Syllabus

Unit - 3

P.M.R

→ process statement → variable assignment → signal assignment statement
→ wait statement → if statement → case statement → null statement
→ loop statement → exit statement → next statement → Assertion statement
→ more on signal assignment → inertial delay model → transport delay model → creating signal waveforms → signal drivers → other sequential elements → multiple processes → logic synthesis → inside a logic synthesizer

---

Statements in VHDL

Sequential statements

→ while using these statements, the ordering of the statements is important because ordering may effect the meaning of the code.

features of sequential elements

→ while using these statements, the ordering of the statements is important.
→ the sequential statements execute one after another as per the writing order.
→ they must be placed inside a "process statement"
→ variables are only used in sequential elements.
→ sequential elements do not generate sequential hardware

www.Jntufastupdates.com

Scanned by CamScanner

Process Statement

→ process is main Concurrent Statement in VHDL code which describes the sequential behaviour of design.

→ All Statements with in the process executed sequentially in zero time.

→ only one driver is placed on signal

→ the signal is updated with the last value assigned to it with in the process.

Syntax:

```
PROCESS ( sensitivity list)
-- Declarative part
BEGIN
  sequential Statements
END PROCESS.
```

Ex:

```
PROCESS (en, a, b, c, d)
BEGIN
if en = '1' THEN
z <= a AND b;
z <= c AND d;
END IF
END PROCESS.
```

→ A process statement is concurrent that can appear with in architecture body.

→ A process statement also has a declarative part (before keyword BEGIN)

→ Statement part (between the keywords BEGIN and END PROCESS)

→ the Statements appearing with in the Statement part are sequential Statements and executed sequentially.

Sensitivity list

→ List of signals on which process should executed after changing its State.

→ Every process must have either sensitivity list

www.Jntufastupdates.com
Scanned by CamScanner

2 ess types

Combinational process.

→ Generates a combinational logic
→ All i/ps must be present in sensitivity list.

clocked process

→ Generate a register logic.
→ Input under clock transition may not included in sensitivity list.

IF statement

→ Any IF statement Selects a sequence of elements for execution based on the value of condition.
→ the condition can be any expression that evaluated boolean value.

syntax:

```
IF condition1 THEN
    sequential statement
ELSIF condition2 THEN
    sequential statement
ELSE
    sequential statement
END IF.
```

ex:

```
PROCESS (sel, a, b, c, d)
BEGIN
IF sel = '00' THEN
    z <= a;
ELSIF sel = '01' THEN
    z <= b;
ELSIF sel = '10' THEN
    z <= c;
ELSE
    z <= d;
END IF
END PROCESS.
```

→ if statement produces priority encoded logic.
→ Here, priority encoded logic means firstly the IF statement executes sequentially.
→ so depending on the first IF condition that particular i/p is passed to o/p.

www.Jntufastupdates.com

Scanned by CamScanner

→ Next depending on the second IF condition, the another i/p passed to o/p.

→ According to this Logic, for this example 3 mux generated by tool.

→ So in this case hardware is more so the delay generated by logic gates is more so the speed is Reduced.

CASE Statement

→ The Case statement selects one of the branches for execution based on the value of the expression

→ the value of the expression must be of discrete type of one dimensional array type.

Syntax:

```
CASE expression is
  WHEN choice1 =>
      (Statements)
  WHEN choice2 =>
      (Statements)
  WHEN OTHERS
      (Statements)
END CASE
```

fx:

```
PROCESS (sel, a, b, c, d)
BEGIN
CASE sel IS
WHEN '00' =>
   z <= a;
WHEN '01' =>
   z <= b;
WHEN '10' =>
   z <= c;
WHEN OTHERS
   z <= d;
END CASE
END PROCESS;
```

→ CASE Statement produces parallel Logic

→ Here in CASE Statement depending on the value of "sel" one of the four i/p's passed through o/p.

→ So single mux of 4 i/p's with two selection lines and one o/p is generated by tool.

→ So the general hardware is less, so the delay by logic Gates

www.Jntufastupdates.com

Scanned by CamScanner

...cked process

→ clocked process don't have Else clause.

ex:

```
PROCESS (clk)
IF clk = '1' AND clk'EVENT THEN
  z <= a;
END IF;
END PROCESS.
```

→ VHDL code for Rising edge of clock : clk = '1' and clk'EVENT.

→ VHDL code for falling edge of clock : clk = '0' and clk'EVENT.

