Circuit Design with VHDL

Chapter 14: Packages and Subprograms

Instructor: Ali Jahanian

Outline

II-SYSTEM DESIGN

10 Packages and Components

10.1 Introduction

10.2 PACKAGE

10.3 COMPONENT

10.4 PORT MAP

10.5 GENERIC MAP

10.6 Problems

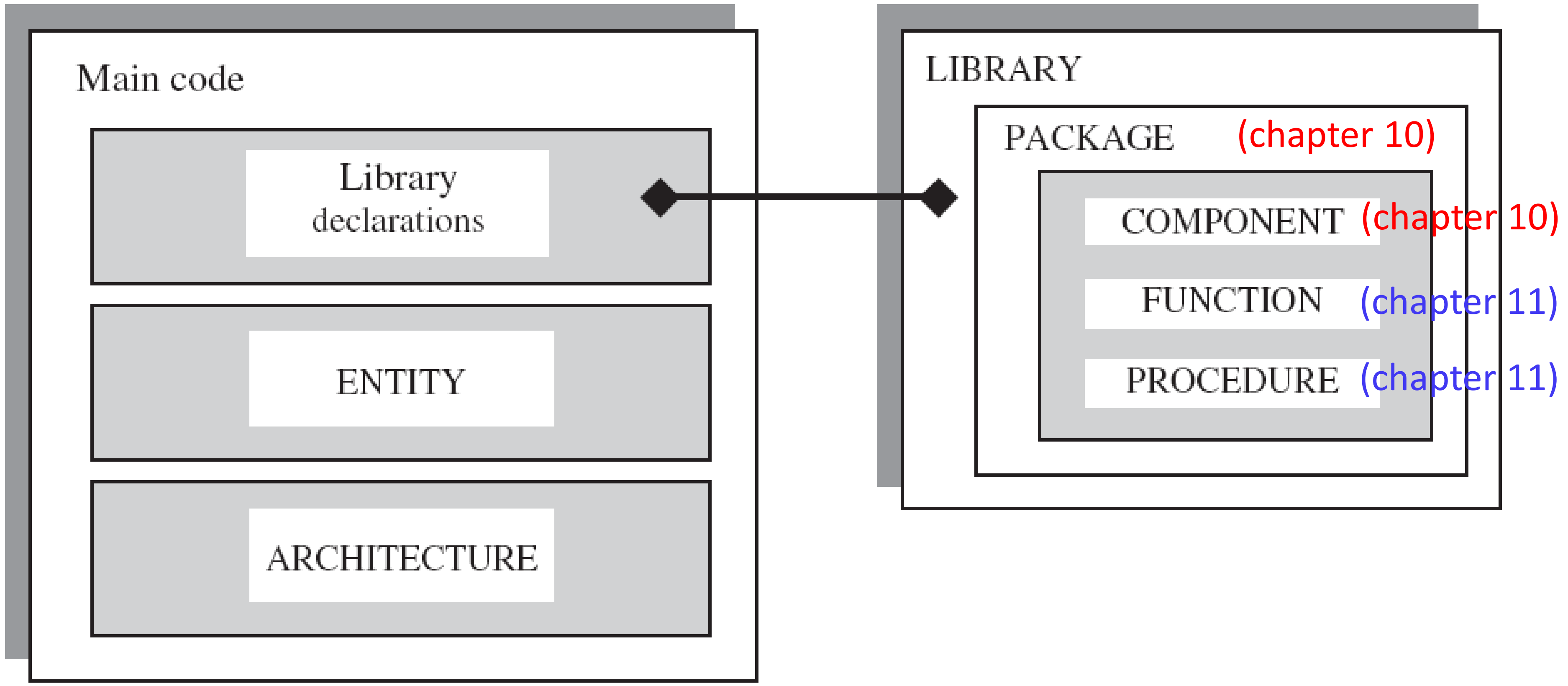


Figure 10.1
Fundamental units of VHDL code.

PACKAGE

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

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

- The first part is mandatory and contains all declarations.
  - The declarations list can contain the following: COMPONENT, FUNCTION, PROCEDURE, TYPE, CONSTANT, etc.

Example 10.1: Simple 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";
END my_package;
------------------------------------------------
```

Example 10.2: Package with a Function

```
----------------------------------------------------
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;
----------------------------------------------------
```

To make use of my_pachage

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

COMPONENT

- It’s simply a piece of conventional code that allowing the construction of hierarchical designs.
- It’s also another way of partitioning a code and providing code sharing and code reuse.

COMPONENT

- 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);
```

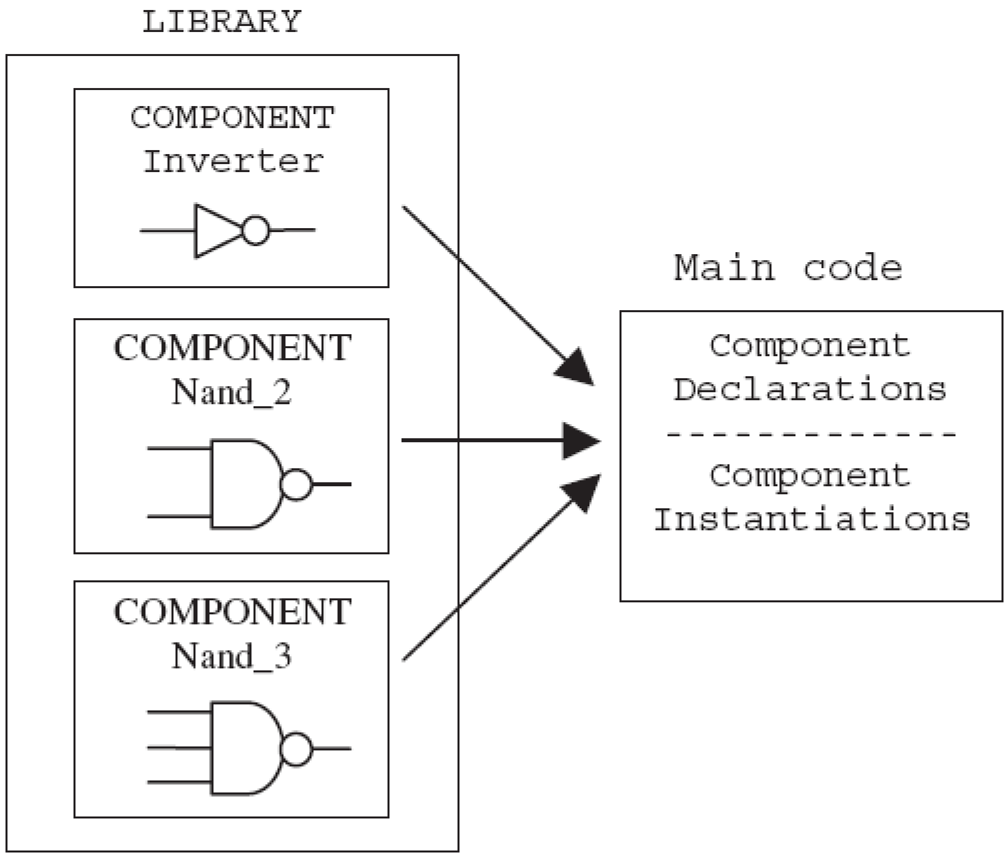
Example Component declaration & instaniation

