POWER & ENERGY CONVERSION LABORATORY

PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

PACKAGES and COMPONENTS
SYSTEM DESIGN

It is time to integrate several circuits into a bigger system

VHDL

VHSIC HARDWARE DESCRIPTION LANGUAGE

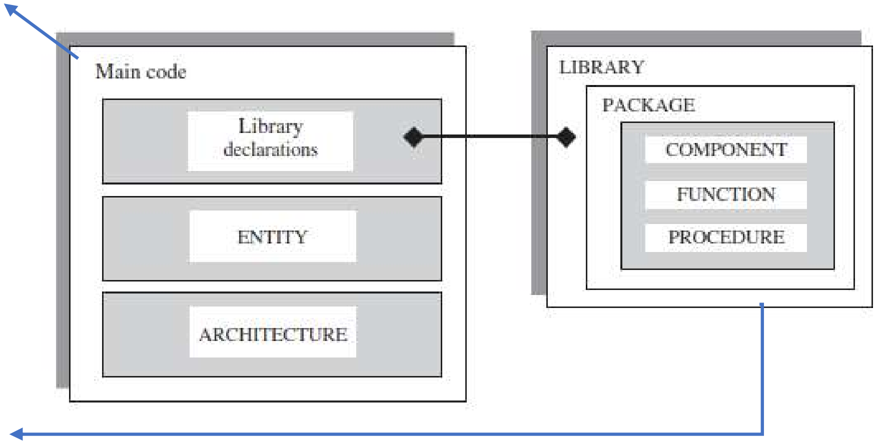
Packages and Components

System Design

- We understand already the **main code of a VHDL** circuit (code structure, data types, operators, attributes, concurrent and sequential statements, signals, variables, constants and state machines).

- To make the code more readable and reusable, we can wrap some code into **COMPONENTS, FUNCTIONS or PROCEDURES**, which are placed into a PACKAGE. This is compiled into a **LIBRARY**, that can be used later into our main code.

- The code within **COMPONENTS, FUNCTIONS or PROCEDURES** can obviously be written into the **main code**, but in that case, it is now modular or easily reusable.



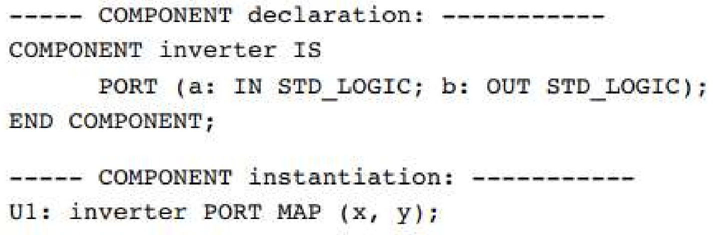
Packages and Components

COMPONENT Definition

- COMPONENT is simply a piece of conventional code (LIBRARY+ENTITY+ARCHITECTURE).
- It can be used into another circuits (hierarchical design). Creating modularity for our designs.
- For example: adders, divisors, multiplexers, etc, can be used as **COMPONENTS** available in our **own LIBRARY**.
- The following syntax is used to ***declare*** and then ***instantiate*** a **COMPONENT:**

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

label: component_name PORT MAP (port_list);

