

# ECE 1195: Advanced Digital Design

## *VHDL – Part 1*

Dr. Amr Mahmoud





# VHDL

# VHDL

- **VHDL** is a language for describing digital hardware.
- **VHDL** is an acronym for **V**HSIC (**V**ery **H**igh **S**peed Integrated **C**ircuit) **H**ardware **D**escription **L**anguage

# Case Sensitivity

- VHDL is not case sensitive

Example:

Names or labels

**databus**

**Databus**

**DataBus**

**DATABUS**

are all equivalent

# Naming and Labeling

## General rules of thumb

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
6. All names should not be from the reserved words list

# 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

# 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

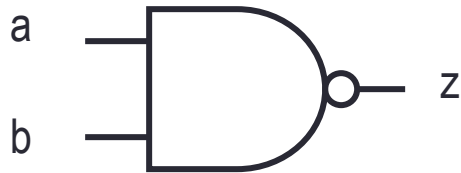
- Explain function of module to other designers
- Explanatory, not just restatement of code
- Locate close to code described
  - Put near executable code, not just in a header





# DESIGN ENTITY

# 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 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;
```

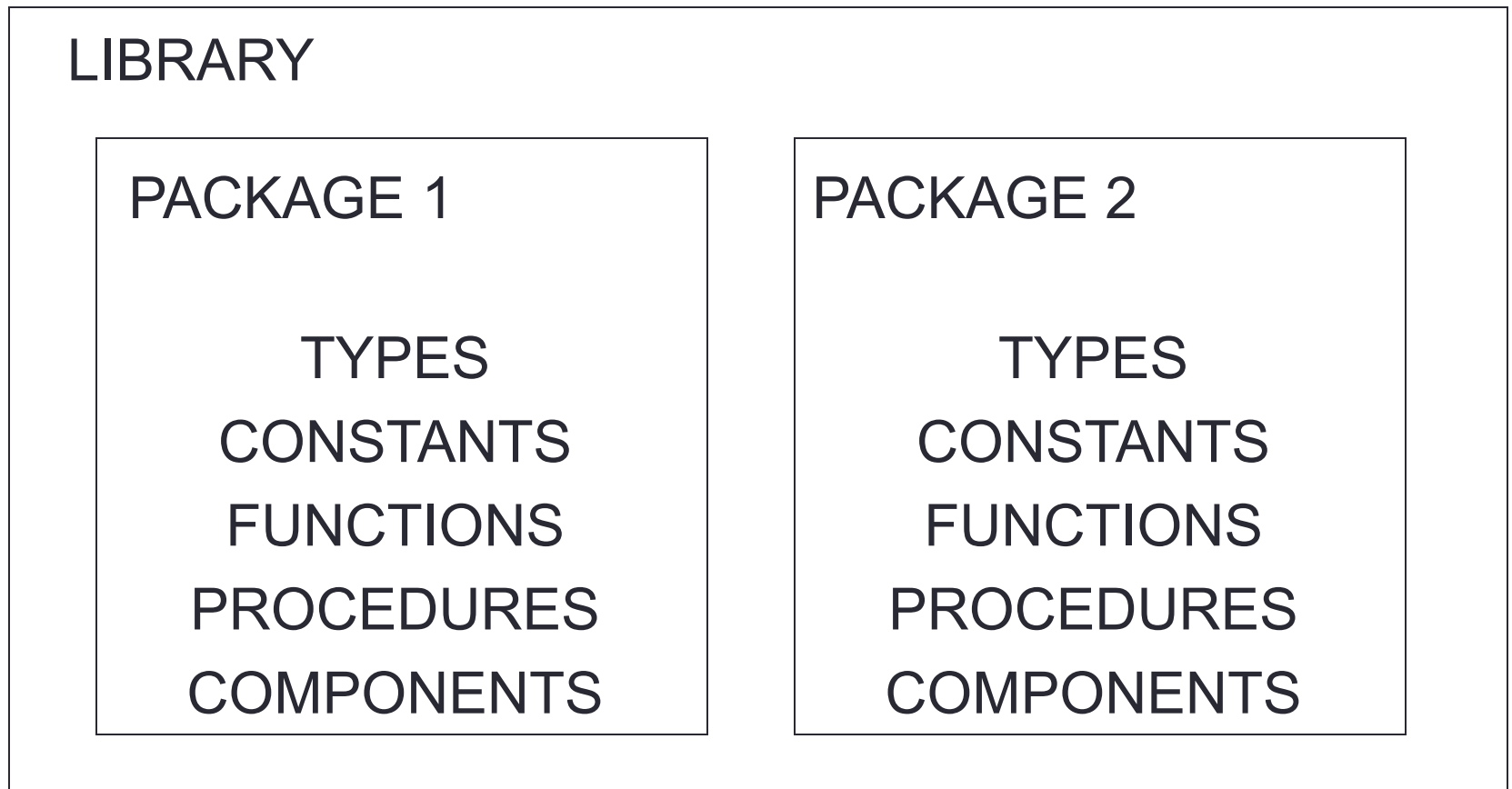
LIBRARY DECLARATION

ENTITY

ARCHITECTURE

# Fundamental Parts Of A Library

- Library is a collection of commonly used pieces of code, grouped for reuse.



# Library Declarations

