# Integrated Circuits Design by FPGA

م.م. أحمد مؤيد عبدالحسين

جامعة الفرات الأوسط التقنية / الكلية التقنية الهندسية / نجف

# Lecture 11

## Packages and Components

## Arithmetic Logic Unit (ALU)

## Objectives of this Lecture

- To review **Packages** and **Components**
- To implement **ALU** example using **Components**.

## Contents of this Lecture

- Introduction (Review)
- PACKAGE (Review)
- COMPONENT (Review)
- ALU Example using COMPONENTS

# Introduction

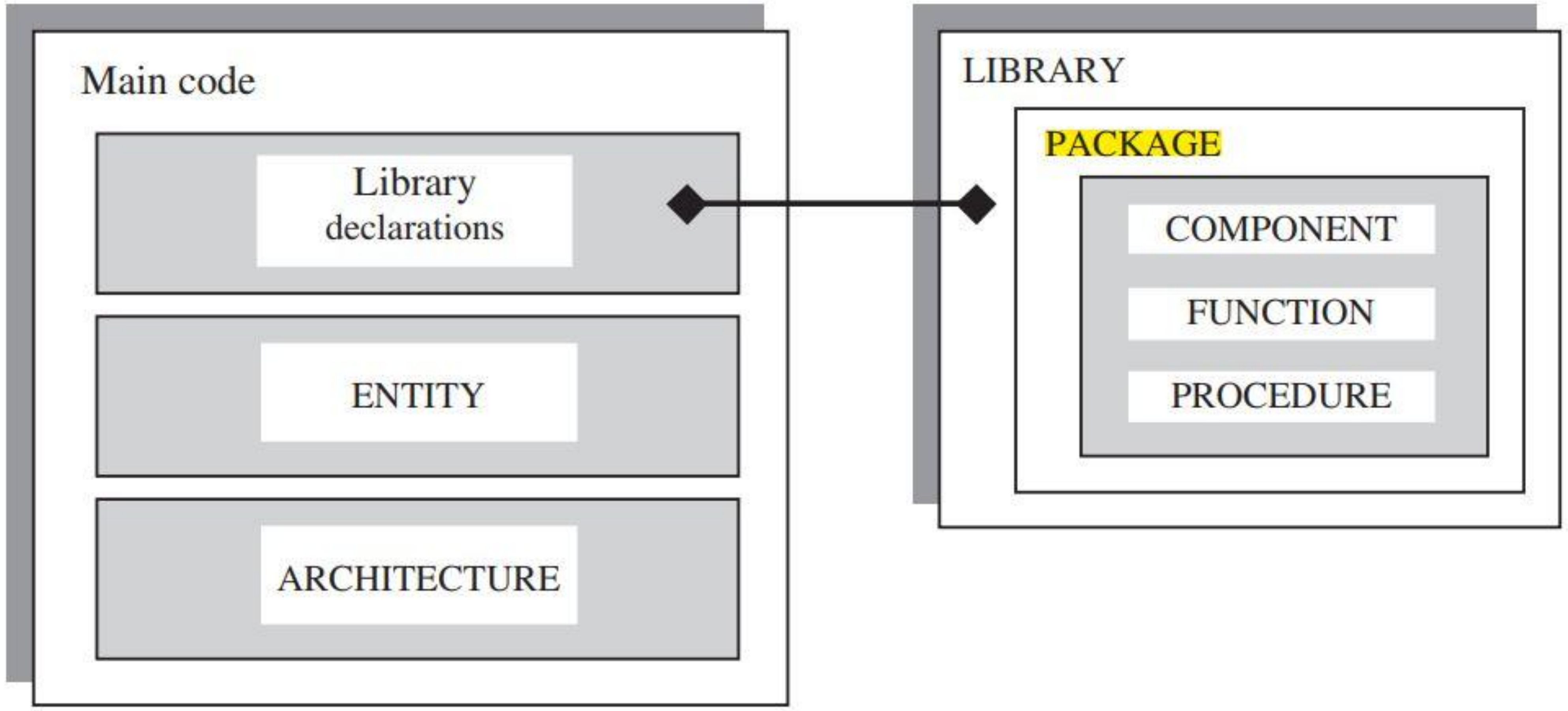


**Figure 10.1**
Fundamental units of VHDL code.

# Introduction

- Packages
- Components

Lecture 10 & 11

- Functions
- Procedures

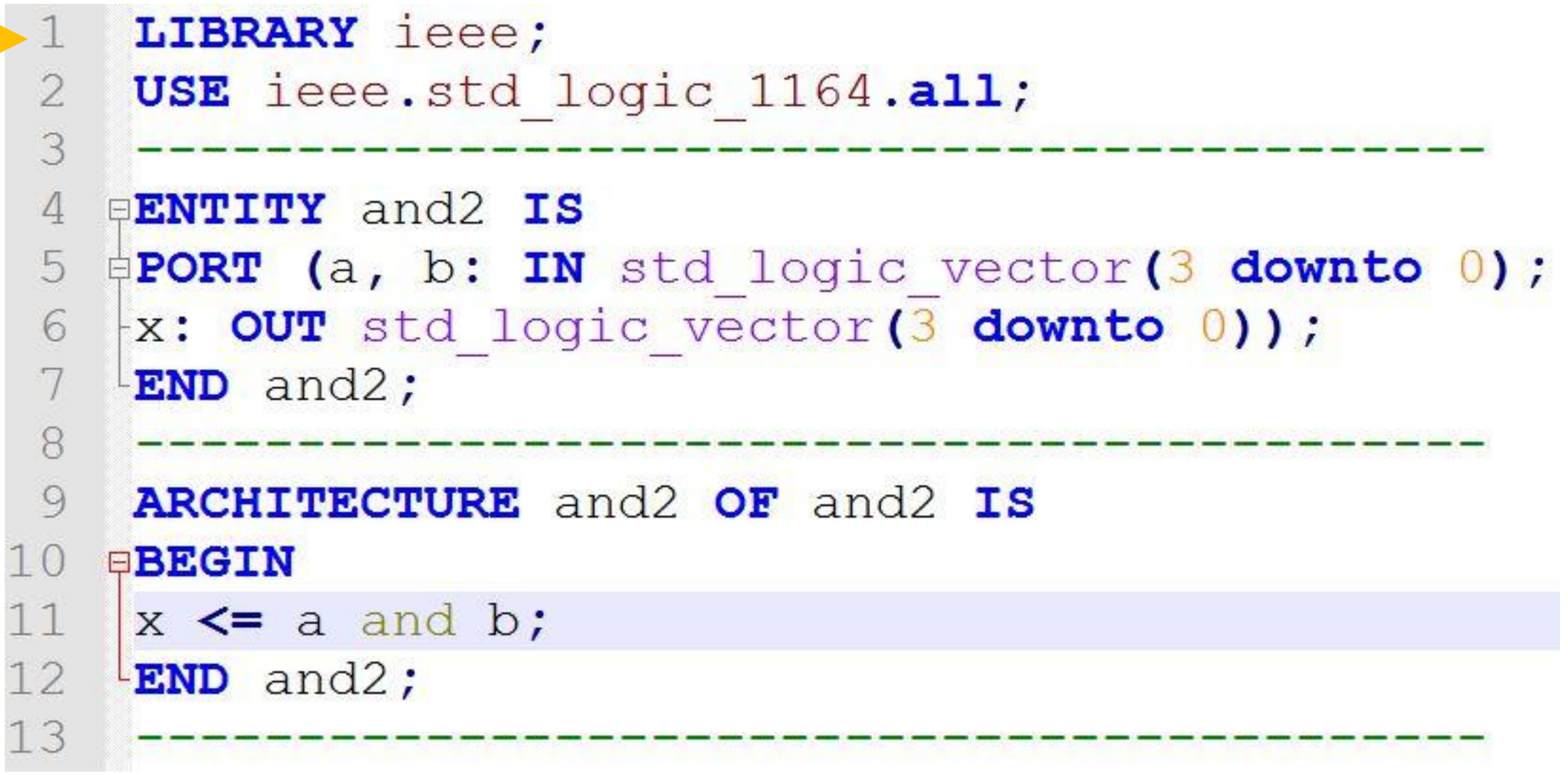
Lecture 12 & 13

## Introduction

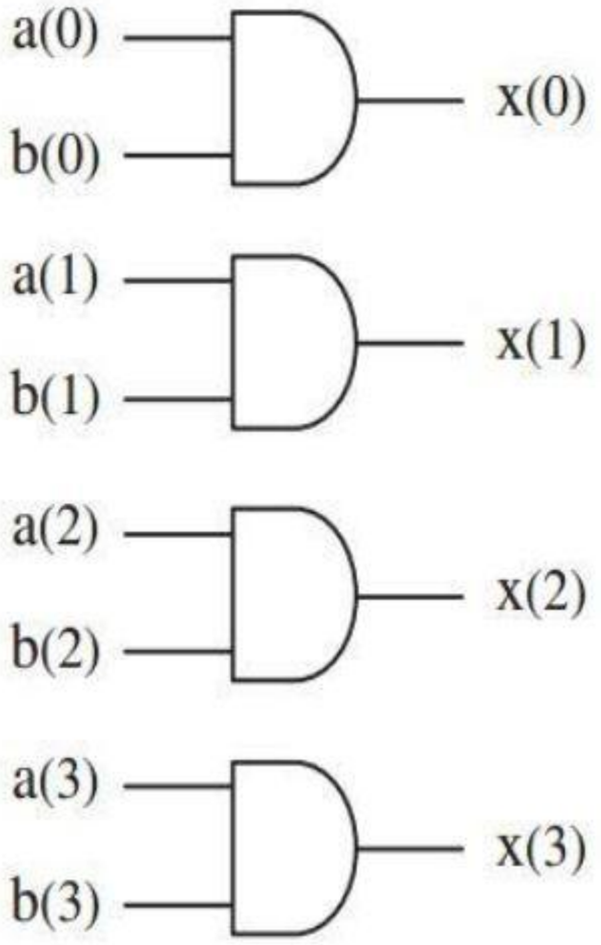
- These new units can be located in the main code itself (that is, on the left-hand side of figure 10.1)
- However, since their main purpose is to allow common pieces of code to be reused and shared, it is more usual to place them in a LIBRARY.
- This also leads to code partitioning, which is helpful when dealing with long codes.
