Assignment 1

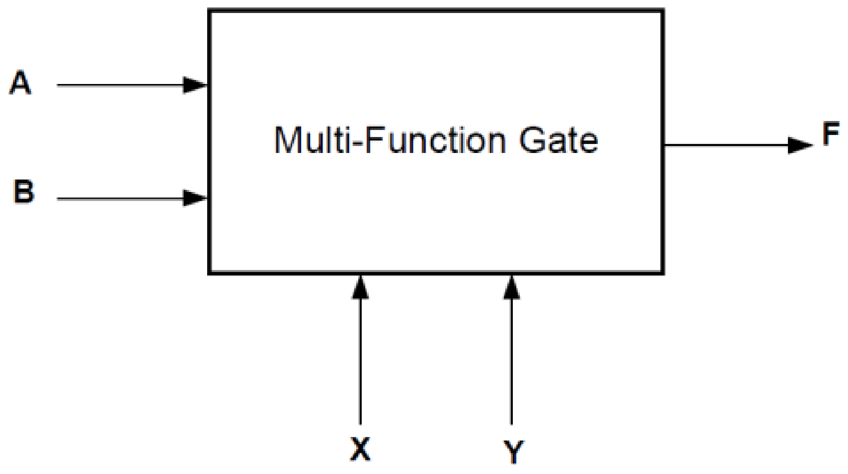
Submitted by: Adwaith K J (B190245EC), adwaith_b190245ec@nitc.ac.in

- **Questions**

1) Write the structural level Verilog code of the following:
    a) 2x1 Mux
    b) 4x1 Mux
    c) 4x1 Mux using 2x1 Mux
    d) 8x1 Mux using 4x1 and 2x1 Mux
    e) Half Adder
    f) Full Adder using Half Adders
    g) 4-bit Ripple Carry Adder using Full Adders

2) Write data flow level Verilog code using conditional operator for the following:
    a) 2x1 Mux
    b) 4x1 Mux
    c) 2 to 4 Decoder
    d) 4 to 2 Encoder

3) Design a Multi-function gate which can work as a two input (A, B) one output (F) logic gate based on the control values placed on two other inputs X and Y. Control input values and the corresponding function is given in the table below. After obtaining the schematic, write structural level verilog code for it.



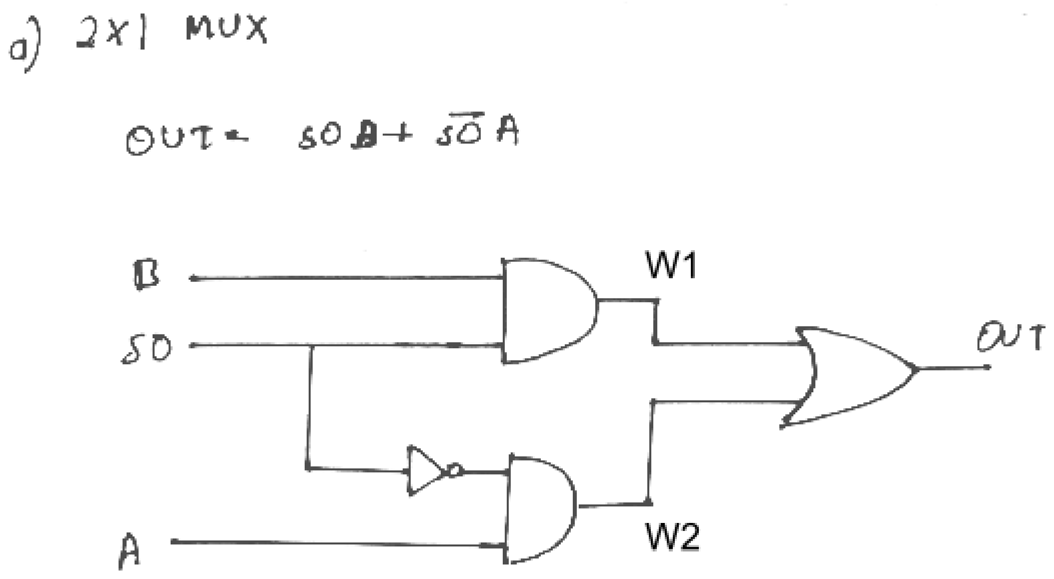
X	Y	Function
0	0	AND
0	1	OR
1	0	NOR
1	1	NAND

4) Write behavioral level Verilog code for the following:
   a) T Flip flop
   b) 4-bit up-down counter
   c) 8-bit shift register which is capable of doing right shift and left shift
   d) 4-bit Johnson counter

Answers

1) Gate level code
   a) 2x1MUX

Gate Level Circuit



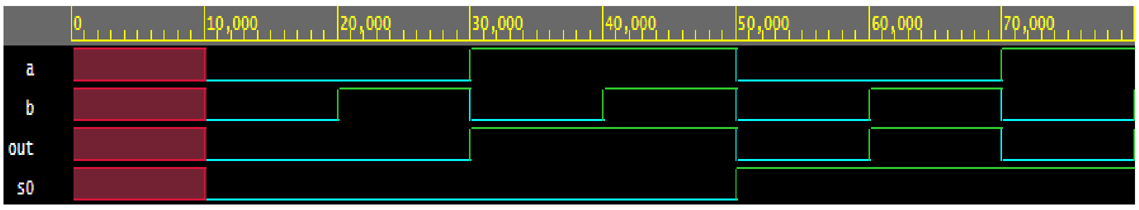
Code

```
module q2x1_MUX(input s0,a,b, output out);
wire w1,w2,inv_s0;
not a1(inv_s0,s0);
and a2(w1,inv_s0,a);
and a3(w2,s0,b);
or a4(out,w1,w2);
endmodule
```

Testbench

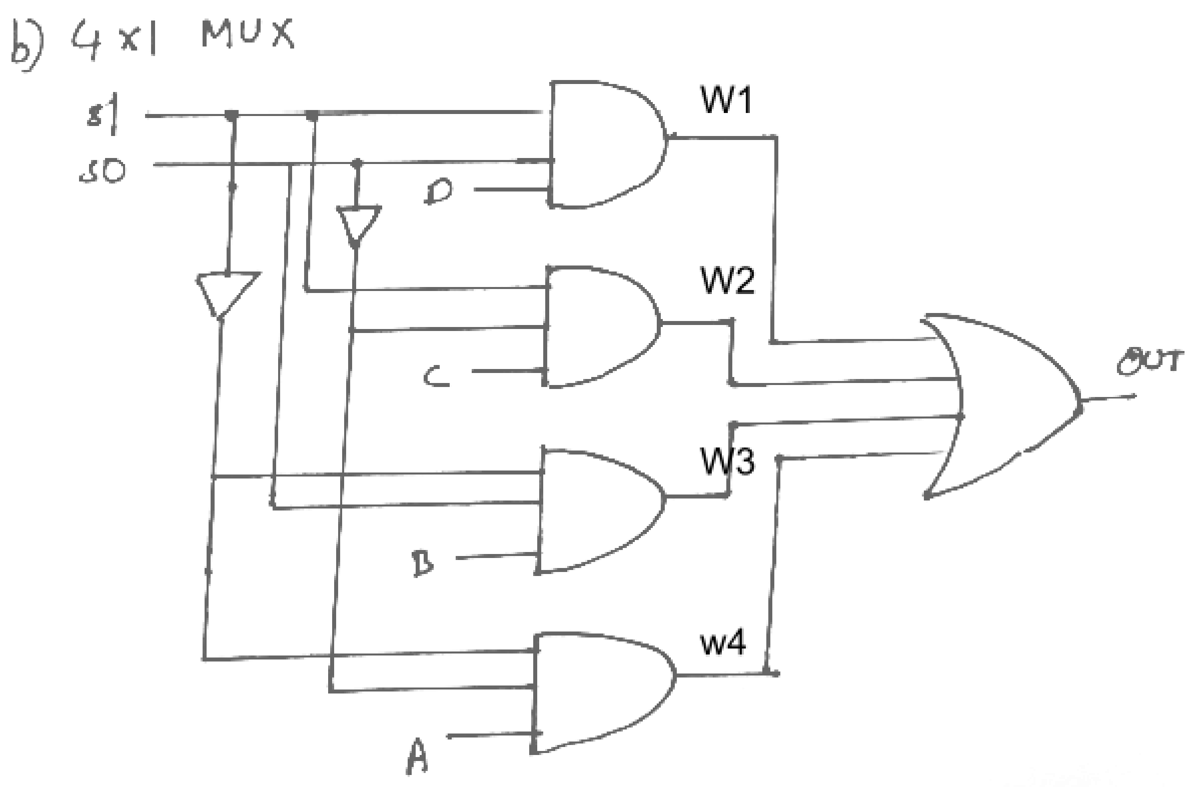
```
`timescale  1ns/10ps

module mux_tb;

reg s0,a,b;
wire out;

q2x1_MUX DUT(s0,a,b,out);

initial
begin



   #10 s0=0;a=0;b=0;
   #10 s0=0;a=0;b=1;
   #10 s0=0;a=1;b=0;
   #10 s0=0;a=1;b=1;
   #10 s0=1;a=0;b=0;
   #10 s0=1;a=0;b=1;
   #10 s0=1;a=1;b=0;
   #10 s0=1;a=1;b=1;

end

initial begin
      $monitor("%g s0=%b a=%b b=%b ",$time,s0,a,b,out);

      $dumpfile("q_1a.vcd");
      $dumpvars;
end
endmodule
```

