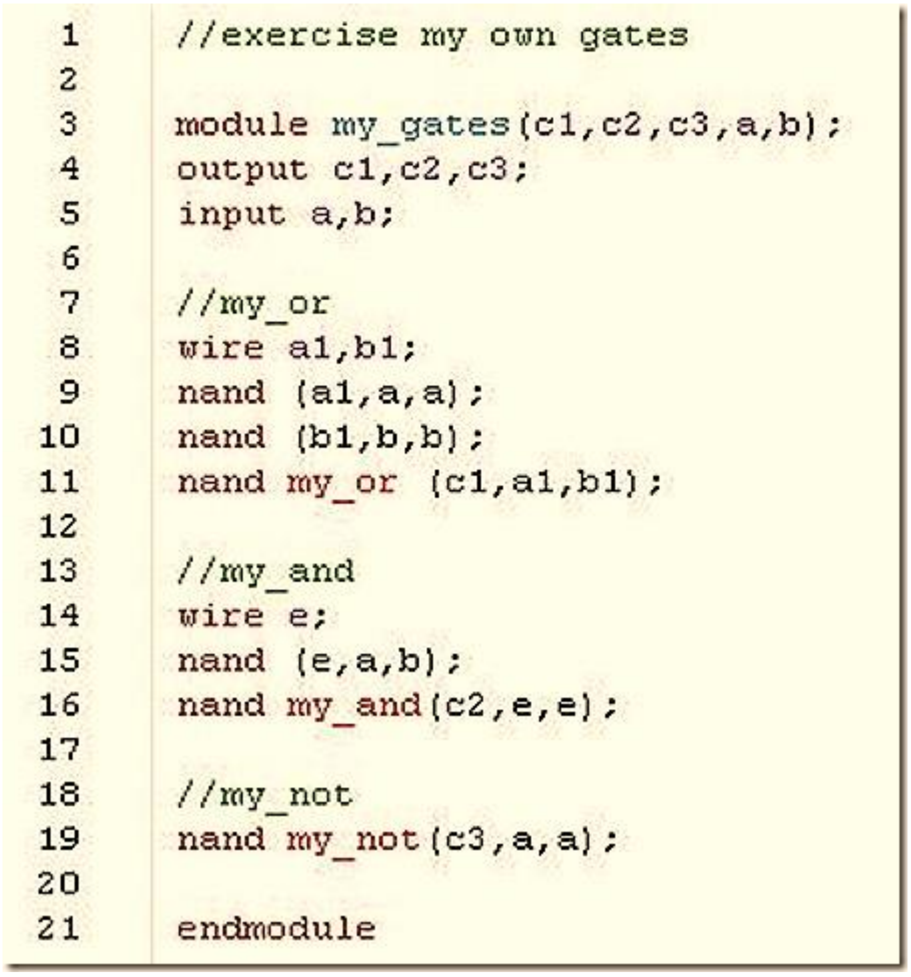
Chapter 5:

Chapter 5. Gate-level Modeling

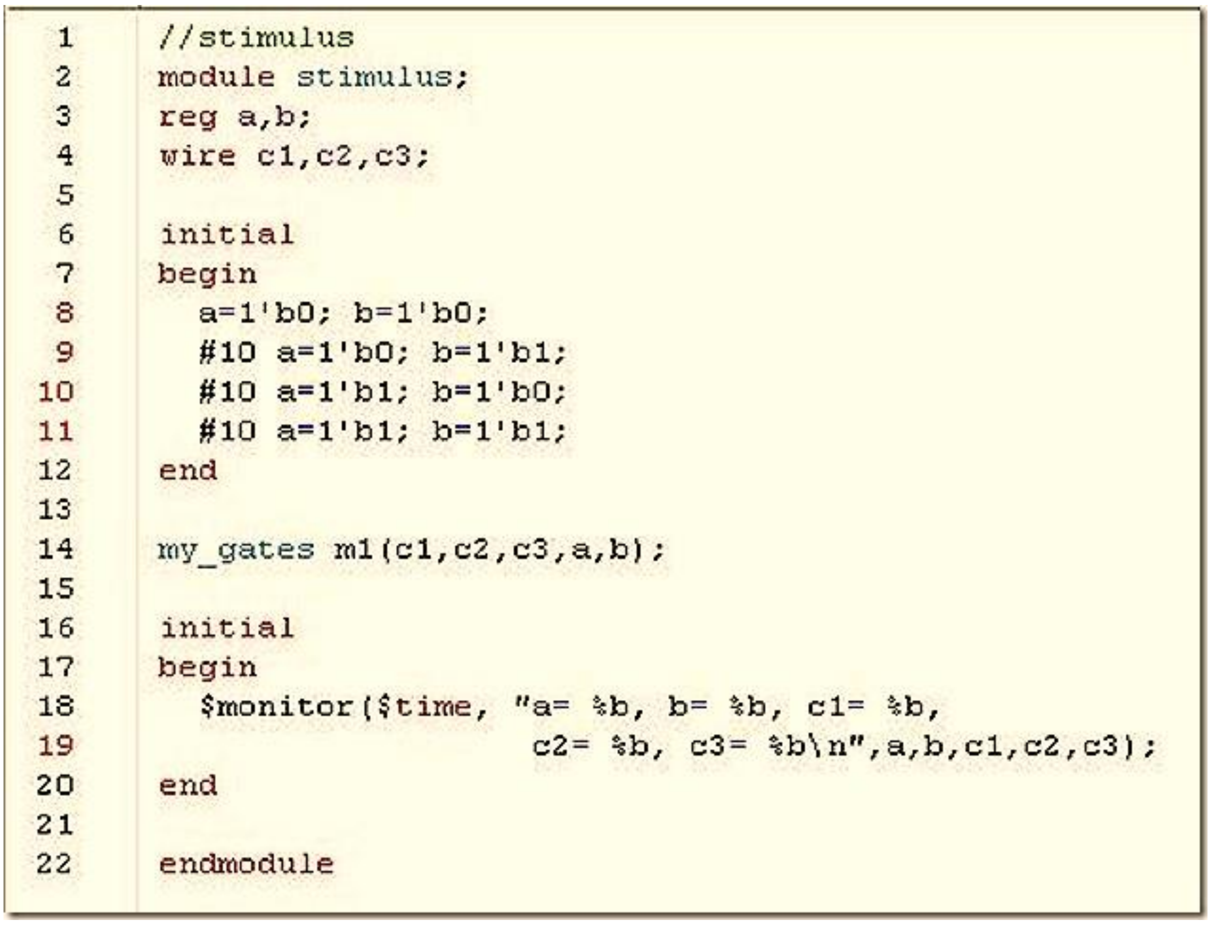
5.4 Exercises

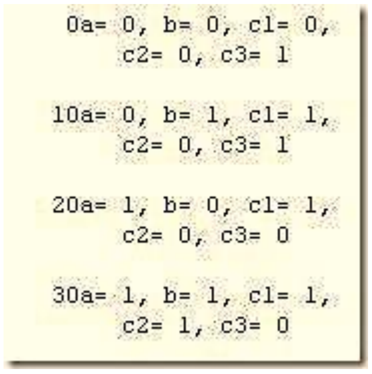
1. Create your own 2-input Verilog gates called my_or, my_and and my_not from 2-input Nand gates. Check the functionality of these gates with a stimulus module.

My answer:



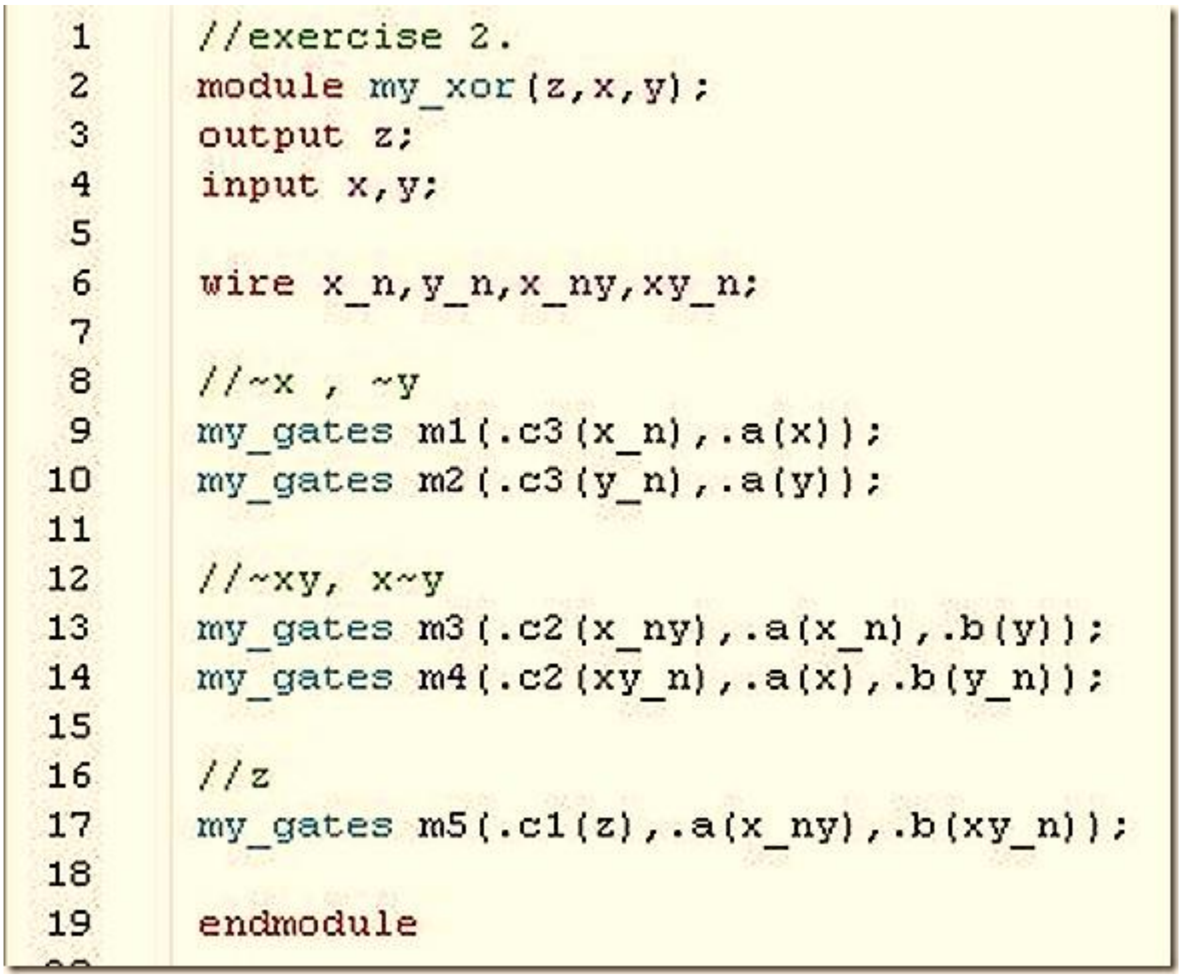
```
//exercise my own gates

module my_gates(c1,c2,c3,a,b);
output c1,c2,c3;
input a,b;

//my_or
wire a1,b1;
nand (a1,a,a);
nand (b1,b,b);
nand my_or (c1,a1,b1);

//my_and
wire e;
nand (e,a,b);
nand my_and(c2,e,e);

//my_not
nand my_not(c3,a,a);

endmodule
```


```verilog
//stimulus
module stimulus;
reg a,b;
wire c1,c2,c3;

initial
begin
  a=1'b0; b=1'b0;
  #10 a=1'b0; b=1'b1;
  #10 a=1'b1; b=1'b0;
  #10 a=1'b1; b=1'b1;
end

my_gates m1(c1,c2,c3,a,b);

initial
begin
  $monitor($time, "a= %b, b= %b, c1= %b,
                  c2= %b, c3= %b\n",a,b,c1,c2,c3);
end

endmodule
```