- In summary, frequently used pieces of code can be written in the form of COMPONENTS, FUNCTIONS, or PROCEDURES, then placed in a **PACKAGE**, which is finally compiled into the destination LIBRARY.
- We have already seen that at least two LIBRARIES are generally needed in a design: **ieee**, and **work.**
- After studying lectures 10 to 13, we will be able to construct our own libraries, which can then be added to the list above.

# Introduction

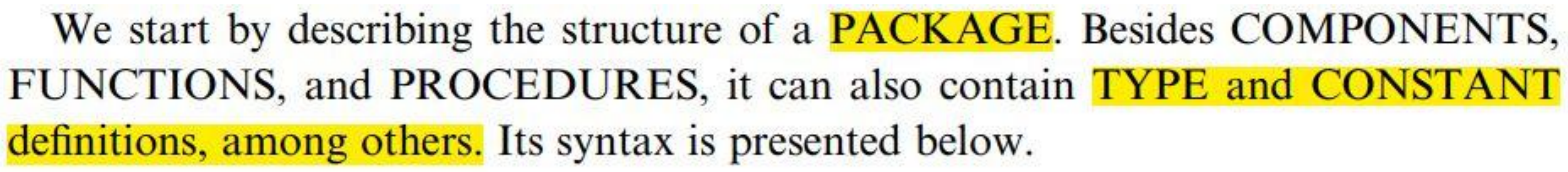
- For example, commonly used circuits, like flip-flops, multiplexers, adders, logic gates, etc., can be placed in a LIBRARY, so any project can make use of them without having to explicitly rewrite such codes.



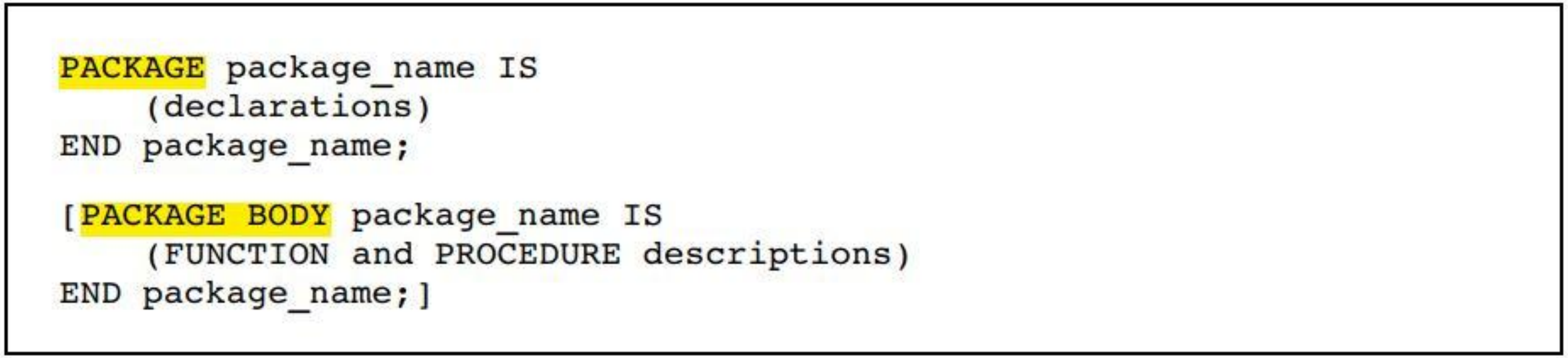
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY and2 IS
PORT (a, b: IN std_logic_vector(3 downto 0);
x: OUT std_logic_vector(3 downto 0));
END and2;
---------------------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
x <= a and b;
END and2;
---------------------------------------
```



# PACKAGE



We start by describing the structure of a PACKAGE. Besides COMPONENTS, FUNCTIONS, and PROCEDURES, it can also contain TYPE and CONSTANT definitions, among others. Its syntax is presented below.



