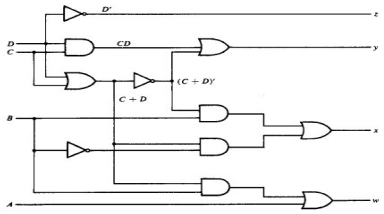
1) Design and simulate BCD to Excess 3 converter using Verilog programming.



$$w = A + BC + BD$$
$$x = B'C + B'D + BC'D'$$
$$y = CD + C'D'$$
$$z = D'$$

BCD(8421)				Excess-3			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

```
module BCEtoE3(a,b,c,d,w,x,y,z);
input a,b,c,d;
output w,x,y,z;
assign w = (a | (b & c) | (b & d));
assign x = (((~b) & c) | ((~b) & d) | (b & (~c) & (~d)));
assign y = ((c & d) | ((~c) & (~d)));
assign z = ~d;
endmodule
```

2) Design and simulate the Boolean function, F= X'Y'Z+X'YZ+XY' using Verilog programming.

```
module boolxyz(x,y,z,f);
input x,y,z;
output f;
assign f = (((~x)&(~y)&z)|((~x)&(y)&z)|((x)&(~y)));
endmodule
```

3) Design and simulate the Boolean function, F= B'D'+B'C'+A'C'D using Verilog programming.

```
module boolbd(a,b,c,d,f);
input a,b,c,d;
output f;
assign f = (((~b)&(~d))|((~b)&(~c))|((~a)&(~c)&(d)));
endmodule
```

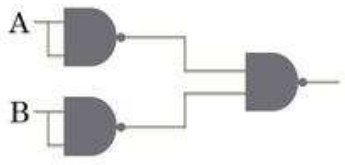
4) Design and simulate the Boolean function, F= (A'+B'). (C'+D'). (B'+D) using Verilog programming.

```
module boolab(a,b,c,d,f);
input a,b,c,d;
output f;
assign f = (((~a)|(~b))&((~c)|(~d))&((~b)|(d)));
endmodule
```

5) Design an OR gate by using a NAND gate and simulate using Verilog programming.

We first compliment the inputs A and B.Using de morgan's law (A'B')' = A + B.

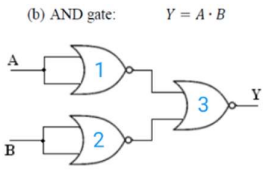
```
module or_nand(a,b,y);
input a,b;
output y;
wire p,q;
nand(p,a,a);
nand(q,b,b);
nand(y,p,q);
endmodule
```



6) Design an AND gate by using a NOR gate and simulate using Verilog programming.

We first compliment the inputs A and B. Then we perform the NOR operation on these complemented inputs. We get (A'+B')'.

Using demorgan's law: (A'+B')' = A.B

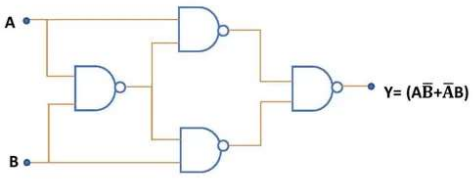


A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

```
module and_nor(a,b,y);
input a,b;
output y;
wire p,q;
nor(p,a,a);
nor(q,b,b);
nor(y,p,q);
endmodule
```

7) Design an XOR gate by using a NAND gate and simulate using Verilog programming.

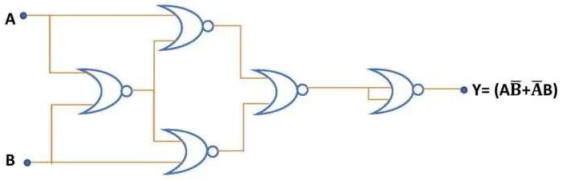
1. P = A nand B = (AB)'
2. Q = A nand (AB)' = (A(AB)')'
3. R = B nand (AB)' = (B(AB)')'
4. Y = (A(AB)')' nand (B(AB)')' = ((A(AB)'). (B(AB)')')'




```
module xor_nand(a,b,y);
input a,b;
output y;
wire p,q,r;
nand(p,a,b);
nand(q,a,p);
nand(r,b,p);
nand(y,q,r);
endmodule
```

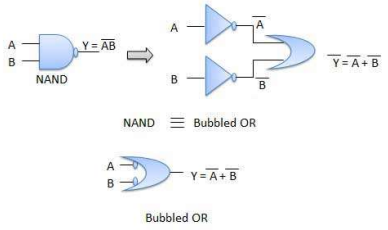
8) Design an XOR gate by using a NOR gate and simulate using Verilog programming.

