
VHDL Basics

Module 2

Overview

- Packages
 - user created
 - Standard Package
 - IEEE
 - std_logic_1164
 - std_logic_unsigned
- Data Objects
- Data Types
- Operations

Libraries and Packages

- A package allows items to be shared, such as:
 - constants,
 - data types,
 - subprograms (procedures and functions)
- A package contains a package declaration and (optional) package body.

Packages (cont'd)

- Example (user created package):

```
PACKAGE vending_package IS
    SUBTYPE value IS integer RANGE 0 to 100;
    CONSTANT dime      : value := 10;
    FUNCTION add_dime (amount : integer) RETURN integer;
END vending_package;

PACKAGE BODY vending_package IS -- only required for subprograms
    FUNCTION add_dime (amount : integer) RETURN integer IS
    BEGIN
        RETURN (amount + dime);
    END add_dime;
END vending_package;

USE work.vending_package.ALL; -- USE clause allows access
ENTITY vending IS ...
```

Standard Package

- Used implicitly by all design entities
 - included in VHDL source files by implicit USE clause
 - defined in VHDL Language Reference Manual

```
-- This is not the complete package
PACKAGE standard IS
    TYPE boolean IS (false, true); -- return value of logical and
                                   -- relational operations
    TYPE bit IS ('0', '1');        -- used with logical operations
    TYPE character IS (nul, soh, stx, etx, eot, ...           -- 'q'
                      '0', '1', '2', ...
                      '@', 'A', 'B'....);
    TYPE integer IS RANGE -2147483648 TO 2147483647;        -- 32 bits
    TYPE severity_level IS (note, warning, error, failure);
    SUBTYPE natural IS integer RANGE 0 to integer'HIGH;    -- non-neg
    SUBTYPE positive IS integer RANGE 1 to integer'HIGH;  -- positive
    TYPE string IS ARRAY (positive RANGE <>) OF character; -- "hello"
    TYPE bit_vector IS ARRAY (natural RANGE <>) OF bit;    -- "010111"
END standard;
```

IEEE.std_logic_1164 Package

- Provides a signal with multiple values (9 value MVL)
 - `std_logic` and `std_logic_vector`
 - Only three useful for synthesis
 - '0' Forcing Low
 - '1' Forcing High
 - 'Z' High Impedance
 - 'X' Forcing Unknown (useful for simulation)
- To use the definitions and functions of `std_logic` package
 - Include before design entities in file
 - `LIBRARY ieee;`
 - `USE ieee.std_logic_1164.ALL;`

Data Objects

- Three classes of data objects in VHDL:
 - variables
 - signals
 - constants

- Variables
 - used to hold temporary data
 - only used in a process or subprogram (procedures and functions)
 - must declare the variable type (with optional range and initial value)

```
VARIABLE left_operand    : integer;  
VARIABLE temperature     : integer RANGE 0 TO 100 := 42;  
  
left_operand := 35 + ...
```

Data Objects cont'd

- Signals

- creates signals (wires, nets) for communication between processes and components. Examples:

```
SIGNAL zero_flag, overflow_flag    : std_logic;  
SIGNAL a_bus, b_bus : std_logic_vector (7 DOWNTO 0);  
  
a_bus <= b_bus AND "00001111";
```

- Constant

- an object initialized to a specific value - cannot be modified
- can improve readability of program - better documented model
- a model that is easier to update

```
CONSTANT add_opcode : std_logic_vector(3 DOWNTO 0) := "0101";  
CONSTANT cost_of_item : integer := 65; -- price of 65 cents  
CONSTANT max_value : integer := cost_of_item + 20;
```

Data Types

- All data objects must be defined with a data type
 - integers
 - real
 - enumeration type
 - arrays
 - records
- Note: VHDL is a strongly typed language - can not mix types.
 - type conversion is allowed only between *closely related* types
 - operations restricted to allowed types
 - arithmetic operations not supported for bit_vectors only integers
 - use additional libraries (std_logic_unsigned) with overloading

Integer Types

- Integer Types

- Integer (defined in Standard Package) is defined as

```
TYPE integer IS RANGE -2147483648 TO 2147483647
```

- Natural (non-negative) is defined as

```
SUBTYPE natural IS integer RANGE 0 TO integer'HIGH
```

- Positive is defined as

```
SUBTYPE positive IS integer RANGE 1 TO integer'HIGH;
```

