# Διάχυτα και ενσωματωμένα συστήματα

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Γιάννης Βογιατζής

## Εισαγωγή στη VHDL

# Genesis of VHDL

**Κατάσταση στη δεκαετία του ‘80**

- Πολλαπλές μέθοδοι design entry methods και γλώσσες περιγραφής υλικού
- Περιορισμένη ή μηδενική φορητότητα στα σχέδια μεταξύ CAD tools από διαφορετικούς vendors
- Στόχος: μείωση του χρόνου από design concept μέχρι το implementation από τους 18 μήνες στους 6 μήνες

# 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 χρησιμοποιουνται punctuation ή reserved characters (!, ?, ., &, +, -, etc.)
4. Δε χρησιμοποιούνται δύο ή περισσότερα underscore characters (__)
5. Όλα τα ονόματα σε ένα entity πρέπει να είναι μοναδικά

# Valid or invalid?

7segment_display

A87372477424

Adder/Subtractor

/reset

And_or_gate

AND__OR__NOT

Kogge-Stone-Adder

Ripple&Carry_Adder

My adder

# Free Format

- Η Vhdl είναι μια “free format” γλώσσα

  Example:

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

  *or*

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

  *or*

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

  are all equivalent

# Σχόλια

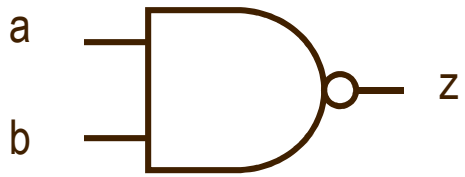
- Με “double dash”, i.e., “--”
  - Μπορούν να μπουν οπουδήποτε στη γραμμή
  - Ότι είναι στην ίδια γραμμή είναι σχόλιο

Examples:

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

# Design Entity

## Example: NAND Gate




| a | b | z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Example VHDL Code

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

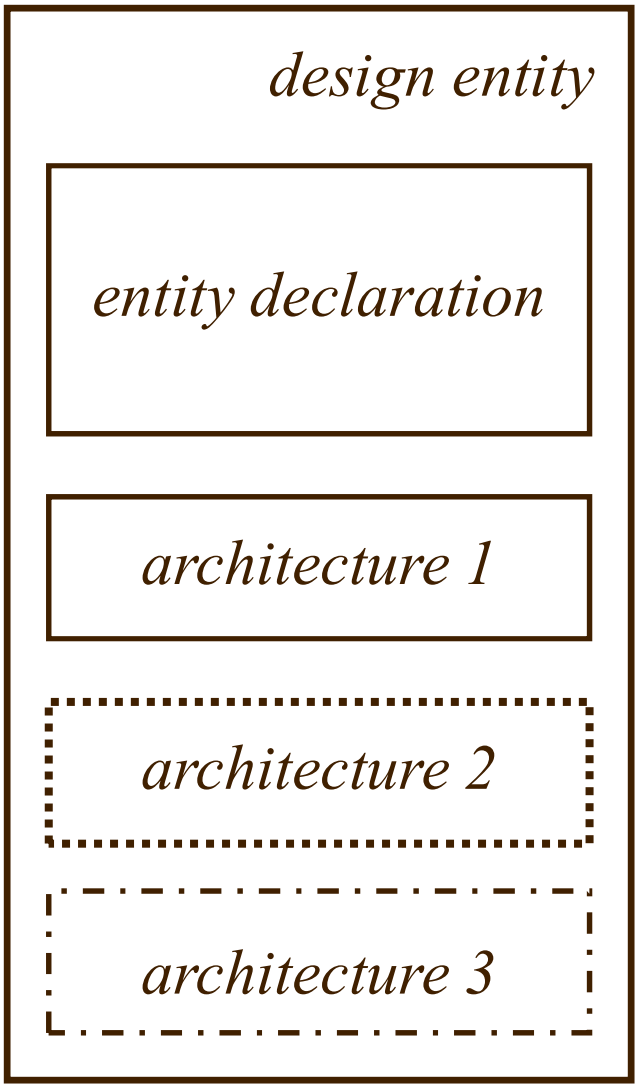
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

