POWER & ENERGY CONVERSION LABORATORY

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

FUNCTIONS and PROCEDURES
SYSTEM DESIGN

It is time to integrate several circuits into a bigger system

VHDL

VHSIC HARDWARE DESCRIPTION LANGUAGE

FUNCTIONS and PROCEDURES
FUNCTION Definition

- FUNCTIONS and PROCEDURES are:
    - known as *subprograms.*
    - a piece of ***sequential*** VHDL code
    - Very similar to PROCESS - The use of **IF, CASE and LOOP** are allowed (WAIT is not allowed).
    - PROCESS is thought for immediate use. However, **FUNCTIONS and PROCEDURES are thought for LIBRARY** allocation. That is their main difference.
    - The later does not means that FUNCTIONS and PROCEDURES can not be also used in main code,
- A FUNCTION has a **BODY** and a **CALL**, with the following syntax:

FUNCTION BODY:

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

In this example a, b are constants (ommision of CONSTANT is allowed), c is a SIGNAL. Data type of a and b is INTEGER and c is STD_LOGOC_VECTOR.

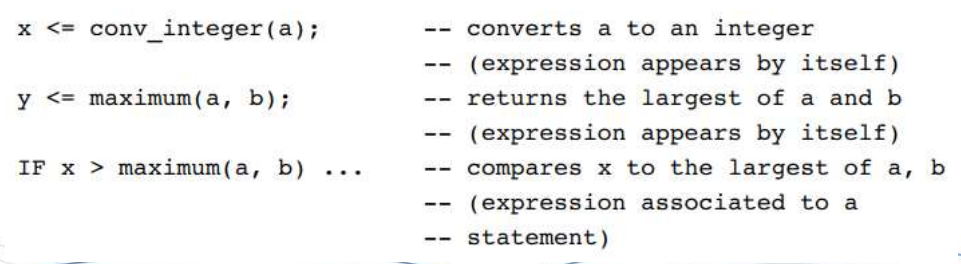
- ***<parameter list>*** can be CONSTANT or SIGNAL (can not be a VARIABLE)
- ***data_type*** can be any (STD_LOGIC, INTEGER, etc) but no range specification is allowed. (dont use: RANGE, TO , DOWNTO, etc)
- Only one return value is allowed.

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

FUNCTIONS and PROCEDURES
FUNCTION Definition

- FUNCTIONS and PROCEDURES are:
  - known as *subprograms.*
  - a piece of ***sequential*** VHDL code
  - Very similar to PROCESS - The use of **IF, CASE and LOOP** are allowed (WAIT is not allowed).
  - PROCESS is thought for immediate use. However, **FUNCTIONS and PROCEDURES are thought for LIBRARY** allocation. That is their main difference.
  - The later does not means that FUNCTIONS and PROCESS can not be also used in main code,
- A FUNCTION has a **BODY** and a **CALL**, with the following syntax:

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)
```

- A **FUNCTION is called** as part of an expression associated to a **concurrent or sequential statement.**

FUNCTIONS and PROCEDURES

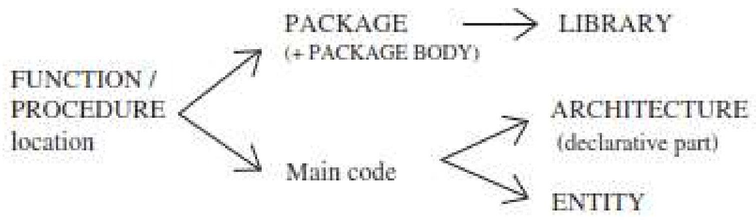
FUNCTION Example

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

This Function converts a STD_LOGIC_VECTOR into an INTEGER

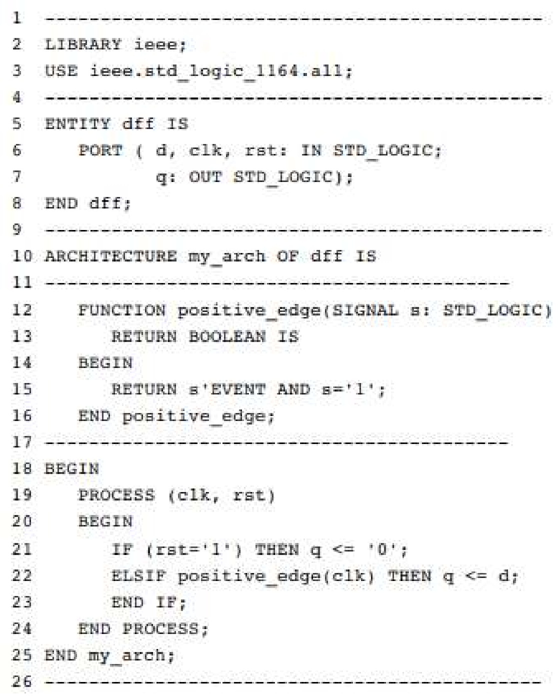
Function Location:

- Usually placed in a PACKAGE.
- Also possible within main code: Inside ARCHITECTURE or ENTITY
- When used in a PACKAGE, then PACKAGE BODY is necessary, containing the body of each FUNCTION, declared in the declarative part of PACKAGE.


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

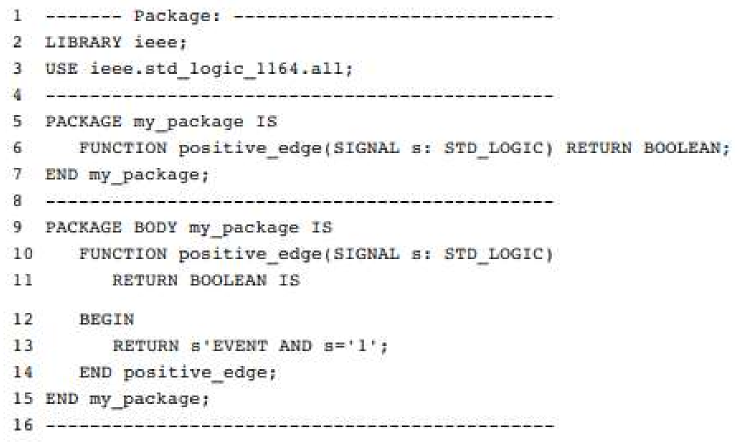
FUNCTIONS and PROCEDURES

FUNCTION Example

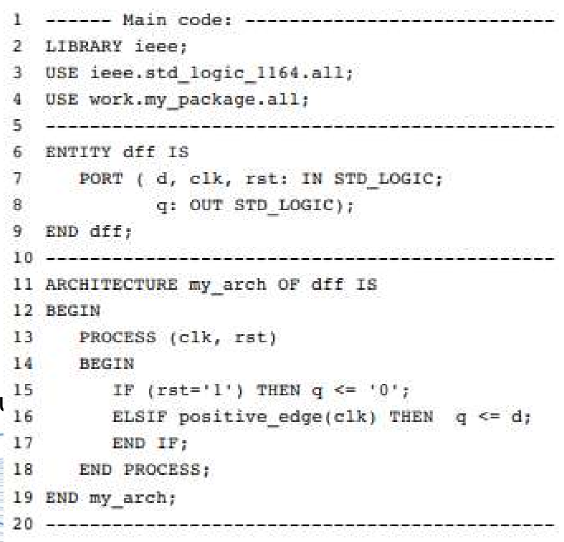


```
-----------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------------
ENTITY dff IS
   PORT ( d, clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
END dff;
-----------------------------------------------
ARCHITECTURE my_arch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

The “positive_edge” function is placed in the declarative part of ARCHITECTURE (main code) and u for creating a DFF within the ARCHITECTURE.



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



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

The “positive_edge” function is:
- Placed in a PACKAGE
- It can be reused and shred
- FUNCTION declared in PACKAGE.
- FUNCTION described in PACKAGE BODY

These two codes can be compiled as two separate files or compiled as a single file sabed as dff.vhd (entity name).

FUNCTIONS and PROCEDURES
PROCEDURE Definition

- PROCEDURE is very similar to FUNCTION and has the same purpose.
- However, a PROCEDURE can **RETURN more than one value.**
- Two parts are required to construct a PROCEDURE: PROCEDURE Body and CALL

PROCEDURE BODY:

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

```
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;
```

In this example a, b, c are input **a** is CONSTANT of type BIT (ommision of CONSTANT is allowed), **b and c** are SIGNALs of type BIT.
Signals x and y are the return signals, mode OUT and INOUT type BIT_VECTOR and INTEGER.