1. P = A nor B = (A+B)'
2. Q = A nor (A+B)' = (A+(A+B)')'
3. R = B nor (A+B)' = (B+(A+B)')'
4. S = Q nor R = ((A+(A+B)')'+(B+(A+B)')')' = ((A'.B)+(A.B'))'
5. Y = S nor S = (A'.B)+(A.B')




```
module xor_nor(a,b,y);
input a,b;
output y;
wire p,q,r,s;
nor(p,a,b);
nor(q,a,p);
nor(r,b,p);
nor(s,q,r);
nor(y,s,s);
endmodule
```

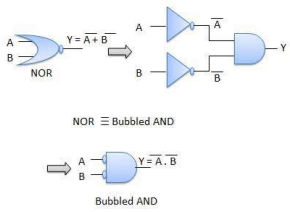
9) Design a circuit to prove DeMorgan's first theorem and simulate using Verilog programming.



A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

```
module demorgan_one(a,b,y_nand,y_or);
input a,b;
output y_nand,y_or;
nand(y_nand,a,b);
wire p,q;
not(p,a);
not(q,b);
or(y_or,p,q);
endmodule
```

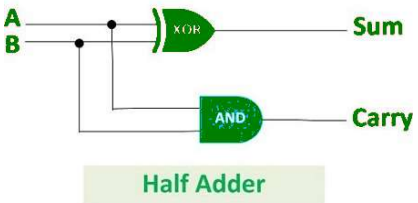
10) Design a circuit to prove DeMorgan's second theorem and simulate using Verilog programming.



A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

```
module demorgan_two(a,b,y_nor,y_and);
input a,b;
output y_nor,y_and;
nor(y_nor,a,b);
wire p,q;
not(p,a);
not(q,b);
and(y_and,p,q);
endmodule
```

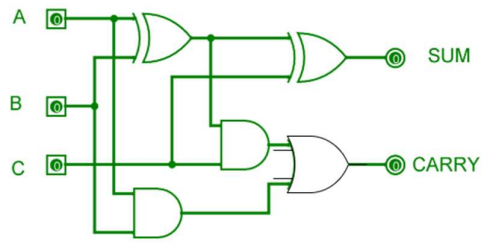
11) Design a combinational circuit to perform addition of two bits and simulate using Verilog programming.



INPUTS		OUTPUTS	
a	b	Sum (a⊕b)	Carry (a.b)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
module HalfAdder(a,b,sum,carry);
input a,b;
output sum,carry;
xor(sum,a,b);
and(carry,a,b);
endmodule
```

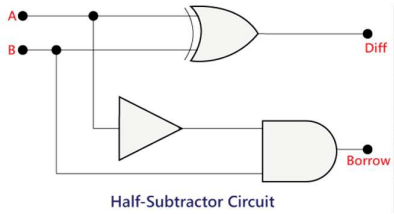
12) Design a combinational circuit to perform addition of three bits and simulate using Verilog programming.



INPUTS			OUTPUTS	
a	b	c	Sum (a⊕b⊕c)	Carry (a.b)+c(a⊕b)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
module FullAdder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
assign sum = ((a^b)^c);
assign carry = ((a&b)|(b&c)|(c&a));
endmodule
```

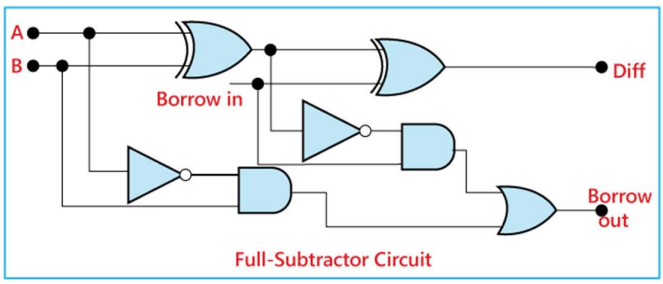
13) Design a combinational circuit to perform subtraction of two bits and simulate using Verilog programming.