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

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

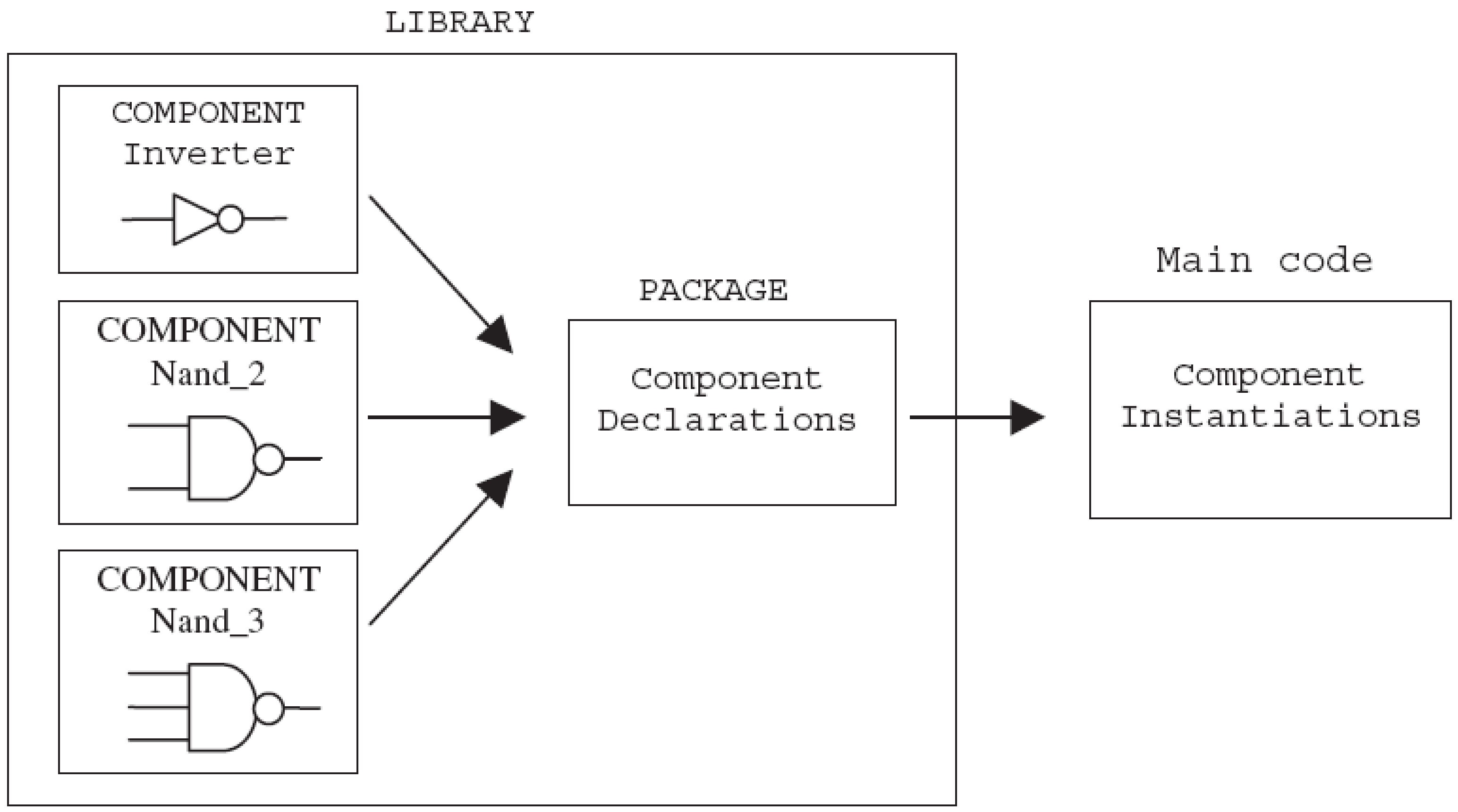
Basic ways of declaring COMPONENTS

- declarations in the main code itself



Basic ways of declaring COMPONENTS …

- declarations in a PACKAGE



Example 10.3: Components Declared in the Main Code

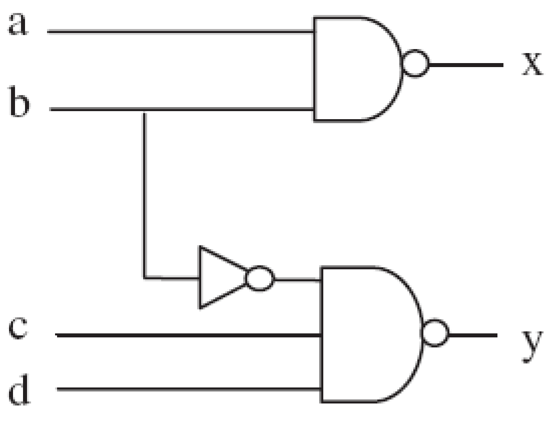
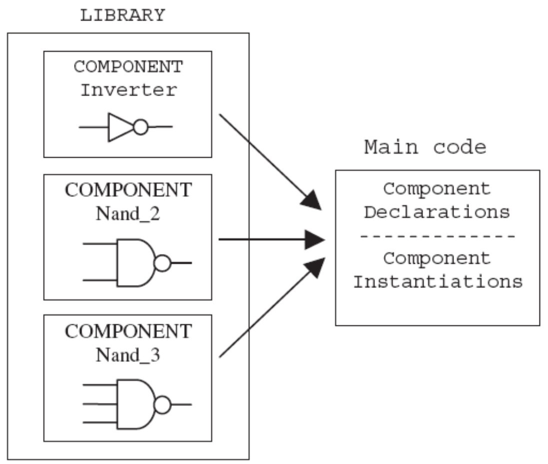


