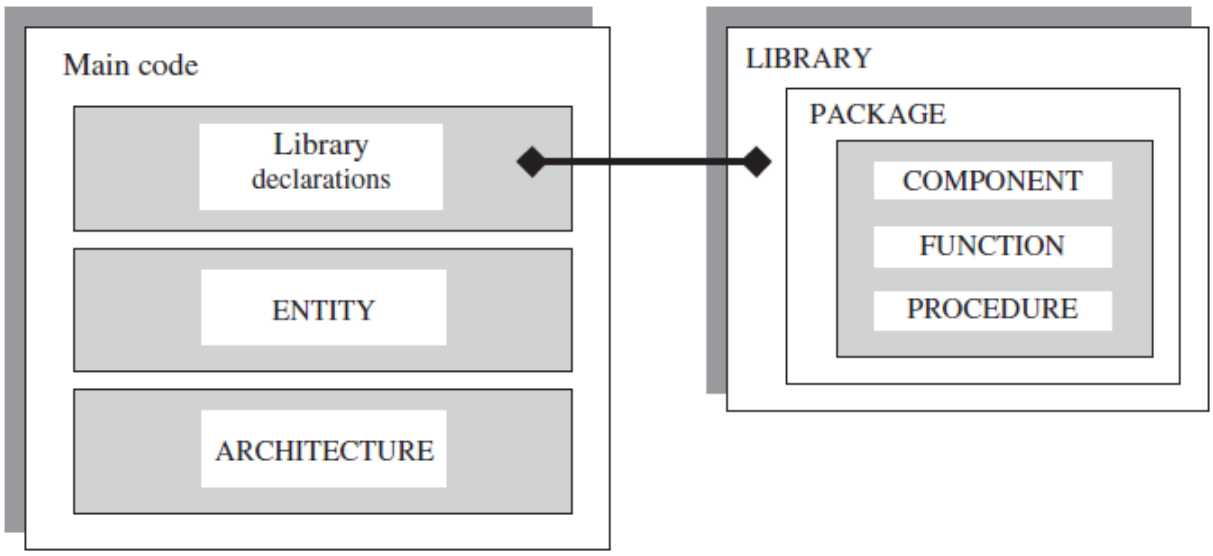
**Packages and Components**

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.




**PACKAGE**

Frequently used pieces of VHDL code are usually written in the form of COMPONENTS, FUNCTIONS, or PROCEDURES.

Such codes are then placed inside a PACKAGE and compiled into the destination LIBRARY. The importance of this technique is that it allows code partitioning, code sharing, and code reuse.

Package syntax is presented below.

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

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

The syntax is composed of two parts: PACKAGE and PACKAGE BODY.

The first part is mandatory and contains all declarations, while the second part is necessary only when one or more subprograms (FUNCTION or PROCEDURE) are declared in the 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.

**Example:** 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.

```
1 ------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------------
5 PACKAGE my_package IS
6       TYPE state IS (st1, st2, st3, st4);
7       TYPE color IS (red, green, blue);
8       CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
9 END my_package;
10 ------------------------------------------------
```

**Example:** Package with a Function

This example contains, besides TYPE and CONSTANT declarations, a FUNCTION.

Therefore, a PACKAGE BODY is now needed. This function returns TRUE when a positive edge occurs on clk.

```
1 ------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------------
5 PACKAGE my_package IS
6       TYPE state IS (st1, st2, st3, st4);
7       TYPE color IS (red, green, blue);
8       CONSTANT vec: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
9       FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN;
```

```
10 END my_package;
11 ---------------------------------------------
12 PACKAGE BODY my_package IS
13      FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS
14      BEGIN
15              RETURN (s'EVENT AND s='1');
16      END positive_edge;
17 END my_package;
18 ---------------------------------------------
```

Any of the PACKAGES above 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 a way of partitioning a code and providing code sharing and code reuse.