```
PACKAGE package_name IS
    (declarations)
END package_name;

[PACKAGE BODY package_name IS
    (FUNCTION and PROCEDURE descriptions)
END package_name;]
```

As can be seen, the syntax is composed of two parts: PACKAGE and PACKAGE BODY

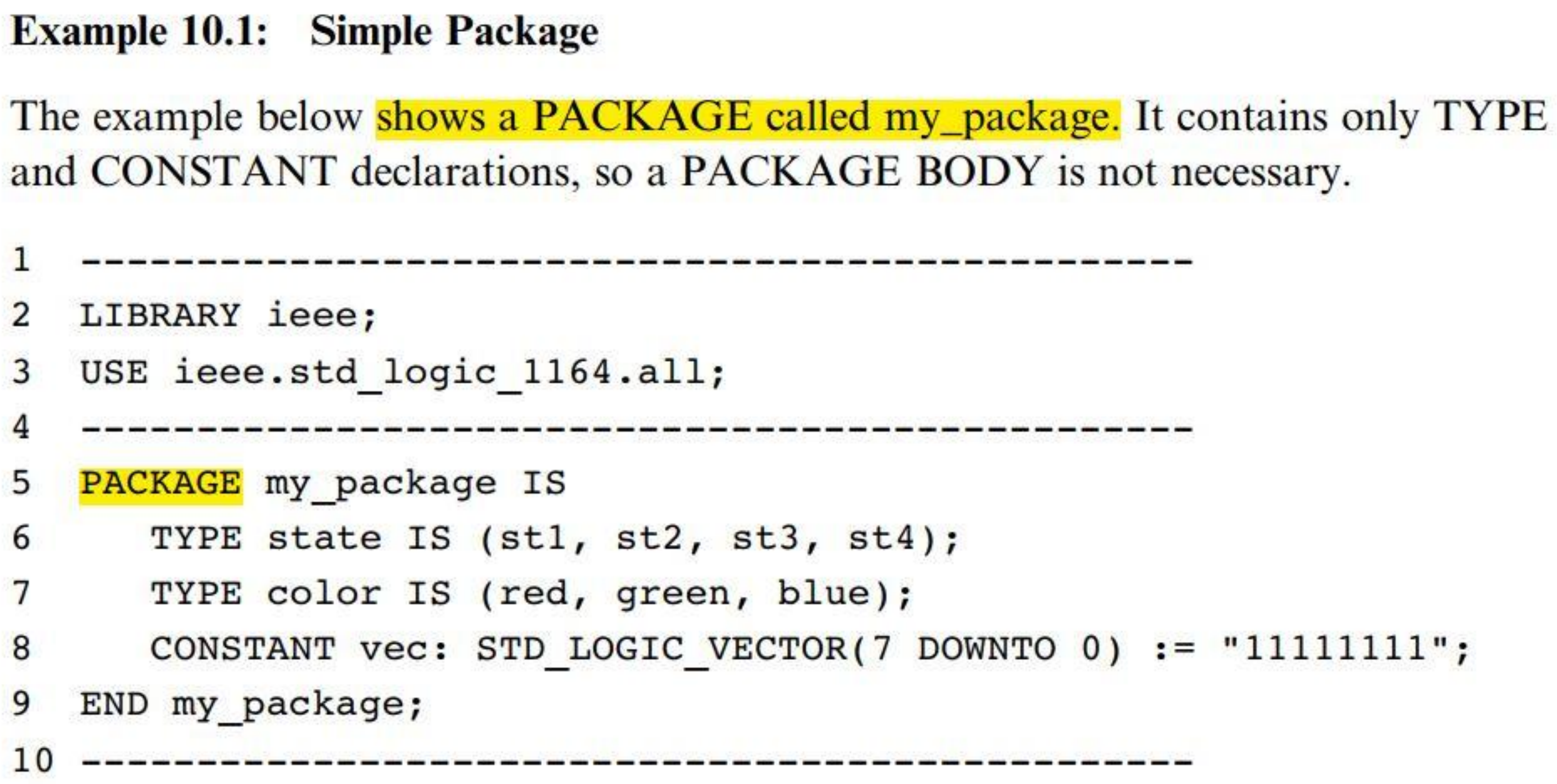
## PACKAGE

- The first part (PACKAGE) is mandatory and contains all declarations, while the second part (PACKAGE BODY) is necessary only when one or more subprograms (FUNCTION or PROCEDURE) are declared in the first upper part, in which case it must contain the descriptions (bodies) of the subprograms.

- PACKAGE and PACKAGE BODY must have the same name.

- The declarations list can contain the following: COMPONENT, FUNCTION, PROCEDURE, TYPE, CONSTANT, etc.

# PACKAGE



**Example 10.1: Simple Package**

The example below shows a PACKAGE called my_package. It contains only TYPE and CONSTANT declarations, so a PACKAGE BODY is not necessary.

```
------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------------------
PACKAGE my_package IS
   TYPE state IS (st1, st2, st3, st4);
   TYPE color IS (red, green, blue);
   CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
END my_package;
------------------------------------------------
```

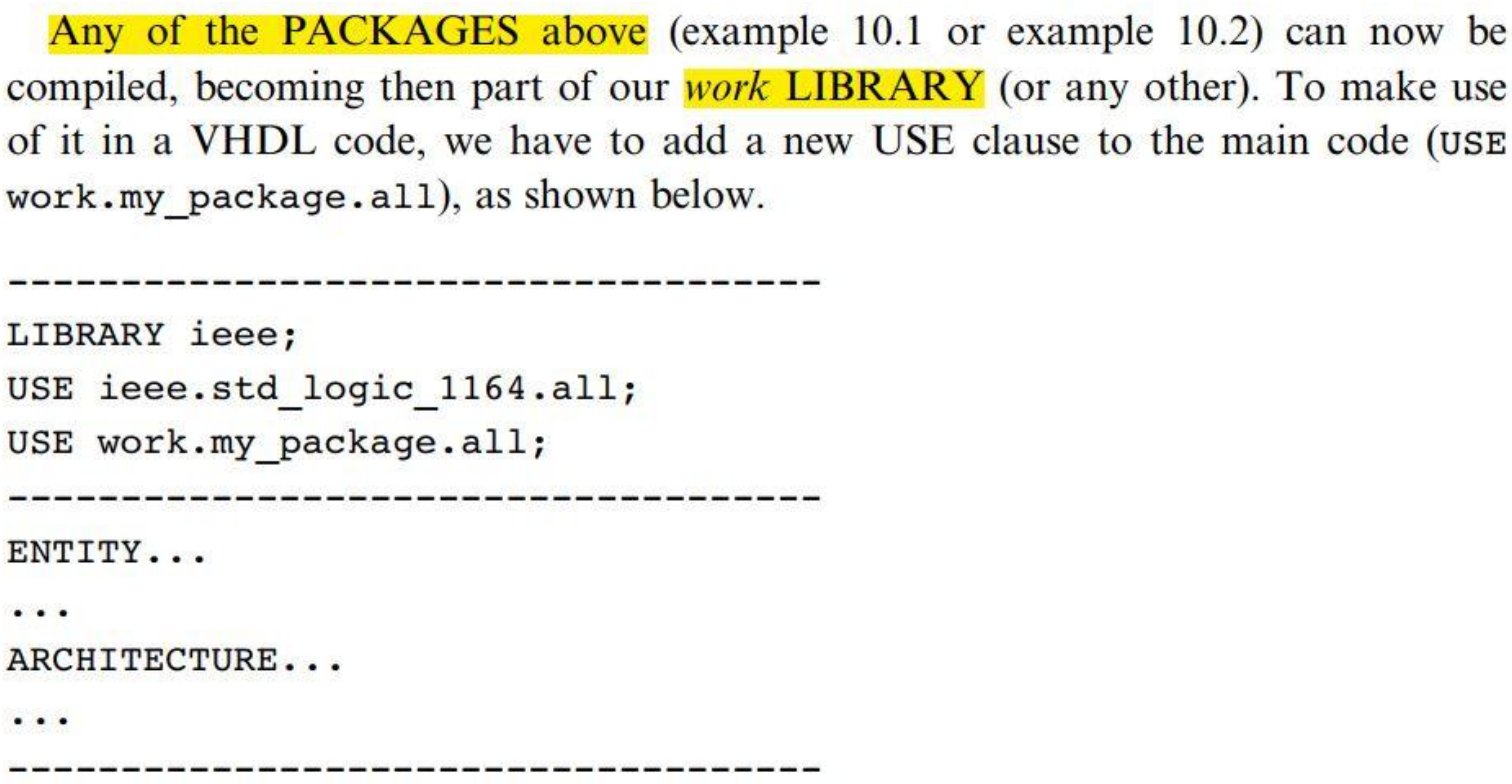
## PACKAGE

- The next example (example 10.2) contains, besides TYPE and CONSTANT declarations, a FUNCTION. Therefore, a PACKAGE BODY is now needed (details on how to write a FUNCTION will be seen in lecture 12). This function returns TRUE when a positive edge occurs on clk.

## PACKAGE

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
PACKAGE my_package IS
   TYPE state IS (st1, st2, st3, st4);
   TYPE color IS (red, green, blue);
   CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
   FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN;
END my_package;
---------------------------------------------
PACKAGE BODY my_package IS
   FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS
   BEGIN
      RETURN (s'EVENT AND s='1');
   END positive_edge;
END my_package;
---------------------------------------------
```

