**LAB 1:**

**i.RTL CODE FOR HALF ADDER USING DATA FLOW MODEL**

module half\_adder1(input a,b,output s,c);

assign s = a^b;

assign c = a&b;

endmodule

**TEST BENCH CODE FOR HALF ADDER:**

module half\_adder1\_tb();

reg a,b;

wire s,c;

integer i;

half\_adder1 dut(.a(a),.b(b),.s(s),.c(c));

initial

begin

for(i=0;i<=3;i=i+1)

begin

{a,b}=i;

#10;

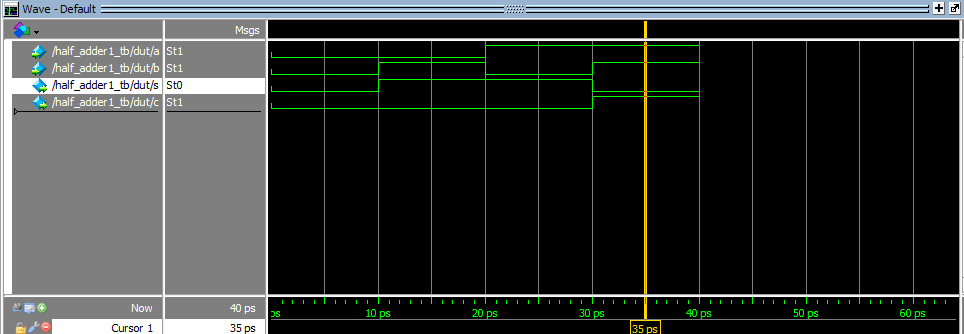
end

end

initial $monitor("input a=%b,b=%b,output s=%b,c=%b",a,b,s,c);

endmodule

**SIMULATION:**



**ii.WRITE VERILOG CODE FOR 1BIT FULL ADDER USING 2 HALFADDER AND ONE OR GATE:**

**RTL CODE:**

module fullAdder(input a,b,cin,output sum,carry);

wire w1,w2,w3;

half\_adder HA1(a,b,w1,w2);

half\_adder HA2(w1,cin,sum,w3);

or N(carry,w2,w3);

endmodule

module half\_adder(input a,b,

output sum,carry);

assign sum = a ^ b;

assign carry = a & b;

endmodule

**TESTBENCH CODE:**

module full\_tb();

reg a,b,cin;

wire sum,carry;

integer i;

full\_adder dut(.a(a),.b(b),.c(cin),.sum(sum),.carry(carry));

initial

begin

a = 1'b0;

b = 1'b0;

cin = 1'b0;

end

initial

begin

for (i=0;i<8;i=i+1)

begin

{a,b,cin}=i;

#10;

end

end

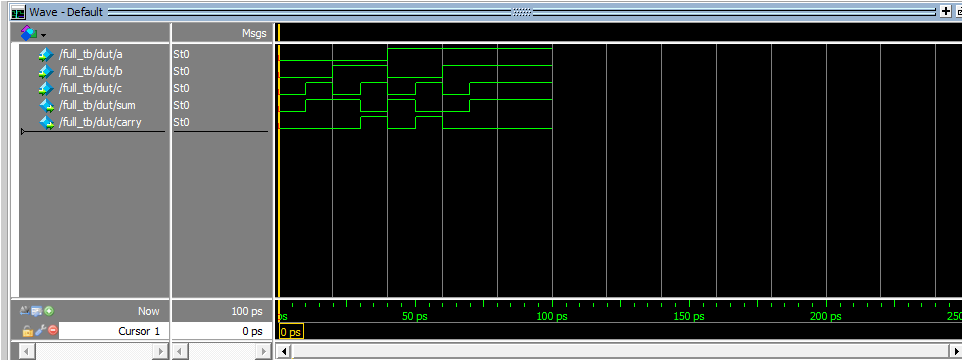
initial

$monitor("Input a=%b, b=%b, c=%b, Output sum =%b, carry=%b",a,b,cin,sum,carry);

initial #100 $finish;

endmodule

SIMULATION:



**iii.RTL CODE FOR FULL ADDER USING DATA FLOW MODEL**

module full\_adder(input a,b,c,output sum,carry);

assign sum = a^b^c;

assign carry = c&(a^b);

endmodule

**TEST BENCH CODE FOR FULL ADDER**

module full\_adder\_tb();

reg a,b,cin;

wire sum,carry;

integer i;

full\_adder DUT(a, b,cin, sum, carry);

initial

begin

a = 1'b0;

b = 1'b0;

cin = 1'b0;

end

initial

begin

for(i=0;i<8;i=i+1)

begin

{a,b,cin}=i;

#10;

end

end

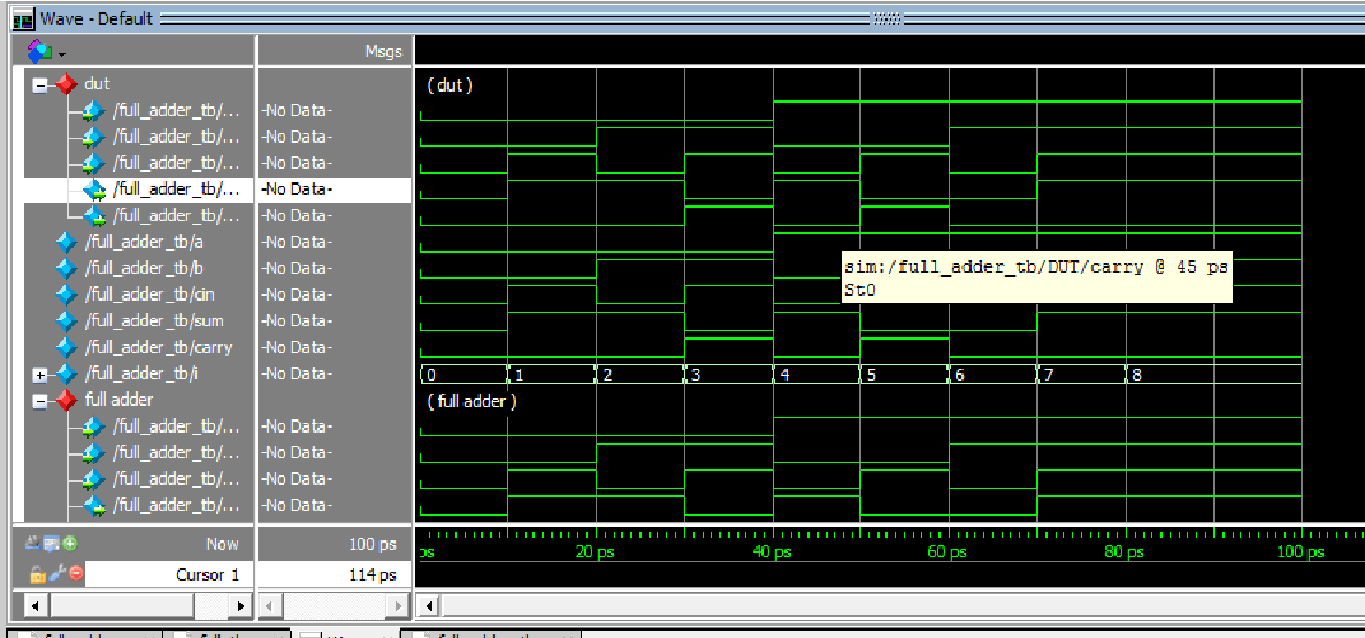
initial

$monitor("Input a=%b, b=%b, c=%b, Output sum =%b, carry=%b",a,b,cin,sum,carry);

initial #100 $finish;

endmodule

**Simulation:**



**iv.RTL Code for 2:4 decoder using data flow model**

module decoder2(input a,b,output y0,y1,y2,y3);

assign y0 = ((~a) & (~b));

assign y1= ((~a) & (b));

assign y2 = ((a)& (~b));

assign y3 = (a & b);

endmodule

**TEST BENCH Code for 2:4 decoder using data flow model**

module decoder\_tb();

wire y0,y1,y2,y3;

reg a,b;

integer i;

decoder2 dut(.a(a),.b(b),.y0(y0),.y1(y1),.y2(y2),.y3(y3));

initial

begin

for(i=0;i<=3;i=i+1)

begin

{a,b} = i;