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

LIBRARY DECLARATION

ENTITY DECLARATION

ARCHITECTURE BODY

# Design Entity

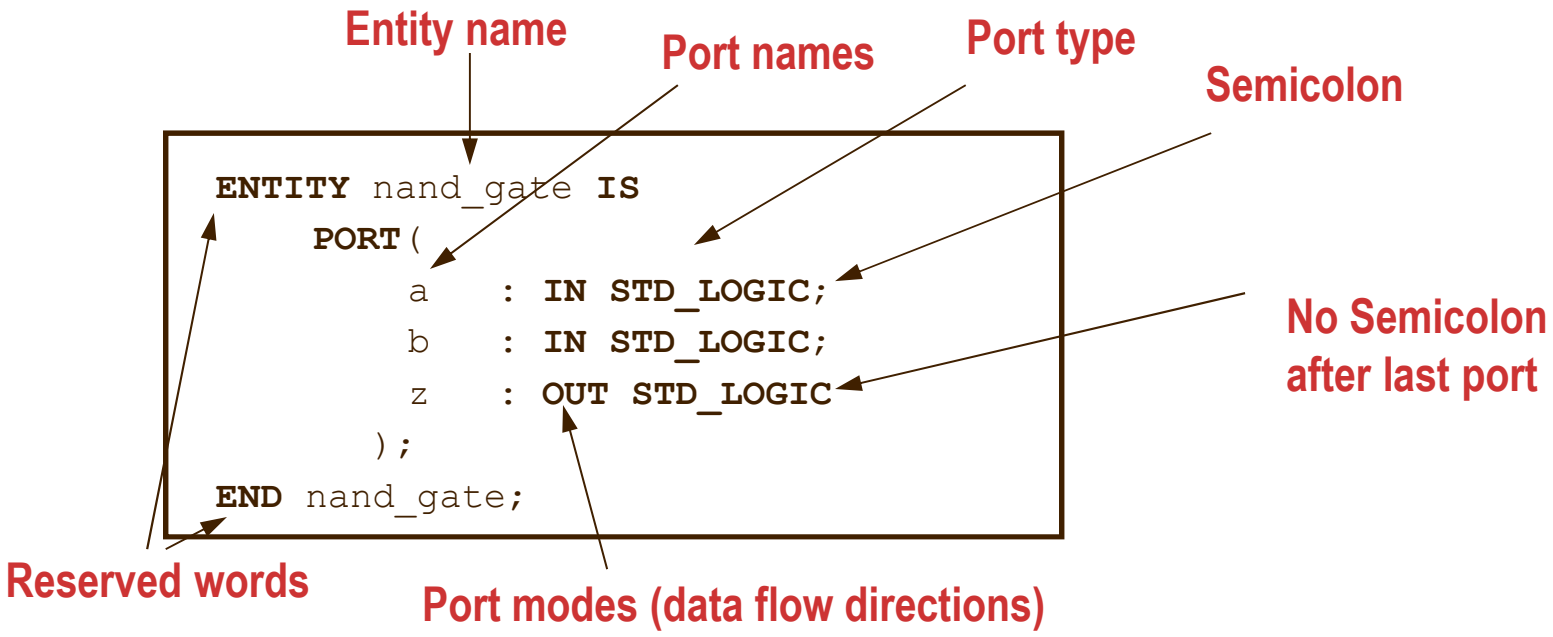



*Design Entity* - most basic building block of a design.

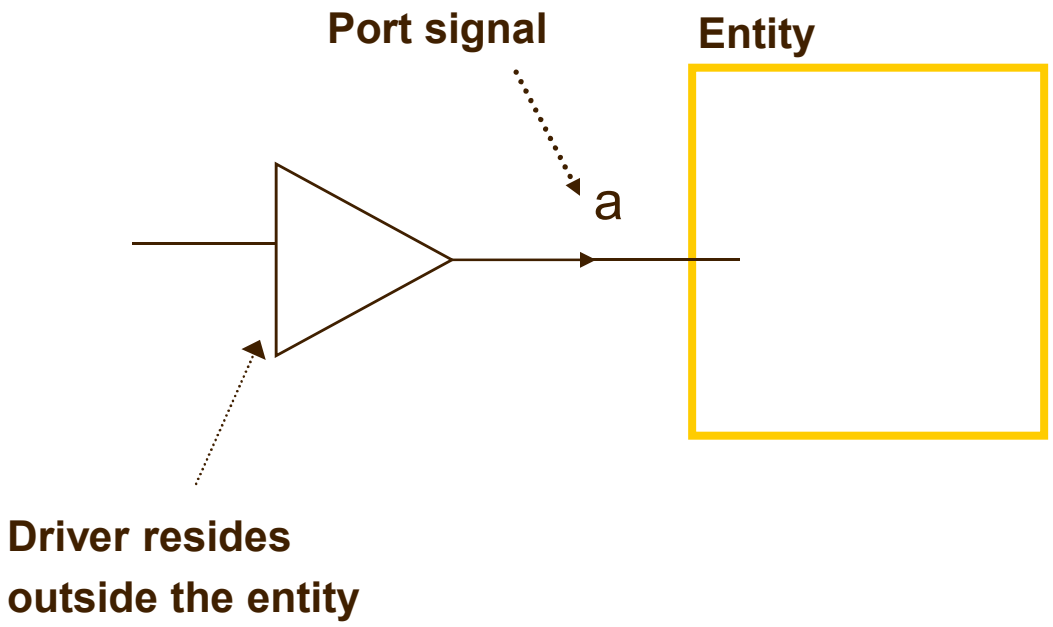
One *entity* can have many different *architectures.*

