Εισαγωγή στην VHDL for Synthesis

G.Kornaros

Resources

- Volnei A. Pedroni, *Circuit Design with VHDL*

  Chapter 1, Introduction

  Chapter 2, Code Structure

  Chapter 3.1, Pre-Defined Data Types

- Sundar Rajan, *Essential VHDL: RTL Synthesis Done Right*

  Chapter 1, VHDL Fundamentals

  Chapter 2, Getting Your First Design Done

  (see errata at http://www.vahana.com/bugs.htm)

Brief History of VHDL

VHDL

- VHDL is a language for describing digital hardware used by industry worldwide

✓**VHDL** is an acronym for **V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) **H**ardware **D**escription **L**anguage

Genesis of VHDL

State of art circa 1980

- ✓Multiple design entry methods and hardware description languages in use
- ✓No or limited portability of designs between CAD tools from different vendors
- ✓Objective: shortening the time from a design concept to implementation from 18 months to 6 months

A Brief History of VHDL

- June 1981: Woods Hole Workshop
- July 1983: contract awarded to develop VHDL
  - ✓Intermetrics
  - ✓IBM
  - ✓Texas Instruments
- August 1985: VHDL Version 7.2 released
- December 1987:

  VHDL became IEEE Standard 1076-1987 and in 1988 an ANSI standard

Three versions of VHDL

- VHDL-87
- VHDL-93
- VHDL-01

Verilog

Verilog

- ✓ Essentially identical in function to VHDL
  - o No generate statement
- ✓ Simpler and syntactically different
  - o **C**-like

- ✓ Gateway Design Automation Co., 1983
- ✓ Early *de facto* standard for ASIC programming
- ✓ Open Verilog International Standard

- ✓ Programming language interface to allow connection to non-Verilog code

VHDL vs. Verilog

Government Developed	Commercially Developed
Ada based	C based
Strongly Type Cast	Mildly Type Cast
Difficult to learn	Easier to Learn
More Powerful	Less Powerful

Examples

- **VHDL Example:**

```
process (clk, rstn)
begin
    if (rstn = '0') then
        q <= '0';
    elseif (clk'event and clk = '1') then
        q <= a + b;
    end if;
end process;
```

- **Verilog Example:**

```
always@(posedge clk or negedge rstn)
begin
    if (! rstn)
        q <= 1'b0;
    else
        q <= a + b;
end
```

Features of VHDL and Verilog

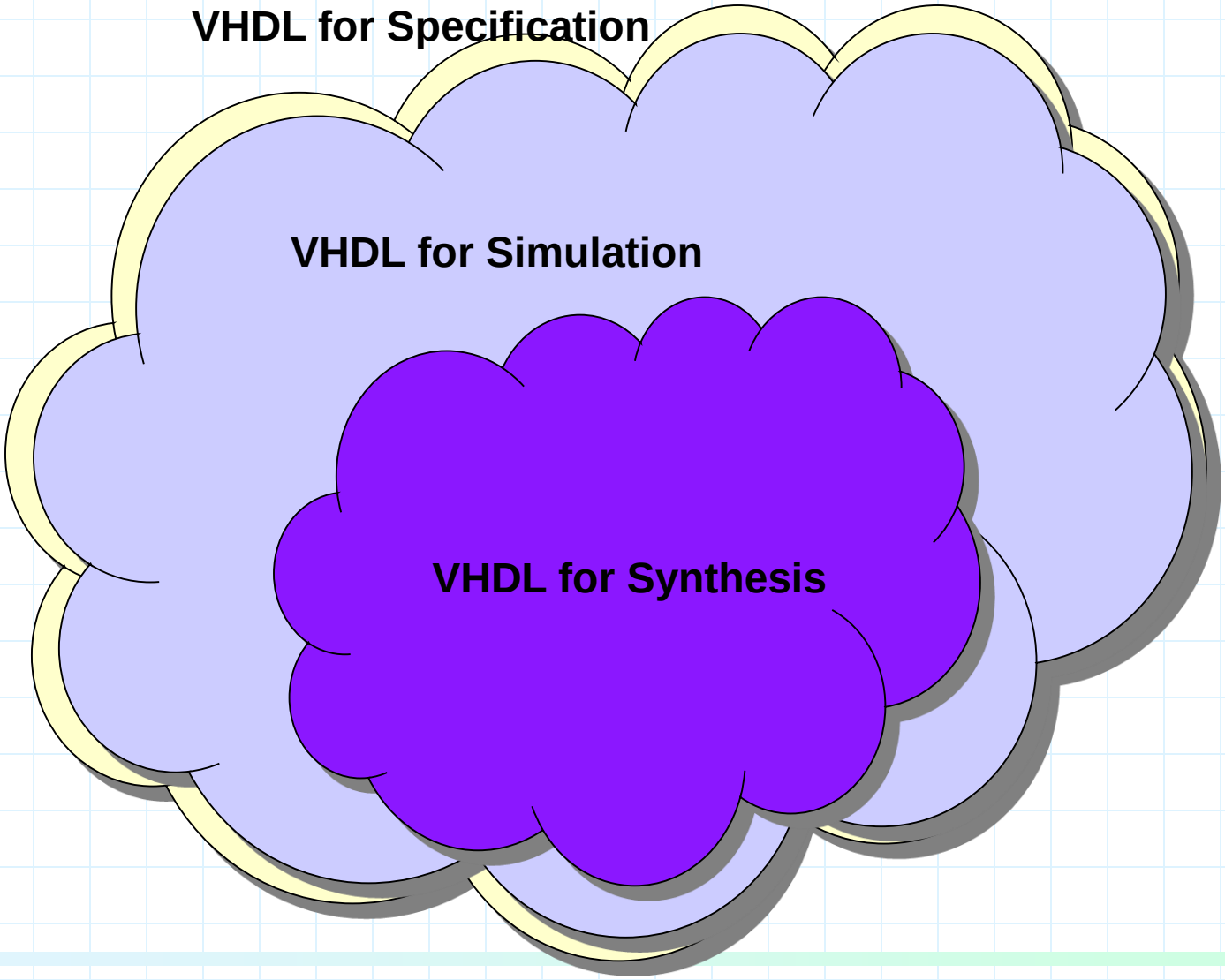
- Technology/vendor independent
- Portable
- Reusable

Other
Hardware Description Languages

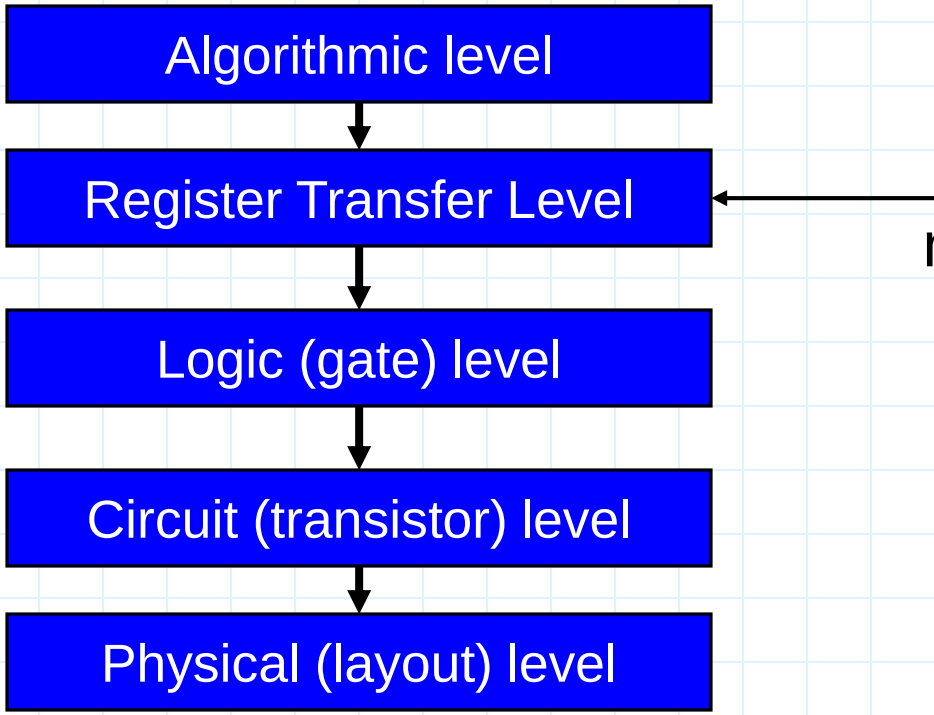
Other hardware description languages

- ABEL
- AHDL: Altera Hardware Description Language
- AHPL: A Hardware Programming Language
- CDL: Computer Design Language
- CONLAN: CONsensus LANguage
- IDL: Interactive Design Language
- ISPS: Instruction Set Processor Specification
- TEGAS: TEst Generation And Simulation
- TI-HDL: Texas Instruments Hardware Description Language
- ZEUS

