# AIM:

# Write verilog code for Basic Logic gates

1. **Verify the code using test bench**

module NOTgate1(A,F);

input A;

output F;

// reg F;

not x1(F,A); **//Gate level modeling**

assign F = ~A; **//Data Flow modeling**

always @ (A) **//Behavior modeling**

begin

F <= ~A;

end

endmodule

//**Test bench for NOT**

module Testbench;

reg A\_t;

wire F\_t;

NOTgate1 NOTgate1\_1(A\_t,F\_t);

initial

begin

//case 0

A\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 1

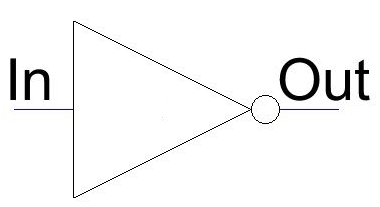
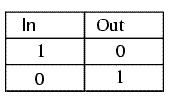
A\_t <= 1;

#1 $display("F\_t = %b",F\_t);

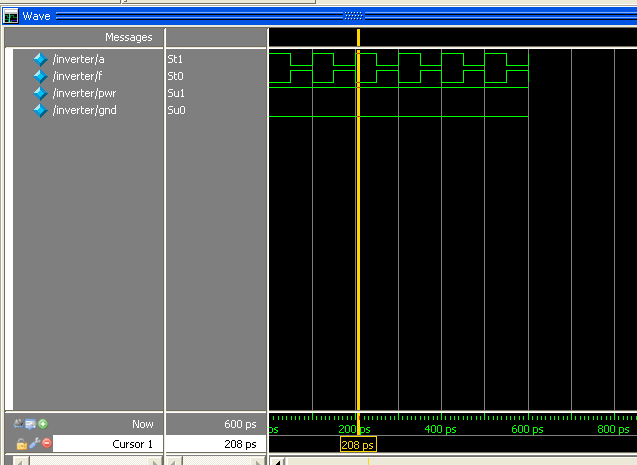
end

endmodule

**Logic symbol and Truth Table:**

** **

**OUTPUT WAVEFORM:**



**Fig:1**

module OR2gate(A,B,F);

input A;

input B;

output F;

// reg F;

1. or x1(F,A,B);

2.assign F=A | B;

3.always @ (A or B)

begin

F <= A | B;

end

endmodule

//**Test bench for OR**

module Testbench;

reg A\_t,B\_t;

wire F\_t;

OR2gate OR2gate\_1(A\_t,B\_t,F\_t);

initial

begin

//case 0

A\_t <= 0; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 1

A\_t <= 0; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

//case 2

A\_t <= 1; B\_t <= 0;

#1 $display("F\_t = %b", F\_t);

//case 3

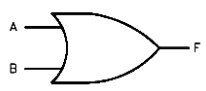
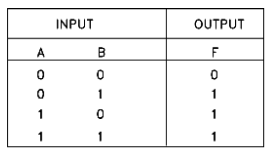
A\_t <= 1; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

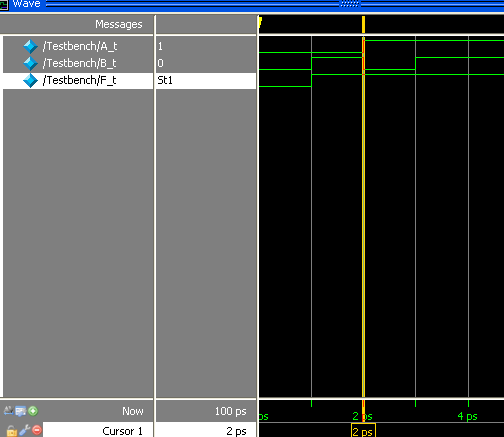
end

endmodule

**Logic symbol and Truth Table:**

****

**OUTPUT WAVEFORM:**



**Fig: 2**

module AND2gate(A,B,F);

input A;

input B;

output F;

// reg F;

* 1. and x1(F,A,B);
  2. assign F=A & B;

3. always @ (A or B)

begin

F <= A & B;

end

endmodule

**Test bench for AND**

module Testbench;

reg A\_t, B\_t;

wire F\_t;

AND2gate AND2gate\_1(A\_t,B\_t,F\_t);

initial

begin

//case 0

A\_t <= 0; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 1

A\_t <= 0; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

//case 2

A\_t <= 1; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 3

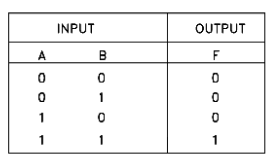
A\_t <= 1; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