## PACKAGE



Any of the PACKAGES above (example 10.1 or example 10.2) can now be compiled, becoming then part of our *work* LIBRARY (or any other). To make use of it in a VHDL code, we have to add a new USE clause to the main code (`USE work.my_package.all`), as shown below.

```
-----------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-----------------------------------
ENTITY...
...
ARCHITECTURE...
...
-----------------------------------
```

## COMPONENT

- A **COMPONENT** is simply a piece of conventional code (that is, LIBRARY declarations + ENTITY + ARCHITECTURE, as seen in previous lectures).

- However, by declaring such code as being a **COMPONENT**, it can then be used within another circuit, thus allowing the construction of hierarchical designs.

- A **COMPONENT** is also another way of partitioning a code and providing code sharing and code reuse.

# COMPONENT

- To use (instantiate) a COMPONENT, it must first be declared. The corresponding syntaxes are shown below.

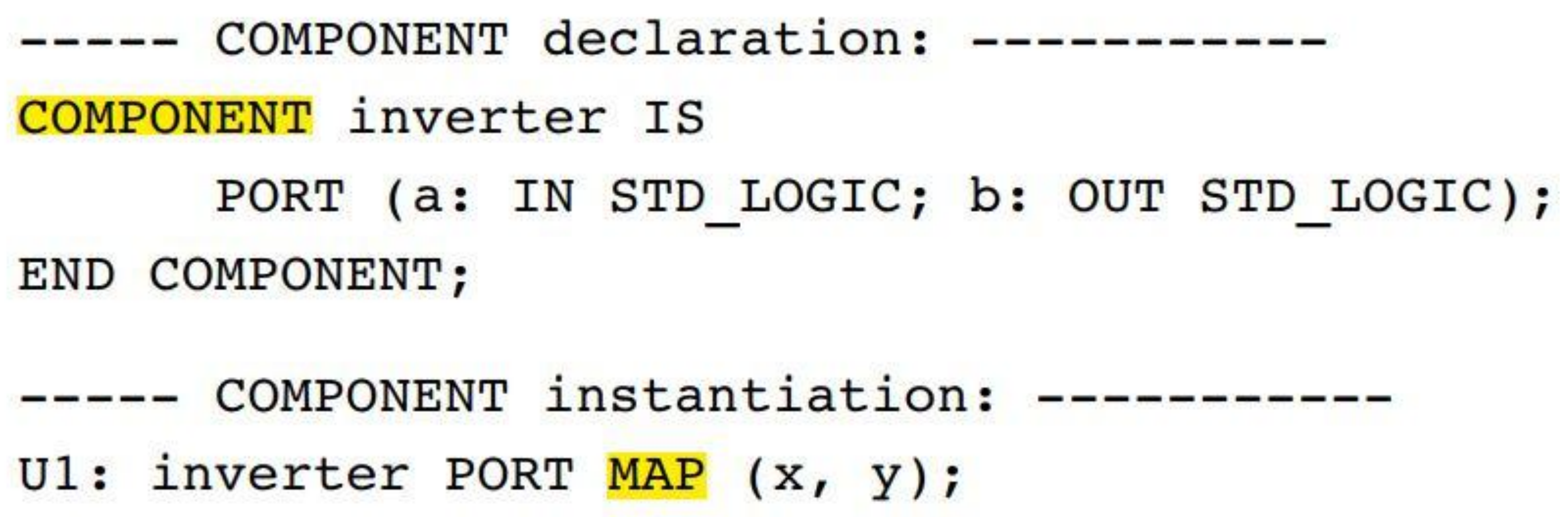
COMPONENT declaration:

```
COMPONENT component_name IS
    PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END COMPONENT;
```

COMPONENT instantiation:

```
label: component_name PORT MAP (port_list);
```

## COMPONENT



```
----- COMPONENT declaration: -----------
COMPONENT inverter IS
      PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;

----- COMPONENT instantiation: -----------
U1: inverter PORT MAP (x, y);
```