VHDL for Synthesis

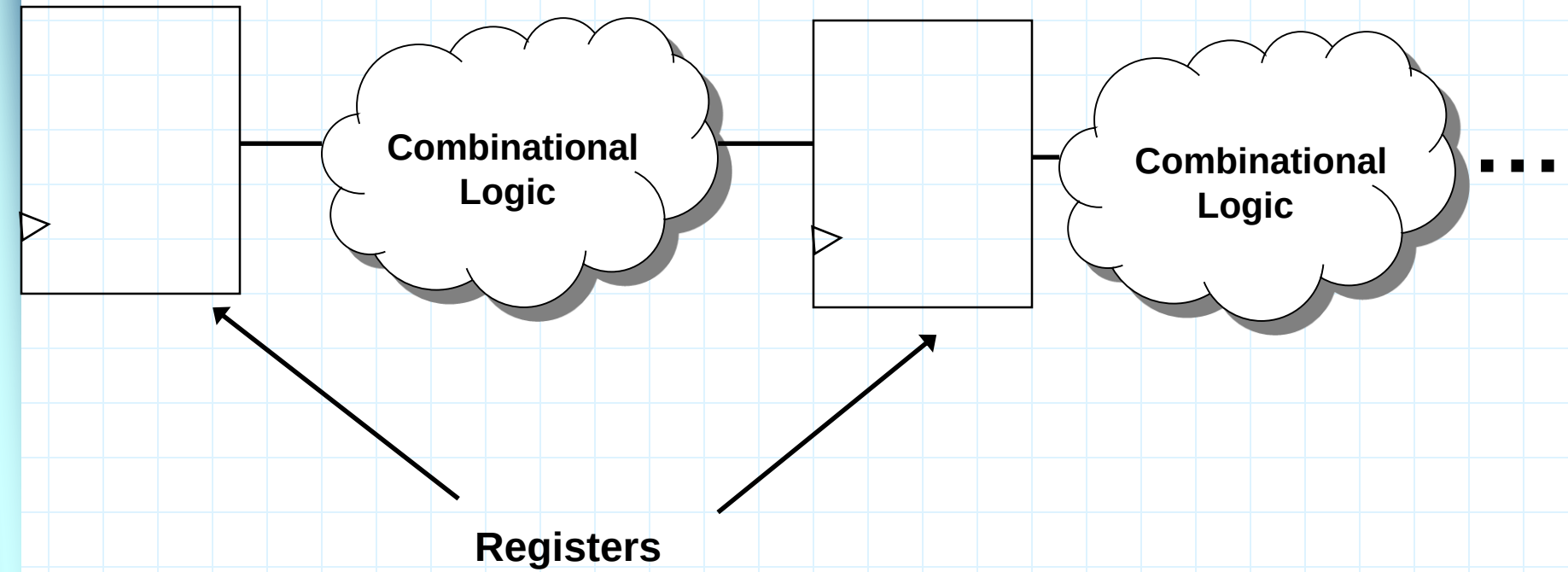

VHDL for Specification
VHDL for Simulation
VHDL for Synthesis

Levels of design description



Level of description
most suitable for synthesis

Register Transfer Logic (RTL) Design Description



VHDL Fundamentals

Naming and Labeling (1)

- VHDL is not case sensitive

  Example:

  Names or labels

  `databus`

  `Databus`

  `DataBus`

  `DATABUS`

  are all equivalent

Naming and Labeling (2)

General rules of thumb (according to VHDL-87)

1. All names should start with an alphabet character (a-z or A-Z)
2. Use only alphabet characters (a-z or A-Z) digits (0-9) and underscore (_)
3. Do not use any punctuation or reserved characters within a name (!, ?, ., &, +, -, etc.)
4. Do not use two or more consecutive underscore characters (__) within a name (e.g., Sel__A is invalid)
5. All names and labels in a given entity and architecture must be unique

Free Format

- VHDL is a “free format” language

  No formatting conventions, such as spacing or indentation imposed by VHDL compilers. Space and carriage return treated the same way.

  Example:

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

or

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

or

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

are all equivalent

Readability standards

ESA VHDL Modelling Guidelines

published by

European Space Research and Technology Center

in September 1994

available at the course web page

Readability standards

Selected issues covered by ESA Guidelines:

- Consistent Writing Style
- Consistent Naming Conventions
- Consistent Indentation
- Consistent Commenting Style
- Recommended File Headers
- File naming and contents
- Number of statements/declarations per line
- Ordering of port and signal declarations
- Constructs to avoid

Comments

- Comments in VHDL are indicated with a “double dash”, i.e., “--”
  - Comment indicator can be placed anywhere in the line
  - Any text that follows in the same line is treated as a comment
  - Carriage return terminates a comment
  - No method for commenting a block extending over a couple of lines

Examples:

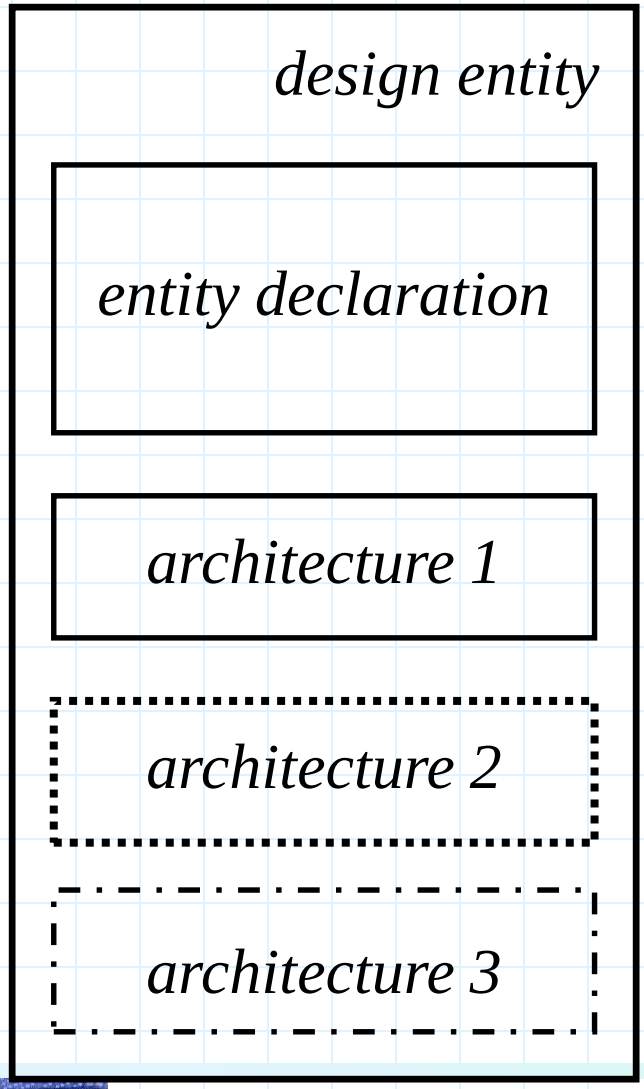
```
-- main subcircuit
Data_in <= Data_bus;     -- reading data from the input FIFO
```

Comments

- Explain Function of Module to Other Designers
- Explanatory, Not Just Restatement of Code
- Locate Close to Code Described
  - ✓Put near executable code, not just in a header