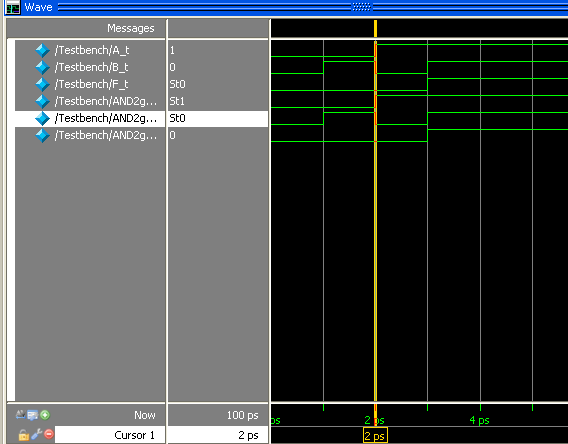
end

endmodule

**Logic symbol and Truth Table:**

**OUTPUT WAVEFORM:**



**Fig: 3**

module NAND2gate(A,B,F);

input A;

input B;

output F;

// reg F;

1.nand x1(F,A,B);

2.assign F= ~(A & B);

3.always @ (A or B)

begin

F <= ~(A & B);

end

endmodule

**Test bench for NAND**

module Testbench;

reg A\_t,B\_t;

wire F\_t;

NAND2gate NAND2gate\_1(A\_t,B\_t,F\_t);

initial

begin

//case 0

A\_t <= 0; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 1

A\_t <= 0; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

//case 2

A\_t <= 1; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 3

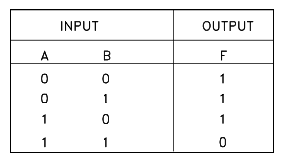
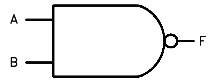
A\_t <= 1; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

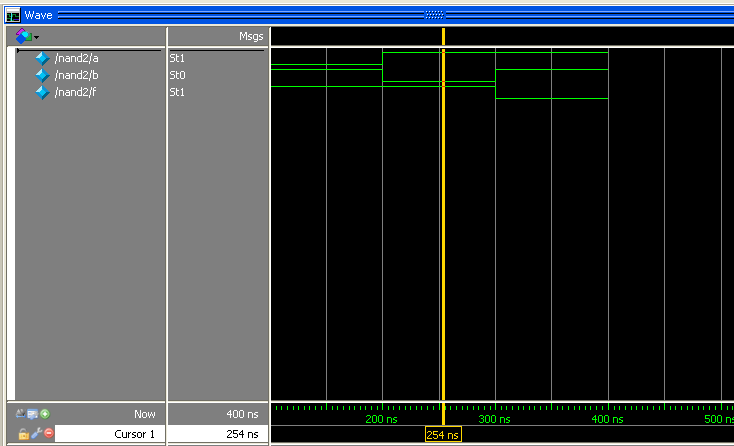
end

endmodule

**Logic symbol and Truth Table:**



**OUTPUT WAVEFORM:**



**Fig: 4**

module NOR2gate(A,B,F);

input A;

input B;

output F;

// reg F;

* + 1. nor x1(F,A,B);
    2. assign F= ~(A | B);

3. always @ (A or B)

begin

F <= ~(A | B);

end

endmodule

**Test bench for NOR**

module Testbench;

reg A\_t,B\_t;

wire F\_t;

NOR2gate NOR2gate\_1(A\_t,B\_t,F\_t);

initial

begin

//case 0

A\_t <= 0; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 1

A\_t <= 0; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

//case 2

A\_t <= 1; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 3

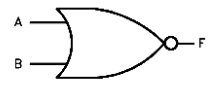
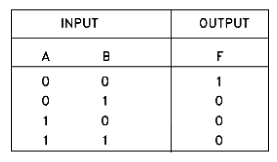
A\_t <= 1; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

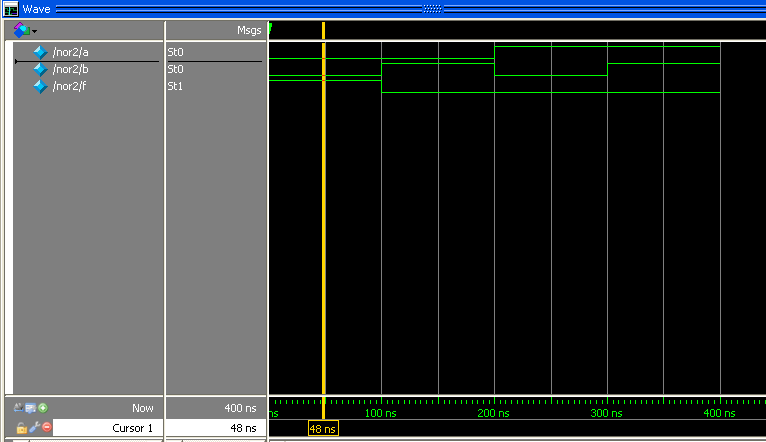
end

endmodule

**Logic symbol and Truth Table:**

** **

**OUTPUT WAVEFORM:**



**Fig: 5**

module XOR2gate(A,B,F);

input A;

input B;

output F;

// reg F;

* + 1. xor x1(F,A,B);
    2. assign F= A ^ B;

3.always @ (A or B)

begin

F <= A ^ B;

end

endmodule

**Test bench for XOR**

module Testbench;

reg A\_t,B\_t;

wire F\_t;

XOR2gate XOR2gate\_1(A\_t,B\_t,F\_t);

initial

begin

//case 0

A\_t <= 0; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 1

A\_t <= 0; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

//case 2

A\_t <= 1; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 3

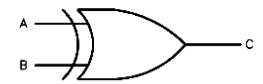
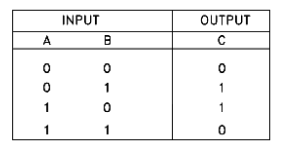
A\_t <= 1; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

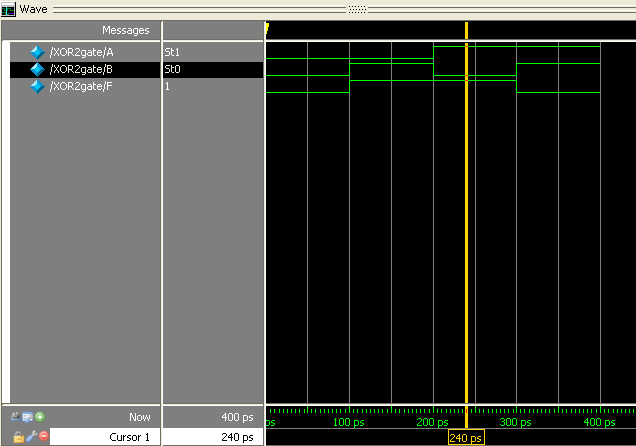
end

endmodule

**Logic symbol and Truth Table:**

** **

**OUTPUT WAVEFORM:**

****

**Fig: 6**

module XNOR2gate(A,B,F);

input A;

input B;

output F;

// reg F;

* + 1. xnor x1(F,A,B);
    2. assign F= ~(A ^ B);

3. always @ (A or B)

begin

F <= A ~^ B;

end

endmodule

**Test bench for XNOR**

module Testbench;

reg A\_t,B\_t;

wire F\_t;

XNOR2gate XNOR2gate\_1(A\_t,B\_t,F\_t);

initial

begin

//case 0

A\_t <= 0; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 1

A\_t <= 0; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

//case 2

A\_t <= 1; B\_t <= 0;

#1 $display("F\_t = %b",F\_t);

//case 3

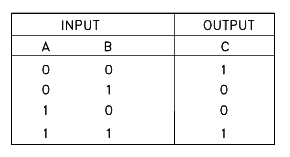
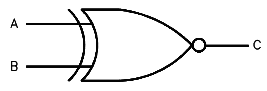
A\_t <= 1; B\_t <= 1;

#1 $display("F\_t = %b",F\_t);

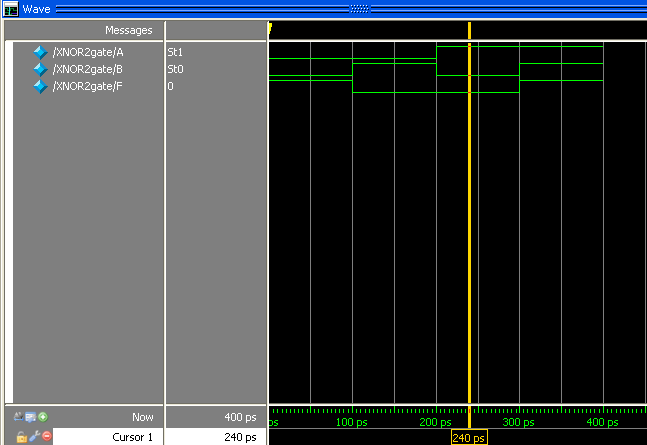
end

endmodule

**Logic symbol and Truth Table:**

****

**OUTPUT WAVEFORM:**



**fig:7**

**1. ARITHMETIC UNITS: ADDERS & SUBTRACTORS**

# AIM:

# Write verilog code for halfadder, fulladder, 4-bit parallel adder and full subtractor.

1. **Verify the code using test bench**

# PROGRAMME:

***Half Adder:***

module halfadder(a,b,sum,carry);