Waveform



b) 4x1MUX

Gate Level Ckt



Code

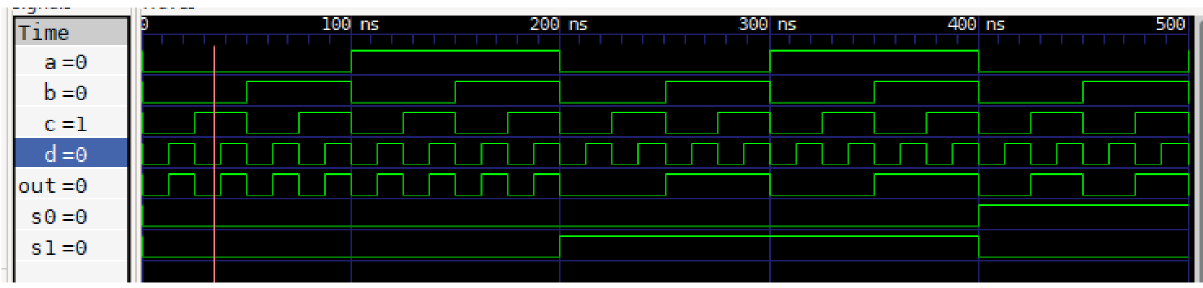
```
module mux_4x1(input s0,s1,a,b,c,d,output out);

wire inv_s0,inv_s1,w1,w2,w3,w4;

not a1(inv_s0,s0);
not a2(inv_s1,s1);

and a3(w1,s0,s1,a);
and a4(w2,inv_s0,s1,b);
and a5(w3,s0,inv_s1,c);
and a6(w4,inv_s0,inv_s1,d);

or a7(out,w1,w2,w3,w4);

endmodule
```

Testbench

```
`timescale  1ns/10ps

module mux_tb;
```

```
reg s0,s1,a,b,c,d;
wire out;

mux_4x1 DUT(s0,s1,a,b,c,d,out);


initial begin
    s0=1'b0;
    s1=1'b0;
    a=1'b0;
    b=1'b0;
    c=1'b0;
    d=1'b0;
    #500 $finish;
end

always #400 s0= ~s0;
always #200 s1= ~s1;
always #100 a= ~a;
always #50   b= ~b;
always #25 c= ~c;
always #12.5 d= ~d;

initial begin
    $monitor("%g Output=%b s0=%b s1=%b a=%b b=%b c=%b
d=%b",$time,out,s0,s1,a,b,c,d);

    $dumpfile("q_2a.vcd");
    $dumpvars;
end
endmodule
```

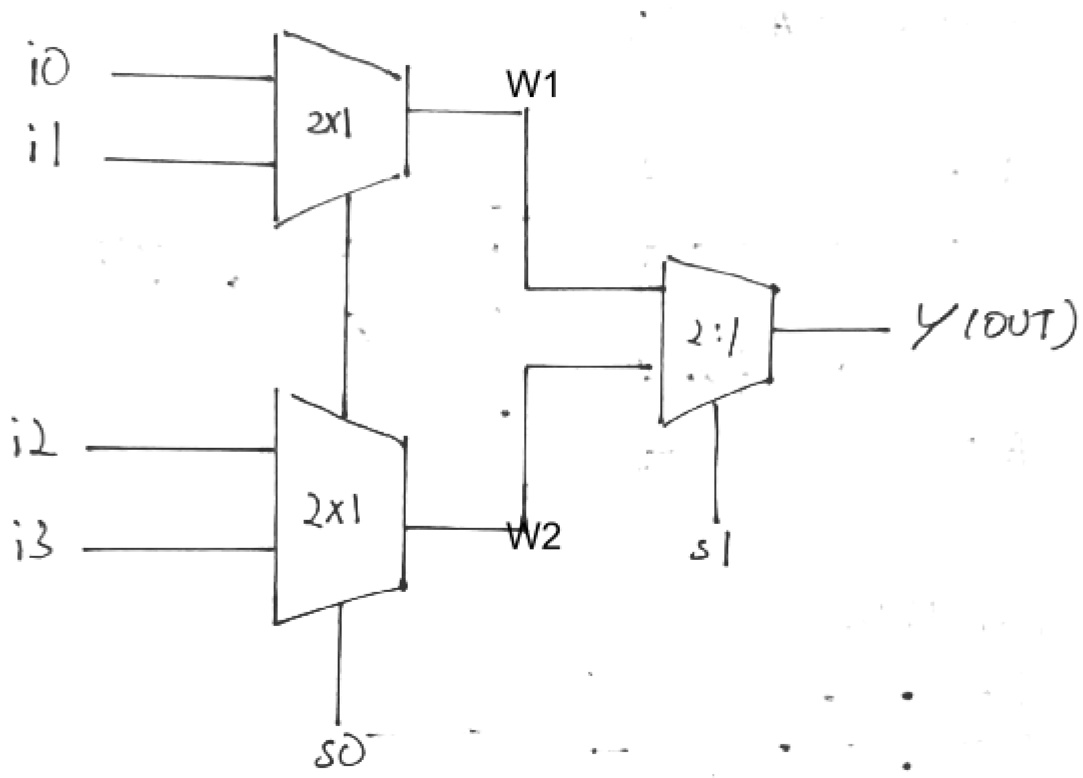
Waveform



c) 4x1MUX using 2x1 MUX

Gate Level Circuit Diagram

c) 4x1 MUX Using 2x1 MUX



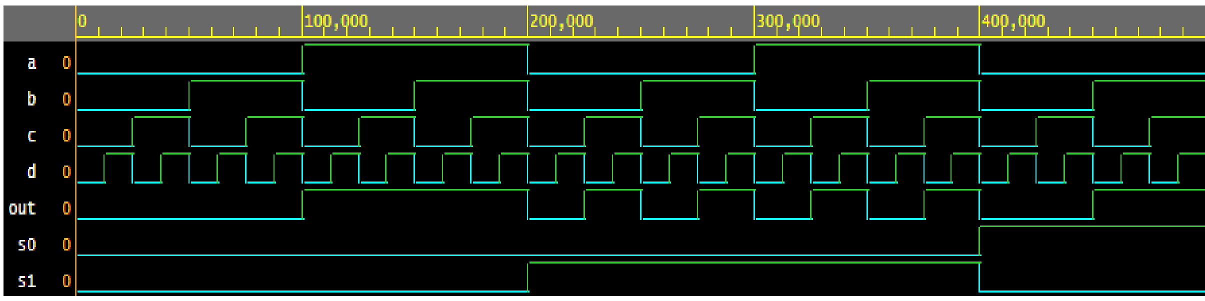
Code

```
module q2x1_MUX(input s0,a,b, output out);
wire w1,w2,inv_s0;
not a1(inv_s0,s0);
and a2(w1,inv_s0,a);
and a3(w2,s0,b);
or a4(out,w1,w2);
endmodule



module mux4x1(input s0,s1,a,b,c,d, output out);

wire w1,w2;
q2x1_MUX m21_1(s0,a,b,w1);
q2x1_MUX m21_2(s0,c,d,w2);
q2x1_MUX m21_3(s1,w1,w2,out);


endmodule
```

Testbench

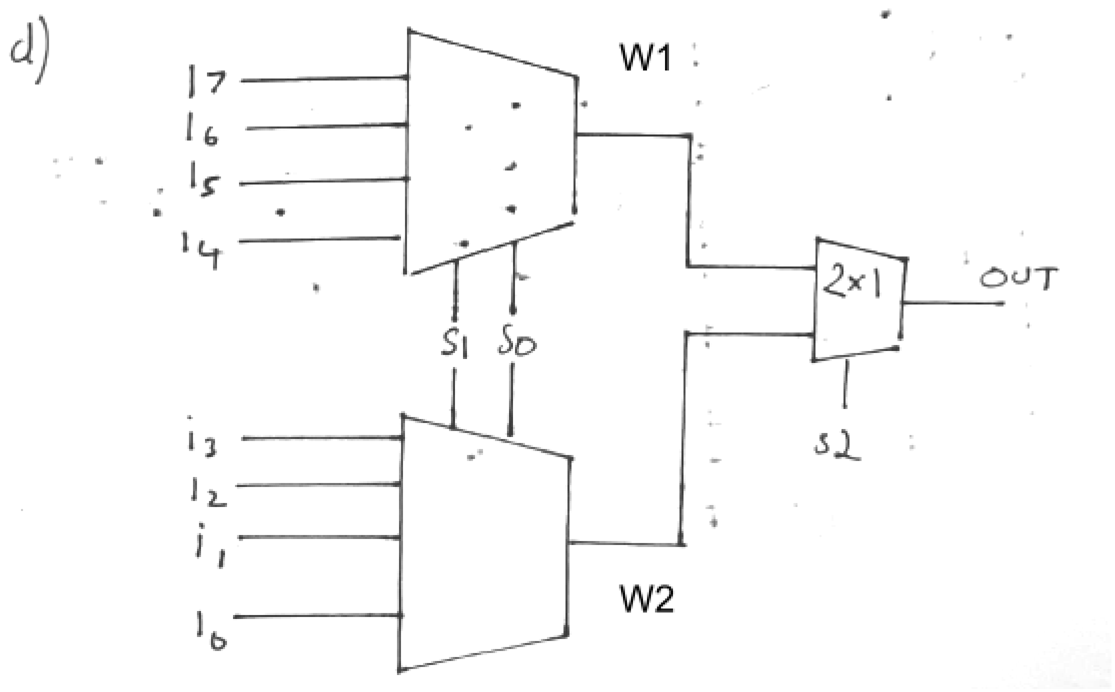
```
`timescale  1ns/10ps

module mux_tb;

reg s0,s1,a,b,c,d;
wire out;

mux4x1 DUT(s0,s1,a,b,c,d,out);


initial begin
    s0=1'b0;
    s1=1'b0;
    a=1'b0;
    b=1'b0;
    c=1'b0;
    d=1'b0;
    #500 $finish;
end

always #400 s0= ~s0;
always #200 s1= ~s1;
always #100 a= ~a;
always #50   b= ~b;
always #25 c= ~c;
always #12.5 d= ~d;

initial begin
    $monitor("%g Output=%b s0=%b s1=%b a=%b b=%b c=%b
d=%b",$time,out,s0,s1,a,b,c,d);

    $dumpfile("q_2a.vcd");
    $dumpvars;
end
endmodule
```