Design Entity

Design Entity

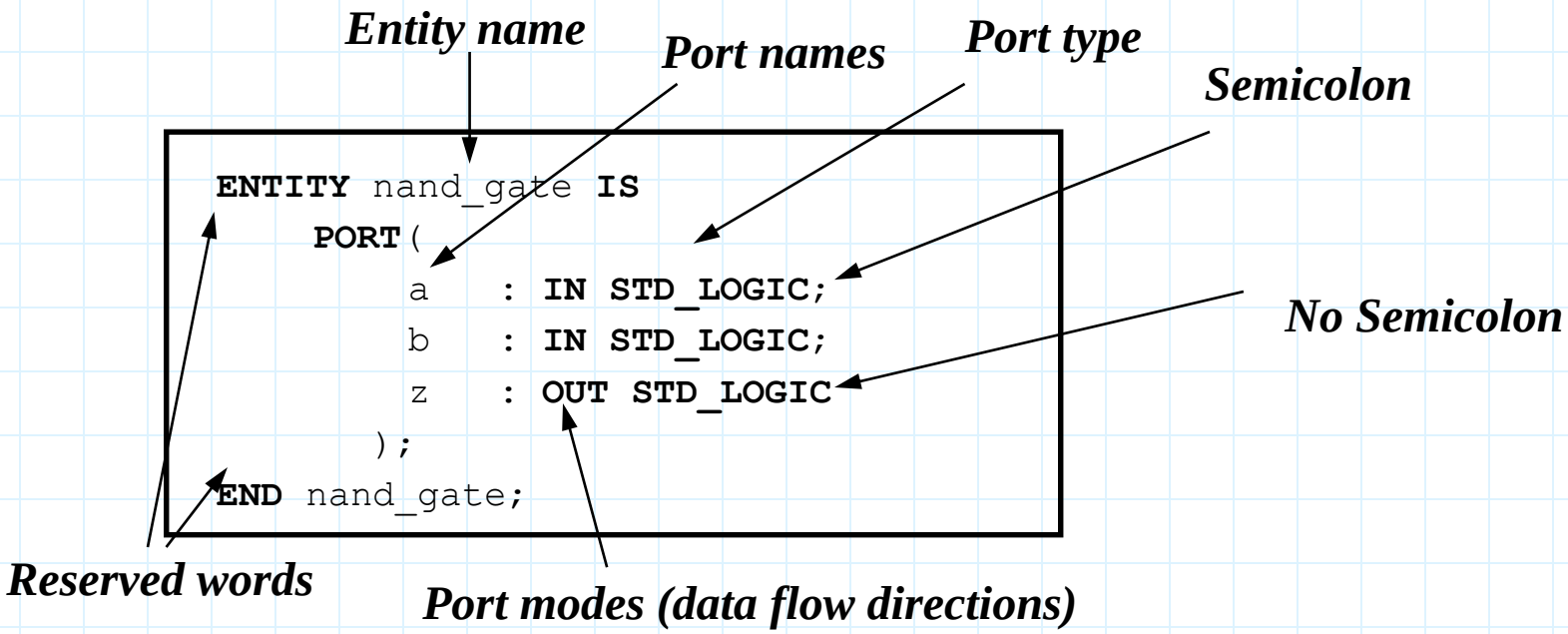


Design Entity - most basic building block of a design.

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

Entity Declaration

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



Entity declaration – simplified syntax

```
ENTITY entity_name IS
   PORT (
      port_name :  signal_mode  signal_type;
      port_name :  signal_mode  signal_type;
      ……..
      port_name :  signal_mode  signal_type);
END entity_name;
```

Architecture

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

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

Architecture – simplified syntax

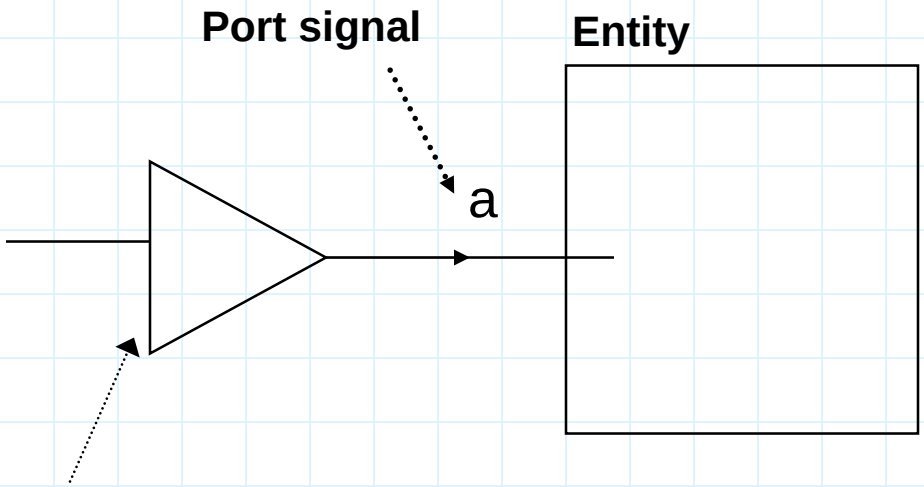
```
ARCHITECTURE architecture_name OF entity_name IS
   [ declarations ]
BEGIN
    code
END architecture_name;
```

Entity Declaration & Architecture

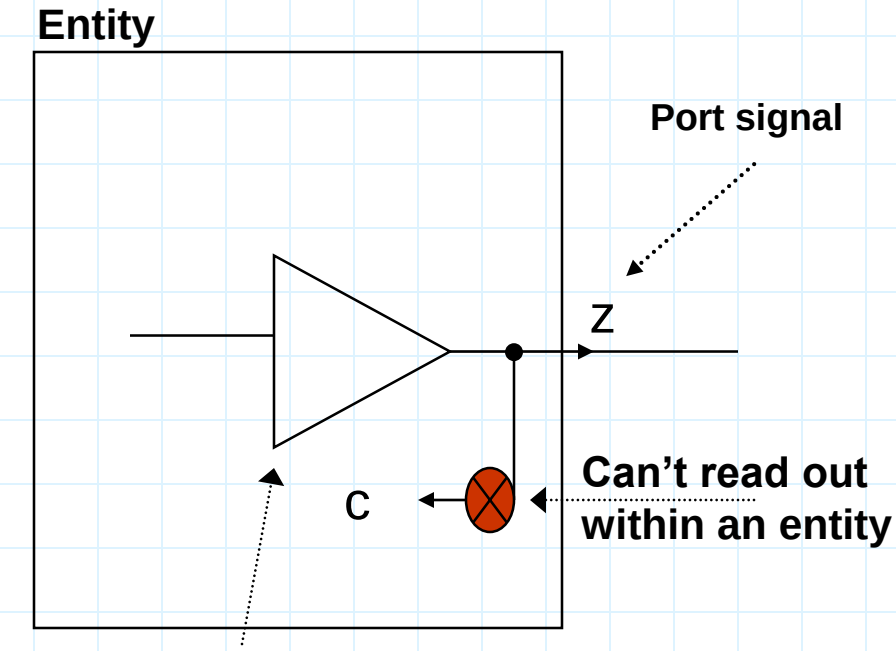
nand_gate.vhd

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

Mode *In*

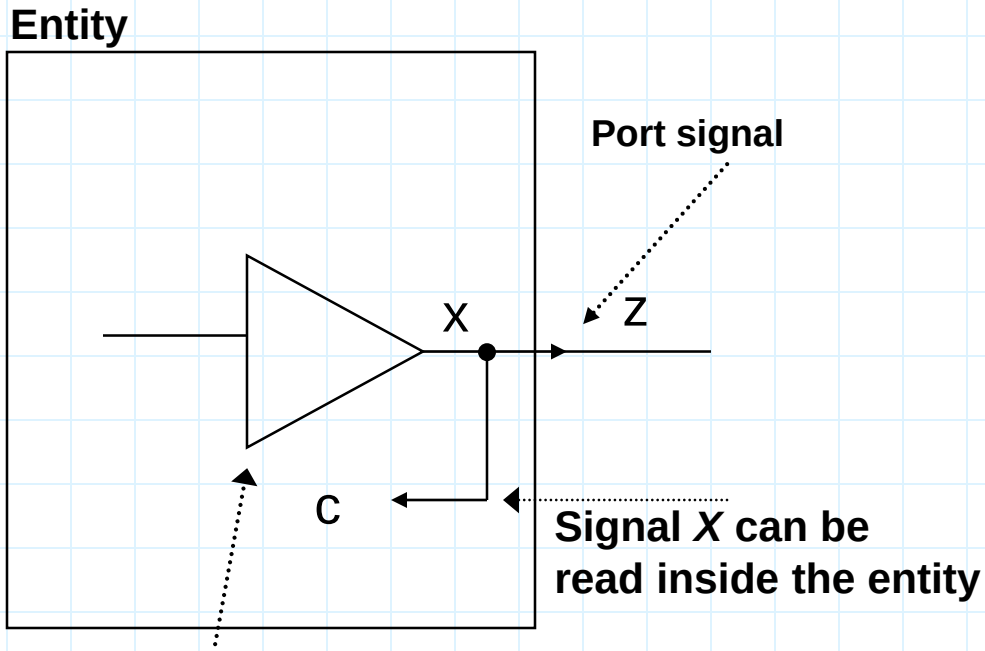


Mode *out*


Entity
Port signal
Z
C
Can’t read out
within an entity
Driver resides
inside the entity

