

## ECE-238 Logic Design – Lab Syntax (Data type, CASE, IF THEN ELSE)

We have seen that the Data in VHDL are classified in Scalar and Compounds. The Scalars are the ones which can take only one value; the compound can take several values.

1. The Scalar ones are classified in: *Integers, Reals, Enumerated and Physicals.*
  - The Integers and Reals are used to represent numbers that are integers or reals, respectively. The integers are defined in the range  $-2^{31}-1$  to  $2^{31}-1$ . The reals are defined in the interval  $-1.0E38$  to  $1.0E38$ .

The syntax for these two types is:

**object identifier** : **type range** [values];

A **range** is a reserved word by VHDL and it is used to define a set of values.

- The Enumerated are use to list all the values that an object can take.

The syntax for this type is:

**type identifier is** definition;

As an example we have: **type names is** (Bill, Mike, Peter)

Actually, the data types bit and boolean are special case of enumerated:

**type boolean is** (true, false)

**type bit is** (0,1)

- The Physical types are values that are used as measurement units. In VHDL the only predefined physical type that exists is **time**, and the fundamental unit is femtosecond (fs).

Example: **We can define a physical type data as follows:**

**type** time **is range** 0 to 1E20

**units**

fs;

ps = 1000 fs;

ns = 1000 ps;

us = 1000 ns;

ms = 1000 us;

s = 1000 ms;

min = 60 sec;

h = 60 min;

**end units;**

If required you can defined other physical units as meters, grams, etc.

2. The Compounds ones are classified as arrays and records.

- The array type is form by multiple elements with one type in common. These arrays can also be considered as vectors since they group elements of the same type.

The syntax used to define an array is:

**type** identifier **is array** (range) of object\_type:

The standard IEEE 1076 and 1164 defined two important arrays: the **bit\_vector** and the **std\_logic\_vector**, these are part of the bit and std\_logic type, respectively.

The following are declarations of these types:

**type** bit\_vector **is array** (range) of bit;

**type** std\_logic\_vector **is array** (range) of std\_logic;

Examples of these declarations:

**type** digits **is array** (9 **downto** 0) **of** integer;

**type** byte **is array** (7 **downto** 0) **of** bit;

**type** address **is array** (10 to 62) **of** bit;

Another example is that we can create a truth table:

**type** table **is array** (0 to 3, 0 to 2) **of** bit;

**constant** example: table := (

    "00\_0",

    "01\_0",

    "10\_0",

    "11\_1",

This is a two dimensional array, two bits for the inputs and one for the output. The underscore symbol are to distinguish between inputs and outputs.

- The Record type is a data type where the elements are of different types, which received the name of fields. Each of the fields must contain a name for identification purposes.

A declaration of this type has the following syntax:

**type** identifier **is record**

Identifier : type;

**end record**;

---

Finally, we want to recall the syntax of the IF..THEN... ELSE and CASE –WHEN logical functions.

### **IF – THEN -- ELSE Function**

This declaration is used to select a condition or conditions based in the result of several logic evaluations (true or false). The syntax is the following:

**If** condition is true **then**

    do the operation 1;

**else**

    do the operation 2;

**end if**;

This means that if (**if**) condition is true, then (**then**) the instruction indicates that operation 1 will be executed. Similarly, if the condition is false, the operation 2 will be executed.

A similar scheme could include the execution of more operations. For this cases, the function **IF-THEN - ELSE IF - THEN – ELSE** will be more useful.

The syntax of this operation is:

```
If condition is true then  
    do the operation 1;  
elseif condition 2 is true then  
    do the operation 2;  
else  
    do the operation 3;  
end if;
```

An example of this logical function is the following: suppose we have to compare two input numbers “a” and “b” (these could be 4 bit numbers, for instance) and activate output x if a = b, activate output y if a < b and activate output z . Then the comparator will have the following format:

```
library ieee;  
use ieee.std_logic_1164.all;  
entity comp is port(  
a,b: in std_logic_vector(3 downto 0);  
x,y,z : out std_logic);  
end comp;
```

```
architecture arq_comp of comp is  
begin  
    process (a,b)  
    begin  
        if (a = b) then  
            x <= '1';  
        elseif (a < b) then  
            y <= '1';  
        else  
            z <= '1';  
        end if;  
    end process;  
end arq_comp;
```

---

## CASE -- WHEN

In the instruction CASE – WHEN we can allow that a signal adopts a specific value if a condition is satisfied.

Recall the multiplexer example of lab 3:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4x1 is port(
D0,D1,D2,D3: in std_logic;
Y : out std_logic);
end mux4x1;

architecture multiplexor4x1 of mux4x1 is
begin
  process(S, D0, D1, D2, D3) begin
    case S is
      when "00" => Y <= D0;
      when "01" => Y <= D1;
      when "10" => Y <= D2;
      when others => Y <= D3;
    end case;
  end process;
end multiplexor4x1;
```