- ***<parameter list>*** can be CONSTANT, SIGNAL or VARIABLE. (input and output parameters.)
- More than one return value is allowed.
- For inputs CONSTANT is the default parameter type, while for output is VARIABLE.
- Note that :
  - WAIT , SIGNAL declarations and COMPONENTS are not synthesizable either for FUNCTIONS or PROCEDURES.
  - Exceptionally SIGNAL can be declared in PROCEDURE, but then PROCEDURE must be declared in PROCESS.
  - A synthesizable procedure should not infer registers, i.e. no use of WAIT of any other edge detection.

FUNCTIONS and PROCEDURES
PROCEDURE Definition

- PROCEDURE is very similar to FUNCTION and has the same purpose.
- However, a PROCEDURE can **RETURN more than one value.**
- Two parts are required to construct a PROCEDURE: PROCEDURE Body and CALL

PROCEDURE CALL:

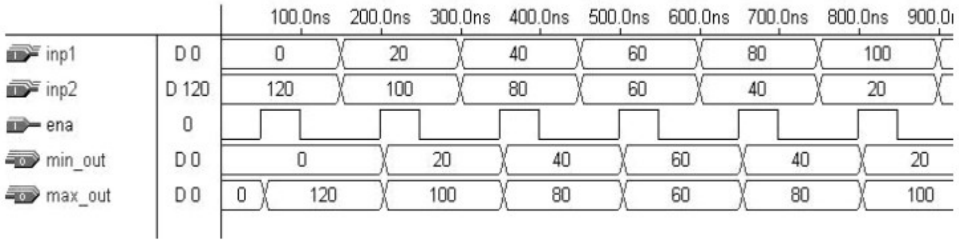
```
compute_min_max(in1, in2, 1n3, out1, out2);
             -- statement by itself

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 call is a statement on its own. Not like FUNCTION that is part of an expression.
- PROCEDURE can appear but itself or associated to a statement (concurrent or sequential).

FUNCTIONS and PROCEDURES
PROCEDURE in the Main Code

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

- This code uses PROCEDURE called sort.
- PROCEDURE is located in the declarative part of ARCHITECTURE (main code).
- PROCEDURE call is a statement by its own.
- **What does our sort PROCEDURE do?**



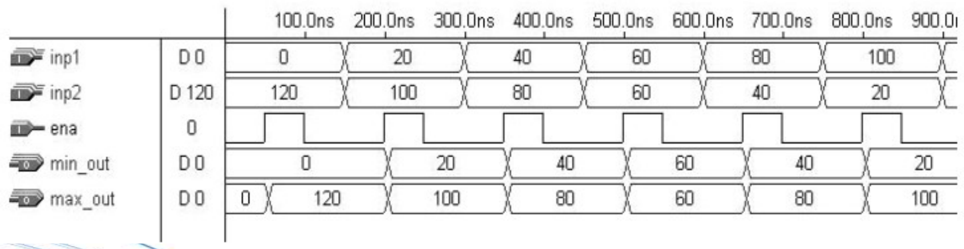
FUNCTIONS and PROCEDURES

PROCEDURE in 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;
-------------------------------------------------
```

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

- This code uses PROCEDURE called sort.
- PROCEDURE is in a PACKAGE called my_package.
- Now this PROCEDURE can be reused and shared.
- PROCEDURE call is a statement by its own.
- **What does our sort PROCEDURE do?**



FUNCTIONS and PROCEDURES

Summary

FUNCTION	PROCEDURE	
Zero or more input parameters. They can only be CONSTANTS (default) or SIGNALS (VARIABLES not allowed)	Any number of IN, OUT or INOUT parameters. They can be CONSTANTS, VARIABLES or SIGNALS	Differences
By default, input parameter is CONSTANT	By default, input (IN) parameter is CONSTANT, and output (OUR and INOUT) parameters are VARIABLE.	
Called as part of an expression	Statement by its own	
WAIT and COMPONENTS are not Synthesizable	WAIT and COMPONENTS are not Synthesizable	In common
Usually placed in PACKAGE. PACKAGE BODY is neccesary.	Usually placed in PACKAGE. PACKAGE BODY is neccesary.	
Less common, can be placed also into ENTITY or ARCHITECTIURE (main code)	Less common, can be placed also into ENTITY or ARCHITECTIURE (main code)	