Waveform



d) 8x1 Mux using 4x1 and 2x1 Mux

Gate level Ckt



Code

```
module q2x1_MUX(input s0,a,b, output out);
wire w1,w2,inv_s0;
not a1(inv_s0,s0);
and a2(w1,inv_s0,a);
and a3(w2,s0,b);
or a4(out,w1,w2);
endmodule


module mux_4x1(input s0,s1,a,b,c,d,output out);

wire inv_s0,inv_s1,w1,w2,w3,w4;

not a1(inv_s0,s0);
not a2(inv_s1,s1);
```

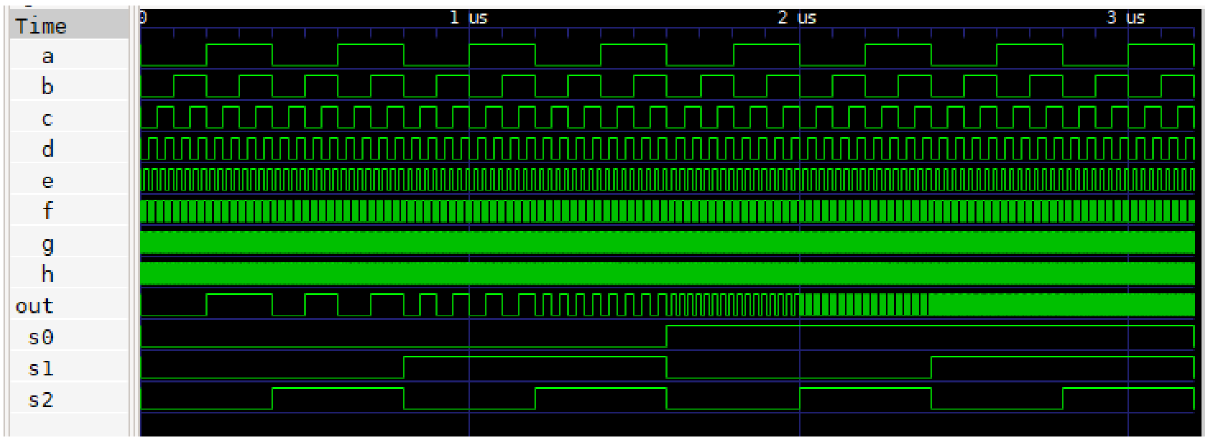
```
and a3(w1,s0,s1,a);
and a4(w2,inv_s0,s1,b);
and a5(w3,s0,inv_s1,c);
and a6(w4,inv_s0,inv_s1,d);

or a7(out,w1,w2,w3,w4);

endmodule


module mux8x1(input s0,s1,s2,a,b,c,d,e,f,g,h, output out);
wire w1,w2;

// mux_4x1 m1(s0,s1,e,f,g,h,w1);
// mux_4x1 m2(s0,s1,a,b,c,d,w2);
mux_4x1 m1(s0,s1,g,c,e,a,w1);
mux_4x1 m2(s0,s1,h,d,f,b,w2);

q2x1_MUX m3(s2,w1,w2,out);

endmodule
```

Testbench

```
`timescale  1ns/10ps

module mux_tb;

reg s0,s1,s2,a,b,c,d,e,f,g,h;
wire out;

mux8x1 DUT(s0,s1,s2,a,b,c,d,e,f,g,h,out);


initial begin
    s0=1'b0;
    s1=1'b0;
    s2=1'b0;
    a=1'b0;
    b=1'b0;
    c=1'b0;
    d=1'b0;
    e=1'b0;
    f=1'b0;
    g=1'b0;
    h=1'b0;
    #3200 $finish;
end
```

```
always #1600 s0= ~s0;
always #800 s1= ~s1;
always #400 s2= ~s2;
always #200 a= ~a;
always #100   b= ~b;
always #50 c= ~c;
always #25 d= ~d;
always #12.5 e= ~e;
always #6.25 f= ~f;
always #3.125 g= ~g;
always #1.5625 h= ~h;

initial begin
    $monitor("%g Output=%b s0=%b s1=%b a=%b b=%b c=%b d=%b
e=%b f=%b g=%b h=%b",$time,out,s0,s1,a,b,c,d,e,f,g,h);

    $dumpfile("q_2a.vcd");
    $dumpvars;
end
endmodule
```

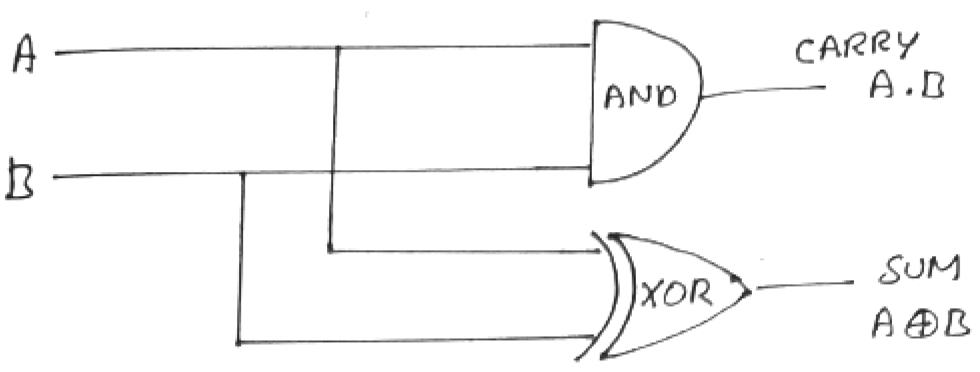
Waveform



e) Half Adder

Gate Level Ckt:

e) Half Adder



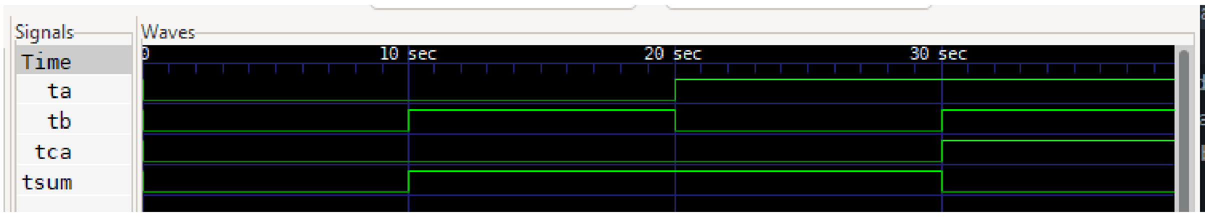
Code:

```
module halfadder ( a,b , sum,ca);
input a,b;
output sum, ca;
    assign sum=a^b;
    assign ca=a&b;
endmodule
```

Testbench:

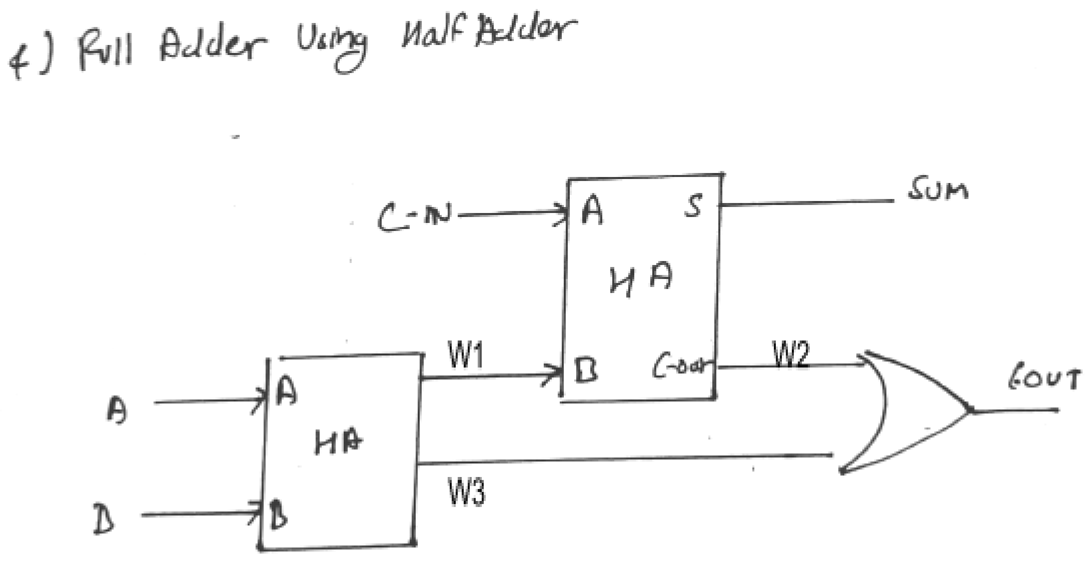
```
module halfadder_tb;

reg ta,tb;
wire tsum, tca;

halfadder ha(ta,tb,tsum,tca);

initial
begin
 ta=0;tb=0;
#10 ta=0;tb=1;
#10 ta=1;tb=0;
#10 ta=1;tb=1;

end



initial
begin
    $monitor($time,"
a=%b,b=%b,sum=%b,ca=%b",ta,tb,tsum,tca);
    $dumpfile("halfadder.vcd");
    $dumpvars;
    #40 $finish;

end
endmodule
```