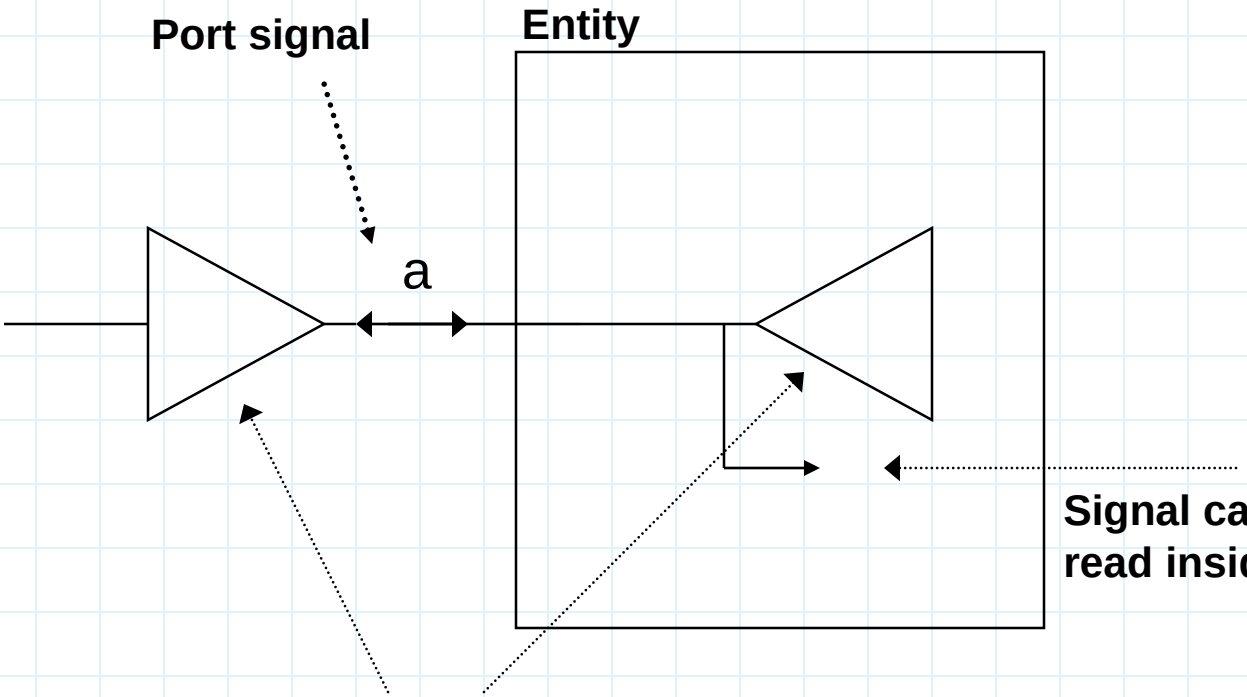
~~C <= Z~~

Mode *out* with *signal*

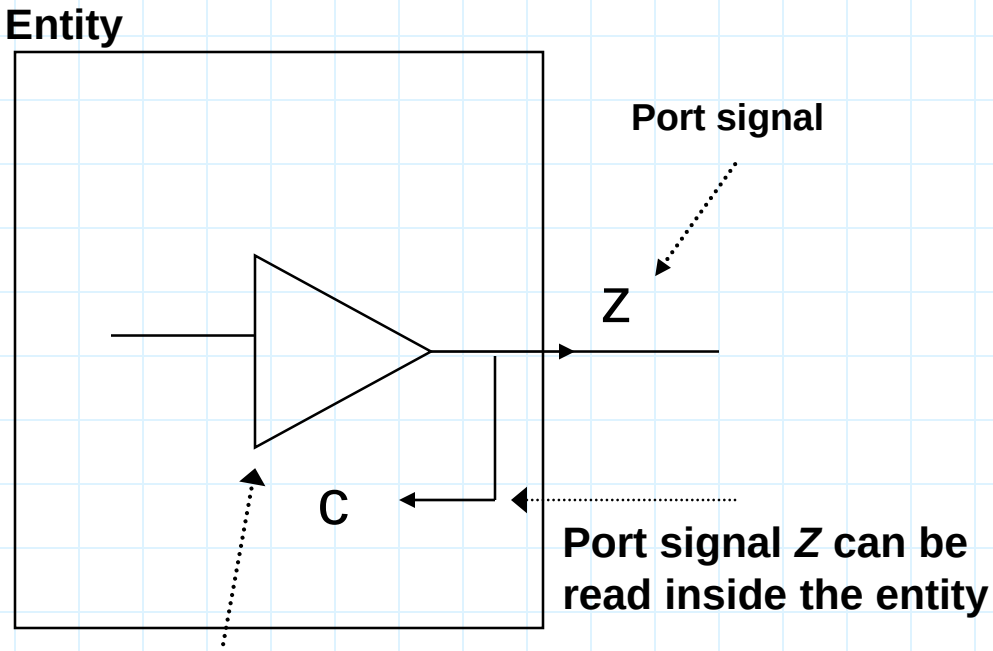



```
z <= x
c <= x
```

Mode *inout*


Port signal
Entity
a
Signal can be
read inside the entity
Driver may reside
both inside and outside
of the entity

Mode *buffer*


Entity
Port signal
Z
C
Port signal *Z* can be read inside the entity
Driver resides inside the entity
C <= Z

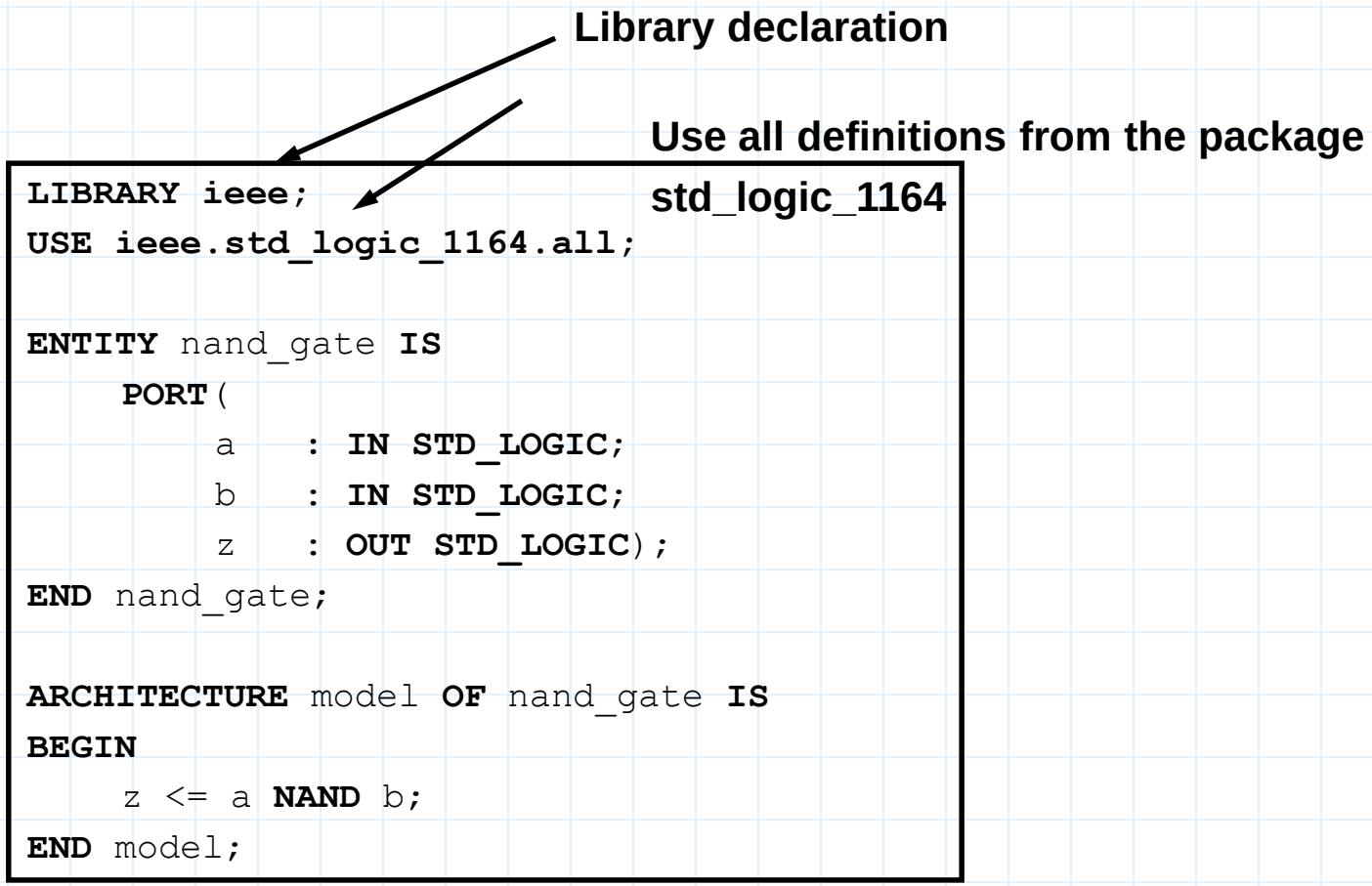
Port Modes

The *Port Mode* of the interface describes the direction in which data travels with respect to the c*omponent*

- ✓ **In**: Data comes in this port and can only be read within the entity. It can appear **only on the right side** of a signal or variable assignment.
- ✓ **Out**: The value of an output port can only be updated within the entity. **It cannot be read**. It can only appear **on the left side** of a signal assignment.
- ✓ **Inout**: The value of a bi-directional port can be read and updated within the entity model. It can appear on **both sides** of a signal assignment.
- ✓ **Buffer:** Used for a signal that is an output from an entity. The value of the signal can be used inside the entity, which means that in an assignment statement the signal can appear on the left and right sides of the <= operator