INPUTS		OUTPUTS	
A	B	Difference = (A⊕B)	Borrow = A'B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

```
module HalfSubtractor(A,B,Diff,Borrow);
input A,B;
output Diff,Borrow;
wire x;
xor (Diff, A,B);
not(x,A);
and(Borrow,x,B);
endmodule
```

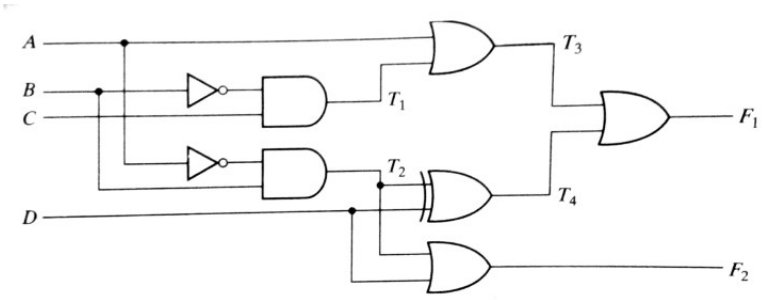
14) Design a combinational circuit to perform subtraction of three bits and simulate using Verilog programming.



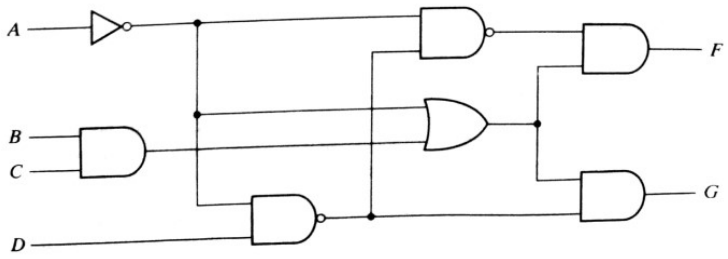
INPUTS			OUTPUTS	
A	B	C	Difference (A⊕B⊕C)	Borrow = A'B+A'C+BC
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

```
module FullSubtractor(A,B,C,Diff,Borrow);
input A,B,C;
output Diff,Borrow;
wire p;
assign Diff = ((A^B)^C);
not(p,A);
assign Borrow = ((p&B)|(p&C)|(B&C));
endmodule
```

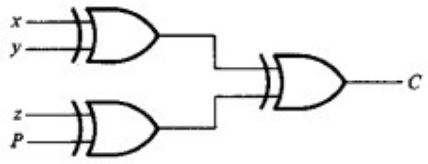
15) Obtain the Boolean expression for F1, F2 and simulate the output using Verilog programming.



16) Obtain the Boolean expression for F, G and simulate the output using Verilog programming.



17) Design and simulate 4-bit parity generator using Verilog programming.



4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

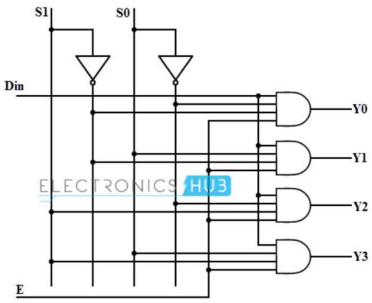
```
module parity(x,y,z,p,c);
input x,y,z,p;
output c;
wire a,b;
xor(a,x,y);
xor(b,z,p);
xor(c,a,b);
endmodule
```

18) Implement the function F (A, B, C) =∑m (1,3,5,6) by 4:1 multiplexer and simulate using Verilog programming.

19) Implement the function F (p, q, r, s) = ∑m (0,1,3,4,8,9,15) by 8:1 multiplexer and simulate using Verilog programming.

20) Design a 1:4 De-multiplexer and simulate using Verilog programming.

S1	S0	D	Y3	Y2	Y1	Y0
0	0	0	0	0	0	**0**
0	0	1	0	0	0	**1**
0	1	0	0	0	**0**	0
0	1	1	0	0	**1**	0
1	0	0	0	**0**	0	0
1	0	1	0	**1**	0	0
1	1	0	**0**	0	0	0
1	1	1	**1**	0	0	0



