Delay and Conditional Statement

Chap 6: Introduction to HDL (e)

Credit to: MD Rizal Othman
Faculty of Electrical & Electronics Engineering
Universiti Malaysia Pahang

Ext: 6036

Inter-statement Delay

- The most common approach of delay modeling in test-benches
- In this technique a delay is inserted before or in-between statements
- Below is an example of inter-statement delay

```
initial
begin
 #0   X = 1; #0   Y = 2;  #0   Z = 0;
 #10 Z = X + Y;
end
```

Inter-statement Delay

- In the example all three variables X, Y and Z will be initialized at time 0
- The variable Z will get a sum of variables X and Y after delay of 10 time steps.

Intra-statement Delay

- This is an alternate technique of delay modeling by inserting delay as a part of statement execution
- Below is an example of inter-statement delay

```
initial
begin
 #0   X = 1;  #0   Y = 2; #0   Z = 0;
 Z =  #10 X + Y;
end
```

Intra-statement Delay

- In the above example # delay is moved inside the statement after ‘=’ sign
- This method of coding will cause the value of X+Y to be stored in a temporary register for 10 time units
- After that delay (10 time units) the register Z will get updated

Blocking and Nonblocking assignment

- Blocking assignments are executed in the order they are coded, hence they are sequential.
- “Blocking assignments” - they block the execution of next statement, till the current statement is executed.
- Assignment are made with "=" symbol. Example a = b;

Blocking and Nonblocking assignment

- Nonblocking assignments are executed in parallel.
- Since the execution of next statement is not blocked due to execution of current statement, they are called nonblocking statement.
- Assignments are made with "<=" symbol. Example a <= b;

Example **Blocking and Nonblocking assignment**

```
module blocking_nonblocking();

reg a,b,c,d;
// Blocking Assignment
initial begin
  #10 a = 0;
  #11 a = 1;
  #12 a = 0;
  #13 a = 1;
end

initial begin
  #10 b <= 0;
  #11 b <= 1;
  #12 b <= 0;
  #13 b <= 1;
end

initial begin
   c = #10 0;
   c = #11 1;
   c = #12 0;
   c = #13 1;
end

initial begin
   d <= #10 0;
   d <= #11 1;
   d <= #12 0;
   d <= #13 1;
end
```

begin ... end : Sequential Statement Groups

- Group several statements together.
- Cause the statements to be evaluated sequentially (one at a time)
  - Any timing within the sequential groups is relative to the previous statement.
  - Delays in the sequence accumulate (each delay is added to the previous delay)
  - Block finishes after the last statement in the block.

begin ... End (Example)

```
module sequential();

reg a;

initial begin
  $monitor ("%g a = %b",
$time, a);
  #10 a = 0;
  #11 a = 1;
  #12 a = 0;
  #13 a = 1;
end

endmodule
```

Simulator Output

```
 0 a = x
 10 a = 0
 21 a = 1
 33 a = 0
 46 a = 1
```

fork….join : **Parallel Statement Groups**

- Group several statements together.
- Cause the statements to be evaluated in parallel (all at the same time).
  - Timing within parallel group is absolute to the beginning of the group.
  - Block finishes after the last statement completes (Statement with highest delay, it can be the first statement in the block).

fork....join : **Parallel Statement Groups**

```
module parallel();

reg a;

initial
fork
  #10 a = 0;
  #11 a = 1;
  #12 a = 0;
  #13 a = 1;
join

endmodule
```

Simulator Output

```
 0 a = x
10 a = 0
11 a = 1
12 a = 0
13 a = 1
```

Looping Statements

- Looping statements appear inside procedural blocks only
- Verilog has four looping statements like any other programming language.
  - forever
  - repeat
  - while
  - for

The for loop statement

- for loop is the same as the for loop used in any other programming language.
  - Executes an < initial assignment > once at the start of the loop.
  - Executes the loop as long as an < expression > evaluates as true.
  - Executes a < step assignment > at the end of each pass through the loop.
- syntax : for (< initial assignment >; < expression >, < step assignment >) < statement >
- Note : verilog does not have ++ operator as in the case of C language.

The for loop statement

```
module for_example();

integer i;
reg [7:0] ram [0:255];

initial begin
  for (i = 0; i < 256; i = i + 1) begin
    #1 $display(" Address = %g  Data = %h",i,ram[i]);
    ram[i] <= 0; // Initialize the RAM with 0
    #1 $display(" Address = %g  Data = %h",i,ram[i]);
  end
  #1 $finish;
end

endmodule
```

while loop statement

- while loop executes as long as an < expression > evaluates as true. This is the same as in any other programming language.
- syntax : while (< expression >) < statement >