Care for clocked process.

→ when initial State of flipflop may not be known and it may cause undesired operation of ckt.

→ therefore, inferred flipflop must be reset / preset to initial value at startup.

Asynchronous Reset

→ flipflop is cleared as soon as reset is asserted. Irrespective of clock when the reset signal is asserted, the flipflop is cleared.

→ if Reset is inactive and when clock is there, the i/p 'd' is passed to o/p.

ex:

```
PROCESS (clk, rst)
BEGIN
IF rst = '1' THEN
  Q <= 0;
ELSIF clk = '1' AND clk'EVENT THEN
```


www.Jntufastupdates.com

Scanned by CamScanner

```
Q <= d;
END IF;
END PROCESS.
```

Synchronous Reset:

→ flipflop is cleared on the edge of clk when reset is active.

ex:
```
PROCESS (clk)
BEGIN.
IF clk='1' AND clk'EVENT THEN
  IF RST='1' THEN
    Q <= 0;
  ELSE
    Q <= d;
ELSE IF
END IF;
END PROCESS.
```

How latch is inferred

→ when the feedback is Required in Combinational logic to hold the previous o/p value due to unspecific State, a latch is inferred.

ex: if 'en' is 1 then 'd' is assigned to o/p, but if you don't define what to do if 'en' is 0 then latch is inferred.

⇒
```
PROCESS (en, d)
BEGIN
IF en='1' THEN
  Q <= d;
END IF;
```


www.Jntufastupdates.com

Scanned by CamScanner

→ Latch can be avoided by Specifing Condition Completely.

fx:
```
PROCESS (en, d)
BEGIN
IF en='1' THEN
  Q<=d;
ELSE
  Q <= '1';
END IF
END PROCESS.
```

Concurrent Statements

Simple Assigment statement

→ A logic (or) arithmatic expression is called a simple assignment statement

fx: `f<= X AND Y.`

Selected signal Assignment

→ A selected signal Assignment allows a signal to be assigned one of multiple values based on a selection criterion.

fx:
→ 2 to 1 multiplexer.
→ w0, w1, S are the i/p's of mux.
→ f is o/p.

→ the selected signal assigment with the keyword WITH which specifies that 'S' is to be used for the selection criterion.

→ the two WHEN clauses States that

→ f is assigned the value of w0 when S=0

→ f is assigned the value of w1 when S=1

WHEN "01" => z <= L

www.Jntufastupdates.com

Scanned by CamScanner

Ex: WITH S select

f <= w0 . WHEN '0' ;

[illegible] WHEN OTHERS ;

Conditional signal Assignment

→ It is similar to that of selected signal assignments.

→ It allows a signal to be set to one of several values.

Ex: 2 to 1 mux signal Assignment statement is

f <= w0 when S='0'

ELSE w1

→ The common property of all the above Statements is that the ordering doesn't effect the Meaning of the code.

→ Hence these Statements are called Concurrent Assignment statements.

multiple Drivers

→ Hardware is not always a sequential ckt.

→ There are many ckts which generates output signal in parllel.

→ To model such hardware we need current Statements.

→ The VHDL code supports Concurrent Statements. when Statements are Concurrent, they execute in parllel.

→ Concurrent signal assignment Statement produces multiple drivers

Ex: C <= A

multiple Drivers

C <= A AND B ;

C <= D AND E ;

→ Drawback of concurrent Statements is it provides multiple

www.Jntufastupdates.com

Scanned by CamScanner

So to Solve this problem you have to resolve the o/p. user written resolution function defines it.

→ The value of each driver is an input to the resolution function and based on the computation performed with in resolution function

→ the value returned from this function becomes the resolved value for the signal

---

Variable Assignment

→ A variable assignment Statement replaces the current value of a variable with a new value specified.

ex: PROCESS (A, B)

variable M, N : Integer;

begin

M := A;

N := B;

Z <= M+N;

end process.

NULL Statements

→ the null statement performs no action. It is usually used with the case statements., to indicate that under certain conditions, no action required

ex: PROCESS (S, Z)

BEGIN

CASE S is

WHEN "00" => Z <= a;

WHEN "01" => Z <= b

www.Jntufastupdates.com

Scanned by CamScanner

WHEN "10" => z <= c;

WHEN "11" => z <= d;

WHEN OTHERS => NULL;

END CASE

END PROCESS.

Previous Question (2nd unit & 3rd unit)

→ Explain the Structure of various loop Statements in VHDL with examples

→ Explain the difference in program structure of VHDL and any other procedural language. Given an example

→ Design a 2 to 4 decoder ckt. Give its entity declaration behavioural model. also draw the waveform giving Relation btw inputs & o/ps.

→ Explain about variable assignment, signal assignment Statement & wait Statement.

→ write a process based VHDL program for the prime-number detector of 4 bit i/p and explain the flow using logic ckt

→ Discuss Inertial Delay model

→ Explain the Concept of internal logic synthesizer and also draw the Schematic.

→ Discuss the binding ? Discuss the binding btw entity & Components

→ Explain about data flow design elements of VHDL

→ write a VHDL program for Comparing 8 bit unsigned no's.

→ with examples explain the sequential Assignment Statements

→ Discuss transport Delay model

→ Explain about Inside of a logic Synthesizer and Give its Sc

→ Discuss about signal Drivers.

→ Explain about the following i) packages with Syntax ii) libraries with syntax

→ what is the significance of time dimension in VHDL ? Explain its

www.Jntufastupdates.com

Scanned by CamScanner

Types of Concurrent Statements

WHEN Statement

→ In this concurrent statement whenever an event occured on signal used in condition, the conditional signal assignment is used

```
D <= A WHEN SEL_A = "0" ELSE
     B WHEN SEL_B = "1" ELSE
     C;
```

WITH SELECT

Syntax:

```
WITH expression SELECT
    Target <= Expression1 WHEN choice1
              Expression2 WHEN choice2
              Expression3 WHEN choice3
```

→ It evaluates each choice expression and compares the value with each choice

→ the selected signal assignment statement selects different values for target signal based on the value of select expression.

→ the semantics of selected signal assignment statement is statement is, when ever an event occurs on a signal in the select expression (or) on the expressions the statement is executed.

→ Based on value of select expression that matches the choice value Specified, the value of corresponding expression is scheduled to be assigned to the target signal.

Note:

choices are not evaluated in sequence

An other choices cover values not covered explicitly

www.Jntufastupdates.com

Scanned by CamScanner

Ex: WITH SELECT

WITH sel-a SELECT

? ← a WHEN "00"

b WHEN "01"

c WHEN "10"

0 WHEN OTHERS.

Component Statement

www.Jntufastupdates.com

Scanned by CamScanner

Wait Statement

→ the wait statement is a statement that causes suspension of a process or a procedure

Simplified syntax

```
wait;
wait on signal_list;
wait until condition;
wait for time;
```

Description

→ The wait statement suspends the execution of the process (or) procedure in which it is specified.

→ Resuming the process (or) procedure depends on meting the condition(s) specified in the wait statement

→ there are 3 types of conditions supported with wait statements

→ sensitivity list → Conditional clause → timeout clause

Sensitivity list

→ A sensitivity list defines a set a signals to which the process is sensitive and causes the process to ~~result~~ resume.

fx:

```
Signal S1, S2 : Std_Logic;
...
process
 begin
...
wait on S1, S2;
end process;
```

→ after executing all statements, the process will be Suspended on the wait statements and will be Resumed when one of the S1 or S2

Scanned by CamScanner

Condition clause

→ A process is resumed when the logical condition turns true after any signal used in the condition

```
Ex: wait until Enable = '1'
    -- this is equivalent to
    -- loop
    -- wait on Enable
    -- exit when Enable = 1
    -- end loop.
```

→ the wait statement will resume the process when the Enable signal change its value to '1'

→ this is equivalent to the loop described in the comment below the first line

→ please note that the process resumed on any change of Enable signal

→ however, it will awake the rest of process only when the new value is '1'

Time out clause:

→ the time out clause defines the maximum time interval during which the process is not active.

→ when the time elapses, the process is automatically resumed.

Ex:

```
G: process
begin
Go <= '1' after 5ns
      '0' after 10ns
      '1' after 15ns
      '0' after 20ns
G1 <= '1' after 5ns.
      '0' after 15ns
wait ;
end process G;
```

wait for 50 ns — A process containing this statement will be suspended for 50ns.

→ In this process the value of signals G1 and Go are set to '11', '10', '01' and '00' at the time intervals 5, 10, 15 and 20 ns respectively

→ when the wait statement is encountered, the process is suspended forever

www.Jntufastupdates.com

Scanned by CamScanner

Condition clause

→ A process is resumed when the logical condition turns true due to change of any signal listed in the condition

```
ex: wait until Enable = '1'
    -- this is equivalent to
    -- loop
    -- wait on Enable
    -- exit when Enable 1
    -- end loop.
```

→ the wait statement will Resume the process when the Enable signal change its value to '1'

→ this is equivalent to the loop described in the comment below the first line

→ please note that the process resumed on any change of Enable signal

→ however, it will awake the rest of process only when the new value is '1'

Time out clause:

→ The time out clause defines the maximum time interval during which the process is not active.

→ when the time elapses, the process is automatically resumed.

ex: wait for 50ns

A process containing this statement will be suspended for 50ns.

```
G: Process
begin
G0 <= '1' after 5ns
      '0' after 10ns
      '1' after 15ns
      '0' after 20ns
G1 <= '1' after 5ns,
      '0' after 15ns.
wait ;
end process G;
```

→ In this process the value of signals G1 and G0 are set to '11', '10', '01' and '00' at the time intervals 5, 10, 15 and 20 ns respectively

→ when the wait statement is encountered, the process is suspended forever

Note: → The wait statement can be located any where btw begin & end process

www.Jntufastupdates.com

Scanned by CamScanner

Statement

→ Statement that includes a sequence of statements that is to be executed repeatedly; zero or more times.

Syntax:

i) while loop

```
loop_label : while condition loop
    sequence of statements
    end loop loop_label ;
```

ii) for loop

```
loop_label : for loop_parameter in 'range' loop
    sequance of statements
    end loop loop_label;
```

Description

→ the loop statement contains a sequence of statements, which are supposed to be repeated many time.

→ the statement also lists the conditions for repeating the sequance or specifies the no. of iterations.

~~Ex 1: the process (which is a clocking signal generator) contains a loop~~

→ A loop statement can have several different forms depending on the iteration scheme preceding the reserved word loop.

Ex 1:

The process (which is a clocking signal generator) contains a loop without an iteration scheme, which will iterate indefinetly.

```
Signal clock : BIT := '0';
...
clk_1 : process (clock)
    begin
```


Scanned by CamScanner

```
L1 : loop
    clock <= not clock after 5ns;
    end loop L1;
end process clk_1;
```

Ex: The infinate loop becomes in practice a finite, as the Iterations will terminate as soon as the variable "A" becomes greater than 10.

```
L2 : loop
    A := A+1;
    exit L2 when A>10;
    end loop L2
```

Ex: The loop 3 will be Repeated as long as the value of the variable i is not greater than 8.

→ when i reaches the value of 9 the loop is no longer repeated.

```
shift-3 : process (Input_x)
    variable i: POSITIVE := 1;
    begin
    L3: while i <= 8 loop
     output_x(i) <= Input_x(i+8) after 5ns;
      i := i+1;
     end loop L3;
     end process shift_3;
```

Ex: The loop statement parameter count_value will cause the loop to execute 8 times, with the value of count_value changing from 1 to 8.

```
Shift-4 : process (Input_x)
    begin
```


www.Jntufastupdates.com

Scanned by CamScanner

```
L4: for count_value in 1 to 8 loop
    output_x (count_value) <=
    Input_x (count_value +8) after 5 ns;
    end loop L4
    end process shift_4;
```

Ex:

→ The subtype Range_Type defines the range of Integer values from 1 to 8.

→ The parameter count_value changes according to this specification i.e., from 1 to 8

```
Shift_5 : process (Input_x).
    Subtype Range_Type is POSITIVE range 1 to 8;
    begin
    L5: for Count_value in Range_Type loop
        output_x (count_value) <=
        Input_x (count_value +8) after 5 ns.
        end loop L5
        end process shift_5;
```

Note

→ The parameter for a 'for' loop doesn't need to be specified - The loop declaration implicitly declared it.

→ The loop parameter is a constant with in a loop, which means that it may not be assigned any values inside the loop.

www.Jntufastupdates.com

Scanned by CamScanner

Next Statement

→ the next statement is used to complete execution of one of the iterations of an enclosing loop statement

→ the completion is conditional if the statement includes a condition

Syntax

```
next;
next loop_label;
next loop_label when condition;
```

Description

→ the next statement allows to skip a part of an iteration loop.

→ if the condition specified after the when reserved word is TRUE, or if there is no condition at all, then the statement is executed.

→ this results in skipping all the statements below it until the end of the loop and passing the control to the first statement in the next iteration.

Ex:

```
Loop_1: for count_value in 1 to 8 loop
   Assign_1 : A (count_value) := '0';
              next loop_1 when condition_1;
   Assign_2 : A (count_value+8) := '0'
   end loop Loop_1;
```

→ if the condition_1 in the iteration count_value is TRUE, then the next statement

→ the next statement to be executed will be Assign_1 in the iteration count_value + 1

→ otherwise, the sequence of operation is as specified i.e., Assign_2 executed.

www.Jntufastupdates.com

Scanned by CamScanner

Exit Statement

→ The exit statement is used to finish (or) exit the execution of an enclosing loop statement.

→ If the exit statement includes a condition, then the exit from the loop is conditional

Syntax:

```
exit;
exit loop_label;
exit loop_label when condition;
```

Description

→ The exit statement terminates entirely execution of the loop in which it is located.

→ The execution of a exit statements depends on a condition placed at the end of the statement, right after the when reserved word.

→ When the condition is TRUE (or if there is no condition at all) the exit statement executed and the control is passed to the first statement after end loop.

Ex1:

```
Loop_1 : for count_value in 1 to 10 loop
             exit loop_1 when reset = '1';
A_1 :    A(count_value) := '0';
         end loop loop_1;
A_2 <=   A after 10 ns;
```

→ At the begining of each iteration of the loop_1 loop, the reset = '1' condition is checked.

→ If the condition is FALSE, then the reset of the loop is executed (in this case assignedments labeled A_1)

→ otherwise, the control is passed to ... after the loop.

www.Jntufastupdates.com

Scanned by CamScanner

Assertion Statement

→ A statement that checks that a specified condition is true. Reports an error if it is not.

Syntax:

assert condition
report string
Severity Severity_Level;

Description

→ The assertion statement has 3 optional fields and usually all three are used.

① Condition → The condition specified in an assertion statement must be evaluate to Boolean value (true or false). if it is false, it is said that an assertion violation occured.

String → The condition specified in ~~the report~~ clause must be predefined type String and is a message to be reported when assertion violation occured.

Severity → If the severity clause is present, it must specify an expression of predefined type SEVERITY_LEVEL, which determines the Severity level of the assertion violation.

→ The SEVERITY_LEVEL type is specified in the STANDARD Standard package and contains following values:

NOTE, WARNING, ERROR and FAILURE.

→ If the Severity clause omitted it is implicitly assumed to ERROR.

→ when an assertion violation occurs, the report is issued and displayed on the Screen. The supported Severity Level supplies an information

www.Jntufastupdates.com

Scanned by CamScanner

- The Severity level defines the degree to which the assertion affects operation of the system

NOTE
→ NOTE can be used to pass information messages from simulation

WARNING
→ WARNING can be used to unusual situation in which the simulation can be continued, but the results may be unpredictable

ERROR
→ ERROR can be used when assertion violation makes continuation of simulation not feasible

FAILURE
→ FAILURE can be used when the assertion violation is a fatal error and the simulation must be stopped at once

→ Assertion statements are not only sequential, but used concurrently Statement as well.

→ A concurrent assertion statement represents a passive process Statement containing the specified assertion statement.

Ex 1:

```
assert Status = OPEN_OK
    report "The call to FILE_OPEN was not Successful"
    Severity WARNING;
```

→ Having called the procedure FILE_OPEN, if the Status is different from OPEN_OK, it is indicated by the warning messag

Ex 2

```
assert not (S='1' and R='1')
    report "Both values of signals S and R are equal to '1' "
    Severity ERROR
```

→ When the values of the signals S and R are equal to '1', the message is displayed

www.Jntufastupdates.com

Scanned by CamScanner

is set to ERROR.

Ex3:

```
assert operation_code = "0000"
  report "Illegal code of operation"
  Severity FAILURE;
```

→ Event like illegal operation code are server errors and should cause immediate termination of the simulation, which is forced by the Severity level FAILURE.

Note:

→ The message is displayed when condition is NOT met, therefore the message should be an opposite to the condition

→ Concurrent assertion statement is a passive process and as such can be specified in an entity.

→ concurrent assertion statement monitors specified condition continiously.

→ Synthesis tools generally ignore assertion statements.

www.Jntufastupdates.com

Scanned by CamScanner