## COMPONENT

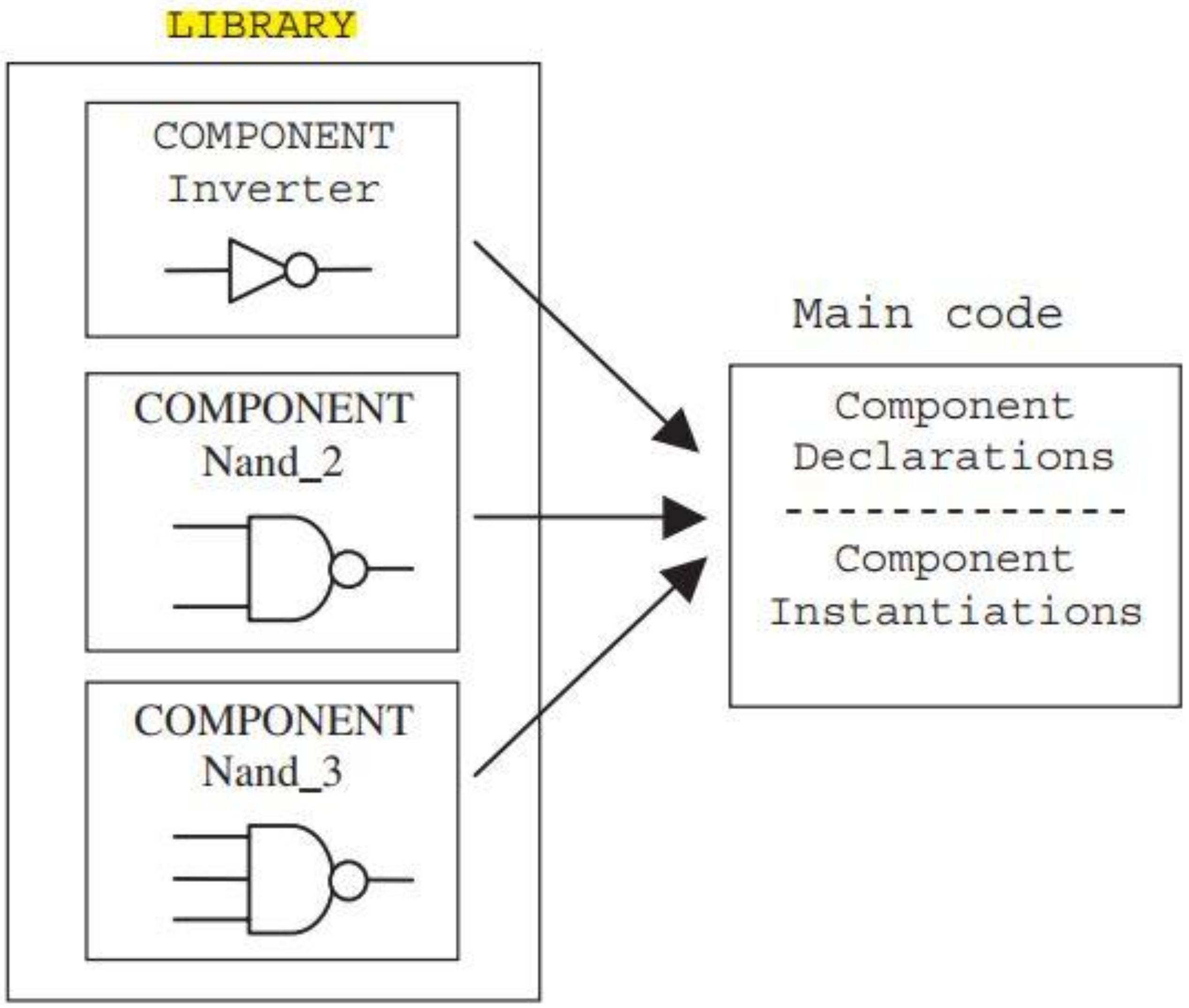


Figure 10.2 a. : Declarations in the main code itself