Libraries

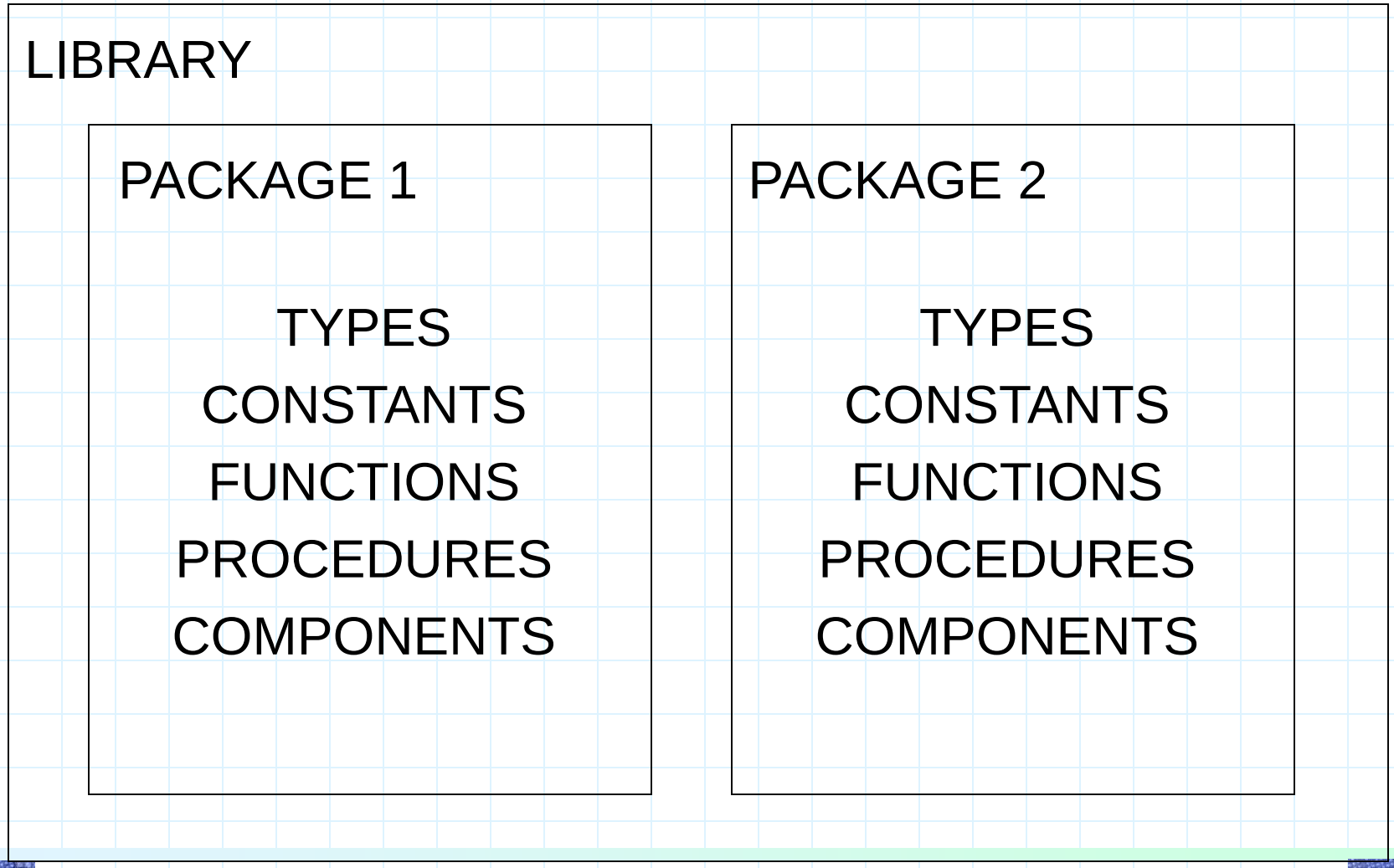
Library declarations



Library declarations - syntax

```
LIBRARY  library_name;
USE library_name.package_name.package_parts;
```

Fundamental 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

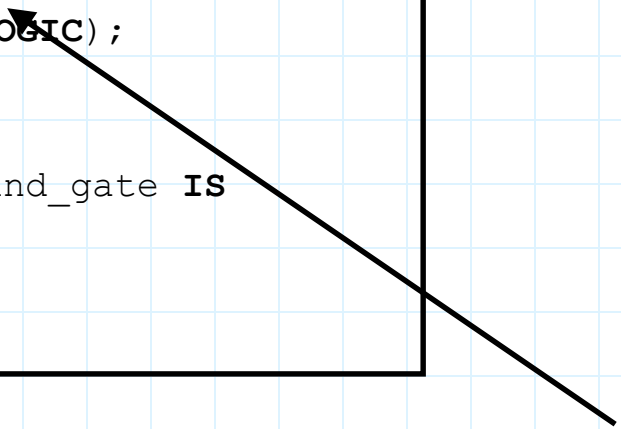
  Current designs after compilation

Visible by default

Οι τύποι STD_LOGIC

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

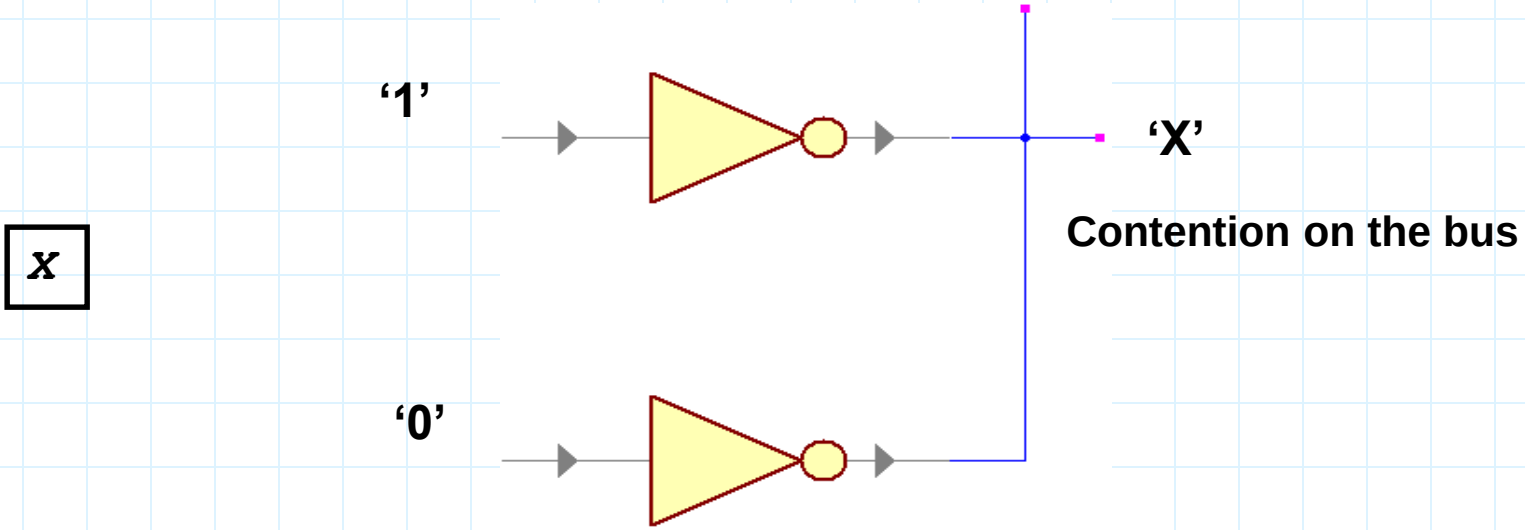


What is **STD_LOGIC** you ask?

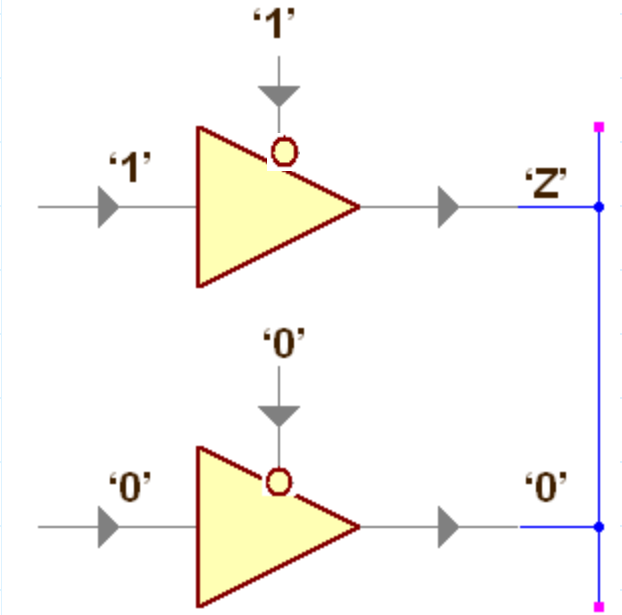
STD_LOGIC *type* demystified

Value	Meaning
‘X’	Forcing (Strong driven) Unknown
‘0’	Forcing (Strong driven) 0
‘1’	Forcing (Strong driven) 1
‘Z’	High Impedance
‘W’	Weak (Weakly driven) Unknown
‘L’	Weak (Weakly driven) 0. Models a pull down.
‘H’	Weak (Weakly driven) 1. Models a pull up.
‘-’	Don't Care