$Y0 = \overline{S1}\ \overline{S0}\ D$

$Y1 = \overline{S1}\ S0\ D$

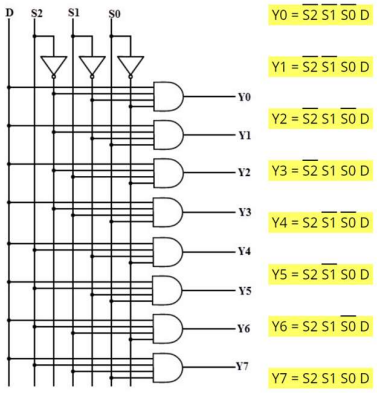
$Y2 = S1\ \overline{S0}\ D$

$Y3 = S1\ S0\ D$

```
module demux(I,S0,S1,Y0,Y1,Y2,Y3);
input I,S0,S1;
output Y0,Y1,Y2,Y3;
wire S0C,S1C;
not(S0C,S0);
not(S1C,S1);
and(Y0,I,S0C,S1C);
and(Y1,I,S0C,S1);
and(Y2,I,S0,S1C);
and(Y3,I,S0,S1);
endmodule
```

21) Design a 1:8 De-multiplexers and simulate using Verilog programming.

S2	S1	S0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	**D**
0	0	1	0	0	0	0	0	0	**D**	0
0	1	0	0	0	0	0	0	**D**	0	0
0	1	1	0	0	0	0	**D**	0	0	0
1	0	0	0	0	0	**D**	0	0	0	0
1	0	1	0	0	**D**	0	0	0	0	0
1	1	0	0	**D**	0	0	0	0	0	0
1	1	1	**D**	0	0	0	0	0	0	0



$Y0 = \overline{S2}\ \overline{S1}\ \overline{S0}\ D$

$Y1 = \overline{S2}\ \overline{S1}\ S0\ D$

$Y2 = \overline{S2}\ S1\ \overline{S0}\ D$

$Y3 = \overline{S2}\ S1\ S0\ D$

$Y4 = S2\ \overline{S1}\ \overline{S0}\ D$

$Y5 = S2\ \overline{S1}\ S0\ D$

$Y6 = S2\ S1\ \overline{S0}\ D$

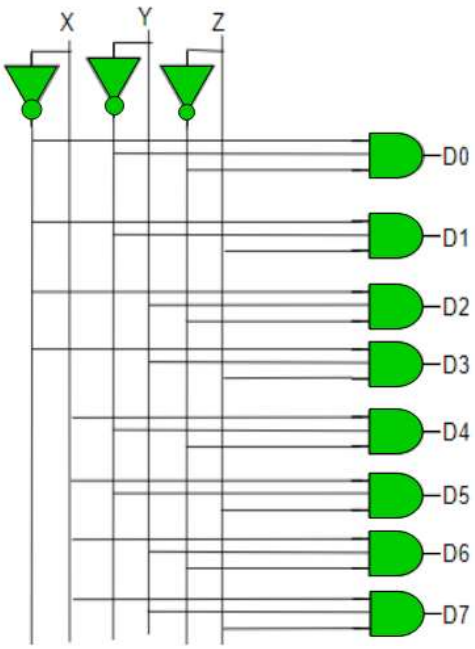
$Y7 = S2\ S1\ S0\ D$

```
module demul_eight(D,S0,S1,S2,Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7);
input D,S0,S1,S2;
output Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
wire S0C,S1C,S2C;
not(S0C,S0);
not(S1C,S1);
not(S2C,S2);
and(Y0,I,S2C,S1C,S0C);
and(Y1,I,S2C,S1C,S0);
and(Y2,I,S2C,S1,S0C);
and(Y3,I,S2C,S1,S0);
and(Y4,I,S2,S1C,S0C);
and(Y5,I,S2,S1C,S0);
and(Y6,I,S2,S1,S0C);
and(Y7,I,S2,S1,S0);
endmodule
```

22) Design an 8:3 Decoder and simulate using Verilog programming.

Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

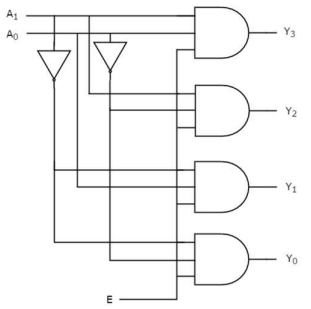
$$D_0 = \bar{A}\bar{B}\bar{C}, \quad D_1 = \bar{A}\bar{B}C, \quad D_2 = \bar{A}B\bar{C},$$
$$D_3 = \bar{A}BC, \quad D_4 = A\bar{B}\bar{C}, \quad D_5 = A\bar{B}C,$$
$$D_6 = AB\bar{C}, \quad D_7 = ABC$$




```
module decode(a,b,c,d0,d1,d2,d3,d4,d5,d6,d7);
input a,b,c;
output d0,d1,d2,d3,d4,d5,d6,d7;
assign d0 = (~a&~b&~c);
assign d1 = (~a&~b&c);
assign d2 = (~a&b&~c);
assign d3 = (~a&b&c);
assign d4 = (a&~b&~c);
assign d5 = (a&~b&c);
assign d6 = (a&b&~c);
assign d7 = (a&b&c);
endmodule
```

23) Design a 4:2 Decoder and simulate using Verilog programming.