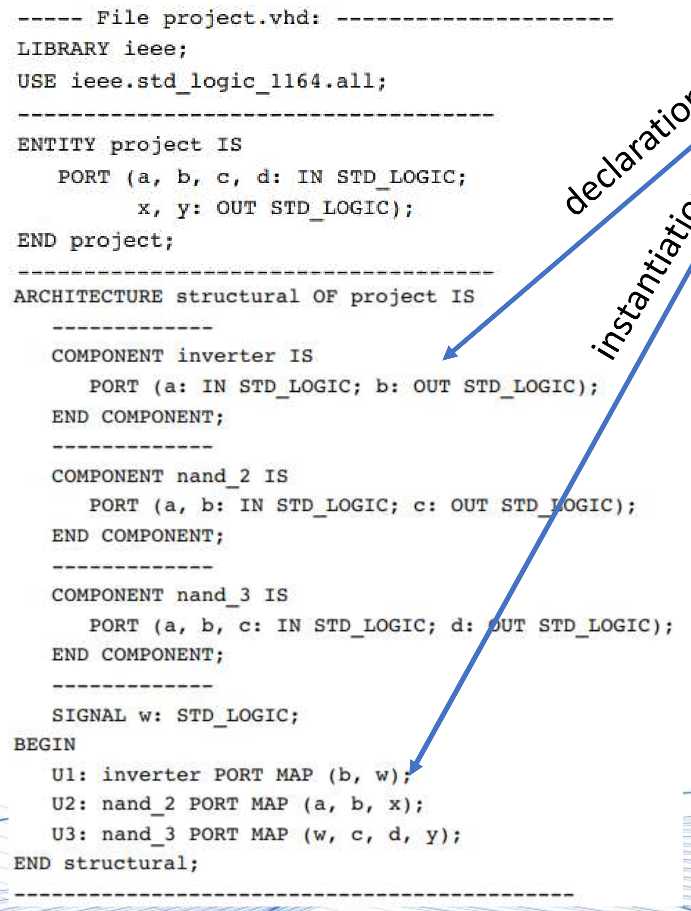


```
----- COMPONENT declaration: -----------
COMPONENT inverter IS
      PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;

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

- This code would **make use of the component "inverter"** For that, we had previously defined our circuit *inverter.vhd* and compiled into the work library.
- Label U1 is chosen by user.
- Inputs and outputs of the actual circuit are **x and y**. Internally assigned to the component input/output **a and b.**

Packages and Components

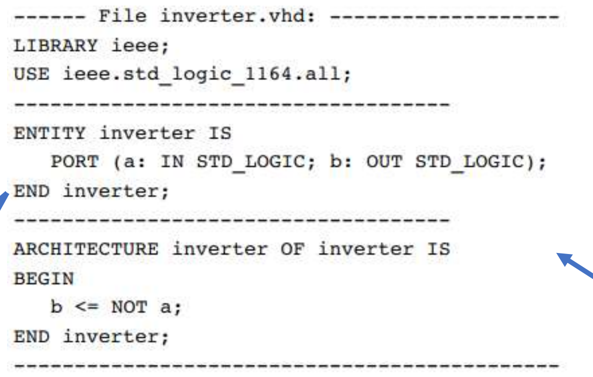
COMPONENT without PACKAGE



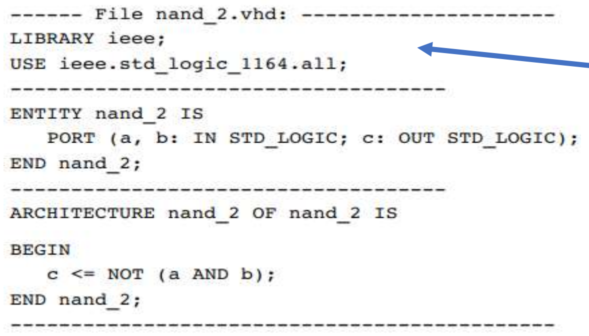
```
----- File project.vhd: ---------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY project IS
   PORT (a, b, c, d: IN STD_LOGIC;
         x, y: OUT STD_LOGIC);
END project;
------------------------------------
ARCHITECTURE structural OF project IS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