```
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

Use all definitions from the package  
std\_logic\_1164

# Library Declarations - Syntax

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

# Commonly Used Libraries

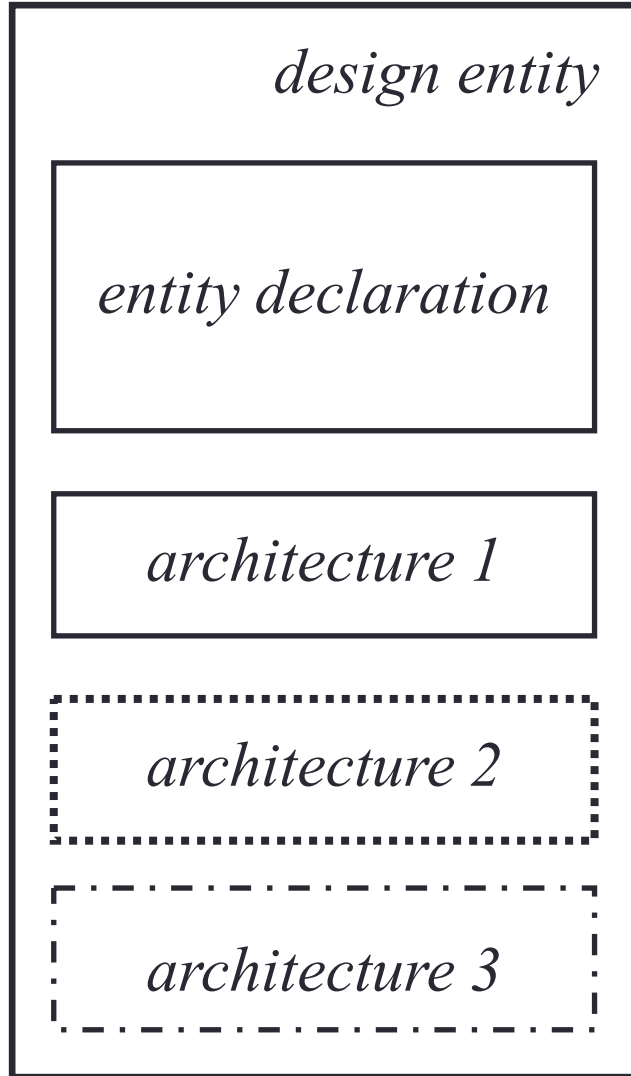
- `ieee`
    - Specifies multi-level logic system including `STD_LOGIC`, and `STD_LOGIC_VECTOR` data type
  - `std`
    - Specifies pre-defined data types (`BIT`, `BOOLEAN`, `INTEGER`, `REAL`, `SIGNED`, `UNSIGNED`, etc.), arithmetic operations, basic type conversion functions, basic text i/o functions, etc.
  - `work`
    - User-created designs after compilation
- Needs to be explicitly declared**
- Visible by default**