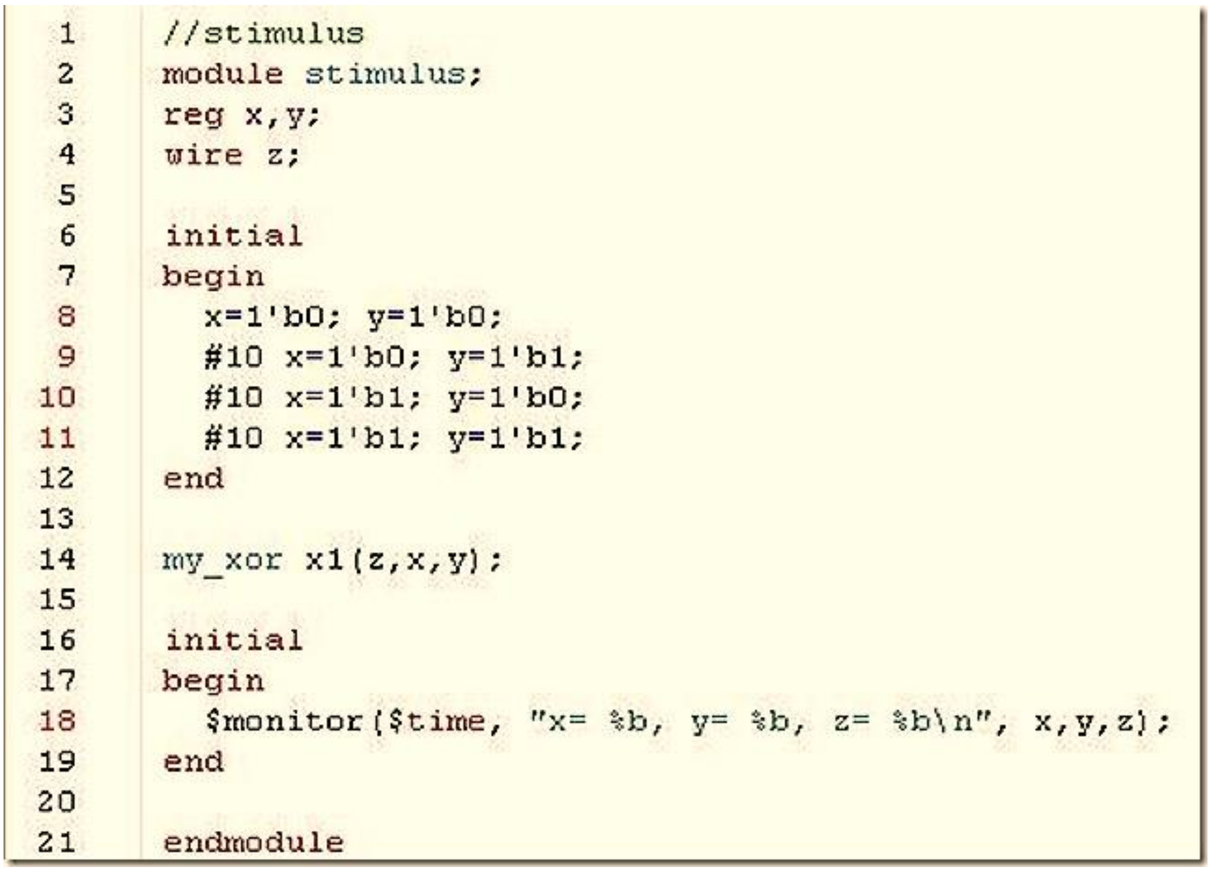

```
 0a= 0, b= 0, c1= 0,
     c2= 0, c3= 1

10a= 0, b= 1, c1= 1,
     c2= 0, c3= 1

20a= 1, b= 0, c1= 1,
     c2= 0, c3= 0

30a= 1, b= 1, c1= 1,
     c2= 1, c3= 0
```

2. A 2-input xor gate can be built from my_and, my_or and my_not gates. Construct an xor module in Verilog that realizes the logic function, z=xy'+x'y. Inputs are x and y, and z is the output. Write a stimulus module that exercises all four combinations of x and y inputs.