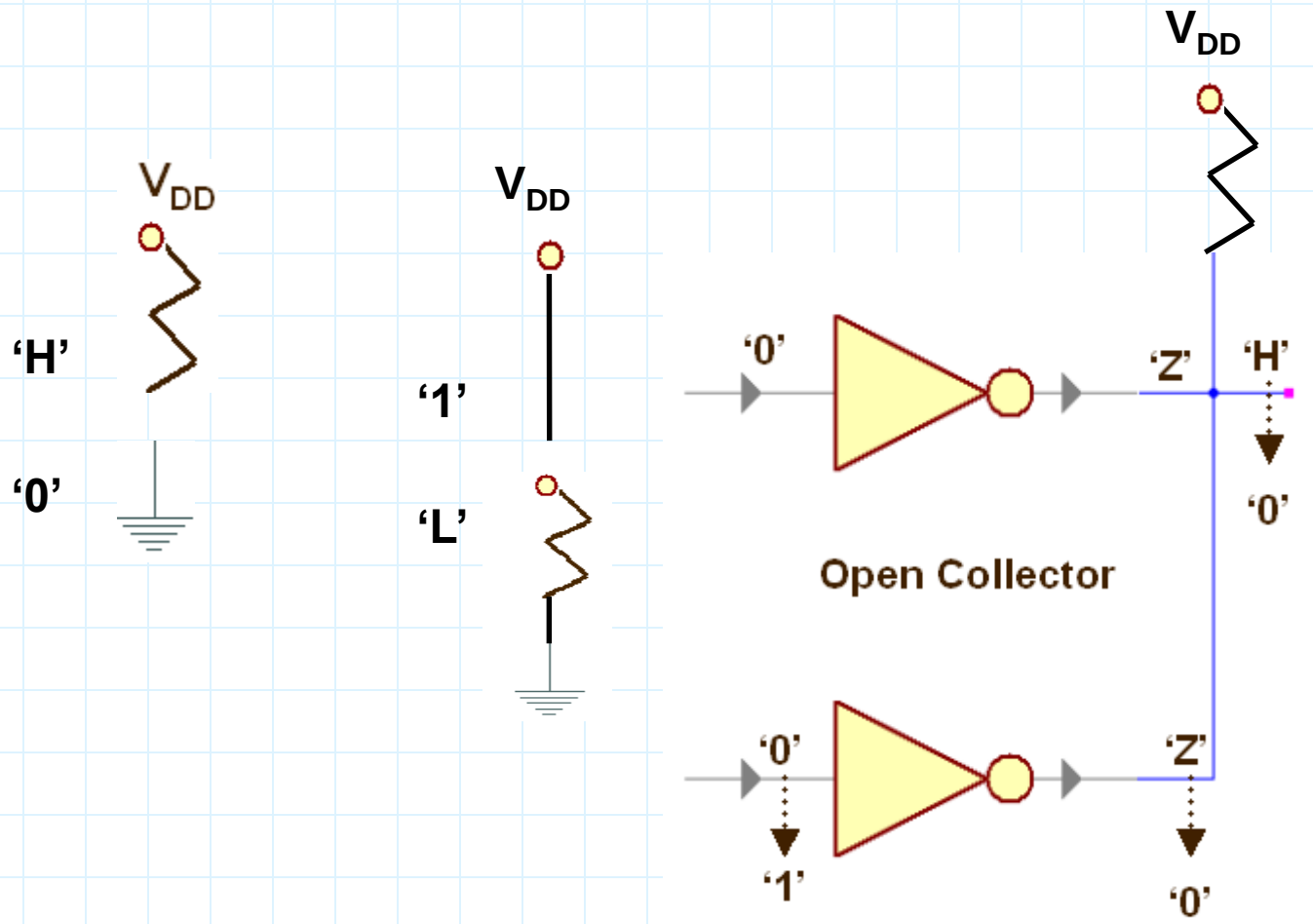
More on STD_LOGIC Meanings (1)



More on STD_LOGIC Meanings (2)



More on STD_LOGIC Meanings (3)



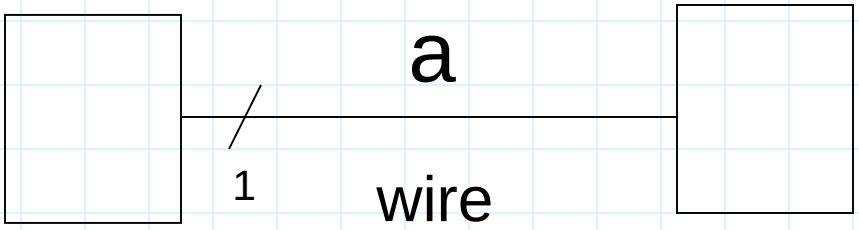
Resolving logic levels

	X	0	1	Z	W	L	H	-
X	X	X	X	X	X	X	X	X
0	X	0	X	0	0	0	0	X
1	X	X	1	1	1	1	1	X
Z	X	0	1	Z	W	L	H	X
W	X	0	1	W	W	W	W	X
L	X	0	1	L	W	L	W	X
H	X	0	1	H	W	W	H	X
-	X	X	X	X	X	X	X	X

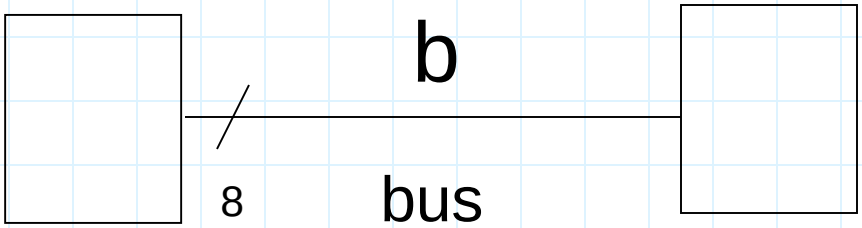
Modeling Wires and Buses

Signals

SIGNAL a : STD_LOGIC;



SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);



Standard Logic Vectors

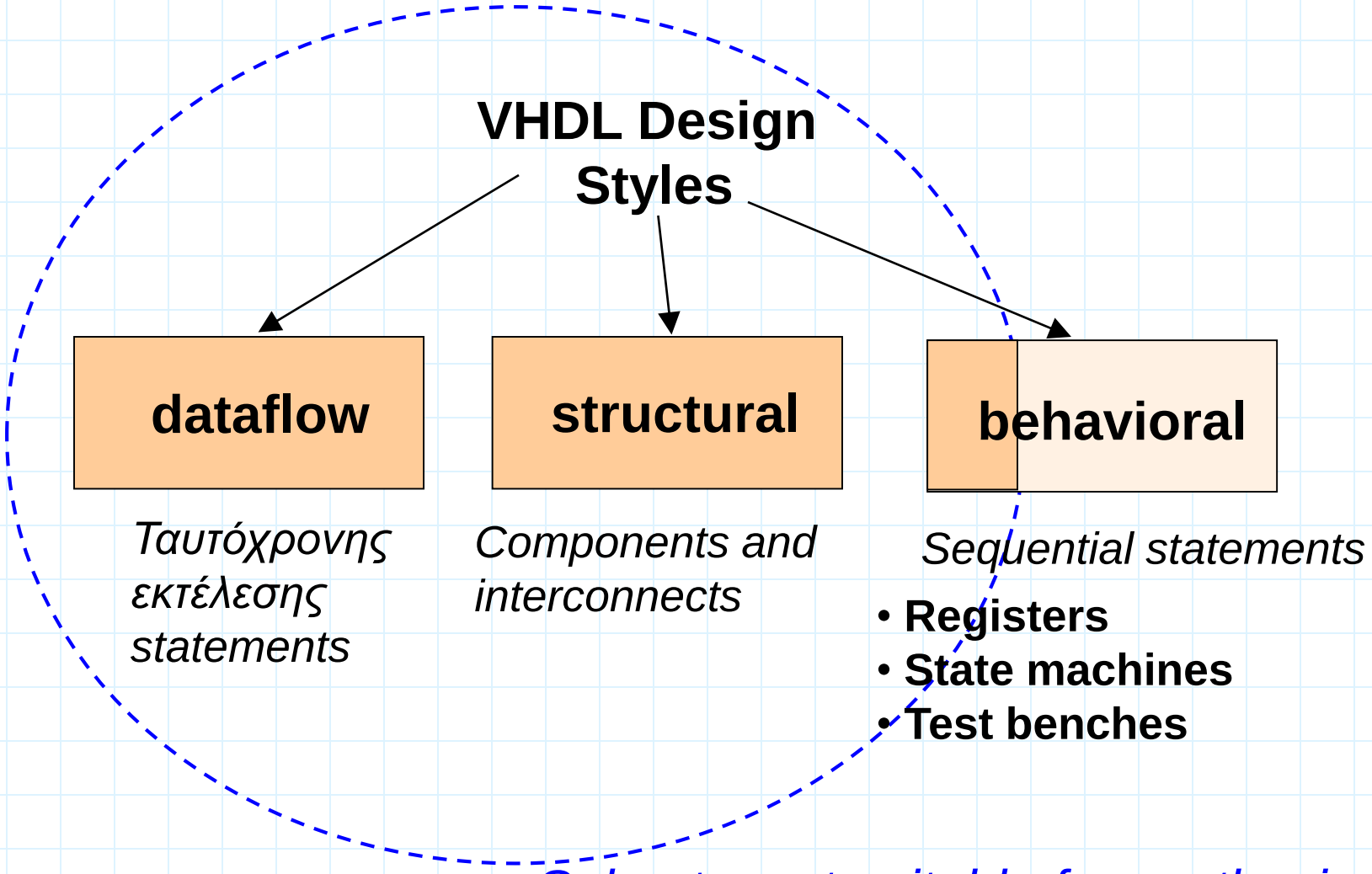
```
SIGNAL a: STD_LOGIC;
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL f: STD_LOGIC_VECTOR(8 DOWNTO 0);
                                .....
a <= '1';
b <= "0000";            -- Binary base assumed by default
c <= B"0000";           -- Binary base explicitly specified
d <= "0110_0111";   -- You can use '_' to increase readability
e <= X"AF67";           -- Hexadecimal base
f <= O"723";            -- Octal base
```