declaration

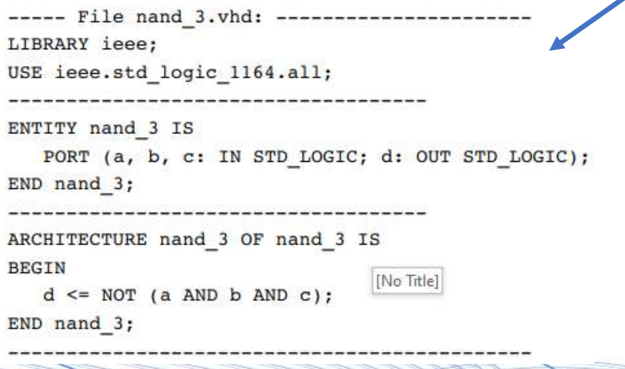
instantiation



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

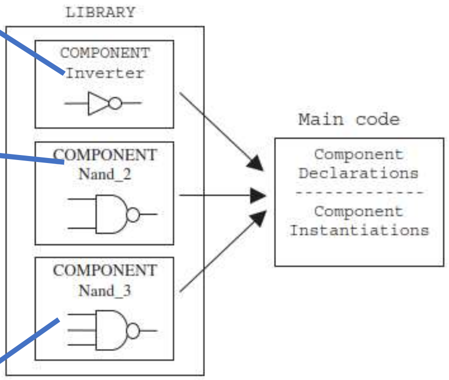


```
------ File nand_2.vhd: ---------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------
ENTITY nand_2 IS
   PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
END nand_2;
-----------------------------------
ARCHITECTURE nand_2 OF nand_2 IS
BEGIN
   c <= NOT (a AND b);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



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

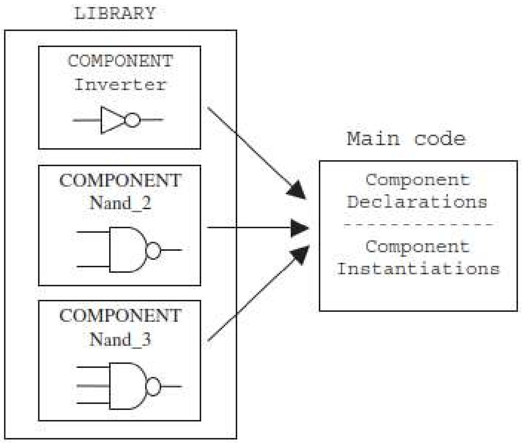
[No Title]

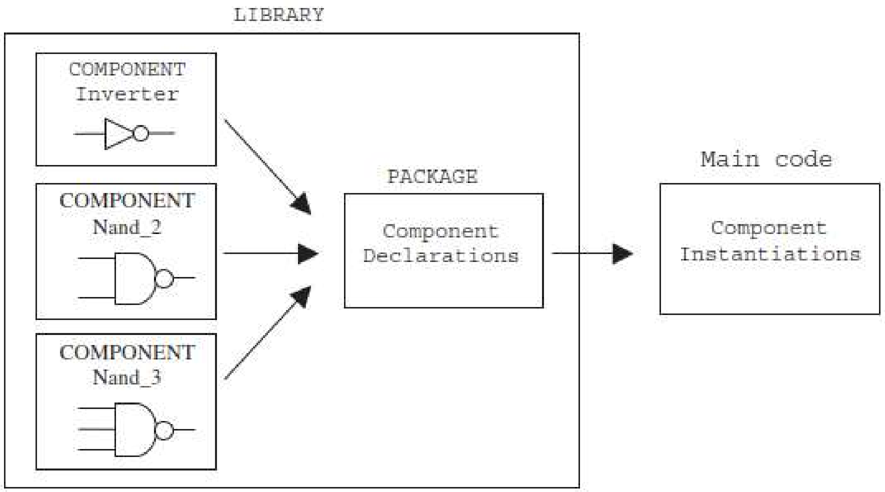


Packages and Components

COMPONENT Definition

- A COMPONENT can be defined as a single independent file. In this case, we need to declare the component into the main code.
- A COMPONENT can be also integrated into a PACKAGE, where it is declared. In this case, we do not need to declare the component into the main code.





Packages and Components

PACKAGE Definition

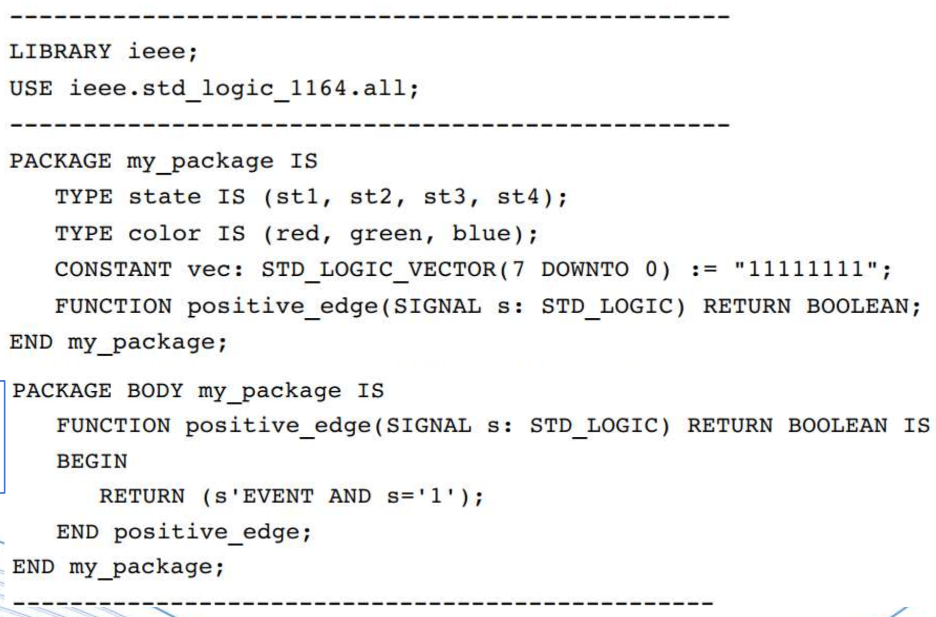
- It is the vessel for a COMPONENT, FUNCTION or PROCEDURE.
- Packages allows code partitioning, sharing and reuse.
- The syntax of a PACKAGE is composed of two parts. PACKAGE and PACKAGE BODY:
    - **PAKAGE**: Mandatory and contains all declarations
    - **PACKAGE BODY**: Necessary when PACKAGE declare FUNCTIONS or PROCEDURE. It contain its descriptions.

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

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

Note: PACKAGE and PACKAGE BODY must have same name.

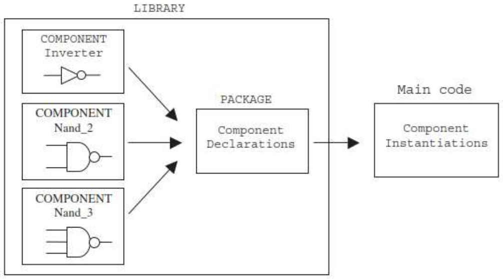
To make use of our PACKAGE, we need to include our *work* library.
USE work.my_package.all

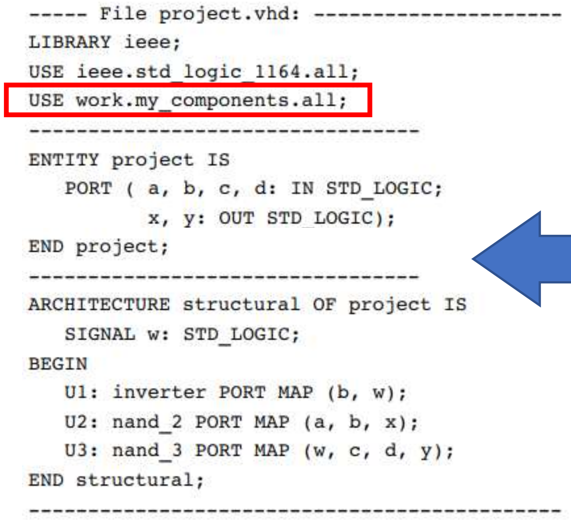