Figure 10.3
Circuit of example 10.3.



```
----- File project.vhd: ----------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY project IS
   PORT (a, b, c, d: IN STD_LOGIC;
               x, y: OUT STD_LOGIC);
END project;
------------------------------------
```

Example 10.3 …

```
ARCHITECTURE structural OF project IS
   -------------
   COMPONENT inverter IS
      PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
   END COMPONENT;
   -------------
   COMPONENT nand_2 IS
      PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
   END COMPONENT;
   -------------
   COMPONENT nand_3 IS
      PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
   END COMPONENT;
   -------------
   SIGNAL w: STD_LOGIC;
BEGIN
   U1: inverter PORT MAP (b, w);
   U2: nand_2 PORT MAP (a, b, x);
   U3: nand_3 PORT MAP (w, c, d, y);
END structural;
---------------------------------------------
```

Example 10.3 …(inverter.vhd)

```
------ File inverter.vhd: --------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END inverter;
------------------------------------
ARCHITECTURE inverter OF inverter IS
BEGIN
   b <= NOT a;
END inverter;
---------------------------------------------
```

Example 10.3 … (nand_2.vhd)

```
------ File nand_2.vhd: ----------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY nand_2 IS
   PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
END nand_2;
------------------------------------
ARCHITECTURE nand_2 OF nand_2 IS
BEGIN
   c <= NOT (a AND b);
END nand_2;
---------------------------------------------
```

Example 10.3 … (nand_3.vhd)

```vhdl
----- File nand_3.vhd: ----------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY nand_3 IS
   PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
END nand_3;
------------------------------------
ARCHITECTURE nand_3 OF nand_3 IS
BEGIN
   d <= NOT (a AND b AND c);
END nand_3;
---------------------------------------------
```

Example 10.3 …

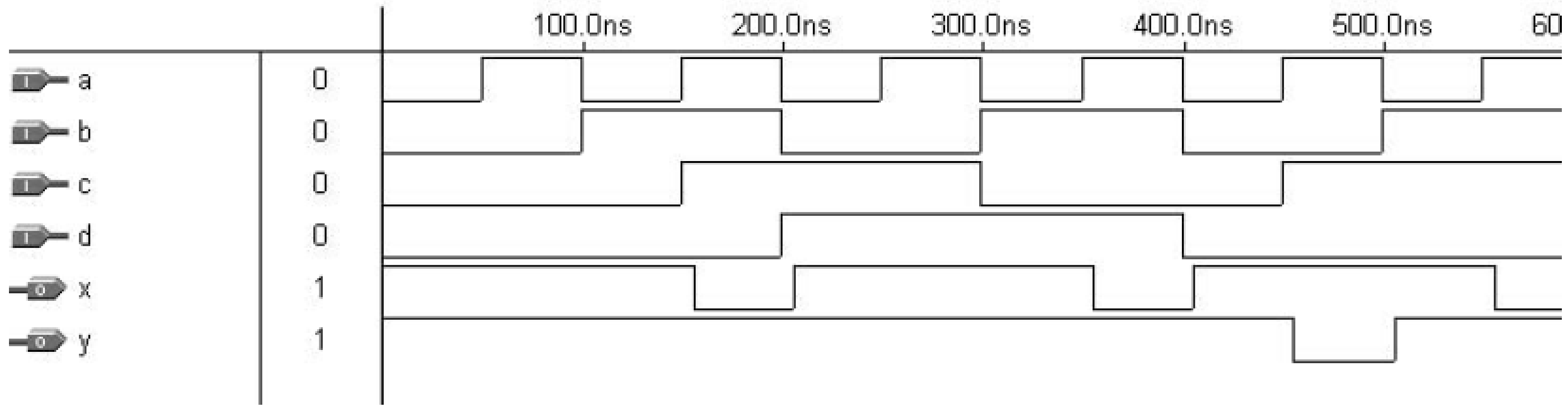


Figure 10.4
Experimental results of example 10.3.

Example 10.4: Components Declared in a Package

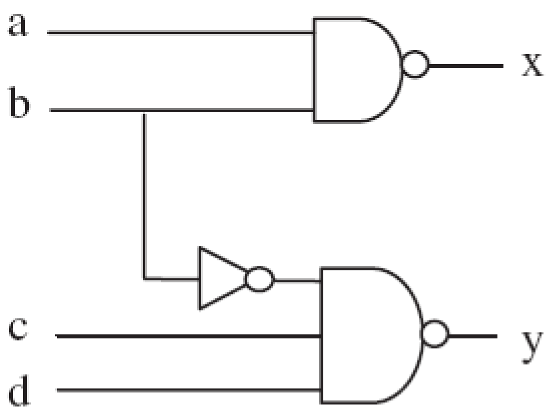
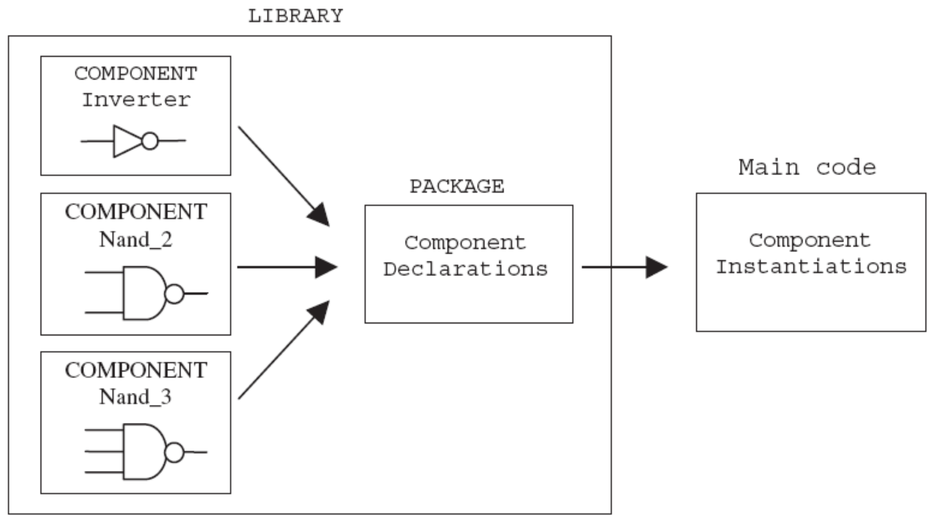


Figure 10.3
Circuit of example 10.3.



