Lab # 3

Scalar Data Types and Operations
Composite Data Types and Operations

October, 2013
**Introduction**

- The concept of type is important when describing data in VHDL model.
- The type of a data object defines the set of values that the object can assume, as well as the set of operation that can be performed on those values.
- VHDL along with its packages provides pre-defined types.
- Additionally the user can define new types.

---

**Scalar Data Types and Operations**

**Integer Types:**
- “integer” is a pre-defined type used to represent whole numbers from $-2^{31}$ to $2^{31}-1$
- e.g.

  ```
  variable x, y : integer ;
  ```

- User can define new “integer” types using a range_constrained type definition,
  e.g.

  ```
  type month is range 1 to 12 ;
  type count_down is range 10 downto 0;
  ```

- Default value is left hand side of range.
- Operations on Integer Types:
  - Addition: +
  - Subtraction or negation: -
Multiplication: *
Division: /
Modulo: mod
Remainder: rem
Absolute value: abs
Exponentiation: **
Logical: =, /=, <, >, <=, >=

Floating-point Types:
• They are used to represent real numbers. It is not possible to represent real numbers
  exactly on computers since there is infinite number of real numbers in an interval;
  thus Floating-point Types are only approximation of real numbers.
• "Floating point" is represented using a mantissa and an exponent part.
• User can define new “Floating-point” types using a range constrained type definition,
  e.g.

  type probability is range 0.0 to 1.0;

• Operations on Floating-point Types are: +, -, *, /, abs, **, =, /=, <, >, <=, >=.

Physical Types:
• They represent real world physical quantities, such as length, mass, time and current.
• A physical type includes the primary unit, which is the smallest represented unit, and
  may include some secondary units, which are integral multiples of the primary unit,
  e.g.

  type resistance is range 0 to 1E9
  units
  ohm; -- primary unit
  kohm = 1000 ohm; -- secondary unit
  end units resistance;

• Time is a pre-defined physical type.
• Operations on physical types are: +, -, *, /, abs, **, =, /=, <, >, <=, >=.

Enumeration Types:
• Useful for giving names to values of an object (variable or signal),
  e.g.

  type alu_func is (disable, pass, add, sub, mult, div);

• Predefined Enumeration types are: Character, Boolean, Bit, and Standard Logic.
• Characters Type:

  type character is ( nul, soh,...,’a’, ‘b’, ‘c’, ..........);

  • Operations on Characters Type : =, /=, <, >, <=, >=
• **Boolean Type:**
  ```
  type boolean is (false, true);
  ```
  - Operations on Boolean Type: and, or, nand, nor, xor, xnor, not, =, /=, <, >, <=, >=

• **Bit Type:**
  ```
  type bit is ('0', '1');
  ```
  - Operations on Bit Type:
    - Logical: =, /=, <, >, <=, >=
    - Boolean: and, or, nand, nor, xor, xnor, not
    - Shift: sll, srl, sla, sra, rol, ror

• **Standard Logic Type:**
  ```
  type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
  ```

> **Composite Data Types and Operations**

**Arrays:**
- An array consists of a collection of values of the same type; each of these values has an index indicating its position in the array.
- **type word1 is array (0 to 31) of bit;** this array stores values of bit type indexed by the integers from 0 to 31
- Indexes may be of any scalar data type,

  ```
  type state is (initial, idle, active, error);
  type state_counts1 is array (state) of natural;
  ```

**Multi-dimensional arrays:**
A Multi-dimensional array type is declared by specifying a list of index ranges each of them can be specified for single-dimensional array,

```
  type symbol is ('a', 't', 'd', 'h');
  type state is range 0 to 6;
  type trans_matrix is array(state, symbol) of state;
```
Array aggregate:
- It is used for initializing a variable or defining a constant,
e.g.

```vhdl
type point is array (1 to 3) of real;
constant origin : point := (0.0,0.0,0.0);
variable view_point : point := (10.0,10.0,0.0);
```

Array attributes:
- A’left(N) : Left bound of index range of dim. N of A
- A’right(N) : Right bound of index range of dim. N of A
- A’low(N) : Lower bound of index range of dim. N of A
- A’high(N) : Upper bound of index range of dim. N of A
- A’range(N) : Index range of dim. N of A
- A’reverse_range(N) : Reverse index range of dim. N of A
- A’length(N) : Length of index range of dim. N of A
- A’ascending(N) : true if index range of dim. N of A is ascending, false otherwise.