# Standard VHDL Packages

- **library IEEE;**
  - **use IEEE.std\_logic\_1164.all;**
  - **use IEEE.std\_logic\_textio.all;**
  - **use IEEE.std\_logic\_arith.all;**
  - **use IEEE.numeric\_bit.all;**
  - **use IEEE.numeric\_std.all;**
  - **use IEEE.std\_logic\_signed.all;**
  - **use IEEE.std\_logic\_unsigned.all;**
  - **use IEEE.math\_real.all;**
  - **use IEEE.math\_complex.all;**
- **library STD;**
  - **use STD.textio;**



# Design Entity

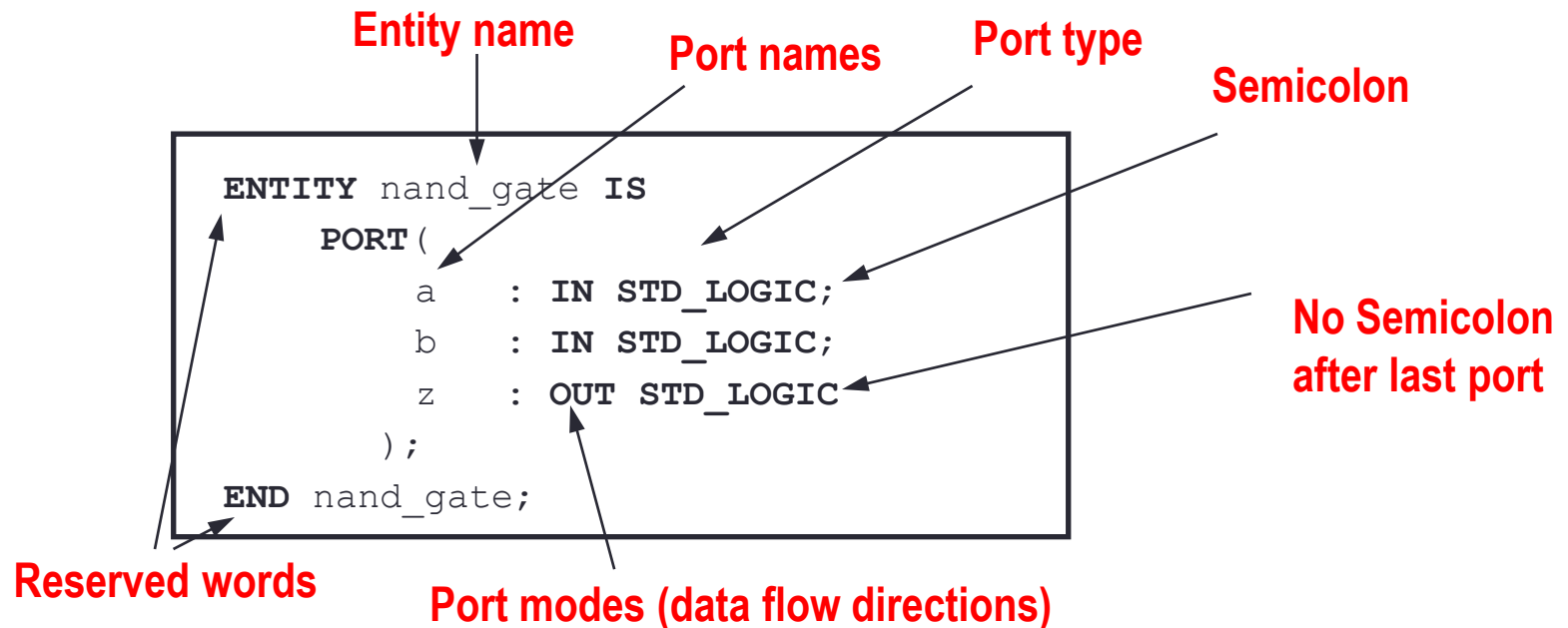


*Design Entity* - most basic building block of a design.