# Entity Declaration

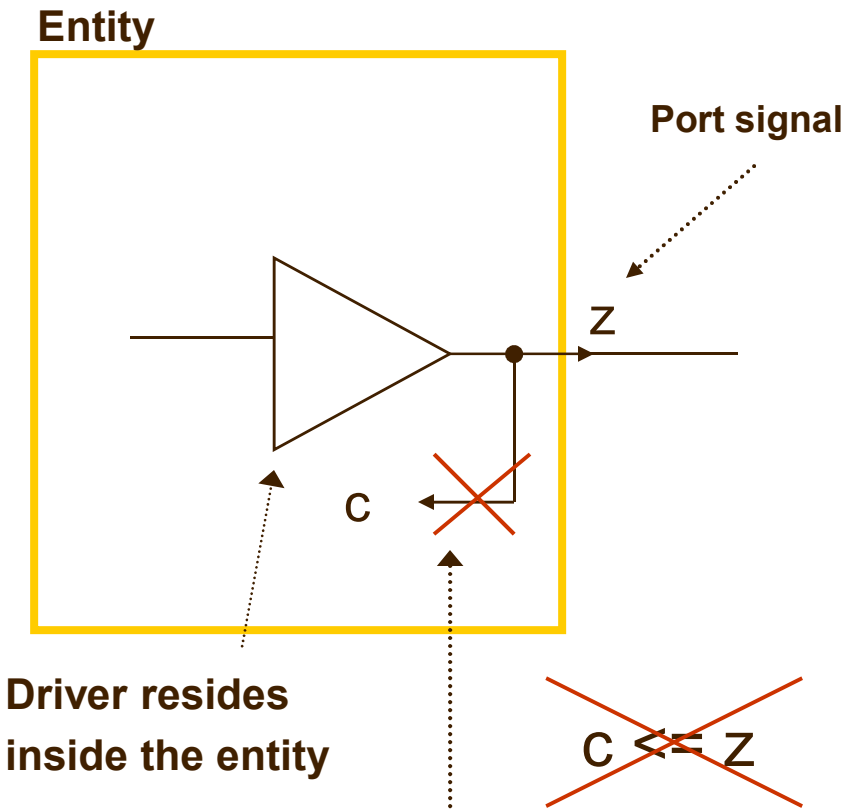
- Entity Declaration describes the interface of the component, i.e. input and output ports.