```
---------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------------
PACKAGE my_package IS
   TYPE state IS (st1, st2, st3, st4);
   TYPE color IS (red, green, blue);
   CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";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

Packages and Components

COMPONENT using PACKAGE

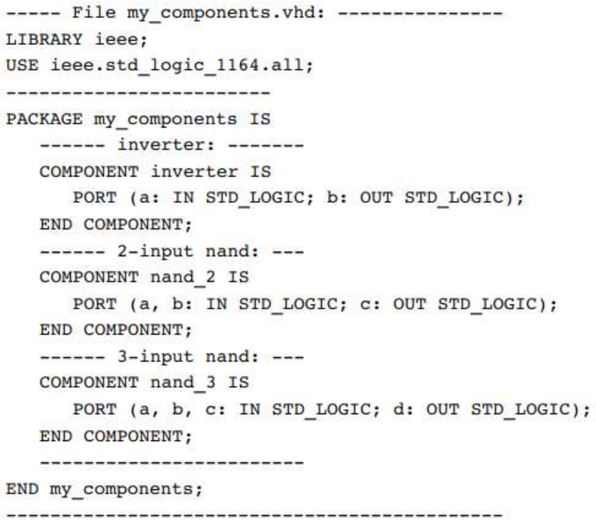




```
----- File project.vhd: ----------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_components.all;
--------------------------------
ENTITY project IS
   PORT ( a, b, c, d: IN STD_LOGIC;
          x, y: OUT STD_LOGIC);
END project;
--------------------------------
ARCHITECTURE structural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

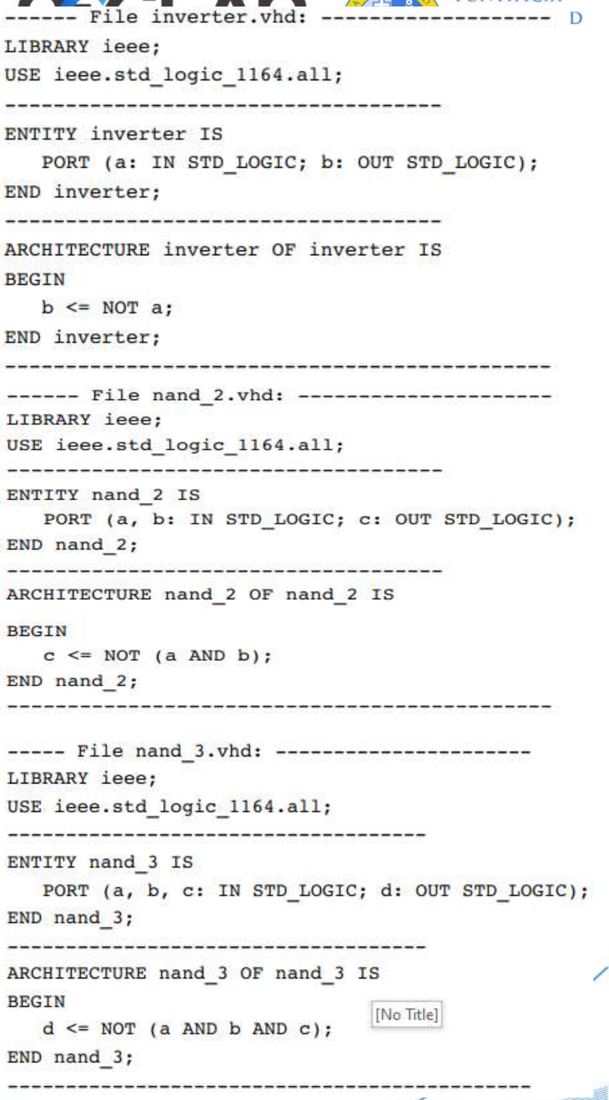
Main Code:

- Library is included in this case
- No declaration is needed.
- Only instantiation is needed



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

Extra file for PACKAGE is needed



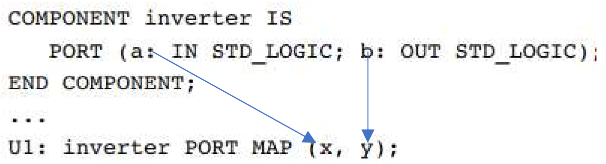
```
------ File inverter.vhd: -------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------
ENTITY inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END inverter;
------------------------------------
ARCHITECTURE inverter OF inverter IS
BEGIN
   b <= NOT a;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