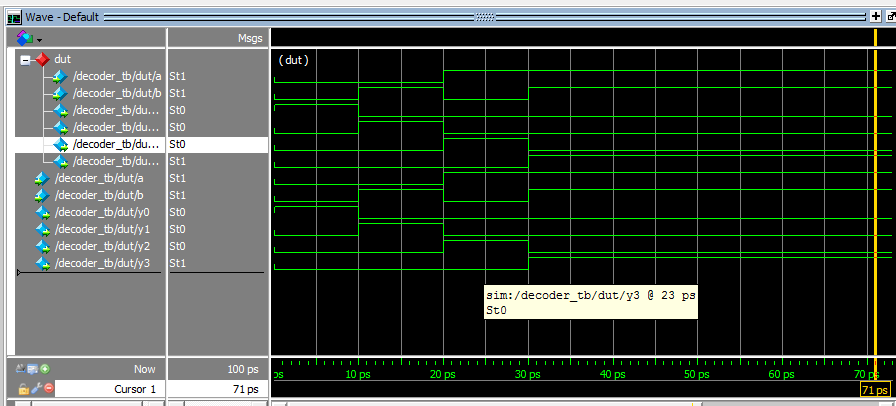
#10;

end

end

initial $monitor("Input a=%b, b=%b , Output y0 =%b, y1=%b, y2=%b, y3=%b",a,b,y0,y1,y2,y3);

initial #100 $finish; endmodule

**simulation**

v.4:1 mux using 2:1 mux

**RTL CODE:**

module mux\_4(input i0,i1,i2,i3,s0,s1,output w1,w2,y);

mux2 M1(i0,i1,s1,w1);

mux2 M2(i2,i3,s1,w2);

mux2 M3(w1,w2,s0,y);

endmodule

**TEST BENCH CODE:**

module mux\_4\_tb();

wire y,w1,w2;

reg i0,i1,i2,i3,s0,s1;

integer i;

mux\_4 dut(i0,i1,i2,i3,s0,s1,w1,w2,y);

initial

begin

assign i0 = 1'b1;

assign i1 = 1'b0;

assign i2 = 1'b0;

assign i3 = 1'b1;

end

initial

begin

for(i=0;i<=3;i=i+1)

begin

{s0,s1} =i;

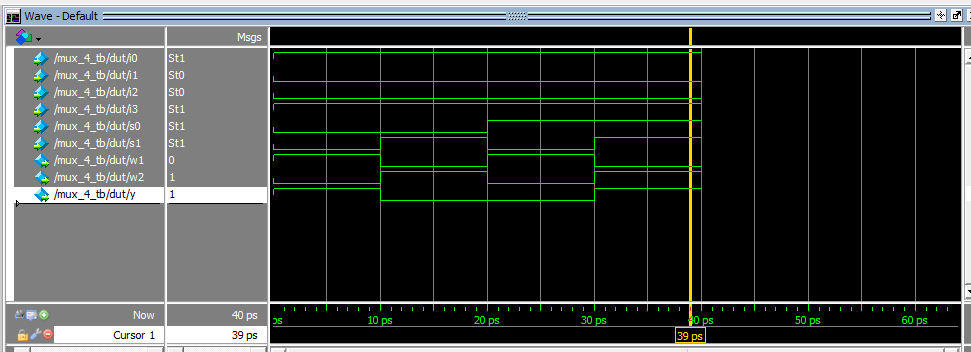
#10;

end

end

endmodule

**Simulation:**

****

**vi.**BIDIRECTIONAL BUFFER:

RTL CODE:

module bufif(c,a,b);

input c;

inout a;

inout b;

bufif1(b,a,c);

bufif0(a,b,c);

endmodule

TESTBENCH CODE:

module bufif\_tb();

reg w1,w2,c;

wire a,b;

integer i;

assign a=c?w1:1'bz;

assign b=~c?w2:1'bz;

bufif dut(c,a,b);

initial

begin

w1=1'b0;

w2=1'b0;

c=1'b0;

end

initial

begin

for(i=0;i<8;i=i+1)

begin

{c,w1,w2}=i;

#10;

end

end

initial

$monitor("input c=%b,w1=%b,w2=%b,a=%b,b=%b",c,w1,w2,a,b);

initial

#100

$finish;

endmodule

SIMULATION:



**4BIT FULLADDER USING 1BIT FULLADDER:**

**RTL CODE:**

module Adder4bit(A,B,Cin, Sum, Cout);

input [3:0] A,B;

input Cin;

output [3:0] Sum;

output Cout;

wire [2:0] transC;

fullAdder FA1 ( .In1(A[0]),.In2(B[0]), .Cin(Cin),.Sum(Sum[0]),.Cout(transC[0]));

fullAdder FA2 ( .In1(A[1]),.In2(B[1]),.Cin(transC[0]),.Sum(Sum[1]),.Cout(transC[1]));

fullAdder FA3 ( .In1(A[2]),.In2(B[2]),.Cin(transC[1]),.Sum(Sum[2]),.Cout(transC[2]));

fullAdder FA4 ( .In1(A[3]),.In2(B[3]),.Cin(transC[2]),.Sum(Sum[3]),.Cout(Cout));

endmodule  
**TEST BENCH CODE:**

module Adder4bit\_tb;

reg [3:0] A;

reg [3:0] B;

reg Cin;

wire [3:0] Sum;

wire Cout;

integer i;

Adder4bit dut (.A(A),.B(B),.Cin(Cin),.Sum(Sum),.Cout(Cout));

initial

begin

A = 4'b0;

B = 4'b0;

Cin = 4'b0;

#10;

end

initial

begin

for(i=0;i<+15;i=i+1)

begin

A = i;

B = i;

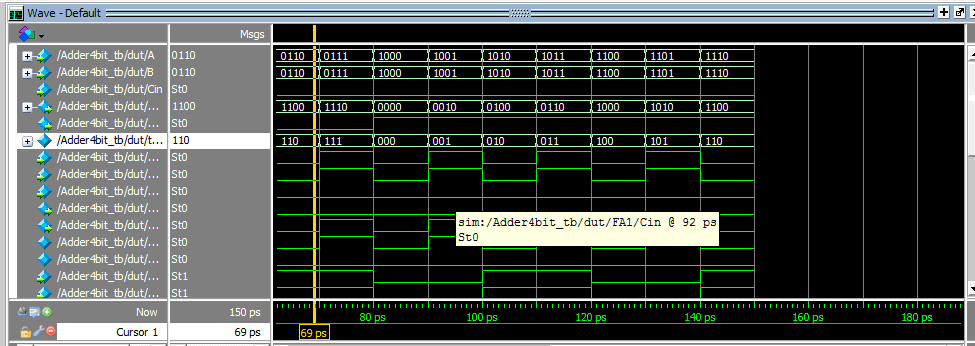
#10;

End

end

endmodule

**SIMULATION:**



**viii.4:1 MUX USING DECODER AND TRISTATE BUFFER:**

**RTL CODE:**

module dec4mux(input a,b,i1,i2,i3,i4,output z);

wire y0,y1,y2,y3,w1,w2,w3,w4;

assign y0 = (~a&~b);

assign w1 = y0&i1;

assign y1 = (~a&b);

assign w2 = y1&i2;

assign y2 = (a&~b);

assign w3 = y2&i3;

assign y3 = (a&b);

assign w4 = y3&i4;

assign z = (w1|w2|w3|w4);endmodule

**TESTBENCH CODE:**

module decmux\_tb();

reg a,b,i1,i2,i3,i4;

wire z;

integer i;

dec4mux dut(a,b,i1,i2,i3,i4,z);

initial

begin

assign i1 =1'b0;

assign i2 =1'b1;

assign i3 =1'b1;

assign i4 =1'b0;

end

initial

begin

for(i=0;i<=3;i=i+1)

begin

{a,b} = i;