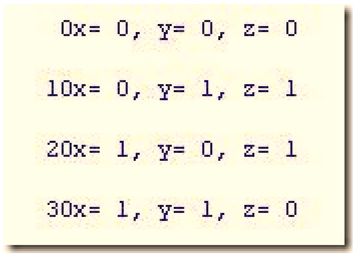
My answer:



```verilog
//exercise 2.
module my_xor(z,x,y);
output z;
input x,y;

wire x_n,y_n,x_ny,xy_n;

//~x , ~y
my_gates m1(.c3(x_n),.a(x));
my_gates m2(.c3(y_n),.a(y));

//~xy, x~y
my_gates m3(.c2(x_ny),.a(x_n),.b(y));
my_gates m4(.c2(xy_n),.a(x),.b(y_n));

//z
my_gates m5(.c1(z),.a(x_ny),.b(xy_n));

endmodule
```



```verilog
//stimulus
module stimulus;
reg x,y;
wire z;

initial
begin
  x=1'b0; y=1'b0;
  #10 x=1'b0; y=1'b1;
  #10 x=1'b1; y=1'b0;
  #10 x=1'b1; y=1'b1;
end

my_xor x1(z,x,y);

initial
begin
  $monitor($time, "x= %b, y= %b, z= %b\n", x,y,z);
end

endmodule
```



```
 0x= 0, y= 0, z= 0

10x= 0, y= 1, z= 1

20x= 1, y= 0, z= 1

30x= 1, y= 1, z= 0
```

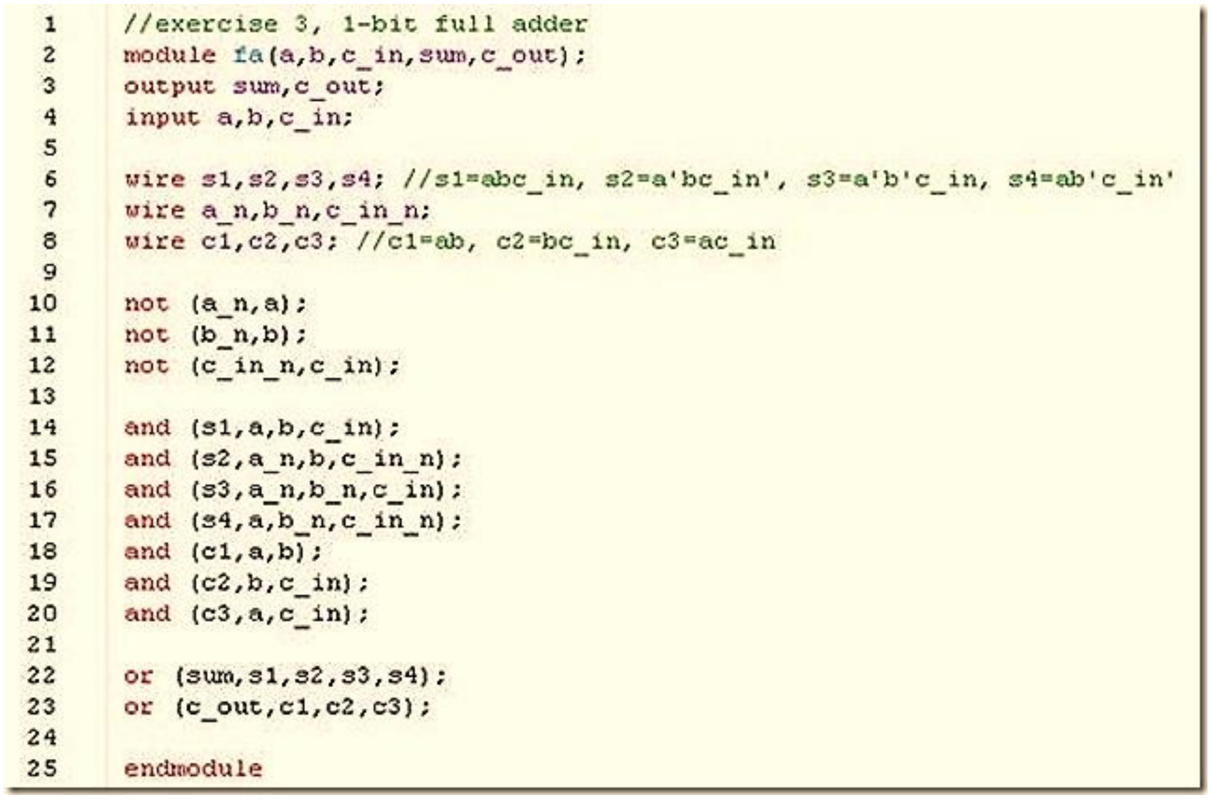
3. The 1-bit full adder described in the chapter can be expressed in a sum of products form.

sum=a.b.c_in+a'.b.c_in'+a'.b'.c_in+a.b'c_in'