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

[No Title]

Packages and Components

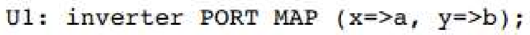
PORT MAP

- Mapping a COMPONENT PORT can be done in two ways:
  - Positional mapping, where an automatic assignment is done by the position of the variables:



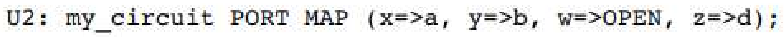
```
COMPONENT inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;
...
U1: inverter PORT MAP (x, y);
```

  - Nominal mapping, where the mapping is assigned explicitly:



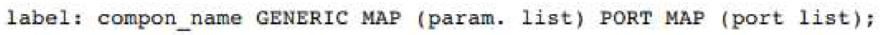
```
U1: inverter PORT MAP (x=>a, y=>b);
```

  - We can also leave a pin unconnected using OPEN:



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

  - Also, GENERIC parameters (as seen in operators and attributes section) can be used. See next example.-

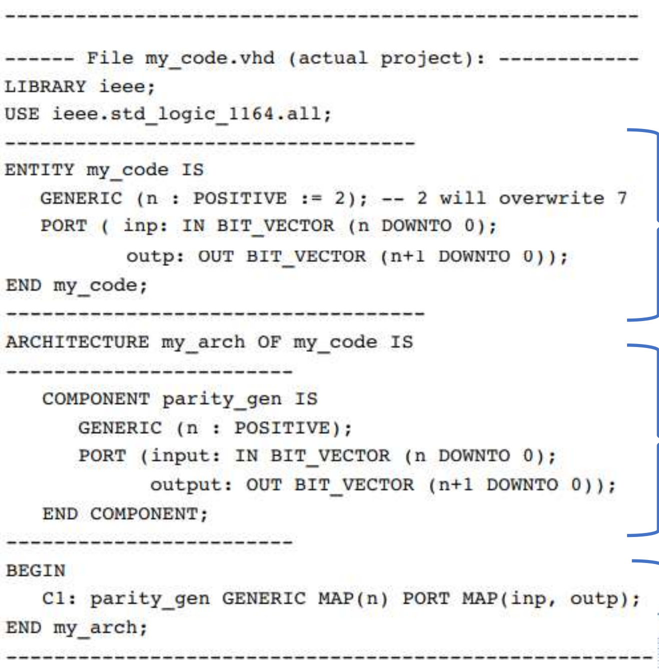


```
label: compon_name GENERIC MAP (param. list) PORT MAP (port list);
```

Packages and Components

PORT MAP Example – Parity Generator

The circuit must add one bit to the input vector (on its left). Such bit must be a ‘0’ if the number of ‘1’s in the input vector is even, or a ‘1’ if it is odd, such that the resulting vector will always contain an even number of ‘1’s (even parity).