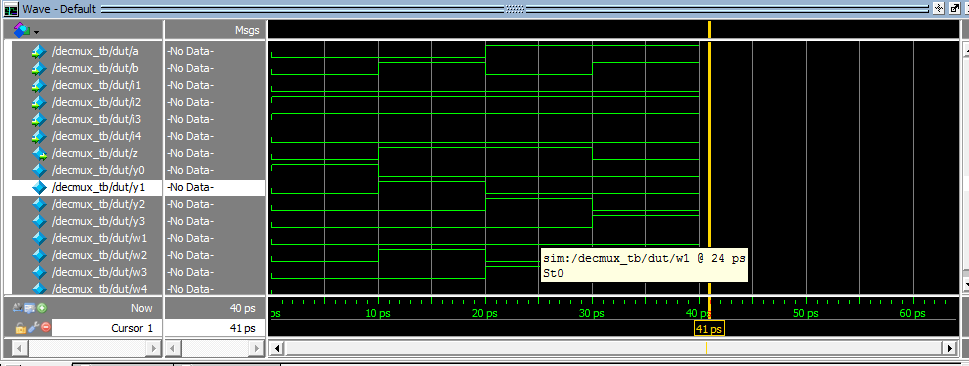
#10;

end

end

endmodule

**SIMULATION:**



ix.8:3PRIORITY ENCODER USING STRUCTURAL MODEL:

RTL CODE:

module p83s(input i0,i1,i2,i3,i4,i5,i6,i7,output y0,y1,y2);

wire w0,w1,w2,w3,w4,w5,w6,w7;

pri\_cod T1(i0,i1,i2,i3,i4,i5,i6,i7,w0,w1,w2,w3,w4,w5,w6,w7);

bin\_cod T2(w0,w1,w2,w3,w4,w5,w6,w7,y0,y1,y2);

endmodule

module pri\_cod(input i0,i1,i2,i3,i4,i5,i6,i7,output h0,h1,h2,h3,h4,h5,h6,h7,idle);

wire idle;

assign h7=i7;

assign h6= (i6&(~i7));

assign h5= (i5&(~i7)&(~i6));

assign h4= (i4&(~i5)&(~i7)&(~i6));

assign h3= (i3&(~i4)&(~i5)&(~i7)&(~i6));

assign h2= (i2&(~i3)&(~i4)&(~i5)&(~i7)&(~i6));

assign h1= (i1&(~i2)&(~i3)&(~i4)&(~i5)&(~i7)&(~i6));

assign h0= (i0&(~i1)&(~i2)&(~i3)&(~i4)&(~i5)&(~i7)&(~i6));

assign idle= ((~i0)&(~i1)&(~i2)&(~i3)&(~i4)&(~i5)&(~i7)&(~i6));

endmodule

module bin\_cod(input i0,i1,i2,i3,i4,i5,i6,i7,output y0,y1,y2);

assign yo=(i1|i3|i5|i7);

assign y1=(i2|i3|i6|i7);

assign y3=(i4|i5|i6|i7);

endmodule

TESTBENCH CODE:

module p83s\_tb();

reg i0,i1,i2,i3,i4,i5,i6,i7;

wire y0,y1,y2;

p83s dut(i0,i1,i2,i3,i4,i5,i6,i7,y0,y1,y2);

initial

begin

i0=1'b1;

i7=1'b1;

#10;

i2=1'b1;

i4=1'b1;

#10;

i6=1'b1;

i4=1'b1;

#10;

i5=1'b1;

i3=1'b1;

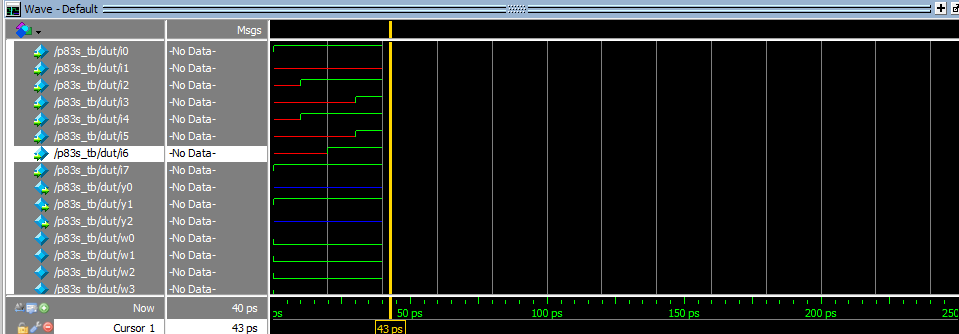
i2=1'b1;

#10;

end

endmodule

SIMULATION:



**LAB2:**

**RTL CODE FOR ALL OPERATORS:**

module alu(a,b,s,y);

input [3:0]a;

input[3:0]b;

input [4:0]s;

output reg[3:0]y;

always @(s)

begin

case(s)

5'b00000:y=a&&b;

5'b00001:y=a||b;

5'b00010:y=!b;

5'b00011:y=a&b;

5'b00100:y=a|b;

5'b00101:y=a^b;

5'b00110:y=a^~b;

5'b00111:y=~a;

5'b01000:y=&b;

5'b01001:y=|a;

5'b01010:y=^b;

5'b01011:y=~^a;

5'b01100:y=a==b;

5'b01101:y=a!=b;

5'b01110:y=a===b;

5'b01111:y=a!==b;

5'b10000:y=a>b;

5'b10001:y=a<b;

5'b10010:y=a>=b;

5'b10011:y=a<=b;

5'b10100:y=a&&b;

5'b10101:y=a>>2;

5'b10110:y=a<<2;

5'b10111:y=a>>>2;

5'b11000:y=a<<<2;

default:y=6'bxxxx;

endcase

end

endmodule

**TESTBENCH CODE:**

module alu\_tb();

reg [3:0]a;

reg [3:0]b;

reg [4:0]s;

wire [3:0]y;

integer i;

alu dut(a,b,s,y);

initial

begin

a=4'b1010;

b=4'b0101;

for(i=0;i<=24;i=i+1)

begin

s=i;

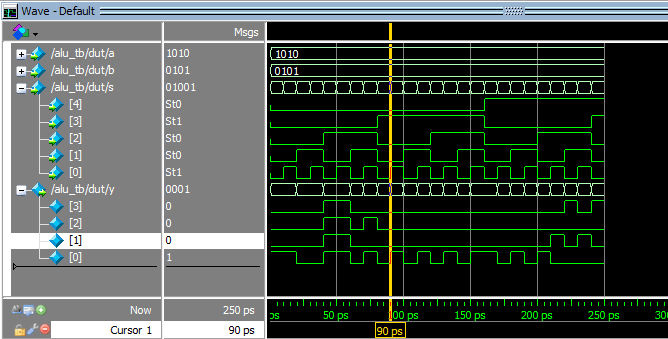
#10;

end

end

endmodule

**SIMULATION:**

****

**ii.ALU AND LOGICAL OPERATORS:**

**RTL CODE:**

module alu(input [7:0]a,b,

input [3:0]commandin,

input oe,

output [15:0]d\_out);

parameter ADD = 4'b0000,

INC = 4'b0001,

SUB = 4'b0010,

DEC = 4'b0011,

MUL = 4'b0100,

DIV = 4'b0101,

SHL = 4'b0110,

SHR = 4'b0111,

AND = 4'b1000,

OR = 4'b1001,

INV = 4'b1010,

NAND = 4'b1011,

NOR = 4'b1100,

XOR = 4'b1101,

XNOR = 4'b1110,

BUF = 4'b1111;

reg [5:0]out;

always@(commandin)

begin

case(commandin)

ADD:out = a+b;

INC:out = a+1;

SUB:out = a-b;

DEC:out = b-1;

MUL:out = a\*b;

DIV:out = a/b;

SHL:out = a<<2;

SHR:out = b>>2;

AND:out = a&b;

OR:out = a|b;

INV:out = ~b;

NAND:out = ~(a&b);

NOR:out = ~(a|b);

XOR:out = a^b;

XNOR:out = ~(a^b);

BUF:out = b;

endcase

end

assign d\_out = (oe) ? out : 16'hzzzz;

endmodule

TESTBENCH CODE:

module alu\_tb();

reg [7:0]a;

reg [7:0]b;

reg [3:0]commandin;

wire [7:0]y;

integer i;

alu dut(a,b,commandin,y);

initial

begin

a=4'b1010;

b=4'b0101;

for(i=0;i<=15;i=i+1)

begin

commandin=i;

#10;

end

end

endmodule

**SIMULATION:**

****

**LAB 3:**

**WRITE A CODE FOR 4:1 MUX USING BEHAVIOURAL MODEL**

**RTL CODE:**

module mux41beh(input reg [3:0]i,reg [1:0]s,output reg y);

always @(s,i)

begin

case(s)

2'b00:y = i[0];

2'b01:y = i[1];

