EXPERIMENT- 2
Verilog Sequential Modeling

Objective:

To understand the concepts related to sequential modeling style and write Verilog programs using the same.

Theory: To model the behaviour of a digital description using sequential modeling the following two statements are primarily used:

i. Initial statement

ii. Always statement

Initial statement: An *initial* statement executes only once. It begins its execution at the start of simulation which is at time t = 0.

Syntax

```
initial
[timing_control] procedural_statement
```

Always statement: An *always* statement executes repeatedly. Just like the *initial* statement, an *always* statement also begins execution at time t = 0.

Syntax

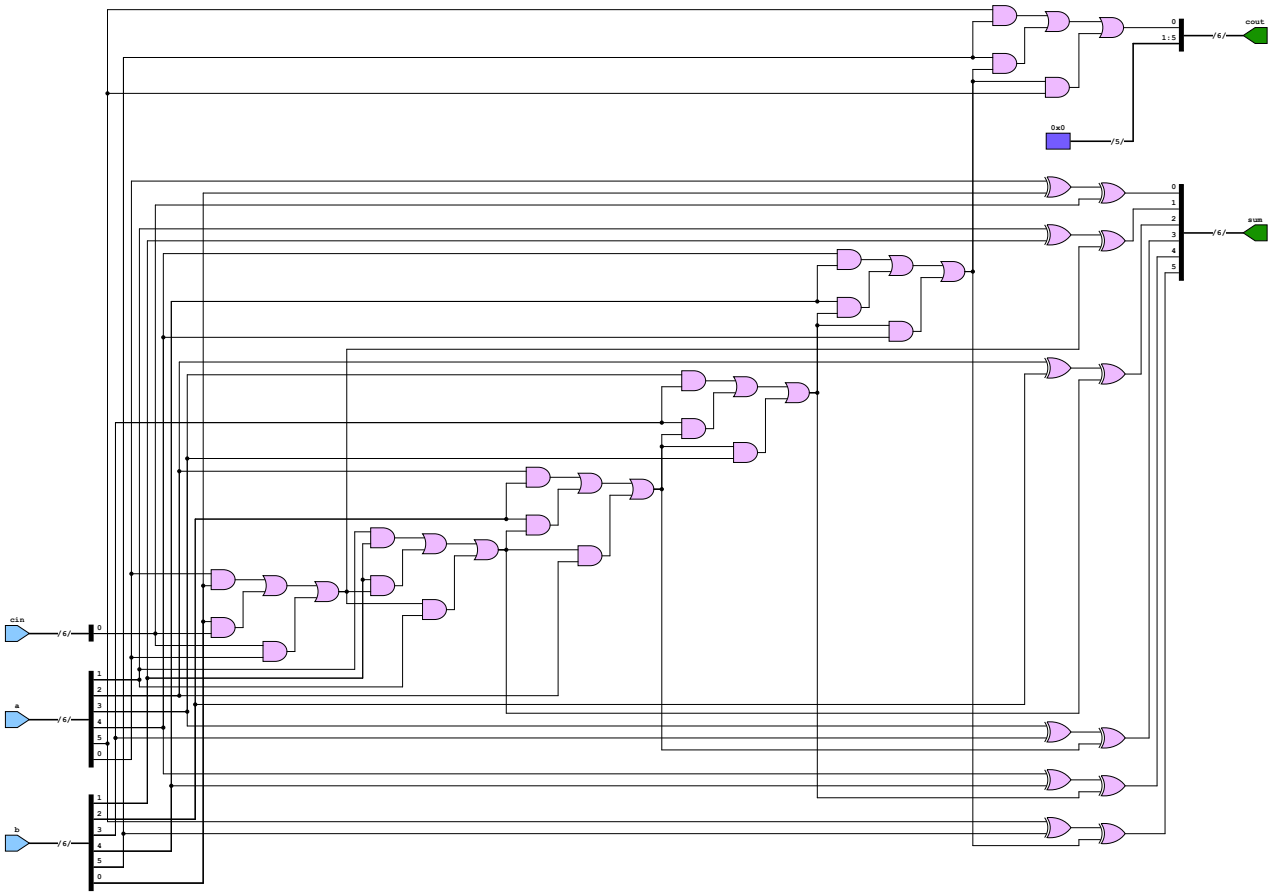
```
always
[timing_control] procedural_statement
```

Only a *register* data type can be assigned a value in either of these statements. Such a data type retains its value until a new value is assigned. All *initial* and *always* statements begin execution at time t=0 concurrently. If no delays are specified in a procedural assignment, zero delay is the default, that is, assignment occurs instantaneously.

Exercise Problems

1. **Write the sequential Verilog code for N bit full adder (assume N = 6 and use for-loop statement) and verify the design by simulation.**

Circuit



Source Code

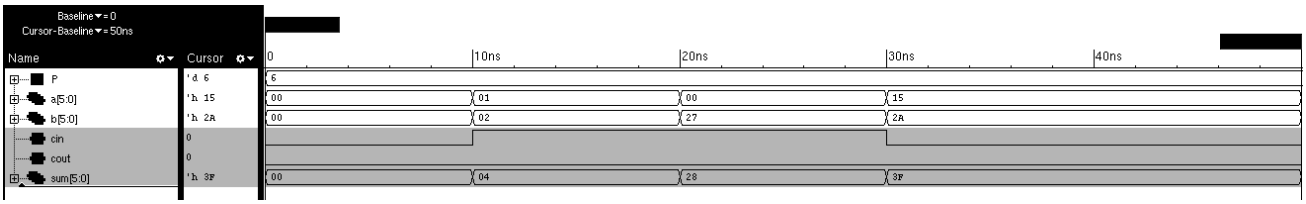
```
module nbitfa #(parameter P=6)(sum,cout,a,b,cin);
input wire [P-1:0]a,b,cin;
output reg [P-1:0]sum,cout;
reg [P:0]s;
integer i;
always @(*)
begin
s[0]=cin;
for(i=0;i<=P-1;i=i+1)
begin
sum[i]=a[i]^b[i]^s[i];
s[i+1]=(a[i]&b[i])|(b[i]&s[i])|(s[i]&a[i]);
```

```
end
cout = s[P];
end
endmodule
```

Testbench

```
module tb();
parameter P=6;
reg [P-1:0]a,b;
reg cin;
wire [P-1:0]sum;
wire cout;
nbitfa uut(sum,cout,a,b,cin);
initial begin
    a=6'b000000; b=6'b000000; cin=1'b0;
#10 a=6'b000001; b=6'b000010; cin=1'b1;
#10 a=6'b000000; b=6'b100111; cin=1'b1;
#10 a=6'b010101; b=6'b101010; cin=1'b0;
end
initial begin
$monitor($time,"a= %b, b= %b, cin=%b, sum=%b, cout=%b",a,b,cin,sum,cout);
#50 $finish;
end
endmodule
```

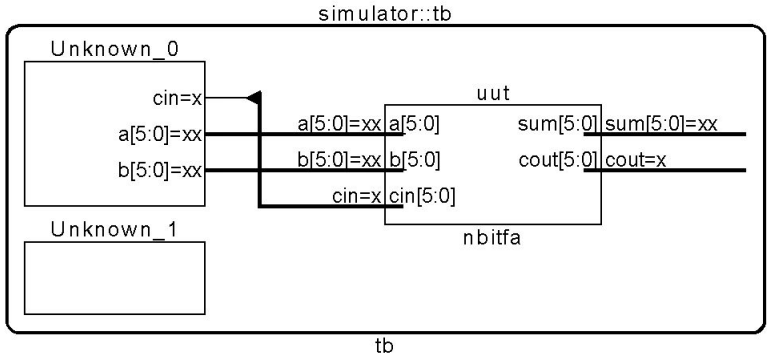
Waveform



Console

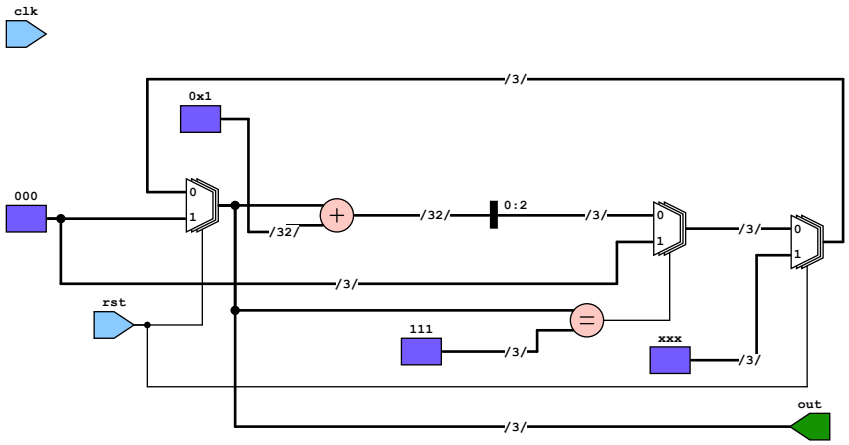
```
ncsim>
ncsim> run
                 0a= 000000, b= 000000, cin=0, sum=000000, cout=0
                10a= 000001, b= 000010, cin=1, sum=000100, cout=0
                20a= 000000, b= 100111, cin=1, sum=101000, cout=0
                30a= 010101, b= 101010, cin=0, sum=111111, cout=0
Simulation complete via $finish(1) at time 50 NS + 0
./nbitfa_tb.v:20 #50 $finish;
ncsim>
```

