Verilog Behavioral Modeling Part-I

Feb-9-2014

Verilog HDL Abstraction Levels

- Behavioral Models : Higher level of modeling where behavior of logic is modeled.
- RTL Models : Logic is modeled at register level
- Structural Models : Logic is modeled at both register level and gate level.

Procedural Blocks

Verilog behavioral code is inside procedure blocks, but there is an exception: some behavioral code also exist outside procedure blocks. We can see this in detail as we make progress.

There are two types of procedural blocks in Verilog:

- **initial :** initial blocks execute only once at time zero (start execution at time zero).
- **always :** always blocks loop to execute over and over again; in other words, as the name suggests, it executes always.

Example - initial

```
1  module initial_example();
2  reg clk,reset,enable,data;
3
4  initial  begin
5   clk = 0;
6   reset = 0;
7   enable = 0;
8   data = 0;
9  end
10
11 endmodule
```

You could download file initial_example.v here

In the above example, the initial block execution and always block execution starts at time 0. Always block waits for the event, here positive edge of clock, whereas initial block just executed all the statements within begin and end statement, without waiting.

Example - always

```
1  module always_example();
2  reg clk,reset,enable,q_in,data;
3
4  always @ (posedge clk)
5  if (reset)  begin
```

```
6     data <= 0;
7  end else if (enable) begin
8     data <= q_in;
9  end
10
11 endmodule
```

You could download file always_example.v here

In an always block, when the trigger event occurs, the code inside begin and end is executed; then once again the always block waits for next event triggering. This process of waiting and executing on event is repeated till simulation stops.

Procedural Assignment Statements

- Procedural assignment statements assign values to reg, integer, real, or time variables and can not assign values to nets (wire data types)
- You can assign to a register (reg data type) the value of a net (wire), constant, another register, or a specific value.

Example - Bad procedural assignment

```
1  module initial_bad();
2  reg clk,reset;
3  wire enable,data;
4
5  initial begin
6   clk = 0;
7   reset = 0;
8   enable = 0;
9   data = 0;
10 end
11
12 endmodule
```

You could download file initial_bad.v here

Example - Good procedural assignment

```
1  module initial_good();
2  reg clk,reset,enable,data;
3
4  initial begin
5   clk = 0;
6   reset = 0;
7   enable = 0;
8   data = 0;
9  end
10
11 endmodule
```

You could download file initial_good.v here

Procedural Assignment Groups

If a procedure block contains more than one statement, those statements must be enclosed within

- Sequential **begin - end** block
- Parallel **fork - join** block

When using begin-end, we can give name to that group. This is called **named blocks**.

Example - "begin-end"

```
1  module initial_begin_end();
2  reg clk,reset,enable,data;
3
4  initial begin
5   $monitor(
6     "%g clk=%b reset=%b enable=%b data=%b",
7     $time, clk, reset, enable, data);
8   #1   clk = 0;
9   #10  reset = 0;
10  #5   enable = 0;
11  #3   data = 0;
12  #1  $finish;
13 end
14
15 endmodule
```

You could download file initial_begin_end.v here

Begin : clk gets 0 after 1 time unit, reset gets 0 after 11 time units, enable after 16 time units, data after 19 units. All the statements are executed sequentially.

Simulator Output

```
0 clk=x reset=x enable=x data=x
1 clk=0 reset=x enable=x data=x
11 clk=0 reset=0 enable=x data=x
16 clk=0 reset=0 enable=0 data=x
19 clk=0 reset=0 enable=0 data=0
```

Example - "fork-join"

```
1  module initial_fork_join();
2  reg clk,reset,enable,data;
3
4  initial begin
5   $monitor("%g clk=%b reset=%b enable=%b data=%b",
6     $time, clk, reset, enable, data);
7   fork
8     #1   clk = 0;
9     #10  reset = 0;
10    #5   enable = 0;
11    #3   data = 0;
12  join
13  #1  $display ("%g Terminating simulation", $time);
14  $finish;
15 end
16
17 endmodule
```

You could download file initial_fork_join.v here

Fork : clk gets its value after 1 time unit, reset after 10 time units, enable after 5 time units, data after 3 time units. All the statements are executed in parallel.

