1. To study, design and verify the truth table of various logic gates (NOT, AND, OR, NAND, NOR, EX-OR, & EX-NOR).

//AND GATE

/\*`timescale 1ns / 1ps

module andGate;

// Inputs

reg x;

reg y;

// Outputs

wire z;

// Instantiate the Unit Under Test (UUT)

andComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

//#20 x = 1;

//#40;

end

initial begin

$monitor("x=%d,y=%d,z=%d \n",x,y,z);

end

endmodule

module andComp(

input x,

input y,

output z

);

assign z = x&y;

endmodule

/\*

OUTPUT

x y Z

0 0 0

0 1 0

1 0 0

1 1 1

\*/

\*/

//NAND GATE

/\*`timescale 1ns / 1ps

module nandGate;

// Inputs

reg x;

reg y;

// Outputs

wire z;

// Instantiate the Unit Under Test (UUT)

nandComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

//#20 x = 1;

//#40;

end

initial begin

$monitor("x=%d,y=%d,z=%d \n",x,y,z);

end

endmodule

module nandComp(

input x,

input y,

output z

);

assign z = ~(x&y);

endmodule

/\*

OUTPUT

x y Z

0 0 1

0 1 1

1 0 1

1 1 0

\*/

\*/

//NOR GATE

/\*`timescale 1ns / 1ps

module norGate;

// Inputs

reg x;

reg y;

// Outputs

wire z;

// Instantiate the Unit Under Test (UUT)

norComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

//#20 x = 1;

//#40;

end

initial begin

$monitor("x=%d,y=%d,z=%d \n",x,y,z);

end

endmodule

module norComp(

input x,

input y,

output z

);

assign z = ~(x|y);

endmodule

/\*

OUTPUT

x y Z

0 0 1

0 1 0

1 0 0

1 1 0

\*/

\*/

//OR GATE

/\*`timescale 1ns / 1ps

module orGate;

// Inputs

reg x;

reg y;

// Outputs

wire z;

// Instantiate the Unit Under Test (UUT)

orComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

//#20 x = 1;

//#40;

end

initial begin

$monitor("x=%d,y=%d,z=%d \n",x,y,z);

end

endmodule

module orComp(

input x,

input y,

output z

);

assign z = (x|y);

endmodule

/\*

OUTPUT

x y Z

0 0 0

0 1 1

1 0 1

1 1 1

\*/

\*/

//XOR GATE

/\*`timescale 1ns / 1ps

module xorGate;

// Inputs

reg x;

reg y;

// Outputs

wire z;

// Instantiate the Unit Under Test (UUT)

xorComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

#20 y = 1;

#20 y=0;

x=1;

#20 x = 1;

y=1;

//#20 x = 1;

//#40;

end

initial begin

$monitor("x=%d,y=%d,z=%d \n",x,y,z);

end

endmodule

module xorComp(

input x,

input y,

output z

);

assign z = ~((x&y)|(~x&~y));

endmodule

/\*

OUTPUT

x y Z

0 0 0

0 1 1

1 0 1

1 1 0

\*/

\*/

//XNOR GATE

/\*

`timescale 1ns / 1ps

module xnorGate;

// Inputs

reg x;

reg y;

// Outputs

wire z;

// Instantiate the Unit Under Test (UUT)

xnorComp uut (

.x(x),

.y(y),

.z(z)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

#20 y = 1;

#20 y=0;

x=1;

#20 x = 1;

y=1;

//#20 x = 1;

//#40;

end

initial begin

$monitor("x=%d,y=%d,z=%d \n",x,y,z);

end

endmodule

module xnorComp(

input x,

input y,

output z

);

assign z = ((x&y)|(~x&~y));

endmodule

/\*

OUTPUT

x y Z

0 0 1

0 1 0

1 0 0

1 1 1

\*/

\*/

//NOT GATE

/\* `timescale 1ns / 1ps

module notGate;

// Inputs

reg x;