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

# Entity Declaration

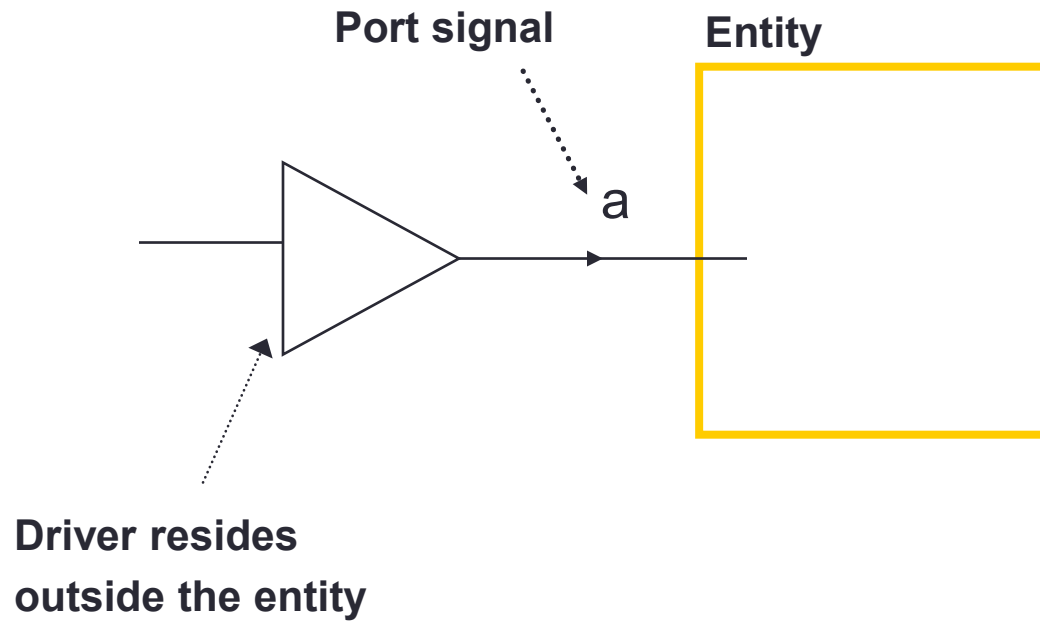
- Entity Declaration describes the interface of the component, i.e. input and output ports.



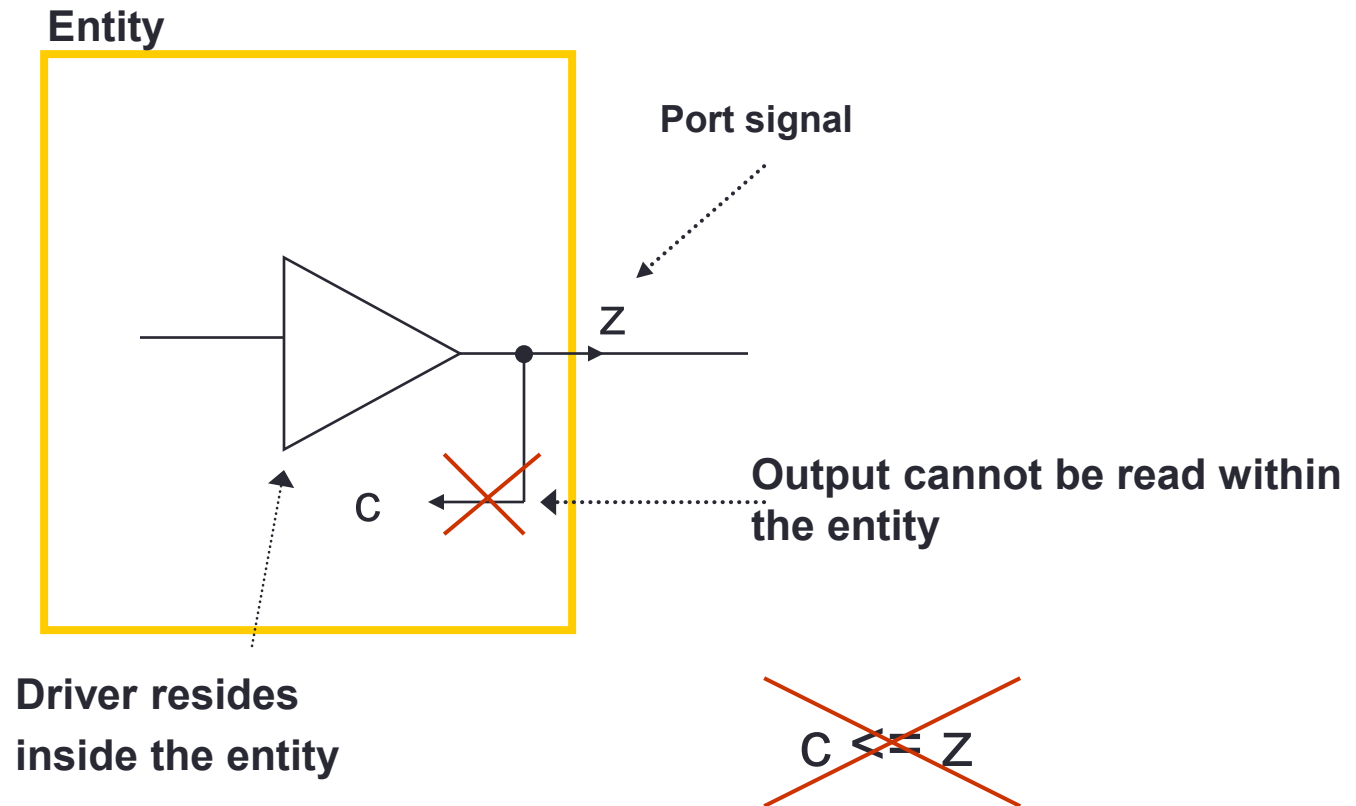
# Entity Declaration – Simplified Syntax