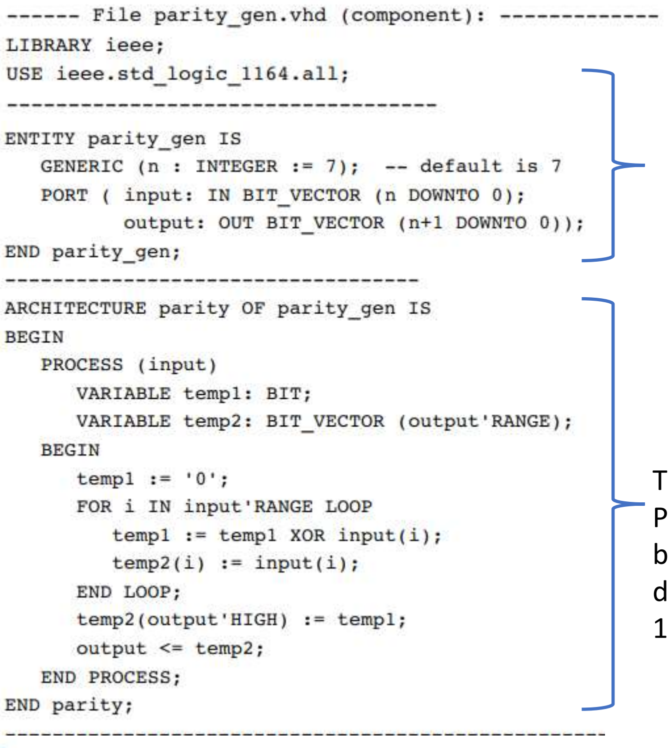


```
-------------------------------------------------------
------ File my_code.vhd (actual project): ------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------
ENTITY my_code IS
   GENERIC (n : POSITIVE := 2); -- 2 will overwrite 7
   PORT ( inp: IN BIT_VECTOR (n DOWNTO 0);
          outp: OUT BIT_VECTOR (n+1 DOWNTO 0));
END my_code;
-----------------------------------
ARCHITECTURE my_arch OF my_code IS
------------------------
   COMPONENT parity_gen IS
      GENERIC (n : POSITIVE);
      PORT (input: IN BIT_VECTOR (n DOWNTO 0);
            output: OUT BIT_VECTOR (n+1 DOWNTO 0));
   END COMPONENT;
------------------------
BEGIN
   C1: parity_gen GENERIC MAP(n) PORT MAP(inp, outp);
END my_arch;
-------------------------------------------------------
```

Entity definition of the main code defined as generic.

COMPONENT declaration, assigning its GENERIC parameter (2 overwrite 7) and PORTs.

COMPONENT instantiation with generic PORT MAP



```
------ File parity_gen.vhd (component): -------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------
ENTITY parity_gen IS
   GENERIC (n : INTEGER := 7);  -- default is 7
   PORT ( input: IN BIT_VECTOR (n DOWNTO 0);
          output: OUT BIT_VECTOR (n+1 DOWNTO 0));
END parity_gen;
-----------------------------------
ARCHITECTURE parity OF parity_gen IS
BEGIN
   PROCESS (input)
      VARIABLE temp1: BIT;
      VARIABLE temp2: BIT_VECTOR (output'RANGE);
   BEGIN
      temp1 := '0';
      FOR i IN input'RANGE LOOP
         temp1 := temp1 XOR input(i);
         temp2(i) := input(i);
      END LOOP;
      temp2(output'HIGH) := temp1;
      output <= temp2;
   END PROCESS;
END parity;
-----------------------------------------------
```

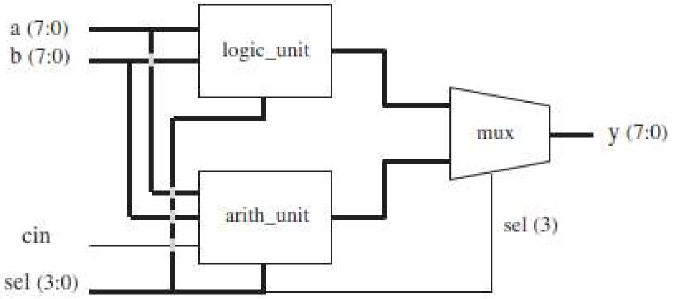
Define our entity with generic number of input and output ports. (COMPONENT)

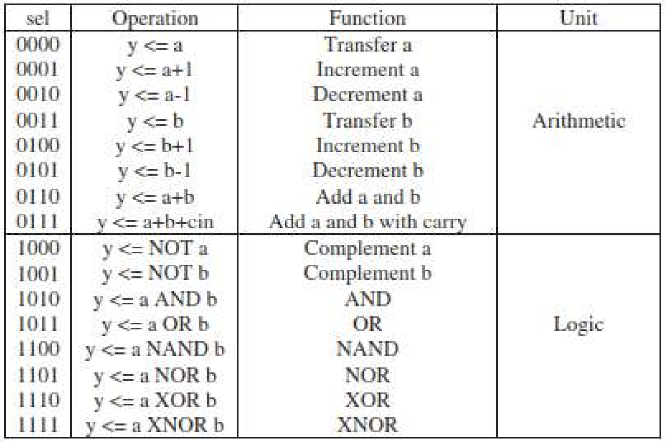
The COMPONENT makes its PROCESS of adding an extra bit, which is 1 or 0 depending on the number of 1s in the input.