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

### 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:** Inverter as a component

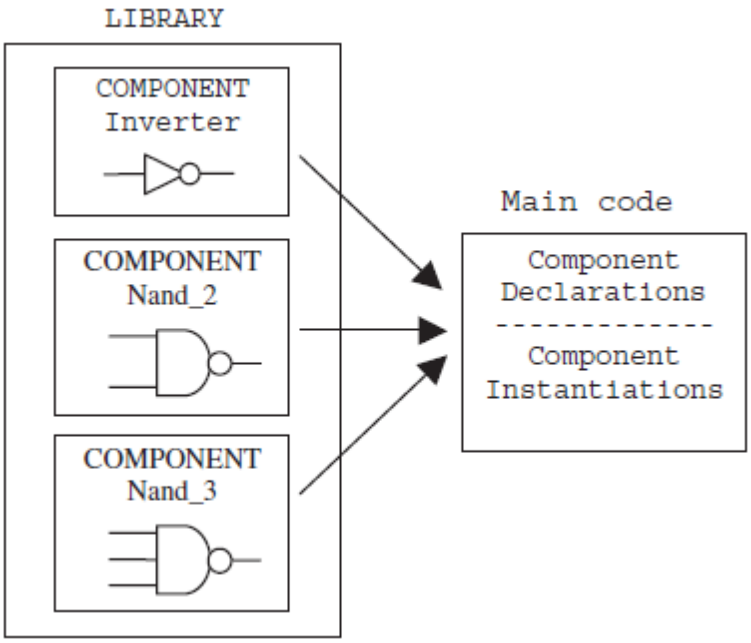
----- COMPONENT declaration: ----------

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

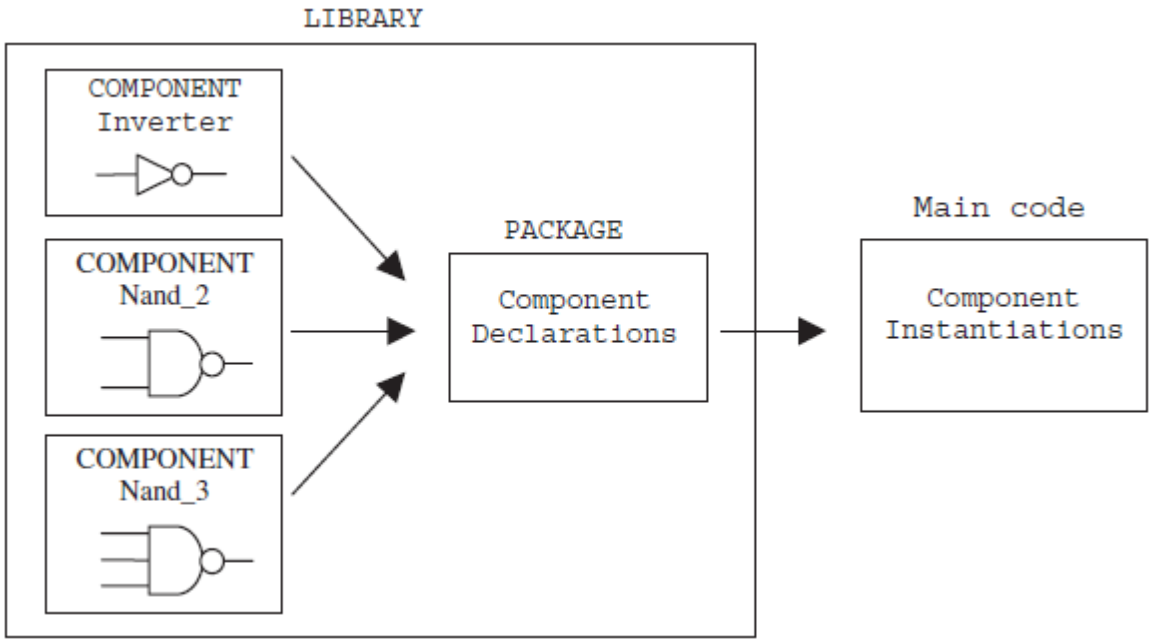
----- COMPONENT instantiation: ----------

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

**Basic ways of declaring COMPONENTS**



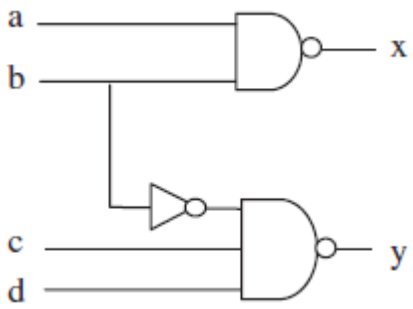

Declarations in the main code itself




Declarations in a PACKAGE.