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

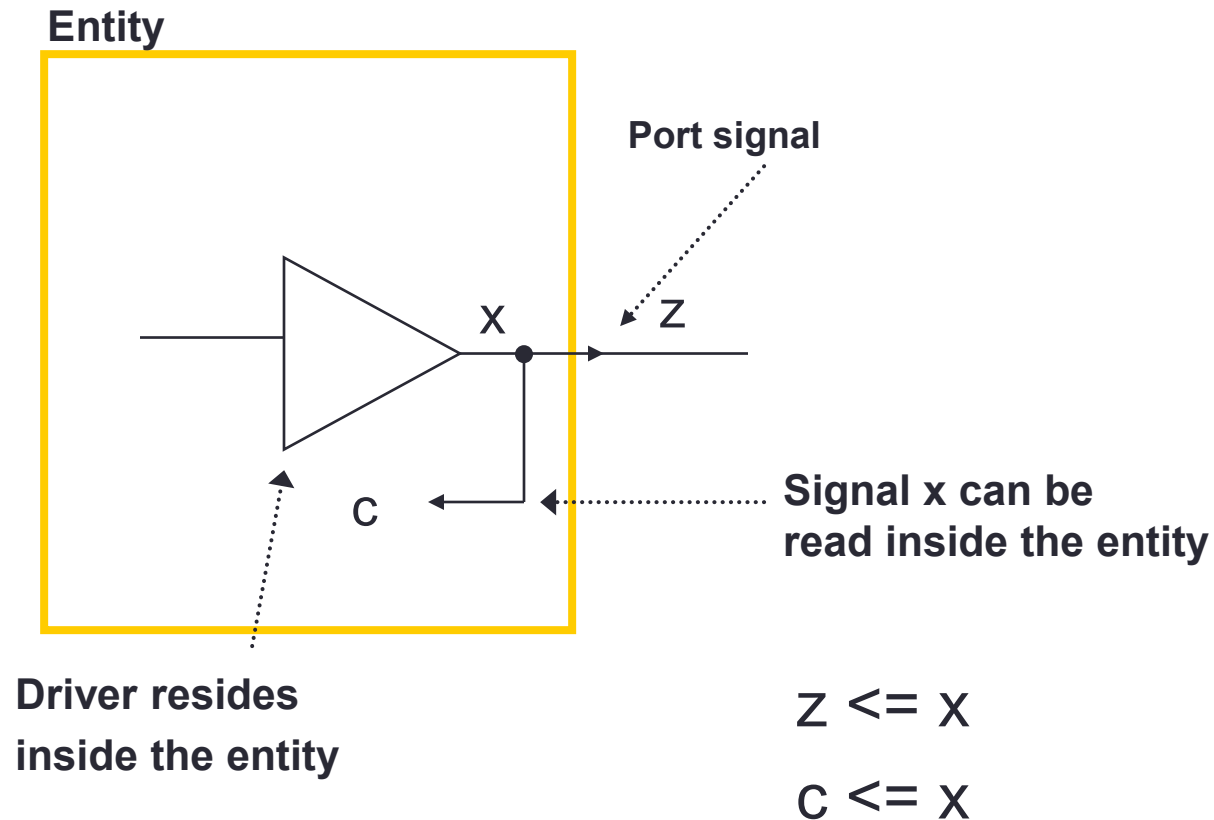
# Port Mode IN



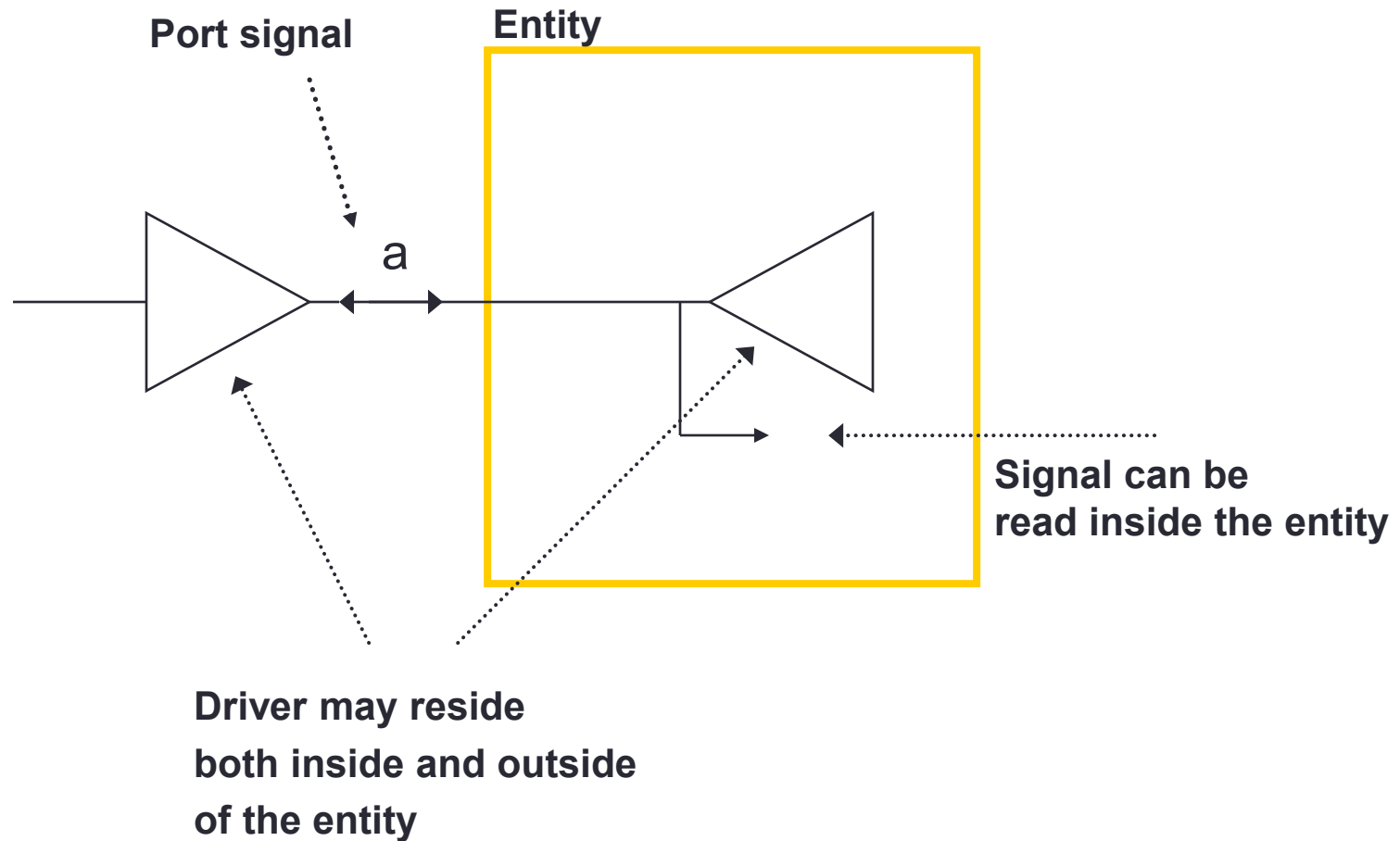
# Port Mode OUT



# Port Mode OUT (with extra signal)



# Port Mode INOUT



# Port Modes: Summary

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

- **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.