# Port Mode IN

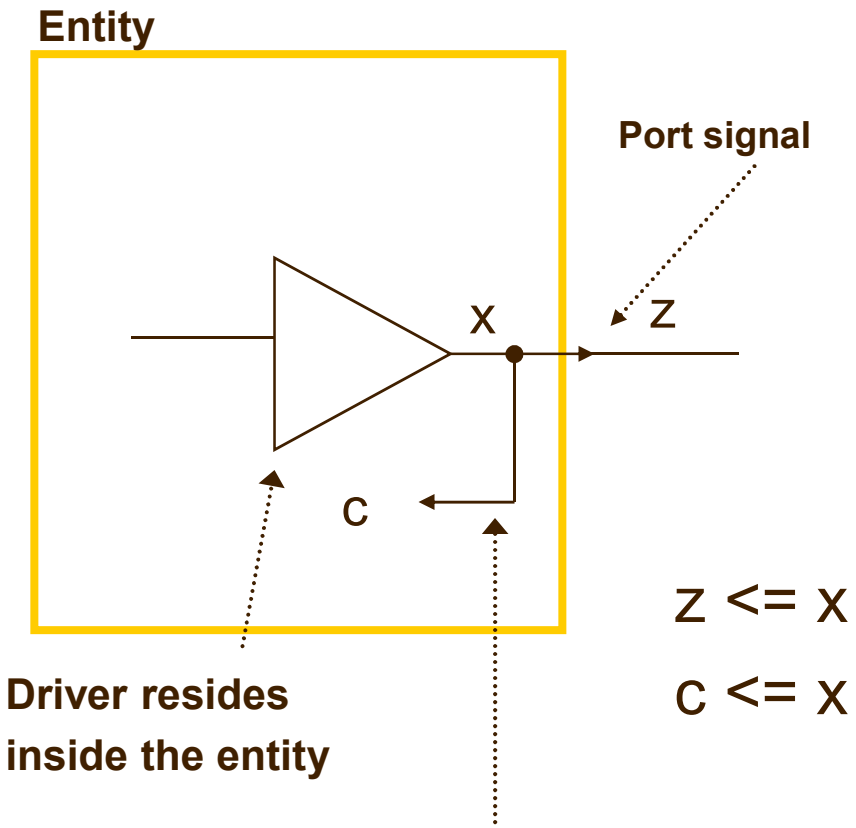


# Port Mode OUT



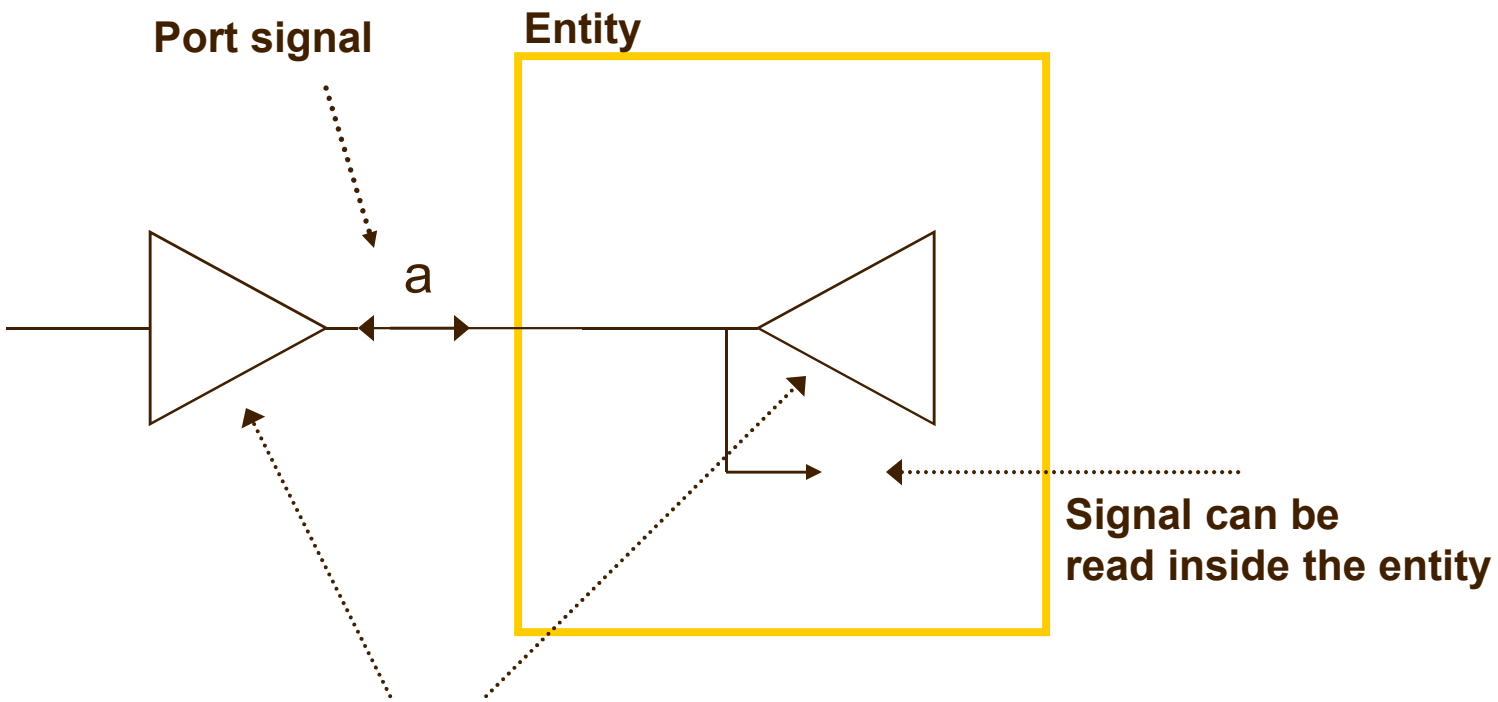

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY nand_gate IS PORT(
         a,b : IN STD_LOGIC;
         c,d : OUT STD_LOGIC);
END nand_gate;

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 STD_LOGIC;
         c,d : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
signal e: STD_LOgic;

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

# Port Mode INOUT



# *Architecture (Architecture body)*

- Describes an implementation of a design entity

```
ARCHITECTURE architecture_name OF entity_name IS
   [ declarations ]
BEGIN
    code
END architecture_name;
```

- Architecture example:

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

# Entity Declaration & Architecture

nand_gate.vhd

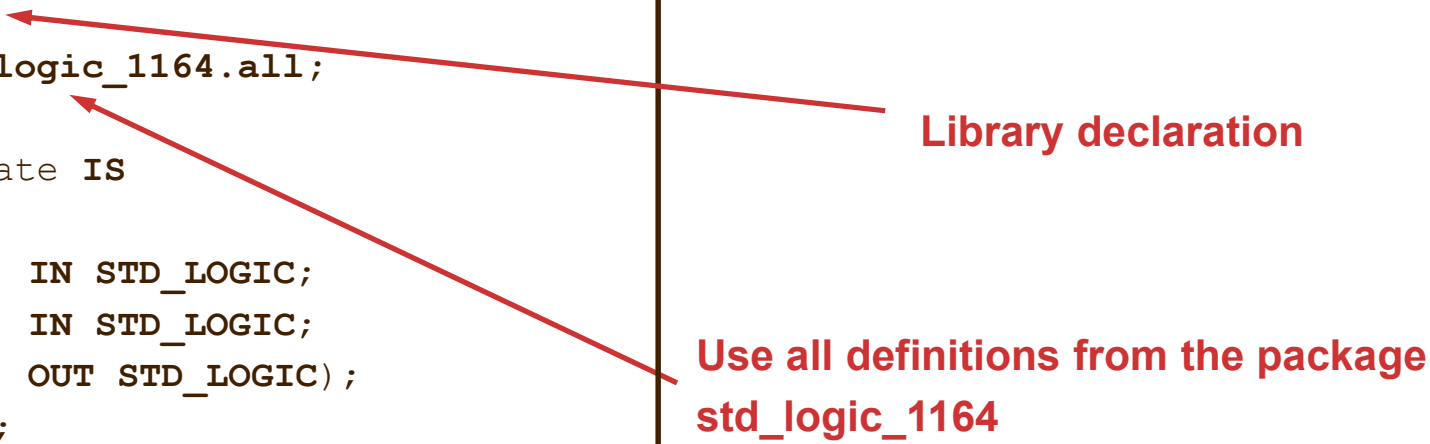
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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;
```

# Library declarations

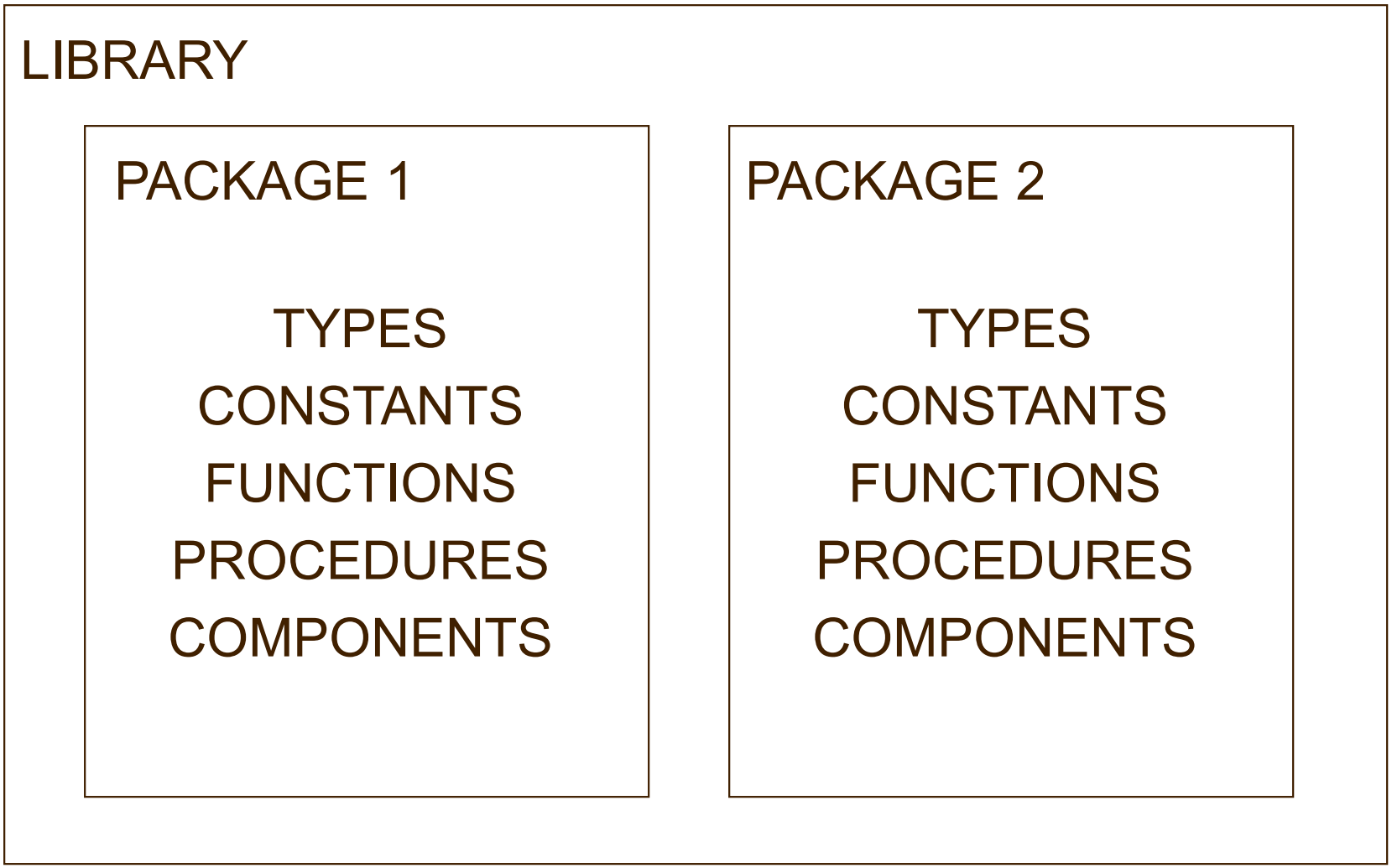
**LIBRARY library_name;**

**USE library_name.package_name.package_parts;**

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



# Parts of a library



# Libraries

- ieee