1. input a,b;

output sum, carry;

wire sum, carry;

assign sum = a^b; // sum bit

assign carry = (a&b) ; //carry bit

endmodule

2. xor x1(s,a,b);

and a1(c,a,b);

endmodule

***Test bench for half adder:***

module halfadder\_testreg a, b;

wire sum, carry;

halfadder add(a,b,sum,carry);

always @(sum or carry)

begin

$display("time=%d:%b + %b = %b,carry = %b\n",$time,a,b,sum,carry);

end

initial

begin

a = 0; b = 0;

#5 a = 0; b = 1;

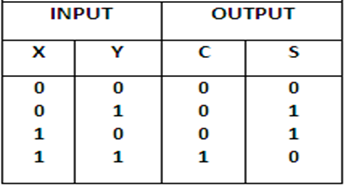
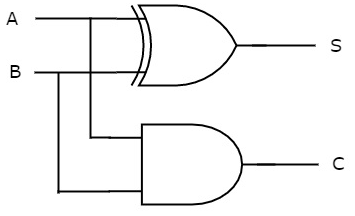
#5 a = 1; b = 0;

#5 a = 1; b = 1;

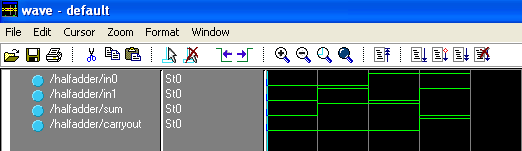
end

endmodule

**Logic symbol and Truth Table:**

****

**OUTPUT WAVEFORM:**



**Fig. 8**

***Full – Adder:***

module fulladder (Cin, x, y, s, Cout);

input Cin, x, y;

output s, Cout;

reg s, Cout;

1. ***Data flow model*** :

assign s = x ^ y ^ Cin; //concurrent dataflow

assign Cout = (x & y) | (x & Cin) | (y & Cin);

2. ***Behavioral model:***

always @(x or y or Cin)

{Cout, s} = x + y + Cin;

3.***Gate level model:***

wire z1,z2,z3,z4 // connecting wires z1,z2,z3,z4 ;

and (z1, x, y);

and (z2, x, Cin);

and (z3, y, Cin);

or (Cout, z1, z2, z3);

xor (z4, x, y);

xor (s, z4, Cin);

xor (s, x, y, Cin);

and (z1, x, y), (z2, x, Cin),(z3, y, Cin); //multiple instantiations

or (Cout, z1, z2, z3);

endmodule

***Test bench for full adder:***

module fulladder\_test;

reg a, b, c;

wire sum, carry;

fulladder add(a,b,c,sum,carry);

always @(sum or carry)

begin

$display("time=%d:%b + %b + %b = %b, carry =

%b\n",$time,a,b,c,sum,carry);

end

initial

begin

a = 0; b = 0; c = 0;

#5 a = 0; b = 1; c = 0;

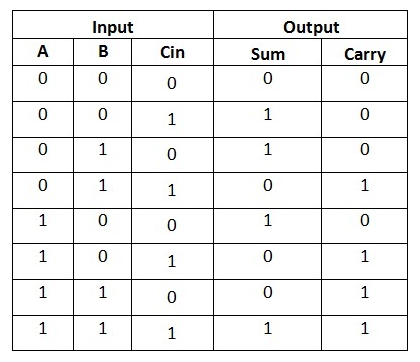
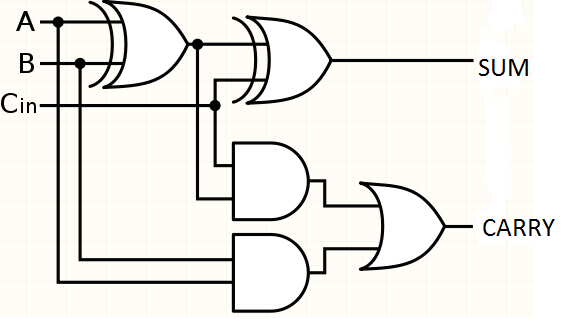
#5 a = 1; b = 0; c = 1;

#5 a = 1; b = 1; c = 1;

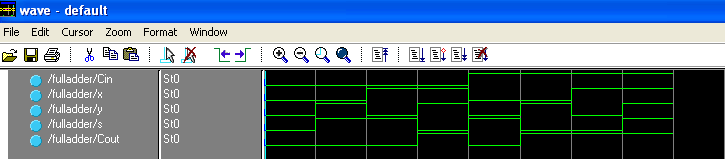
end

endmodule

**Logic symbol and Truth Table:**



**OUTPUT WAVEFORM:**



**Fig. 9**

***4-bit Full Adder:***

module adder4bit (carryin, X, Y, S, carryout);

input carryin;

input [3:0] X, Y;

output [3:0] S;

output carryout;

wire [3:1] C;