Packages and Components

Exercise ALU

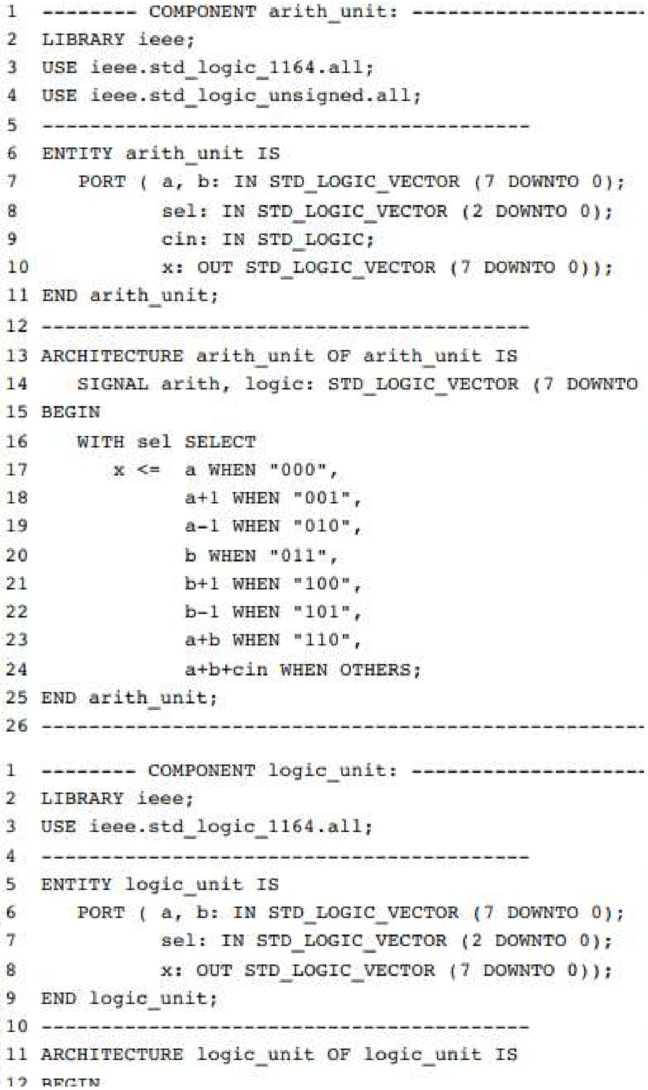
In previous example we designed an ALU using only a self-contained code. In this code you should create three components logic_unit, arith_unit and mux.



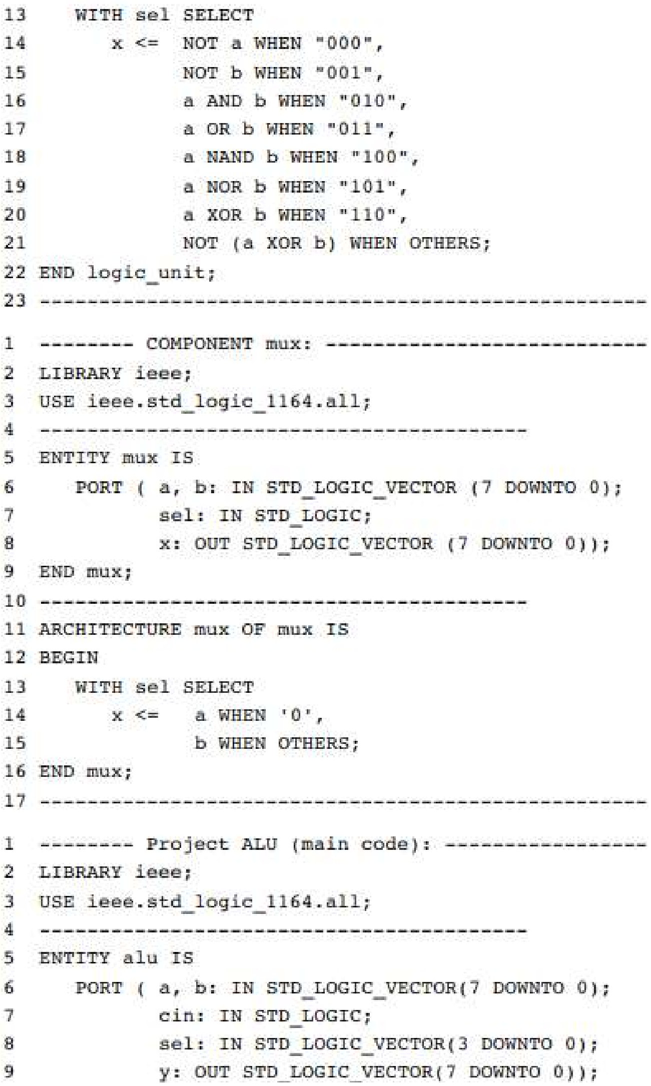


sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	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	