- Example:

```
PROCESS
    VARIABLE num1, num2, num3 : integer;
    VARIABLE alu_result : integer RANGE 0 TO 255;
BEGIN
    num1 := 123;
    num2 := -15;
    num3 := 1_000_000;    -- underscore adds readability
    alu_result := 16#5D#; -- 5D in hexadecimal (base 16)
```

Real Types (NOT for synthesis)

- Real Types
 - Floating point (defined in Standard Package) is defined as
 - `TYPE real IS RANGE -1.0E38 to 1.0E38`

Enumeration Types

- Enumeration Types
 - defined by listing all possible values for that type
 - each identifier has a position (order is 0, 1, etc.)
 - synthesis (bits and flip-flops required to encode all values)
 - one hot
 - sequential
 - other
 - very useful (required) for specifying states for state machines

Enumeration Types (cont'd)

- **examples:**

- TYPE weekday IS (sun, mon, tues, wed, thurs, fri, sat);
- TYPE states IS (s0, s1, s2, s3, s4, s5, s6, s7);

```
ARCHITECTURE behav OF controller IS
    TYPE state_type IS (start, reset, fault);
    SIGNAL state : state_type;    -- size of signal is 2 bits
BEGIN
    PROCESS(clk)
    BEGIN
        IF clk'EVENT AND clk = '1' THEN    -- rising edge of clk
            CASE state IS
                WHEN start =>
                    IF ...
                        state <= fault;
            END CASE;
        END IF;
    END PROCESS;
END ARCHITECTURE;
```

Array Types

- Array Types
 - groups one or more elements of the same type together

```
TYPE address_bus IS ARRAY (63 DOWNT0 0) OF std_logic; -- one dimension
TYPE memory IS ARRAY(0 to memsize, memwidth DOWNT0 0) OF std_logic;

SIGNAL cpu_address      : address_bus;      -- declare arrays
SIGNAL z                : std_logic;

-- also declaring a constant array
CONSTANT eprom : memory :=
    ( ('0', '0', '0', '0'),
      ('0', '1', ...
      ('1', '1', '1', '1') );

-- now using these arrays in statements
cpu_address(63 DOWNT0 56) <= "01010111"; -- assign to slice of array
z <= eprom(6,1); -- element at 2nd column of 7th row.
```

Aliases

- Aliases
 - An alias creates a new name for all or part of the range of an array type.
 - Useful for naming parts of a range as if they were subfields.
 - Aliases provide a mechanism to name each of the subfields and to reference these fields directly by the alias names.

```
SIGNAL instruction : std_logic_vector(22 DOWNTO 0);

ALIAS opcode : std_logic_vector(6 DOWNTO 0) IS instruction (22 DOWNTO 16);
ALIAS src     : std_logic_vector(7 DOWNTO 0) IS instruction (15 DOWNTO 8);
ALIAS dest    : std-logic_vector(7 DOWNTO 0) IS instruction (7 DOWNTO 0);

CASE opcode IS
    WHEN add =>
```

Record Types

- groups objects of many types together as a single object.
- each element of the record can be accessed by its field name.

```
TYPE operation IS (add, sub, mul, div, ror, rol);
TYPE instruction IS
    RECORD
        opcode    : operation;
        src       : integer;
        dest      : integer;
    END RECORD;

-- example statements
PROCESS(X)
    VARIABLE inst           : instruction;
    VARIABLE source, dest   : integer;
BEGIN
    inst := (add, destination, 3);
    source := inst.src;
    destination := inst.dest;

    IF inst.opcode = add THEN
```

Operators

- Logical (lowest precedence)
AND, OR, NAND, NOR, XOR, XNOR
- Relational
=, /=, <, <=, >, >=
- Shift
SLL, SRL, SLA, SRA, ROL, ROR
- Adding (including concatenation)
+, -, &
- Multiplying
*, /, MOD (modulus), REM (remainder)
- Misc (highest precedence)
ABS (absolute), ** (exponentiation), NOT

Operators (cont'd)

- Logical

- may be used with predefined (single element and one-dimensional array) bit, std_logic, and Boolean types

- `d <= a AND c; -- assume a, b, c, d, e, are type std_logic`
- `d <= b NOR c;`
- `d <= a NAND b NAND c; -- illegal NAND/NOR sequence`
`-- not associative`
- `d <= (a NAND b) NAND c; -- use parentheses`
`NOTE: not three-input NAND gate`
- `d <= a OR (b AND c);`
- `e <= (b AND d) XOR (a NAND c);`