# COMPONENT

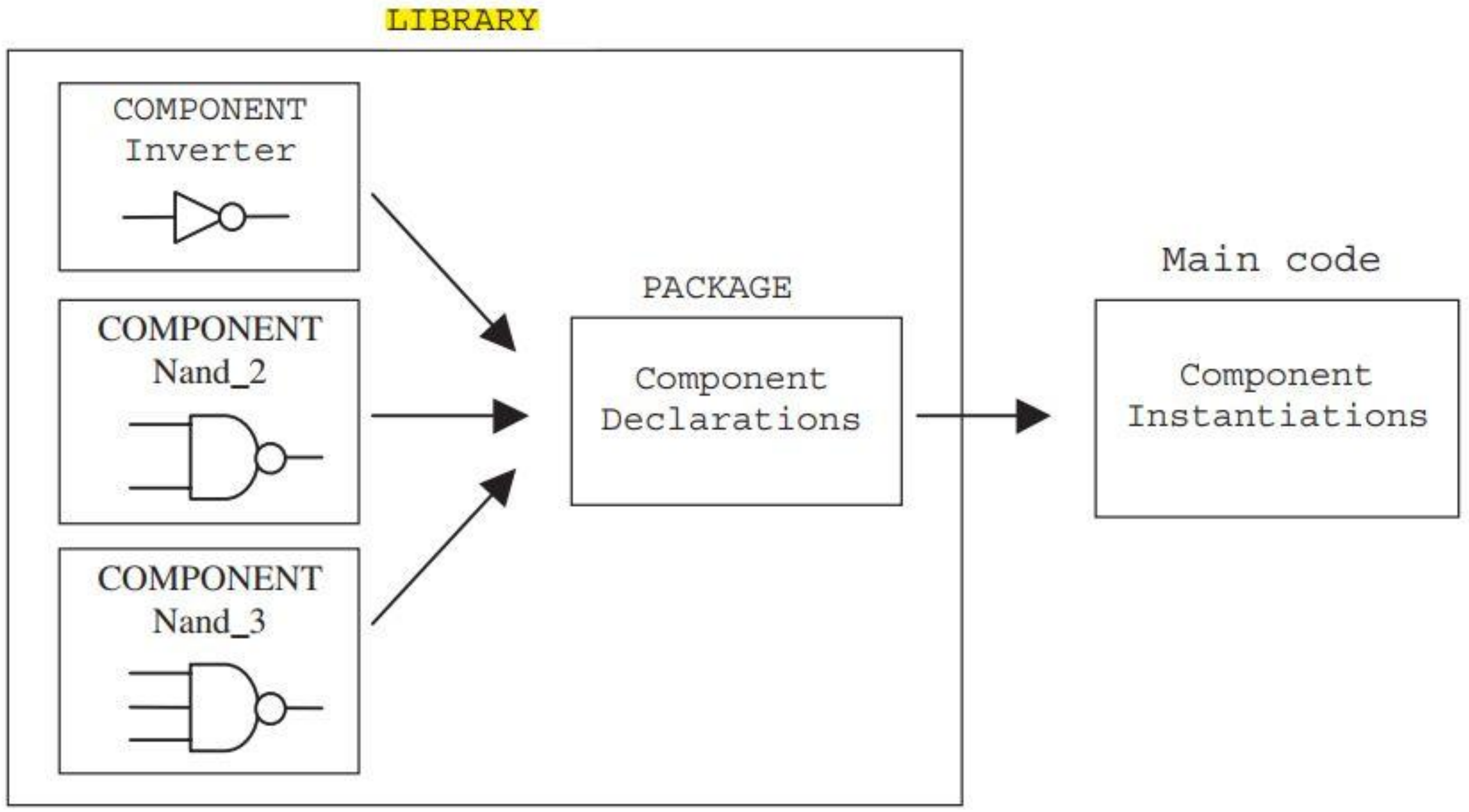
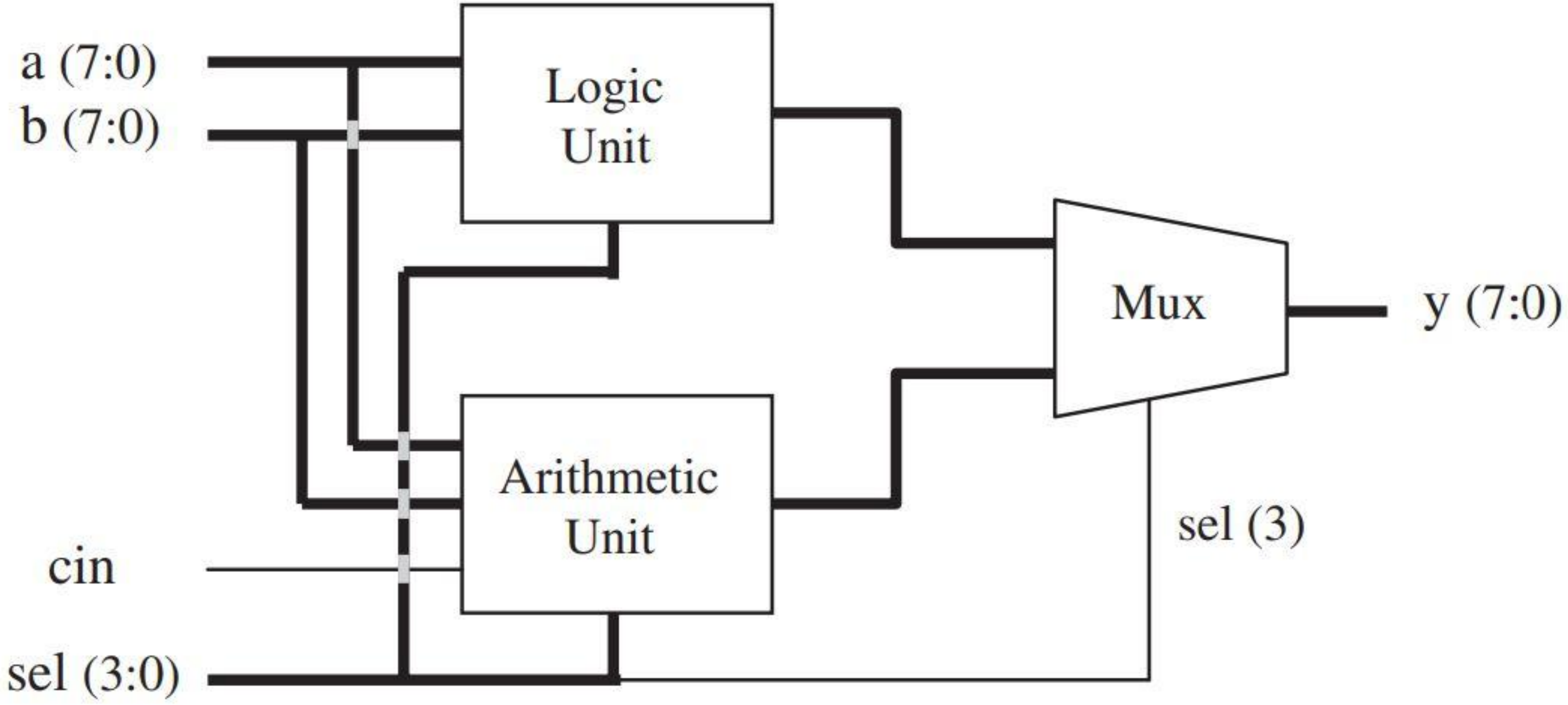
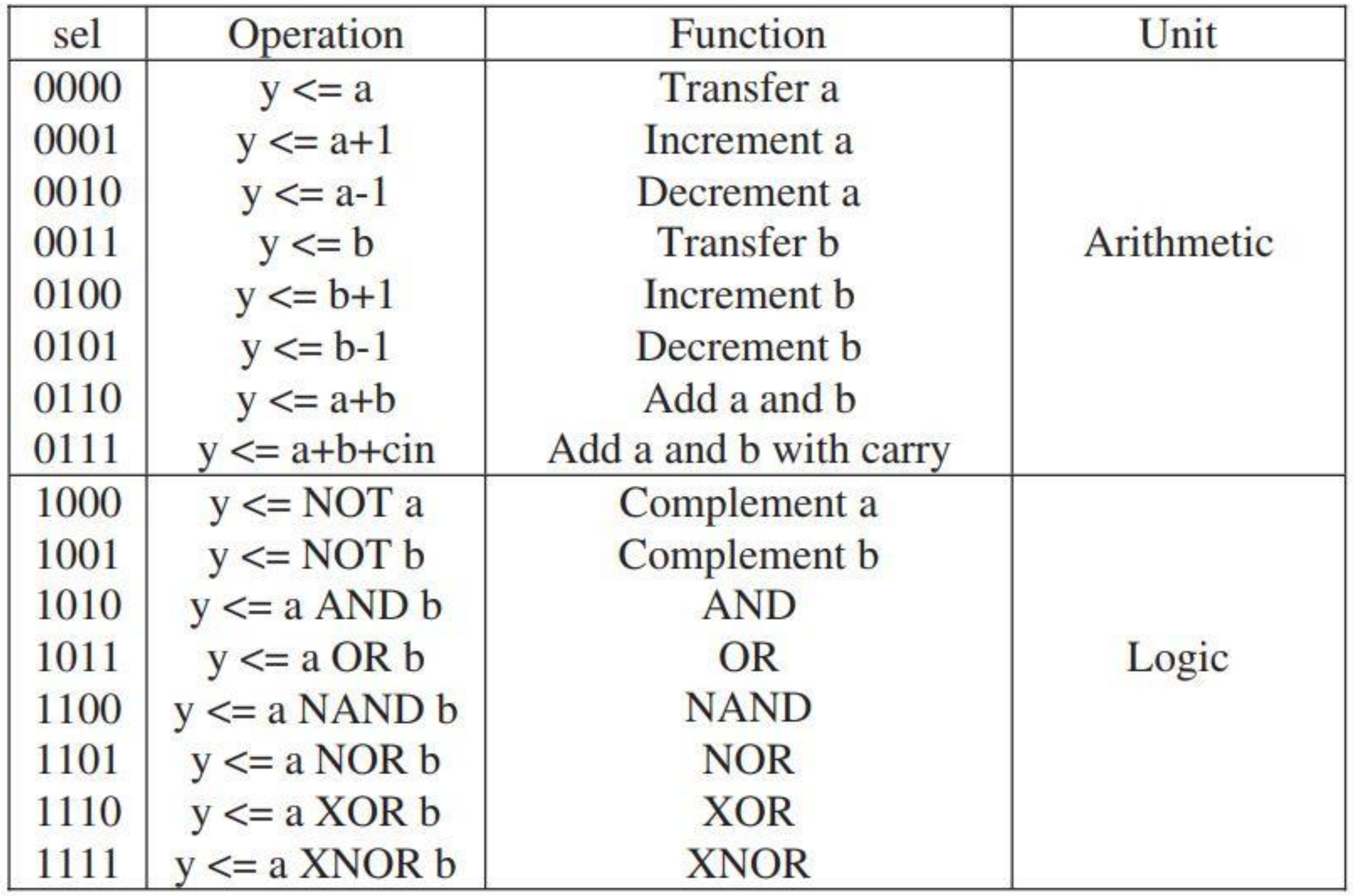