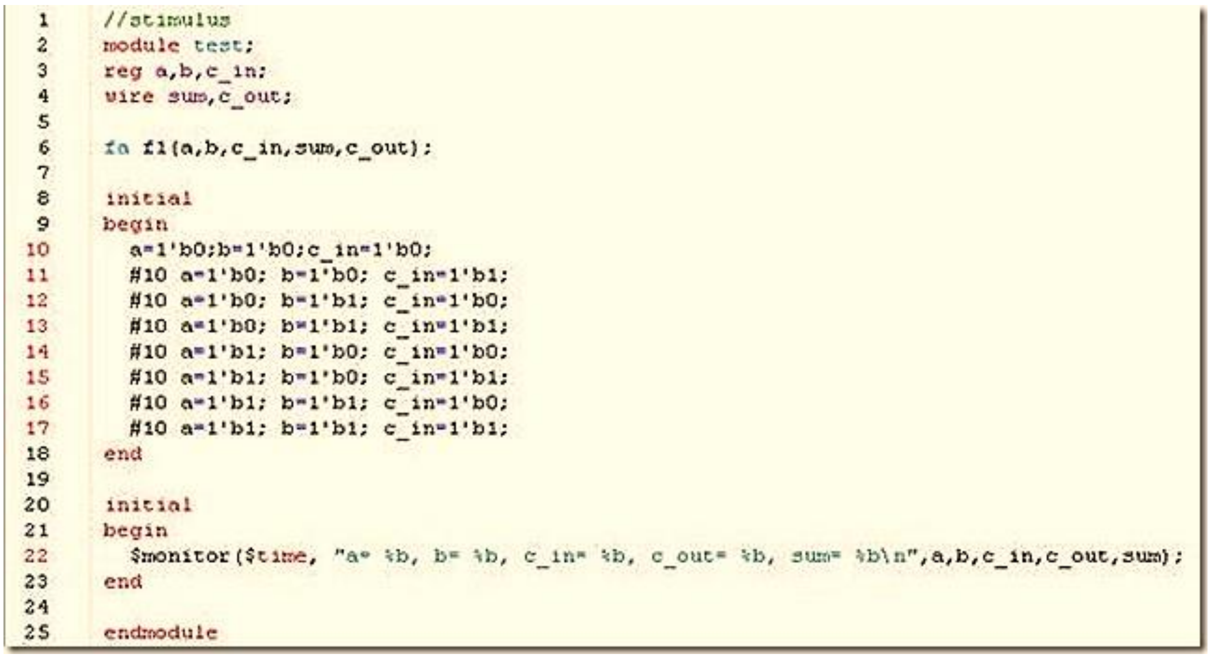
c_out=a.b+b.c_in+a.c_in

Assuming a,b,c_in are the inputs and sum and c_out are the outputs, design a logic circuit to implement the 1-bit full adder, using only and, not ,and or gates. Write the Verilog description for the circuit. You may use up to 4-input Verilog primitive and and or gates. Write the stimulus for the full adder and check the functionality for all input combinations.

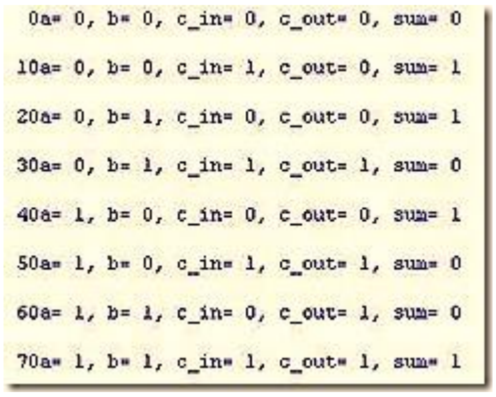
my answer:



```
//exercise 3, 1-bit full adder
module fa(a,b,c_in,sum,c_out);
output sum,c_out;
input a,b,c_in;

wire s1,s2,s3,s4; //s1=abc_in, s2=a'bc_in', s3=a'b'c_in, s4=ab'c_in'
wire a_n,b_n,c_in_n;
wire c1,c2,c3; //c1=ab, c2=bc_in, c3=ac_in

not (a_n,a);
not (b_n,b);
not (c_in_n,c_in);

and (s1,a,b,c_in);
and (s2,a_n,b,c_in_n);
and (s3,a_n,b_n,c_in);
and (s4,a,b_n,c_in_n);
and (c1,a,b);
and (c2,b,c_in);
and (c3,a,c_in);

or (sum,s1,s2,s3,s4);
or (c_out,c1,c2,c3);

endmodule
```

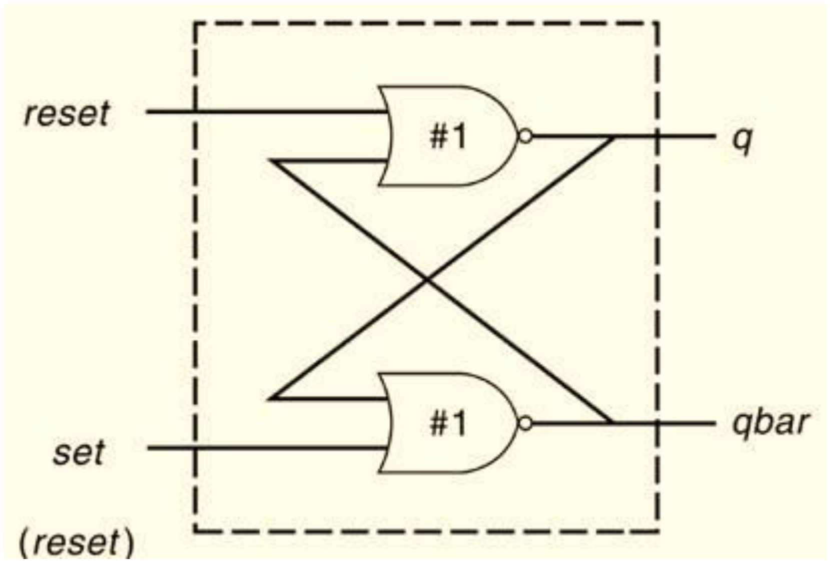


```
//stimulus
module test;
reg a,b,c_in;
wire sum,c_out;

fa f1(a,b,c_in,sum,c_out);

initial
begin
  a=1'b0;b=1'b0;c_in=1'b0;
  #10 a=1'b0; b=1'b0; c_in=1'b1;
  #10 a=1'b0; b=1'b1; c_in=1'b0;
  #10 a=1'b0; b=1'b1; c_in=1'b1;
  #10 a=1'b1; b=1'b0; c_in=1'b0;
  #10 a=1'b1; b=1'b0; c_in=1'b1;
  #10 a=1'b1; b=1'b1; c_in=1'b0;
  #10 a=1'b1; b=1'b1; c_in=1'b1;
end

initial
begin
  $monitor($time, "a= %b, b= %b, c_in= %b, c_out= %b, sum= %b\n",a,b,c_in,c_out,sum);
end

endmodule
```