// Structural Model for 4-bit Full Adder

fulladder stage0 (carryin, X[0], Y[0], S[0], C[1]);

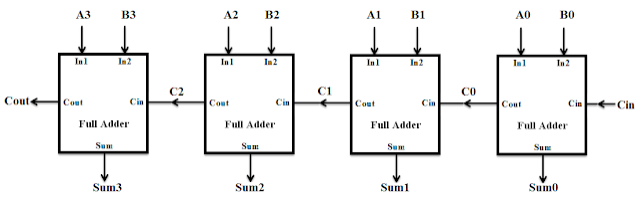
fulladder stage1 (C[1], X[1], Y[1], S[1], C[2]);

fulladder stage2 (C[2], X[2], Y[2], S[2], C[3]);

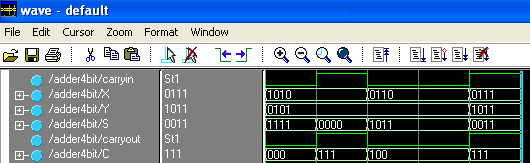
fulladder stage3 (C[3], X[3], Y[3], S[3], carryout);

endmodule

**Block Diagram:**



**OUTPUT WAVEFORM:**



**Fig.10**

***N-Bit Adder(using parameter decleration):***

module addern (carryin, X, Y, S, carryout);

parameter n=32;

input carryin;

input [n-1:0] X, Y;

output [n-1:0] S;

output carryout;

reg [n-1:0] S;

reg carryout;

reg [n:0] C;

integer k;

1. always @(X or Y or carryin)

begin

C[0] = carryin;

for (k = 0; k <= n-1; k = k+1)

begin

S[k] = X[k] ^ Y[k] ^ C[k];

C[k+1] = (X[k] & Y[k]) | (X[k] & C[k]) | (Y[k] & C[k]);

end

carryout = C[n];

end

2. always @(X or Y or carryin)

S = X + Y + carryin;

3. always @(X or Y or carryin) // output carryout, overflow;

begin // reg carryout, overflow;

Sum = {1'b0,X} + {1'b0,Y} + carryin; // reg [n:0] Sum;

S = Sum[n-1:0];

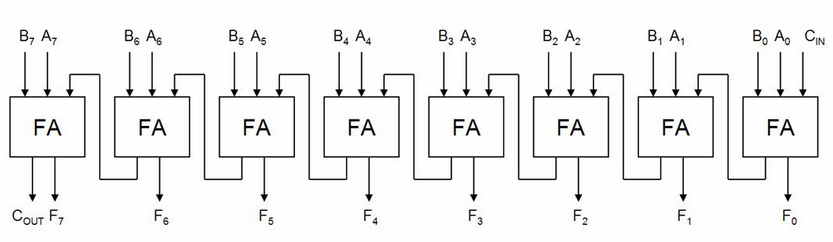
carryout = Sum[n];

overflow = carryout ^ X[n-1] ^ Y[n-1] ^ S[n-1];

end

endmodule

**Block Diagram:**



***Carry Look Ahead adder:***

module CLA\_4b(sum,c\_4,a,b,c\_0);

input [3:0]a,b;

input c\_0;

output [3:0]sum;

output c\_4;

wire p0,p1,p2,p3,g0,g1,g2,g3;

wire c1,c2,c3,c4;

assign

p0=a[0]^b[0],

p1=a[1]^b[1],

p2=a[2]^b[2],

p3=a[3]^b[3],

g0=a[0]&b[0],

g1=a[1]&b[1],

g2=a[2]&b[2],

g3=a[3]&b[3];

assign

c1=g0|(p0&c\_0),

c2=g1|(p1&g0)|(p1&p0&c\_0),

c3=g2|(p2&g1)|(p2&p1&g0)|(p2&p1&p0&c\_0),

c4=g3|(p3&g2)|(p3&p2&p1&g1)|(p3&p2&p1&g0)|(p3&p2&p1&p0&c\_0);

assign

sum[0]=p0^c\_0,

sum[1]=p1^c1,

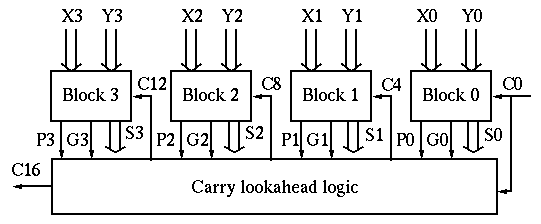
sum[2]=p2^c2,

sum[3]=p3^c3,

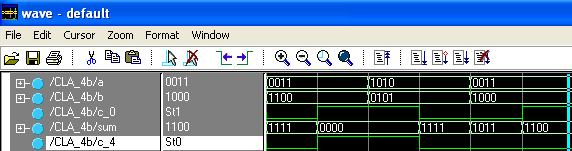
c\_4=c4;

endmodule

**Block Diagram:**



**OUTPUT WAVEFORM:**

****

**Fig.11**

***Full Substractor:***

module fullsub(x1,x2,x3,d,b);

input x1,x2,x3;