Waveform:



f) Full Adder using Half Adders

Gate Level ckt



Code:

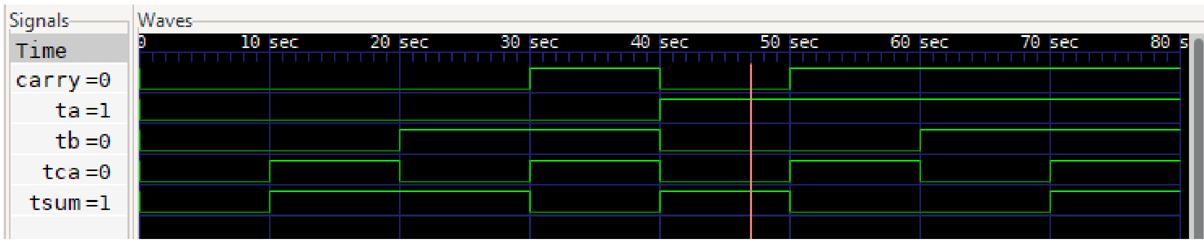
```
module halfadder ( a,b , sum,ca);
input a,b;
output sum, ca;
    assign sum=a^b;
    assign ca=a&b;
endmodule


module fulladder(input a,b,ca,output sum,carry );

wire w1, w2,w3;
```

```
halfadder h1(a,b,w1,w2);
halfadder h2(w1,ca,sum,w3);

assign carry= w3^w2;

endmodule
```

Testbench

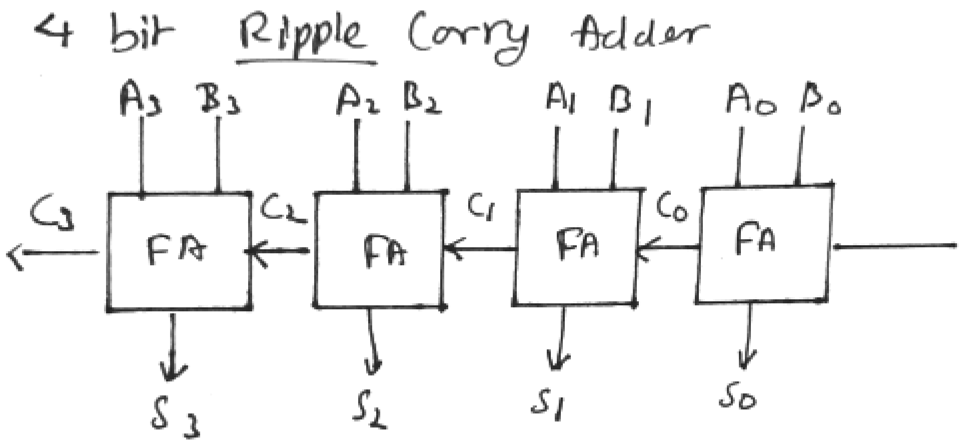
```
module fulladder_tb;

reg ta,tb,tca;
wire tsum, carry;

fulladder fa(ta,tb,tca,tsum,carry);

initial
begin
 tca=0;ta=0;tb=0;
 #10 tca=1;ta=0;tb=0;
 #10 tca=0;ta=0;tb=1;
 #10 tca=1;ta=0;tb=1;
 #10 tca=0;ta=1;tb=0;
 #10 tca=1;ta=1;tb=0;
 #10 tca=0;ta=1;tb=1;
 #10 tca=1;ta=1;tb=1;

end



initial
begin
    $monitor($time,"  a=%b,b=%b,
ca=%b,sum=%b,carry=%b",ta,tb,tca,tsum,carry);
    $dumpfile("fulladder.vcd");
    $dumpvars;
    #80 $finish;

end
endmodule
```

Output:



g) 4-bit Ripple Carry Adder using Full Adders

Gate level ckt



Code:

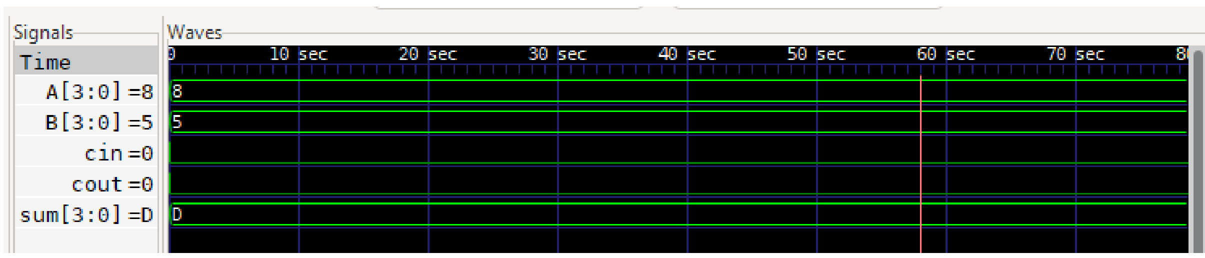
```
module halfadder ( a,b , sum,ca);
input a,b;
output sum, ca;
    assign sum=a^b;
    assign ca=a&b;
endmodule


module fulladder(input a,b,ca,output sum,carry );

wire w1, w2,w3;
halfadder h1(a,b,w1,w2);
halfadder h2(w1,ca,sum,w3);

assign carry= w3^w2;

endmodule



module RCA(a , b, cin ,s,c3);

input [3:0] a;
input [3:0] b;
input cin;

output [3:0] s;
output c3;

wire c0,c1,c2;

fulladder f0(cin,a[0],b[0],s[0],c0);
fulladder f1(c0,a[1],b[1],s[1],c1);
fulladder f2(c1,a[2],b[2],s[2],c2);
fulladder f3(c2,a[3],b[3],s[3],c3);

endmodule
```

Testbench

```
module rca_testbench();

reg [3:0] A;
reg [3:0] B;
reg cin;

wire [3:0] sum;
wire cout;



RCA dut(A,B,cin,sum, cout);

initial begin

A= 4'b1000;
B=4'b0101;
cin=1'b0;
end



initial
begin
    $monitor($time,"  A=%4b, B=%4b, cin=%b,sum=%4b,
cout=%b",A,B,cin,sum,cout);
    $dumpfile("rca.vcd");
    $dumpvars;
    #80 $finish;

end
```

```
endmodule
```

Waveform:



2) Write data flow level Verilog code using conditional operator for the following:
   a) 2x1 Mux

Truth Table

s0	A	B	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Code:

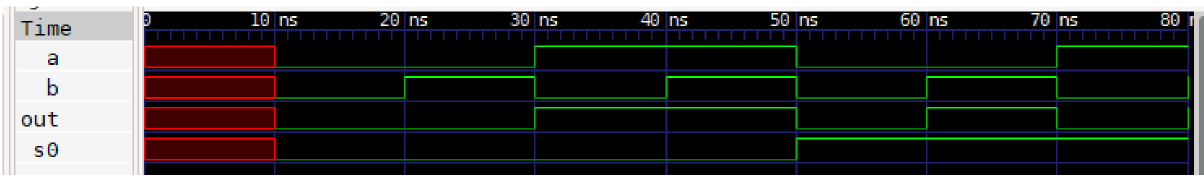
```
module mux2x1(input s0,a,b,output out);
assign out= s0? b:a;
endmodule
```

Testbench:

```
`timescale  1ns/10ps

module mux_tb;
```

```
reg s0,a,b;
wire out;

mux2x1 DUT(s0,a,b,out);

initial
begin



   #10 s0=0;a=0;b=0;
   #10 s0=0;a=0;b=1;
   #10 s0=0;a=1;b=0;
   #10 s0=0;a=1;b=1;
   #10 s0=1;a=0;b=0;
   #10 s0=1;a=0;b=1;
   #10 s0=1;a=1;b=0;
   #10 s0=1;a=1;b=1;

end

initial begin
      $monitor("%g s0=%b a=%b b=%b, out=%b
",$time,s0,a,b,out);

      $dumpfile("q_2a.vcd");
      $dumpvars;
end
endmodule
```

Waveform:



b) 4x1 Mux

s0	s1	Out
0	0	A
0	1	B
1	0	C
1	1	D

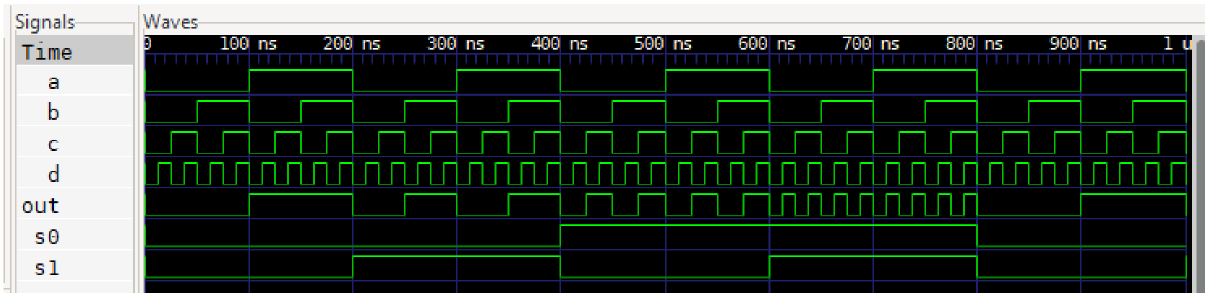
Code:

```
module mux4x1(input s0,s1,a,b,c,d,output out);
assign out= s1? (s0?d:b): (s0?c:a);
endmodule
```