Schematic



2. Write the sequential Verilog code for synchronous mod 8 counter and verify the design by simulation.

Circuit



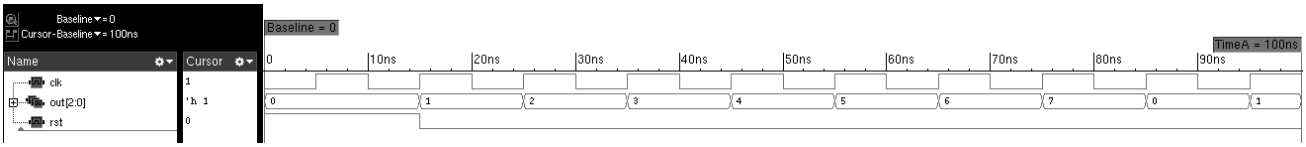
Source Code

```
module mod8counter(input clk, input rst, output reg [2:0] out);
always @(posedge clk or rst)
begin
if (rst)
out<=0;
else if(out == 7)
out<=0;
else
out=out+1;
end
endmodule
```

Testbench

```
module tb();
reg clk;
reg rst;
wire [2:0] out;
mod8counter uut(clk, rst, out);
always #5 clk = ~clk;
initial
begin
{clk,rst}<=1'b1;
$monitor ("T=%t rst=%b out=%b", $time, rst, out);
repeat (2) @(posedge clk);
rst<=0;
repeat (20) @(posedge clk);
end
initial
begin
#100 $finish;
end
endmodule
```

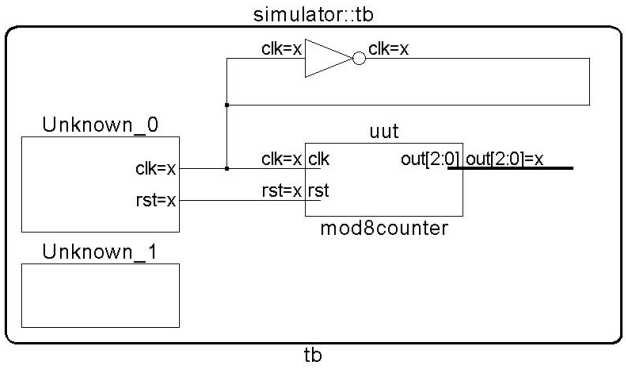
Waveform



Console

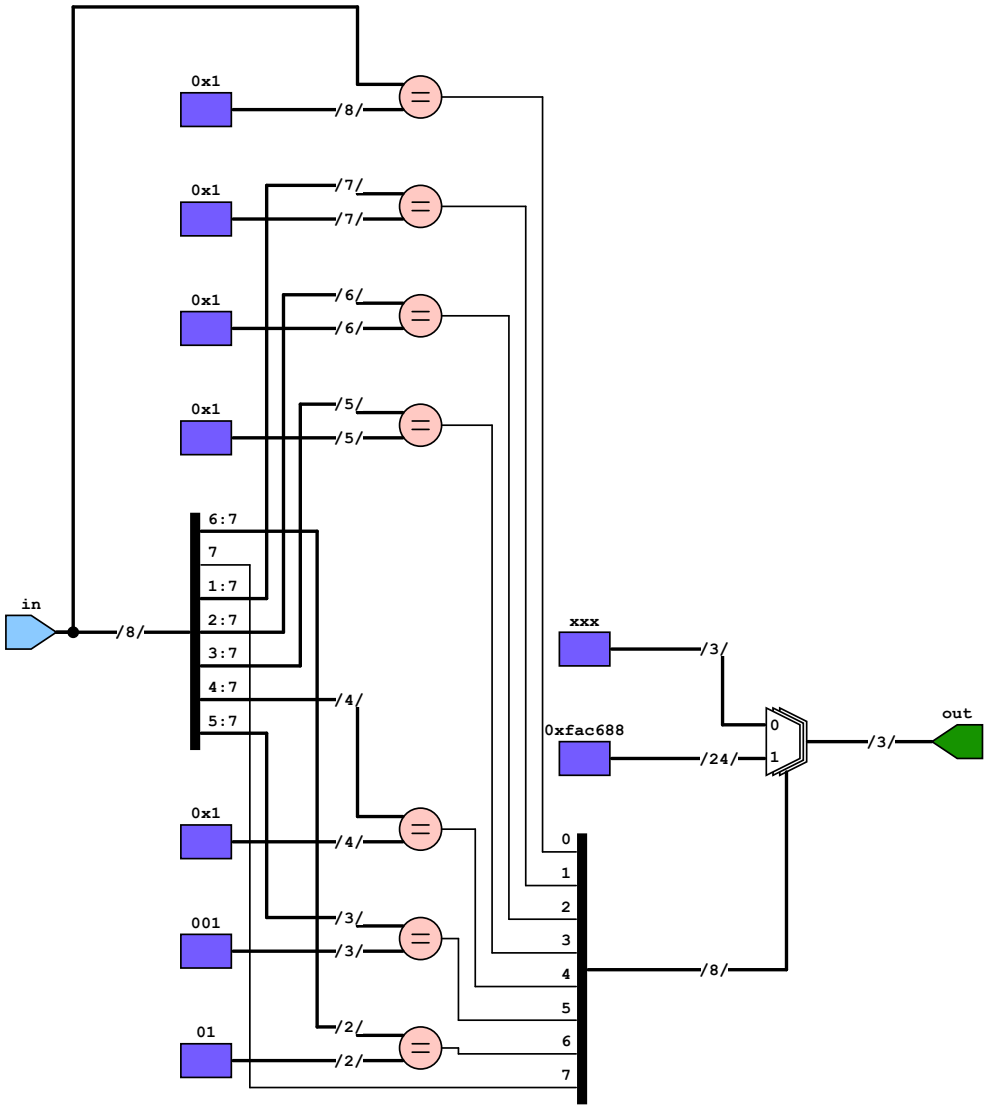
```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
T=                    0 rst=1 out=000
T=                   15 rst=0 out=001
T=                   25 rst=0 out=010
T=                   35 rst=0 out=011
T=                   45 rst=0 out=100
T=                   55 rst=0 out=101
T=                   65 rst=0 out=110
T=                   75 rst=0 out=111
T=                   85 rst=0 out=000
T=                   95 rst=0 out=001
Simulation complete via $finish(1) at time 100 NS + 0
./mod8counter_tb.v:17 #100 $finish;
ncsim>
```

Schematic



3. **Write a sequential Verilog code for 8-bit priority encoder and verify the design by simulation.**

Circuit



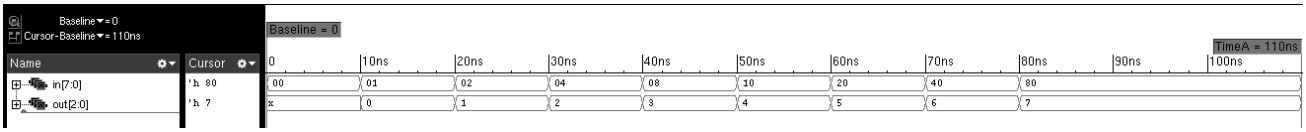
Source Code

```
module priorityencoder (input wire [7:0] in,output reg [2:0] out);
always @(*) begin
casez (in)
8'b1???????: out = 3'b111;
8'b01??????: out = 3'b110;
8'b001?????: out = 3'b101;
8'b0001????: out = 3'b100;
8'b00001???: out = 3'b011;
8'b000001??: out = 3'b010;
8'b0000001?: out = 3'b001;
8'b00000001: out = 3'b000;
default: out = 3'bxxx;
endcase
end
endmodule
```

Testbench

```
module tb();
reg [7:0] in; wire [2:0] out;
priorityencoder uut (.in(in),.out(out));
initial begin
in = 8'b00000000; #10;
in = 8'b00000001; #10;
in = 8'b00000010; #10;
in = 8'b00000100; #10;
in = 8'b00001000; #10;
in = 8'b00010000; #10;
in = 8'b00100000; #10;
in = 8'b01000000; #10;
in = 8'b10000000; #10;
#20 $finish;
end
initial begin
$monitor($time,"in = %b, out = %b",in,out);
end
endmodule
```