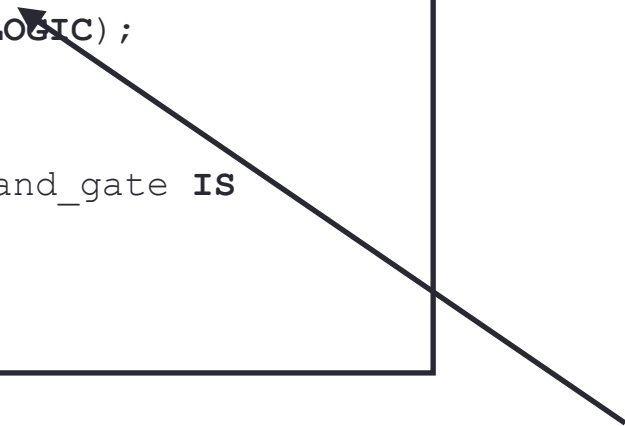
# STD\_LOGIC

# 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;
```



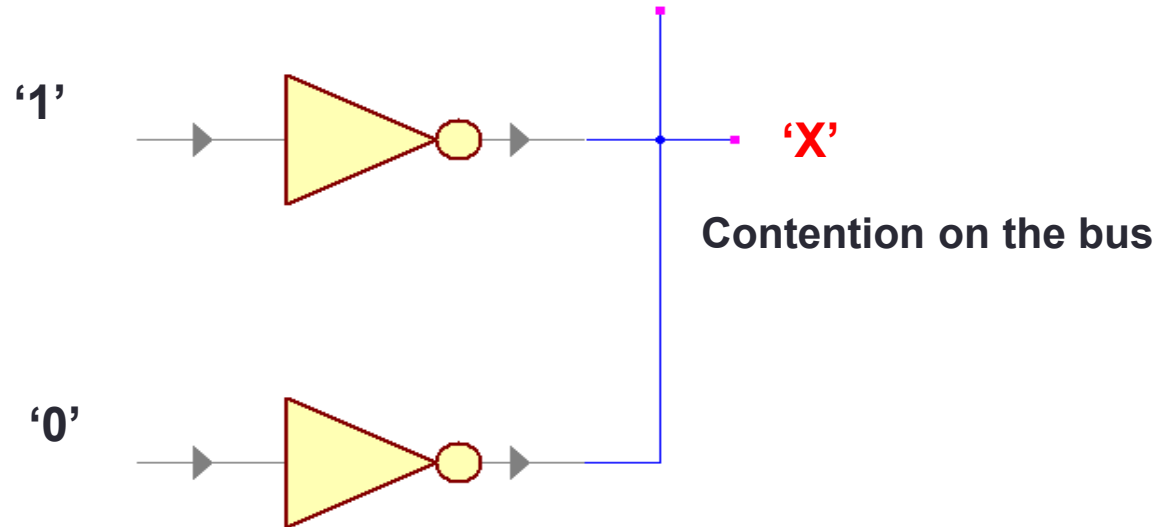
# BIT versus STD\_LOGIC

- BIT type can only have a value of '0' or '1'
- STD\_LOGIC can have nine values
  - 'U','0','1','X','Z','W','L','H','-'
  - Useful mainly for simulation
  - '0','1', and 'Z' are synthesizable

# STD\_LOGIC *type*

Value	Meaning
'U'	Uninitialized
'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)



- *Do not care.*
- *Can be assigned to outputs for the case of invalid inputs (may produce significant improvement in resource utilization after synthesis).*
- *Use with caution*
  - '1' = '-' gives FALSE*

# Resolving Logic Levels

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

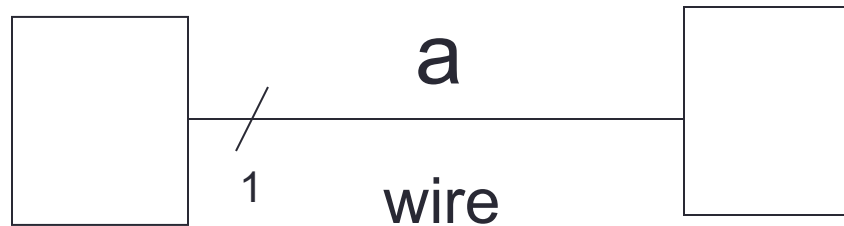


# **MODELING WIRES AND BUSES**

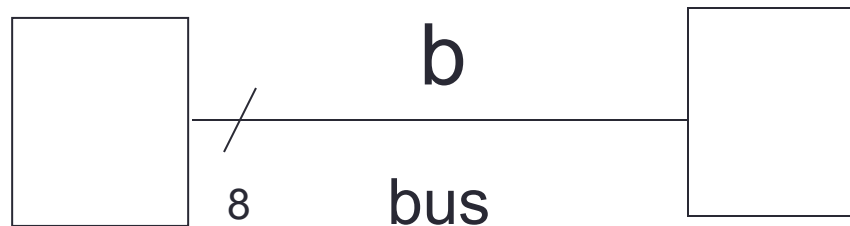


# Single Wire Versus Bus