Figure 10.2 b. : Declarations in a PACKAGE

# ALU Example using COMPONENTS



# ALU Example using COMPONENTS

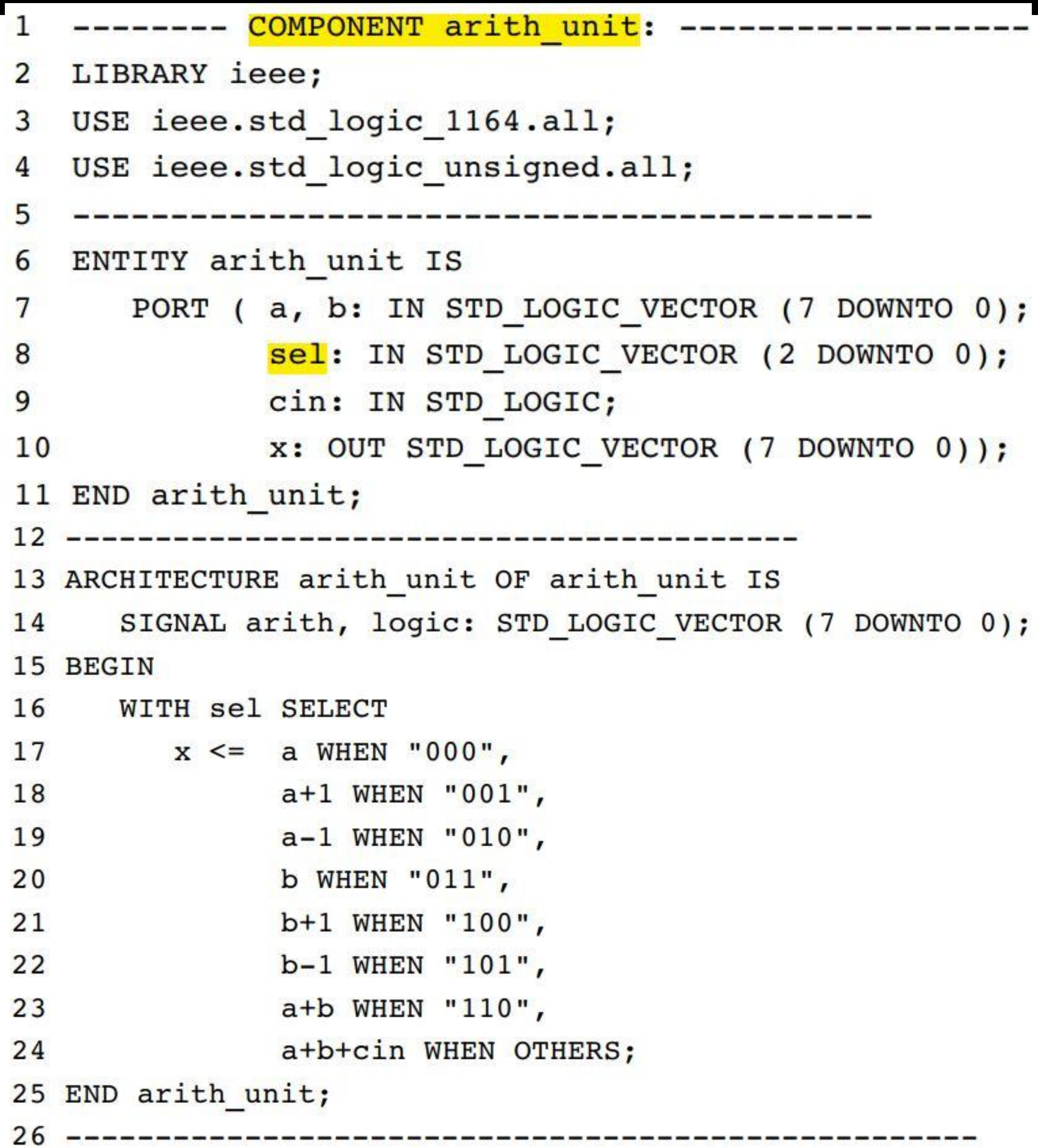


| sel | Operation | Function | Unit |
|---|---|---|---|
| 0000 | y <= a | Transfer a | Arithmetic |
| 0001 | y <= a+1 | Increment a | |
| 0010 | y <= a-1 | Decrement a | |
| 0011 | y <= b | Transfer b | |
| 0100 | y <= b+1 | Increment b | |
| 0101 | y <= b-1 | Decrement b | |
| 0110 | y <= a+b | Add a and b | |
| 0111 | y <= a+b+cin | Add a and b with carry | |
| 1000 | y <= NOT a | Complement a | Logic |
| 1001 | y <= NOT b | Complement b | |
| 1010 | y <= a AND b | AND | |
| 1011 | y <= a OR b | OR | |
| 1100 | y <= a NAND b | NAND | |
| 1101 | y <= a NOR b | NOR | |
| 1110 | y <= a XOR b | XOR | |
| 1111 | y <= a XNOR b | XNOR | |