Vectors and Concatenation

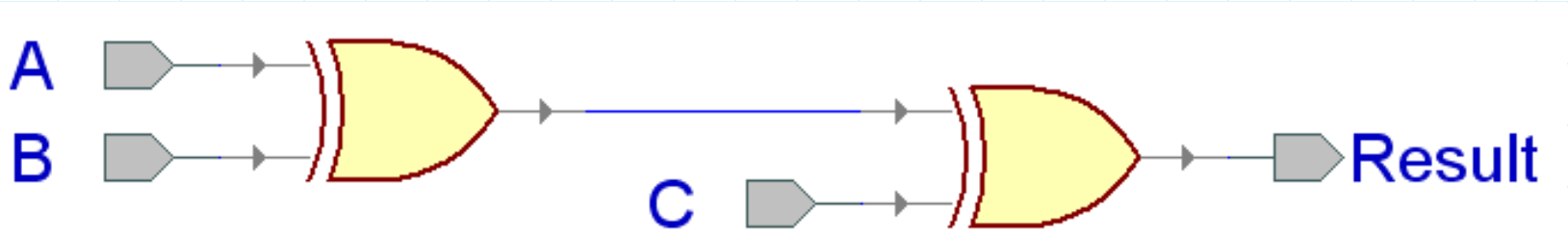
```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

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

VHDL Design Styles



xor3 Example

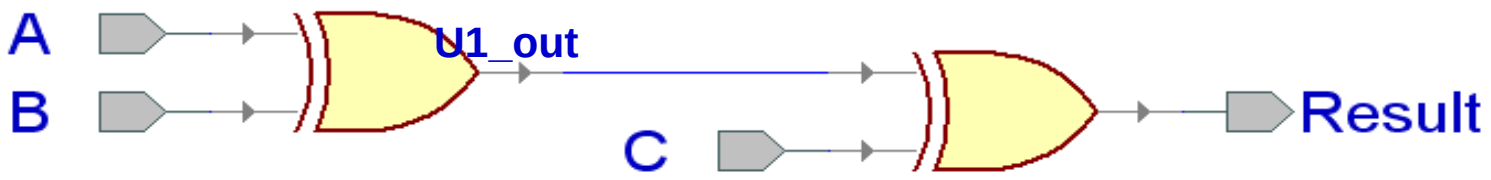


Entity xor3

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

Dataflow Architecture (xor3 gate)

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

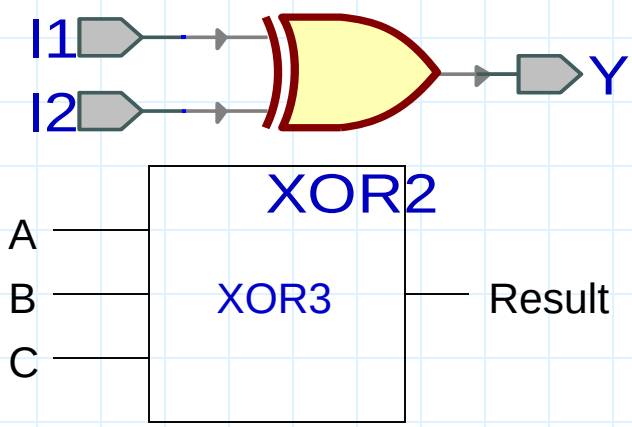


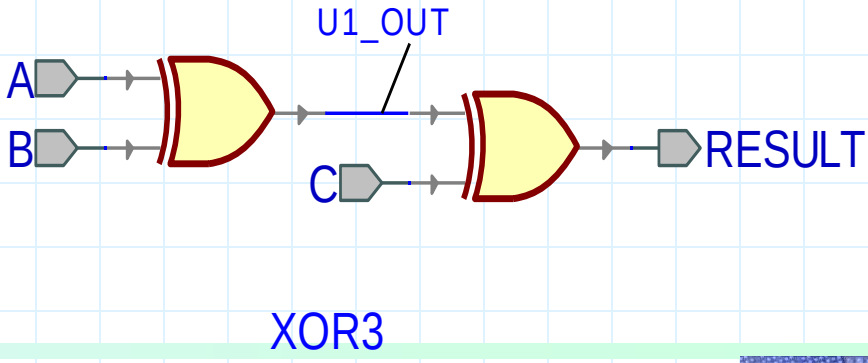
Dataflow Description

- Describes how data moves through the system and the various processing steps.
- Data Flow uses series of concurrent statements to realize logic. Concurrent statements are evaluated at the same time; thus, order of these statements doesn’t matter.
- Data Flow is most useful style when series of Boolean equations can represent a logic.

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 (I1 => A,
                        I2 => B,
                        Y  => U1_OUT);

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

END structural;
```





Component and Instantiation (1)

- Named association connectivity (recommended)

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

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

Component and Instantiation (2)

- Positional association connectivity (not recommended)

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

U1: xor2 PORT MAP (A, B, U1_OUT);
```

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)

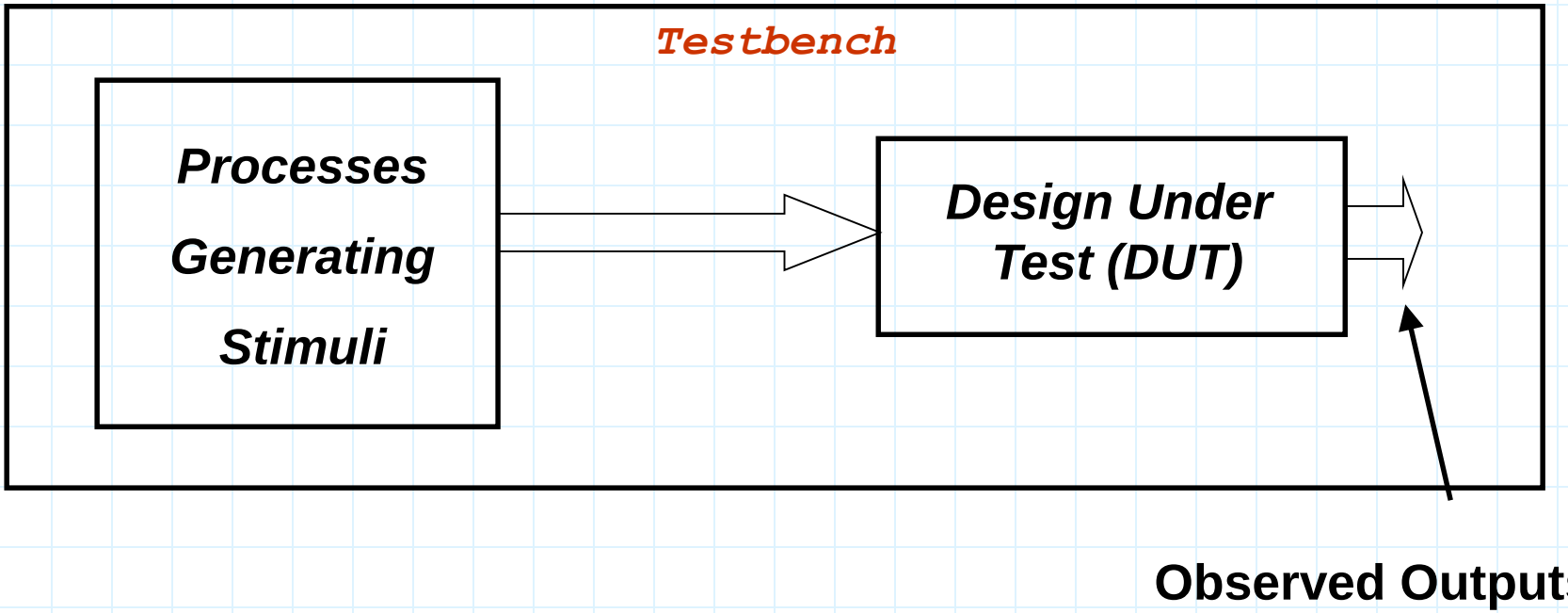
```
ARCHITECTURE behavioral OF xor3 IS
BEGIN
xor3_behave: PROCESS (A,B,C)
BEGIN
    IF ((A XOR B XOR C) = '1') THEN
        Result <= '1';
    ELSE
        Result <= '0';
    END IF;
END PROCESS xor3_behave;
END behavioral;
```

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*.

Testbenches

Testbench Block Diagram



Testbench Defined

- *Testbench* applies stimuli (drives the inputs) to the Design Under Test (DUT) and (optionally) verifies expected outputs.
- The results can be viewed in a waveform window or written to a file.
- Since *Testbench* is written in VHDL, it is not restricted to a single simulation tool (portability).
- The same *Testbench* can be easily adapted to test different implementations (i.e. different *architectures*) of the same design.

Testbench Anatomy