```
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
```

# Single versus Double Quote

- Use single quote to hold a single bit signal
  - `a <= '0'`, `a <= 'Z'`
- Use double quote to hold a multi-bit signal
  - `b <= "00"`, `b <= "11"`

# 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"
```

# Architecture

- Entity describes the **ports** of the module
- Architecture describes the **functionality** of the module (i.e. what does the module do)
- 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

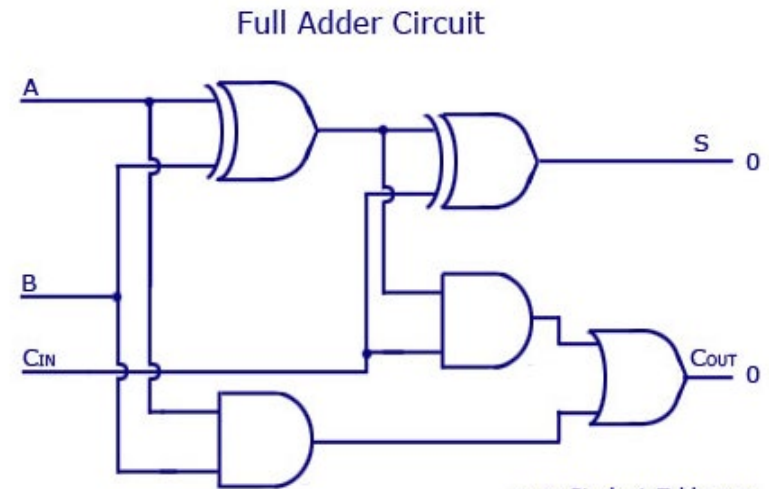
## nand\_gate.vhd

```
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;
```

# Task

- ❑ Write the VHDL code that describe  
**Full adder.**

```
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;
```



[www.StudentsTable.com](http://www.StudentsTable.com)



# Task (solution)

□ Write the VHDL code that describe

**Full adder.**

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY FULL_Adder IS -- Full adder with inputs A,B,Cin & outputs Sum,Cou
    PORT( A: IN STD_LOGIC;
          B: IN STD_LOGIC;
          Cin: IN STD_LOGIC;
          Sum: OUT STD_LOGIC;
          Cout: OUT STD_LOGIC);
END FULL_Adder;

Architecture comb OF FULL_Adder IS
BEGIN
    Sum <= A XOR B XOR Cin;
    Cout <= (A AND B) OR (B AND Cin) OR (Cin AND A);

END comb;
```

# Do it yourself

- ❑ Write the VHDL code that describe one cell of **adder/subtractor** as shown in figure:

