

VHDL

Instructor: Saraju P. Mohanty

In this Lecture

- If, Case, Null, Exit, Next, While, For statements
- Assertion and Report Statements
- Arrays and Records
- Basic Modeling
- Subprograms
- Packages and Use Clauses
- Aliases
- Resolved Signals and Ports

Note: The slides are from text book or reference book authors or publishers, or from Dr. Katkoori, Dept. of CSE, Univ. of South Florida.

Sequential Statements

These statements can appear inside a process description :

- variable assignments
- if-then-else
- case
- loop
 - infinite loop
 - while loop
 - for loop
- assertion and report
- signal assignments
- function and procedure calls

If Statement: Examples

- When circuit behavior depends on a set of conditions that may or may not hold good during simulation.

- Example 1:

```
if sel = 0 then
    result <= input_0; -- executed if sel = 0
else
    result <= input_1; -- executed if sel /= 0
end if;
```

- Example 2:

```
if sel = 0 then
    result <= input_0; -- executed if sel = 0
elseif sel = 1 then
    result <= input_1; -- executed if sel = 1
else
    result <= input_2; -- executed if sel /= 0, 1
end if;
```

Case Statement: Examples

- When circuit behavior depends on the values of a single expression, we can use a case statement.
- Example: Suppose we are modeling an ALU with the following control input func declared as enumeration type.

```
type alu_func is (pass1, pass2, add, sub);
```

The behavior can be described as follows:

case func is

```
    when pass1 => results := operand1;
```

```
    when pass2 => results := operand2;
```

```
    when add => results := operand1 + operand2;
```

```
    when sub => results := operand1 - operand2;
```

```
end case;
```

Null Statement

- To take care of the conditions when no action is needed to be performed.

- Example:

case func is

when pass1 => results := operand1;

when pass2 => results := operand2;

when add => results := operand1 + operand2;

when sub => results := operand1 - operand2;

when nop => null;

end case;

Loop Statements

VHDL provides three types of loop or iterative constructs:

- infinite loop
- while loop
- for loop

Infinite Loop

- Repeats a sequence of statements indefinitely.
- We avoid this situation in any high level programming language.
- In digital systems this is useful as hardware devices repeatedly perform the same operation as long as power supply is on.
- Typical structure: Loop statement in process body with a wait statement.

Infinite Loop: Example

```
p1: process is
begin
    .....
    L1: loop
        .....
        L2 : loop
            .....
            -- nested loop
            .....
        end loop;
    .....
end loop;
wait;
end process;
```


Exit Statement: Examples

- Exit a loop when some conditions arise.
- Exit statement is needed inside a loop.
- Examples:

```
loop
```

```
...
```

```
exit ; -- jumps out of the inner most loop
```

```
...
```

```
end loop;
```

```
..... -- exit causes the execution to start from here
```

```
exit loop1; -- jumps out of loop  
-- with label loop1
```

```
exit when x = 1; -- jumps out of inner  
-- most loop when  
-- condition is true
```

While loop: Example

entity cos is

port (theta: in real; result: out real;);

end entity;

architecture series of cos is

begin

P1: process (theta) is

variable sum, term, n: real;

begin

sum := 1.0; term := 1.0; n := 0.0;

while abs term > abs (sum/1.0E6) loop

n := n + 2.0;

term := (-term) * (theta ** 2) / ((n-1) * n);

sum := sum + term;

end loop;

result <= sum;

end process;

end architecture;

$$\cos \theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \dots$$

For Loop

- Construct for specifying deterministic iteration.

```
for_loop_stmt <=
    [ loop_label : ]
    for id in discrete_range loop
        { sequential_stmt }
    end loop [ loop_label ] ;
discrete_range <= expr ( to | downto ) expr
```

```
for count in 0 to 127 loop
    count_out <= count;
    wait for 5 ns;
end loop;
```

For Loop: rules

- Loop parameter's type is the base type of the discrete range.
- Loop parameter is a constant inside the loop body.
- It can be used in an expression but not written to.
- Loop parameter is not required to be explicitly declaration.
- Loop parameter's scope is defined by the loop body.
- Consequently, it hides any variable of the same name inside the loop body.

For Loop: Example

```
P1: process is
    variable i, j: integer;
begin
    i := loop_param;           -- ERROR
    for loop_param in 1 to 10 loop
        ...
        loop_param := 5;      -- ERROR
        ...
    end loop;
    j := loop_param;          -- ERROR
end process;
```

Assertion and Report Statement

- A functionally correct model may need to satisfy certain conditions.
- Some of these can be specified by “assert” statements.
- Report Statement are useful for providing extra information from specific assertion statements (as there can be several assertion statements).
- Assert and report statements are particularly useful for de-bugging.

Assert and Report Statement: Example

```
assert value <= max_value;
```

```
assert value <= max_value  
    report "Value too large";
```

```
type severity_level is (note,warning,error,failure);
```

```
assert clock_width >= 100 ns  
    report "clock width too small"  
    severity failure;
```

Composite Data Types and Operations

- Composite data elements consists of collections of related data elements: Arrays and Records
- An object of a composite type can be treated as a single object.
- Also, individual elements of it can be manipulated.

Arrays

- Array: Collection of values, all of which are of same type.
- Position of each element is given by a scalar value, called index.
- Example 1: type word is array (0 to 31) of bits;
- Example 2: type word is array (31 to 0) of bits;
- Non-numeric index Example:

type controller_state is (initial, idle, active, error)

type state_counts is array (idle to error) of natural;

Arrays

- Multidimensional Arrays:

type matrix is array (1 to 3, 1 to 3) of real;

variable transform: matrix;

- Unconstrained array types: Just types are indicated and bounds are not specified.

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

variable short_sample_buf: sample (0 to 63); -- indicates
that index variable is natural number from 0 to 63

- Strings, bit vectors, standard logic arrays are predefined unconstrained array type.

Records

- A record is a composite value comprising of elements that may be of different types from one another.

- Example:

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 time_stamp;

variable sample_time, current_time: time_stamp;

Basic Modeling Constructs

- Modules description:
 - Entity: describes external declaration
 - Architecture: describes internal implementation
- Example:

```
ENTITY example1 IS
```

```
    PORT ( x1, x2, x3  : IN  BIT ;  
           f           : OUT BIT ) ;
```

```
END example1 ;
```

```
ARCHITECTURE LogicFunc OF example1 IS  
BEGIN
```

```
    f <= (x1 AND x2) OR (NOT x2 AND x3) ;
```

```
END LogicFunc ;
```

Processes and Signals

- Concurrent statement in an architecture body describe a module's operation, for example a process.
- Signals are used for communicating between concurrently executing processes.
- A process that writes to a signal is called its driver.
- A “normal” signal can have one and only process as its driver.
- In other words, two processes cannot write to the same signal.

