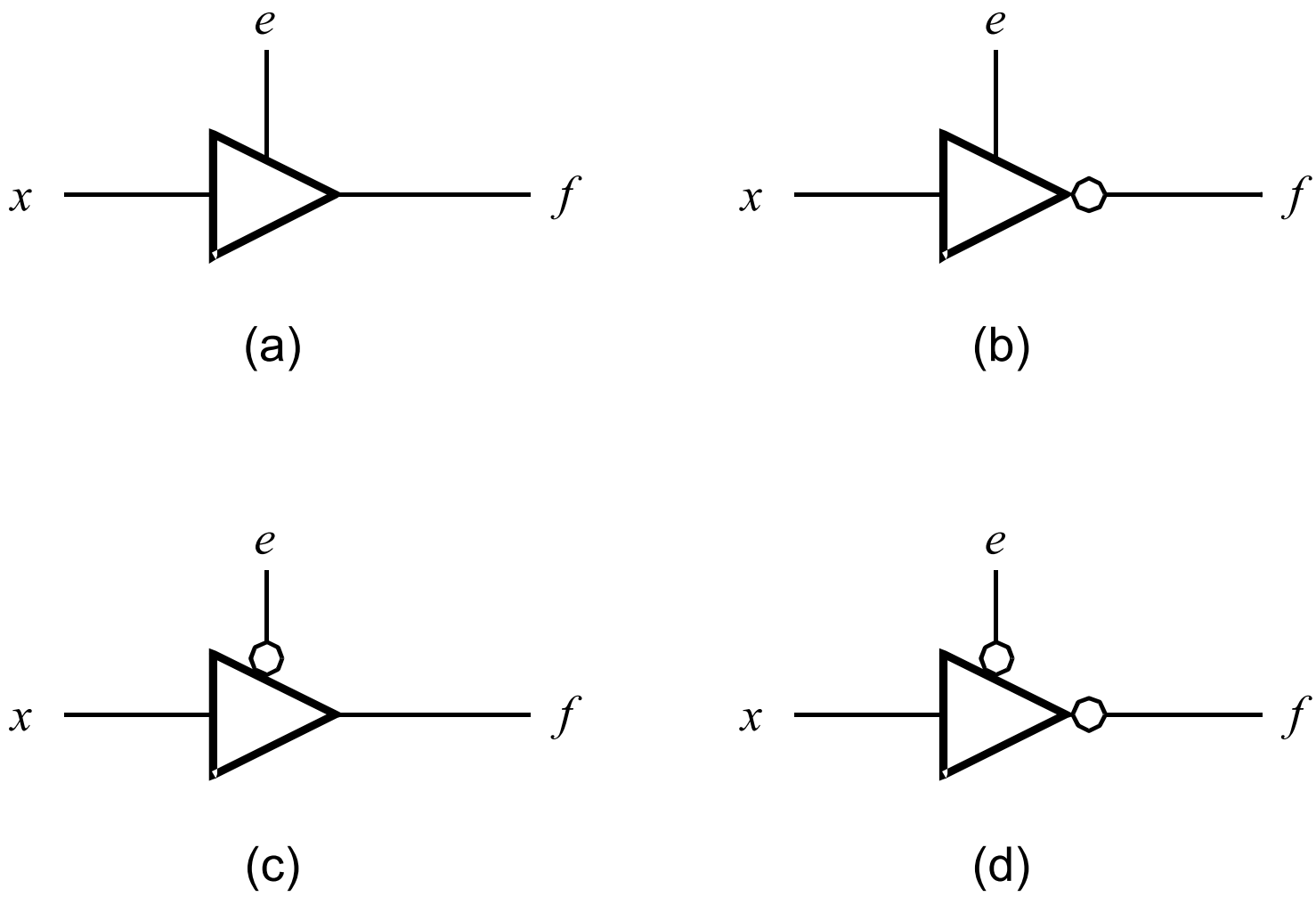
(c) Truth table



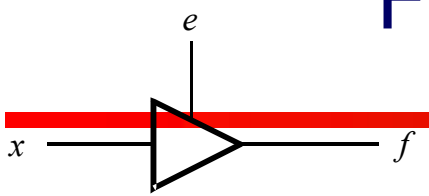




(b) Equivalent circuit

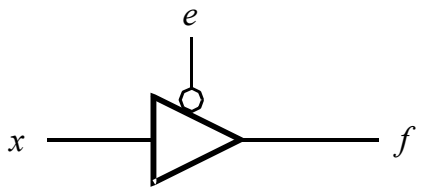
# Four types of Tri-state Buffers



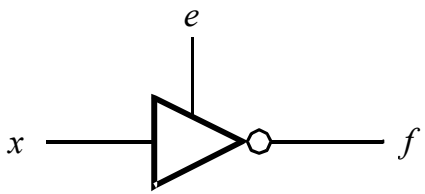
# Four types of Tri-state Buffers



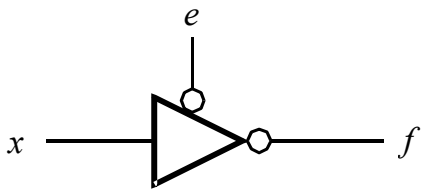

(a)




(c)




(b)




(d)

```
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;
```