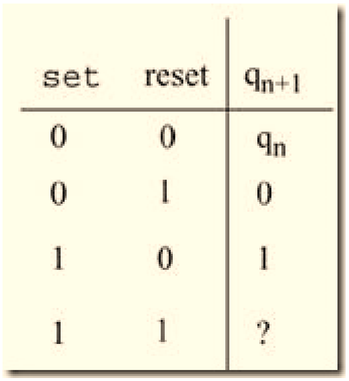


```
 0a= 0, b= 0, c_in= 0, c_out= 0, sum= 0

10a= 0, b= 0, c_in= 1, c_out= 0, sum= 1

20a= 0, b= 1, c_in= 0, c_out= 0, sum= 1

30a= 0, b= 1, c_in= 1, c_out= 1, sum= 0

40a= 1, b= 0, c_in= 0, c_out= 0, sum= 1

50a= 1, b= 0, c_in= 1, c_out= 1, sum= 0

60a= 1, b= 1, c_in= 0, c_out= 1, sum= 0

70a= 1, b= 1, c_in= 1, c_out= 1, sum= 1
```

4. The logic diagram for an RS latch with delay is shown below.

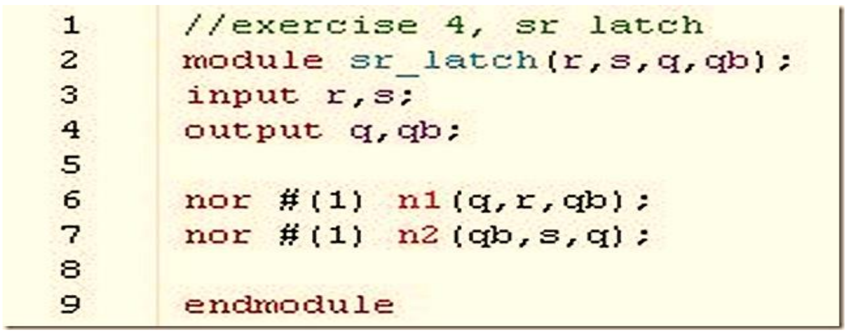


Write the Verilog description for the RS latch. Include delays of 1 unit when instantiating the nor gates. Write the stimulus module for the RS latch, using the following table, and verify the outputs.

