University of Bridgeport

INTRO TO VLSI DESIGN
CPE 448

VHDL
Tutorial-IX

Package

For: Fall 2002 By: Prakash S. Thapa

A VHDL package is simply a way of grouping a collection of related declarations that serve a common purpose. They might be a set of subprograms that provide operations on a particular type of data, or they might just be the set of declarations needed to model a particular design. The important thing is that they can be collected together into a separate design unit that can be worked on independently and reused in different parts of a model.

Another important aspect of packages is that they separate the external view of the items they declare from the implementation of those items. The external view is specified in a *package declaration*, whereas the implementation is defined in a separate *package body*.

Syntax follows:

```
package identifier is
        { package_declarative_item }
begin
        { sequential_statement }
end package [ identifier ];
```

Example 1:

```
package cpu_types is
        constant word_size : positive       := 16;
        constant address_size : positive    := 24;

        subtype word is bit_vector (word_size-1 downto 0);
        subtype address is bit_vector (address_size-1 downto 0);

        type status_value is (halted, idle, fetch, mem_read, read,
mem_write, io_read, io_write, int_ack);

        subtype opcode is bit_vector (5 downto 0);
        function extract_opcode (instr_word: word) return opcode;

        constant op_nop : opcode    := "000000";
        constant op_breq : opcode   := "000001";
        constant op_brne : opcode   := "000010";
        constant op_add : opcode    := "000011";

end package cpu_types;
```

Example 2:

```
entity address_decoder is
      port ( addr        : in   work.cpu_type.address;
             status      : in   work.cpu_type.status_value;
             mem_sel, int_sel, io_sel      :      out bit);
end entity address_decoder;

architecture behavioral of cpu  is

-- define constant mem_low: work.cpu_type.address:=X"000000";

begin
      interpreter : process is
            variable instr_reg      :      work.cpu_types.word;
            variable instr_opcode :      work.cpu_types.opcode;
      begin
            -- initialize
         loop
            -- fetch instruction
           instr_opcode:= work.cpu_types.extract_opcode( instr_reg):
           case instr_opcode is
                  when work.cpu_types.op_nop => null;
                  when work.cpu_types.op_breq => null;
            end case;
          end loop;
      end process interpreter;
end architecture behavioral;
```