2'b10:y = i[2];

2'b11:y = i[3];

default:y = 1'bx;

endcase

end

endmodule

**TESTBENCH CODE:**

module muxbeh\_tb();

reg [3:0]i;

reg [1:0]s;

wire y;

integer n;

mux41beh dut(i,s,y); //order based instantiation

initial

begin

i=4'b1010;

for(n=0;n<=3;n=n+1)

begin

s=n;

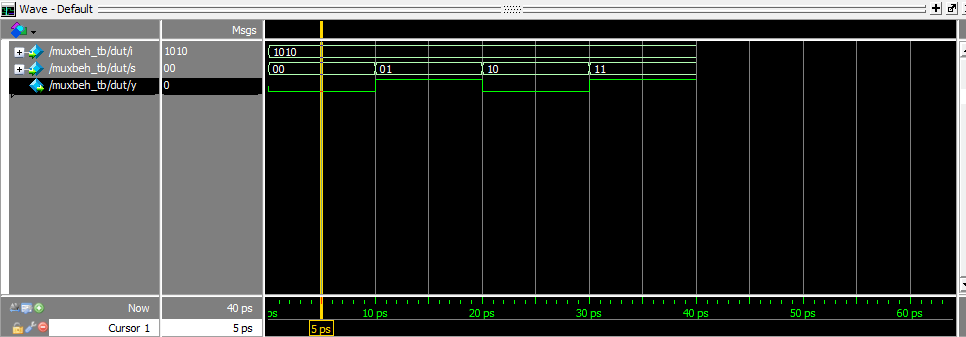
#10;

$monitor("The value [1:0]s=%b,y=%b",s,y);

end

end

endmodule

**SIMULATION:**

**ii.3:8 DECODER USING BEHAVIOURAL MODEL:**

**RTL CODE:**

module beh38(input wire [2:0]i,output reg [7:0] y);

always @(i)

begin

case(i)

3'b000:y = 8'b00000001;

3'b001:y = 8'b00000010;

3'b010:y = 8'b00000100;

3'b011:y = 8'b00001000;

3'b100:y = 8'b00010000;

3'b101:y = 8'b00100000;

3'b110:y = 8'b01000000;

3'b111:y = 8'b10000000;

default:y = 8'bxxxxxxxx;

endcase

end

endmodule

**TESTBENCH CODE:**

module beh38\_tb();

reg [2:0] i;

wire [7:0] y;

integer n;

beh38 dut(i,y);

initial

begin

for(n=0;n<=7;n=n+1)

begin

i=n;

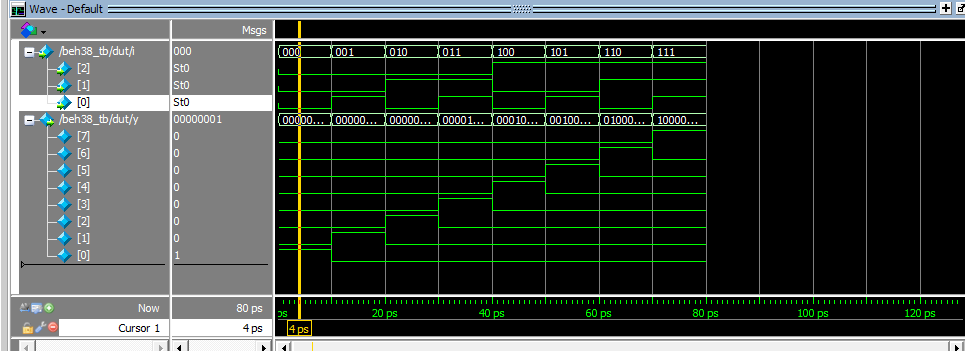
#10;

end

end

endmodule

**SIMULATION:**

****

**iii.WRITE BEHAVIOURAL DESCRIPTION FOR 8:3 PRIORITY ENCODER:**

**RTL CODE:**

module beh83p(input [7:0]i,output reg [2:0]y);

always @(i)

begin

casex(i)

8'b1xxxxxxx:y = 3'b111;

8'b01xxxxxx:y = 3'b110;

8'b001xxxxx:y = 3'b101;

8'b0001xxxx:y = 3'b100;

8'b00001xxx:y = 3'b011;

8'b000001xx:y = 3'b010;

8'b0000001x:y = 3'b001;

8'b00000001:y = 3'b000;

default: y =3'bxxx;

endcase

end

endmodule

**TESTBENCH CODE:**

module beh83p\_tb();

reg [7:0]i;

wire [2:0]y;

beh83p dut(i,y);

initial

begin

i=8'b10000000;

#10;

i=8'b01000000;

#10;

i=8'b10000000;

#10;

i=8'b00000001;

#10;

i=8'b00000101;

#10;

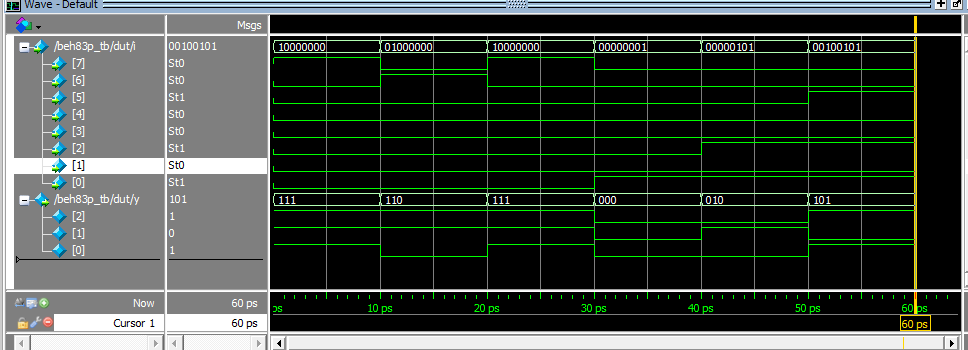
i=8'b00100101;

#10;

end

endmodule

**SIMULATION:**

****

**LAB4:**

**i.D FLIP FLOP:**

**RTL CODE:**

module dff(d,clk,q,qn);

input d,clk;

output reg q;

output qn;

always @(posedge clk)

begin

q = d;

end

assign qn = ~q;

endmodule

**TEST BENCH CODE:**

module dff\_tb();

reg D;

reg clk;

wire q,qn;

dff dut(D,clk,q,qn);

initial

begin

clk=0;

forever #10 clk = ~clk;

end

initial

begin

D = 0;

#100;

D = 1;

#100;

D = 0;

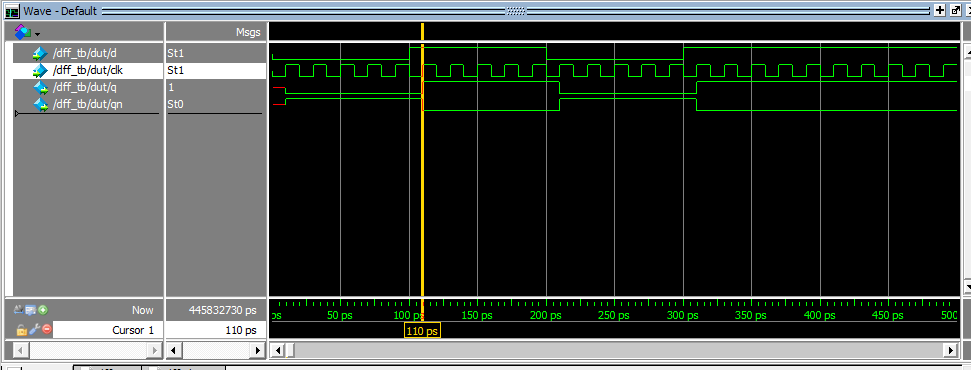
#100;

D = 1;

end

endmodule

**SIMULATION:**

****

ii.T FLIP FLOP USING D FLIP FLOP:

RTL CODE:

module tff(t,clk,rst,q,qn);

input t,clk,rst;

output q;

output qn;

wire b;

assign b = t^q;

dff T1(b,clk,rst,q,qn);

endmodule

module dff(d,clk,rst,q,qn);

input d,clk,rst;

output reg q;

output qn;

always @(posedge clk)

begin

if(rst)

q<=1'b0;

else

q <= d;

end

assign qn=~q;endmodule

TESTBENCH CODE:

module tff\_tb();

reg t,rst;

reg clk;

wire q,qn;

tff dut(t,clk,rst,q,qn);

initial

begin

clk=0;

forever #10 clk = ~clk;

end

initial

begin

@(negedge clk)

rst=1;

t = 0;