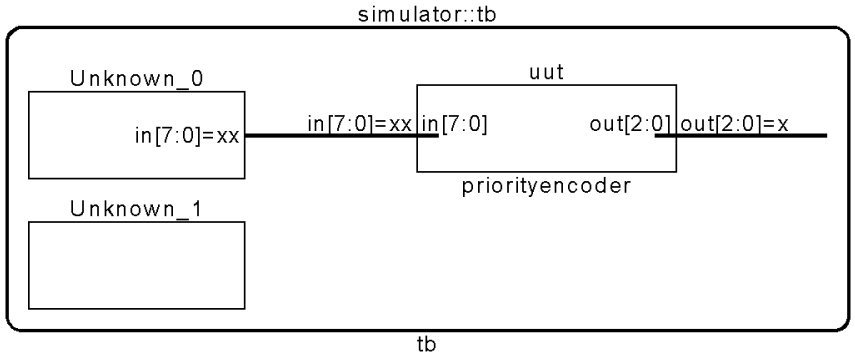
Waveform



Console

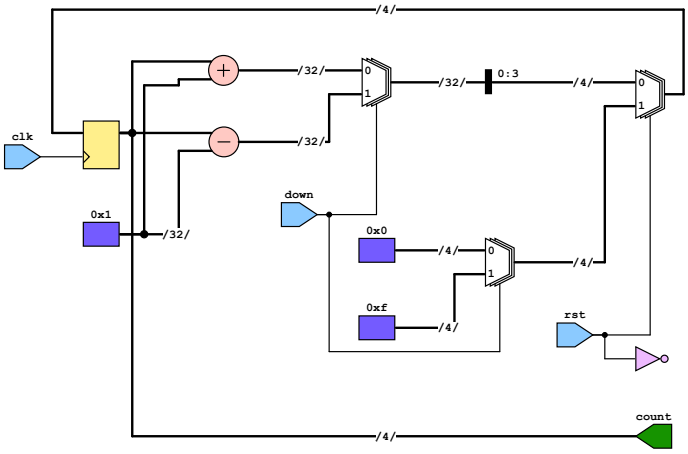
```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.in tb.out
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                    0in = 00000000, out = xxx
                 10in = 00000001, out = 000
                 20in = 00000010, out = 001
                 30in = 00000100, out = 010
                 40in = 00001000, out = 011
                 50in = 00010000, out = 100
                 60in = 00100000, out = 101
                 70in = 01000000, out = 110
                 80in = 10000000, out = 111
Simulation complete via $finish(1) at time 110 NS + 0
./priorityencoder_tb.v:18 #20 $finish;
ncsim>
```

Schematic



4. **Write a Verilog code and test bench to model and simulate 4 bit synchronous up/ down counter using Cadence tool.**

Circuit



Source Code

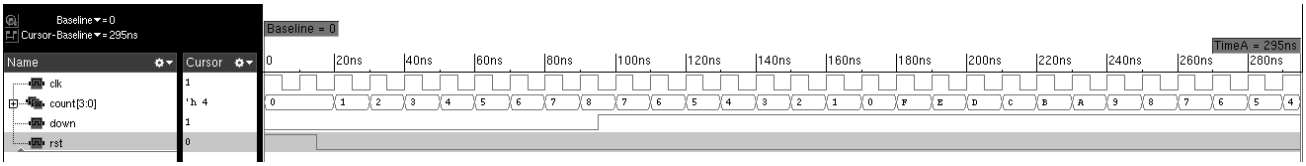
```
module updown(input clk, input down, input rst, output reg [3:0]count);
always @(posedge clk)
begin
if(!rst)
count <= down? count-1 : count+1;
else
count <= down? 4'b1111 : 4'b0000;
end
endmodule
```

Testbench

```
module tb();
reg clk, down, rst;
wire [3:0] count;
updown uut(clk, down, rst, count);
initial
clk = 1;
always
#5 clk = ~clk;
initial
begin
```

```
rst = 1;
down = 0;
#15 rst = 0;
#80 down = 1;
#200 $finish;
end
initial
$monitor($time,"rst=%b, down=%b, count=%b", rst, down, count);
endmodule
```

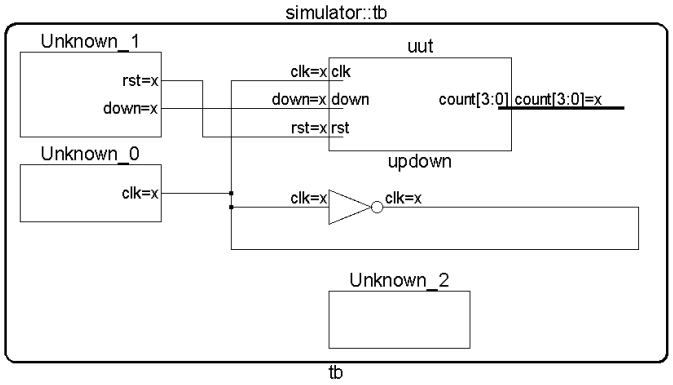
Waveform



Console

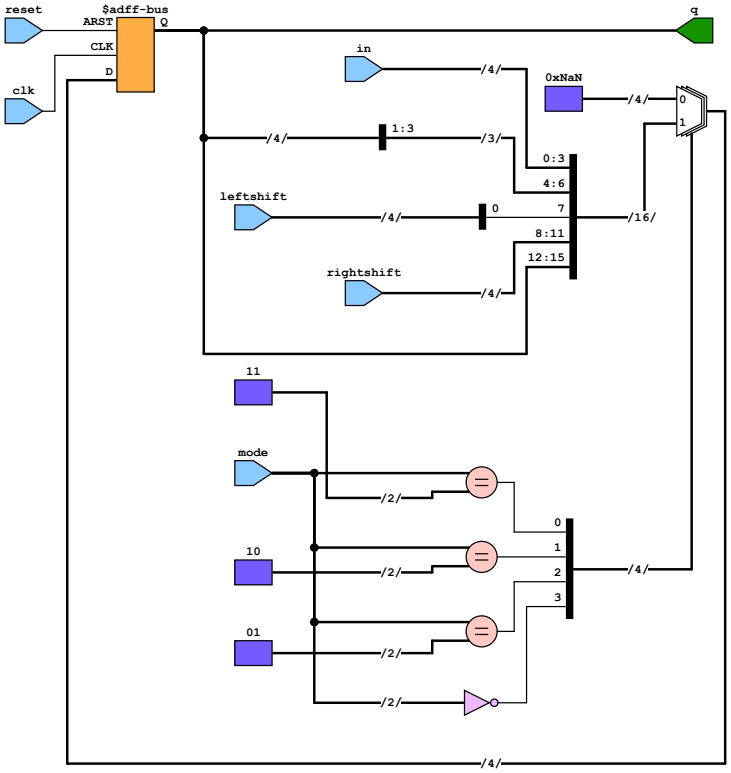
```
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                    0rst=1, down=0, count=0000
                 15rst=0, down=0, count=0000
                 20rst=0, down=0, count=0001
                 30rst=0, down=0, count=0010
                 40rst=0, down=0, count=0011
                 50rst=0, down=0, count=0100
                 60rst=0, down=0, count=0101
                 70rst=0, down=0, count=0110
                 80rst=0, down=0, count=0111
                 90rst=0, down=0, count=1000
                 95rst=0, down=1, count=1000
                100rst=0, down=1, count=0111
                110rst=0, down=1, count=0110
                120rst=0, down=1, count=0101
                130rst=0, down=1, count=0100
                140rst=0, down=1, count=0011
                150rst=0, down=1, count=0010
                160rst=0, down=1, count=0001
                170rst=0, down=1, count=0000
                180rst=0, down=1, count=1111
                190rst=0, down=1, count=1110
                200rst=0, down=1, count=1101
                210rst=0, down=1, count=1100
                220rst=0, down=1, count=1011
                230rst=0, down=1, count=1010
                240rst=0, down=1, count=1001
                250rst=0, down=1, count=1000
                260rst=0, down=1, count=0111
                270rst=0, down=1, count=0110
                280rst=0, down=1, count=0101
                290rst=0, down=1, count=0100
Simulation complete via $finish(1) at time 295 NS + 0
./updown_tb.v:15 #200 $finish;
ncsim>
```

Schematic



5. **Write sequential Verilog code for 4-bit universal shift register and verify the design by simulation.**

Circuit



Source Code

```
module universalshiftreg(input clk,reset,input [1:0] mode,input [3:0]
in,leftshift,rightshift,output reg [3:0] q);
always @(posedge clk or posedge reset) begin
if (reset) begin
q <= 4'b0000;
end else begin
```

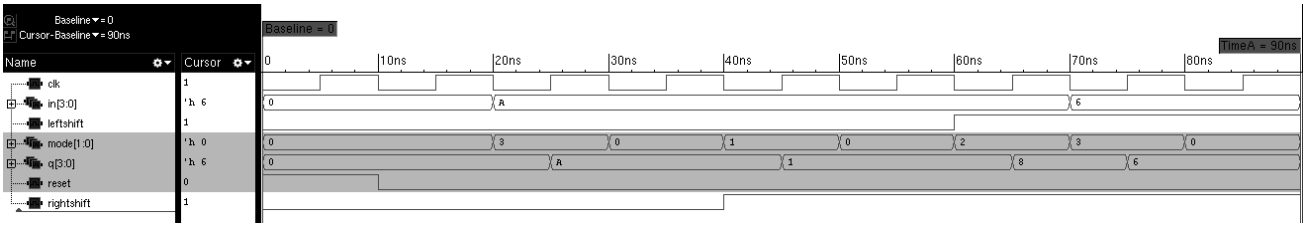
```
 case (mode)
 2'b00: q <= q; //hold
 2'b01: q <= {q[2:0], rightshift};
 2'b10: q <= {leftshift, q[3:1]};
 2'b11: q <= in; //load
 default: q <= q;
endcase
end
end
endmodule
```

Testbench

```
module tb();
reg clk,reset;
reg [1:0] mode;
reg [3:0] in;
reg leftshift,rightshift;
wire [3:0] q;
universalshiftreg
uut(.clk(clk),.reset(reset),.mode(mode),.in(in),.leftshift(leftshift),.ri
ghtshift(rightshift),.q(q));
always #5 clk = ~clk;
initial begin
clk = 0;
reset = 1;
mode = 2'b00;
in = 4'b0000;
leftshift = 0;
rightshift = 0;
#10 reset = 0;
#10 mode = 2'b11;
in = 4'b1010;
#10 mode = 2'b00;
#10 mode = 2'b01;
rightshift = 1;
#10 mode = 2'b00;
#10 mode = 2'b10;
leftshift = 1;
```

```
#10 mode = 2'b11;
in = 4'b0110;
#10 mode = 2'b00;
#10 $finish;
end
initial begin
$monitor($time, " mode = %b  q = %b",mode, q);
end
endmodule
```

