Behavioural Description Problem Statement

Prof. Sachin Patkar, Jay Adhaduk & Rahul C P

6th March 2019

Problem 1 : Concurrent Statements and Processes

Part A

- The following is the code segment discussed in the class. As you can see there will be glitch in the output of C signal because of delay (delta). Simulate the following code and find out the glitch time using wave form and compare with handwritten calculations.

```
library ieee ;
use ieee.std_logic_1164.all ;

entity example_1 is
  port ( c : out std_logic ) ;
end entity ;

architecture behav_1 of example_1 is
  signal a , b : std_logic := '0' ;
begin
  a <= not a after 5 ns ;
  b <= not a after 1 ns ;
  c <= a or b after 1 ns ;
end behav_1 ;
```

Part B

- Now we will see how we can write concurrent statement using the process statement. In the above example one can see drivers are *a* , *a* and (*a* or *b*) respectively. So we need to put that inside sensitivity list of the process.
- Compile and simulate the following code and compare with part A.

```
architecture behav_2 of example_1 is
  signal a , b : std_logic := '0' ;
begin
  a_proc: process ( a ) begin
             a <= not a after 5 ns ;
          end process ;
  b_proc: process ( a ) begin
            b <= not a after 1 ns ;
          end process ;
  c_proc: process (a,b) begin
            c <= a or b after 1 ns ;
           end process ;
end behav_2 ;
```

Part C

- If you don't specify anything inside the sensitivity list then all the signals inside the process will be considered as sensitivity list.
- You can also write Part B as following. Here after value of *not(a)* is assign to *a*, *a_proc* will wait for the event in signal *a*.
- Compile and simulate the following code and compare with part A and part B.

```
architecture behav_3 of example_1 is
  signal a , b : std_logic := '0' ;
begin
  a_proc: process  begin
              a <= not a after 5 ns ;
                      wait on a ;
               end process ;
  b_proc: process  begin
              b <= not a after 1 ns ;
                      wait on a ;
               end process ;
  c_proc : process begin
              c <= a or b after 1 ns ;
                      wait on a, b ;
  end process ;
end behav_3 ;
```

Problem 2 : for...generate and for...loop

Part A

- When there is a repetition of the same module in the design one can use for...generate structure to do in a simple way.
- For an example to make an 8-bit adder one needs 8 full adder modules. Instead of writing 8 time instances of the full adder for...generate statement will make designing easier.
- The following is the example of 8 bit adder using the instance of full adder 8 times using the for...generate.
- Compile the design and check the RTL viewer for generated net-list of the design.

```
      architecture Behav of adder_8bit is
       component fa is
           Port ( a : in  STD_LOGIC;
                  b : in  STD_LOGIC;
                  cin : in  STD_LOGIC;
                  sum : out  STD_LOGIC;
                  cout : out  STD_LOGIC);
       end component;

       signal co : std_logic_vector(8 downto 0) ;

       begin
       co(0) <= cin;
       gen_for:
           for i in 0 to 7 generate
                  faX : fa port map(a(i), b(i), co(i), sum(i), co(i+1));
               end generate gen_for;
end Behavioral;
```

Part B

- In this part we will use the design of full adder instead of using component instance of the full adder inside the for...generate loop.
- Compile the design and check the RTL viewer for generated net-list of the design and compare with the previous case.

```
architecture Behav_2 of adder_8bit is
     signal C : std_logic_vector(8 downto 0) ;
     begin
           C(0) <= Cin;
      genAdd: for I in 0 to 7 generate
            C(I+1) <= ((A(I) or B(I)) and C(I)) or(A(I) and B(I));
            SUM(I) <= A(I) xor B(I) xor C(I);
      end generate genAdd;
  end Behav_2
```

Part C

- In this part we will use process statement to create the 8 bit adder. Here we will use for...loop and variable.
- Compile the design and check the RTL viewer for generated net-list of the design and compare with the previous case.

```
architecture behave of add8 is
  begin
  process(A,B,Cin)
    variable Cvar : std_logic;
         begin
          -- note this assignment.. it takes effect
          -- immediately.
          Cvar := Cin;
          for I in 0 to 7 loop
             -- this assignment takes effect only
             -- after a delta-delay.
             SUM(I) <= A(I) xor B(I) xor Cvar;
             -- immediate.
             Cvar := ((A(I) or B(I)) and Cvar) or
             (A(I) and B(I));
          end loop;
  end process;
end behave;
```

Problem 3 : Signals vs Variables

Part A

- When we are using signal inside process then it will take delta time for the assignment. In the following example when change in clock signal happens, it will go inside the process and value of a will be assigned to ***b*** after delta delay but *c* and *d* will not change.
- So *d* signal will have value of *a* after 3 clock cycle.
- Simulate the following code and check for the generated net-list and find the when a signal value is getting assigned to d?

```
library ieee ;
use ieee.std_logic_1164.all ;
```

```
entity example_2 is
        port ( clock ,a : in std_logic ;  d : out std_logic ) ;
end example_2 ;

architecture behav_1 of example_2 is
  signal b,c : std_logic := '0' ;
begin
  process ( clock )
    begin
        if(CLOCK='1' and CLOCK'EVENT) then

                 b <= a ;
                 c <= b ;
                 d <= c ;
               end if;
  end process ;

end behav_1 ;
```

Part B

- When we are using variables inside process then it be directly assigned without any delay. In the following example when a will change it will go inside the process and value of a will be assigned to b without any delay.
- So d signal will have value of a after 1 clock cycle.
- Simulate the following code and check for the generated net-list and find the when a signal value is getting assigned to d?

```
architecture behav_2 of example_2 is
  begin

  process (clock)
          variable b_v , c_v : std_logic ;
    begin
      if(CLOCK='1' and CLOCK'EVENT) then
        b_v := a ;--after 1 ns ;
                c_v := b_v ;--after 1 ns;
                d <= c_v ;--after 1 ns ;
          end if;
  end process ;
end behav_2 ;
```