**Example:** Components Declared in the Main Code

Implement the circuit below employing only COMPONENTS (inverter, nand_2, and nand_3), but without creating a specific PACKAGE to declare them.



```
1 ------ File inverter.vhd: ------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------
5 ENTITY inverter IS
6       PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
7 END inverter;
8 ---------------------------------
9 ARCHITECTURE inverter OF inverter IS
10 BEGIN
11      b <= NOT a;
12 END inverter;
13 -----------------------------------------
```

```
1 ------ File nand_2.vhd: --------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------
5 ENTITY nand_2 IS
6       PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
7 END nand_2;
8 ---------------------------------
9 ARCHITECTURE nand_2 OF nand_2 IS
10 BEGIN
11      c <= NOT (a AND b);
12 END nand_2;
```

```
1 ----- File nand_3.vhd: ----------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----------------------------------
5 ENTITY nand_3 IS
6       PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
7 END nand_3;
8 -----------------------------------
9 ARCHITECTURE nand_3 OF nand_3 IS
10 BEGIN
11      d <= NOT (a AND b AND c);
12 END nand_3;
13 -------------------------------------------
```

```
1 ----- File project.vhd: ----------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----------------------------------
5 ENTITY project IS
6       PORT (a, b, c, d: IN STD_LOGIC;
7                 x, y: OUT STD_LOGIC);
8 END project;
```

```
10 ARCHITECTURE structural OF project IS
11 -------------
12 COMPONENT inverter IS
13      PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
14 END COMPONENT;
15 -------------
16 COMPONENT nand_2 IS
17      PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
18 END COMPONENT;
19 -------------
20 COMPONENT nand_3 IS
21      PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
22 END COMPONENT;
23 -------------
24 SIGNAL w: STD_LOGIC;
25 BEGIN
26      U1: inverter PORT MAP (b, w);
27      U2: nand_2 PORT MAP (a, b, x);
28      U3: nand_3 PORT MAP (w, c, d, y);
29 END structural;
30 ---------------------------------------------
9 ----------------------------------
```

**Example:** Components Declared in a Package

We want to implement the same project of the previous example. However, we will now create a PACKAGE where all the COMPONENTS (inverter, nand_2, and nand_3) will be declared,

```
1 ------ File inverter.vhd: -------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------
5 ENTITY inverter IS
6       PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
7 END inverter;
8 ------------------------------------
9 ARCHITECTURE inverter OF inverter IS
10 BEGIN
11      b <= NOT a;
12 END inverter;
13 ------------------------------------------
```

```
1 ------ File nand_2.vhd: ---------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------
5 ENTITY nand_2 IS
6       PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
7 END nand_2;
8 ------------------------------------
```

```
9 ARCHITECTURE nand_2 OF nand_2 IS
10 BEGIN
11      c <= NOT (a AND b);
12 END nand_2;
13 ----------------------------------------
```

```
1 ----- File nand_3.vhd: ----------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----------------------------------
5 ENTITY nand_3 IS
6       PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
7 END nand_3;
8 -----------------------------------
9 ARCHITECTURE nand_3 OF nand_3 IS
10 BEGIN
11      d <= NOT (a AND b AND c);
12 END nand_3;
13 ----------------------------------------
```

```
1 ----- File my_components.vhd: ---------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------
5 PACKAGE my_components IS
6 ------ inverter: -------
7 COMPONENT inverter IS
8       PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
9 END COMPONENT;
10 ------ 2-input nand: ---
11 COMPONENT nand_2 IS
12      PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC);
13 END COMPONENT;
14 ------ 3-input nand: ---
15 COMPONENT nand_3 IS
16      PORT (a, b, c: IN STD_LOGIC; d: OUT STD_LOGIC);
17 END COMPONENT;
18 ------------------------
19 END my_components;
20 -------------------------------------------
```

```
1 ----- File project.vhd: ----------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE work.my_components.all;
5 ---------------------------------
6 ENTITY project IS
7       PORT ( a, b, c, d: IN STD_LOGIC;
8                   x, y: OUT STD_LOGIC);
9 END project;
10 ---------------------------------
11 ARCHITECTURE structural OF project IS
12      SIGNAL w: STD_LOGIC;
13 BEGIN
14      U1: inverter PORT MAP (b, w);
15      U2: nand_2 PORT MAP (a, b, x);
16      U3: nand_3 PORT MAP (w, c, d, y);
17 END structural;
18 -----------------------------------------
```

