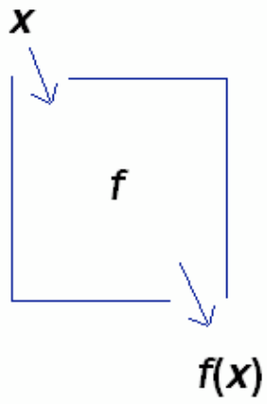
CPE 462
VHDL: Simulation and Synthesis

Topic #08 - c) Functions

What is a FUNCTION in VHDL?



- In mathematics a function is just a box... You add some inputs, and you will get some modified outputs

- A FUNCTION is a section of sequential code.

- Its purpose is to create new functions to deal with commonly encountered problems, like data type conversions, logical operations, arithmetic computations, and new operators and attributes.

- By writing such code as a FUNCTION, it can be shared and reused, also propitiating the main code to be shorter and easier to understand.

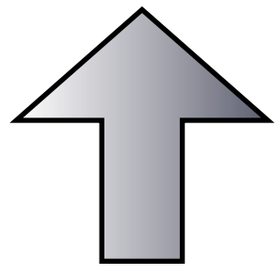
Function

- The same statements that can be used in a process (IF, WAIT, CASE, and LOOP) can also be used in a function, with the exception of WAIT.
- Other two prohibitions in a function are SIGNAL declarations and COMPONENT instantiations.
- To construct and use a function, two parts are necessary: the function itself (function body) and a call to the function.

Example: convert std_logic to 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);
...
```



Typical function call!

- The FUNCTION presented parameter of type STD_LOGIC_ VECTOR into an INTEGER.
- Code is generic, that is, it works for any range or order (TO/ DOWNTO) of the input STD_LOGIC_VECTOR parameter.

Syntax - function name

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

- You need a function name...
- Can't be a name already used in VHDL

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

Syntax - Specifying input parameters

- You now write the function inputs and their data types

- You **CAN'T** use VARIABLES as function inputs, only SIGNALS

- There can be any number of such parameters (even zero), each separated by commas

- You can use CONSTANT as an input

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

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

Syntax - Do **NOT** include ranges in function parameters

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

- IMPORTANT: you CAN'T specify ranges in the input & output parameters

- For example, do not enter RANGE when using INTEGER

- ... or TO/ DOWNTO when using STD_LOGIC_VECTOR).

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

Syntax - Multiple input parameters

This is how you would define multiple input parameters on other functions

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

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

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

Syntax - Function output type

- There can be only one return data type

- In functions only a single return value, in procedures you can have more than one.

- In this particular function, the return data type is an INTEGER.

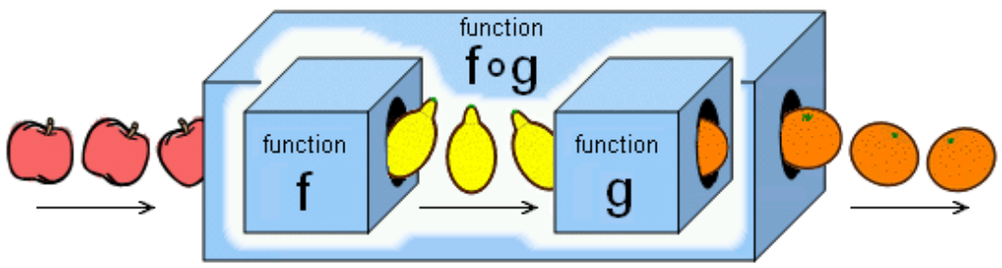
```
FUNCTION function_name [<parameter list>] RETURN data_type IS
    [declarations]
BEGIN
    (sequential statements)
END function_name;
```

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

Syntax - Function code

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

- You now write a bunch of code that will use the input signals to create some output




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

Syntax - Function output

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

- Use the RETURN keyword to return a specific signal out of the function.

- The output of the function is an 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;
```

Different ways of using functions

This is the function

```
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)
```

