**EMBEDDED SYSTEMS DESIGN- UCS704**

**Lab Assignment - 1**

**Submitted By**

**Devansh Gupta 102103028**

# B.E 4COE1

**A logo for a university

Description automatically generated**

# Computer Science & Engineering Department

**Thapar Institute of Engineering & Technology, Patiala**

**July – December 2024**

**1.) Logic Gates**

**(a)** **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,

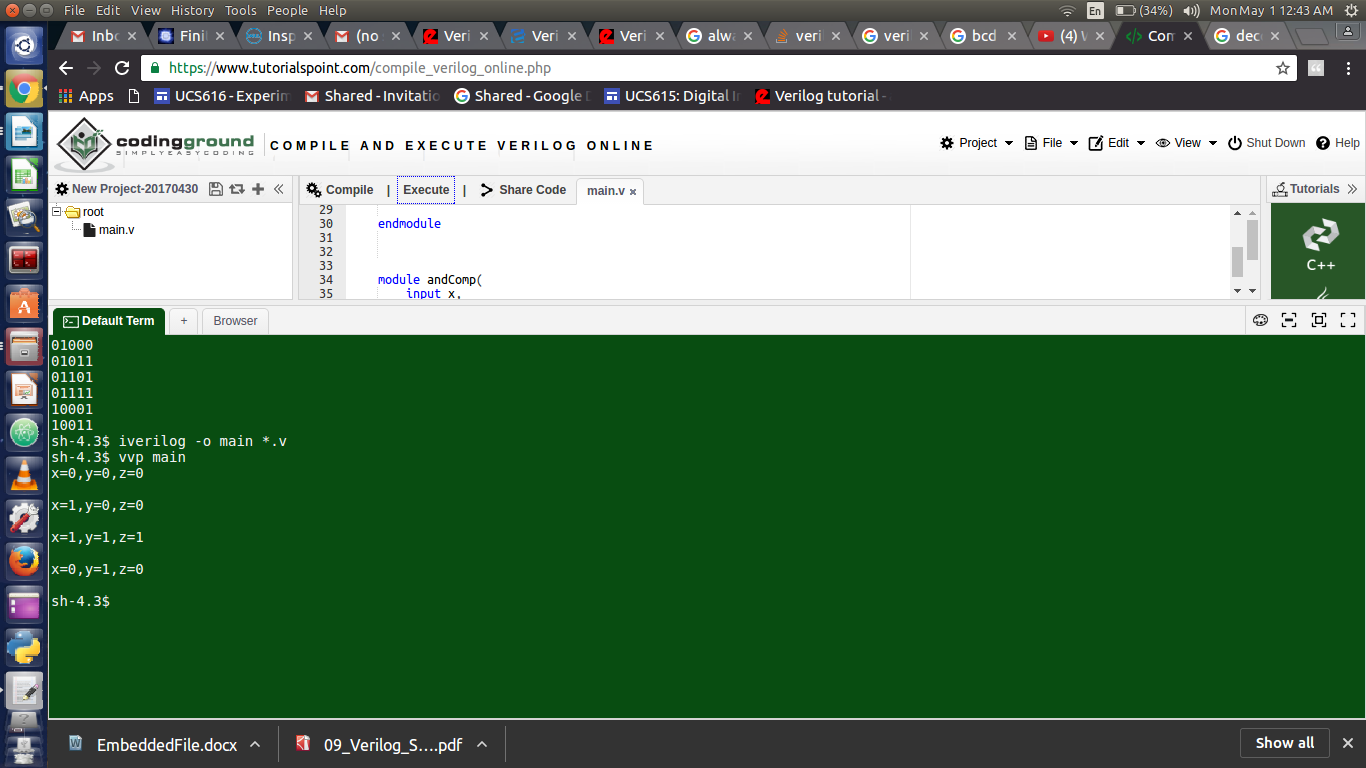
input y,

output z

);

assign z = x&y;

endmodule



**(b)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

x = 0;

y = 0;