**PORT MAP**

There are two ways to map the PORTS of a COMPONENT during its instantiation: positional mapping and nominal mapping.

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

In it, the mapping is positional; that is, PORTS x and y correspond to a and b, respectively. On the other hand, a nominal mapping would be the following:

```
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:

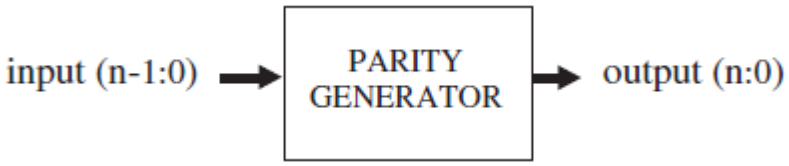
```
U2: my_circuit PORT MAP (x=>a, y=>b, w=>OPEN, z=>d);
```

**GENERIC MAP**

GENERIC units can also be instantiated. In that case, a GENERIC MAP must be used in the COMPONENT instantiation to pass information to the GENERIC parameters.

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

**Example:** Instantiating a Generic Component




```
1 ------ File parity_gen.vhd (component): ----------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------
5 ENTITY parity_gen IS
6       GENERIC (n : INTEGER := 7); -- default is 7
7       PORT ( input: IN BIT_VECTOR (n DOWNTO 0);
8                       output: OUT BIT_VECTOR (n+1 DOWNTO 0));
9 END parity_gen;
10 ---------------------------------
11 ARCHITECTURE parity OF parity_gen IS
12 BEGIN
13      PROCESS (input)
14              VARIABLE temp1: BIT;
15              VARIABLE temp2: BIT_VECTOR (output'RANGE);
16      BEGIN
17              temp1 := '0';
18              FOR i IN input'RANGE LOOP
19                      temp1 := temp1 XOR input(i);
20                      temp2(i) := input(i);
21              END LOOP;
```

```
22              temp2(output'HIGH) := temp1;
23              output <= temp2;
24 END PROCESS;
25 END parity;
26 -------------------------------------------------
```

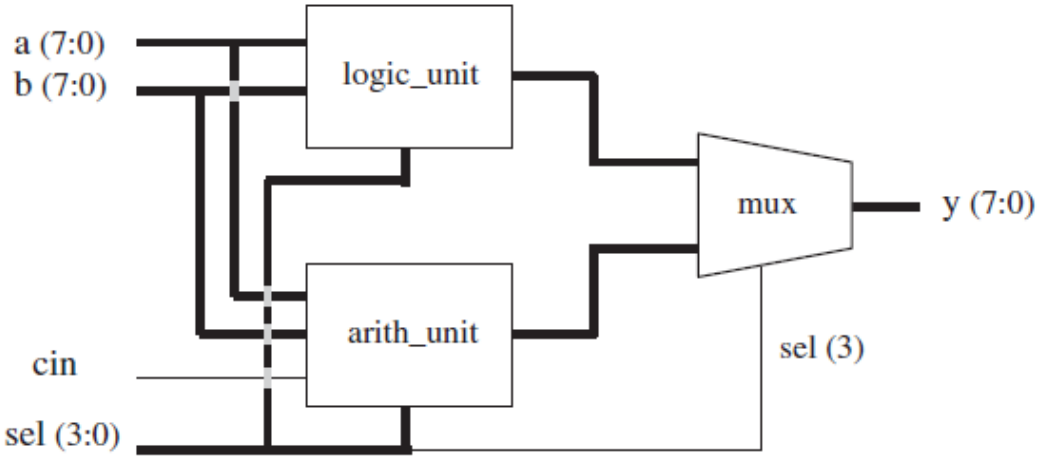
```
1 ------ File my_code.vhd (actual project): ------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------
5 ENTITY my_code IS
6 GENERIC (n : POSITIVE := 2); -- 2 will overwrite 7
7               PORT ( inp: IN BIT_VECTOR (n DOWNTO 0);
8                       outp: OUT BIT_VECTOR (n+1 DOWNTO 0));
9 END my_code;
10 ---------------------------------
11 ARCHITECTURE my_arch OF my_code IS
12 ------------------------
13 COMPONENT parity_gen IS
14     GENERIC (n : POSITIVE);
15     PORT (input: IN BIT_VECTOR (n DOWNTO 0);
16              output: OUT BIT_VECTOR (n+1 DOWNTO 0));
17 END COMPONENT;
18 ------------------------
19 BEGIN
```

20 C1: parity_gen GENERIC MAP(n) PORT MAP(inp, outp);

21 END my_arch;

22 ----------

**Example:** ALU Made of 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 | |

