University of Bridgeport

INTRO TO VLSI DESIGN
CPE 448

VHDL
Tutorial-VIII

Subprograms

For: Fall 2002 By: Prakash S. Thapa

I. Subprograms

When we write complex behavioral models it is useful to divide the code into sections, each dealing with a relatively self-contained part of the behavior.

These self-contained parts can be called subprograms. There are two kinds of subprograms: ***procedures*** and ***functions***.

The difference between the two is that a procedure encapsulates a collection of sequential statements that are executed for their effect, whereas a function encapsulates a collection of statements that compute a result. Thus a procedure is a generalization of a statement, whereas a function is a generalization of an expression.

1. PROCEDURES:

There are two aspects to using procedures in a model:

First the procedure is declared,

Then elsewhere the procedure is ***called***. The syntax rule for procedure declaration is

```
procedure identifier [ (parameter_interface_list) ] is
        { subprogram_declarative_part }
begin
        { sequential_statement }
end procedure [ identifier ];
```

Example 1:

```
procedure average_samples is
        variable total: real := 0.0;
begin
        assert samples'length > 0 severity failure;
        for index in samples'range loop
                total := total + samples (index);
        end loop;
        average := total / real (samples'length);
end procedure average_samples;
```

Unlike variables in processes, procedure local variables are created new and initialized each time the procedure is called.

This procedure can be called just like:

average_samples;

Example 2:

```
procedure addu (    a, b        :    in word32;
                    result      :    out word32;
                    overflow    :    out Boolean ) is
    variable sum: word32;
    variable carry: bit := '0';
begin
    for index in sum'reverse_range loop
        sum(index):=a(index) xor b(index) xor carry;
        carry:=(a(index) and b(index) )
            or (carry and (a(index) xor b(index) ) );
    end loop;
    result:=sum;
    overflow:=carry = '1';
end procedure addu;
```

A call to this procedure may appear as follows:

```
variable PC, next_PC: word32;
variable overflow_flag: boolean;

addu ( PC, X"0000_0004",next_PC, overflow_flag);
```

2. FUNCTIONS:

The syntax rule for a function declaration is very similar to that for a procedure declaration:

```
function identifier [ (parameter_interface_list) ] return type_mark is
        { subprogram_declarative_part }
begin
        { sequential_statement }
end function [ identifier ];
```

Following example is a simple function that calculated whether a value is within given bounds and returns a result limited to those bounds. A call to this function might be included in a variable assignment statement, as follows:

```
new_temperature := limit ( current_temperature + increment, 10,100);
```

Example 1:

```
function limit ( value, min, max : integer) return integer is
begin
        if value > max then
                return max;
        elsif value < min then
                return min;
        else
                return value;
        end if;
end function limit;
```