Testbench:

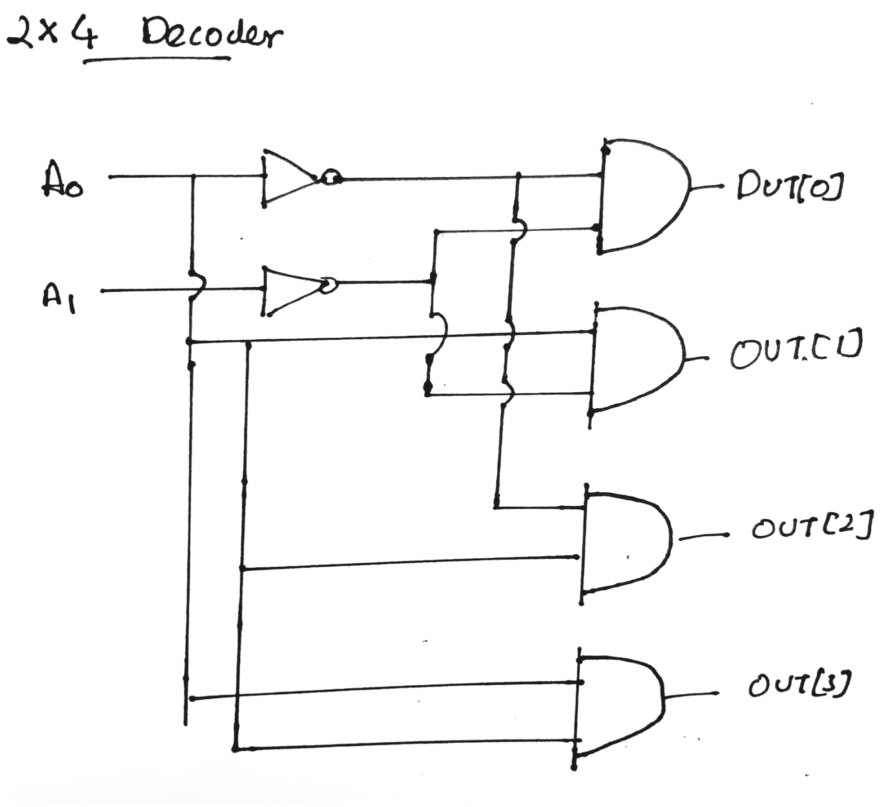
```
`timescale  1ns/10ps

module mux_tb;

reg s0,s1,a,b,c,d;
wire out;

mux4x1 DUT(s0,s1,a,b,c,d,out);


initial begin
    s0=1'b0;
    s1=1'b0;
    a=1'b0;
    b=1'b0;
    c=1'b0;
    d=1'b0;
    #1000 $finish;
end

always #400 s0= ~s0;
always #200 s1= ~s1;
always #100 a= ~a;
always #50   b= ~b;
always #25 c= ~c;
always #12.5 d= ~d;

initial begin
    $monitor("%g Output=%b s0=%b s1=%b a=%b b=%b c=%b
d=%b",$time,out,s0,s1,a,b,c,d);

    $dumpfile("q_2b.vcd");
    $dumpvars;
end
endmodule
```

Waveform



c) 2 to 4 Decoder

Circuit Diagram:



Truth Table:

A[1]	A[0]	Out
0	0	0001
0	1	0010
1	0	0100
1	1	1000

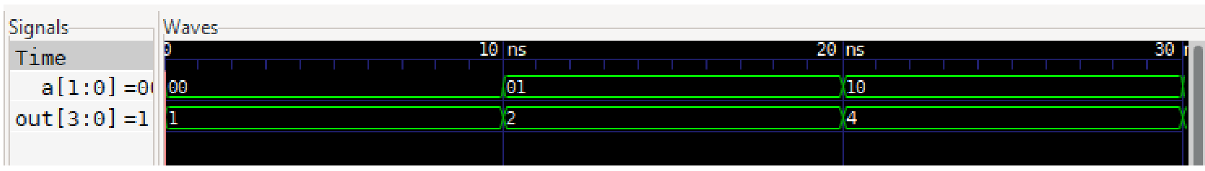
Code:

```
module decoder2x4(a,out);
input [1:0] a;
output [3:0] out;


assign out= (a==2'b00)? 4'b0001:
            (a==2'b01)? 4'b0010:
            (a==2'b10)?4'b0100:4'b1000;


endmodule
```

Testbench:

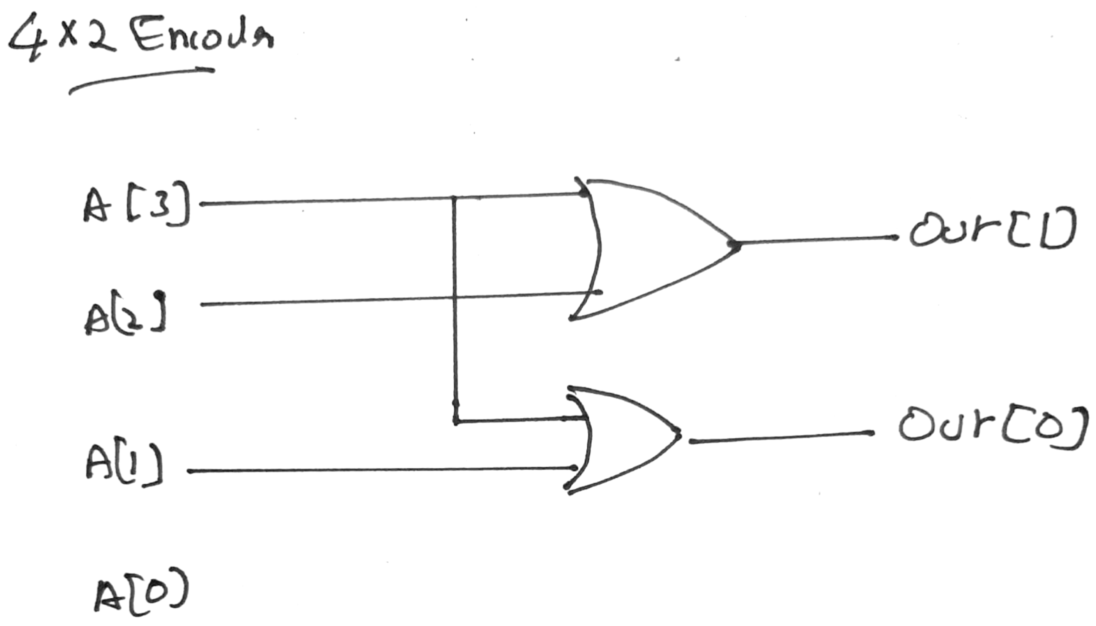
```
`timescale  1ns/10ps

module testbench;

reg [1:0] a;
wire [3:0] out;

decoder2x4 DUT(a,out);

initial
begin

 a=2'b00;
#10  a=2'b01;
#10  a=2'b10;
#10  a=2'b11;
end

initial begin
      $monitor("%g a=%2b out=%4b",$time,a,out);

      $dumpfile("q_2c.vcd");
      $dumpvars;
end
endmodule
```

Waveform:



d) 4 to 2 Encoder

<u>Ckt Diagram</u>



<u>Truth table</u>

A	Output[1]	Output[0]
0001	0	0
0010	0	1
0100	1	0
1000	1	1

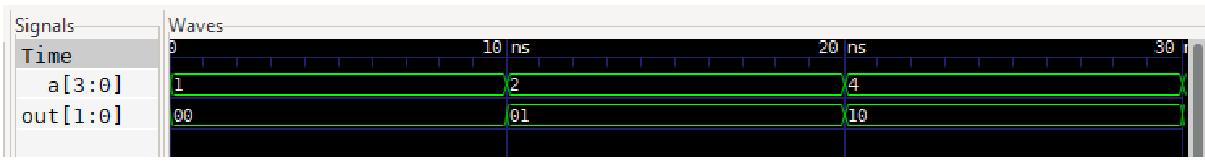
Code:

```
module encoder4x2(A,out);
input [3:0] A;
output [1:0] out;

assign out= (A==4'b0001)?2'b00:
            (A==4'b0010)?2'b01:
            (A==4'b0100)?2'b10:2'b11;


endmodule
```

Testbench:

```
`timescale  1ns/10ps

module testbench;

reg [3:0] a;
wire [1:0] out;

encoder4x2 DUT(a,out);

initial
begin

 a=4'b0001;
#10  a=4'b0010;
#10  a=4'b0100;
#10  a=4'b1000;
end

initial begin
      $monitor("%g a=%4b out=%2b",$time,a,out);

      $dumpfile("q_2d.vcd");
      $dumpvars;
end
endmodule
```