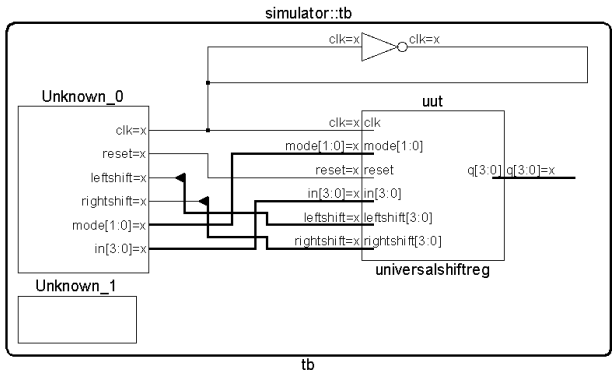
Waveform



Console

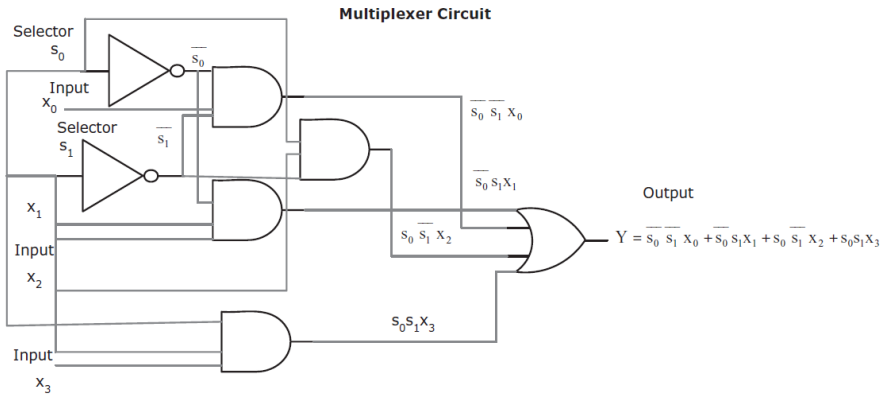
```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.clk tb.in tb.leftshift tb.mode tb.q tb.reset tb.rightshift
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                    0 mode = 00  q = 0000
                         20 mode = 11  q = 0000
                         25 mode = 11  q = 1010
                         30 mode = 00  q = 1010
                         40 mode = 01  q = 1010
                         45 mode = 01  q = 0001
                         50 mode = 00  q = 0001
                         60 mode = 10  q = 0001
                         65 mode = 10  q = 1000
                         70 mode = 11  q = 1000
                         75 mode = 11  q = 0110
                         80 mode = 00  q = 0110
Simulation complete via $finish(1) at time 90 NS + 0
./universalshiftreg_tb.v:29 #10 $finish;
ncsim>
```

Schematic



Open Ended Problems

1. **Develop the output functions for the multiplexer circuit in Figure**



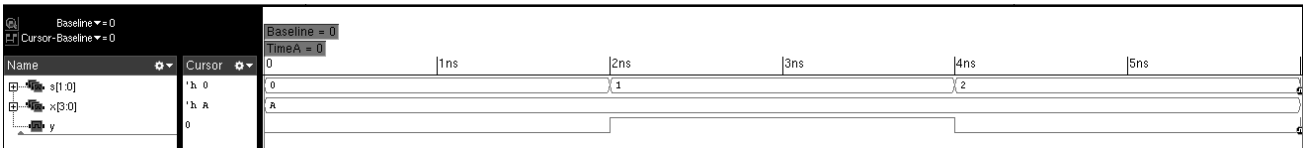
Source Code

```
module mux41(input [3:0]x,input [1:0]s,output y);
wire w1,w2,w3,w4;
and(w1,~s[1],~s[0],x[0]);
and(w2,~s[1],s[0],x[1]);
and(w3,s[1],~s[0],x[2]);
and(w4,s[1],s[0],x[3]);
or(y,w1,w2,w3,w4);
endmodule
```

Testbench

```
module tb();
reg [3:0]x;
reg [1:0]s;
wire y;
mux41 uut(.x(x),.s(s),.y(y));
initial begin
x=4'b1010; s=2'd0;
#2 x=4'b1010; s=2'd1;
#2 x=4'b1010; s=2'd2;
#2 x=4'b1010; s=2'd3;
end
initial begin
$monitor($time,"x=%b,s=%d,y=%b",x,s,y);
end
endmodule
```

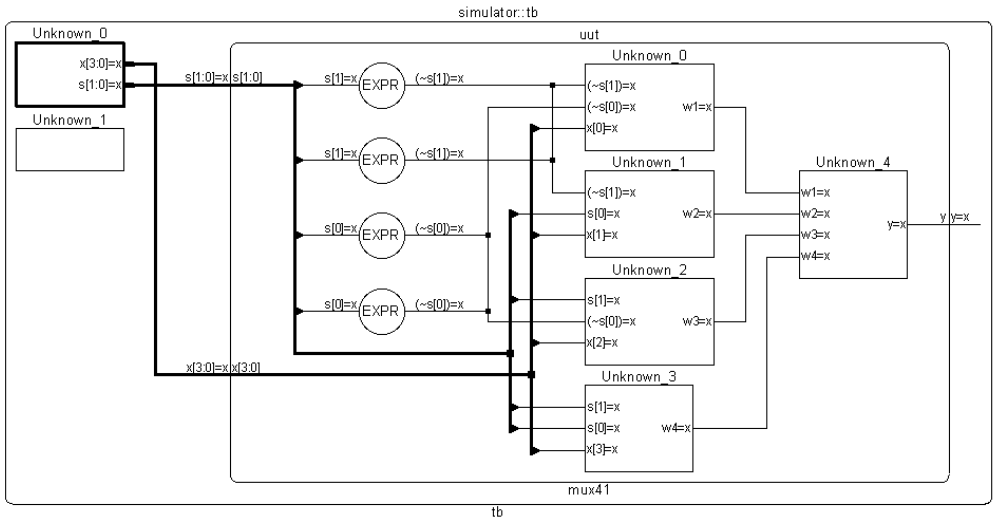
Waveform



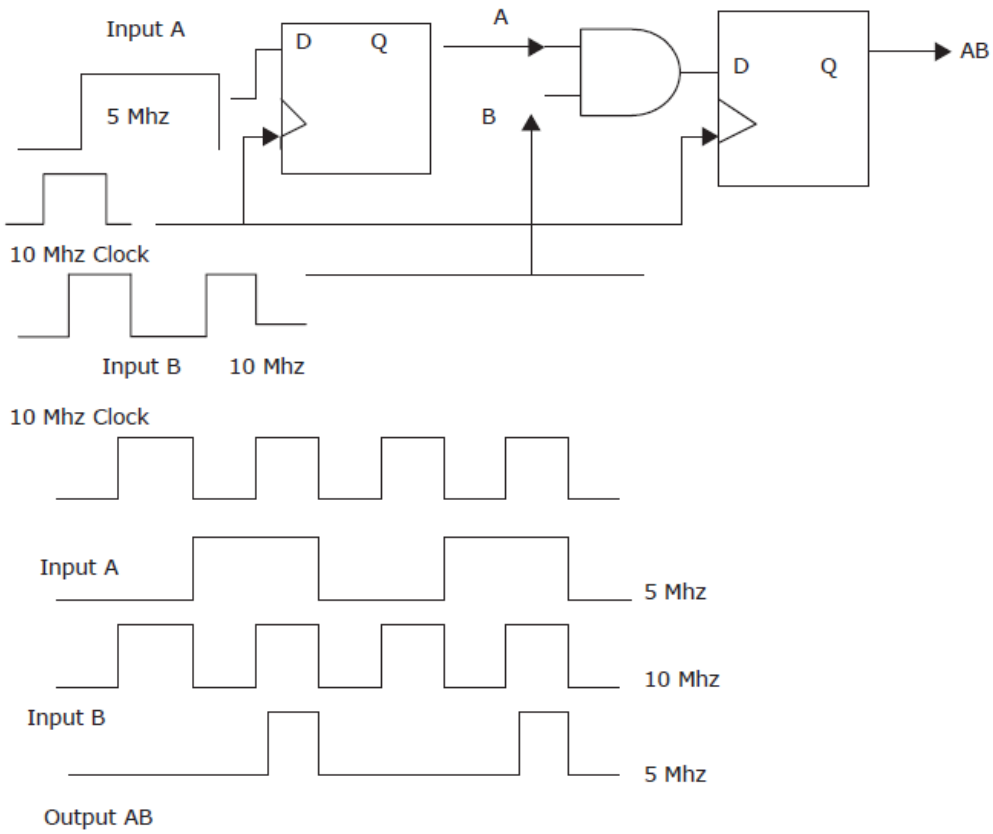
Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.s tb.x tb.y
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                      0x=1010, s=0, y=0
                   2x=1010, s=1, y=1
                   4x=1010, s=2, y=0
                   6x=1010, s=3, y=1
ncsim: *W,RNQUIE: Simulation is complete.
ncsim>
```

Schematic



2. **What is the timing diagram for input signals A and B and what is the value of the output?**



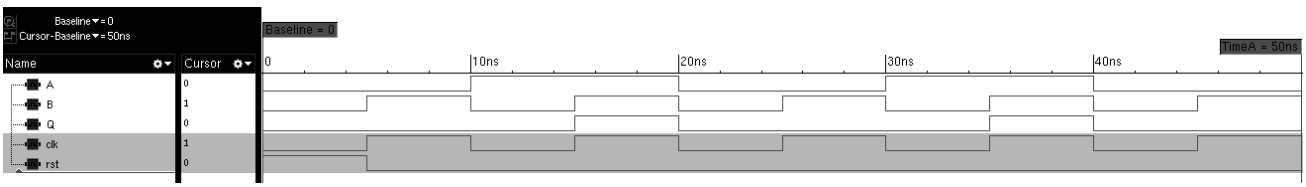
Source Code