Simulator Output

```
0 clk=x reset=x enable=x data=x
1 clk=0 reset=x enable=x data=x
3 clk=0 reset=x enable=x data=0
5 clk=0 reset=x enable=0 data=0
```

```
10 clk=0 reset=0 enable=0 data=0
11 Terminating simulation
```

Sequential Statement Groups

The **begin - end** keywords:

- 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.

Example - sequential

```
1  module sequential();
2
3  reg a;
4
5  initial begin
6    $monitor ("%g a = %b", $time, a);
7    #10  a = 0;
8    #11  a = 1;
9    #12  a = 0;
10   #13  a = 1;
11   #14  $finish;
12 end
13
14 endmodule
```

You could download file sequential.v here

Simulator Output

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

Parallel Statement Groups

The fork - join keywords:

- 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).

Example - Parallel

```
1  module parallel();
2
3  reg a;
4
5  initial
6  fork
7    $monitor ("%g a = %b", $time, a);
8    #10  a = 0;
9    #11  a = 1;
```

```
10    #12  a = 0;
11    #13  a = 1;
12    #14  $finish;
13  join
14
15  endmodule
```

You could download file parallel.v here

Simulator Output

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

Example - Mixing "begin-end" and "fork - join"

```
1  module fork_join();
2
3  reg clk,reset,enable,data;
4
5  initial  begin
6    $display ("Starting simulation");
7    $monitor("%g clk=%b reset=%b enable=%b data=%b",
8      $time, clk, reset, enable, data);
9    fork : FORK_VAL
10     #1  clk = 0;
11     #5  reset = 0;
12     #5  enable = 0;
13     #2  data = 0;
14   join
15   #10  $display ("%g Terminating simulation", $time);
16   $finish;
17 end
18
19 endmodule
```

You could download file fork_join.v here

Simulator Output

```
0 clk=x reset=x enable=x data=x
1 clk=0 reset=x enable=x data=x
2 clk=0 reset=x enable=x data=0
5 clk=0 reset=0 enable=0 data=0
15 Terminating simulation
```

Blocking and Nonblocking assignment

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

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;

Note : Correct way to spell 'nonblocking' is 'nonblocking' and not 'non-blocking'.

Example - blocking and nonblocking

```
1  module blocking_nonblocking();
2
```

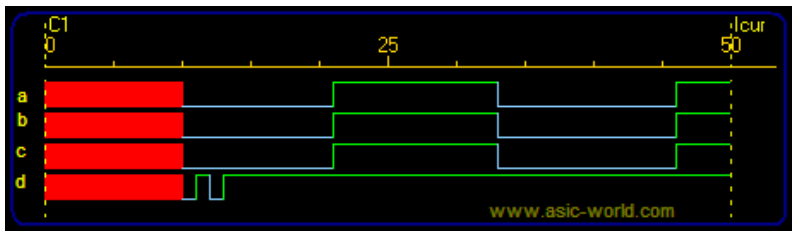
```
 3 reg a,b,c,d;
 4 // Blocking Assignment
 5 initial begin
 6   #10 a = 0;
 7   #11 a = 1;
 8   #12 a = 0;
 9   #13 a = 1;
10 end
11
12 initial begin
13   #10 b <= 0;
14   #11 b <= 1;
15   #12 b <= 0;
16   #13 b <= 1;
17 end
18
19 initial begin
20    c = #10 0;
21    c = #11 1;
22    c = #12 0;
23    c = #13 1;
24 end
25
26 initial begin
27    d <= #10 0;
28    d <= #11 1;
29    d <= #12 0;
30    d <= #13 1;
31 end
32
33 initial begin
34   $monitor("TIME = %g A = %b B = %b C = %b D = %b",$time, a, b, c, d);
35   #50 $finish;
36 end
37
38 endmodule
```

You could download file blocking_nonblocking.v here

Simulator Output

```
TIME = 0 A = x B = x C = x D = x
TIME = 10 A = 0 B = 0 C = 0 D = 0
TIME = 11 A = 0 B = 0 C = 0 D = 1
TIME = 12 A = 0 B = 0 C = 0 D = 0
TIME = 13 A = 0 B = 0 C = 0 D = 1
TIME = 21 A = 1 B = 1 C = 1 D = 1
TIME = 33 A = 0 B = 0 C = 0 D = 1
TIME = 46 A = 1 B = 1 C = 1 D = 1
```

Waveform



assign and deassign

The assign and deassign procedural assignment statements allow continuous assignments to be placed onto registers for controlled periods of time. The assign procedural statement overrides procedural assignments to a **register**. The deassign procedural statement ends a continuous assignment to a register.