```
1 -------- COMPONENT arith_unit: --------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE ieee.std_logic_unsigned.all;
5 ---------------------------------------
6 ENTITY arith_unit IS
7       PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8                    sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
9                   cin: IN STD_LOGIC;
10                   x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END arith_unit;
12 ---------------------------------------
13 ARCHITECTURE arith_unit OF arith_unit IS
14     SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15 BEGIN
16     WITH sel SELECT
17              x <= a WHEN "000",
18                   a+1 WHEN "001",
19                   a-1 WHEN "010",
20                     b WHEN "011",
21                  b+1 WHEN "100",
22                  b-1 WHEN "101",
23                 a+b WHEN "110",
24               a+b+cin WHEN OTHERS;
25 END arith_unit;
```

```
26 ------------------------------------------------


1 -------- COMPONENT logic_unit: --------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY logic_unit IS
6       PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7                    sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8                     x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9 END logic_unit;
10 ----------------------------------------
11 ARCHITECTURE logic_unit OF logic_unit IS
12 BEGIN
13     WITH sel SELECT
14              x <= NOT a WHEN "000",
15                    NOT b WHEN "001",
16                 a AND b WHEN "010",
17                    a OR b WHEN "011",
18               a NAND b WHEN "100",
19                 a NOR b WHEN "101",
20                 a XOR b WHEN "110",
21       NOT (a XOR b) WHEN OTHERS;
22 END logic_unit;
23 ------------------------------------------------
```

```
1 -------- COMPONENT mux: --------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY mux IS
6 PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7               sel: IN STD_LOGIC;
8                 x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9 END mux;
10 ---------------------------------------
11 ARCHITECTURE mux OF mux IS
12 BEGIN
13      WITH sel SELECT
14              x <= a WHEN '0',
15              b WHEN OTHERS;
16 END mux;
17 -----------------------------------------------
```

```
1 -------- Project ALU (main code): -----------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY alu IS
6       PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
```

```
7                 cin: IN STD_LOGIC;
8                 sel: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
9                  y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
10 END alu;
11 ----------------------------------------
12 ARCHITECTURE alu OF alu IS
13 ----------------------
14 COMPONENT arith_unit IS
15     PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
16                cin: IN STD_LOGIC;
17                 sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
18                   x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
19 END COMPONENT;
20 ----------------------
21 COMPONENT logic_unit IS
22     PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
23               sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
24                x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
25 END COMPONENT;
26 ----------------------
27 COMPONENT mux IS
28     PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
29               sel: IN STD_LOGIC;
30                 x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
31 END COMPONENT;
```

```
32 ----------------------
33 SIGNAL x1, x2: STD_LOGIC_VECTOR(7 DOWNTO 0);
34 ----------------------
35 BEGIN
36      U1: arith_unit PORT MAP (a, b, cin, sel(2 DOWNTO 0), x1);
37      U2: logic_unit PORT MAP (a, b, sel(2 DOWNTO 0), x2);
38      U3: mux PORT MAP (x1, x2, sel(3), y);
39 END alu;
40 -------------------------------------------------
```