- Relational

- used for comparison operations.
- two operands must be same type - result is a Boolean value
 - `c <= a = b; -- assume a and b are type integer`

Operators (cont'd)

- Adding Operators
 - +, -, * for integer operands (NOT for bits or bit_vectors)
 - also supported for SIGNED and UNSIGNED data types in signed and unsigned packages
 - & (concatenation) for single elements or one-dimensional array

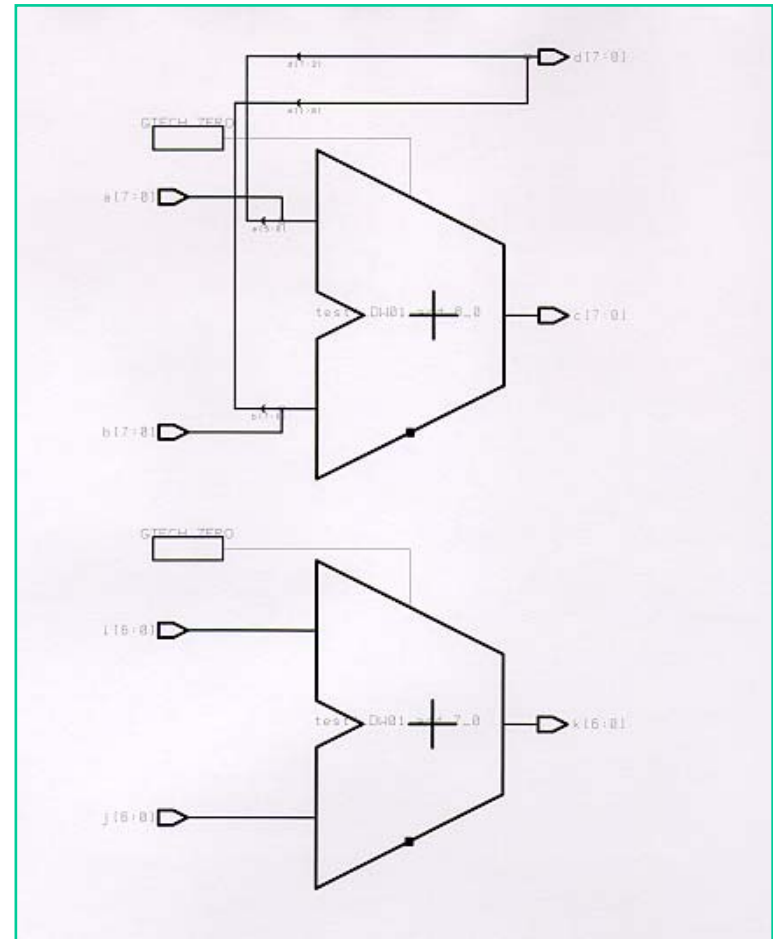
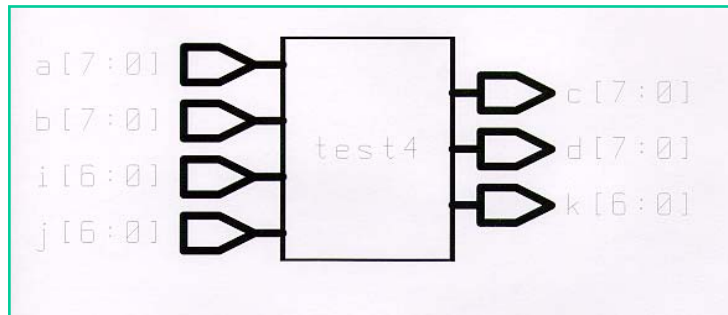
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL; -- for arithmetic operations

ENTITY test4 IS
    PORT(i, j : IN integer RANGE 0 TO 99;
         a, b   : IN std_logic_vector(7 DOWNTO 0);
         k      : OUT integer RANGE 0 TO 99;
         c, d   : OUT std_logic_vector (7 DOWNTO 0));
END test4;
```

Operators (cont'd)

```
ARCHITECTURE arch OF test4 IS
BEGIN
  k <= i + j;           -- ok for integers
  c <= a + b;           -- ok for std_logic if unsigned included
  d <= a(5 DOWNTO 0) & b(7 DOWNTO 6);
END arch;
```

Synthesis Results



Synthesis Results