Unconstrained array types:
- In these types only the type of the index values is declared without specifying bounds,
e.g.

```vhdl
type sample is array (natural range <>) of integer;
```

- Constrains can be added when creating an object,
e.g.

```vhdl
variable short_sample: sample (0 to 63);
```

- “string” is pre-defined unconstrained arrays of type character.
  ```vhdl
type string is array (positive range<>) of character;
  ```

- “bit_vector” are pre-defined unconstrained arrays of type bit.
  ```vhdl
type bit_vector is array (natural range <>) of bit;
  ```

Records:
- A record is a composite value comprising elements of different types,
e.g.

```vhdl
type time_stamp is record
seconds: integer range 0 to 59;
minutes : integer range 0 to 59;
hours : integer range 0 to 23;
end record;
```

- Each value can be selected separately using the (.) operator.
Examples

Example 1

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package int_types is
  type small_int is range 0 to 255;
end package int_types;

use int_types.all;

entity Small_Adder is
  port(
    a, b : in small_int;
    sum : out small_int
  );
end Small_Adder;

architecture behav of Small_Adder is
begin
  sum <= a + b;
end behav;
```

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Value</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00000002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>00000007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>00000009</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Signal name | Value  |   |   |
-------------|--------|---|---|
• Example 2

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Array_Sum is
end Array_Sum;

architecture behav of Array_Sum is
  type sample_array is array(0 to 4) of integer;
begin
  variable sum : integer := 0;
  variable sample : sample_array := (10,20,30,40,50);
  begin
    process is
      for index in 0 to 4 loop
        sum := sum + sample(index);
      end loop;
      wait;
    end process;
  end behav;
```

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>150</td>
</tr>
<tr>
<td>sample</td>
<td>30</td>
</tr>
<tr>
<td>sample[0]</td>
<td>10</td>
</tr>
<tr>
<td>sample[1]</td>
<td>20</td>
</tr>
<tr>
<td>sample[2]</td>
<td>30</td>
</tr>
<tr>
<td>sample[3]</td>
<td>40</td>
</tr>
<tr>
<td>sample[4]</td>
<td>50</td>
</tr>
</tbody>
</table>
• Example 3:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Matrix_Int is
end Matrix_Int;

architecture behav of Matrix_Int is
type symbol is ('a', 't', 'd', 'h', digit, cr, other);
type state is range 0 to 6;
type matrix is array(state, symbol) of state;
begin
  process is
    variable m1 : matrix :=
      (0 => ('a' => 1, others => 6),
       1 => ('t' => 2, others => 6),
       2 => ('d' => 3, 'h' => 5, others => 6),
       3 => (digit => 4, others => 6),
       4 => (digit => 4, cr => 0, others => 6),
       5 => (cr => 0, others => 6),
       6 => (cr => 0, others => 6));
    variable m2 : matrix;
  begin
    m2 := m1;
    wait;
  end process;
end behav;
```

➤ Lab. Exercises

• L. Exercise:

1. Write a VHDL code to get the attributes of the second dimension of the array m1 used in example 3.
2. Define a record type declaration that includes a day, month and year values. Declare a variable of this record type and get its elements separately.

• H. Exercise:

1. write a model for a tri-state buffer using the standard-logic type for its data and enable inputs and its data output achieving the following behavior:

<table>
<thead>
<tr>
<th>Enable</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or L</td>
<td></td>
<td>Z</td>
</tr>
<tr>
<td>1 or H</td>
<td>0 or L</td>
<td>0</td>
</tr>
<tr>
<td>1 or H</td>
<td>1 or H</td>
<td>1</td>
</tr>
<tr>
<td>Other cases</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
2. develop a model for a priority encoder with a 8-element bit-vector input port, an output port of type *natural* that encodes the index of the leftmost '1' value in the input and an output of type bit that indicates whether any input elements are '1'.

3. develop a model for a general and-or-invert gate, with two standard-logic vector input ports $x$ and $y$ and a standard-logic output port $z$. The output of the gate is

$$x_0.y_0 + x_1.y_1 + ..... + x_{n-1}.y_{n-1}$$