Example 10.4 …

```vhdl
------ File inverter.vhd: --------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------
ENTITY inverter IS
    PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END inverter;
------------------------------
ARCHITECTURE inverter OF inverter IS
BEGIN
    b <= NOT a;
END inverter;
-----------------------------------------
```

Example 10.4 …

```
------ File nand_2.vhd: ---------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY nand_2 IS
   PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
END nand_2;
------------------------------------
ARCHITECTURE nand_2 OF nand_2 IS
BEGIN
   c <= NOT (a AND b);
END nand_2;
---------------------------------------------
```

Example 10.4 …

```
----- File nand_3.vhd: ----------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY nand_3 IS
   PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
END nand_3;
------------------------------------
ARCHITECTURE nand_3 OF nand_3 IS
BEGIN
   d <= NOT (a AND b AND c);
END nand_3;
---------------------------------------------
```

Example 10.4 …

```
----- File my_components.vhd: ---------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------
PACKAGE my_components IS
   ------ inverter: -------
   COMPONENT inverter IS
      PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
   END COMPONENT;
   ------ 2-input nand: ---
   COMPONENT nand_2 IS
      PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
   END COMPONENT;
   ------ 3-input nand: ---
   COMPONENT nand_3 IS
      PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
   END COMPONENT;
   ------------------------
END my_components;
---------------------------------------------
```

Example 10.4 …

```
----- File project.vhd: ----------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_components.all;
----------------------------------
ENTITY project IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          x, y: OUT STD_LOGIC);
END project;
----------------------------------
ARCHITECTURE structural OF project IS
   SIGNAL w: STD_LOGIC;
BEGIN
   U1: inverter PORT MAP (b, w);
   U2: nand_2 PORT MAP (a, b, x);
   U3: nand_3 PORT MAP (w, c, d, y);
END structural;
----------------------------------------------
```

PORT MAP

- There are two ways:
  - positional mapping
  - nominal mapping

```
COMPONENT inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;
...
U1: inverter PORT MAP (x, y);

U1: inverter PORT MAP (x=>a, y=>b);
```

Positional mapping is easier to write, but nominal mapping is less error-prone.

Ports can also be left unconnected (using the keyword OPEN). For example:

```
U2: my_circuit PORT MAP (x=>a, y=>b, w=>OPEN, z=>d);
```

GENERIC MAP

```
label: compon_name GENERIC MAP (param. list) PORT MAP (port list);
```

Introduction

- FUNCTIONS and PROCEDURES are collectively called ***subprograms***.
  - From a construction point of view
    - they are very similar to a PROCESS (sequential code).
  - From the applications point of view
    - there is a fundamental difference between a PROCESS and a FUNCTION or PROCEDURE.
      - PROCESS: for immediate use in the main code
      - FUNCTIONS and PROCEDURES: mainly for LIBRARY allocation.

FUNCTION

- Its purpose:
  - data type conversions,
  - logical operations,
  - arithmetic computations,
  - new operators and attributes.
- By writing such code as a FUNCTION
  - it can be shared and reused,
  - propitiating the main code to be shorter and easier to understand.
- prohibitions in a function:
  - WAIT
  - SIGNAL declarations
  - COMPONENT instantiations.
- Two necessary parts of a function:
  - function body
  - A function call.

Function Body

```
FUNCTION function_name [<parameter list>] RETURN data_type IS
    [declarations]
BEGIN
    (sequential statements)
END function_name;
```

⟨parameter list⟩ = [CONSTANT] constant_name: constant_type; or

⟨parameter list⟩ = SIGNAL signal_name: signal_type;

(VARIABLES are not allowed)

(No range specification)

```
FUNCTION f1 (a, b: INTEGER; SIGNAL c: STD_LOGIC_VECTOR)
    RETURN BOOLEAN IS
BEGIN
    (sequential statements)
END f1;
```

(the word “CONSTANT” can be omitted)

Function Call

```
x <= conv_integer(a);          -- converts a to an integer
                               -- (expression appears by itself)
y <= maximum(a, b);            -- returns the largest of a and b
                               -- (expression appears by itself)
IF x > maximum(a, b) ...       -- compares x to the largest of a, b
                               -- (expression associated to a
                               -- statement)
```

Example 11.1: Function positive_edge( )

```
------ Function body: --------------------------------
FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS
BEGIN
   RETURN (s'EVENT AND s='1');
END positive_edge;
------ Function call: --------------------------------
...
IF positive_edge(clk) THEN...
...
-----------------------------------------------------
```

Example 11.2: Function conv_integer( )

```
------ Function body: -------------------------------
FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
      RETURN INTEGER IS
   VARIABLE result: INTEGER RANGE 0 TO 2**vector'LENGTH-1;
BEGIN
   IF (vector(vector'HIGH)='1') THEN result:=1;
   ELSE result:=0;
   END IF;
   FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
      result:=result*2;
      IF(vector(i)='1') THEN result:=result+1;
      END IF;
   END LOOP;
   RETURN result;
END conv_integer;

------ Function call: ------------------------------
...
y <= conv_integer(a);
...
----------------------------------------------------
```