```
module dffab(input clk,rst,A,B,output reg Q);
reg w1;
always@(negedge clk)
begin
if(rst)
w1<=0;
else
w1<=A;
end
always@(*)
begin
if(rst)
Q<=0;
else
Q=A&B;
end
endmodule
```

Testbench

```
module tb();
reg clk,rst,A,B;
wire Q;
dffab uut(.clk(clk),.A(A),.B(B),.rst(rst),.Q(Q));
initial begin
clk=0;
forever #5 clk=~clk;
end
initial begin
rst=1; #5;
rst=0;
end
initial begin
A=0;
forever #10 A=~A;
end
initial begin
B=0;
forever #5 B=~B;
end
initial begin
$monitor("clk=%b,rst=%b,A=%b,B=%b",clk,rst,A,B);
#50 $finish;
end
endmodule
```

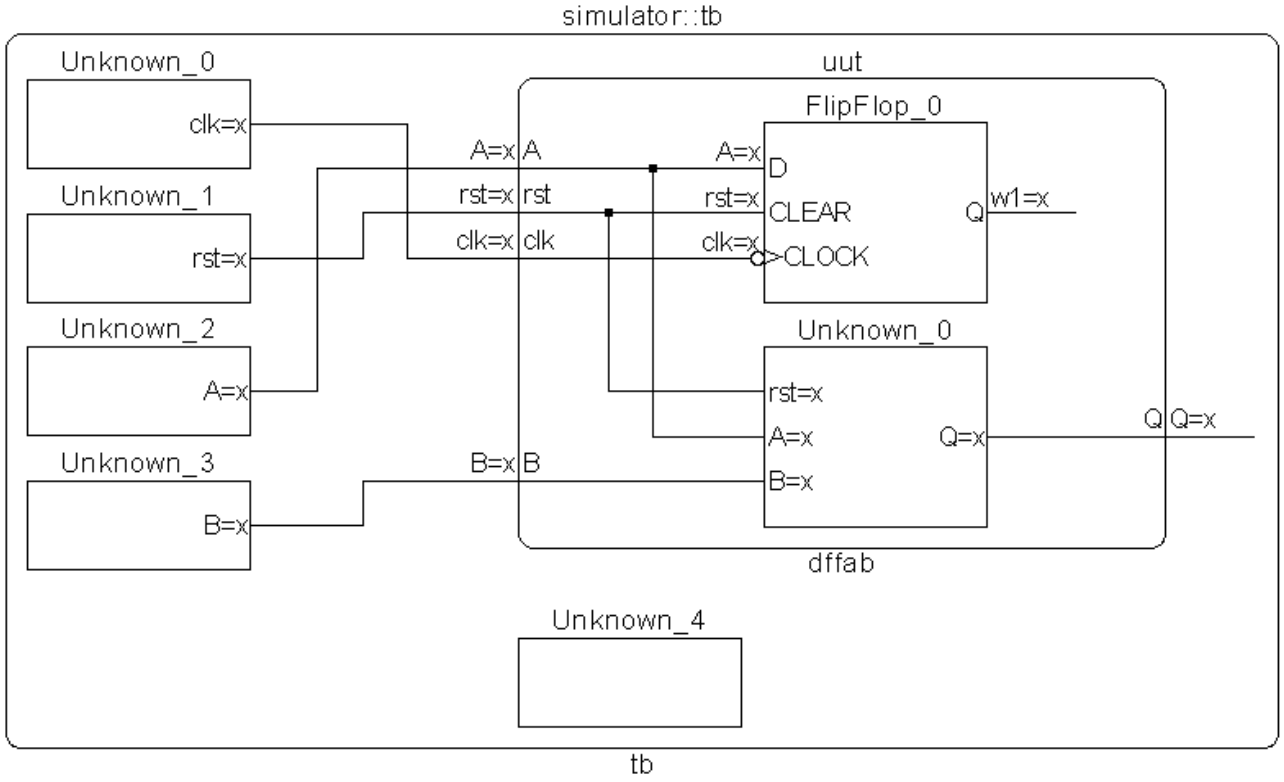
Waveform



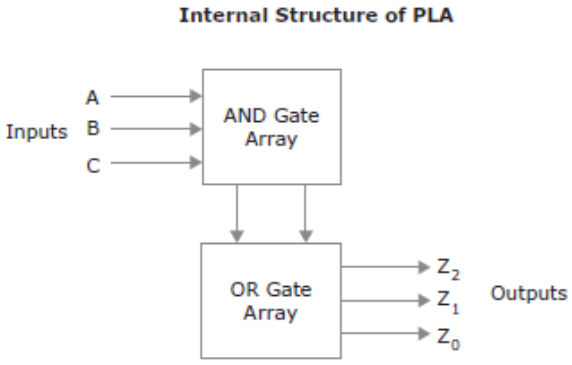
Console

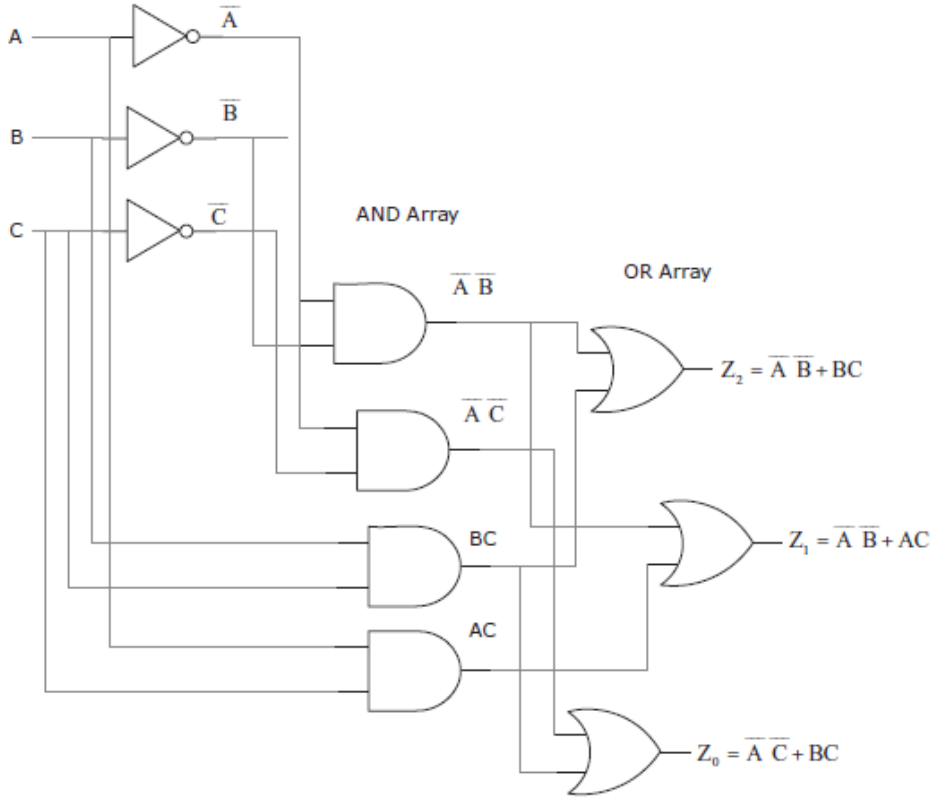
```
ncsim>
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.A tb.B tb.Q tb.clk tb.rst
run
Created default SHM database waves
ncsim> Created probe 1
ncsim> clk=0, rst=1, A=0, B=0
clk=1, rst=0, A=0, B=1
clk=0, rst=0, A=1, B=0
clk=1, rst=0, A=1, B=1
clk=0, rst=0, A=0, B=0
clk=1, rst=0, A=0, B=1
clk=0, rst=0, A=1, B=0
clk=1, rst=0, A=1, B=1
clk=0, rst=0, A=0, B=0
clk=1, rst=0, A=0, B=1
Simulation complete via $finish(1) at time 50 NS + 0
./dffab_tb.v:23 #50 $finish;
ncsim>
```

Schematic



3. Draw a logic diagram showing how the PLA should be designed to implement the functions Z2 = A'B' + BC, Z1 = A'B' + AC, Z0 = A'B' + BC.





Source Code

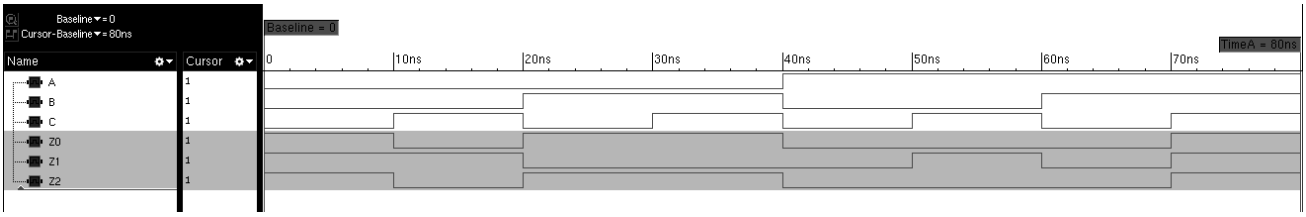
```
module pla(
input wire A,input wire B,input wire C,output reg Z0,output reg Z1,output
reg Z2);
wire A_bar = ~A;
wire B_bar = ~B;
wire C_bar = ~C;
```