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



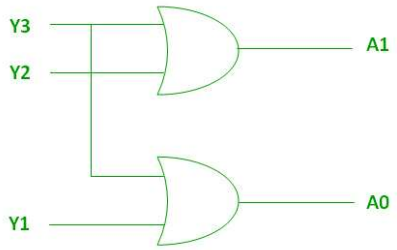
$$Y_3 = E.A_1.A_0$$
$$Y_2 = E.A_1.A_0'$$
$$Y_1 = E.A_1'.A_0$$
$$Y_0 = E.A_1'.A_0'$$

```
module decode(a,b,d0,d1,d2,d3);
input a,b;
output d0,d1,d2,d3;
assign d0 = (~a&~b);
assign d1 = (~a&b);
assign d2 = (a&~b);
assign d3 = (a&b);
endmodule
```

24) Design a 2:4 Encoder and simulate using Verilog programming.

INPUTS				OUTPUTS	
Y3	Y2	Y1	Y0	A1	A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

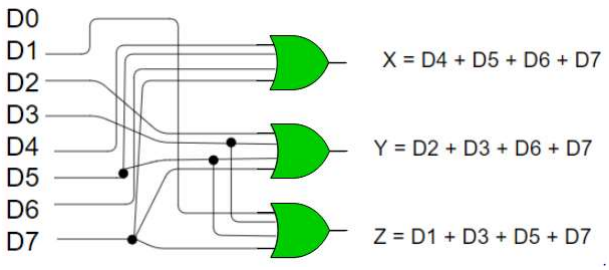
```
A1 = Y3 + Y2
A0 = Y3 + Y1
```




```
module encod(d0,d1,d2,d3,a,b);
input d0,d1,d2,d3;
output a,b;
or(a,d3,d2);
or(b,d3,d1);
endmodule
```

25) Design a 3:8 Encoder and simulate using Verilog programming.

INPUTS								OUTPUTS		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



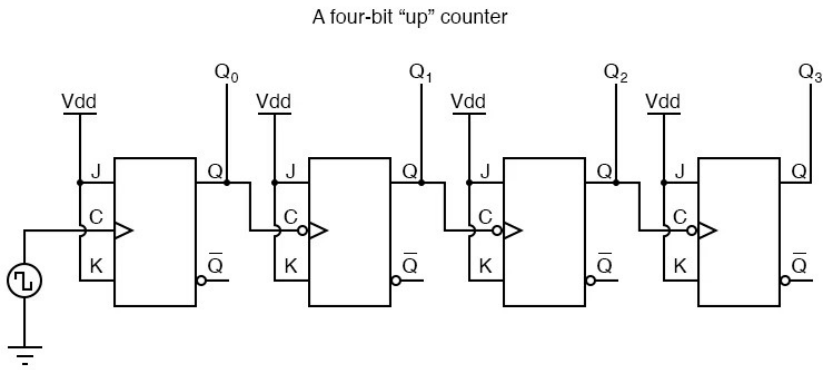

```
module encod(d0,d1,d2,d3,d4,d5,d6,d7,a,b,c);
input d0,d1,d2,d3,d4,d5,d6,d7;
output a,b,c;
or(a,d4,d5,d6,d7);
or(b,d2,d3,d6,d7);
or(c,d1,d3,d5,d7);
endmodule
```

26) Design a mod 5 counter and simulate using Verilog programming.

```
module sr(clock,up);
input clock;
output reg[0:3]up;
always @(posedge clock)
begin
up=up+1;
if(up==5)
up=0;
end
endmodule
```

27) Design a Decade counter and simulate using Verilog programming.

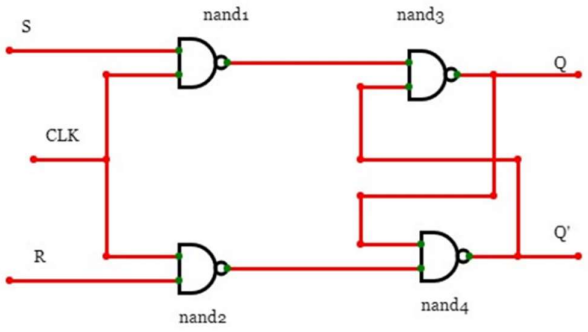
```
module sr(clock,up);
input clock;
output reg[0:3]up;
always @(posedge clock)
begin
up=up+1;
if(up==10)
up=0;
end
endmodule
```



28) Design a clocked flip flop for the given truth table and simulate using Verilog programming. – SR Flip Flop

Input		Output		State
0	0	1	1	Invalid
0	1	0	1	Q' is set as 1
1	0	1	0	Q is set as 1
1	1	1	0	No change

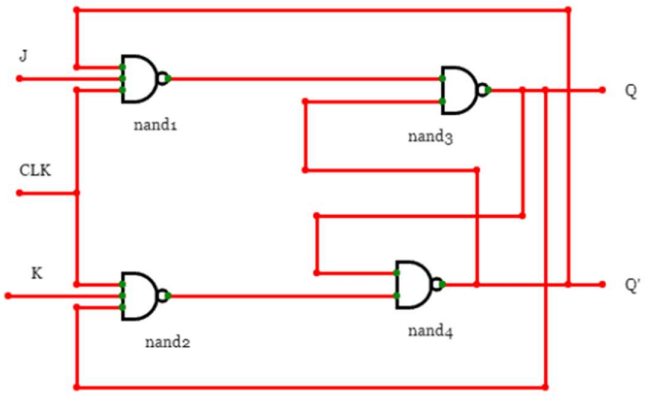
```
module sr(s,r,clock,q,qbar);
input s,r,clock;
output q,qbar;
wire x,y;
nand(x,s,clock);
nand(y,r,clock);
nand(q,x,qbar);
nand(qbar,y,q);
endmodule
```



29) Design a clocked flip flop for the given truth table and simulate using Verilog programming. – JK Flip Flop

Input		Output		State
0	0	1	1	No change
0	0	1	1	No change
0	1	0	1	Q' is set as 1
1	0	1	0	Q is set as 1
1	1	1	0	Toggle
1	1	0	1	Toggle

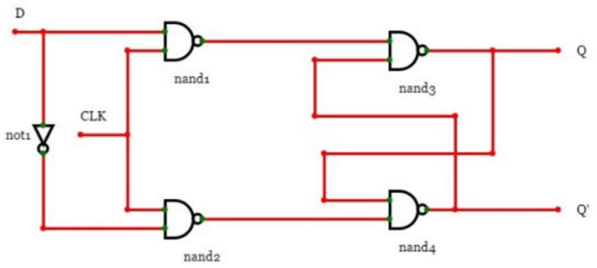
```
module jkflp(J,K,Clock,Q,Qbar);
input J,Clock,K;
output Q,Qbar;
wire S,R;
nand (S,J,Clock,Qbar);
nand (R,K,Clock,Q);
nand (Q,S,Qbar);
nand (Qbar,R,Q);
endmodule
```



30) Design a clocked flip flop for the given truth table and simulate using Verilog programming. – D Flip Flop

Clk	Input	Output		State
0	0	1	1	No change
0	1	1	1	No change
1	0	0	1	Q' is set as 1
1	1	1	0	Q is set as 1

```
module dflp(D,Clock,Q,Qbar);
input D,Clock;
output Q,Qbar;
assign Dbar = ~D;
wire X,Y;
nand (X,D,Clock);
nand (Y,Dbar,Clock);
nand (Q,X,Qbar);
nand (Qbar,Y,Q);
endmodule
```



31) Design a clocked flip flop for the given truth table and simulate using Verilog programming. – T Flip Flop

Clk	Input	Output		State
0	0	1	1	No change
0	1	1	1	No change
1	0	1	1	No change
1	1	1	0	Complement

```
module tflp(T,Clock,Q,Qbar);
input T,Clock;
output Q,Qbar;
wire A,B;
nand (A,T,Clock,Qbar);
nand (B,T,Clock,Q);
nand (Q,A,Qbar);
nand (Qbar,B,Q);
endmodule
```