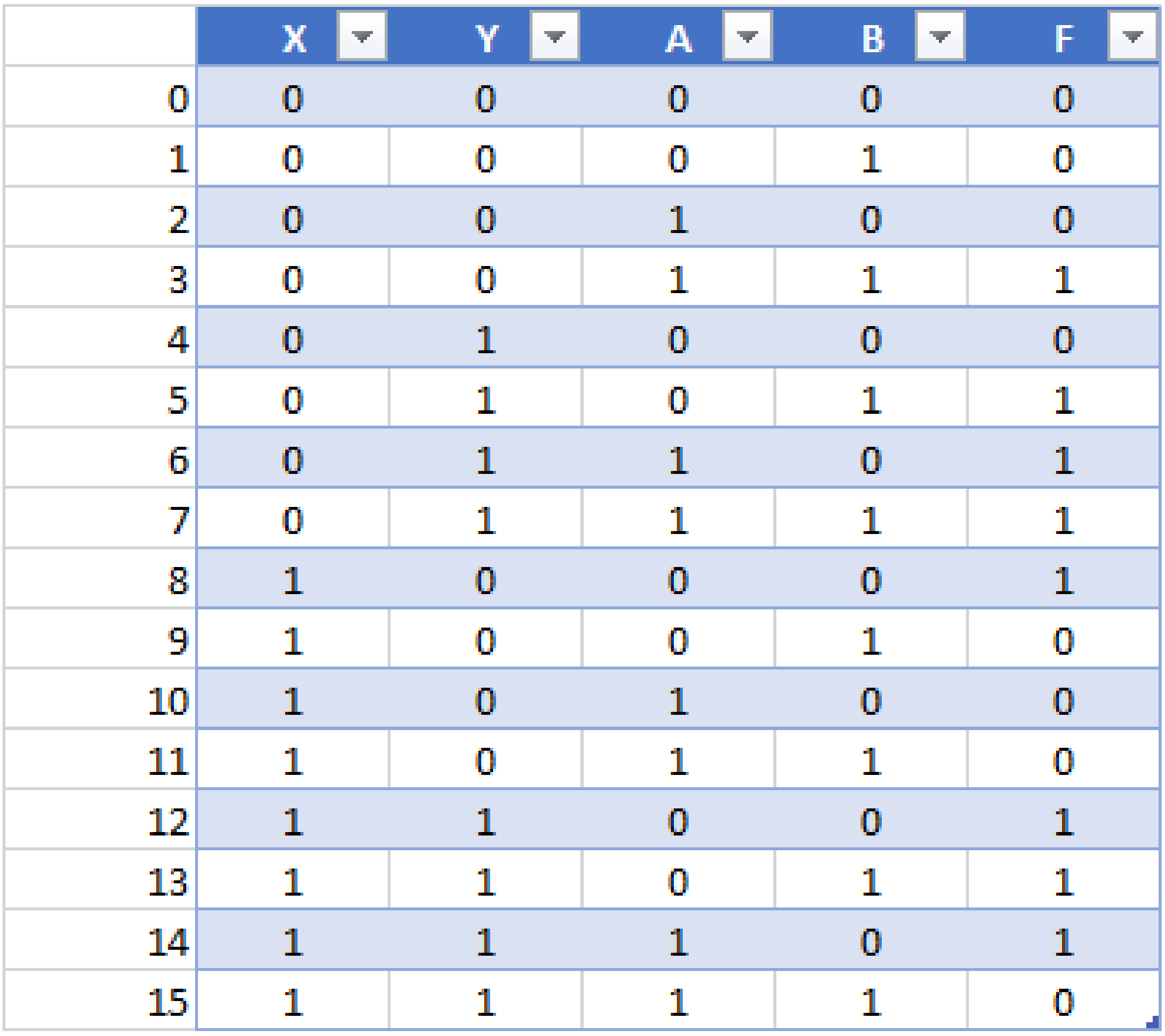
Waveform:



3) Let us consider the truth table

X	Y	Function
0	0	AND
0	1	OR
1	0	NOR
1	1	NAND

To find the structural level code, we have to find how the actual circuit would look like. For that, let us consider all the possibilities of A, B m X and Y and find the gate level implementation using a kmap.



	X	Y	A	B	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

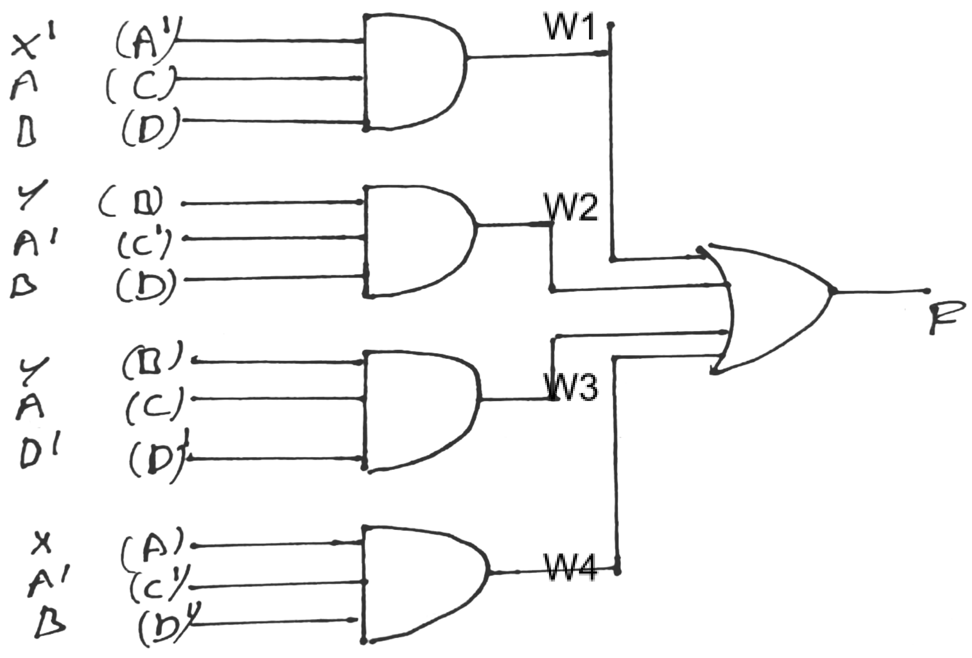
Gate level ckt

Let C & D be A & B Respectively
& A & B be control signals X & Y

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	0
$\bar{A}B$	0	1	1	1
AB	1	1	0	1
$A\bar{B}$	1	0	0	0

$$Y = F = A'CD + BC'D + BCD' + AC'D'$$

$$= X'AB + YA'B + YAD' + XA'D'$$



Code:

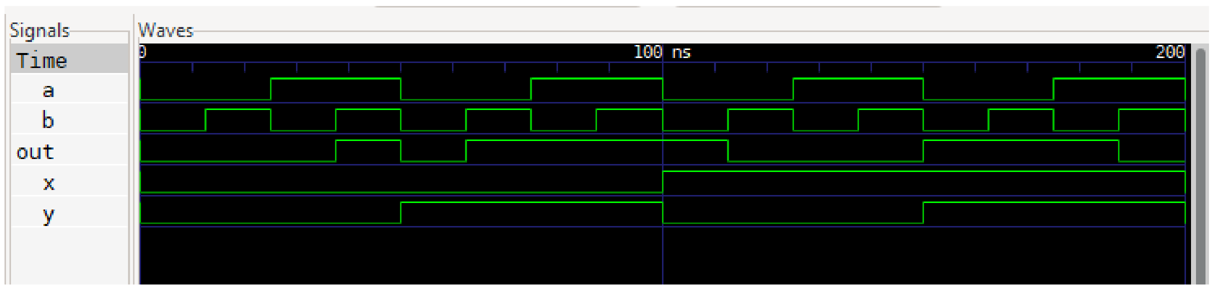
```
module multifunctiongate( a,b,c,d , F);

input a,b,c,d;
output F;

wire inv_a,inv_c,inv_d;
wire w1,w2,w3,w4;
```

```
not n1(inv_a,a);
not n2(inv_d,d);
not n3(inv_c,c);

and a1(w1,inv_a,c,d);
and a2(w2,inv_c,b,d);
and a3(w3,b,c,inv_d);
and a4(w4,a,inv_c,inv_d);

or o1(F,w1,w2,w3,w4);


endmodule
```

Testbench

```
`timescale  1ns/10ps

module testbench;

reg x,y,a,b;
wire out;

multifunctiongate DUT(x,y,a,b,out);


initial begin

    a=1'b0;
    b=1'b0;
    x=1'b0;
    y=1'b0;
    #200 $finish;
end

always #100 x= ~x;
always #50   y= ~y;
always #25 a= ~a;
always #12.5 b= ~b;

initial begin
    $monitor("%g Output=%b x=%b y=%b a=%b b=%b ",$time,out,x,y,a,b);

    $dumpfile("q_3.vcd");
    $dumpvars;
end
endmodule
```

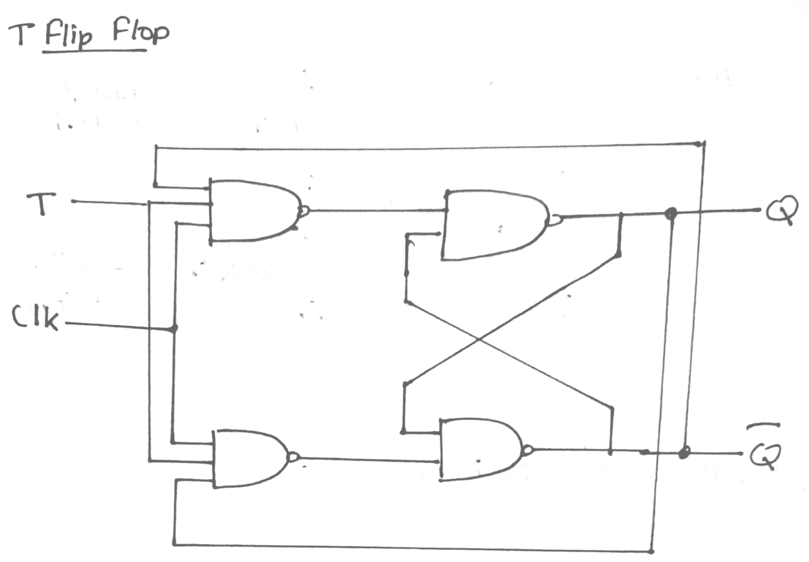
Waveform:



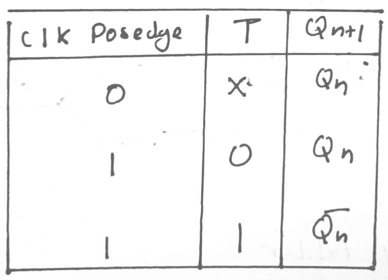
4) Write behavioral level Verilog code for the following:

   a) T Flip flop

   Truth table and Ckt:



Truth Table of TFF



clk Posedge	T	Q_{n+1}
0	X	Q_n
1	0	Q_n
1	1	$\overline{Q_n}$

Code:

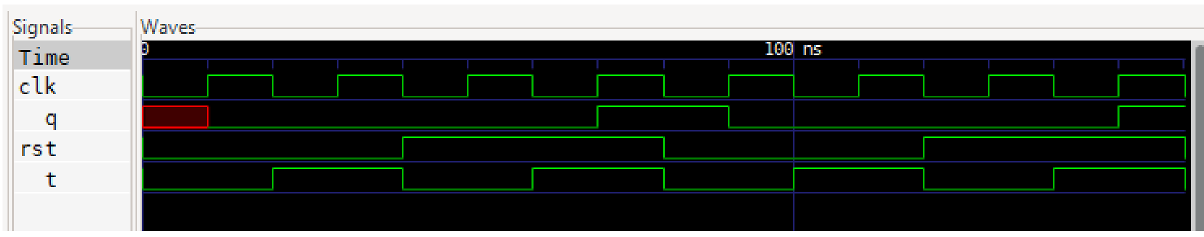
```
module t_flipflop(input clk,rst,t,output q);

reg q;
```

```
always@(posedge clk or posedge rst) begin
    if (rst==1'b0)
    q<=0;
    else
        if (t==1'b1)
        q<=~q;
        else
        q<=q;

end
endmodule
```

Testbench:

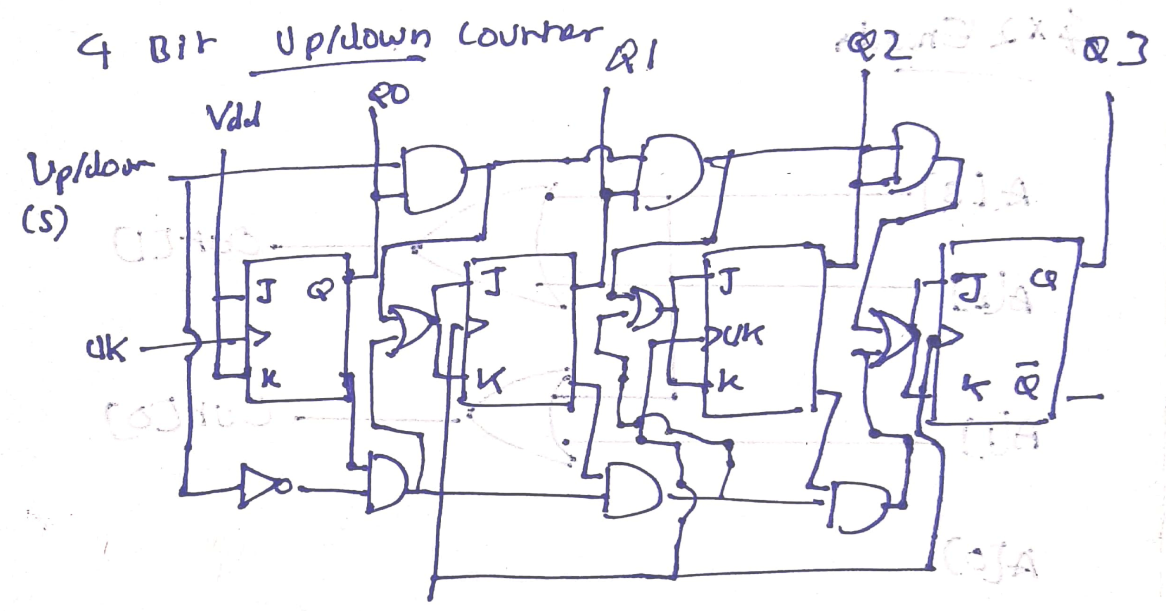
```
`timescale  1ns/10ps

module testbench;

reg clk,rst,t;

t_flipflop DUT(clk,rst,t,q);


initial begin

    clk=1'b0;
    rst=1'b0;
    t=1'b0;

    #160 $finish;
end

always #10 clk= ~clk;
always #20   t= ~t;
always #40 rst=~rst;

initial begin
    $monitor("%g Output=%b clk=%b y=%b rst=%b t=%b
",$time,q,clk,rst,t);

    $dumpfile("q_4a.vcd");
    $dumpvars;
end
endmodule
```

Waveform:



b) 4-bit up-down counter

Circuit Diagram



Code:

```
module updowncounter(clk,rst,s,count);

input clk,rst,s;// s =0 for count down and vice versa

output reg [3:0] count;

initial count=4'b0000;


always @(posedge (clk) or posedge(rst))
begin
if (rst==1'b1)
    count<=0;
else
    if (s==1)
    count<=count+1;
    else
    count<=count-1;
```

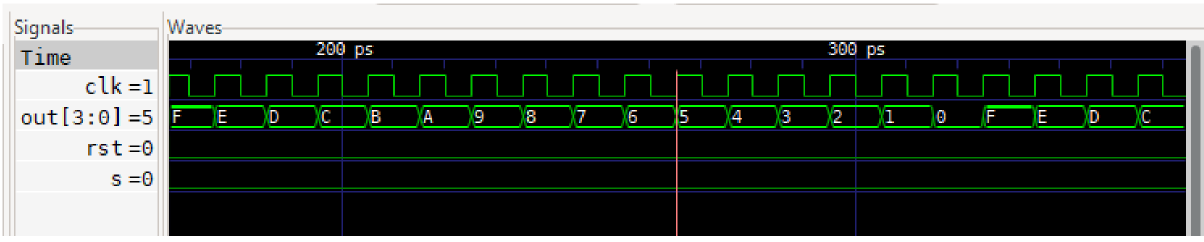
```
end


endmodule
```

Testbench

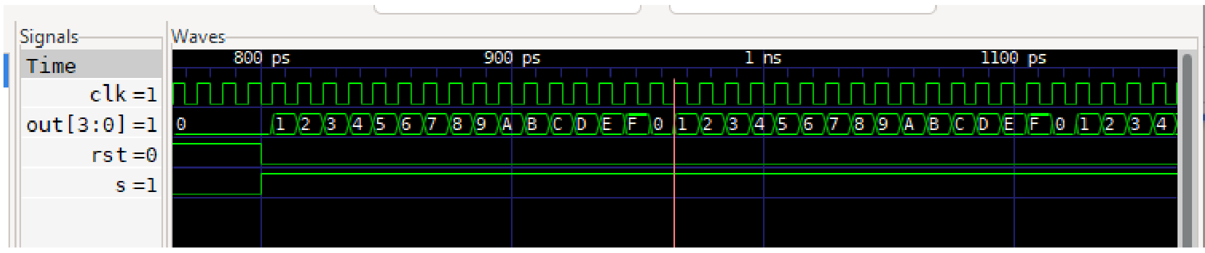
```
`timescale 1ps/1ps

module testbench;
reg clk,rst,s;
wire [3:0] out;
updowncounter DUT(clk,rst,s,out);


initial begin
    clk=1'b0;
    rst=1'b0;
    s=1'b0;
    #1600 $finish;
end

always #5 clk=~clk;
always #400 rst=~rst;
always #800 s=~s;


initial begin
    $dumpfile("4b.vcd");
    $dumpvars;
    $monitor("%g clk=%b rst=%2b s=%b, count=%4b",$time,clk,
rst,s, out);
end

endmodule
```

Waveform:

Downcounting



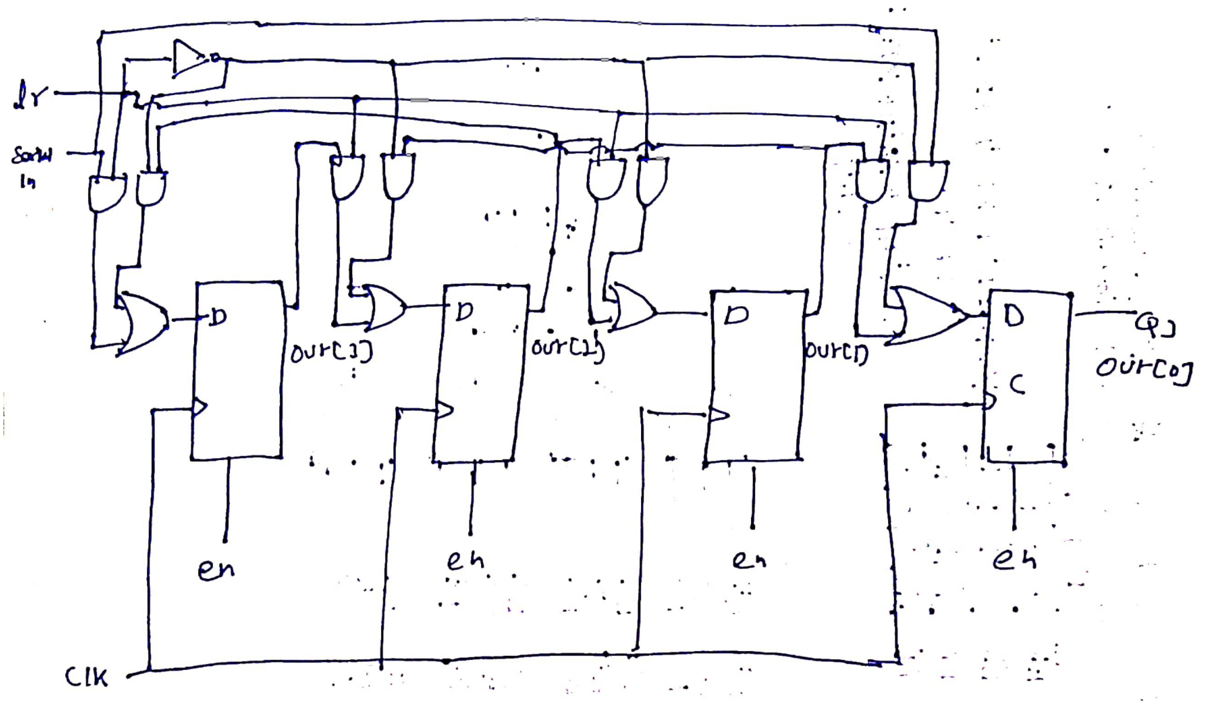
Upcounting


Signals
Waves
Time
800 ps
900 ps
1 ns
1100 ps
clk =1
out[3:0] =1
rst =0
s =1
0
1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4

c) 8-bit shift register which is capable of doing right shift and left shift

Circuit Diagram:

(illustrated 4 bit shift register) In we can extend the design to 8 bits also



Code:

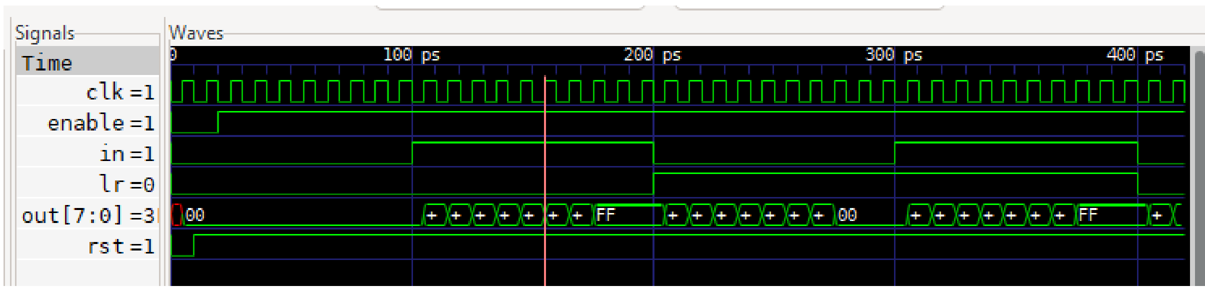
```
module shiftreg(input clk,lr,rst,in,enable,output reg [7:0]
out); //lr to define which direction it is shifting. Lr=0 for
left shift

always@ (posedge clk or posedge rst)
begin
    if (rst==1'b0)
    out<=0;
    else
    begin
        if (enable)
        begin
            if (lr==1'b0)
            out<={out[6:0],in};
            else
```

```
              out<={in,out[7:1]};
          end
      end

end



endmodule
```

Testbench

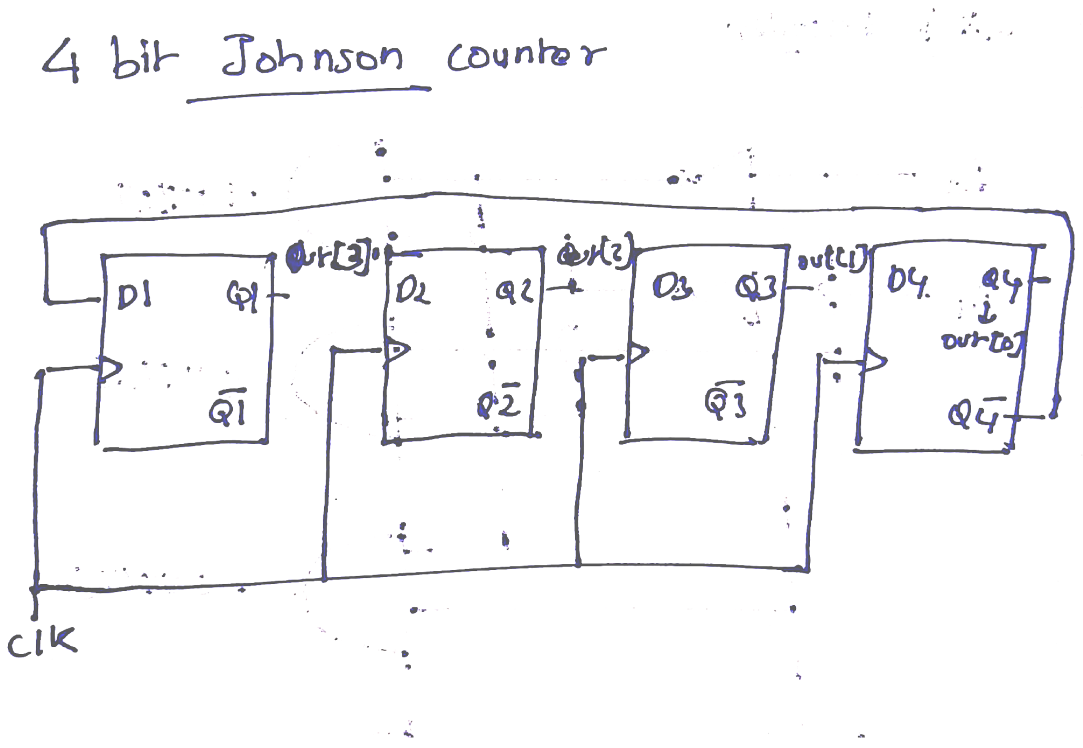
```
`timescale 1ps/1ps

module testbench;
reg clk,lr,rst,in,enable;
wire [7:0] out;

shiftreg DUT(clk,lr,rst,in,enable, out);


initial begin
    clk=1'b0;
    rst=1'b0;
    in=1'b0;
    enable=1'b0;
    lr=1'b0;

    #10 rst =1'b1;
    #10 enable =1'b1;
    #400 $finish;
end

always #5 clk=~clk;
always #100 in=~in;
always #200 lr=~lr;



initial begin
    $dumpfile("4c.vcd");
    $dumpvars;
    $monitor("%g clk=%b rst=%2b lr=%b, input=%b, enable = %b
,regvalue=%8b",$time,clk, rst,lr,in,enable, out);
end

endmodule
```

Waveform:



d) 4-bit Johnson counter

Circuit Diagram



Code:

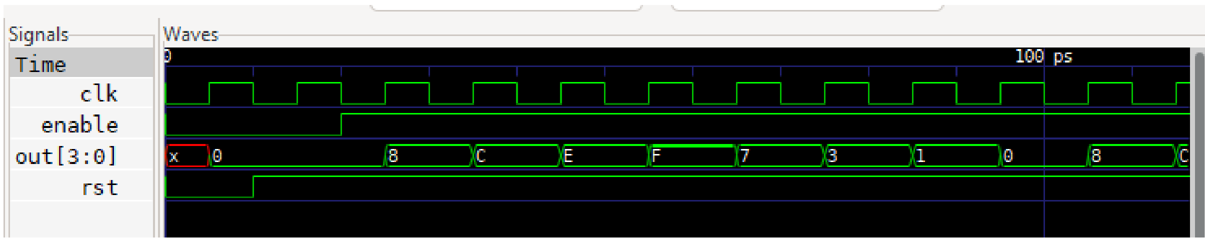
```
module johnsoncounter( clk,rst,enable, out);


input clk,rst,enable;
output reg [3:0] out;
integer i;

always@ (posedge clk or posedge rst)
begin
```

```
    if (rst==1'b0)
    out<=0;
    else
    begin
        if (enable)
        begin
            out[3]<=~out[0];
            for (i=0; i<3;i=i+1)
            begin
                out[i]<=out[i+1];
            end
        end

    end

end



endmodule
```

Testbench

```
`timescale 1ps/1ps

module testbench;
reg clk,lr,rst,in,enable;
wire [3:0] out;

johnsoncounter DUT(clk,rst,enable, out);


initial begin
    clk=1'b0;
    rst=1'b0;
    enable=1'b0;
    #10 rst =1'b1;
    #10 enable =1'b1;
    #100 $finish;
end

always #5 clk=~clk;


initial begin
    $dumpfile("4d.vcd");
    $dumpvars;
    $monitor("%g clk=%b, rst=%2b , enable = %b
,counterval=%4b",$time,clk, rst,enable, out);
end

endmodule
```

Waveform



←The End→