Function Location

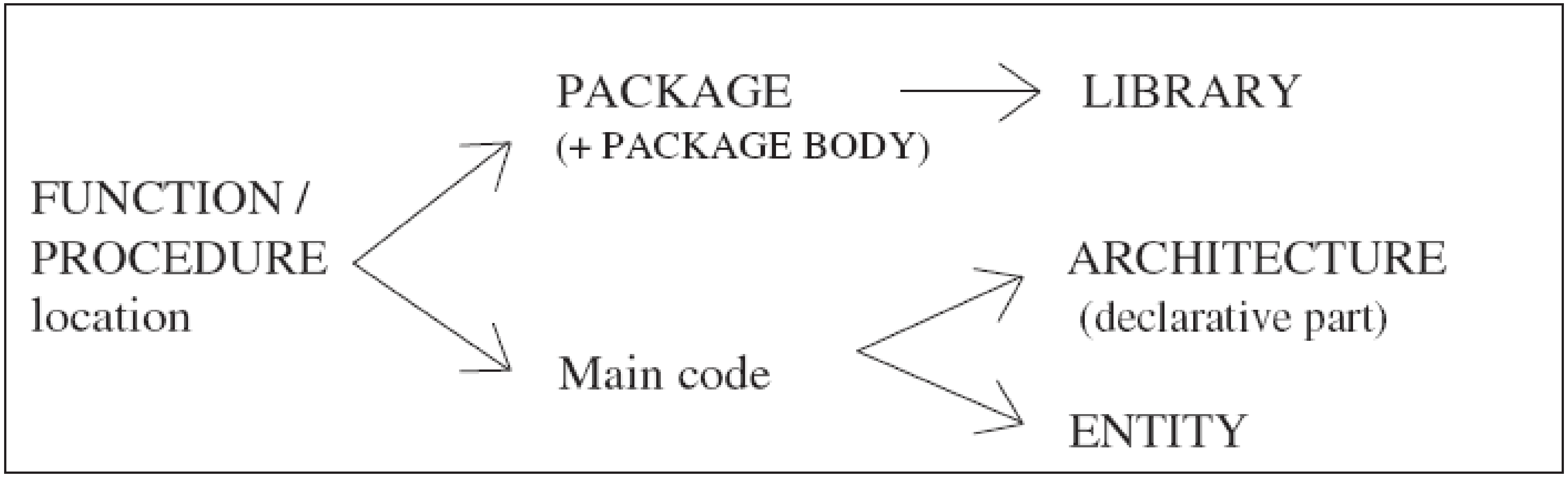

FUNCTION / PROCEDURE location
PACKAGE (+ PACKAGE BODY)
LIBRARY
Main code
ARCHITECTURE (declarative part)
ENTITY

Figure 11.1
Typical locations of a FUNCTION or PROCEDURE.

Example 11.3: FUNCTION Located in the Main Code

```
---------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY dff IS
   PORT ( d, clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
END dff;
---------------------------------------
```

Example 11.3 …

```
ARCHITECTURE my_arch OF dff IS
------------------------------------------
   FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
      RETURN BOOLEAN IS
   BEGIN
      RETURN s'EVENT AND s='1';
   END positive_edge;
------------------------------------------
BEGIN
   PROCESS (clk, rst)
   BEGIN
      IF (rst='1') THEN q <= '0';
      ELSIF positive_edge(clk) THEN q <= d;
      END IF;
   END PROCESS;
END my_arch;
---------------------------------------------
```

Example 11.4: FUNCTION Located in a PACKAGE

```
------- Package: ----------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------------
PACKAGE my_package IS
   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;
----------------------------------------------
```

Example 11.4 …

```
------ Main code: ------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
----------------------------------------------
ENTITY dff IS
   PORT ( d, clk, rst: IN STD_LOGIC;
            q: OUT STD_LOGIC);
END dff;
----------------------------------------------
ARCHITECTURE my_arch OF dff IS
BEGIN
   PROCESS (clk, rst)
   BEGIN
      IF (rst='1') THEN q <= '0';
      ELSIF positive_edge(clk) THEN  q <= d;
      END IF;
   END PROCESS;
END my_arch;
----------------------------------------------
```

Example 11.5: Function conv_integer( )

```
---------- Package: ------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--------------------------------------------------
PACKAGE my_package IS
   FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
      RETURN INTEGER;
END my_package;
--------------------------------------------------
```

Example 11.5 …

```
PACKAGE BODY my_package IS
   FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
         RETURN INTEGER IS
      VARIABLE result: INTEGER RANGE 0 TO 2**vector'LENGTH-1;
   BEGIN
      IF (vector(vector'HIGH)='1') THEN result:=1;
      ELSE result:=0;
      END IF;
      FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
         result:=result*2;
         IF(vector(i)='1') THEN result:=result+1;
         END IF;
      END LOOP;
      RETURN result;
   END conv_integer;
END my_package;
```

Example 11.5 …

```
--------- Main code: --------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
----------------------------------------------
ENTITY conv_int2 IS
   PORT ( a: IN STD_LOGIC_VECTOR(0 TO 3);
          y: OUT INTEGER RANGE 0 TO 15);
END conv_int2;
----------------------------------------------
ARCHITECTURE my_arch OF conv_int2 IS
BEGIN
   y <= conv_integer(a);
END my_arch;
----------------------------------------------
```

Example 11.6: Overloaded “+” Operator

```
-------- Package: ----------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
PACKAGE my_package IS
   FUNCTION "+" (a, b: STD_LOGIC_VECTOR)
      RETURN STD_LOGIC_VECTOR;
END my_package;
-----------------------------------------
```

Example 11.6 …

```
PACKAGE BODY my_package IS
   FUNCTION "+" (a, b: STD_LOGIC_VECTOR)
         RETURN STD_LOGIC_VECTOR IS
      VARIABLE result: STD_LOGIC_VECTOR;
      VARIABLE carry: STD_LOGIC;
   BEGIN
      carry := '0';
      FOR i IN a'REVERSE_RANGE LOOP
         result(i) := a(i) XOR b(i) XOR carry;
         carry := (a(i) AND b(i)) OR (a(i) AND carry) OR
                  (b(i) AND carry);
      END LOOP;
      RETURN result;
   END "+";
END my_package;
----------------------------------------------
```

Example 11.6 …

```
--------- Main code: ---------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
----------------------------------------
ENTITY add_bit IS
   PORT ( a: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          y: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END add_bit;
----------------------------------------
ARCHITECTURE my_arch OF add_bit IS
   CONSTANT b: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0011";
   CONSTANT c: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0110";
BEGIN
   y <= a + b + c;    -- overloaded "+" operator
END my_arch;
----------------------------------------
```

Example 11.7: Arithmetic Shift Function

```
--------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY shift_left IS
   GENERIC (size: INTEGER := 4);
   PORT ( a: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
          x, y, z: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
END shift_left;
---------------------------------------------
```

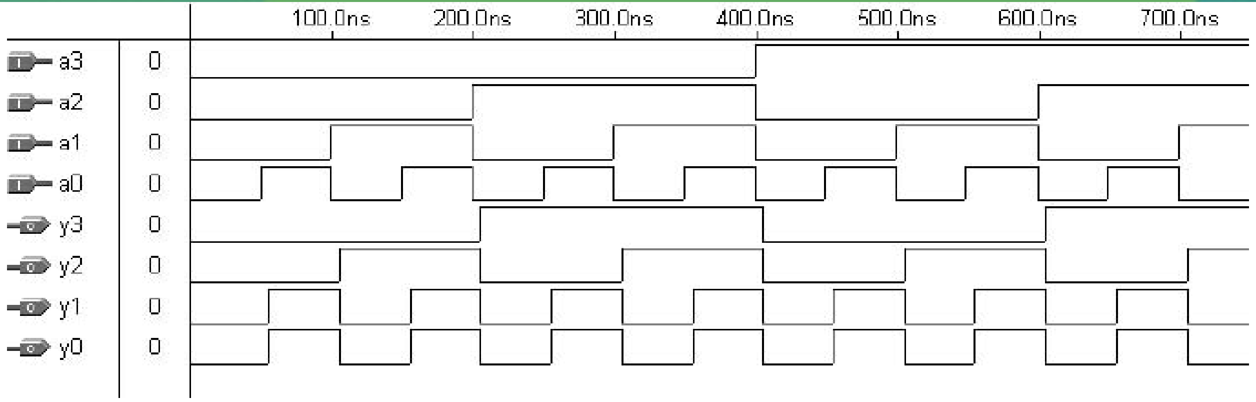
Example 11.7 …

```
ARCHITECTURE behavior OF shift_left IS
-----------------------------------------
   FUNCTION slar (arg1: STD_LOGIC_VECTOR; arg2: NATURAL)
         RETURN STD_LOGIC_VECTOR IS
      VARIABLE input: STD_LOGIC_VECTOR(size-1 DOWNTO 0) := arg1;
      CONSTANT size : INTEGER := arg1'LENGTH;
      VARIABLE copy: STD_LOGIC_VECTOR(size-1 DOWNTO 0)
         := (OTHERS => arg1(arg1'RIGHT));
      VARIABLE result: STD_LOGIC_VECTOR(size-1 DOWNTO 0);
   BEGIN
      IF (arg2 >= size-1) THEN result := copy;
      ELSE result := input(size-1-arg2 DOWNTO 1) &
         copy(arg2 DOWNTO 0);
      END IF;
      RETURN result;
   END slar;
-----------------------------------------
```

Example 11.7 …

```
-----------------------------------------
BEGIN
   x <= slar(a, 0);
   y <= slar(a, 1);
   z <= slar(a, 2);
END behavior;
-----------------------------------------
```

Example 11.7 …



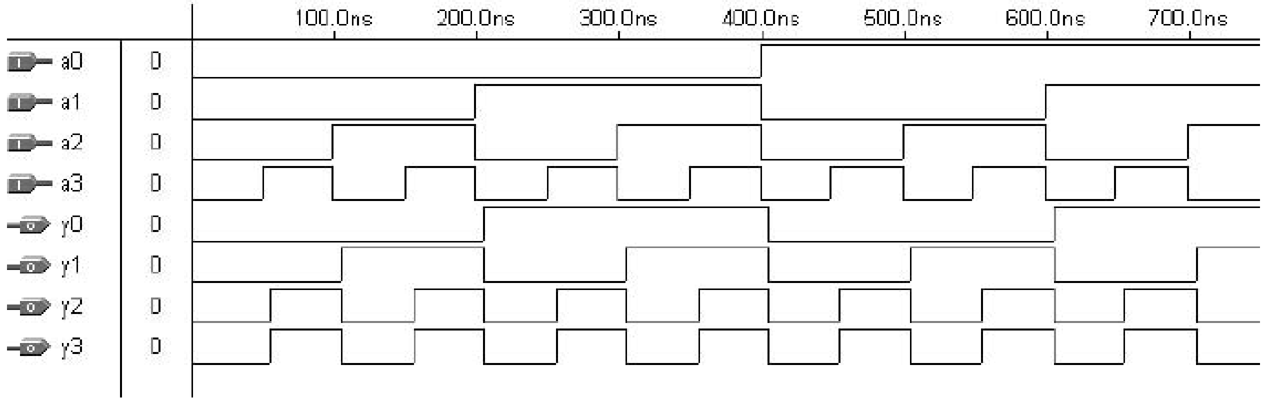


Figure 11.3
Simulation results of example 11.7.

Example 11.8: Multiplier

```
--------- Package: -------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
---------------------------------------------
PACKAGE pack IS
   FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED;
END pack;
---------------------------------------------
```

Example 11.8 …

```
-------------------------------------------------------
PACKAGE BODY pack IS
    FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED IS
        CONSTANT max: INTEGER := a'LENGTH + b'LENGTH - 1;
        VARIABLE aa: UNSIGNED(max DOWNTO 0) :=
            (max DOWNTO a'LENGTH => '0')
            & a(a'LENGTH-1 DOWNTO 0);
        VARIABLE prod: UNSIGNED(max DOWNTO 0) := (OTHERS => '0');
    BEGIN
        FOR i IN 0 TO a'LENGTH-1 LOOP
            IF (b(i)='1') THEN prod := prod + aa;
            END IF;
            aa := aa(max-1 DOWNTO 0) & '0';
        END LOOP;
        RETURN prod;
    END mult;
END pack;
-------------------------------------------------------
```

Example 11.8 …

```
-------- Main code: ---------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE work.my_package.all;
-----------------------------------------
ENTITY multiplier IS
   GENERIC (size: INTEGER := 4);
   PORT ( a, b: IN UNSIGNED(size-1 DOWNTO 0);
          y: OUT UNSIGNED(2*size-1 DOWNTO 0));
END multiplier;
-----------------------------------------
ARCHITECTURE behavior OF multiplier IS
BEGIN
   y <= mult(a,b);
END behavior;
-----------------------------------------------------
```

PROCEDURE

- A PROCEDURE is very similar to a FUNCTION and has the same basic purposes.
- A procedure can return more than one value.
- Two necessary parts of a PROCEDURE: (Like a FUNCTION)
  - The procedure body
  - A procedure call.

Procedure Body

```
PROCEDURE procedure_name [<parameter list>] IS
    [declarations]
BEGIN
    (sequential statements)
END procedure_name;
```

⟨parameter list⟩ = [CONSTANT] constant_name: mode type;

⟨parameter list⟩ = SIGNAL signal_name: mode type; or

⟨parameter list⟩ = VARIABLE variable_name: mode type;

```
PROCEDURE my_procedure ( a: IN BIT; SIGNAL b, c: IN BIT;
                         SIGNAL x: OUT BIT_VECTOR(7 DOWNTO 0);
                         SIGNAL y: INOUT INTEGER RANGE 0 TO 99) IS
BEGIN
   ...
END my_procedure;
```

(for outputs the default object is VARIABLE)

Procedure Call

```
compute_min_max(in1, in2, 1n3, out1, out2);
             -- statement by itself

divide(dividend, divisor, quotient, remainder);
             -- statement by itself

IF (a>b) THEN compute_min_max(in1, in2, 1n3, out1, out2);
             -- procedure call associated to another statement
```

Procedure Location

- The typical locations of a PROCEDURE are the same as those of a FUNCTION

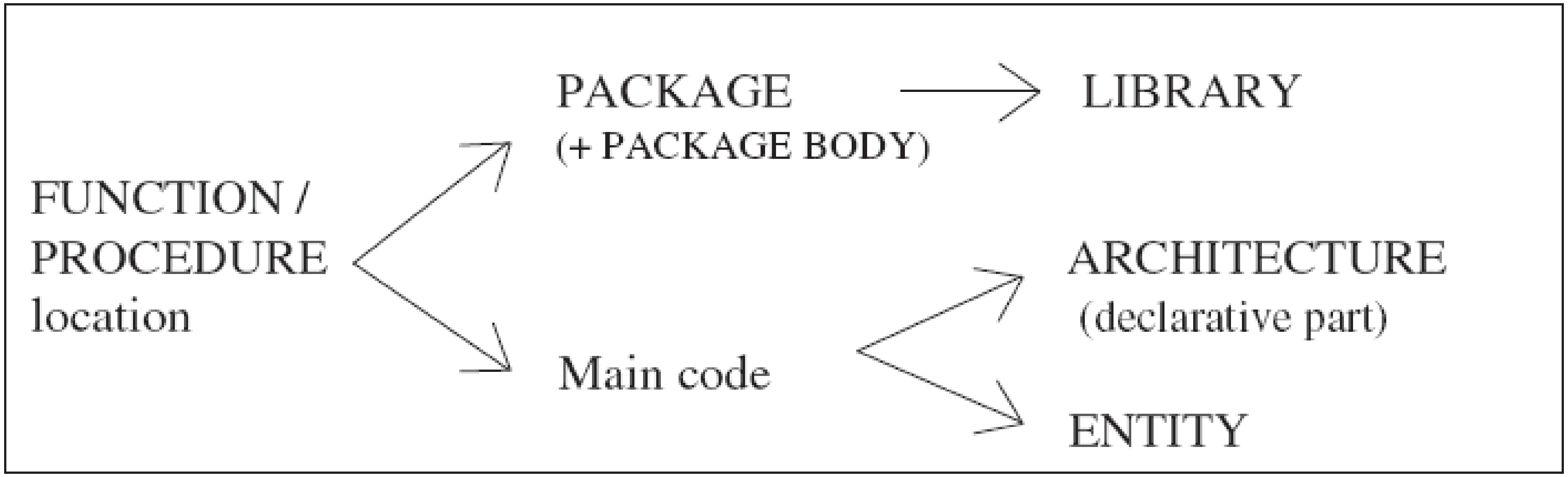

FUNCTION / PROCEDURE location
PACKAGE (+ PACKAGE BODY)
LIBRARY
Main code
ARCHITECTURE (declarative part)
ENTITY

Figure 11.1
Typical locations of a FUNCTION or PROCEDURE.