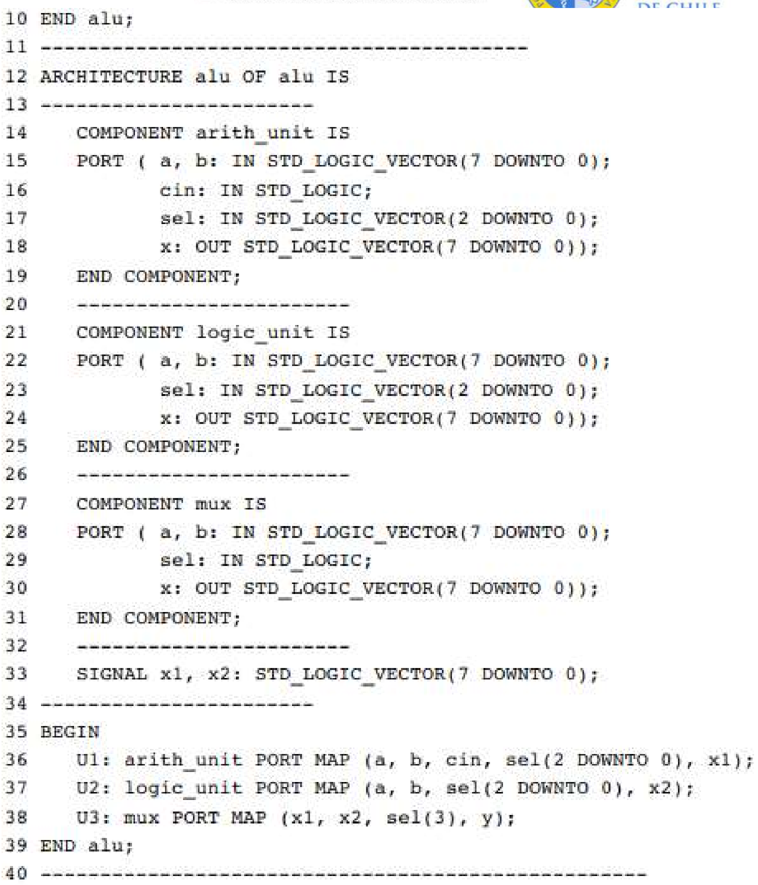
Packages and Components



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

-------- 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;
--------------------------------------------------

-------- 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));
```

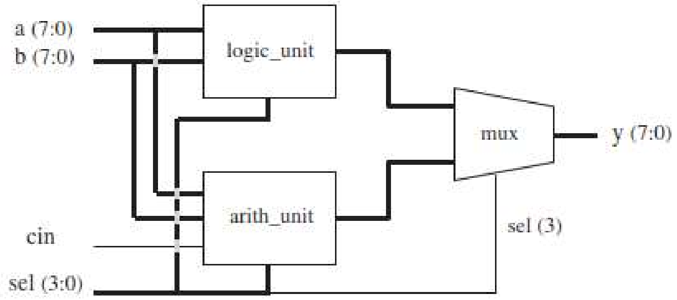


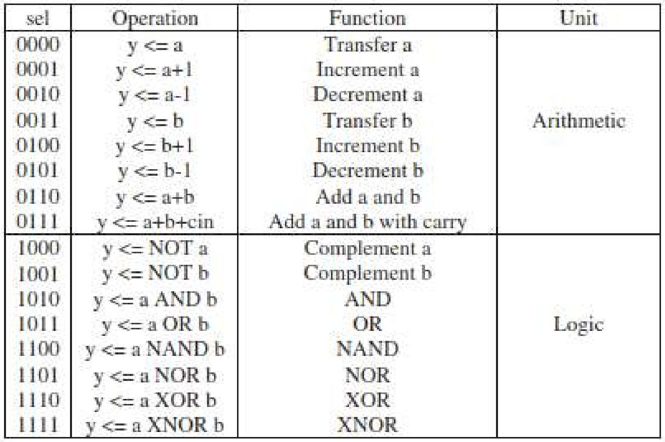
```
END alu;
-----------------------------------------
ARCHITECTURE alu OF alu IS
-----------------------
   COMPONENT arith_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          cin: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          x: OUT STD_LOGIC_VECTOR(7 DOWNTO 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;
--------------------------------------------------
```

Packages and Components

Exercise ALU

Do the same previous exercise and use PACKAGE that contain all components.
Recompile and simulate the results.





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	

peclab
POWER & ENERGY CONVERSION LABORATORY

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

END-PACKAGES and COMPONENTS
SYSTEM DESIGN

It is time to integrate several circuits into a bigger system

VHDL

VHSIC HARDWARE DESCRIPTION LANGUAGE