Subprograms \{ 
  • FUNCTIONS \\
  • PROCEDURES \\
\} \\

pieces of SEQUENTIAL code

mainly intended for LIBRARY allocation
(store commonly use pieces of code)

vs. PROCESSES that are intended for immediate use in the MAIN code.

however it is possible to locate them in the main code.

FUNCTION and PROCEDURES have the same basic purposes
FUNCTION vs. PROCEDURE

1. FUNCTION
   - zero or more input parameters and a single return value
   - the input parameters can only be CONSTANTS (default) or SIGNALS
     (N.B. variables are not allowed)

2. PROCEDURE
   - can have any number of IN, OUT and INOUT parameters
     which can be SIGNALS, VARIABLES or CONSTANTS
   - the default for IN parameters is CONSTANTS
   - the default for OUT and INOUT parameters is VARIABLE

3. a FUNCTION is called as part of an EXPRESSION, while a PROCEDURE is a STATEMENT on its own.

4. both for FUNCTION and PROCEDURE: SIGNAL WAIT and COMPONENTS are not synthesizable

5. both for FUNCTION and PROCEDURE the possible locations are in PACKAGES (code reuse, code sharing, code partitioning) or in the main code (inside the ARCHITECTURE def. section or inside the ENTITY)

most common practice (both PACKAGE DEF. and PACKAGE body are necessary)
FUNCTIONS

Function Body

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

In the syntax above, <parameter list> specifies the function's input parameters, that is:

<parameter list> = [CONSTANT] constant_name: constant_type; or
<parameter list> = SIGNAL signal_name: signal_type;

Example

FUNCTION f1 (a, b: INTEGER; c: STD_LOGIC_VECTOR)
  RETURN BOOLEAN IS
BEGIN
  (sequential statements)
END f1;
Example of function calls

\[ x \leftarrow \text{conv integer}(a); \]
\[ y \leftarrow \text{maxmin}(a, b); \]
\[ \text{IF } x > \text{maxmin}(a, b) \text{ THEN } \ldots ; \]

**PROCEDURES**

Procedure Body

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

In the syntax above, 〈parameter list〉 specifies the procedure's input and output parameters; that is:

〈parameter list〉 = [CONSTANT] constant_name: mode type;

〈parameter list〉 = SIGNAL signal_name: mode type; or

〈parameter list〉 = VARIABLE variable_name: mode type;

Example

```
PROCEDURE my-procedure (a : IN BIT;
    SIGNAL b, c : IN BIT;
    SIGNAL x : OUT BIT_VECTOR (0 TO 7);
    SIGNAL y : OUT INTEGER RANGE 0 TO 99) IS
BEGIN
    ...
END my-procedure;
```

*not an issue with procedures*
Example: Function in the main code

```vhdl
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;
```
Example: FUNCTION in PACKAGE and OVERLOADING

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'LENGTH RANGE REV
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 -----------------------------
Example: PROCEDURE in main code

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY min_max IS
  GENERIC (limit : INTEGER := 255);
  PORT ( ena: IN BIT;
        in1, in2: 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
    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 (in1, in2, min_out, max_out);
        END IF;
      END PROCESS;
    END my_architecture;
assert

assert condition
[ report "message"]
[ severity severity-level];

Note, Warning, Error, Failure

( Default )