// Outputs

wire z;

// Instantiate the Unit Under Test (UUT)

notComp uut (

.x(x),

.z(z)

);

initial begin

// Initialize Inputs

x = 0;

#20 x = 1;

end

initial begin

$monitor("x=%d,z=%d \n",x,z);

end

endmodule

module notComp(

input x,

output z

);

assign z = ~x;

endmodule

/\*

1. To design and verify a half adder using S= (x+y)(x’+y’) C= xy

//HALF ADDER

`timescale 1ns / 1ps

module halfAdder;

// Inputs

reg x;

reg y;

// Outputs

wire z;

wire c;

// Instantiate the Unit Under Test (UUT)

halfAdderComp uut (

.x(x),

.y(y),

.S(S),

.C(C)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

#20 y = 1;

#20 y=0;

x=1;

#20 x = 1;

y=1;

//#20 x = 1;

//#40;

end

initial begin

$monitor("x=%d,y=%d\tS=%d,C=%d \n",x,y,S,C);

end

endmodule

module halfAdderComp(

input x,

input y,

output S,

output C

);

assign S = ((x|y)&(~x|~y));

assign C = x&y;

endmodule

1. To design and verify a full adder using   
   S = x’y’z+x’yz’+xy’z’+xyz and C=xy+xz+yz

module fulladder(x,y,z,s,c);

input x,y,z;

output s,c;

assign s = (~x & ~y & z) | (~x & y & ~z) | (x & ~y & ~z) | (x & y & z);

assign c = (x&y) | (x&z) | (y&z);

endmodule

module test();

reg x,y,z;

wire s,c;

fulladder fa(x,y,z,s,c);

initial

begin

x=0;y=0;z=0;

#1 x=0;y=0;z=1;

#1 x=0;y=1;z=0;

#1 x=0;y=1;z=1;

#1 x=1;y=0;z=0;

#1 x=1;y=0;z=1;

#1 x=1;y=1;z=0;

#1 x=1;y=1;z=1;

end

initial

begin

$dumpfile("fulladder.vcd");

$dumpvars();

$display("X\t Y\t Z\t S\t C");

$monitor("%d\t%d\t%d\t%d\t%d",x,y,z,s,c);

end

endmodule

1. Design a BCD to Excess 3 code converter using combinational circuits.

module bcd2ex3(b3,b2,b1,b0,e3,e2,e1,e0);

input b3,b2,b1,b0;

output e3,e2,e1,e0;

assign e3= ~b3;

assign e2= (~b2 & ~b3) | (b2 & b3);

assign e1 = (~b1 & b2) | (~b1 & b3) | (b1 & ~b2 & ~b3);

assign e0 = b0 | (b1 & b3) | (b1 & b2);

endmodule

module bcd2ex3\_tb;

wire t\_e3,t\_e2,t\_e1,t\_e0;

reg t\_b0,t\_b1,t\_b2,t\_b3;

bcd2ex3 mygate(t\_b3,t\_b2,t\_b1,t\_b0,t\_e3,t\_e2,t\_e1,t\_e0);

initial

begin

$monitor(t\_b0," ",t\_b1," ",t\_b2," ",t\_b3," ",t\_e0," ",t\_e1," ",t\_e2," ",t\_e3);

t\_b0=1'b0;

t\_b1=1'b0;

t\_b2=1'b0;

t\_b3=1'b0;

#5

t\_b0=1'b0;

t\_b1=1'b0;

t\_b2=1'b0;

t\_b3=1'b1;

#5

t\_b0=1'b0;

t\_b1=1'b0;

t\_b2=1'b1;

t\_b3=1'b0;

#5

t\_b0=1'b0;

t\_b1=1'b0;

t\_b2=1'b1;

t\_b3=1'b1;

#5

t\_b0=1'b0;

t\_b1=1'b1;

t\_b2=1'b0;

t\_b3=1'b0;

