Scalar Data Types

Integer, Real, Bit, and Std.Logic.

Composite Data Type (Arrays)

1-Dimensional Array (1D) - Vectors

-- not preferred
type vector is array (7 downto 0) of std_logic;
signal x : vector;

--better
signal y : std_logic_vector (7 downto 0);
Array of Arrays (1D*1D)

type matrix is array (3 downto 0) of std_logic_vector (7 downto 0);
signal x : matrix;

to initialize it :

type matrix is array (3 downto 0) of std_logic_vector (7 downto 0);
signal t : matrix := ("00000000", "11111111", others=>'0', others=>'1');

2-Dimentional Array (2D)

type matrix2d is array (3 downto 0,7 downto 0) of std_logic;
signal m2d : matrix:= ("00000000", "11111111", others=>'0', others=>'1');

Sequential statements

Sequential VHDL statements allow you to describe the operation, or behavior, of your circuit as a sequence of related events. The use of sequential statements to describe combinational logic implies that our use of the term sequential in VHDL is somewhat different from the term as it is often used to describe digital logic. Specifically, sequential statements written in VHDL do not necessarily represent sequential digital logic circuits. As we will see, it is possible, and quite common, to write sequential VHDL statements, using processes and subprograms, to describe what is essentially combinational logic.

In this lab, we will look at examples using theses sequential statements. We will also examine the various types of sequential statements available in VHDL. Our primary focus will be on those styles of sequential VHDL that are most appropriate for synthesizable design descriptions and for test benches.

Sequential statements are found within processes, functions, and procedures. Sequential statements differ from concurrent statements in that they have order dependency. This order dependency may or may not imply a sequential circuit, one involving memory elements.
1. Sequential statements types

- If statement
- Case statement
- Loop statement
- Wait statement
- Return statement
- Assertion and report statements

A. If statement: Conditionally Execute Statements

Syntax:

```plaintext
if condition then
    sequential statements...
[elsif condition then
    sequential statements...]
[else
    sequential statements...]
end if;
```

B. Case statement

Conditionally Select One-of-Many Statements to Execute Based on value of selector expression. There can only be one, unique match. Several choices may be lumped together

E.g. when ch1|ch2|ch3 => action to be performed for all 3 alternate choices
E.g. when A to B => action to be performed for a range of choices

Syntax:

Case expression is
when choice => statements...
...
[others statements...;]
end case;
C. Loop statement: Executes a group of statements repeatedly

Syntax:

[label:] iteration scheme loop
{sequential statements}
end loop [loop_label];

Where
1) iteration scheme may be:
   while condition
   for identifier in range

   Identifier is Self-defining and visible Only in the Loop
Range Specifications examples
   i in 0 to 7
   letter in ( 'A', 'B', 'C')
   j in 7 downto 0

2) The Following Statements may alter the Loop flow:
   next - Skip Execution of Following Statements and Iterate Loop
   exit - Terminate Execution of the Loop

D. Wait statement: Halts Process Execution

Syntax:
    wait on signal name;
    wait until Boolean expression;
    wait for time expression;
    wait;

we used the wait for expression in the previous lab. And used Run for command too. If you noticed, using Run command will cause an infinite loop so to stop this add wait; statement at the end of the process;
2. Example: Enc-7Seg

Circuit diagram

- Encoder using if statement

The Entity for the encoder

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Encoder is
  port (enc_in : in  std_logic_vector(7 downto 0);
       enc_out: out std_logic_vector(2 downto 0));
end Encoder;

Behavioral description of encoder using if statement

architecture Encoder_beh of Encoder is
begin
  process (enc_in) begin
    if enc_in(7) = '1' then
      enc_out <= "111";
    elsif enc_in(6) = '1' then
      enc_out <= "110";
    elsif enc_in(5) = '1' then
      enc_out <= "101";
    elsif enc_in(4) = '1' then
      enc_out <= "100";
```
elsif enc_in(3) = '1' then
  enc_out <= "011";
elsif enc_in(2) = '1' then
  enc_out <= "010";
elsif enc_in(1) = '1' then
  enc_out <= "001";
elsif enc_in(0) = '1' then
  enc_out <= "000";
else
  enc_out <= "000";
end if;
end process;
end EncoderBeh;

7-Segment Register Using case statement, Null statement

The Entity for the 7-Segment Register

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sevenSegmentReg is
  port(BCD : in std_logic_vector (2 downto 0);
       A,B,C,D,E,F,G : out std_logic);
end sevenSegmentReg;

Behavioral description of 7-segment using Case statement

architecture sevenSegmentReg_beh of sevenSegmentReg is
begin
  bcd2sevSeg: process (BCD)
  begin
    -- Assign default to "off"
    A<='1';B<='1';C<='1';D<='1';E<='1';F<='1';G<='1';

    case BCD is
      when "000"=>
        A<='0';B<='0';C<='0';D<='0';E<='0';F<='0';G<='0';
  end case;
end process;
end sevenSegmentRegBeh;
when "001"=>
B='0';C='0';
when "010"=>
A='0';B='0';D='0';E='0';G='0';
when "011"=>
A='0';B='0';C='0';D='0';G='0';
when "100"=>
B='0';C='0';F='0';G='0';
when "101"=>
A='0';C='0';D='0';F='0';G='0';
when "110"=>
A='0';C='0';D='0';E='0';F='0';G='0';
when "111"=>
A='0';B='0';C='0';
when others=>
null;
end case;
end process bcd2sevSeg;
end sevenSegmentReg_beh;

3. **Lab Exercise**

Build a component (Enc_7Seg) from the encoder and the 7-seg then write a testbench code to test this circuit.

4. **Homework Exercises**

1. Write an entity declaration & a behavioral architecture body for a 8-bit multiplexer.

2. Consider a Binary decoder having the following signals

   - ain, bin - select control inputs
   - en - enable input
   - b0, b1, b2, b3 – outputs.

   The inputs ain and bin control which output is asserted when input en is high. If en is low, the outputs are always low. The 'd' symbol means 'don't-care'.
Write an entity declaration & a behavioral architecture body for a Binary decoder, then Write the test bench for the Binary decoder model, and test it using VHDL simulator.

<table>
<thead>
<tr>
<th>en</th>
<th>ain</th>
<th>bin</th>
<th>b3</th>
<th>b2</th>
<th>b1</th>
<th>b0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d</td>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>