Example - assign and deassign

```
module assign_deassign ();

reg clk,rst,d,preset;
wire q;

initial begin
  $monitor("@%g clk %b rst %b preset %b d %b q %b",
    $time, clk, rst, preset, d, q);
  clk = 0;
  rst = 0;
  d  = 0;
  preset = 0;
  #10  rst = 1;
  #10  rst = 0;
  repeat (10) begin
    @ (posedge clk);
    d <= $random;
    @ (negedge clk) ;
    preset <= ~preset;
  end
  #1  $finish;
end
// Clock generator
always  #1  clk = ~clk;

// assign and deassign q of flip flop module
always @(preset)
if (preset) begin
  assign U.q = 1; // assign procedural statement
end else begin
 deassign U.q;     // deassign procedural statement
end

d_ff U (clk,rst,d,q);

endmodule

// D Flip-Flop model
module d_ff (clk,rst,d,q);
input clk,rst,d;
output q;
reg q;

always @ (posedge clk)
if (rst) begin
  q <= 0;
end else begin
  q <= d;
end

endmodule
```

You could download file assign_deassign.v here

Simulator Output

```
@0  clk 0 rst 0 preset 0 d 0 q x
@1  clk 1 rst 0 preset 0 d 0 q 0
@2  clk 0 rst 0 preset 0 d 0 q 0
@3  clk 1 rst 0 preset 0 d 0 q 0
@4  clk 0 rst 0 preset 0 d 0 q 0
@5  clk 1 rst 0 preset 0 d 0 q 0
```

```
@6  clk 0 rst 0 preset 0 d 0 q 0
@7  clk 1 rst 0 preset 0 d 0 q 0
@8  clk 0 rst 0 preset 0 d 0 q 0
@9  clk 1 rst 0 preset 0 d 0 q 0
@10 clk 0 rst 1 preset 0 d 0 q 0
@11 clk 1 rst 1 preset 0 d 0 q 0
@12 clk 0 rst 1 preset 0 d 0 q 0
@13 clk 1 rst 1 preset 0 d 0 q 0
@14 clk 0 rst 1 preset 0 d 0 q 0
@15 clk 1 rst 1 preset 0 d 0 q 0
@16 clk 0 rst 1 preset 0 d 0 q 0
@17 clk 1 rst 1 preset 0 d 0 q 0
@18 clk 0 rst 1 preset 0 d 0 q 0
@19 clk 1 rst 1 preset 0 d 0 q 0
@20 clk 0 rst 0 preset 0 d 0 q 0
@21 clk 1 rst 0 preset 0 d 0 q 0
@22 clk 0 rst 0 preset 1 d 0 q 1
@23 clk 1 rst 0 preset 1 d 1 q 1
@24 clk 0 rst 0 preset 0 d 1 q 1
@25 clk 1 rst 0 preset 0 d 1 q 1
@26 clk 0 rst 0 preset 1 d 1 q 1
@27 clk 1 rst 0 preset 1 d 1 q 1
@28 clk 0 rst 0 preset 0 d 1 q 1
@29 clk 1 rst 0 preset 0 d 1 q 1
@30 clk 0 rst 0 preset 1 d 1 q 1
@31 clk 1 rst 0 preset 1 d 1 q 1
@32 clk 0 rst 0 preset 0 d 1 q 1
@33 clk 1 rst 0 preset 0 d 1 q 1
@34 clk 0 rst 0 preset 1 d 1 q 1
@35 clk 1 rst 0 preset 1 d 0 q 1
@36 clk 0 rst 0 preset 0 d 0 q 1
@37 clk 1 rst 0 preset 0 d 1 q 0
@38 clk 0 rst 0 preset 1 d 1 q 1
@39 clk 1 rst 0 preset 1 d 1 q 1
@40 clk 0 rst 0 preset 0 d 1 q 1
```

force and release

Another form of procedural continuous assignment is provided by the force and release procedural statements. These statements have a similar effect on the assign-deassign pair, but a force can be applied to nets as well as to registers.

One can use force and release while doing gate level simulation to work around reset connectivity problems. Also can be used insert single and double bit errors on data read from memory.

Example - force and release

```
1  module force_release ();
2
3  reg clk,rst,d,preset;
4  wire q;
5
6  initial begin
7    $monitor("@%g clk %b rst %b preset %b d %b q %b",
8      $time, clk, rst, preset, d, q);
9    clk = 0;
10   rst = 0;
11   d  = 0;
12   preset = 0;
13   #10  rst = 1;
14   #10  rst = 0;
15   repeat (10) begin
16     @ (posedge clk);
17     d <= $random;
18     @ (negedge clk) ;
19     preset <= ~preset;
20   end
21   #1  $finish;
```

```
22 end
23 // Clock generator
24 always #1  clk = ~clk;
25
26 // force and release of flip flop module
27 always @(preset)
28 if (preset) begin
29   force U.q = preset; // force procedural statement
30 end else begin
31   release U.q;     // release procedural statement
32 end
33
34 d_ff U (clk,rst,d,q);
35
36 endmodule
37
38 // D Flip-Flop model
39 module d_ff (clk,rst,d,q);
40 input clk,rst,d;
41 output q;
42 wire q;
43 reg q_reg;
44
45 assign q = q_reg;
46
47 always @ (posedge clk)
48 if (rst) begin
49   q_reg <= 0;
50 end else begin
51   q_reg <= d;
52 end
53
54 endmodule
```

You could download file force_release.v here

Simulator Output

```
@0  clk 0 rst 0 preset 0 d 0 q x
@1  clk 1 rst 0 preset 0 d 0 q 0
@2  clk 0 rst 0 preset 0 d 0 q 0
@3  clk 1 rst 0 preset 0 d 0 q 0
@4  clk 0 rst 0 preset 0 d 0 q 0
@5  clk 1 rst 0 preset 0 d 0 q 0
@6  clk 0 rst 0 preset 0 d 0 q 0
@7  clk 1 rst 0 preset 0 d 0 q 0
@8  clk 0 rst 0 preset 0 d 0 q 0
@9  clk 1 rst 0 preset 0 d 0 q 0
@10 clk 0 rst 1 preset 0 d 0 q 0
@11 clk 1 rst 1 preset 0 d 0 q 0
@12 clk 0 rst 1 preset 0 d 0 q 0
@13 clk 1 rst 1 preset 0 d 0 q 0
@14 clk 0 rst 1 preset 0 d 0 q 0
@15 clk 1 rst 1 preset 0 d 0 q 0
@16 clk 0 rst 1 preset 0 d 0 q 0
@17 clk 1 rst 1 preset 0 d 0 q 0
@18 clk 0 rst 1 preset 0 d 0 q 0
@19 clk 1 rst 1 preset 0 d 0 q 0
@20 clk 0 rst 0 preset 0 d 0 q 0
@21 clk 1 rst 0 preset 0 d 0 q 0
@22 clk 0 rst 0 preset 1 d 0 q 1
@23 clk 1 rst 0 preset 1 d 1 q 1
@24 clk 0 rst 0 preset 0 d 1 q 0
@25 clk 1 rst 0 preset 0 d 1 q 1
@26 clk 0 rst 0 preset 1 d 1 q 1
@27 clk 1 rst 0 preset 1 d 1 q 1
@28 clk 0 rst 0 preset 0 d 1 q 1
@29 clk 1 rst 0 preset 0 d 1 q 1
@30 clk 0 rst 0 preset 1 d 1 q 1
@31 clk 1 rst 0 preset 1 d 1 q 1
```

```
@32 clk 0 rst 0 preset 0 d 1 q 1
@33 clk 1 rst 0 preset 0 d 1 q 1
@34 clk 0 rst 0 preset 1 d 1 q 1
@35 clk 1 rst 0 preset 1 d 0 q 1
@36 clk 0 rst 0 preset 0 d 0 q 1
@37 clk 1 rst 0 preset 0 d 1 q 0
@38 clk 0 rst 0 preset 1 d 1 q 1
@39 clk 1 rst 0 preset 1 d 1 q 1
@40 clk 0 rst 0 preset 0 d 1 q 1
```

Do you have any Comment? mail me at:deepak@asic-world.com