  Specifies multi-level logic system, including STD_LOGIC, and STD_LOGIC_VECTOR data types

Need to be explicitly declared

- std

  Specifies pre-defined data types (BIT, BOOLEAN, INTEGER, REAL, SIGNED, UNSIGNED, etc.), arithmetic operations, basic type conversion functions, basic text i/o functions, etc.

- work

  Holds current designs after compilation

Visible by default

# Bit vs STD_LOGIC

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

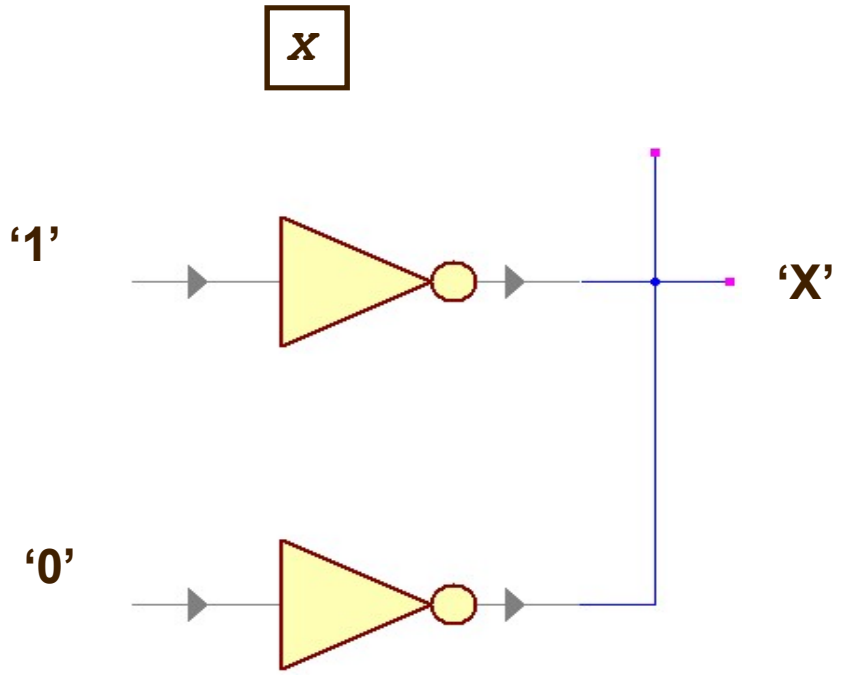
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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**?

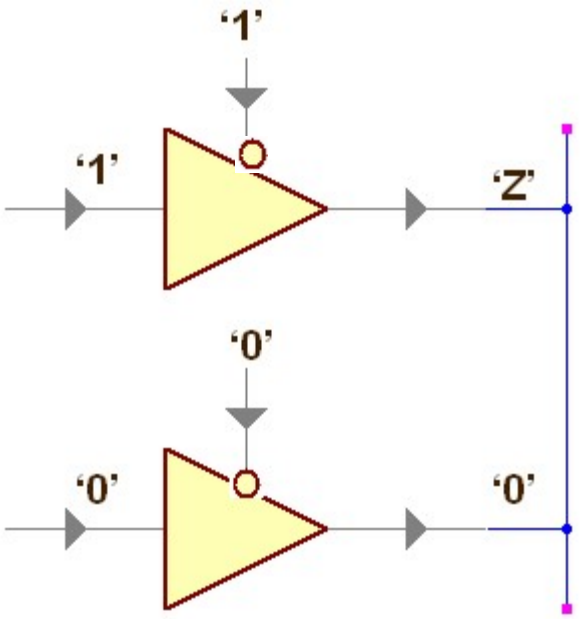
# BIT versus STD_LOGIC

- BIT type can 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’ vs ‘Z’

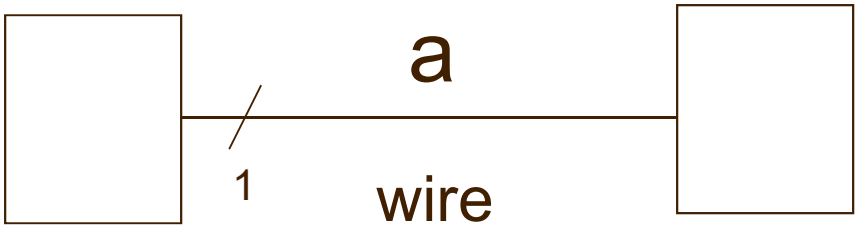



Contention on the bus

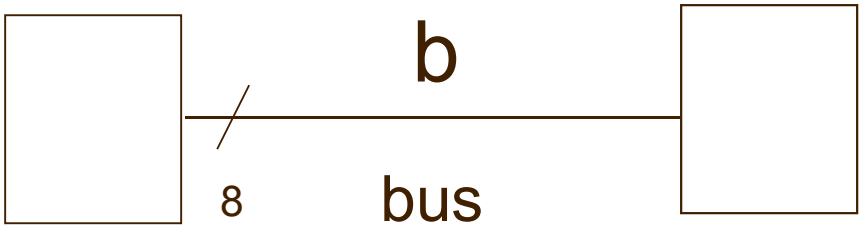


# Signals

SIGNAL a : STD_LOGIC;




SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);



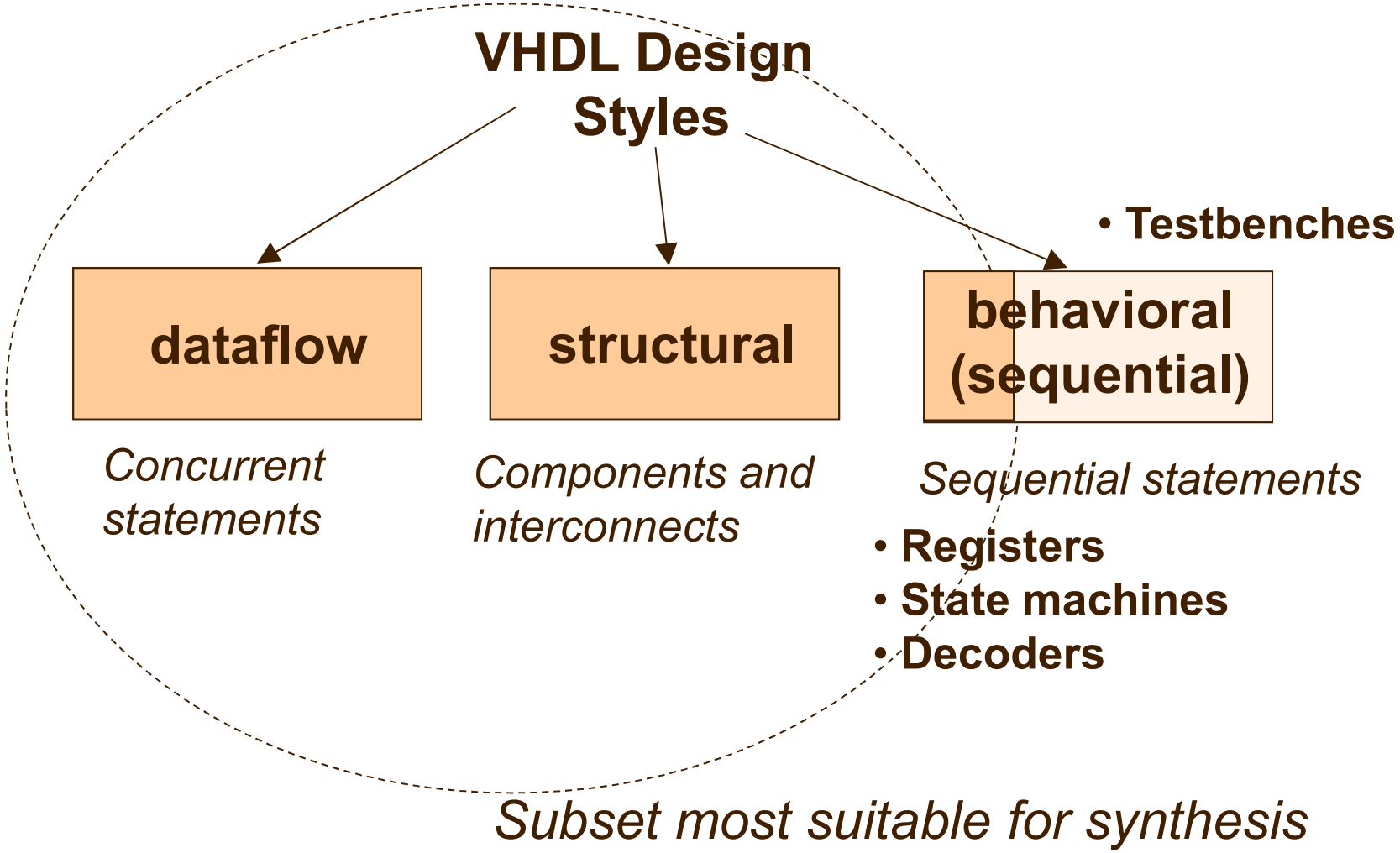
# Standard Logic Vector examples

```
SIGNAL a: STD_LOGIC;
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL d: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL e: STD_LOGIC_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

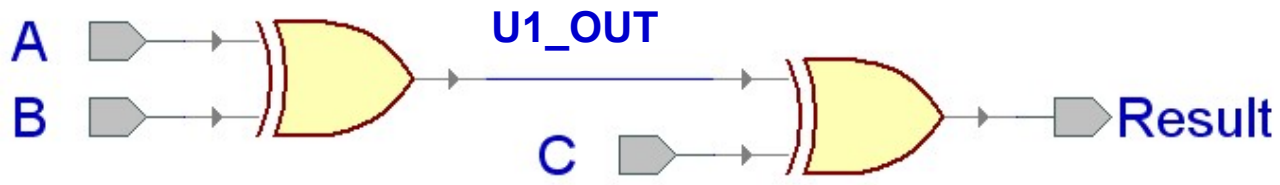
```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

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"
```

# VHDL Design Styles



# xor3 Example

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY xor3_gate IS PORT(
        A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        C : IN STD_LOGIC;
        Result : OUT STD_LOGIC);
end xor3_gate;
ARCHITECTURE df OF xor3_gate IS
SIGNAL U1_OUT: STD_LOGIC;
BEGIN
       U1_OUT <= A XOR B;
       Result  <= U1_OUT XOR C;
END dataflow;
```



# Dataflow Description

- Describes how data moves through the system and the various processing steps.
  - Dataflow uses series of concurrent statements to realize logic.
  - Dataflow is most useful style when series of Boolean equations can represent a logic → used to implement simple combinational logic
  - Dataflow code also called “concurrent” code
- Concurrent statements are evaluated at the same time; thus, the order of these statements doesn’t matter

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;
```

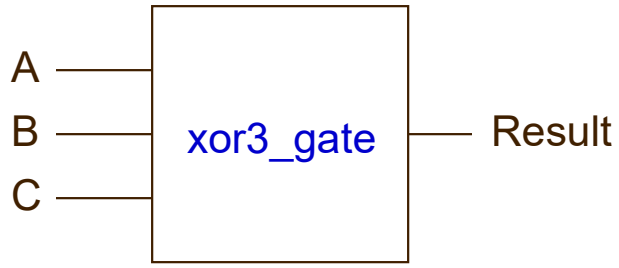
# Structural architecture - xor2

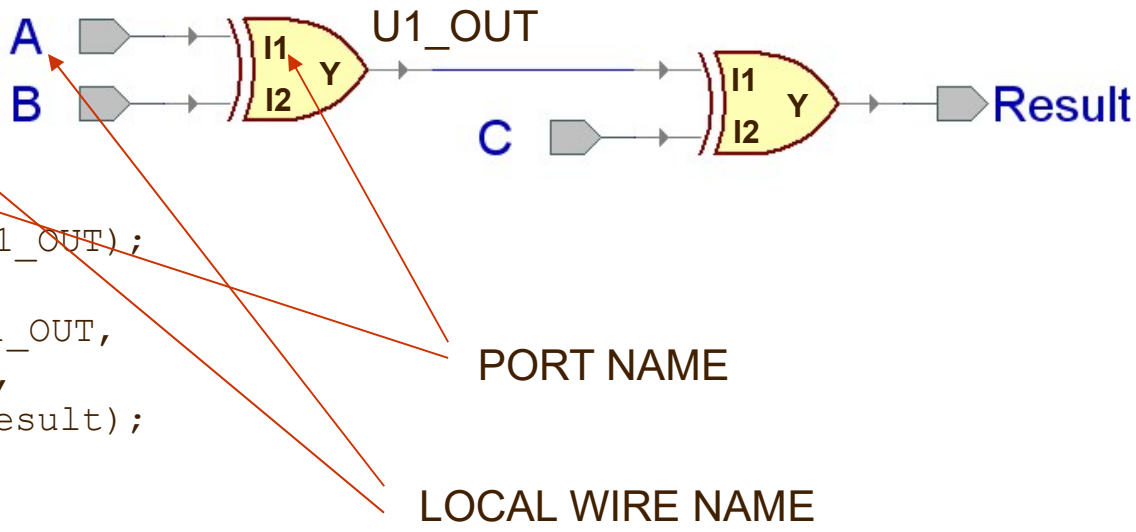
xor2.vhd

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

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

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

# Structural Architecture in VHDL 87

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

COMPONENT xor2
     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);

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

END structural;
```



# Structural Description

- Structural design is the simplest to understand. This style is the closest to schematic capture 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.

# 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

- 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.*