@(negedge clk)

rst=0;

t = 1;

@(negedge clk)

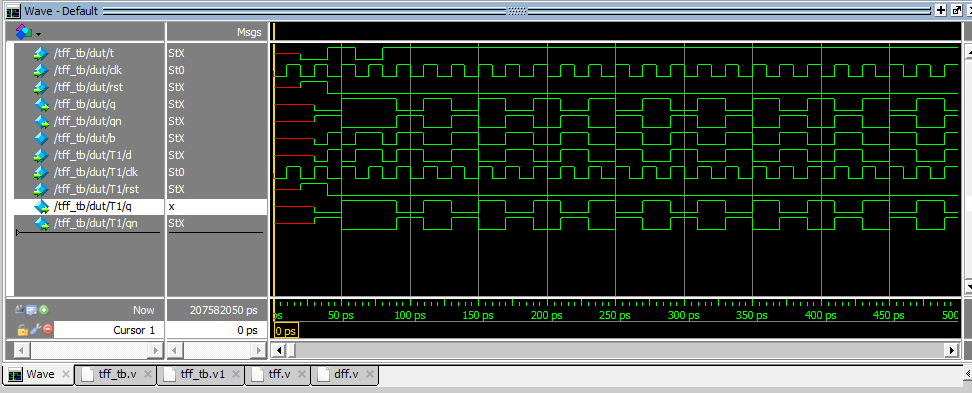
t = 0;

@(negedge clk)

t = 1;

end endmodule

SIMULATION:



**iii.4 BIT LOADBLE UP COUNTER:**

**RTL CODE:**

module counter4b (input clk,rst,load,[3:0]din,output reg[3:0]dout);

always @(posedge clk) begin

if (! rst)

dout <= 0;

else if(load)

dout <= din;

else

dout <= dout+1'b1;

end

endmodule

**TESTBENCH CODE:**

module tb\_counter();

reg clk;

reg rst;

reg load;

reg[3:0]din;

wire [3:0] out;

counter4b dut (clk,rst,load,din,dout);

always #15 clk = ~clk;

initial

begin

clk <= 0;

rst <= 0;

din <= 4'b0000;

#20 rst <= 1;

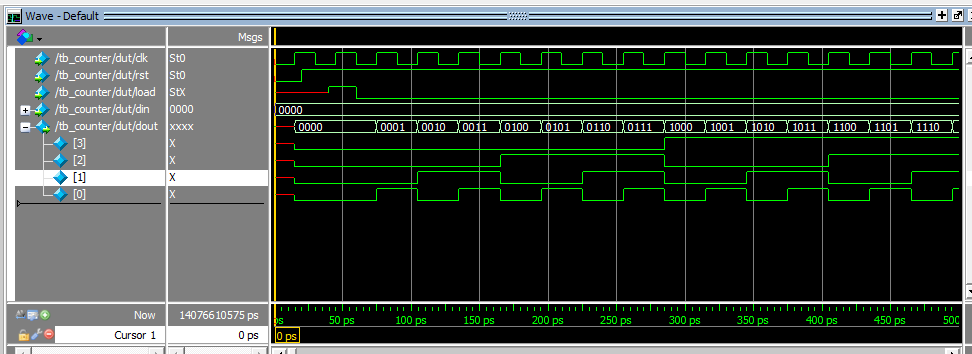
#20 load <= 1;

#20 load <= 0;

end

endmodule

**SIMULATION:**

****

iv.JK FLIP FLOP USING PARAMETERS DECLARATION:

RTL CODE:

module jkffp(j,k,clk,q,qn);

input j,k,clk;

output reg q;

output reg qn;

parameter hold=1,

set=1,

reset=0,

toggle=1;

always @(posedge clk)

begin

case ({j,k})

2'b00 :q <= (hold & q);

2'b01 :q <= reset;

2'b10 :q <= set;

2'b11 :q <= (toggle & qn);

endcase

assign qn = ~q;

end

endmodule

TESTBENCH CODE:

module jkffp\_tb();

reg j;

reg k;

reg clk;

wire q;

wire qn;

jkffp dut(j,k,clk,q,qn);

always #10 clk=~clk;

initial

begin

clk<=0;

j<=1;

k<=0;

#20;

j<=0;

k<=1;

#20;

j<=1;

k<=0;

#20;

j<=0;

k<=0;

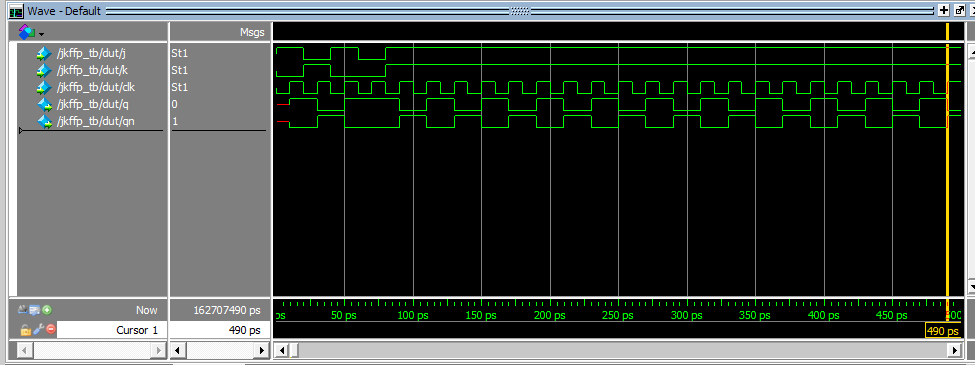
#20;

j<=1;

k<=1;

endendmodule

SIMULATION:



v.SR LATCH USING GATE LEVEL MODELING:

RTL CODE:

module srgm(s,r,q,qn);

input s,r;

inout q;

inout qn;

nand N1(q,r,qn);

nand N2(qn,s,q);

endmodule

TESTBENCHCODE:

module srgm1\_tb();

reg s;

reg r;

wire q;

wire qn;

srgm dut(s,r,q,qn);

initial

begin

s=1'b1;

r=1'b0;

#20;

s=1'b0;

r=1'b1;

#20;

s=1'b1;

r=1'b0;

#20;

s=1'b1;

r=1'b1;

#20;

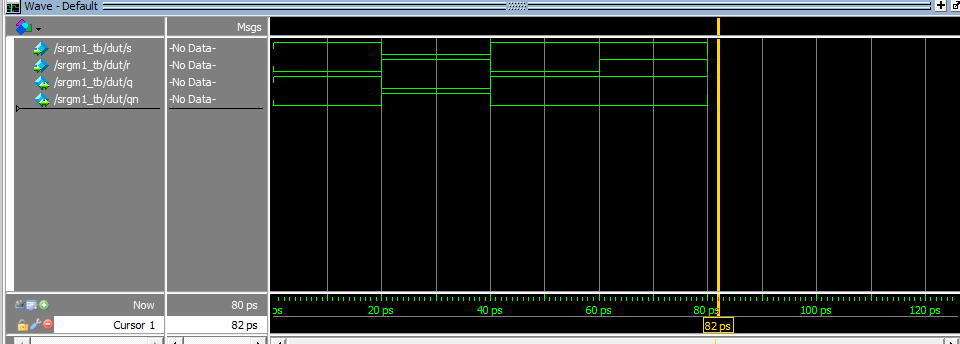
s=1'b0;

r=1'b0;

end

endmodule

SIMULATION:



vi.BIT LOADABLE MOD12 UP COUNTER:

RTL CODE:

module cou12(input clk,rst,load,[3:0]din,output reg[3:0]dout);

always @(posedge clk)

begin

if (! rst)

dout <= 0;

else if(load)

dout <= din;

else

begin

if(dout==12)

dout<=0;

else

dout <= dout+1'b1;

end

end

endmodule

TESTBENCH CODE:

module cou12\_tb();

reg clk;

reg rst;

reg load;

reg[3:0]din;

wire [3:0]dout;

cou12 dut(clk,rst,load,din,dout);

always #15 clk = ~clk;

initial

begin

clk <= 0;

rst <= 0;

din <=4'b0000;

#20 rst <= 1;