```
ENTITY tb IS
              --TB entity has no ports
END tb;

ARCHITECTURE arch_tb OF tb IS

              --Local signals and constants

  COMPONENT TestComp --All Design Under Test component declarations
            PORT (  );
  END COMPONENT;
-------------------------------------------------------
BEGIN
  testSequence: PROCESS
                                        -- Input stimuli
  END PROCESS;

  DUT:TestComp PORT MAP(                -- Instantiations of DUTs
                        );
END arch_tb;
```

Testbench for XOR3 (1)

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

ENTITY xor3_tb IS
END xor3_tb;

ARCHITECTURE xor3_tb_architecture OF xor3_tb IS
-- Component declaration of the tested unit
COMPONENT xor3
PORT(
 A : IN STD_LOGIC;
 B : IN STD_LOGIC;
 C : IN STD_LOGIC;
 Result : OUT STD_LOGIC );
END COMPONENT;

-- Stimulus signals - signals mapped to the input and inout ports of tested entity
SIGNAL test_vector: STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL test_result : STD_LOGIC;
```

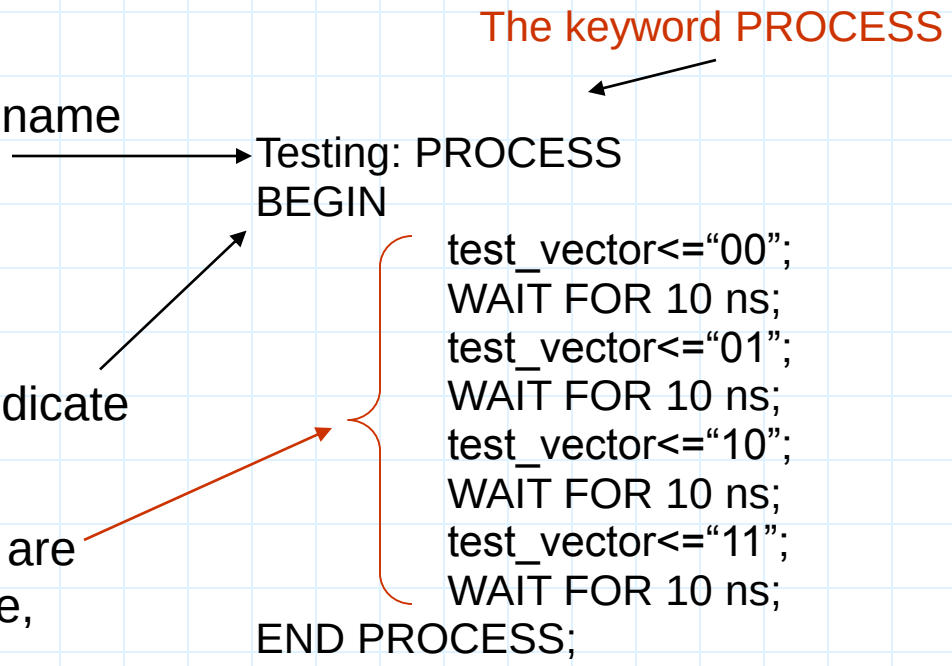
Testbench for XOR3 (2)

```
BEGIN
UUT : xor3
 PORT MAP (
                A => test_vector(0),
                B => test_vector(1),
                C => test_vector(2),
                Result => test_result);
                                        );
Testing: PROCESS
BEGIN
test_vector <= "000";
WAIT FOR 10 ns;
test_vector <= "001";
WAIT FOR 10 ns;
test_vector <= "010";
WAIT FOR 10 ns;
test_vector <= "011";
WAIT FOR 10 ns;
test_vector <= "100";
WAIT FOR 10 ns;
test_vector <= "101";
WAIT FOR 10 ns;
test_vector <= "110";
WAIT FOR 10 ns;
test_vector <= "111";
WAIT FOR 10 ns;
END PROCESS;
END xor3_tb_architecture;
```

What is a PROCESS?

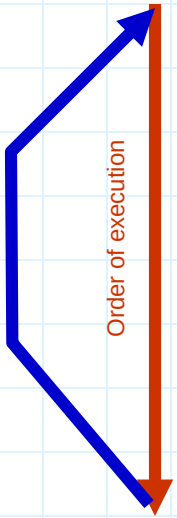
- ✓ A process is a sequence of instructions referred to as sequential statements.

- A process can be given a unique name using an optional LABEL
- This is followed by the keyword PROCESS
- The keyword BEGIN is used to indicate the start of the process
- All statements within the process are executed SEQUENTIALLY. Hence, order of statements is important.
- A process must end with the keywords END PROCESS.



Execution of statements in a PROCESS

- The execution of statements continues sequentially till the last statement in the process.
- After execution of the last statement, the control is again passed to the beginning of the process.




```
Testing: PROCESS
BEGIN
    test_vector<="00";
    WAIT FOR 10 ns;
    test_vector<="01";
    WAIT FOR 10 ns;
    test_vector<="10";
    WAIT FOR 10 ns;
    test_vector<="11";
    WAIT FOR 10 ns;
END PROCESS;
```

Program control is passed to the first statement after BEGIN

PROCESS with a WAIT Statement

- The last statement in the PROCESS is a WAIT instead of WAIT FOR 10 ns.
- This will cause the PROCESS to suspend indefinitely when the WAIT statement is executed.
- This form of WAIT can be used in a process included in a testbench when all possible combinations of inputs have been tested or a non-periodical signal has to be generated.

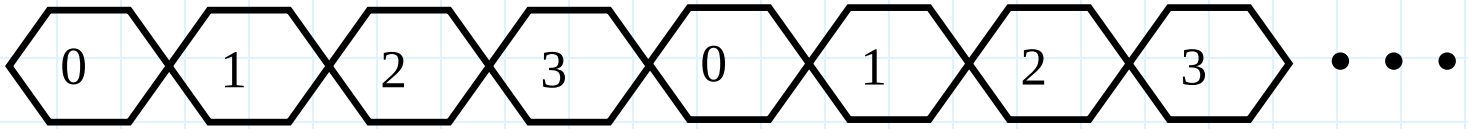
```
Testing: PROCESS
BEGIN
        test_vector<="00";
        WAIT FOR 10 ns;
        test_vector<="01";
        WAIT FOR 10 ns;
        test_vector<="10";
        WAIT FOR 10 ns;
        test_vector<="11";
        WAIT;
END PROCESS;
```

Order of execution

Program execution stops here

WAIT FOR vs. WAIT

WAIT FOR: waveform will keep repeating itself forever



WAIT : waveform will keep its state after the last wait instruction.

Loop Statement

- Loop Statement

```
FOR i IN range LOOP
    statements
END LOOP;
```

- Repeats a Section of VHDL Code
  - Example: process every element in an array in the same way

Loop Statement – Example (1)

```
Testing: PROCESS
BEGIN
        test_vector<="000";
        FOR i IN 0 TO 7 LOOP
            WAIT FOR 10 ns;
            test_vector<=test_vector+”001";
        END LOOP;
END PROCESS;
```

Loop Statement – Example (2)

```
Testing: PROCESS
BEGIN
        test_ab<="00";
        test_sel<="00";
        FOR i IN 0 TO 3 LOOP
                FOR j IN 0 TO 3 LOOP
                        WAIT FOR 10 ns;
                        test_ab<=test_ab+"01";
                END LOOP;
                test_sel<=test_sel+"01";
        END LOOP;
END PROCESS;
```

?