Example 11.9: PROCEDURE Located in the Main Code

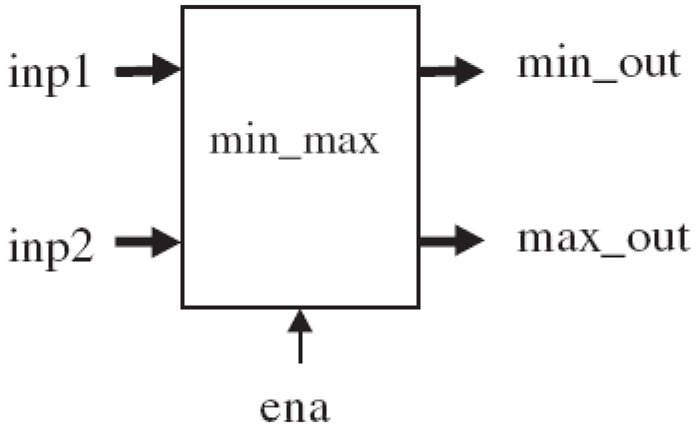
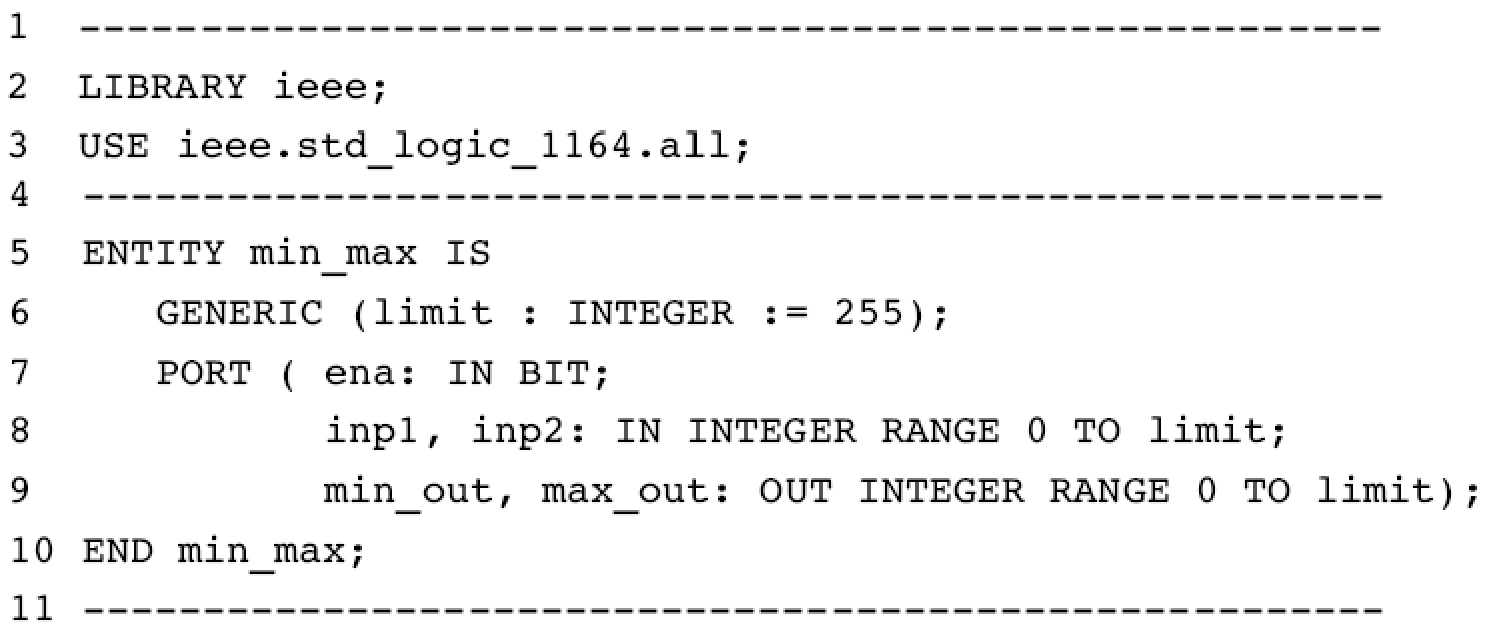


Figure 11.5
min_max circuit of example 11.9.

Example 11.9 …



```
------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------------------------
ENTITY min_max IS
   GENERIC (limit : INTEGER := 255);
   PORT ( ena: IN BIT;
          inp1, inp2: IN INTEGER RANGE 0 TO limit;
          min_out, max_out: OUT INTEGER RANGE 0 TO limit);
END min_max;
------------------------------------------------------
```

Example 11.9 …

```
ARCHITECTURE my_architecture OF min_max IS
   --------------------------
   PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
      SIGNAL min, max: OUT INTEGER RANGE 0 TO limit) IS
   BEGIN
      IF (in1 > in2) THEN
         max <= in1;
         min <= in2;
      ELSE
         max <= in2;
         min <= in1;
      END IF;
   END sort;
   --------------------------
BEGIN
   PROCESS (ena)
   BEGIN
      IF (ena='1') THEN sort (inp1, inp2, min_out, max_out);
      END IF;
   END PROCESS;
END my_architecture;
-------------------------------------------------------
```

Example 11.10: PROCEDURE Located in a PACKAGE

```
------------ Package: ---------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------
PACKAGE my_package IS
   CONSTANT limit: INTEGER := 255;
   PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
      SIGNAL min, max: OUT INTEGER RANGE 0 TO limit);
END my_package;
-------------------------------------
PACKAGE BODY my_package IS
   PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
      SIGNAL min, max: OUT INTEGER RANGE 0 TO limit) IS
   BEGIN
      IF (in1 > in2) THEN
         max <= in1;
         min <= in2;
      ELSE
         max <= in2;
         min <= in1;
      END IF;
   END sort;
END my_package;
-----------------------------------------------
```

Example 11.10 …

```
--------- Main code: ----------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-------------------------------------
ENTITY min_max IS
   GENERIC (limit: INTEGER := 255);
   PORT ( ena: IN BIT;
          inp1, inp2: IN INTEGER RANGE 0 TO limit;
          min_out, max_out: OUT INTEGER RANGE 0 TO limit);
END min_max;
-------------------------------------
ARCHITECTURE my_architecture OF min_max IS
BEGIN
   PROCESS (ena)
   BEGIN
      IF (ena='1') THEN sort (inp1, inp2, min_out, max_out);
      END IF;
   END PROCESS;
END my_architecture;
----------------------------------------
```

FUNCTION versus PROCEDURE Summary

- A FUNCTION has zero or more input parameters and a single return value. The input parameters can only be CONSTANTS (default) or SIGNALS (VARIABLES are not allowed).

- A PROCEDURE can have any number of IN, OUT, and INOUT parameters, which can be SIGNALS, VARIABLES, or CONSTANTS. For input parameters (mode IN) the default is CONSTANT, whereas for output parameters (mode OUTor INOUT) the default is VARIABLE.

FUNCTION versus PROCEDURE Summary

- A FUNCTION is called as part of an expression, while a PROCEDURE is a statement on its own.

- In both, WAIT and COMPONENTS are not synthesizable.

- The possible locations of FUNCTIONS and PROCEDURES are the same

ASSERT

```
ASSERT condition
[REPORT "message"]
[SEVERITY severity_level];
```

The severity level can be: **Note**, **Warning**, **Error (default)**, or **Failure**.
The message is written when the condition is **FALSE**.

```
ASSERT a'LENGTH = b'LENGTH
REPORT "Error: vectors do not have same length!"
SEVERITY failure;
```

Thanks for your attention

- Don’t forget

Problems!!!