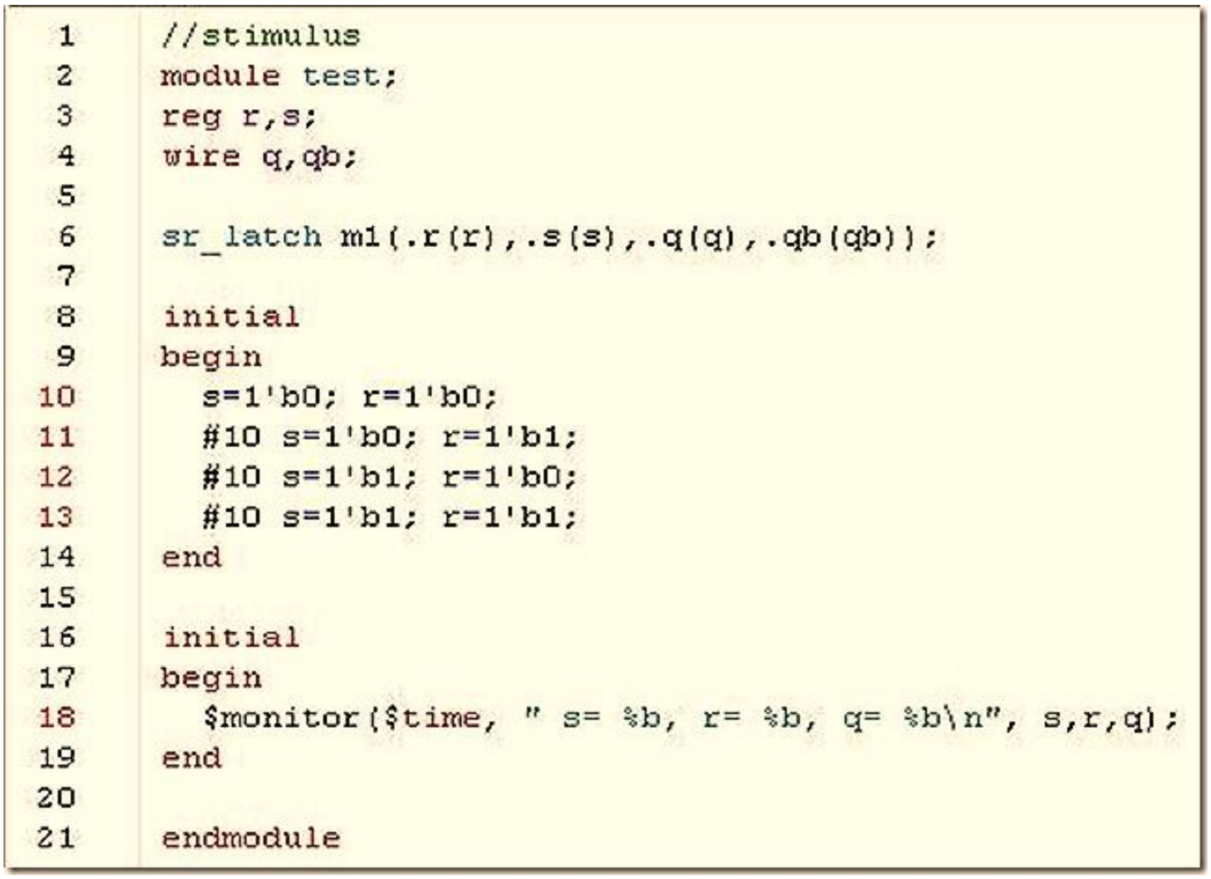


set	reset	q_{n+1}
0	0	q_n
0	1	0
1	0	1
1	1	?

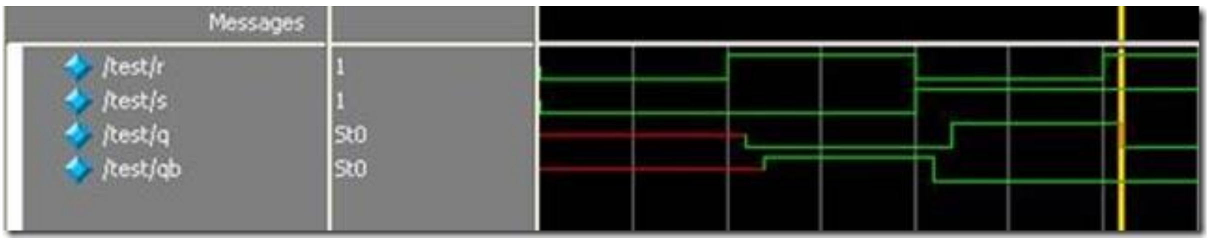
My answer:



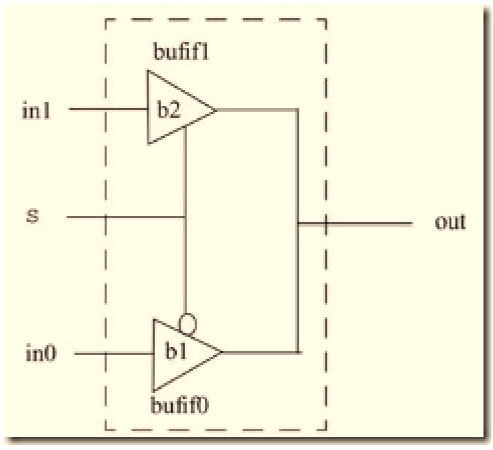
```
//exercise 4, sr latch
module sr_latch(r,s,q,qb);
input r,s;
output q,qb;

nor #(1) n1(q,r,qb);
nor #(1) n2(qb,s,q);

endmodule
```



```
//stimulus
module test;
reg r,s;
wire q,qb;

sr_latch m1(.r(r),.s(s),.q(q),.qb(qb));

initial
begin
  s=1'b0; r=1'b0;
  #10 s=1'b0; r=1'b1;
  #10 s=1'b1; r=1'b0;
  #10 s=1'b1; r=1'b1;
end

initial
begin
  $monitor($time, " s= %b, r= %b, q= %b\n", s,r,q);
end

endmodule
```



5. Design a 2-to-1 multiplexer using bufif0 and bufif1 gates as shown below.

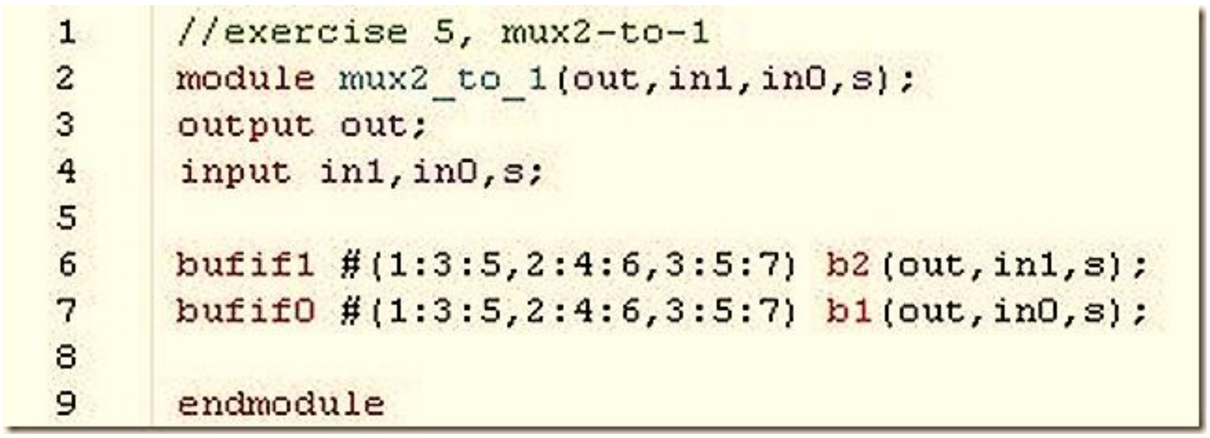


The delay specification for gates b1 and b2 are as follows:

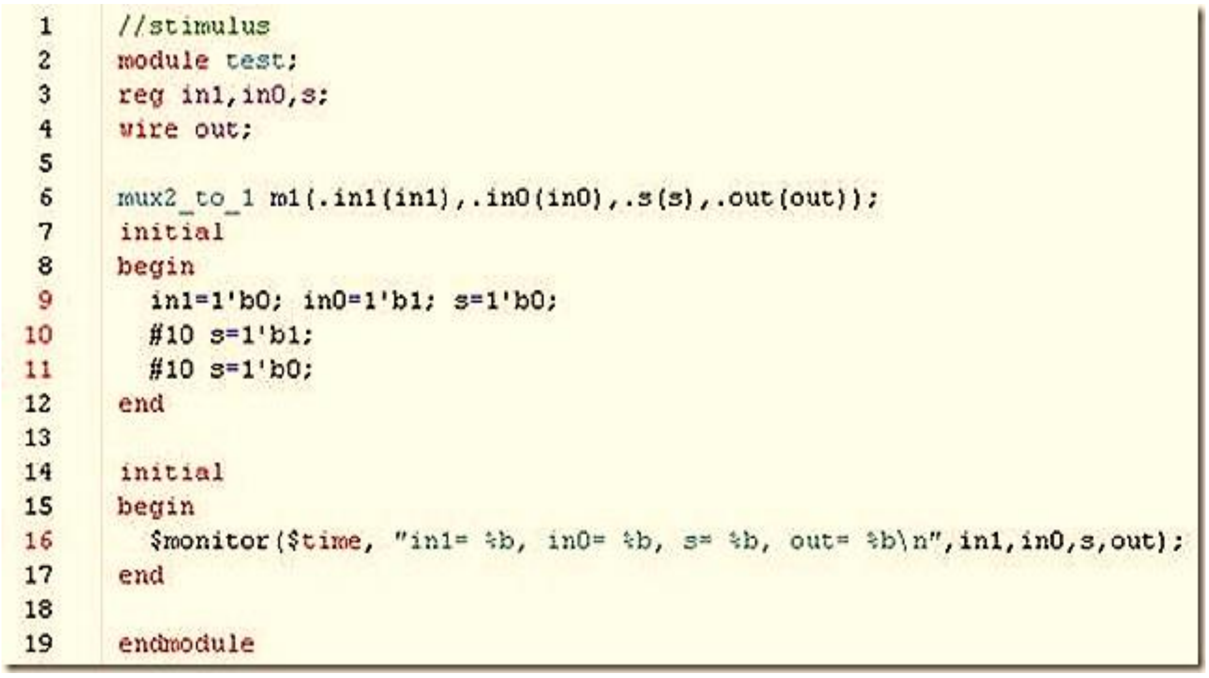
	Min	*Typ*	*Max*
Rise	1	2	3
Fall	3	4	5
Turnoff	5	6	7

Apply stimulus and test the output values.

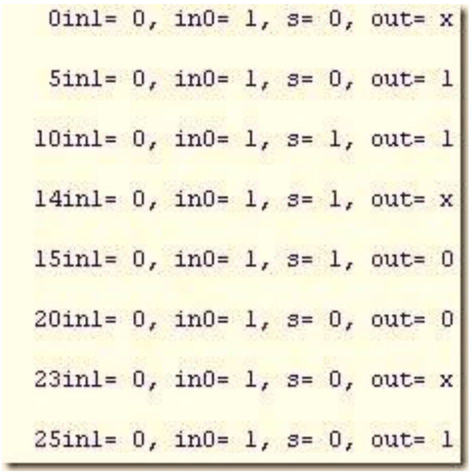
My answer:



```
//exercise 5, mux2-to-1
module mux2_to_1(out,in1,in0,s);
output out;
input in1,in0,s;

bufif1 #(1:3:5,2:4:6,3:5:7) b2(out,in1,s);
bufif0 #(1:3:5,2:4:6,3:5:7) b1(out,in0,s);

endmodule
```



```
//stimulus
module test;
reg in1,in0,s;
wire out;

mux2_to_1 m1(.in1(in1),.in0(in0),.s(s),.out(out));
initial
begin
  in1=1'b0; in0=1'b1; s=1'b0;
  #10 s=1'b1;
  #10 s=1'b0;
end

initial
begin
  $monitor($time, "in1= %b, in0= %b, s= %b, out= %b\n",in1,in0,s,out);
end

endmodule
```



```
  0in1= 0, in0= 1, s= 0, out= x

  5in1= 0, in0= 1, s= 0, out= 1

 10in1= 0, in0= 1, s= 1, out= 1

 14in1= 0, in0= 1, s= 1, out= x

 15in1= 0, in0= 1, s= 1, out= 0

 20in1= 0, in0= 1, s= 0, out= 0

 23in1= 0, in0= 1, s= 0, out= x

 25in1= 0, in0= 1, s= 0, out= 1
```