FUNCTIONS and PROCEDURES
FUNCTION Exercise

Create a function named ***mult()*** that can multiply two UNSIGNED values, returning the UNSIGNED product. The parameters to the FUNCTION do not need to have the same number of bits and their order TO/DOWNTO can be any.

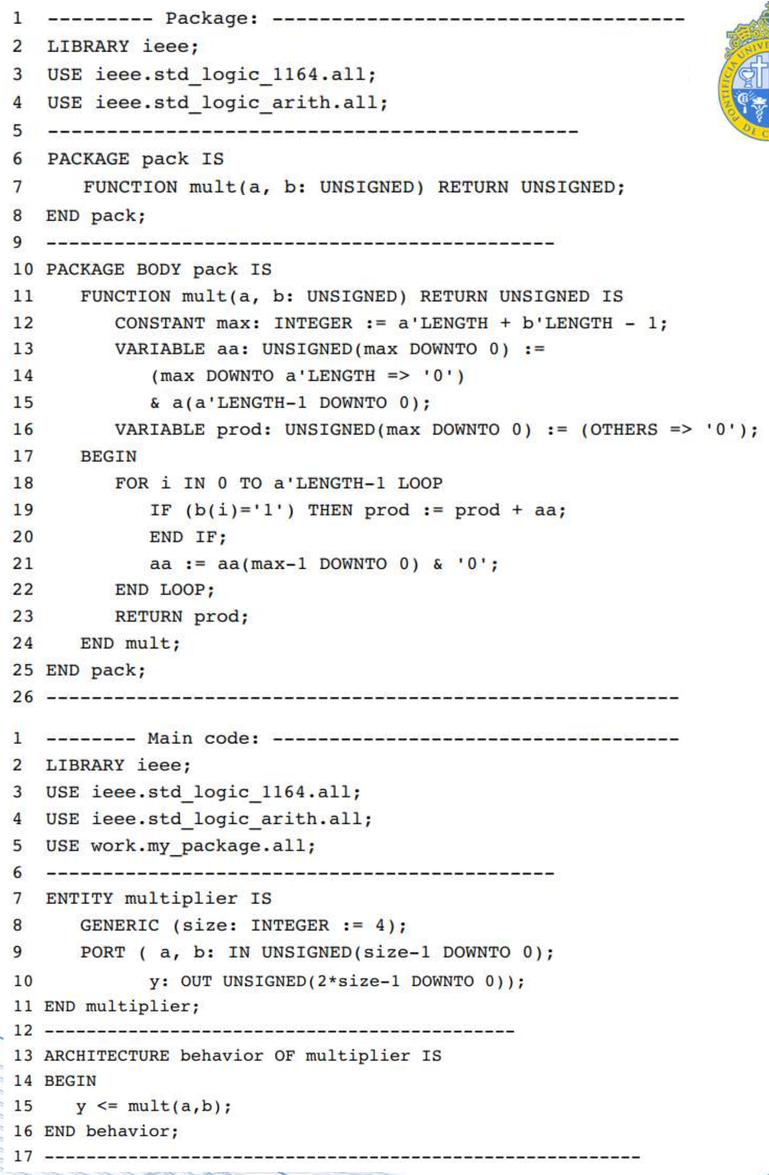
```
--------- Package: ------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
---------------------------------------------
PACKAGE pack IS
   FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED;
END pack;
---------------------------------------------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

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

FUNCTIONS and PROCEDURES
PROCEDURE Exercise

Use the same FUNCTION made before and transform it into a procedure within a PACKAGE. Add a second output to the procedure at your choice.

Previous example:
Use a function named ***mult()*** that can multiply two UNSIGNED values, returning the UNSIGNED product. The parameters to the FUNCTION do not need to have the same number of bits and their order TO/DOWNTO can be any.

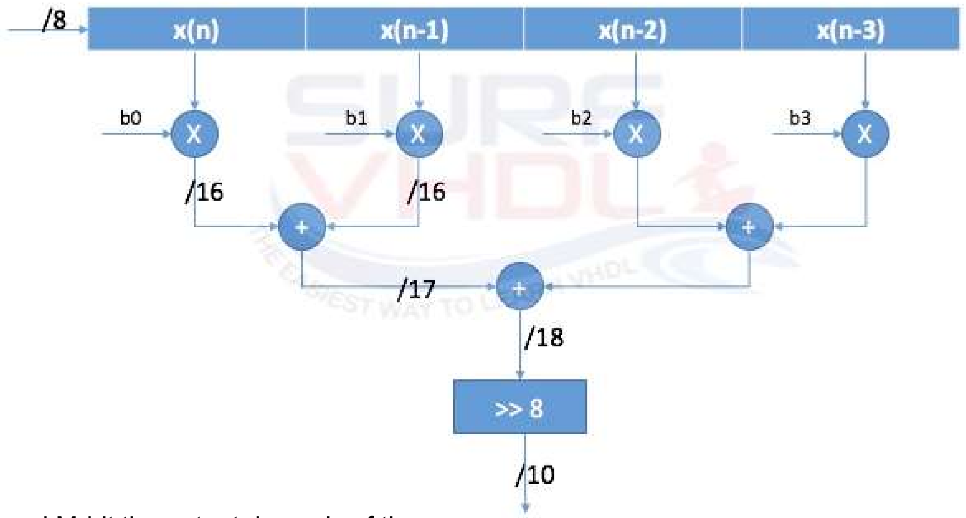


```
--------- Package: -----------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
---------------------------------------------
PACKAGE pack IS
   FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED;
END pack;
---------------------------------------------
PACKAGE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

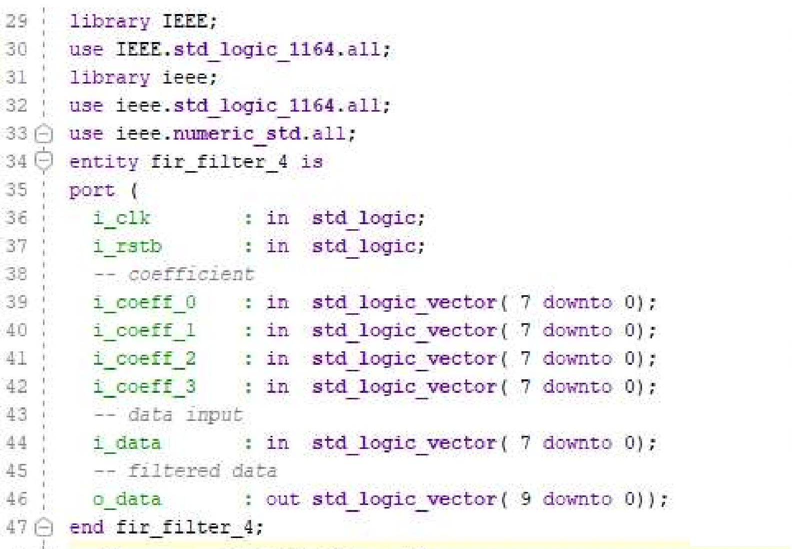
```
-------- Main code: ----------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE work.my_package.all;
---------------------------------------------
ENTITY multiplier IS
   GENERIC (size: INTEGER := 4);
   PORT ( a, b: IN UNSIGNED(size-1 DOWNTO 0);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

Exercise
Preparation for Project

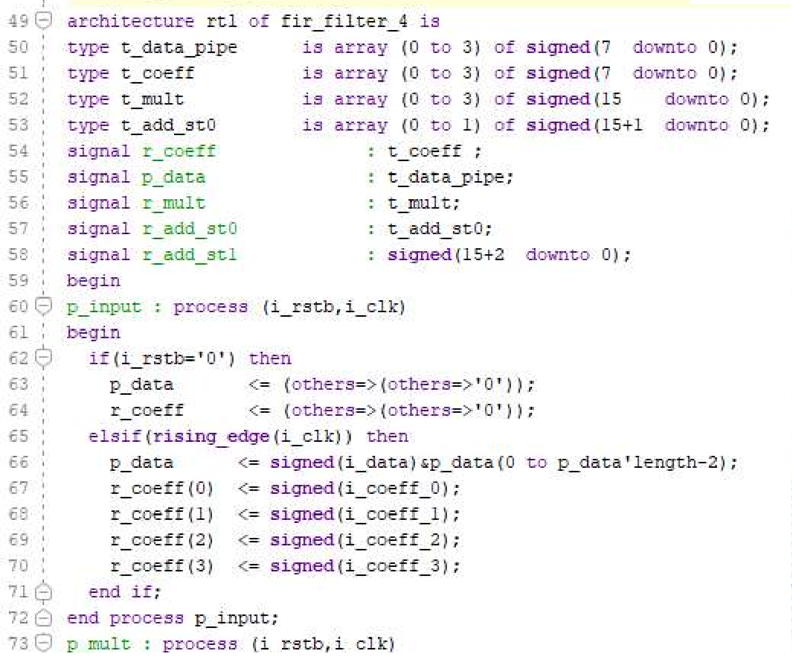
Develop a Finite Impulse Response Filter (FIR) using the last four entries of the signal. Consider generic coeficcients of 8-bits, input of 8bit and output of 10bits, as the figure :



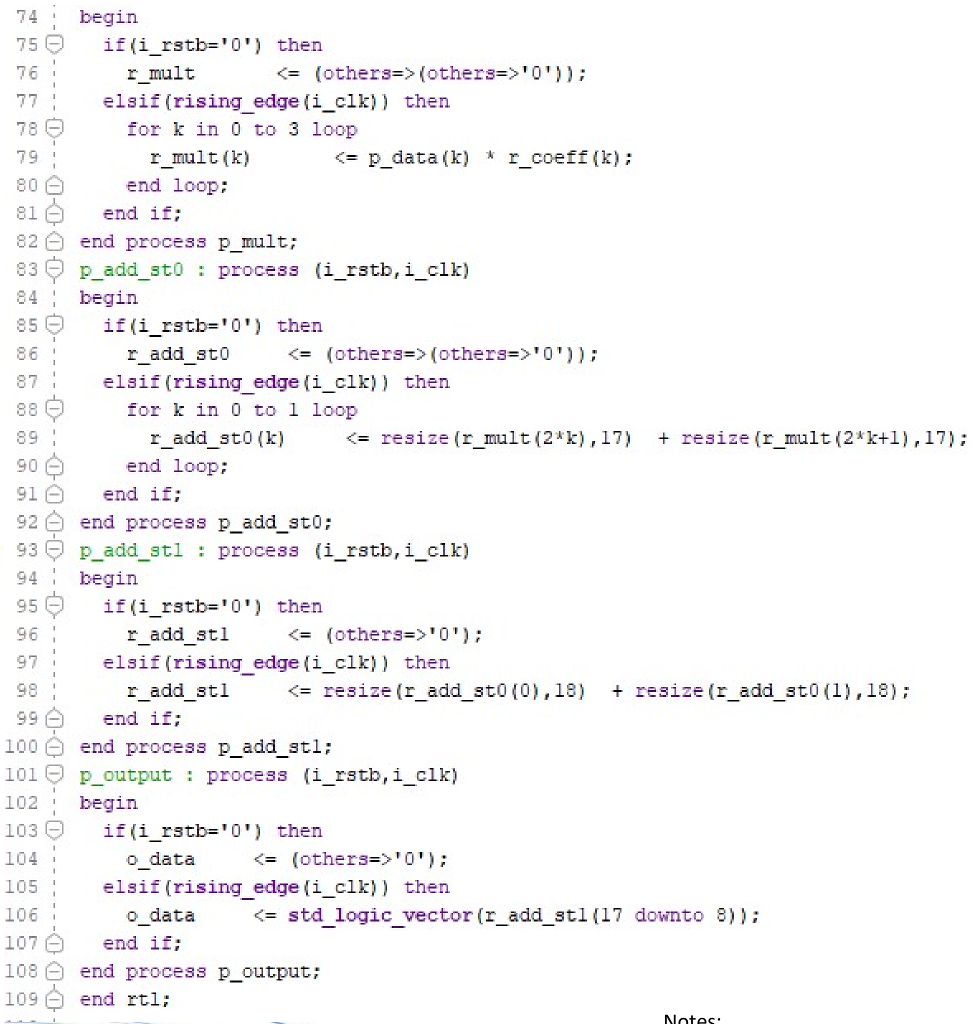
Note:When you multiply two numbers of N-bit and M-bit the output dynamic of the multiplication result is (N+M)-bits. When you perform addition, the number of bit of the result will be incremented by 1.



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity fir_filter_4 is
port (
  i_clk        : in  std_logic;
  i_rstb       : in  std_logic;
  -- coefficient
  i_coeff_0    : in  std_logic_vector( 7 downto 0);
  i_coeff_1    : in  std_logic_vector( 7 downto 0);
  i_coeff_2    : in  std_logic_vector( 7 downto 0);
  i_coeff_3    : in  std_logic_vector( 7 downto 0);
  -- data input
  i_data       : in  std_logic_vector( 7 downto 0);
  -- filtered data
  o_data       : out std_logic_vector( 9 downto 0));
end fir_filter_4;
```



```vhdl
architecture rtl of fir_filter_4 is
type t_data_pipe      is array (0 to 3) of signed(7  downto 0);
type t_coeff          is array (0 to 3) of signed(7  downto 0);
type t_mult           is array (0 to 3) of signed(15    downto 0);
type t_add_st0        is array (0 to 1) of signed(15+1  downto 0);
signal r_coeff              : t_coeff ;
signal p_data               : t_data_pipe;
signal r_mult               : t_mult;
signal r_add_st0            : t_add_st0;
signal r_add_st1            : signed(15+2  downto 0);
begin
p_input : process (i_rstb,i_clk)
begin
  if(i_rstb='0') then
    p_data       <= (others=>(others=>'0'));
    r_coeff      <= (others=>(others=>'0'));
  elsif(rising_edge(i_clk)) then
    p_data      <= signed(i_data)&p_data(0 to p_data'length-2);
    r_coeff(0)  <= signed(i_coeff_0);
    r_coeff(1)  <= signed(i_coeff_1);
    r_coeff(2)  <= signed(i_coeff_2);
    r_coeff(3)  <= signed(i_coeff_3);
  end if;
end process p_input;
p_mult : process (i_rstb,i_clk)
```



```vhdl
begin
  if(i_rstb='0') then
    r_mult       <= (others=>(others=>'0'));
  elsif(rising_edge(i_clk)) then
    for k in 0 to 3 loop
      r_mult(k)       <= p_data(k) * r_coeff(k);
    end loop;
  end if;
end process p_mult;
p_add_st0 : process (i_rstb,i_clk)
begin
  if(i_rstb='0') then
    r_add_st0     <= (others=>(others=>'0'));
  elsif(rising_edge(i_clk)) then
    for k in 0 to 1 loop
      r_add_st0(k)     <= resize(r_mult(2*k),17)  + resize(r_mult(2*k+1),17);
    end loop;
  end if;
end process p_add_st0;
p_add_st1 : process (i_rstb,i_clk)
begin
  if(i_rstb='0') then
    r_add_st1     <= (others=>'0');
  elsif(rising_edge(i_clk)) then
    r_add_st1     <= resize(r_add_st0(0),18)  + resize(r_add_st0(1),18);
  end if;
end process p_add_st1;
p_output : process (i_rstb,i_clk)
begin
  if(i_rstb='0') then
    o_data     <= (others=>'0');
  elsif(rising_edge(i_clk)) then
    o_data     <= std_logic_vector(r_add_st1(17 downto 8));
  end if;
end process p_output;
end rtl;
```

Notes:
numeric library

END-FUNCTIONS and PROCEDURES
SYSTEM DESIGN

It is time to integrate several circuits into a bigger system

VHDL

VHSIC HARDWARE DESCRIPTION LANGUAGE