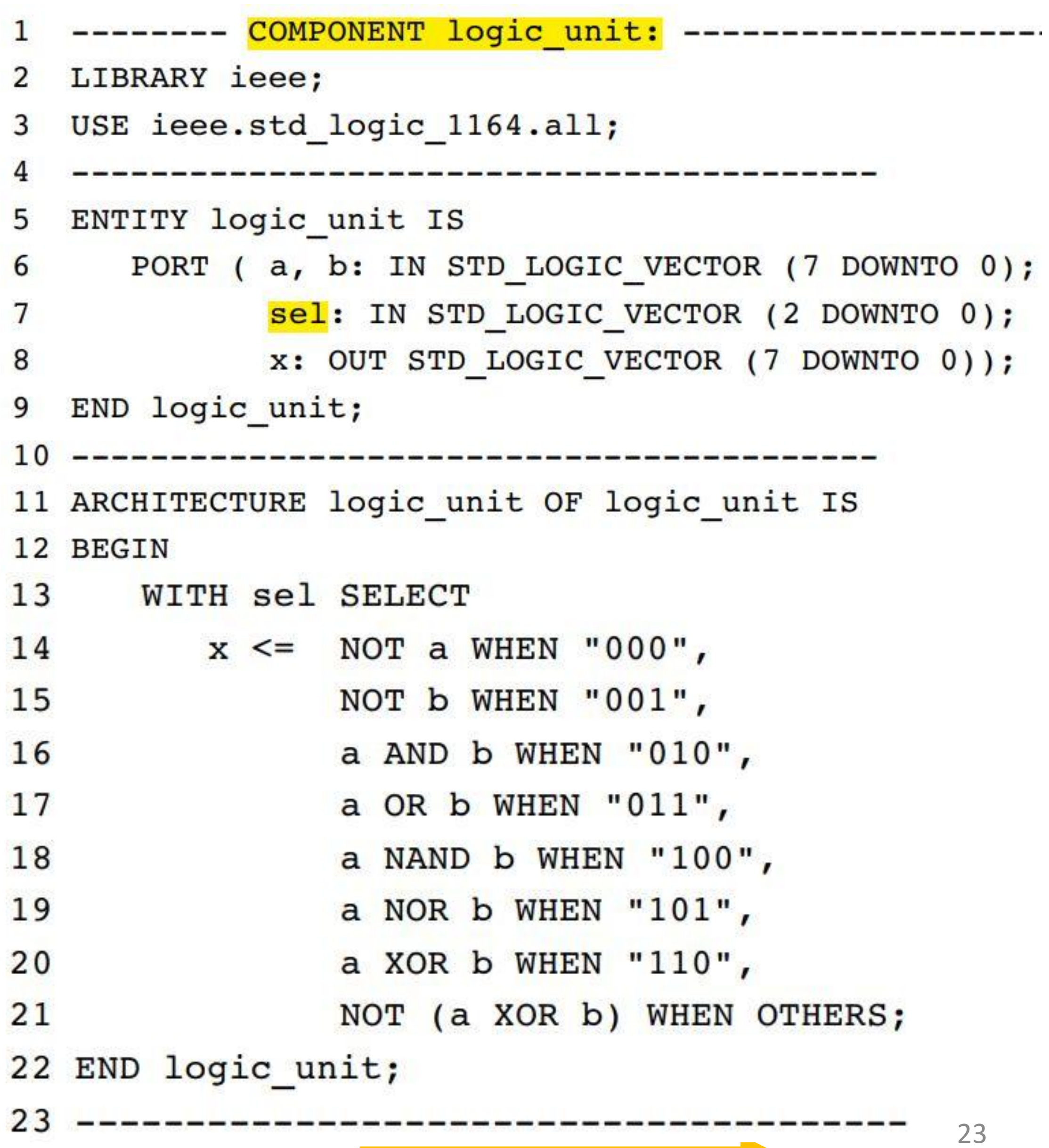
## تنفيذ مباشر في فايل واحد ALU

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
----------------------------------------------
ENTITY ALU IS
   PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
             sel: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
             cin: IN STD_LOGIC;
             y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END ALU;
----------------------------------------------
ARCHITECTURE dataflow OF ALU IS
   SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
   ----- Arithmetic unit: ------
   WITH sel(2 DOWNTO 0) SELECT
      arith <=  a WHEN "000",
                   a+1 WHEN "001",
                   a-1 WHEN "010",
                   b WHEN "011",
                   b+1 WHEN "100",
                   b-1 WHEN "101",
                   a+b WHEN "110",
                   a+b+cin WHEN OTHERS;
   ----- Logic unit: -----------
   WITH sel(2 DOWNTO 0) SELECT
      logic <=   NOT a WHEN "000",
                 NOT b WHEN "001",
                 a AND b WHEN "010",
                 a OR b WHEN "011",
                 a NAND b WHEN "100",
                 a NOR b WHEN "101",
                 a XOR b WHEN "110",
                 NOT (a XOR b) WHEN OTHERS;
   -------- Mux: ---------------
   WITH sel(3) SELECT
      y <=   arith WHEN '0',
             logic WHEN OTHERS;
END dataflow;
------------------------------------------
```

# ALU Example using COMPONENTS

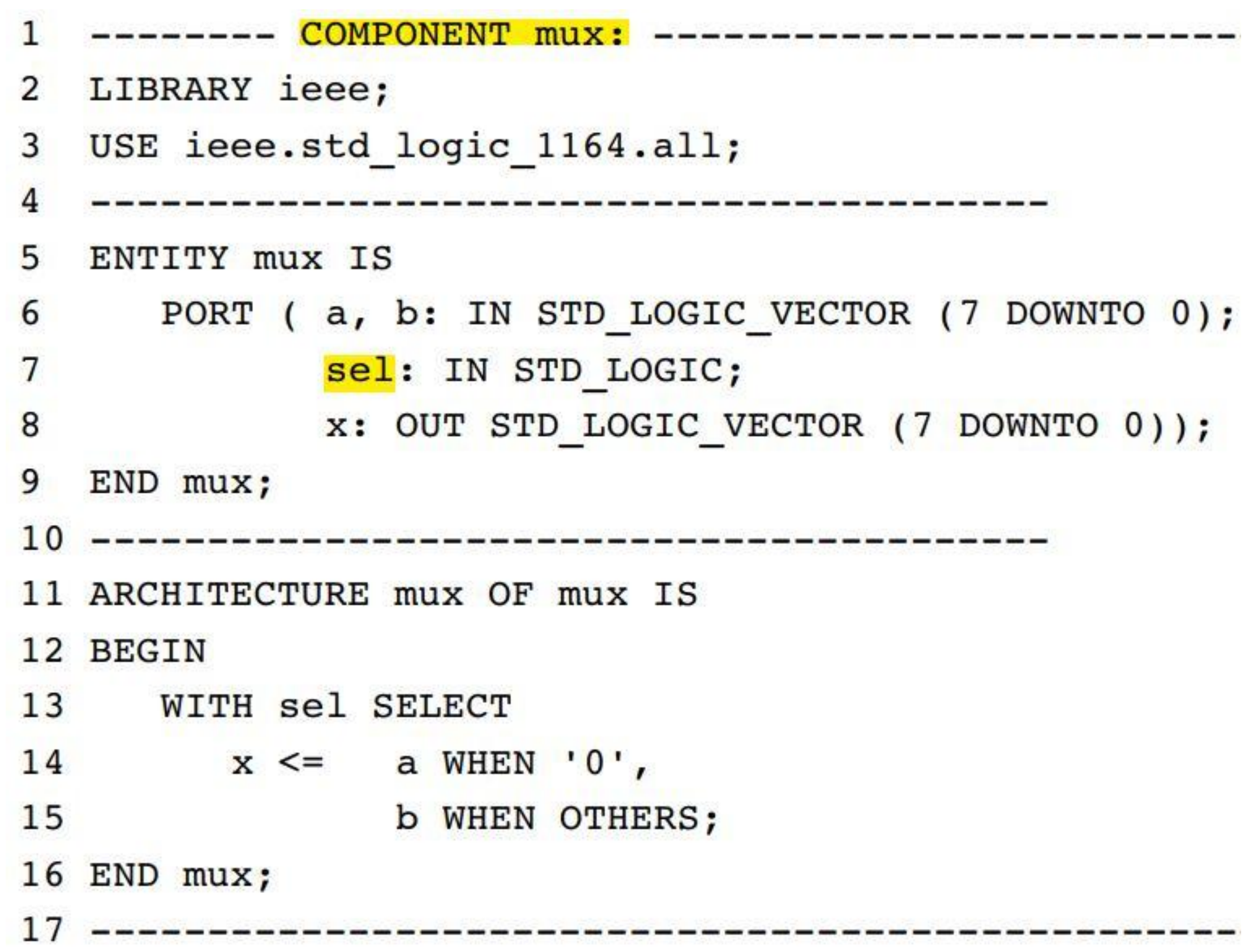


```
-------- COMPONENT arith_unit: ------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
-----------------------------------------
ENTITY arith_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
          cin: IN STD_LOGIC;
          x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END arith_unit;
-----------------------------------------
ARCHITECTURE arith_unit OF arith_unit IS
   SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
   WITH sel SELECT
      x <=  a WHEN "000",
            a+1 WHEN "001",
            a-1 WHEN "010",
            b WHEN "011",
            b+1 WHEN "100",
            b-1 WHEN "101",
            a+b WHEN "110",
            a+b+cin WHEN OTHERS;
END arith_unit;
-----------------------------------------------------
```



```
-------- COMPONENT logic_unit: ------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY logic_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
          x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END logic_unit;
-----------------------------------------
ARCHITECTURE logic_unit OF logic_unit IS
BEGIN
   WITH sel SELECT
      x <=  NOT a WHEN "000",
            NOT b WHEN "001",
            a AND b WHEN "010",
            a OR b WHEN "011",
            a NAND b WHEN "100",
            a NOR b WHEN "101",
            a XOR b WHEN "110",
            NOT (a XOR b) WHEN OTHERS;
END logic_unit;
-----------------------------------------
```

# ALU Example using COMPONENTS



```
-------- COMPONENT mux: -------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY mux IS
   PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          sel: IN STD_LOGIC;
          x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END mux;
-----------------------------------------
ARCHITECTURE mux OF mux IS
BEGIN
   WITH sel SELECT
      x <=    a WHEN '0',
             b WHEN OTHERS;
END mux;
-------------------------------------------------
```

# ALU Example using COMPONENTS

```
-------- Project ALU (main code): --------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY alu IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          cin: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END alu;
-----------------------------------------
ARCHITECTURE alu OF alu IS
-----------------------
   COMPONENT arith_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          cin: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   END COMPONENT;
   -----------------------
   COMPONENT logic_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   END COMPONENT;
   -----------------------
   COMPONENT mux IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          sel: IN STD_LOGIC;
          x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   END COMPONENT;
   -----------------------
   SIGNAL x1, x2: STD_LOGIC_VECTOR(7 DOWNTO 0);
-----------------------
BEGIN
   U1: arith_unit PORT MAP (a, b, cin, sel(2 DOWNTO 0), x1);
   U2: logic_unit PORT MAP (a, b, sel(2 DOWNTO 0), x2);
   U3: mux PORT MAP (x1, x2, sel(3), y);
END alu;
```

# ALU Example using COMPONENTS

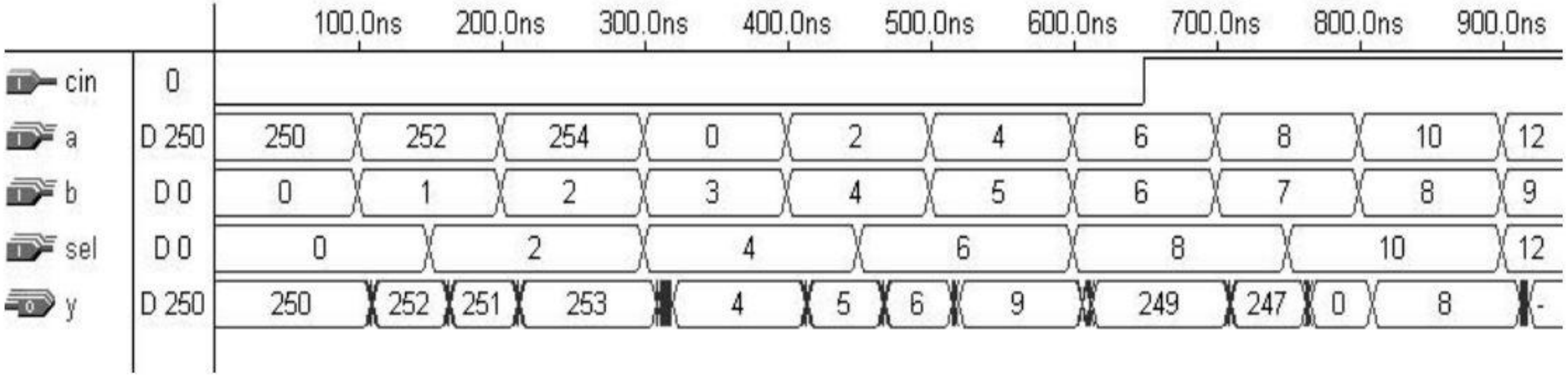


**Figure 10.8**
Simulation results of example 10.6.

## Assignments

- The assignments will be attached to your class room.

**End of lecture 11**
**Any Questions ?**