while loop statement

```
module while_example();

reg [5:0] loc;
reg [7:0] data;

always @ (data or loc)
begin
  loc = 0;
  // If Data is 0, then loc is 32 (invalid value)
  if (data == 0) begin
    loc = 32;
  end else begin
    while (data[0] == 0) begin
      loc = loc + 1;
      data = data >> 1;
    end
  end
  $display ("DATA = %b   LOCATION = %d",data,loc);
end

initial begin
  #1 data = 8'b11;
  #1 data = 8'b100;
  #1 data = 8'b1000;
  #1 data = 8'b1000_0000;
  #1 data = 8'b0;
  #1 $finish;
end

endmodule
```

repeat statement

- The repeat loop executes < statement > a fixed < number > of times.
- syntax : repeat (< number >) < statement >

repeat statement

```
module repeat_example();
reg  [3:0] opcode;
reg  [15:0] data;
reg        temp;

always @ (opcode or data)
begin
  if (opcode == 10) begin
    // Perform rotate
    repeat (8) begin
      #1 temp = data[15];
      data = data << 1;
      data[0] = temp;
    end
  end
end
// Simple test code
initial begin
   $display (" TEMP  DATA");
   $monitor (" %b     %b ",temp, data);
   #1 data = 18'hF0;
   #1 opcode = 10;
   #10 opcode = 0;
   #1 $finish;
end

endmodule
```

forever statement

- forever loop executes continually, the loop never ends. Normally we use forever statements in initial blocks.
- syntax : forever < statement >
- One should be very careful in using a forever statement: if no timing construct is present in the forever statement, simulation could hang. The code below is one such application, where a timing construct is included inside a forever statement.

forever statement

```
module forever_example ();
reg clk;

initial begin
  #1 clk = 0;
  forever begin
    #5 clk = !clk;
  end
end

initial begin
  $monitor ("Time = %d  clk = %b",$time, clk);
  #100 $finish;
end

endmodule
```

Module blocking

```
module blocking;

  reg[7:0] a, b, c, d, e;


  initial begin
    $monitor($time, " :\ta = %d\t", a,
                                  "b = %d\tc = %d\t", b, c,
                                  "d = %d\te = %d", d, e);
    #50 $finish;
  end

  initial begin
      a = 2;
      b = 5;
    #1 a = c;
    #1 a = d;
    #2 a = 4;
    #2 a = 7;
      b = 6;
    #2 a = d;
    $display("a, b - done");
  end

  initial begin
      c = 1;
      d = c;
      e = a;
    #2 e = d;
      c = 0;
      d = 3;
    #5 c = a;
      d = 1;
      d = 2;
    $display("c, d, e - done");
  end

endmodule /* blocking */
```

```
module blocking_intra;

  reg[7:0] a, b, c, d, e;


  initial begin
    $monitor($time, " :\ta = %d\t", a,
                                  "b = %d\tc = %d\t", b, c,
                                  "d = %d\te = %d", d, e);
    #50 $finish;
  end

  initial begin
    a = 2;
    b = 5;
    a = #1 c;
    a = #1 d;
    a = #2 4;
    a = #2 7;
    b = 6;
    a = #2 d;
    $display("a, b - done");
  end

  initial begin
    c = 1;
    d = c;
    e = a;
    e = #2 d;
    c = 0;
    d = 3;
    c = #5 a;
    d = 1;
    d = 2;
    $display("c, d, e - done");
  end

endmodule /* blocking_intra */
```

Module nonblocking

```
module non_blocking;

  reg[7:0] a, b, c, d, e;


  initial begin
    $monitor($time, " :\ta = %d\t", a,
                              "b = %d\tc = %d\t", b, c,
                              "d = %d\te = %d", d, e);
    #50 $finish;
  end

  initial begin
      a <= 2;
      b <= 5;
    #1 a <= c;
    #1 a <= d;
    #2 a <= 4;
    #2 a <= 7;
      b <= 6;
    #2 a <= d;
    $display("a, b - done");
  end

  initial begin
      c <= 1;
      d <= c;
      e <= a;
    #2 e <= d;
      c <= 0;
      d <= 3;
    #5 c <= a;
      d <= 1;
      d <= 2;
    $display("c, d, e - done");
  end

endmodule /* non_blocking */
```

```
module non_blocking_intra;

  reg[7:0] a, b, c, d, e;


  initial begin
    $monitor($time, " :\ta = %d\t", a,
                              "b = %d\tc = %d\t", b, c,
                              "d = %d\te = %d", d, e);
    #50 $finish;
  end

  initial begin
    a <= 2;
    b <= 5;
    a <= #1 c;
    a <= #1 d;
    a <= #2 4;
    a <= #2 7;
    b <= 6;
    a <= #2 d;
    $display("a, b - done");
  end

  initial begin
    c <= 1;
    d <= c;
    e <= a;
    e <= #2 d;
    c <= 0;
    d <= 3;
    c <= #5 a;
    d <= 1;
    d <= 2;
    $display("c, d, e - done");
  end

endmodule /* non_blocking_intra */
```