Subprograms consist of procedures and functions. A procedure can return more than one argument; a function always returns just one. In a function, all parameters are input parameters; a procedure can have input parameters, output parameters, and inout parameters.

Subprograms are very similar to a PROCESS, for they are only pieces of sequential VHDL code, and thus employ the same sequential statements (WAIT not allowed) this from a construction point of view, However, from the applications point of view, there is a fundamental difference between a PROCESS and a FUNCTION or PROCEDURE. While the first is intended for immediate use in the main code, the others are intended mainly for LIBRARY allocation, that is, their purpose is to store commonly used pieces of code, so they can be reused or shared by other projects. Nevertheless, if desired, a FUNCTION or PROCEDURE can also be installed in the main code itself.
Functions

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.

As mentioned above, a FUNCTION is very similar to a PROCESS. The same statements that can be used in a process 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 and a call to the function.

**FUNCTION body syntax**

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

**NOTES:**
1. Any number of parameters is allowed (even zero), which can only be CONSTANT (default) or SIGNAL (VARIABLES are not allowed).
2. Parameters' types can be any of synthesizable data types studied.
3. However, no range specification should be included (do not enter RANGE when using INTEGER, or TO/DOWNTTO when using STD_LOGIC_VECTOR)
4. There is only one return value.

**EXAMPLE**

The example below, named f1, receives three parameters (a, b, and c). a and b are CONSTANTS (default), while c is a SIGNAL. a and b are of type INTEGER, while c is of type STD_LOGIC_VECTOR (note either RANGE nor DOWNTTO was specified). The output parameter (only one) is of type BOOLEAN.
FUNCTION f1 (a, b: INTEGER; SIGNAL c: STD_LOGIC_VECTOR)
  RETURN BOOLEAN IS
BEGIN
  (sequential statements)
END f1;

Function Call
A Function is called as part of an expression. The expression can be obviously appear by itself or associated to a statement (either concurrent or sequential)

Examples
x <= conv_integer(a); --converts a to an integer.
  --(expression appears by itself)
y <= maximum (a, b); --return the largest of a and b.
  --(expression appears by itself)
If x > maximum (a, b) … --compares x to the largest of a and b
  --(expression associated to a
  --statement)

Example: Function conv_integer()
The FUNCTION presented converts a parameter of type STD_LOGIC_VECTOR into an INTEGER. Note that the code is generic, that is, it works for any range or order (TO/DOWNT) of the input STD_LOGIC_VECTOR parameter.

<table>
<thead>
<tr>
<th>Function body</th>
</tr>
</thead>
</table>
| 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; |

<table>
<thead>
<tr>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>Y &lt;= conv_integer(a);</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Exercise:
Write a FUNCTION to detects a positive (rising) clock edge. It is work must similar to the IF(clk'EVENT and clk = ‘1’) statement. Then built a DFF and test your function there. (note that you can declare your function in different places in the code as; in the declarative part of architecture a package or in package ,etc)

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

Procedures

A PROCEDURE is very similar to a FUNCTION and has the same basic purposes, However, a procedure can return more than one value. Like a FUNCTION, two parts are necessary to construct and use a PROCEDURE : the procedure itself and a procedure call.
Procedure body syntax

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

**NOTES:**
1. PROCEDURE can have any number of IN, OUT, or INOUT parameters, which can be SIGNALS, VARIABLES, or CONSTANTS.
2. For mode IN the default is CONSTANT, whereas for OUT or INOUT modes the default is VARIABLE.

**Example**
The PROCEDURE below has three inputs, a, b, and c (mode IN). a is a CONSTANT of type BIT, while b and c are SIGNALS, also of type BIT. There are also two return signals, x (mode OUT, type BIT_VECTOR) and y (mode INOUT, type INTEGER).

```
PROCEDURE my_procedure(a: IN BIT; SIGNAL b,c: IN BIT;
                        SIGNAL x: OUT BIT_VECTOR(7 DOWNTO 0);
                        SIGNAL y: INOUT INTEGER 0 TO 99) IS
 BEGIN
   ...
 END my_procedure;
```

**Procedure Call**
Contrary to a FUNCTION, which is called as part of an expression, a PROCEDURE call is a statement on its own. It can appear by itself or associated to a statement (concurrent or sequential)

**Examples**
Compute\_\_min\_\_max (in1, in2, in3, out1, out2);
   --statement by itself
Divide (dividend, divisor, quotient, remainder);
   --statement by itself
IF (a>b) THEN Compute\_\_min\_\_max (in1, in2, in3, out1, out2);
   --procedure call associated to another statement
Function and Procedure Location

This Figure summaries the Function and procedure location in the VHDL code

Exercise:

Design A limiter that take as an input (Minimum , Maximum and input) of integer type and one output . The output is the input limited by the two other inputs . design Two architectures for the system in one of them use procedure and in the second use a function .

1. Homework Questions

1. Write a brief comparison (table) between the function and the procedure .

2. Write function that convert bit_vector to std_logic_vector format or to unsigned (natural) value according to selector input S.

3. Write a procedure that takes an std_logic_vector and a selector as an input and returns the 1’st complement, 2’nd complement according to the selector and should always return an integer value of this vector.