Reference Designer

Verilog if in Combinatorial circuit

if statement

The `if` statetement in verilog is very similar to the if statements in other programming languages. We will now write a combinatorial verilog example that make use of if statement. Let us try to design a priority encoder. Our priority encoder has 4 bit inputs - call them x[4], x[3],x[2]. x[1]. The x[4] bit has the highest priorty. The x[1] bit has lowest priority. At any point of time more that one inputs may be high. Our circuit will check which is the bit with highest priority. And then it gives the binary code of the position of the highest output. If none of the inputs is high, then its out put is 0. The table below shows the behavior of the priority encoder.

Table: A priority encoder with 4 Inputs

x[4]	x[3]	x[2]	x[1]	Output z
1	-	-	-	100
0	1	-	-	011
0	0	1	-	010
0	0	0	1	001
0	0	0	0	1

The - in the table means - it does not matter if the input is 0 or 1. When the x[4] input is 1, it has highest priority and irrespective of the values of other bits, we will give the output that corresponds to the binary digit corresponding to 4 in x[4] or 100. Similarly, if the x[4] is zero and the priority of the next bit x[3] is high, then irrespective of the values of x[2] and x[1], we give output corresponding to 3 of x[3] - or 011. We follow the same logic as per the table above.

Let us now write the actual verilog code that implement the priority encoder

```
1. module  priory_encoder
2. (
3. input wire  [4:1] x,
4. output reg  [2:0]  pcode
5. );
6.
7. always  @(x[4], x[3],x[2], x[1])
8.
9. if (x[4] == 1'b1)
10.         pcode = 3'b100;
11. else if (x[3] == 1'b1)
12.         pcode = 3'b011;
13. else if (x[2] == 1'b1)
14.         pcode = 3'b010;
15. else if (x[1] == 1'b1)
16.         pcode = 3'b001;
17. else pcode = 3'b000;
18.
19. endmodule
```

Note that the always statement `always @(x[4], x[3],x[2], x[1])` Could be written as `always @ *` We now suggest that you write a test bench for this code and verify that it works. If you have sifficulty, you can check it with following test bench

```
1.
2. `timescale 1ns / 1ps
3. module stimulus;
4.         reg [4:1] x;
5.         wire [2:0] pcode;
6.         // Instantiate the Unit Under Test (UUT)
7.         priory_encoder uut (
8.                 .x(x),
```

```
9.                  .pcode(pcode)
10.         );
11.
12.         initial begin
13.                 // Initialize Inputs
14. x = 4'b0000;
15.
16.         #20 x = 4'b0001;
17.         #20 x = 4'b0010;
18.         #20 x = 4'b0011;
19.         #20 x = 4'b0100;
20.         #20 x = 4'b0101;
21.         #20 x = 4'b0110;
22.         #20 x = 4'b0111;
23.         #20 x = 4'b1000;
24.         #20 x = 4'b1001;
25.         #20 x = 4'b1010;
26.         #20 x = 4'b1011;
27.         #20 x = 4'b1100;
28.         #20 x = 4'b1101;
29.         #20 x = 4'b1110;
30.         #20 x = 4'b1111;
31.         #40 ;
32.
33.         end
34.
35.                 initial begin
36.                  $monitor("t=%3d x=%4b,pcode=%3b",$time,x,pcode );
37.                  end
38.
39. endmodule
40.
41.
```

Notice the use of the if statement

```
if (x[4] == 1'b1)
        pcode = 3'b100;
else if (x[3] == 1'b1)
        pcode = 3'b011;
```

The general syntax of an if statement is as follows

```
if  [boolean-expr]
begin
[procedural statement]  ;
[procedural statement]  ;
end

else
begin
[procedural statement]  ;
[procedural statement];
end
```

The `boolean-expr` is evaluated and if it is true, the list of the procedural statements between `begin` and `end` is executed. If the `boolean-expr` is false the procedural statements in the `else` block is executed. The `begin` and `end` can be omitted if there is only one procedural statement as in the case of our example. The `else` statement can become `else` statement if we wish to check second condition.

A Binary Decoder Example

We will now present another example that will make use of `if` statement. A Binary decoder is a circuit that has n inputs and 2^n outputs. It asserts one and only one 2^n outputs depending upon the input.

Let us say our binary decoder has 2 inputs x[1] and x[0] and 4 outputs y[3], y[2], y[1], y[0].

We are also making the decoder circuit a bit more complicated by requiring an enable signal. If the enable is 0 (means it is disabled), the output will be 4'b0000.

The table below shows the function of the Binary decoder.

Table: A 2 to 4 binary Decoder with Enable Signal

Enable	x[1]	x[0]	Output y[3:0]

0	-	-	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000

Can you now try to implement the above on your own without looking at the code presented below.

```
1.  // Referencedesigner.com
2.  // Binary Decoder Example - Usage of if
3.  // Verilog Tutorial
4.
5.  module  binary_encoder
6.  (
7.  input wire  [1:0] x,
8.  input wire  enable,
9.  output reg  [3:0] y
10. );
11.
12. always  @(enable, x[1],x[0])
13.
14. if (enable == 1'b0)
15.         y=4'b0000;
16. else if (x == 2'b00)
17.         y = 4'b0001;
18. else if (x == 2'b01)
19.         y = 4'b0010;
20. else if (x == 2'b10)
21.         y = 4'b0100;
22. else if (x == 2'b11)
23.         y = 4'b1000;
24.
25. endmodule
```

Note that the if statement

```
if (enable == 1'b0)
```

Could also be written as

```
if (~enable)
```

We now suggest that you write a test bench for this code and verify that it works. If you have sifficulty, you can check it with following test bench

```
1.
2.  `timescale 1ns / 1ps
3.  module stimulus;
4.          reg [1:0] x;
5.          reg enable;
6.          wire [3:0] y;
7.          // Instantiate the Unit Under Test (UUT)
8.          binary_encoder uut (
9.                  .x(x),
10.                 .enable(enable),
11.                 .y(y)
12.         );
13.
14.         initial begin
15.                 // Initialize Inputs
16. x = 4'b0000;
17. enable = 0;
18.
19.         #20 enable  = 1;
20.         #20 x = 2'b01;
21.         #20 x = 2'b10;
22.         #20 x = 2'b11;
23.         #40 ;
24.
25.         end
26.
27.                 initial begin
28.                  $monitor("t=%3d enable=%1b,x=%2b, y=%4b",$time,enable,x,y );
29.                  end
30.
31. endmodule
32.
33.
```

Exercise

1. Run the above two examples and verify that the output is as expected.

< Previous Next >