Where can we place functions? Inside your working program

- Last class we saw how we could include functions inside packages.
- For simplicity sake, we can include functions inside the main VHDL code.

```
1  ----------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY dff IS
6     PORT ( d, clk, rst: IN STD_LOGIC;
7            q: OUT STD_LOGIC);
8  END dff;
9  ----------------------------------------
10 ARCHITECTURE my_arch OF dff IS
11 ----------------------------------------
12    FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
13       RETURN BOOLEAN IS
14    BEGIN
15       RETURN s'EVENT AND s='1';
16    END positive_edge;
17 ----------------------------------------
18 BEGIN
19    PROCESS (clk, rst)
20    BEGIN
21       IF (rst='1') THEN q <= '0';
22       ELSIF positive_edge(clk) THEN q <= d;
23       END IF;
24    END PROCESS;
25 END my_arch;
26 ----------------------------------------
```

Where can we place functions? Inside a package

```
1  -------- Package: ---------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  PACKAGE my_package IS
6     FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
7        RETURN INTEGER;
8  END my_package;
9  ---------------------------------------------
10 PACKAGE BODY my_package IS
11    FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
12          RETURN INTEGER IS
13       VARIABLE result: INTEGER RANGE 0 TO 2**vector'LENGTH-1;
14    BEGIN
15       IF (vector(vector'HIGH)='1') THEN result:=1;
16       ELSE result:=0;
17       END IF;
18       FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
19          result:=result*2;
20          IF(vector(i)='1') THEN result:=result+1;
21          END IF;
22       END LOOP;
23       RETURN result;
24    END conv_integer;
25 END my_package;
26 ---------------------------------------------
```

```
1  -------- Main code: -------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_package.all;
5  ---------------------------------------------
6  ENTITY conv_int2 IS
7     PORT ( a: IN STD_LOGIC_VECTOR(0 TO 3);
8            y: OUT INTEGER RANGE 0 TO 15);
9  END conv_int2;
10 ---------------------------------------------
11 ARCHITECTURE my_arch OF conv_int2 IS
12 BEGIN
13    y <= conv_integer(a);
14 END my_arch;
15 ---------------------------------------------
```

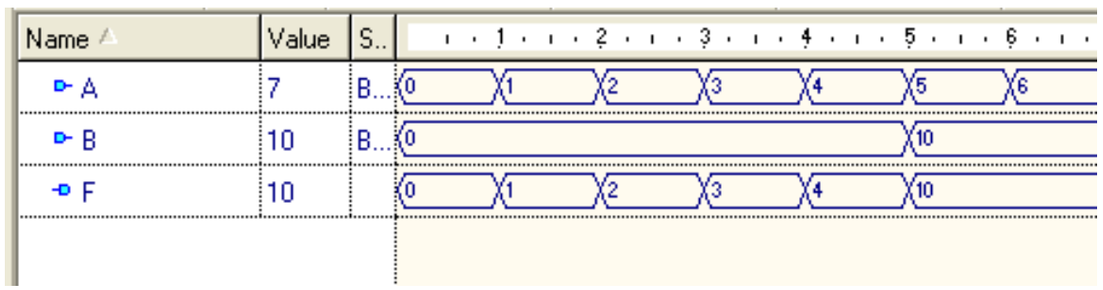
- Using packages requires two separate files
- You must include the working package name in your main code
- **Important** : I can declare internal function variables!

Simple VHDL functions

Nuno Alves (nalves@wne.edu), College of Engineering

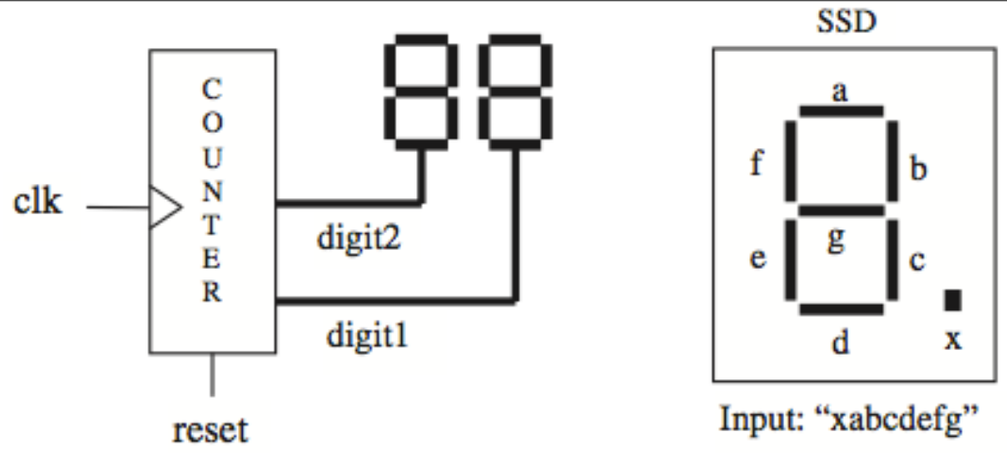
Maximum of two integers

- Very simple function... takes two values and determines which one is greater.




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

entity basic_function is
  port (A, B: in integer;
  F : out integer);
end entity;

architecture myarch of basic_function is

function max_value (signal value1,value2 : integer)
return integer is
begin
    if (value1<value2) then return value2;
    else return value1;
    end if;
end function;

begin
  process(A,B)
  begin
    F <= max_value(A,B);
  end process;

end architecture;
```

Useful function



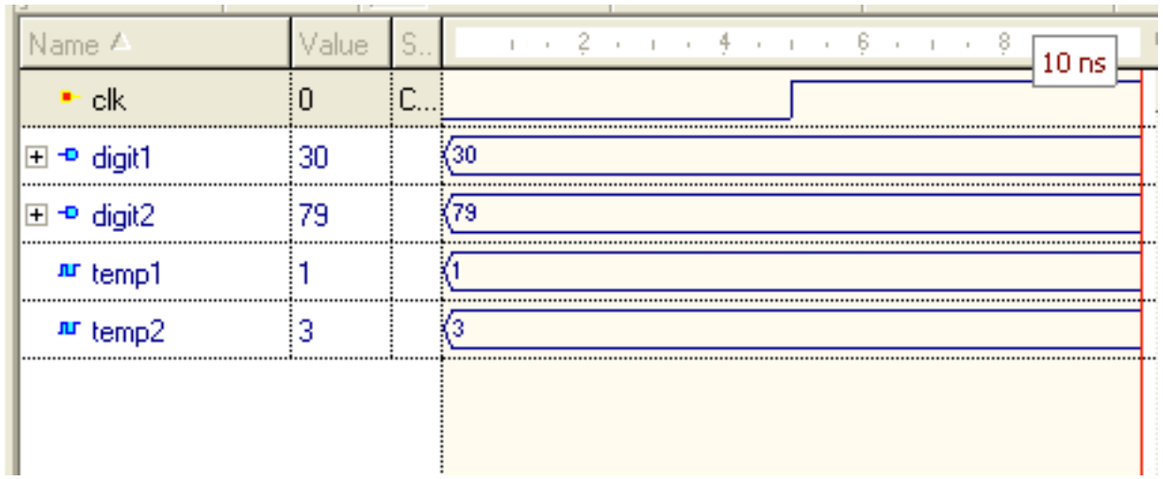
Input: "xabcdefg"

```
1  ------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------------
5  ENTITY counter IS
6     PORT (clk, reset : IN STD_LOGIC;
7           digit1, digit2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
8  END counter;
9  ------------------------------------------------
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12    PROCESS(clk, reset)
13       VARIABLE temp1: INTEGER RANGE 0 TO 10;
14       VARIABLE temp2: INTEGER RANGE 0 TO 10;
15    BEGIN
16    ---- counter: ---------------------
17       IF (reset='1') THEN
18          temp1 := 0;
19          temp2 := 0;
20       ELSIF (clk'EVENT AND clk='1') THEN
21          temp1 := temp1 + 1;
22          IF (temp1=10) THEN
23             temp1 := 0;
24             temp2 := temp2 + 1;
25             IF (temp2=10) THEN
26                temp2 := 0;
```

redundant code?

```
27             END IF;
28          END IF;
29       END IF;
30       ---- BCD to SSD conversion: --------
31       CASE temp1 IS
32          WHEN 0 => digit1 <= "1111110";    --7E
33          WHEN 1 => digit1 <= "0110000";    --30
34          WHEN 2 => digit1 <= "1101101";    --6D
35          WHEN 3 => digit1 <= "1111001";    --79
36          WHEN 4 => digit1 <= "0110011";    --33
37          WHEN 5 => digit1 <= "1011011";    --5B
38          WHEN 6 => digit1 <= "1011111";    --5F
39          WHEN 7 => digit1 <= "1110000";    --70
40          WHEN 8 => digit1 <= "1111111";    --7F
41          WHEN 9 => digit1 <= "1111011";    --7B
42          WHEN OTHERS => NULL;
43       END CASE;
44       CASE temp2 IS
45          WHEN 0 => digit2 <= "1111110";    --7E
46          WHEN 1 => digit2 <= "0110000";    --30
47          WHEN 2 => digit2 <= "1101101";    --6D
48          WHEN 3 => digit2 <= "1111001";    --79
49          WHEN 4 => digit2 <= "0110011";    --33
50          WHEN 5 => digit2 <= "1011011";    --5B
51          WHEN 6 => digit2 <= "1011111";    --5F
52          WHEN 7 => digit2 <= "1110000";    --70
53          WHEN 8 => digit2 <= "1111111";    --7F
54          WHEN 9 => digit2 <= "1111011";    --7B
55          WHEN OTHERS => NULL;
56       END CASE;
57    END PROCESS;
58 END counter;
```

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

entity basic_function is
     port(clk : in std_logic;
          digit1, digit2 : out std_logic_vector(6 downto 0));
end entity;

architecture myarch of basic_function is
     function ssd_function (signal value : integer) return std_logic_vector is

     begin
     case value is
          when 0 => return "1111110";
          when 1 => return "0110000";
          when 2 => return "1101101";
          when 3 => return "1111001";
          when 4 => return "0110011";
          when 5 => return "1011011";
          when 6 => return "1011111";
          when 7 => return "1110000";
          when 8 => return "1111111";
          when 9 => return "1111011";
          when others => null;
     end case;

end function;

     signal temp1 : integer := 1;
     signal temp2 : integer := 3;

begin
     process (clk)
     begin
          digit1 <= ssd_function(temp1);
          digit2 <= ssd_function(temp2);
     end process;

end architecture;
```



More elaborate VHDL functions

Variable initialization

```
1  ----------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY shift_left IS
6     GENERIC (size: INTEGER := 4);
7     PORT ( a: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
8            x, y, z: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
9  END shift_left;
10 ----------------------------------------
11 ARCHITECTURE behavior OF shift_left IS
12 ----------------------------------------
13    FUNCTION slar (arg1: STD_LOGIC_VECTOR; arg2: NATURAL)
14          RETURN STD_LOGIC_VECTOR IS
15       VARIABLE input: STD_LOGIC_VECTOR(size-1 DOWNTO 0) := arg1;
16       CONSTANT size : INTEGER := arg1'LENGTH;
17       VARIABLE copy: STD_LOGIC_VECTOR(size-1 DOWNTO 0)
18          := (OTHERS => arg1(arg1'RIGHT));
19       VARIABLE result: STD_LOGIC_VECTOR(size-1 DOWNTO 0);
20    BEGIN
21       IF (arg2 >= size-1) THEN result := copy;
22       ELSE result := input(size-1-arg2 DOWNTO 1) &
23          copy(arg2 DOWNTO 0);
24       END IF;
25       RETURN result;
26    END slar;
27 ----------------------------------------
28 BEGIN
29    x <= slar(a, 0);
30    y <= slar(a, 1);
31    z <= slar(a, 2);
32 END behavior;
```

- This function shifts a STD_LOGIC_VECTOR value to the left. Arg1 is the vector to be shifted. Arg2 second specifies the amount of shift. the input vector.

- You are allowed to initialize variable inside functions from function arguments.

- The variable copy will be initialized to the value of the rightmost position of arg1.

Another function example: Multiplier

- This function multiplies two UNSIGNED values, returning their UNSIGNED product.
- The parameters passed to the function do not need to have the same number of bits, and their order (TO/DOWNTO) can be any.
- The function was installed in a package called pack.
- An application example (main code) is also presented on the next slide.

```
1  --------- Package: ------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;
5  ------------------------------------------
6  PACKAGE pack IS
7     FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED;
8  END pack;
9  ------------------------------------------
10 PACKAGE BODY pack IS
11    FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED IS
12       CONSTANT max: INTEGER := a'LENGTH + b'LENGTH - 1;
13       VARIABLE aa: UNSIGNED(max DOWNTO 0) :=
14          (max DOWNTO a'LENGTH => '0')
15          & a(a'LENGTH-1 DOWNTO 0);
16       VARIABLE prod: UNSIGNED(max DOWNTO 0) := (OTHERS => '0');
17    BEGIN
18       FOR i IN 0 TO a'LENGTH-1 LOOP
19          IF (b(i)='1') THEN prod := prod + aa;
20          END IF;
21          aa := aa(max-1 DOWNTO 0) & '0';
22       END LOOP;
23       RETURN prod;
24    END mult;
25 END pack;
26 ------------------------------------------------
```

Using the multiplier function that is inside a package

```
1  -------- Package: ------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;
5  ----------------------------------------------
6  PACKAGE pack IS
7     FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED;
8  END pack;
9  ----------------------------------------------
10 PACKAGE BODY pack IS
11    FUNCTION mult(a, b: UNSIGNED) RETURN UNSIGNED IS
12       CONSTANT max: INTEGER := a'LENGTH + b'LENGTH - 1;
13       VARIABLE aa: UNSIGNED(max DOWNTO 0) :=
14          (max DOWNTO a'LENGTH => '0')
15          & a(a'LENGTH-1 DOWNTO 0);
16       VARIABLE prod: UNSIGNED(max DOWNTO 0) := (OTHERS => '0');
17    BEGIN
18       FOR i IN 0 TO a'LENGTH-1 LOOP
19          IF (b(i)='1') THEN prod := prod + aa;
20          END IF;
21          aa := aa(max-1 DOWNTO 0) & '0';
22       END LOOP;
23       RETURN prod;
24    END mult;
25 END pack;
26 ----------------------------------------------------
```

```
1  -------- Main code: ------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;
5  USE work.my_package.all;
6  ----------------------------------------------
7  ENTITY multiplier IS
8     GENERIC (size: INTEGER := 4);
9     PORT ( a, b: IN UNSIGNED(size-1 DOWNTO 0);
10           y: OUT UNSIGNED(2*size-1 DOWNTO 0));
11 END multiplier;
12 ----------------------------------------------
13 ARCHITECTURE behavior OF multiplier IS
14 BEGIN
15    y <= mult(a,b);
16 END behavior;
17 ----------------------------------------------------------
```

How to use user defined data types in functions?

Using user defined data-types as function inputs

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

entity basic_function is
     port (clk : in std_logic);
end entity;

architecture myarch of basic_function
is
signal mountainA : montain := everest;
signal mountainB : montain :=
kilimanjaro;
signal F : montain;
begin

     process (clk)
     begin
     F <=
tallest_montain(mountainA,mountainB);
     end process;

end architecture;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

package my_awesome_package is
type montain is (washington,kilimanjaro,everest);

function tallest_montain (signal a,b: montain) return
montain;
end my_awesome_package;

package body my_awesome_package is

function tallest_montain (signal a,b: montain) return
montain is
begin
     if (a=everest) or (b=everest) then return everest;
     elsif (a=kilimanjaro) or (b=kilimanjaro)then return
kilimanjaro;
     else return  washington;
     end if;

end function;

end package body;
```

Overloading Operators

List of VHDL operators

```
**    exponentiation,  numeric ** integer,  result numeric
abs   absolute value,  abs numeric,  result numeric
not   complement,      not logic or boolean,  result same

*     multiplication,  numeric * numeric,  result numeric
/     division,        numeric / numeric,  result numeric
mod   modulo,          integer mod integer,  result integer
rem   remainder,       integer rem integer,  result integer

+     unary plus,      + numeric,  result numeric
-     unary minus,     - numeric,  result numeric

+     addition,        numeric + numeric,  result numeric
-     subtraction,     numeric - numeric,  result numeric
&     concatenation,   array or element & array or element,
                         result array

sll   shift left logical,     logical array sll integer,  result same
srl   shift right logical,    logical array srl integer,  result same
sla   shift left arithmetic,  logical array sla integer,  result same
sra   shift right arithmetic, logical array sra integer,  result same
rol   rotate left,            logical array rol integer,  result same
ror   rotate right,           logical array ror integer,  result same

=     test for equality, result is boolean
/=    test for inequality, result is boolean
<     test for less than, result is boolean
<=    test for less than or equal, result is boolean
>     test for greater than, result is boolean
>=    test for greater than or equal, result is boolean

and   logical and,                 logical array or boolean,  result is same
or    logical or,                  logical array or boolean,  result is same
nand  logical complement of and,   logical array or boolean,  result is same
nor   logical complement of or,    logical array or boolean,  result is same
xor   logical exclusive or,        logical array or boolean,  result is same
xnor  logical complement of exclusive or,  logical array or boolean,  result is same
```

Overloading operators with functions

- Overloading operators are weirdly fun!
- You can assign different operations for existing operators
- For example you can create an operation that will work whenever "***" is used.
- Or you can assign a different operation of the "+" operator
- For example the pre-defined "+" (addition) operator accepts only INTEGER, SIGNED, or UNSIGNED values. However, we are interested in writing a function which should allow the sum of STD_LOGIC_VECTOR values as well (thus over- loading the "+" operator).

Example of overloading of "+"

```
1  -------- Package: ----------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  PACKAGE my_package IS
6     FUNCTION "+" (a, b: STD_LOGIC_VECTOR)
7        RETURN STD_LOGIC_VECTOR;
8  END my_package;
9  ---------------------------------------------
10 PACKAGE BODY my_package IS
11    FUNCTION "+" (a, b: STD_LOGIC_VECTOR)
12          RETURN STD_LOGIC_VECTOR IS
13       VARIABLE result: STD_LOGIC_VECTOR;
14       VARIABLE carry: STD_LOGIC;
15    BEGIN
16       carry := '0';
17       FOR i IN a'REVERSE_RANGE LOOP
18          result(i) := a(i) XOR b(i) XOR carry;
19          carry := (a(i) AND b(i)) OR (a(i) AND carry) OR
20                   (b(i) AND carry);
21       END LOOP;
22       RETURN result;
23    END "+";
24 END my_package;
25 ---------------------------------------------
```

```
1  --------- Main code: ------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_package.all;
5  ---------------------------------------------
6  ENTITY add_bit IS
7     PORT ( a: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
8            y: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
9  END add_bit;
10 ---------------------------------------------
11 ARCHITECTURE my_arch OF add_bit IS
12    CONSTANT b: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0011";
13    CONSTANT c: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0110";
14 BEGIN
15    y <= a + b + c;     -- overloaded "+" operator
16 END my_arch;
17 ---------------------------------------------
```