VHDL Primer

Tutorial #4

Mike Goldsmith

Feb 24, 2004, ~ 1 hr duration

Outline

- Packages and Functions
- Assert & Report Statements

Packages and Functions

- Functions and procedures are just that: a chunk of code that performs a function or procedure.

- A different construct from an *entity* or *component*

- Procedures do not produce a result

- Can be declared within architecture scope and process scope, or as part of a package

Packages and Functions

procedure *identifier* [(*parameter_list*)] **is**

    {declarative section}

begin

    {sequential statements}

end procedure *identifier*;

- Declarative section can contain types, variables and nested procedure/ function descriptions – **not signals**

- Parameter list is described like **ports**: in, out or inout, and can be treated as constant, variable or signal (default is constant)

Packages and Functions

function *identifier* (*parameter_list*) **return** *type* **is**

   {declarative section}

begin

   {sequential statements}

   return *name*;

end function *identifier*;

- Declarative section same as procedure

- Parameter list **must all** be of direction **in** and **cannot** be of type variable (default is constant)

Packages and Functions

- Procedure and Function bodies contain all the funk that you can put in process blocks

```
function OR_REDUCE (d: in std_logic_vector) return std_logic is
    variable to_return: std_logic := '1';
begin
    for i in d'range loop
        to_return := to_return or d(i);
    end loop;
    return to_return;
end function OR_REDUCE;
```

Packages and Functions

procedure map_codes (**signal** op_word: **in** std_logic_vector(3 downto 0), **signal** op_code: **out** opcode) **is**

begin

    case (op_word) **is**

        when "0000" => op_code <= noop;

        when "0101" => op_code <= bneq;

        …

    end case;

end process map_codes;

Packages and Functions

- A Package is a library where you can group together types, functions and procedures, **components**, etc for reuse

package *identifier* **is**

   {package declarative region}

end package *identifier*;

package body *identifier* **is**

   {package body declarative region}

end package body *identifier*;

- Package and body identifiers must be the same

Packages and Functions

package reduction_functions **is**

    function AND_REDUCE (D:**in** std_logic_vector) **return** std_logic;

    function OR_REDUCE (D:**in** std_logic_vector) **return** std_logic;

    function XOR_REDUCE (D: **in** std_logic_vector) **return** std_logic;

end package reduction_functions;

package body reduction _functions **is**

    function OR_REDUCE (D:**in** std_logic_vector) **return** std_logic **is**

     end OR_REDUCE;

    …

end package body reduction_functions;

- May want to put package declaration and body in different files; only one body

Assert & Report Statements

- Used in simulation; an effective way to communicate with the "outside world"

- Multiple severity levels for breaking simulation

- Ignored in synthesis (mostly)

Code Sample:

assert *condition* [**report** *expression*][**severity** *severity_level*];

Assert & Report Statements

- Severity Levels

type severity_level **is** (note, warning, error, failure);

- Can trigger simulator to stop when a report of a particular severity level is emitted

- All reports are sent to the standard out console (cout), which often can be redirected

Assert & Report Statements

- Sample Code

```
D_ff : process( clk, reset_n, d )is
    variable d_tr: std_logic;
begin
    d_tr := d'delayed( FF_RISETIME );
    if reset_n = '0' then
        q <= '0';
    elsif rising_edge( clk ) then
        assert d_tr = d after 2*FF_RISETIME
        report "D-Flipflop risetime not within tolerance" severity error;
        q <= d;
    end if;
end process D_ff;
```