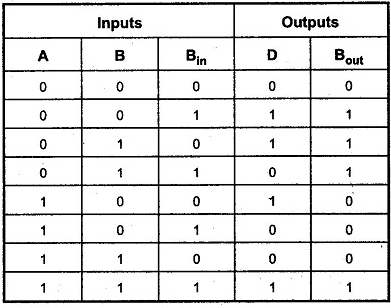
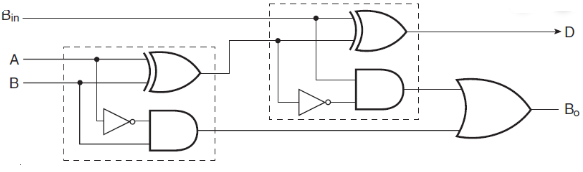
output d,b;

assign d= x1^x2^x3;

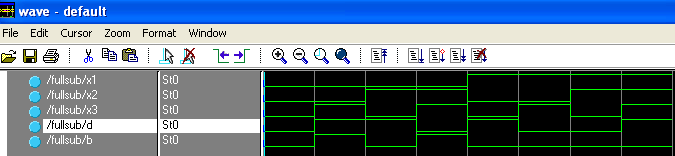
assign b=(~x1)&((x1^x3)|(x2&x3));

endmodule

**Logic symbol and Truth Table:**



**OUTPUT WAVEFORM:**

****

**Fig. 12**

**Test bench for Half Adder**

module half\_test;

reg a, b;

wire sum, carry;

halfadder add(a,b,sum,carry);

always @(sum or carry)

begin

$display("time=%d:%b + %b = %b,

carry = %b\n",$time,a,b,sum,carry);

end

initial

begin

a = 0; b = 0;

#5

a = 0; b = 1;

#5

a = 1; b = 0;

#5

a = 1; b = 1;

end

endmodule

**Test bench for Full Adder**

module main;

reg a, b, c;

wire sum, carry;

fulladder add(a,b,c,sum,carry);

always @(sum or carry)

begin

$display("time=%d:%b + %b + %b = %b, carry =

%b\n",$time,a,b,c,sum,carry);

end

initial

begin

a = 0; b = 0; c = 0;

#5

a = 0; b = 1; c = 0;

#5

a = 1; b = 0; c = 1;

#5

a = 1; b = 1; c = 1;

end

endmodule

**Test bench for Adder&Subtractor**

module addSub(A,B,sel,Result);

input sel;

input [3:0] A,B;

output [3:0] Result;

wire [3:0] Result;

assign Result = (sel)? A + B : A - B;

endmodule module main;

reg [3:0] A, B;

reg sel;

wire [3:0] Result;

addSub as1(A,B, sel,Result);

initial begin

A = 4'b0001;

B = 4'b1010;

end

initial begin

forever begin

#10

A = A + 1'b1;

B = B + 1'b2;

end

end

initial begin

sel = 1;

#200

sel = 0;

end

endmodule

**Test bench for 4bit Adder**

module adder4\_tb(); // testbench

reg [3:0] inputA, inputB;

wire [4:0] outputC;

integer i;

initial begin //a verilog process which only executed once

for (i=0; i < 10; i=i+1) begin

inputA = i;

inputB = i + 5;

#10; // wait 10 time unit

end

end

//DUT

adder4 adder4\_inst ( .A (inputA), .B (inputB), .C(outputC));

endmodule

**VIVA – VOCE QUESTIONS:**

1. What is module instantiation?
2. What are the different ways of association of ports in module instantiation?
3. Which is the fastest Adder?
4. What are the applications of Adders and Subtractors?
5. Which level of abstraction is suitable for combinational circuits?
6. What is the data type supports for the assignment of data in data flow modeling?

**2. MULTIPLEXERS AND DEMULTIPLEXERS**

**AIM:**

1. **Write the verilog code for multiplexers & Demultiplexers**
2. **Verify the code using test bench**

**PROGRAMME:**

***Mux 2x1:***

module mux2to1 (w0, w1, s, f);

input w0, w1, s;

output f;

reg f;

1. assign f = s ? w1 : w0;

2. always @(w0 or w1 or s)

f = s ? w1 : w0;

3. always @(w0 or w1 or s)

if (s==0)

f = w0;

else

f = w1;

endmodule

***Test bench for mux:***

module mux\_tb;

wire c;

reg a,b,s;

mux m1(c, a, b, s) ;

initial begin

#0 a=1'b0; b=1'b0; s=1'b0;

#5 a=1'b1;

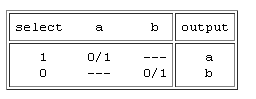
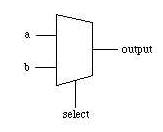
#5 s=1'b1;

#5 $finish; // $finish call ends simulation.

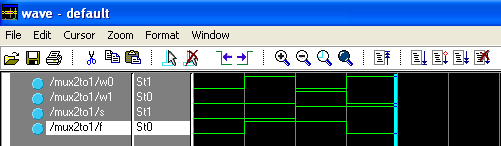
end

endmodule

**Block Diagram and Truth Table:**



**OUTPUT WAVEFORM:**



**Fig. 13**

***Mux 4x1:***

module mux4to1 (w0, w1, w2, w3, S, f);

input w0, w1, w2, w3;

input [1:0] S;

output f;

1. assign f = S[1] ? (S[0] ? w3 : w2) : (S[0] ? w1 : w0);

2. always @(w0 or w1 or w2 or w3 or S)

if (S == 2'b00) // (S == 1)

f = w0;

else if (S == 2'b01) // (S == 2)

f = w1;

else if (S == 2'b10) // (S == 3)

f = w2;

else if (S == 2'b11) // (S == 4)

f = w3;

3. always @(W or S)

case (S)

0: f = W[0];

1: f = W[1];

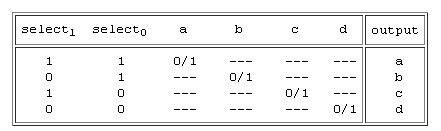
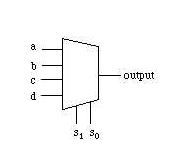
2: f = W[2];

3: f = W[3];

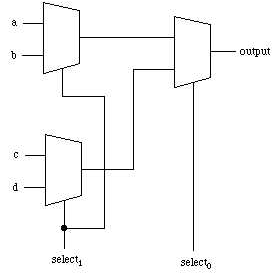
endcase

endmodule

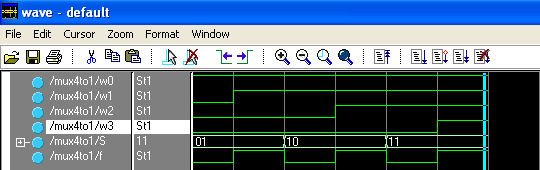
**Block Diagram and Truth Table:**



**4X1 using 2x1 MUX:**



**OUTPUT WAVEFORM:**

****

**Fig. 14**

***Mux 16x1:***

module mux16to1 (W, S16, f); // Structural Modeling

input [0:15] W;

input [3:0] S16;

output f;

wire [0:3] M;

mux4to1 Mux1 (W[0:3], S16[1:0], M[0]);

mux4to1 Mux2 (W[4:7], S16[1:0], M[1]);

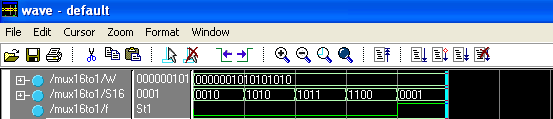
mux4to1 Mux3 (W[8:11], S16[1:0], M[2]);

mux4to1 Mux4 (W[12:15], S16[1:0], M[3]);

mux4to1 Mux5 (M[0:3], S16[3:2], f);

endmodule

**OUTPUT WAVEFORM:**



**Fig. 15**

**//Test bench for Multiplexer**

module mux\_tb();

wire c;

reg a,b,s;

mux m1(c, a, b, s) ;

initial begin

#0 a=1'b0;

b=1'b0;

s=1'b0;

#5 a=1'b1;

#5 s=1'b1;

#5 $finish; // The $finish call ends simulation.

end

initial begin

// Open a db file for saving simulation data

$shm\_open ("mux\_tb.db");

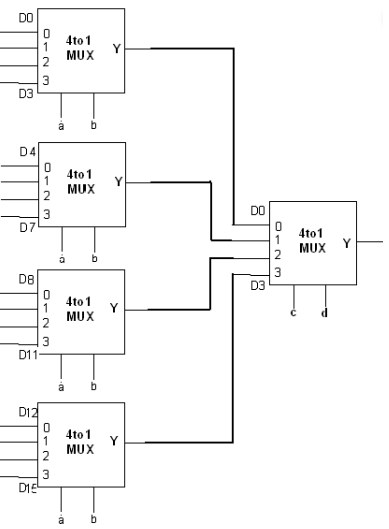
// Collect all signals (hierarchically) from the module "mux\_tb"

$shm\_probe (mux\_tb,"AS");

end

endmodule

**Block Diagram:**

****

**DEMULTIPLEXERS:**

***1x4 Demux:***

module demux(s,a,d);

input [0:1] s;

input a;

output [0:3] d;

reg [0:3] d;

always @(a or s)

begin

if(a==1)

begin

case (s)

2′b00:d=4′b1000;

2′b01:d=4′b0100;

2′b10:d=4′b0010;

2′b11:d=4′b0001;

default:d=0;

endcase

end else if(a==0)

d=0;

end

endmodule

***Test bench for Demux****:*

module demuxt\_b;

reg [0:1] s;

reg a;

wire [0:3] d;

demux uut (.s(s),.a(a),.d(d)  );

initial begin

#10 s=2′b00;a=1′b0;

#10 s=2′b00;a=1′b1;

#10 s=2′b01;a=1′b0;

#10 s=2′b01;a=1′b1;

#10 s=2′b10;a=1′b0;

#10 s=2′b10;a=1′b1;

#10 s=2′b11;a=1′b0;

#10 s=2′b11;a=1′b1;

#10 $stop;

end

endmodule

module demux1x4(y,s,i);

input i;

input [1:0]s;

output [3:0]y;

wire w0,w1;

not(w0,s0);

not(w1,s0);

and(y[0],i,w0,w1);

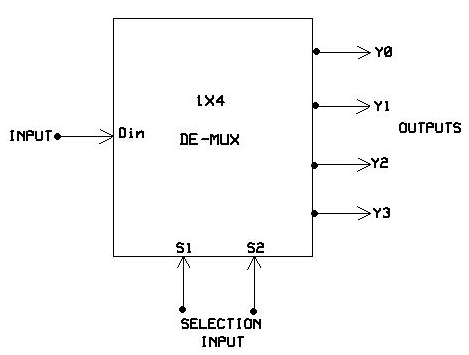
and(y[1],i,w1,s0);

and(y[2],i,s1,w0);

and(y[3],i,s1,s0);

endmodule

**Block Diagram:**



***1x8 Demux:***

module demux18(i,s,f);

input i;

input[2:0] s;

output[7:0] f;

1. always @(s)

begin

case (s)

3'b000: f[0]=i;

3'b001: f[1]=i;

3'b010: f[2]=i;

3'b011: f[3]=i;

3'b100: f[4]=i;

3'b101: f[5]=i;

3'b110: f[6]=i;

3'b111: f[7]=i;

endcase

end

2. wire[2:0] sb;

assign sb[0]=~s[0];

assign sb[1]=~s[1];

assign sb[2]=~s[2];

assign f[0]=i&sb[2]&sb[1]&sb[0];

assign f[1]=i&sb[2]&sb[1]&s[0];

assign f[2]=i&sb[2]&s[1]&sb[0];

assign f[3]=i&sb[2]&s[1]&s[0];

assign f[4]=i&s[2]&sb[1]&sb[0];

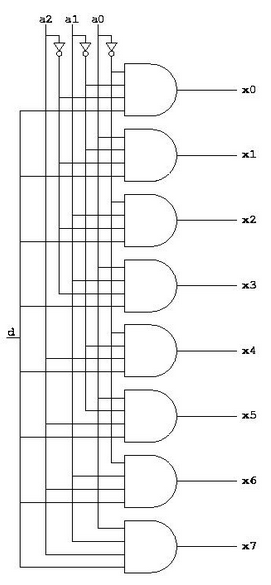
assign f[5]=i&s[2]&sb[1]&s[0];

assign f[6]=i&s[2]&s[1]&sb[0];

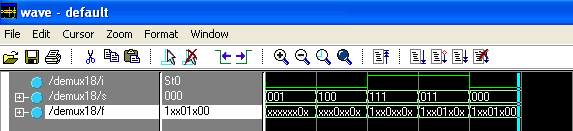
assign f[7]=i&s[2]&s[1]&s[0];

endmodule

**Logic Diagram:**



**OUTPUT WAVEFORM:**



**fig. 16**

**VIVA – VOCE QUESTIONS:**

1. How many 2 x 1 multiplexers are required in implementing a n x 1 multiplexer where ‘n’ is multiple of 2?
2. What are the applications of multiplexers and demultiplexers?
3. How many select lines are required for n x 1 multiplexer?
4. What is other name of behavioral level of abstraction?
5. Why the multiplexer can be called as a Universal element?