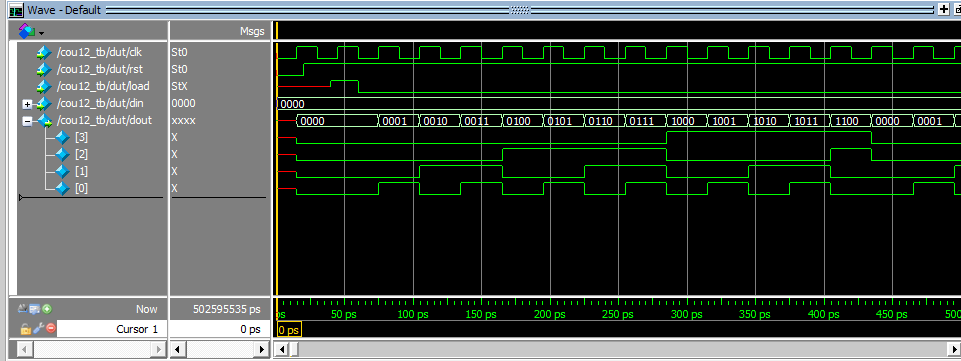
#20 load <= 1;

#20 load <= 0;

end

endmodule

SIMULATION:

**vii.4 BIT LOADBLE UPDOWN COUNTER:**

**RTL CODE:**

module counter4ud(input clk,rst,load,output reg[3:0]dout);

always @(posedge clk)

begin

if (! rst)

dout <= 0;

else if(load)

begin

if(dout==15)

dout<=4'b0000;

else

dout <= dout+1'b1;

end

else

begin

if(dout==0)

dout<=15;

else

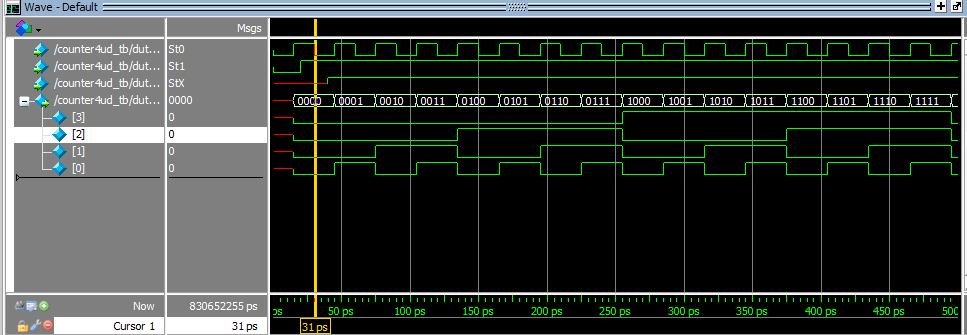
dout <= dout-1'b1;

end

end

endmodule

**SIMULATION:**

****

viii.4 BIT SISO:

RTL CODE

module siso4(input clk,clr,d,output v);

reg[3:0]q;

always @(posedge clk)

begin

if(clr)

q<=4'b0000;

else

begin

q <=q>>1;

q[3]<=d;

end

end

assign v =q[0];

endmodule

TESTBENCH CODE:

module siso4\_tb();

reg clk,clr,d;

wire v;

siso4 dut(clk,clr,d,v);

initial

begin

clk =1'b0;

end

always #10 clk=~clk;

initial

begin

clr=1;

#10;

clr=1;

#10;

clr=0;

#10;

d=1;

#10;

d=0;

#10;

d=0;

#10;

d=1;

#10;

d=1;

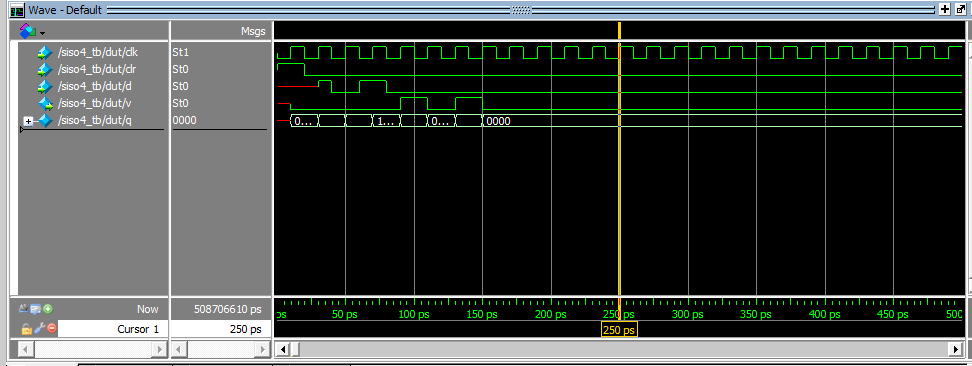
#10;

d=0;

end

endmodule

SIMULATION:



LAB 5:

i.16X8 SYNCHRONOUS DUAL PORT RAM MEMORY:

RTL CODE:

module ram16x8(clk,read,wrt,rst,rd\_ad,wr\_ad,data\_in,data\_out);

input clk,read,wrt,rst;

input [3:0]rd\_ad;

input [3:0]wr\_ad;

input [7:0] data\_in;

output reg[7:0] data\_out;

reg [7:0]mem[15:0];

integer i;

always @(posedge clk)

begin

if(rst)

begin

for(i=0;i<=15;i=i+1)

begin

mem[i] = 0;

end

end

else

begin

if(wrt)

mem[wr\_ad] = data\_in;

if(read)

data\_out = mem[rd\_ad];

end

end

endmodule

TESTBENCH CODE:

module ram16x8\_tb();

reg clk,read,wrt,rst;

reg[3:0]rd\_ad;

reg[3:0]wr\_ad;

reg [7:0] data\_in;

wire [7:0] data\_out;

ram16x8 dut(clk,read,wrt,rst,rd\_ad,wr\_ad,data\_in,data\_out);

always

begin

#10 clk = 1'b0;

#10 clk = 1'b1;

end

task rst1;

begin

@(negedge clk)

rst = 1'b1;

@(negedge clk)

rst = 1'b0;

end

endtask

task write1(input [7:0]s,input [3:0]d,input w,r);

begin

@(negedge clk)

wrt=w;

read=r;

wr\_ad = d;

data\_in = s;

end

endtask

task read1(input [3:0]t,input w,r);

begin

@(negedge clk)

wrt=w;

read=r;

rd\_ad = t;

end

endtask

initial

begin

rst1;

repeat(5)

write1({$random}%256,{$random}%16,1'b1,1'b0);

repeat(10)

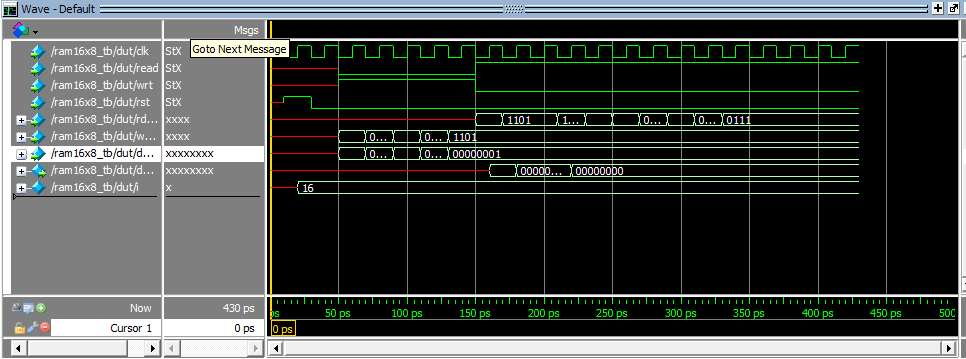
read1({$random}%16,1'b0,1'b1);

#100 $finish;

end

endmodule

SIMULATION:



ii.8x16 ASYNCHRONOUS DUAL PORT RAM MEMORY

RTL CODE:

module ram16x8(clk,read,wrt,rst,rd\_ad,wr\_ad,data\_in,data\_out);

input clk,read,wrt,rst;

input [2:0]rd\_ad;

input [2:0]wr\_ad;

input [15:0] data\_in;

output reg[15:0] data\_out;

reg [15:0]mem[7:0];

integer i;

always @(posedge clk)

begin

if(rst)

begin

for(i=0;i<=7;i=i+1)

begin

mem[i] = 0;

end

end

always @(posedge wr)

begin

if(wrt)

mem[wr\_ad] = data\_in;

end

always @(posedge wr)

begin

data\_out = mem[rd\_ad];

end

end

endmodule

TESTBENCH CODE:

module ram8x16\_tb();

reg clk,read,wrt,rst;

reg[2:0]rd\_ad;

reg[2:0]wr\_ad;

reg [15:0] data\_in;

wire [15:0] data\_out;

ram 8x16 dut(clk,read,wrt,rst,rd\_ad,wr\_ad,data\_in,data\_out);

always

begin

#10 clk = 1'b0;