```
wire P1 = A_bar & B_bar;
wire P2 = B & C;
wire P3 = A & C;
wire P4 = A_bar & C_bar;
always@(*) begin
 Z0 = P4 | P2;
 Z1 = P1 | P3;
 Z2 = P4 | P2;
end
endmodule
```

Testbench

```
module tb();
reg A;
reg B;
reg C;
wire Z2;
wire Z1;
wire Z0;
pla uut(.A(A),.B(B),.C(C),.Z2(Z2),.Z1(Z1),.Z0(Z0));
initial begin
A=0;B=0;C=0;#10;
A=0;B=0;C=1;#10;
A=0;B=1;C=0;#10;
A=0;B=1;C=1;#10;
A=1;B=0;C=0;#10;
A=1;B=0;C=1;#10;
A=1;B=1;C=0;#10;
A=1;B=1;C=1;#10;
$finish();
end
initial begin
$monitor($time," A= %b, B= %b, C=%b, Z2=%b, Z1=%b,
Z0=%b",A,B,C,Z2,Z1,Z0);
end
endmodule
```

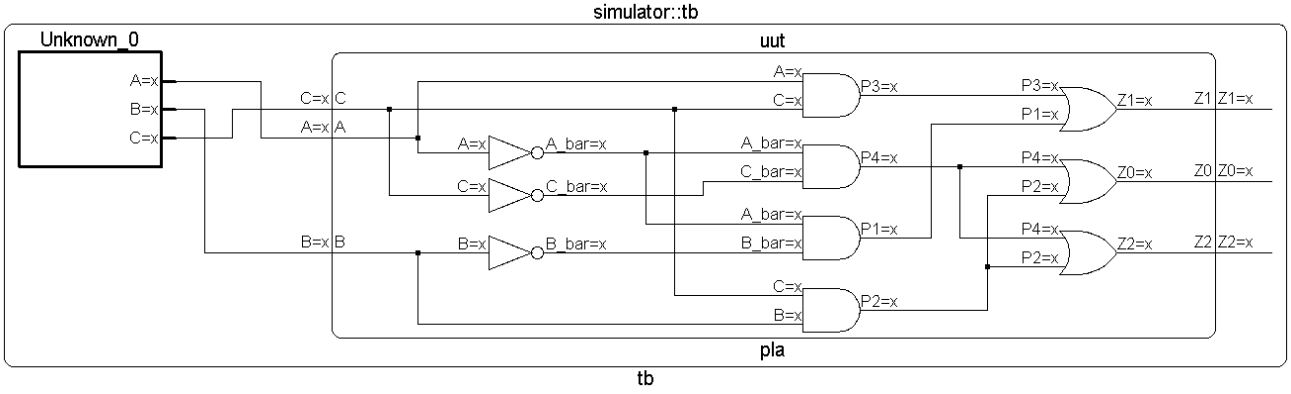
Waveform



Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.A tb.B tb.C tb.Z0 tb.Z1 tb.Z2
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                    0 A= 0, B= 0, C=0, Z2=1, Z1=1, Z0=1
                 10 A= 0, B= 0, C=1, Z2=0, Z1=1, Z0=0
                 20 A= 0, B= 1, C=0, Z2=1, Z1=0, Z0=1
                 30 A= 0, B= 1, C=1, Z2=1, Z1=0, Z0=1
                 40 A= 1, B= 0, C=0, Z2=0, Z1=0, Z0=0
                 50 A= 1, B= 0, C=1, Z2=0, Z1=1, Z0=0
                 60 A= 1, B= 1, C=0, Z2=0, Z1=0, Z0=0
                 70 A= 1, B= 1, C=1, Z2=1, Z1=1, Z0=1
Simulation complete via $finish(1) at time 80 NS + 0
./pla_tb.v:21 $finish();
ncsim>
```

Schematic



4. Implement the given FSM.

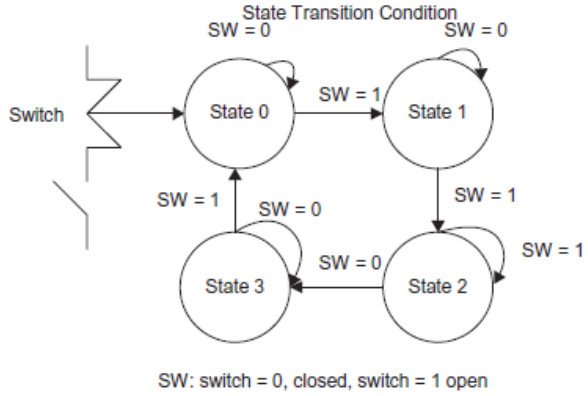


Table Switch State Transition Table

Present switch condition	Present state	State transition condition	Next state
SW = 0	0	SW = 0	0
SW = 0	0	SW = 1	1
SW = 1	1	SW = 0	1
SW = 0	1	SW = 1	2
SW = 1	2	SW = 1	2
SW = 1	2	SW = 0	3
SW = 0	3	SW = 0	3
SW = 0	3	SW = 1	0

Source Code

```
module fsm(input clk,rst,sw);
parameter s0=4'b0000,s1=4'b0001,s2=4'b0010,s3=4'b0011;
reg [3:0] cs,ns;
always@(posedge clk or negedge rst)
begin
cs<=rst?cs:ns;
end
always@(*)
begin
case(cs)
s0:
begin
if(sw)
ns<=s1;
else
ns<=s0;
end
```

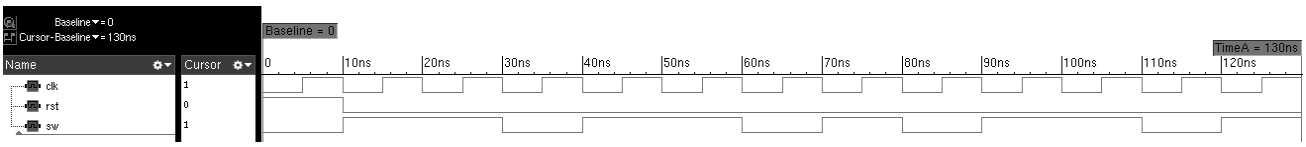
```
s1:
begin
if(sw)
ns<=s2;
else
ns<=s1;
end
s2:
begin
if(sw)
ns<=s2;
else
ns<=s3;
end
s3:
begin
if(sw)
ns<=s0;
else
ns<=s3;
end
default:ns<=s0;
endcase
end
endmodule
```

Testbench

```
module tb();
reg clk,rst,sw;
fsm uut(clk,rst,sw);
initial begin
clk=0;
forever #5 clk=~clk;
end
initial begin
rst=1; #10;
rst=0;
```

```
end
initial begin
sw=0; #10; sw=1; #10;
sw=1; #10; sw=0; #10;
sw=1; #10; sw=1; #10;
sw=0; #10; sw=1; #10;
sw=0; #10; sw=1; #10;
sw=1; #10; sw=0; #10;
sw=1; #10
$finish;
end
initial begin
$monitor("clk=%b,rst=%b,sw=%b",clk,rst,sw);
end
endmodule
```