#5

t\_b0=1'b0;

t\_b1=1'b1;

t\_b2=1'b0;

t\_b3=1'b1;

#5

t\_b0=1'b0;

t\_b1=1'b1;

t\_b2=1'b1;

t\_b3=1'b0;

#5

t\_b0=1'b0;

t\_b1=1'b1;

t\_b2=1'b1;

t\_b3=1'b1;

#5

t\_b0=1'b1;

t\_b1=1'b0;

t\_b2=1'b0;

t\_b3=1'b0;

#5

t\_b0=1'b1;

t\_b1=1'b0;

t\_b2=1'b0;

t\_b3=1'b1;

end

endmodule

1. Design a Excess 3 code to BCD converter using combinational circuits.

module bcd2ex3(b3,b2,b1,b0,e3,e2,e1,e0);

input b3,b2,b1,b0;

output e3,e2,e1,e0;

assign e3= ~b3;

assign e2= (((~b2) & b3) | (b2 & (~b3)));

assign e1 = (((~b1) & (~b2)) | ((~b1) & (~b3)) | (b1 & b2 & b3));

assign e0 = (b0 & b1) | (b0 & b2 & b3);

endmodule

module bcd2ex3\_tb;

wire t\_e3,t\_e2,t\_e1,t\_e0;

reg t\_b0,t\_b1,t\_b2,t\_b3;

bcd2ex3 mygate(t\_b3,t\_b2,t\_b1,t\_b0,t\_e3,t\_e2,t\_e1,t\_e0);

initial

begin

$monitor(t\_b0," ",t\_b1," ",t\_b2," ",t\_b3," ",t\_e0," ",t\_e1," ",t\_e2," ",t\_e3);

t\_b0=1'b0;

t\_b1=1'b0;

t\_b2=1'b1;

t\_b3=1'b1;

#5

t\_b0=1'b0;

t\_b1=1'b1;

t\_b2=1'b0;

Name-Saurav Kumar

Roll No. 101903289

Group- 4CO11

t\_b3=1'b0;

#5

t\_b0=1'b0;

t\_b1=1'b1;

t\_b2=1'b0;

t\_b3=1'b1;

#5

t\_b0=1'b0;

t\_b1=1'b1;

t\_b2=1'b1;

t\_b3=1'b0;

#5

t\_b0=1'b0;

t\_b1=1'b1;

t\_b2=1'b1;

t\_b3=1'b1;

#5

t\_b0=1'b1;

t\_b1=1'b0;

t\_b2=1'b0;

t\_b3=1'b0;

#5

t\_b0=1'b1;

t\_b1=1'b0;

t\_b2=1'b0;

t\_b3=1'b1;

#5

t\_b0=1'b1;

t\_b1=1'b0;

t\_b2=1'b1;

t\_b3=1'b0;

#5

t\_b0=1'b1;

t\_b1=1'b0;

t\_b2=1'b1;

t\_b3=1'b1;

#5

t\_b0=1'b1;

t\_b1=1'b1;

t\_b2=1'b0;

t\_b3=1'b0;

end

endmodule

1. To design and verify a half subtractor using D = x’y +xy’ and B=x’y.

`timescale 1ns / 1ps

module halfSub;

// Inputs

reg x;

reg y;

// Outputs

wire D;

wire B;

// Instantiate the Unit Under Test (UUT)

halfSubComp uut (

.x(x),

.y(y),

.D(D),

.B(B)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

#20 y = 1;

#20 y=0;

x=1;

#20 x = 1;

y=1;

//#20 x = 1;

//#40;

end

initial begin

$display("Half Subtractor\n");

$display("X Y\tD B");

$monitor("%d %d\t%d %d \n",x,y,D,B);

end

endmodule

module halfSubComp(

input x,

input y,

output D,

output B

);

assign D = (x&(~y))|((~x)&y);

assign B = ~x&y;

endmodule

/\*

Output

x y D B

0 0 0 0

0 1 1 1

1 0 1 0

1 1 0 0

\*/