Module #07 : Verilog HDL Behavioral Modeling Part#1

7.1: Behavioral Modeling:

- Verilog procedural statements are used to model a design at a higher level of abstraction
- Behavioral Modeling provides powerful ways of doing complex designs.
- Procedural statements can only be used in procedures (always and initial blocks)

7.2: Procedural Assignments:

- Procedural assignments are assignment statements used within Verilog procedures (always and initial blocks).
- Only reg variables and integers (and their bit/part-selects and concatenations) can be placed left of the "=" or "<=" in procedures.
- The right hand side of the assignment is an expression which may use any of the operator types described in Module #04.

Module #07 : Verilog HDL Behavioral Modeling

7.3: Delay in Assignment (not for synthesis):

- In a ***delayed assignment (Inter-Assignment Delay)*** Δt time units pass before the statement is executed and the left-hand assignment is made.
- With ***intra-assignment delay***, the right side is evaluated immediately but there is a delay of Δt before the result is place in the left hand assignment. If another procedure changes a right-hand side signal during Δt, it does not effect the output.
- Delays are not supported by synthesis tools.

Syntax:

for Procedural Assignment

variable = expression

Delayed assignment

#Δt variable = expression;

Intra-assignment delay

variable = #Δt expression;

Example7.1:

```
reg  sum, a, b, c;
sum = b ^ c; // execute now.
sum = #15 b ^ c; /* b ^ c; evaluated now; sum changed
after 15 time units. */
#10 sum = b ^ c; ; /* 10 units after sum changes, b ^ c; is
evaluated and sum changes. */
```

Module #07 : Verilog HDL Behavioral Modeling

7.4: Blocking Assignments ('='):

- Procedural (blocking) assignments (=) are done sequentially in the order the statements are written.
- A second assignment is not started until the preceding one is complete.
- "=" best corresponds to what c/c++ code would do; use it for combinational procedures

Syntax:

variable = expression;

variable = #Δt expression; //grab inputs now, deliver ans. later, don't delay next statement

#Δt variable = expression; //grab inputs later, deliver ans. later; delay next statement by #Δt

Module #07 : Verilog HDL Behavioral Modeling

Example 7.2: ***For simulation***

```
initial
begin
a=1; b=2; c=3;
#5 a = b + c; // wait for 5 units, and execute a= b + c =5.
d = #2a; // Time continues from last line, execute at t = 5 and assign at t= 5 + 2 = 7 units
```

Example 7.3: ***For synthesis***

```
always @( *)
begin
    Z=Y;
    Y=X;
end
```

Module #07 : Verilog HDL Behavioral Modeling

7.5: Non-Blocking Assignments ('<='):

- RTL (nonblocking) assignments (<=), start in parallel.
- The right hand side of nonblocking assignments is evaluated starting from the completion of the last blocking assignment or if none, the start of the procedure.
- The transfer to the left hand side is made according to the delays. An intra-assignment delay in a non-blocking statement will not delay the start of any subsequent statement blocking or non-blocking. However normal delays are cumulative and will delay the output.
- One must not mix "<=" or "=" in the same procedure.
- "<=" best mimics what physical flip-flops do; use it for "**always @ (posedge clk ..)"** type procedures.

Module #07 : Verilog HDL Behavioral Modeling

Syntax:

variable <= expression;

variable <= #Δt expression; ***//grab inputs now, deliver ans. later, don't delay next statement***

#Δt variable <= expression; ***//grab inputs later, deliver ans. later, don't delay next statement.***

Module #07 : Verilog HDL Behavioral Modeling

Example 7 .4: ***For simulation***

```
initial
  begin
    #3 b <= a; // grab a at t=0 Deliver b at t=3.
    #6 x <= b + c; // grab b + c at t=0, wait and assign x at t=6. x is unaffected by b's change.
  end
```

Example 7 .5: ***For synthesis***

```
always @( posedge clk)
 begin
    Z <= Y;
    Y <= X;
    y <= x;
    z <= y;
 end
```

Module #07 : Verilog HDL Behavioral Modeling

7.6: begin ... end:

- **begin ... end** block statements are used to group several statements for use where one statement is syntactically allowed.
- Such places include functions, always and initial blocks, if, case and for statements.
- Blocks can optionally be named

Syntax:

```
begin : block name
    reg reg_variable_list;
    integer  integer_list;
    parameter  parameter_list;
        ... statements ...
end
```

Example 7 .6:

```
function trivial_one; // The block name is "trivial_one."
   input a;
   begin: adder_blk; // block named adder, with
      integer i; // local integer i
       ... statements ...
   end
```