#10 clk = 1'b1;

end

task rst1;

begin

@(negedge clk)

rst = 1'b1;

@(negedge clk)

rst = 1'b0;

end

endtask

task write1(input [15:0]s,input [2:0]d,input w,r);

begin

@(negedge clk)

wrt=w;

read=r;

wr\_ad = d;

data\_in = s;

end

endtask

task read1(input [2:0]t,input w,r);

begin

@(negedge clk)

wrt=w;

read=r;

rd\_ad = t;

end

endtask

initial

begin

rst1;

repeat(5)

write1({$random}%256,{$random}%16,1'b1,1'b0);

repeat(10)

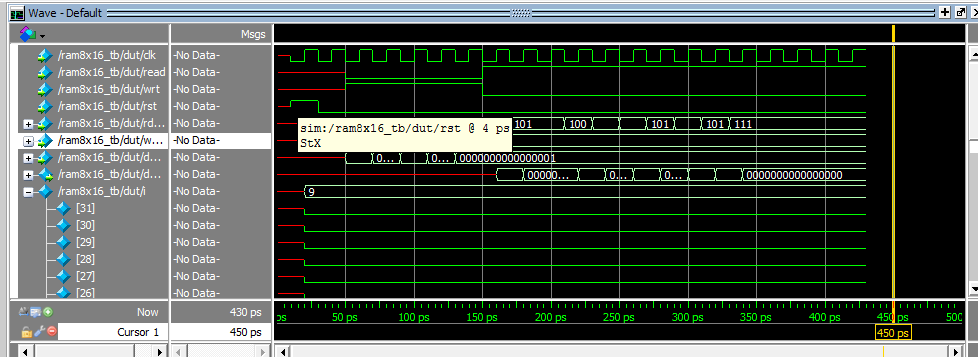
read1({$random}%16,1'b0,1'b1);

#100 $finish;

end

endmodule

**SIMULATION:**

****

iii.CLOCK BUFFER:

RTL CODE:

module clkbuf(input iclk,output oclk);

buf S1(oclk,iclk);

endmodule

Testbench code:

module clkbuf\_tb();

reg iclk;

wire oclk;

clkbuf dut(iclk,oclk);

reg p,f;

time t1,t2,t3,t4,t5,t6;

initial

begin

iclk=1'b0;

end

always #10 iclk=~iclk;

task measure\_iclk();

begin

@(posedge iclk)

t1=$time;

@(posedge iclk)

t3=$time;

end

endtask

task measure\_oclk();

begin

@(posedge oclk)

t2=$time;

@(posedge oclk)

t4=$time;

end

endtask

initial

begin

fork

measure\_iclk();

measure\_oclk();

join

if((t1==t3)&&(t3==t4))

$display("phase is matching");

else

$display("phase is notmatching");

t5=t3-t1;

t6=t4-t2;

if(t5==t6)

$display("frequency is matching");

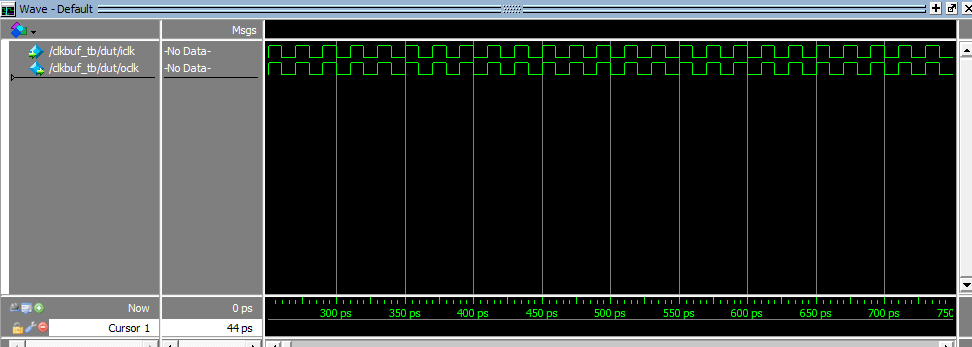
else

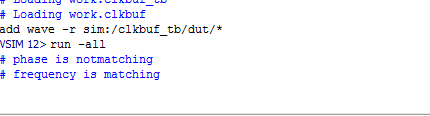
$display("frequency is not matching");

end

endmodule

**SIMULATION:**





iv.WRITE AN RTL CODE TEST BENCH CODE FOR 8X16 ASYNCHRONOUS SINGLE PORT RAM MEMORY:

RTL CODE:

module ram8x16(wclk,rclk,re,wr,clr,rd\_ad,wr\_ad,data\_in,data\_out);

input wclk,rclk,re,wr,clr;

input [2:0]rd\_ad;

input [2:0]wr\_ad;

input [15:0] data\_in;

output reg[15:0] data\_out;

reg [15:0]mem[7:0];

integer i;

always @(posedge wclk or posedge clr)

begin

if(clr)

begin

for(i=0;i<=7;i=i+1)

begin

mem[i] = 0;

end

end

else

begin

if(wr)

mem[wr\_ad] = data\_in;

end

end

always @(posedge rclk or posedge clr)

begin

if(clr)

begin

data\_out <= 0;

end

else

begin

if(re)

data\_out<=mem[rd\_ad];

end

end

endmodule

**TESTBENCH CODE:**

module ram8x16\_tb();

reg wclk,rclk,re,wr,clr;

reg[2:0]rd\_ad;

reg[2:0]wr\_ad;

reg [15:0] data\_in;

wire [15:0] data\_out;

ram8x16 dut(wclk,rclk,re,wr,rst,rd\_ad,wr\_ad,data\_in,data\_out);

initial

begin

wclk = 1'b0;

rclk = 1'b0;

end

always #10 wclk = ~wclk;

always #10 rclk = ~rclk;

task rst1;

begin

clr = 1'b1;

#30;

clr = 1'b0;

end

endtask

task write1(input [15:0]s,input [2:0]d,input w,r);

begin

@(negedge wclk)

wr=w;

re=r;

wr\_ad = d;

data\_in = s;

end

endtask

task read1(input [2:0]t,input w,r);

begin

@(negedge rclk)

wr=w;

re=r;

rd\_ad = t;

end

endtask

initial

begin

rst1;

repeat(5)

write1({$random}%256,{$random}%16,1'b1,1'b0);

repeat(10)

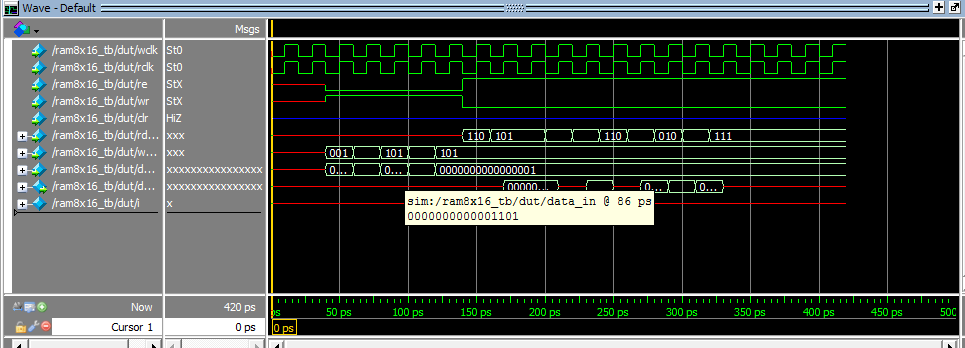
read1({$random}%16,1'b0,1'b1);

#100 $finish;

end

endmodule

**SIMULATION:**



v.WRITE AN RTL CODE TEST BENCH CODE FOR 16X8 FIFO MEMORY :