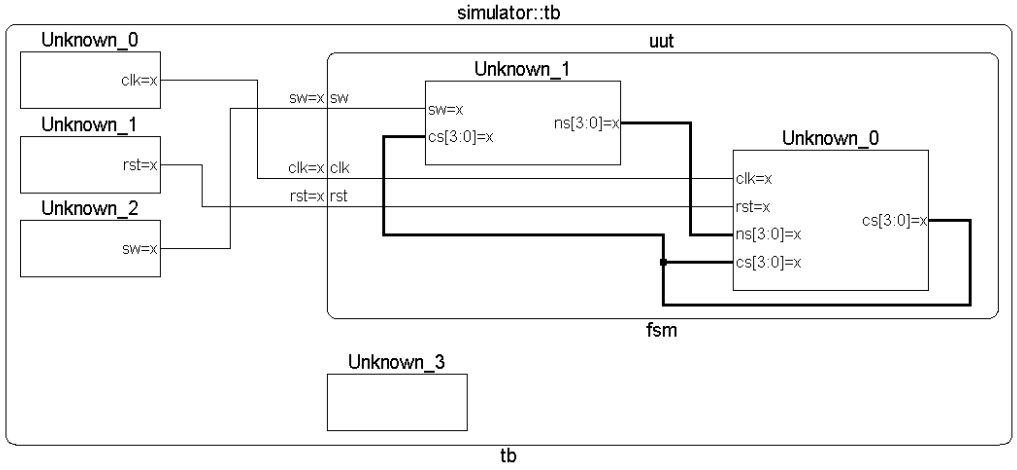
Waveform



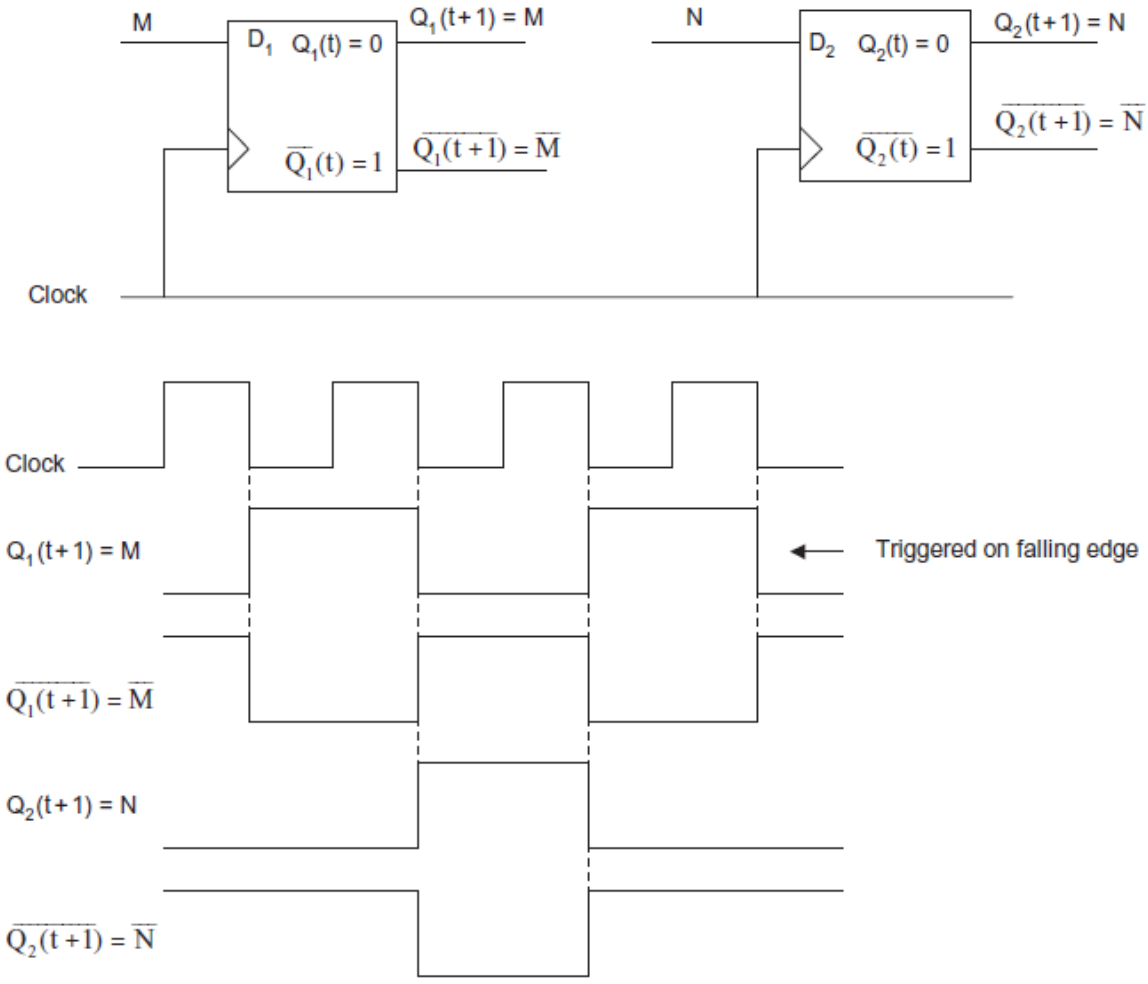
Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.clk tb.rst tb.sw
run
Created default SHM database waves
ncsim> Created probe 1
ncsim> clk=0,rst=1,sw=0
clk=1,rst=1,sw=0
clk=0,rst=0,sw=1
clk=1,rst=0,sw=1
clk=0,rst=0,sw=1
clk=1,rst=0,sw=1
clk=0,rst=0,sw=0
clk=1,rst=0,sw=0
clk=0,rst=0,sw=1
clk=1,rst=0,sw=1
clk=0,rst=0,sw=1
clk=1,rst=0,sw=1
clk=0,rst=0,sw=0
clk=1,rst=0,sw=0
clk=0,rst=0,sw=1
clk=1,rst=0,sw=1
clk=0,rst=0,sw=0
clk=1,rst=0,sw=0
clk=0,rst=0,sw=1
clk=1,rst=0,sw=1
clk=0,rst=0,sw=1
clk=1,rst=0,sw=1
clk=0,rst=0,sw=0
clk=1,rst=0,sw=0
clk=0,rst=0,sw=1
clk=1,rst=0,sw=1
Simulation complete via $finish(1) at time 130 NS + 0
./fsm_tb.v:24 $finish;
ncsim>
```

Schematic



5. Given the characteristic of the D flip - flop, draw the timing diagram for the circuit.



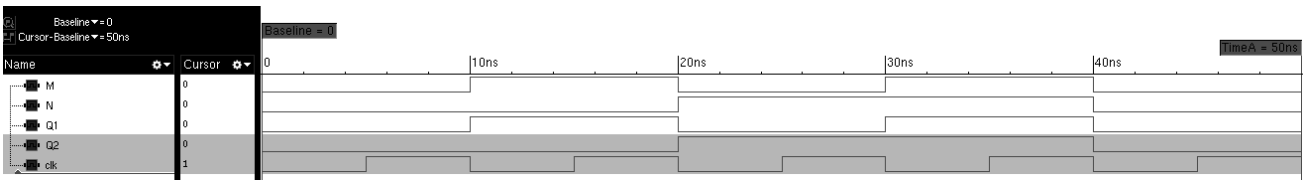
Source Code

```
module dff(input wire clk,M,N,output reg Q1,Q2);
always@(negedge clk)begin
Q1 <= M;
Q2 <= N;
end
endmodule
```

Testbench

```
module tb();
reg clk,M,N;
wire Q1,Q2;
dff uut(.clk(clk),.M(M),.N(N),.Q1(Q1),.Q2(Q2));
initial begin
clk=0;
forever #5 clk = ~clk;
end
initial begin
M=0;
N=0;
#10 M=1;N=0;
#10 M=0;N=1;
#10 M=1;N=1;
#10 M=0;N=0;
#10;$finish();
end
initial begin
$monitor($time," clk= %b, M= %b, N=%b, Q1=%b, Q2=%b",clk,M,N,Q1,Q2);
end
endmodule
```

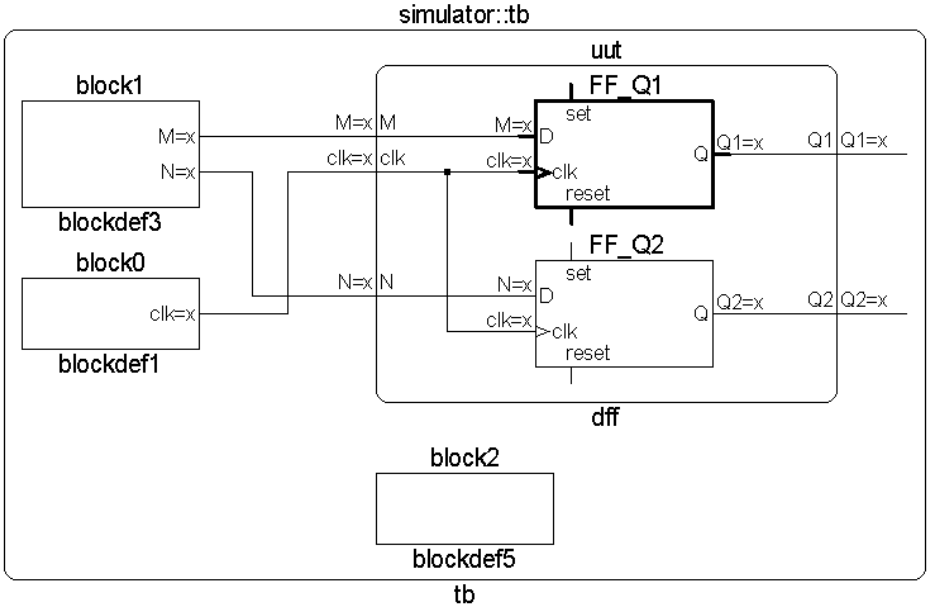
Waveform



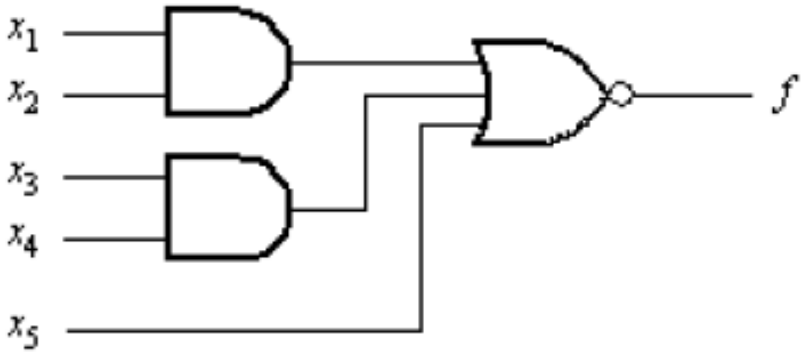
Console