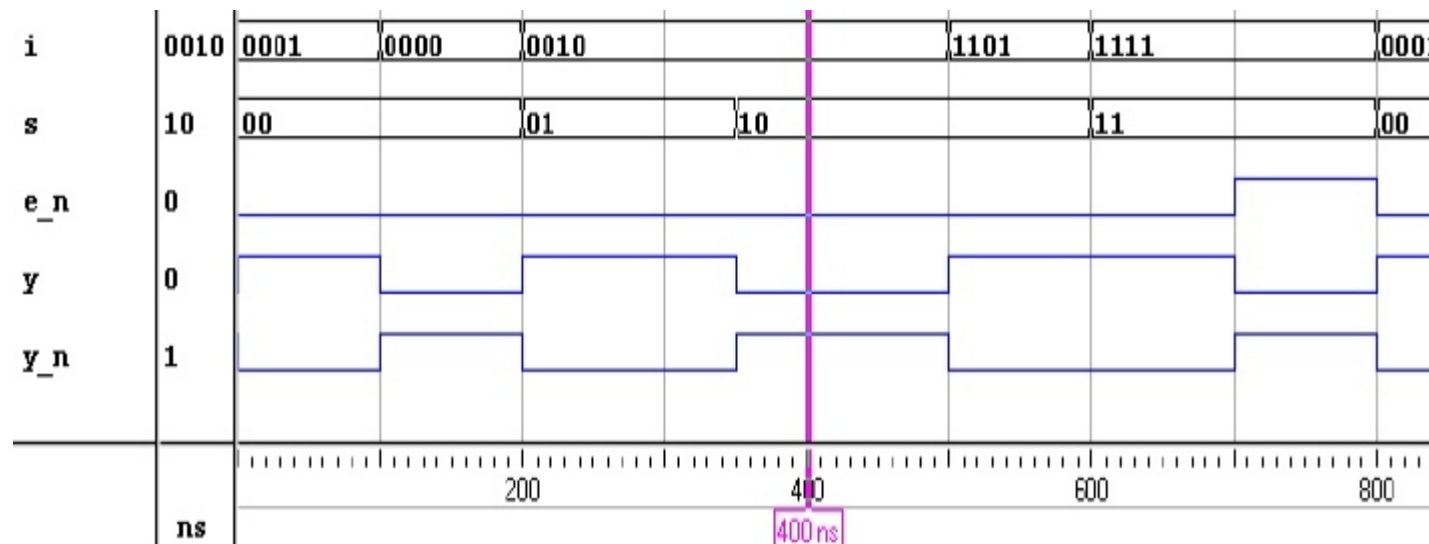
Signals and Transactions

- A signal assignment does not take effect immediately after the execution of the statement.
- A signal assignment schedules a transaction for the signal.
- The transaction is effected only when the process hits a wait statement.
- After all the processes are suspended (that is are at their respective wait statements) the transactions are processed.

Signals and Transactions: Example

```

E_N <= '0'; S <= "00"; I <= "0001"; wait for 100ns;
E_N <= '0'; S <= "00"; I <= "0000"; wait for 100ns;
E_N <= '0'; S <= "01"; I <= "0010"; wait for 150ns;
E_N <= '0'; S <= "10"; I <= "0010"; wait for 150ns;
E_N <= '0'; S <= "10"; I <= "1101"; wait for 100ns;
E_N <= '0'; S <= "11"; I <= "1111"; wait for 100ns;
E_N <= '1'; S <= "11"; I <= "1111"; wait for 100ns;
    
```



source: http://users.etech.haw-hamburg.de/users/reichardt/chapt_9.pdf

Delay Mechanism

- VHDL provides two kinds of delay mechanism for signal assignments:

- Transport delay
- Inertial delay (default)

- Example:

`x <= transport '1' after 5ns;`

`y <= inertial '0' after 10 ns;`

- Due to delay mechanisms a new transaction may effect already scheduled transactions on the same signal.

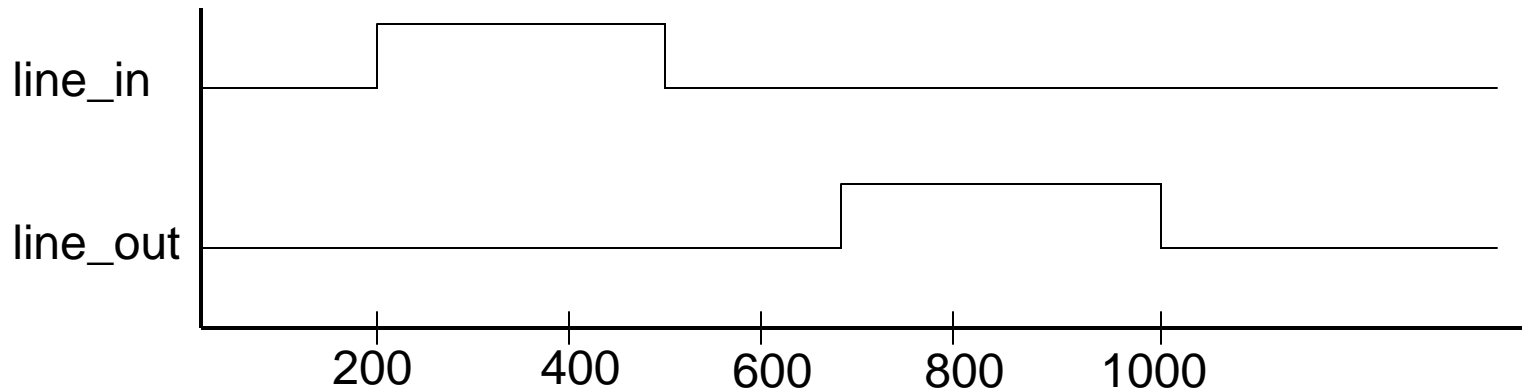
- Inertial delay and reject limit:

`s <= reject 5 ns inertial '1' after 8 ns`

Look in the window of 5 ns previous to the new transaction.

