Verilog Behavioral Modeling Part-II

Feb-9-2014

The Conditional Statement if-else

The if - else statement controls the execution of other statements. In programming language like c, if - else controls the flow of program. When more than one statement needs to be executed for an if condition, then we need to use begin and end as seen in earlier examples.

Syntax : if

```
if (condition)
statements;
```

Syntax : if-else

```
if (condition)
statements;
else
statements;
```

Syntax : nested if-else-if

```
if (condition)
statements;
else if (condition)
statements;
.....
.....
else
statements;
```

Example- simple if

```
1  module simple_if();
2
3  reg latch;
4  wire enable,din;
5
6  always @ (enable or din)
7  if (enable) begin
8    latch <= din;
9  end
10
11 endmodule
```

You could download file simple_if.v here

Example- if-else

```
1  module if_else();
2
3  reg dff;
4  wire clk,din,reset;
5
6  always @ (posedge clk)
7  if (reset) begin
8    dff <= 0;
9  end else  begin
```

```
10   dff <= din;
11 end
12
13 endmodule
```

You could download file if_else.v here

Example- nested-if-else-if

```
1  module nested_if();
2
3  reg [3:0] counter;
4  reg clk,reset,enable, up_en, down_en;
5
6  always @ (posedge clk)
7  // If reset is asserted
8  if (reset == 1'b0) begin
9     counter <= 4'b0000;
10 // If counter is enable and up count is asserted
11 end else if (enable == 1'b1 && up_en == 1'b1) begin
12    counter <= counter + 1'b1;
13 // If counter is enable and down count is asserted
14 end else if (enable == 1'b1 && down_en == 1'b1) begin
15    counter <= counter - 1'b1;
16 // If counting is disabled
17 end else begin
18    counter <= counter; // Redundant code
19 end
20
21 // Testbench code
22 initial begin
23   $monitor ("@%0dns reset=%b enable=%b up=%b down=%b count=%b",
24             $time, reset, enable, up_en, down_en,counter);
25   $display("@%0dns Driving all inputs to know state",$time);
26   clk = 0;
27   reset = 0;
28   enable = 0;
29   up_en = 0;
30   down_en = 0;
31   #3  reset = 1;
32   $display("@%0dns De-Asserting reset",$time);
33   #4  enable = 1;
34   $display("@%0dns De-Asserting reset",$time);
35   #4  up_en = 1;
36   $display("@%0dns Putting counter in up count mode",$time);
37   #10  up_en = 0;
38   down_en = 1;
39   $display("@%0dns Putting counter in down count mode",$time);
40   #8  $finish;
41 end
42
43 always  #1  clk = ~clk;
44
45 endmodule
```

You could download file nested_if.v here

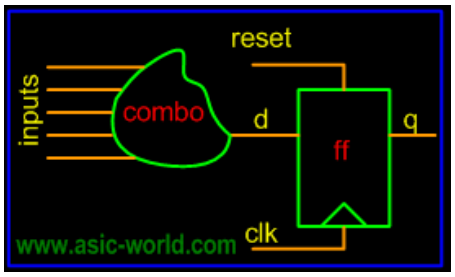
Simulation Log- nested-if-else-if

```
@0ns Driving all inputs to know state
@0ns reset=0 enable=0 up=0 down=0 count=xxxx
@1ns reset=0 enable=0 up=0 down=0 count=0000
@3ns De-Asserting reset
@3ns reset=1 enable=0 up=0 down=0 count=0000
@7ns De-Asserting reset
@7ns reset=1 enable=1 up=0 down=0 count=0000
@11ns Putting counter in up count mode
@11ns reset=1 enable=1 up=1 down=0 count=0001
@13ns reset=1 enable=1 up=1 down=0 count=0010
```

```
@15ns reset=1 enable=1 up=1 down=0 count=0011
@17ns reset=1 enable=1 up=1 down=0 count=0100
@19ns reset=1 enable=1 up=1 down=0 count=0101
@21ns Putting counter in down count mode
@21ns reset=1 enable=1 up=0 down=1 count=0100
@23ns reset=1 enable=1 up=0 down=1 count=0011
@25ns reset=1 enable=1 up=0 down=1 count=0010
@27ns reset=1 enable=1 up=0 down=1 count=0001
```

Parallel if-else

In the above example, the (enable == 1'b1 && up_en == 1'b1) is given highest priority and condition (enable == 1'b1 && down_en == 1'b1) is given lowest priority. We normally don't include reset checking in priority as this does not fall in the combo logic input to the flip-flop as shown in the figure below.