```
ncsim>
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.M tb.N tb.Q1 tb.Q2 tb.clk
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                      0 clk= 0, M= 0, N=0, Q1=0, Q2=0
                    5 clk= 1, M= 0, N=0, Q1=0, Q2=0
                   10 clk= 0, M= 1, N=0, Q1=1, Q2=0
                   15 clk= 1, M= 1, N=0, Q1=1, Q2=0
                   20 clk= 0, M= 0, N=1, Q1=0, Q2=1
                   25 clk= 1, M= 0, N=1, Q1=0, Q2=1
                   30 clk= 0, M= 1, N=1, Q1=1, Q2=1
                   35 clk= 1, M= 1, N=1, Q1=1, Q2=1
                   40 clk= 0, M= 0, N=0, Q1=0, Q2=0
                   45 clk= 1, M= 0, N=0, Q1=0, Q2=0
Simulation complete via $finish(1) at time 50 NS + 0
./dff_tb.v:16 #10;$finish();
ncsim>
```

Schematic



6. **In standard cell technology, circuits are built by interconnecting building-block cells that implement simple functions, like basic logic gates. A commonly used type of standard cell are the and-or-invert (AOI) cells, which can be efficiently built as CMOS complex gates. Consider the AOI cell shown in Figure. This cell implements the function f = x1x2 + x3x4 + x5. Derive the CMOS complex gate that implements this cell.**



Source Code

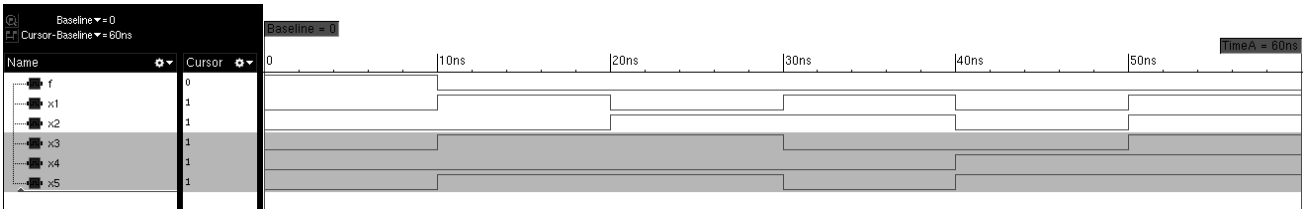
```
module aoi(
input wire x1,
input wire x2,
input wire x3,
input wire x4,
input wire x5,
output reg f
);
wire P1 = x1 & x2;
wire P2 = x3 & x4;
wire P3 = P1 | P2 | x5;
always@(*) begin
f = ~P3;
end
endmodule
```

Testbench

```
module tb();
reg  x1;
reg  x2;
reg  x3;
```

```
reg  x4;
reg  x5;
wire f;
aoi uut (.x1(x1),.x2(x2),.x3(x3),.x4(x4),.x5(x5),.f(f));
initial begin
x1=0;x2=0;x3=0;x4=0;x5=0;#10;
x1=1;x2=0;x3=1;x4=0;x5=1;#10;
x1=0;x2=1;x3=1;x4=0;x5=1;#10;
x1=1;x2=1;x3=0;x4=0;x5=0;#10;
x1=0;x2=0;x3=0;x4=1;x5=1;#10;
x1=1;x2=1;x3=1;x4=1;x5=1;#10;
$finish();
end
initial begin
$monitor($time," x1= %b, x2= %b, x3=%b, x4=%b, x5=%b,
f=%b",x1,x2,x3,x4,x5,f);
end
endmodule
```

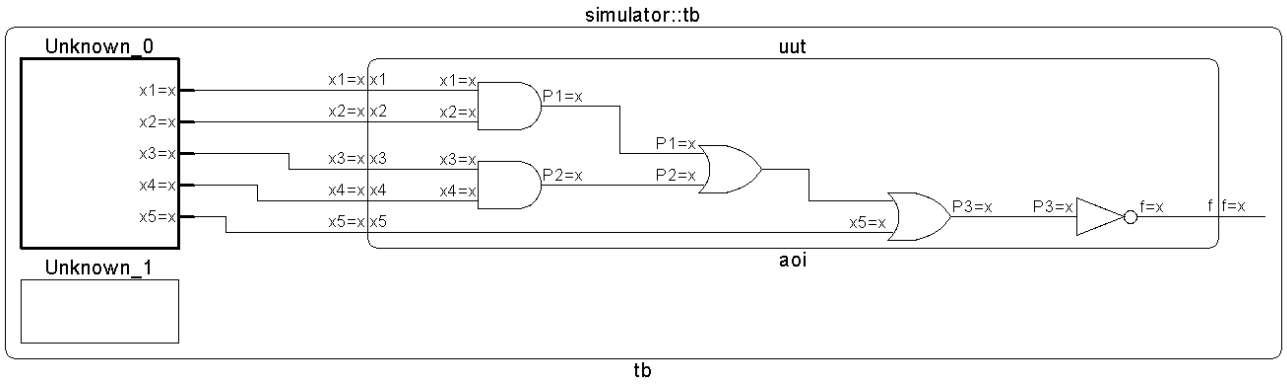
Waveform



Console

```
ncsim>
ncsim> source /home/install/INCISIVE152/tools/inca/files/ncsimrc
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.f tb.x1 tb.x2 tb.x3 tb.x4 tb.x5
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                    0 x1= 0, x2= 0, x3=0, x4=0, x5=0, f=1
                 10 x1= 1, x2= 0, x3=1, x4=0, x5=1, f=0
                 20 x1= 0, x2= 1, x3=1, x4=0, x5=1, f=0
                 30 x1= 1, x2= 1, x3=0, x4=0, x5=0, f=0
                 40 x1= 0, x2= 0, x3=0, x4=1, x5=1, f=0
                 50 x1= 1, x2= 1, x3=1, x4=1, x5=1, f=0
Simulation complete via $finish(1) at time 60 NS + 0
./aoi_tb.v:18 $finish();
ncsim>
```

Schematic



7. **In computer computations it is often necessary to compare numbers. Two four-bit signed numbers, X = x3x2x1x0 and Y = y3y2y1y0, can be compared by using the subtractor circuit in Figure 5.43, which performs the operation X−Y. The three outputs denote the following:**
   - **Z = 1 if the result is 0; otherwise, Z = 0**
   - **N = 1 if the result is negative; otherwise, N = 0**
   - **V = 1 if arithmetic overflow occurs; otherwise, V = 0**

   Show how Z, N, and V can be used to determine the cases X = Y, XY, and X ≥ Y.

Source Code

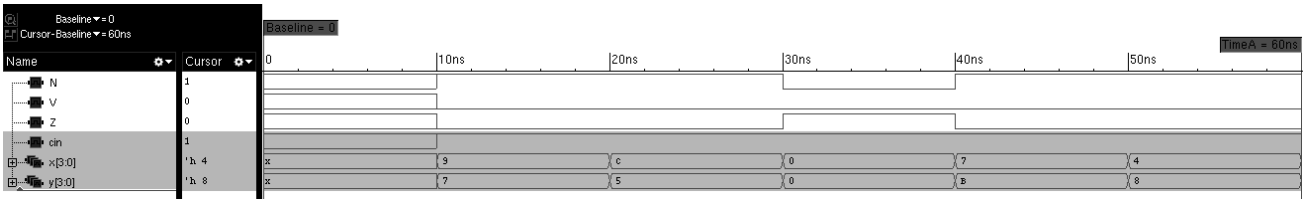
```
module comp(x,y,cin,sum,cout);
input x,y,cin;
output sum,cout;
assign sum = x^y^cin;
assign cout = (x^y)|cin;
endmodule
module com(input [3:0]x ,input [3:0]y ,input cin,output N,output Z,
output V);
wire w1 = ~y[0];
wire w2 = ~y[1];
wire w3 = ~y[2];
wire w4 = ~y[3];
comp f1(x[0],w1,cin,sum1,cin2);
comp f2(x[1],w2,cin2,sum2,cin3);
comp f3(x[2],w3,cin3,sum3,cin4);
comp f4(x[3],w4,cin4,sum4,cout);
```

```
nor n1(Z,sum1,sum2,sum3,sum4);
assign N = sum4;
xor ex(V,cin4,cout);
endmodule
```

Testbench

```
module tb();
reg [3:0]x;
reg [3:0]y;
reg cin;
wire Z;
wire N;
wire V;
com uut(x,y,cin,N,Z,V);
initial begin
#10;x=4'b1001;y=4'b0111;cin=1;
#10;x=4'b1100;y=4'b0101;cin=1;
#10;x=4'b0000;y=4'b0000;cin=1;
#10;x=4'b0111;y=4'b1011;cin=1;
#10;x=4'b0100;y=4'b1000;cin=1;
#10;$finish();
end
initial begin
$monitor($time," x= %b, y= %b, cin=%b, N=%b, Z=%b, V=%b",x,y,cin,N,Z,V);
end
endmodule
```

Waveform



Console

```
ncsim>
ncsim> run
database -open waves -into waves.shm -default
probe -create -shm tb.N tb.V tb.Z tb.cin tb.x tb.y
run
Created default SHM database waves
ncsim> Created probe 1
ncsim>                     0 x= xxxx, y= xxxx, cin=x, N=x, Z=x, V=x
                  10 x= 1001, y= 0111, cin=1, N=1, Z=0, V=0
                  20 x= 1100, y= 0101, cin=1, N=1, Z=0, V=0
                  30 x= 0000, y= 0000, cin=1, N=0, Z=1, V=0
                  40 x= 0111, y= 1011, cin=1, N=1, Z=0, V=0
                  50 x= 0100, y= 1000, cin=1, N=1, Z=0, V=0
Simulation complete via $finish(1) at time 60 NS + 0
./comp_tb.v:17 #10;$finish();
ncsim>
```

Schematic