Delay Mechanism: Example

```
transmission_line : process (line_in) is  
begin  
    line_out <= transport line_in after 500ps;  
end process transmission_line;
```



Wait statements

```
wait_stmt <=  
    [ label : ] wait [ on signal_name{ , ... } ]  
                    [ until boolean_expr ]  
                    [ for time_expr ] ;
```

```
wait;  
wait on a, b, c;  
wait until x = 1;  
wait for 100 ns;
```

Wait for

- “Wait for” results in the process being suspended for the time specified in the construct.

wait for 10 ns;

Wait on

- “Wait on” results in the process being suspended until an event takes place on any one of the signals.
- The list of signals is also called a sensitivity list.

```
half_adder: process is  
begin  
    s <= a xor b after 10 ns;  
    c <= a and b after 10 ns;  
    wait on a, b;  
end process;
```

```
half_adder: process (a, b) is  
begin  
    s <= a xor b after 10 ns;  
    c <= a and b after 10 ns;  
end process;
```

Wait until

- wait until *condition*;
- In the simple case the condition expression must contain at least one signal (maybe more), say “s1”.
- The “wait until” construct is then interpreted as follows:
wait on s1 until *condition*;
- The list of signals (similar to s1) is also called the sensitivity list.

Wait until

- wait on s1, s2, s3 until *condition*;
- The condition is evaluated only when an event occurs on a signal in the sensitivity list.
- The process is resumed when the condition evaluates to TRUE.
- Hence the process is resumed when
 - An event occurs in the sensitivity list and
 - The condition evaluates to TRUE.

Procedures

- Procedures and functions are subprogram facilities in VHDL.
- Procedure: Declared and then called
- Example:

```
procedure average_samples is  
    variable total: real := 0.0;
```

```
-----
```

```
-----
```

```
end procedure average_samples;
```

This can be called inside a process as:

```
    average_samples;
```

Functions

- Syntax is very similar to that of the procedures.
- Unlike a procedure, a function calculates and returns a result that can be used in an expression.
- Parameters of the function must be of 'in' mode and may not be of class variable.
- Example:

```
function bv_add (bv1, bv2 : in bit_vector) return bit_vector is  
begin
```

```
-----
```

```
end function bv_add;
```

```
signal source1, source2, sum: bit_vector (0 to 31);
```

```
adder: sum <= bv_add(source1, source2) after T_delay_adder;
```


Packages

- An important way of organizing the data.
- A collection of related declaration grouped to serve a common purpose.
- The external view of a package is specified in 'package declaration'.
- Its implementation is provided in the 'package body'.
- The packages can be shared among models.
- Several predefined packages exist, such as IEEE standard packages.

Packages: Example

```
package PKG is -- package declaration
    type T1 is ...
    type T2 is ...
    constant C : integer;
    procedure P1 (...);
end PKG;
```

```
package body PKG is -- package body
    type T3 is ...
    C := 17;
    procedure P1 (...) is
        ...
    end P1;
    procedure P2 (...) is
        ...
    end P2;
end PKG;
```

Source: http://www.vhdl-online.de/~vhdl/tutorial/englisch/ct_257.htm

Uses Clauses

- We can write use clause to make a library unit directly visible in a model.
- This allows us to omit library name when referring to the library unit.
- An analyzed package is a library unit, so use clauses also apply to it.
- Example: `use ieee.std_logic_1164.all;`

Aliases

- Alias is a means to make a model clearer.
- We can use alias declaration to declare an alias for a object.
- Example1:
alias binary_string is bit_vector;
Declares objects to be of type binary_string and perform bit_vector operation.