So when we need priority logic, we use nested if-else statements. On the other hand if we don't want to implement priority logic, knowing that only one input is active at a time (i.e. all inputs are mutually exclusive), then we can write the code as shown below.

It's known fact that priority implementation takes more logic to implement than parallel implementation. So if you know the inputs are mutually exclusive, then you can code the logic in parallel if.

```
1  module parallel_if();
2
3  reg [3:0] counter;
4  wire clk,reset,enable, up_en, down_en;
5
6  always @ (posedge clk)
7  // If reset is asserted
8  if (reset == 1'b0) begin
9     counter <= 4'b0000;
10 end else begin
11   // If counter is enable and up count is mode
12   if (enable == 1'b1 && up_en == 1'b1) begin
13     counter <= counter + 1'b1;
14   end
15   // If counter is enable and down count is mode
16   if (enable == 1'b1 && down_en == 1'b1) begin
17     counter <= counter - 1'b1;
18   end
19 end
20
21 endmodule
```

You could download file parallel_if.v here

The Case Statement

The case statement compares an expression to a series of cases and executes the statement or statement group associated with the first matching case:

- case statement supports single or multiple statements.
- Group multiple statements using begin and end keywords.

Syntax of a case statement look as shown below.
case ()
< case1 > : < statement >

< case2 > : < statement >
.....
default : < statement >
endcase

Normal Case

Example- case

```
1  module mux (a,b,c,d,sel,y);
2  input a, b, c, d;
3  input [1:0] sel;
4  output y;
5
6  reg y;
7
8  always @ (a or b or c or d or sel)
9  case (sel)
10   0 : y = a;
11   1 : y = b;
12   2 : y = c;
13   3 : y = d;
14   default : $display("Error in SEL");
15 endcase
16
17 endmodule
```

You could download file mux.v here

Example- case without default

```
1  module mux_without_default (a,b,c,d,sel,y);
2  input a, b, c, d;
3  input [1:0] sel;
4  output y;
5
6  reg y;
7
8  always @ (a or b or c or d or sel)
9  case (sel)
10   0 : y = a;
11   1 : y = b;
12   2 : y = c;
13   3 : y = d;
14   2'bxx,2'bx0,2'bx1,2'b0x,2'b1x,
15   2'bzz,2'bz0,2'bz1,2'b0z,2'b1z : $display("Error in SEL");
16 endcase
17
18 endmodule
```

You could download file mux_without_default.v here

The example above shows how to specify multiple case items as a single case item.

The Verilog case statement does an identity comparison (like the === operator); one can use the case statement to check for logic x and z values as shown in the example below.

Example- case with x and z

```
1  module case_xz(enable);
2  input enable;
3
4  always @ (enable)
5  case(enable)
6    1'bz : $display ("enable is floating");
```

```
  1'bx : $display ("enable is unknown");
  default : $display ("enable is %b",enable);
endcase

endmodule
```

You could download file case_xz.v here

The casez and casex statement

Special versions of the case statement allow the x ad z logic values to be used as "don't care":

- casez : Treats z as don't care.
- casex : Treats x and z as don't care.

Example- casez

```
module casez_example();
reg [3:0] opcode;
reg [1:0] a,b,c;
reg [1:0] out;

always @ (opcode or a or b or c)
casez(opcode)
  4'b1zzx : begin  // Don't care about lower 2:1 bit, bit 0 match with x
              out = a;
              $display("@%0dns 4'b1zzx is selected, opcode %b",$time,opcode);
            end
  4'b01?? : begin
              out = b;  // bit 1:0 is don't care
              $display("@%0dns 4'b01?? is selected, opcode %b",$time,opcode);
            end
  4'b001? : begin   // bit 0 is don't care
              out = c;
              $display("@%0dns 4'b001? is selected, opcode %b",$time,opcode);
            end
  default : begin
              $display("@%0dns default is selected, opcode %b",$time,opcode);
            end
endcase

// Testbench code goes here
always  #2  a = $random;
always  #2  b = $random;
always  #2  c = $random;

initial  begin
  opcode = 0;
  #2  opcode = 4'b101x;
  #2  opcode = 4'b0101;
  #2  opcode = 4'b0010;
  #2  opcode = 4'b0000;
  #2  $finish;
end

endmodule
```

You could download file casez_example.v here

Simulation Output - casez

```
@0ns default is selected, opcode 0000
@2ns 4'b1zzx is selected, opcode 101x
@4ns 4'b01?? is selected, opcode 0101
@6ns 4'b001? is selected, opcode 0010
@8ns default is selected, opcode 0000
```

Example- casex

```
module casex_example();
reg [3:0] opcode;
reg [1:0] a,b,c;
reg [1:0] out;

always @ (opcode or a or b or c)
casex(opcode)
  4'b1zzx : begin  // Don't care  2:0 bits
              out = a;
              $display("@%0dns 4'b1zzx is selected, opcode %b",$time,opcode);
            end
  4'b01?? : begin  // bit 1:0 is don't care
              out = b;
              $display("@%0dns 4'b01?? is selected, opcode %b",$time,opcode);
            end
  4'b001? : begin  // bit 0 is don't care
              out = c;
              $display("@%0dns 4'b001? is selected, opcode %b",$time,opcode);
            end
   default : begin
              $display("@%0dns default is selected, opcode %b",$time,opcode);
            end
endcase

// Testbench code goes here
always  #2  a = $random;
always  #2  b = $random;
always  #2  c = $random;

initial  begin
   opcode = 0;
   #2  opcode = 4'b101x;
   #2  opcode = 4'b0101;
   #2  opcode = 4'b0010;
   #2  opcode = 4'b0000;
   #2  $finish;
end

endmodule
```

You could download file casex_example.v here

Simulation Output - casex

```
@0ns default is selected, opcode 0000
@2ns 4'b1zzx is selected, opcode 101x
@4ns 4'b01?? is selected, opcode 0101
@6ns 4'b001? is selected, opcode 0010
@8ns default is selected, opcode 0000
```

Example- Comparing case, casex, casez

```
module case_compare;

reg sel;

initial  begin
   #1  $display ("\n    Driving 0");
   sel = 0;
   #1  $display ("\n    Driving 1");
   sel = 1;
   #1  $display ("\n    Driving x");
   sel = 1'bx;
   #1  $display ("\n    Driving z");
   sel = 1'bz;
```

```
14    #1  $finish;
15  end
16
17  always @ (sel)
18  case (sel)
19    1'b0 : $display("Normal : Logic 0 on sel");
20    1'b1 : $display("Normal : Logic 1 on sel");
21    1'bx : $display("Normal : Logic x on sel");
22    1'bz : $display("Normal : Logic z on sel");
23  endcase
24
25  always @ (sel)
26  casex (sel)
27    1'b0 : $display("CASEX  : Logic 0 on sel");
28    1'b1 : $display("CASEX  : Logic 1 on sel");
29    1'bx : $display("CASEX  : Logic x on sel");
30    1'bz : $display("CASEX  : Logic z on sel");
31  endcase
32
33  always @ (sel)
34  casez (sel)
35    1'b0 : $display("CASEZ  : Logic 0 on sel");
36    1'b1 : $display("CASEZ  : Logic 1 on sel");
37    1'bx : $display("CASEZ  : Logic x on sel");
38    1'bz : $display("CASEZ  : Logic z on sel");
39  endcase
40
41  endmodule
```

You could download file case_compare.v here

Simulation Output

```
    Driving 0
Normal : Logic 0 on sel
CASEX  : Logic 0 on sel
CASEZ  : Logic 0 on sel

    Driving 1
Normal : Logic 1 on sel
CASEX  : Logic 1 on sel
CASEZ  : Logic 1 on sel

    Driving x
Normal : Logic x on sel
CASEX  : Logic 0 on sel
CASEZ  : Logic x on sel

    Driving z
Normal : Logic z on sel
CASEX  : Logic 0 on sel
CASEZ  : Logic 0 on sel
```

BACK HOME NEXT