**RTL CODE:**

module ram16x8fifo(clk,re,wr,rst,full,empty,data\_in,data\_out);

input clk,re,wr,rst;

reg [3:0]rd\_pt;

reg [3:0]wr\_pt;

reg [4:0]fifo\_counter;

input [7:0] data\_in;

output full,empty;

output reg[7:0] data\_out;

reg [7:0]mem[15:0];

integer i;

assign empty =(fifo\_counter==5'b0)?1'b1:1'b0;

assign full =(fifo\_counter>5'b01111);

always @(posedge clk)

begin

if(rst)a

fifo\_counter<=0;

else if(!full&wr)

fifo\_counter<=fifo\_counter+1;

else if(!empty&&re)

fifo\_counter<=fifo\_counter+1;

else

fifo\_counter<=fifo\_counter;

end

always @(posedge clk)

begin

if(rst)

begin

for(i=0;i<=15;i=i+1)

begin

mem[i] = 0;

end

end

else if((wr==1'b1)&&(full==1'b0))

begin

mem[wr\_pt]<=data\_in;

wr\_pt<=wr\_pt+1;

end

else

wr\_pt<=wr\_pt;

end

always @(posedge clk)

begin

if(rst)

begin

rd\_pt<=4'b0;

data\_out<=8'b0;

end

else if((re==1'b1)&&(empty==1'b0))

begin

data\_out<=mem[rd\_pt];

rd\_pt <=rd\_pt+1;

end

else

rd\_pt<=rd\_pt;

end

endmodule

TEST BENCH CODE:

module ram16x8fifo\_tb();

reg clk,rst,wr,re;

reg [7:0]data\_in;

wire empty,full;

wire [7:0]data\_out;

ram16x8fifo dut(clk,re,wr,rst,full,empty,data\_in,data\_out);

initial

begin

clk=1'b0;

end

always #10 clk =~clk;

task initialize();

begin

wr=1'b0;

re=1'b0;

rst=1'b0;

data\_in=0;

end

endtask

task reset();

begin

@(negedge clk)

rst=1'b1;

@(negedge clk)

rst=1'b0;

end

endtask

task write(input [7:0]data);

begin

@(negedge clk)

wr=1'b1;

data\_in=data;

end

endtask

task read();

begin

@(negedge clk)

re=1'b1;

end

endtask

initial

begin

initialize;

#50;

reset;

repeat(10)

write({$random}%8);

wr=1'b0;

repeat(10)

read();

re=1'b0;

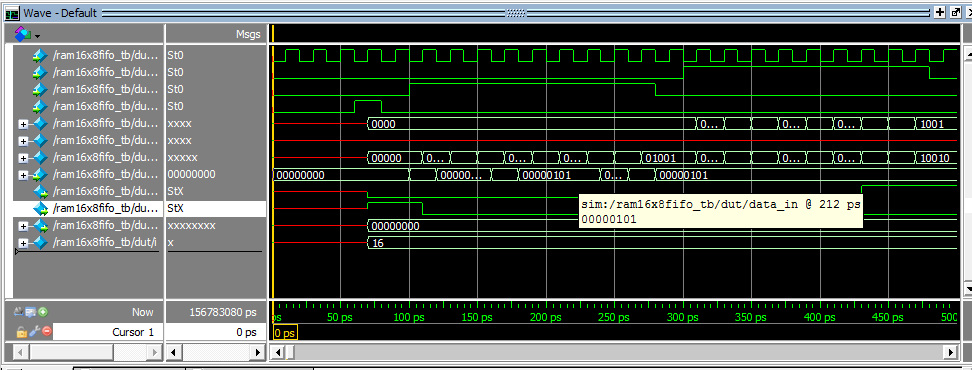
repeat(10)

write({$random}%8);

end

endmodule

**SIMULATION:**



**LAB 5:**

**i.RTL CODE FOR 101 SEQUENCE DETECTOR:**

RTL CODE:

module fsmd(input clk,rst,in,output reg out);

parameter s0=2'b00,

s1=2'b01,

s2=2'b10,

s3=2'b11;

reg [1:0] state;

reg [1:0] next\_state;

always @(posedge clk)

begin

if(rst)

state <= s0;

else

state <= next\_state;

end

always @(state,in)

begin

case(state)

s0:if(in)

next\_state <= s1;

else

next\_state <= s0;

s1:if(in)

next\_state <= s1;

else

next\_state <= s2;

s2:if(in)

next\_state <= s3;

else

next\_state <= s0;

s3:if(in)

next\_state <= s1;

else

next\_state <= s0;

endcase

assign out = (state == s3)?1:0;

end

endmodule

**TESTBENCH CODE:**

module fsmd\_tb();

reg clk;

reg rst;

reg in;

wire out;

fsmd dut(clk,rst,in,out);

initial

begin

clk <=1'b0;

forever #10 clk=~clk;

end

initial

begin

@ (negedge clk)

rst = 1'b1;

#20;

rst = 1'b0;

#20;

in <= 1'b0;

#20;

in <= 1'b1;

#20;

in <= 1'b1;

#20;

in <= 1'b0;

#20;

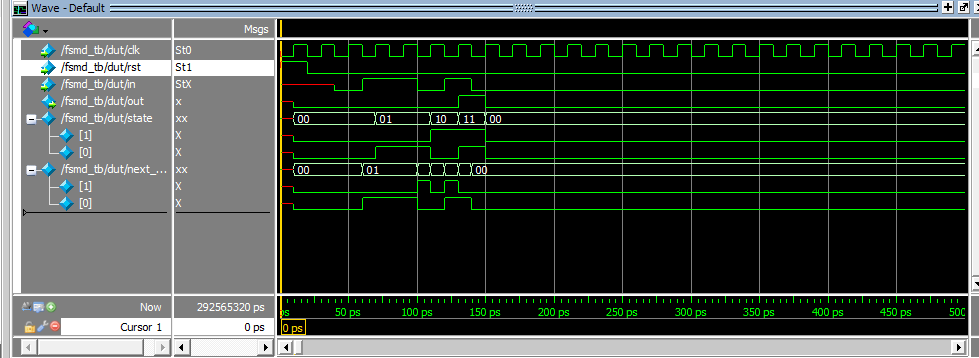
in <= 1'b1;

#20;

in <= 1'b0;

endendmodule

SIMULATION:



ii.DESIGN A VENDING MACHINE FOR SOME SPECIFICATION AS FOLLOWS:

I J

1. X NO COIN
2. 0 1RS

1 1 2RS

RTL CODE:

module fsm2(input clk,rst,i,j,output reg x,y);

parameter idle=2'b00,

s1=2'b01,

s2=2'b10;

reg [1:0] state;

reg [1:0] next\_state;

always @(posedge clk)

begin

if(rst)

state <= idle;

else

state <= next\_state;

end

always @(state,i,j)

begin

case(state)

idle:if(!i)

begin

next\_state <= idle;

x=0;

y=0;

end

else if(j==1)

begin

next\_state <= s2;

x=0;

y=0;

end

else

begin

next\_state <= s1;

x=0;

y=0;

end

s1:if(!i)

begin

next\_state <= s1;

x=0;

y=0;

end

else if(j==1)

begin

next\_state <= idle;

x=1;

y=0;

end

else

begin

next\_state <= s2;

x=0;

y=0;

end

s2:if(!i)

begin

next\_state <= s2;

x=0;

y=0;

end

else if(j==1)

begin

next\_state <= idle;

x=1;

y=1;

end

else

begin

next\_state <= idle;

x=1;

y=0;

end

endcase

end

endmodule

TEST BENCH CODE:

module fsm2\_tb();

reg clk;

reg rst;

reg i,j;

wire x,y;

fsm2 dut(clk,rst,i,j,x,y);

initial

begin

clk <=1'b0;

forever #10 clk=~clk;

end

initial

begin

@ (negedge clk)

rst = 1'b1;

#20;

rst = 1'b0;

#20;

i <= 1'b0;

j <= 1'b1;

#20;

i <= 1'b1;

j <= 1'b1;

#20;

i <= 1'b1;

j <= 1'b0;

#20;

i <= 1'b0;

j <= 1'b1;

#20;

i <= 1'b1;

j <= 1'b1;

#20;

i <= 1'b1;

j <= 1'b0;

#20;

i <= 1'b1;

j <= 1'b1;

#20;

i <= 1'b1;

j <= 1'b0;

#20;

end

endmodule

SIMULATION:

