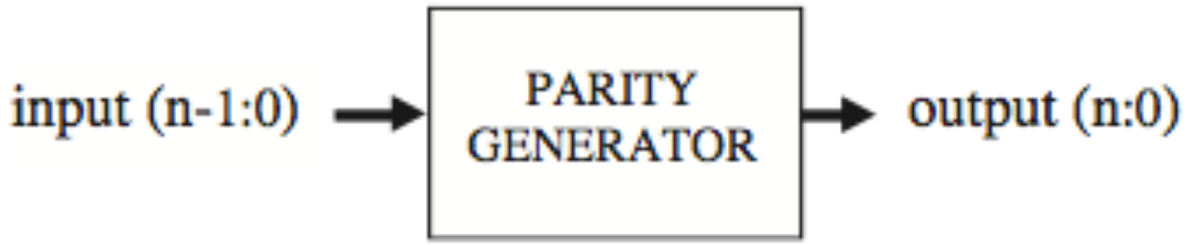
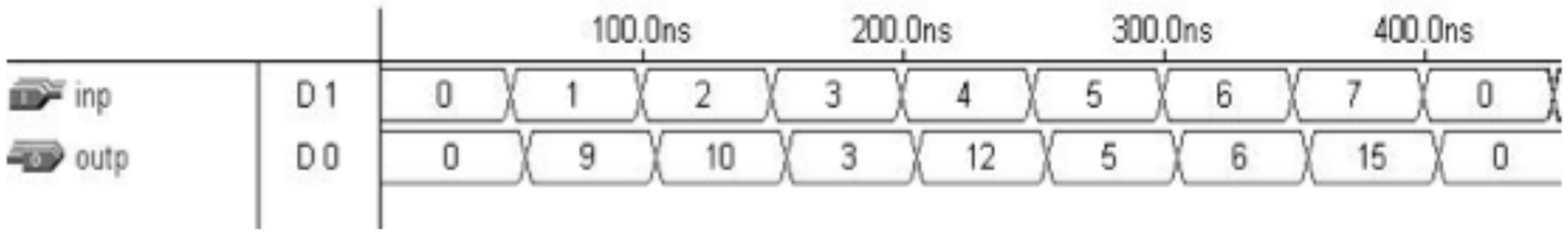
# CPE 462
# VHDL: Simulation and Synthesis

## Topic #08 - b) Generic Components & Packages

# Generic Component




- I want to create a sub-circuit that is a generic parity generator.
- As you know, a parity generator adds one bit to the input vector (on its left-hand side).
- 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.



```
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 ------------------------------------------------
```

# Generic Component

- There are two files: parity_gen.vhd and my_code.vhd
- my_code.vhd is the highest level in the hierarchical design

```vhdl
------ 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;
-------------------------------------------------
```

```vhdl
------ 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;
-------------------------------------------------
```

# Generic Component

- The component has a default generic size of 7
- However, when the component is instantiated, this default size will be overwritten.
- In the top level, the generic size will have the value of 2.

# Packages

# VHDL packages

- Packages are collections of reusable declarations and descriptions of VHDL types, subtypes, subprograms, aliases, constants, attributes, components, etc.

- Packages contain elements that can be globally shared among many number of design units.

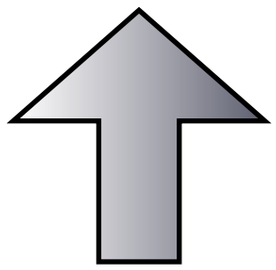
# Package syntax

```
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
- Second part is necessary only when one or more subprograms (FUNCTION or PROCEDURE) are declared in the upper part.

# Example #1 - Two different packages

```
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) := "111
9  END my_package;
10 ------------------------------------------------
```



This package only contains TYPES and CONSTANTS... so there is no need for a package body

```
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 ------------------------------------------------
```

This package has a function, so we need a package body!

# How to use a package?

## Step #1 - Create a package and save it as a new file

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

# Using a package

## Step #2 - Create a file which uses the 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;
-------------------------------------------------
```

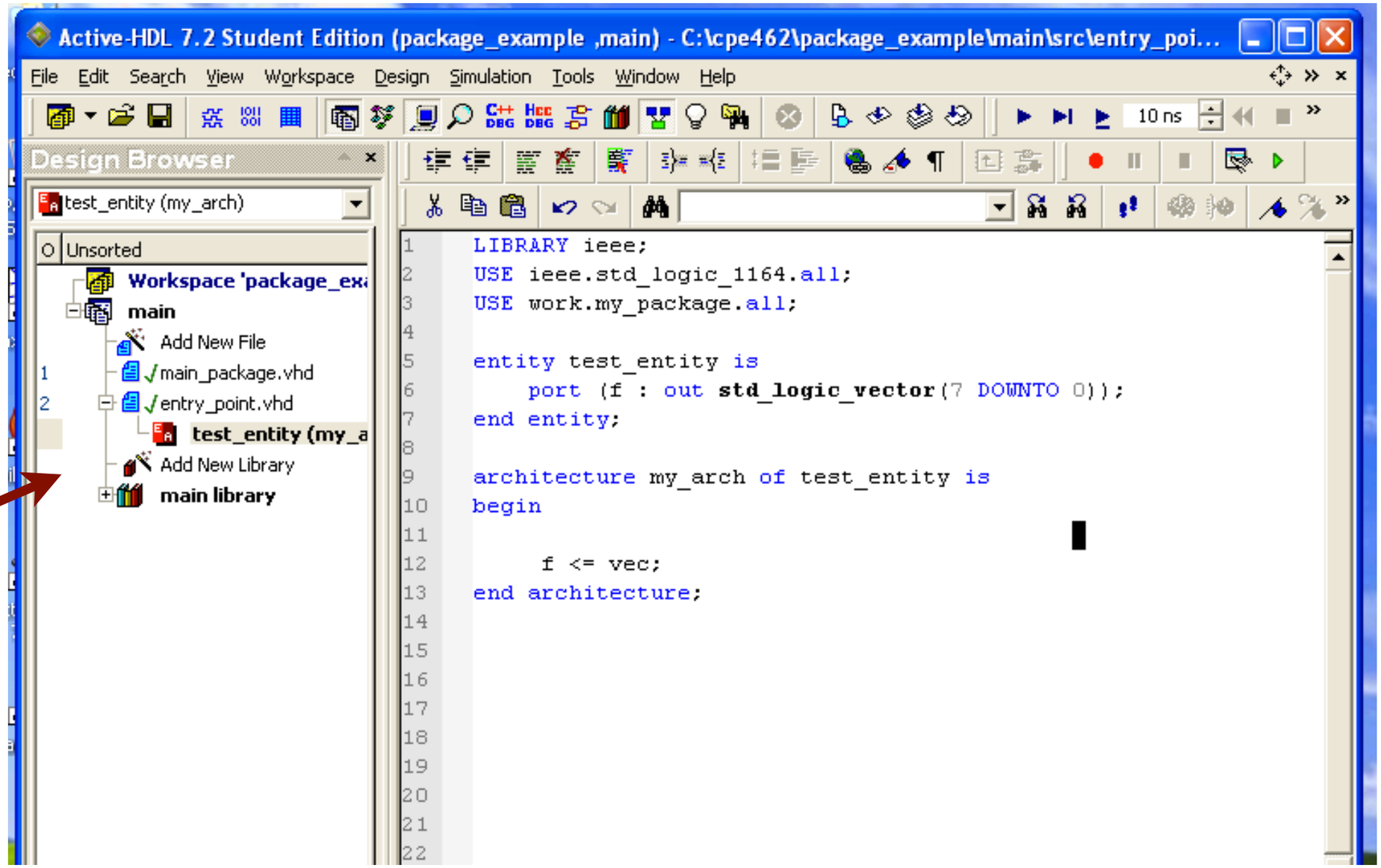
This is my package

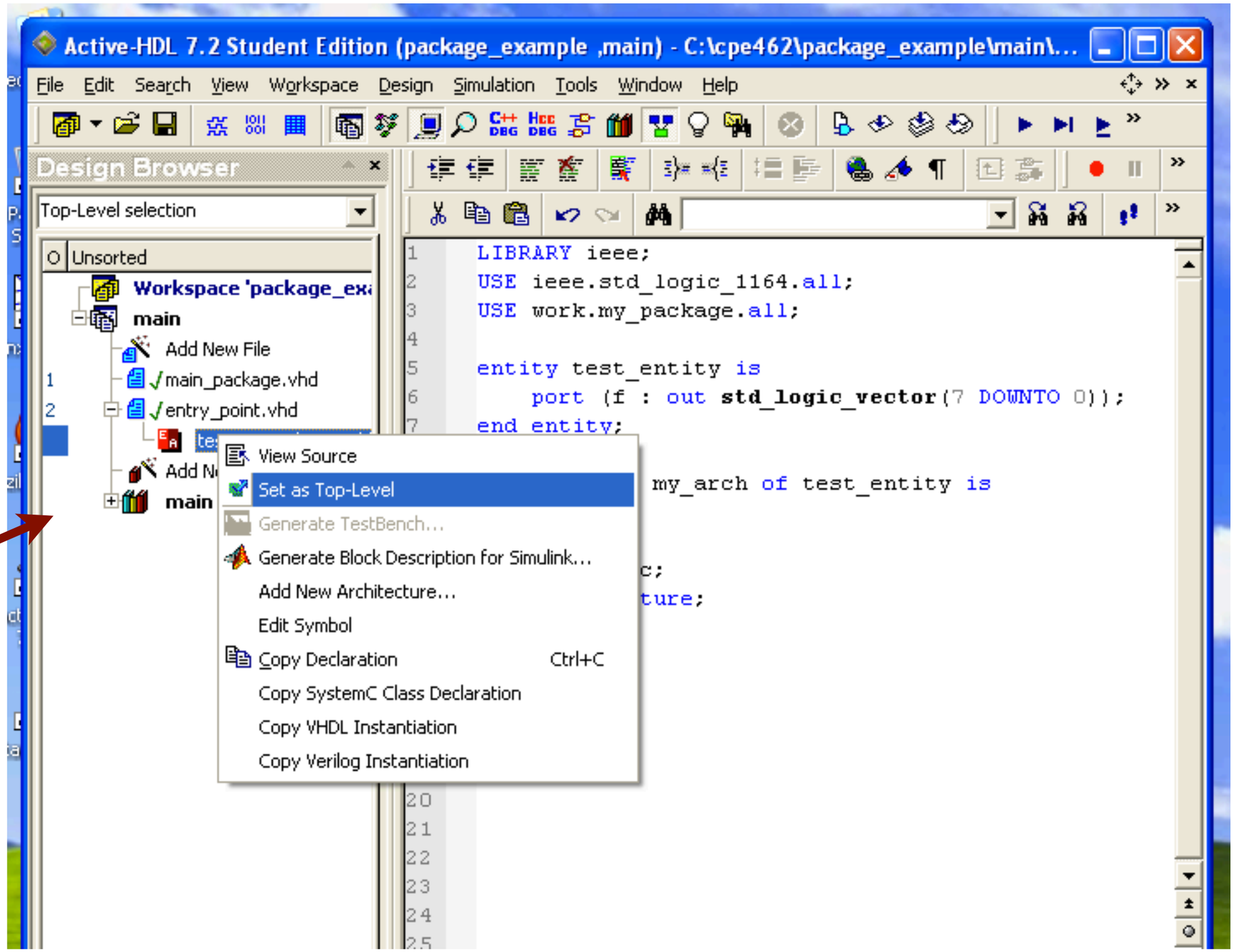
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;

entity test_entity is
    port (f : out std_logic_vector(7 DOWNTO 0));
end entity;

architecture my_arch of test_entity is
begin
     f <= vec;
end architecture;
```

**You must specify the package name here**

And on a different file is my program that uses the package

Here are my two files in Active HDL



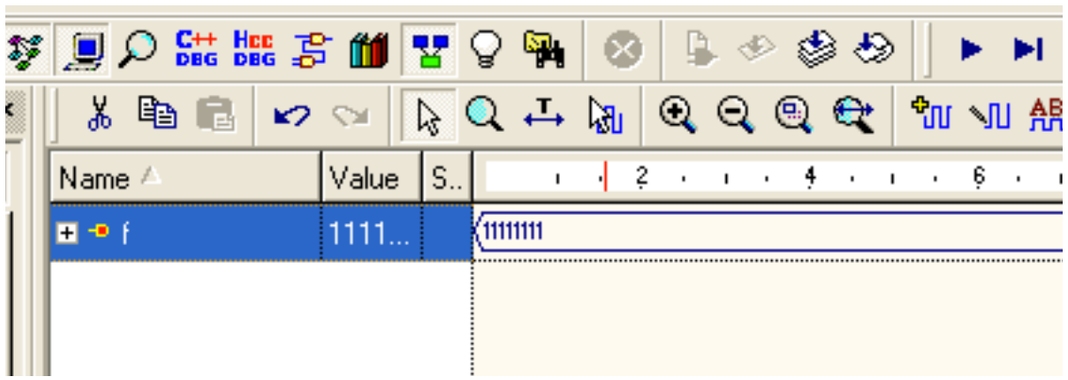



Make sure the program that uses the package is set as the top-level.

# Simulating the 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;
-------------------------------------------------
```

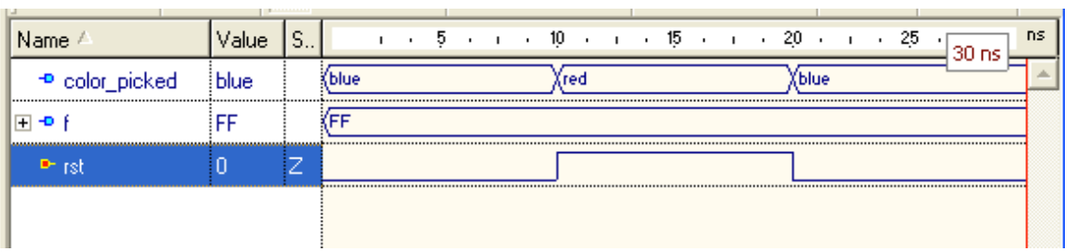
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;

entity test_entity is
    port (f : out std_logic_vector(7 DOWNTO 0));
end entity;

architecture my_arch of test_entity is
begin
      f <= vec;
end architecture;
```



# Any TYPE in the package can now be used as a entity input!

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

Same package

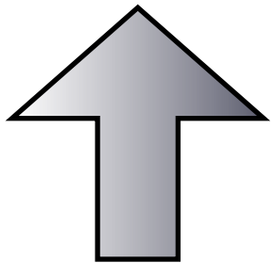



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;

entity test_entity is
port (   rst : in std_logic;
         color_picked : out color;
         f : out std_logic_vector(7 DOWNTO 0));
end entity;

architecture my_arch of test_entity is
begin
    process(rst)
    begin
        case rst is
            when '1' => color_picked <= red;
            when others => color_picked <= blue;
        end case;
    end process;

    f <= vec;
end  architecture;
```

# Packages with functions

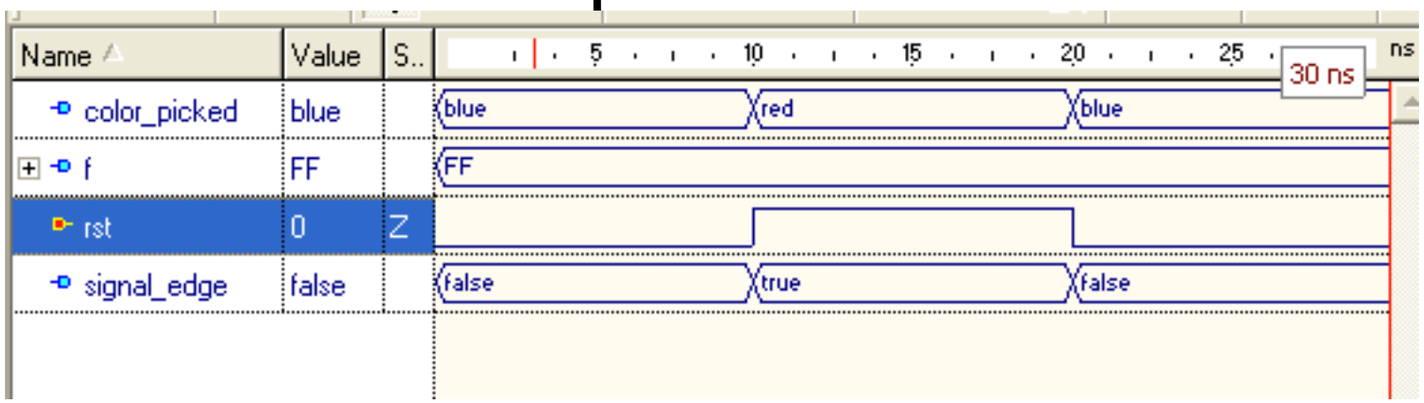
Yes... We haven't learned about functions... but you might as well see how you can use them in a package.

```
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 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 PACKAGE BODY;
-----------------------------------------------
```



Slightly modified package

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;

entity test_entity is
    port (rst : in std_logic;
          color_picked : out color;
          signal_edge : out boolean;
          f : out std_logic_vector(7 DOWNTO 0));
end entity;

architecture my_arch of test_entity is
begin
    process(rst)
    begin
        signal_edge<=positive_edge(rst);
        case rst is
            when '1' => color_picked <= red;
            when others => color_picked <= blue;
        end case;
    end process;

    f <= vec;
end architecture;
```



# Packages

- Examples of packages include:
  - Sub-programs (i.e. functions and procedures)
  - Data and type declarations (i.e. user records, enumerated data-types, constants,...)
- Entities and architectures may **NOT** be declared or defined in a package
- To use a package, it must be visible via the use construct

# How to include

# **Components** in **Packages**?

# Components in Packages (Step 1)

Step #1 - Create a separate VHDL file for each different component:

We are going to re-use our aoi.vhd file

```
--aoi.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity AOI is
port (
  A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC
);
end entity;

architecture V1 of AOI is
begin
  F <= not ((A and B) or (C and D));
end architecture;
```

# Components in Packages (Step 2)

Step #2 - Include the entity into the package

```
--main_package.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE my_package IS
 TYPE color IS (red, green, blue);

 component aoi
  port(A, B, C, D: in std_logic;
  F : out std_logic);
 end component;

END PACKAGE;
```

```
--aoi.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity AOI is
port (
  A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC
);
end entity;

architecture V1 of AOI is
begin
  F <= not ((A and B) or (C and D));
end architecture;
```

# Components in Packages (Step 3)

Step #3 - You may now use the component in the main program

```
--entry_point.vhd
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;

entity topLevel is
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel
is

begin
  G1: AOI port map (A,B,C,D,F);
end architecture;
```