#20 x = 1;

#20 y = 1;

#20 x = 0;

end

initial begin

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

end

endmodule

module nandComp(

input x,

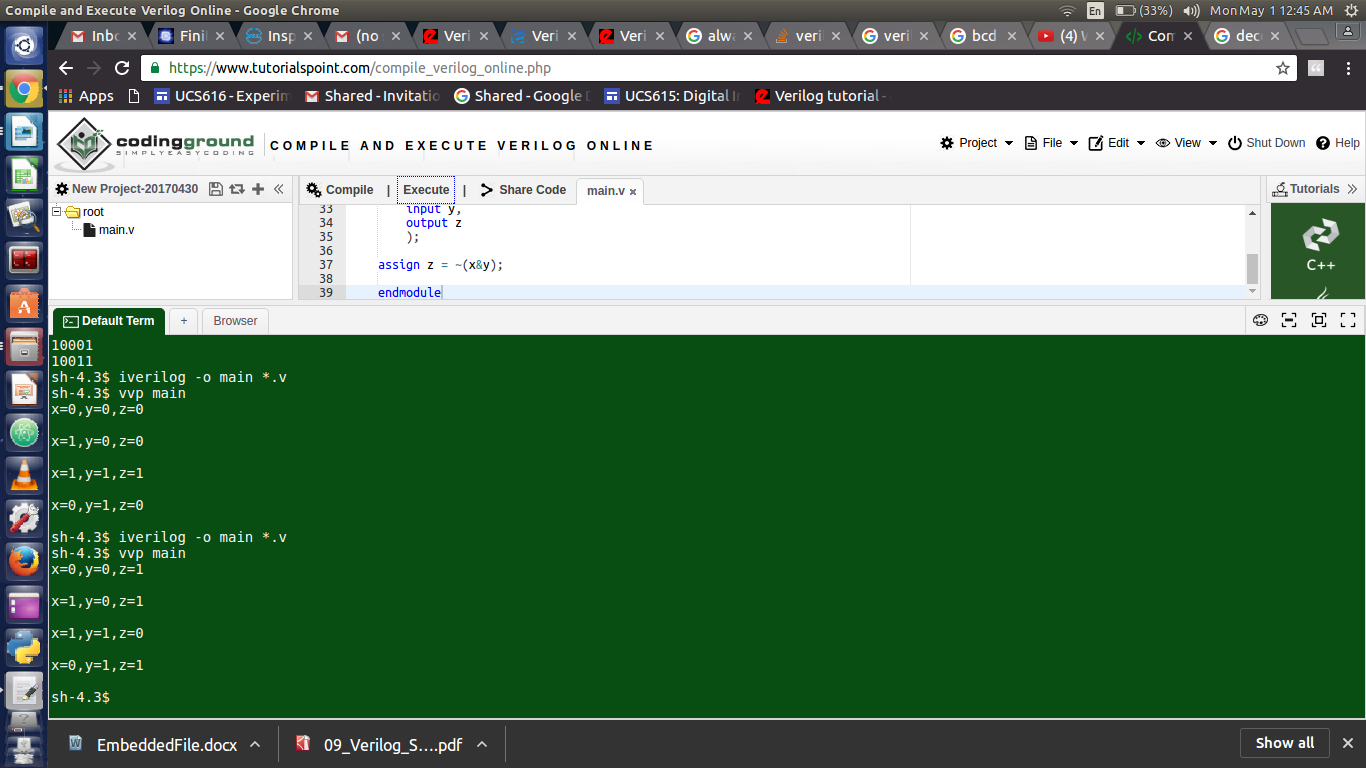
input y,

output z

);

assign z = ~(x&y);

endmodule



**(c)Nor Gate**

`timescale 1ns / 1ps

module norGate;

// Inputs

reg x;

reg y;

// Outputs

wire z;

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,

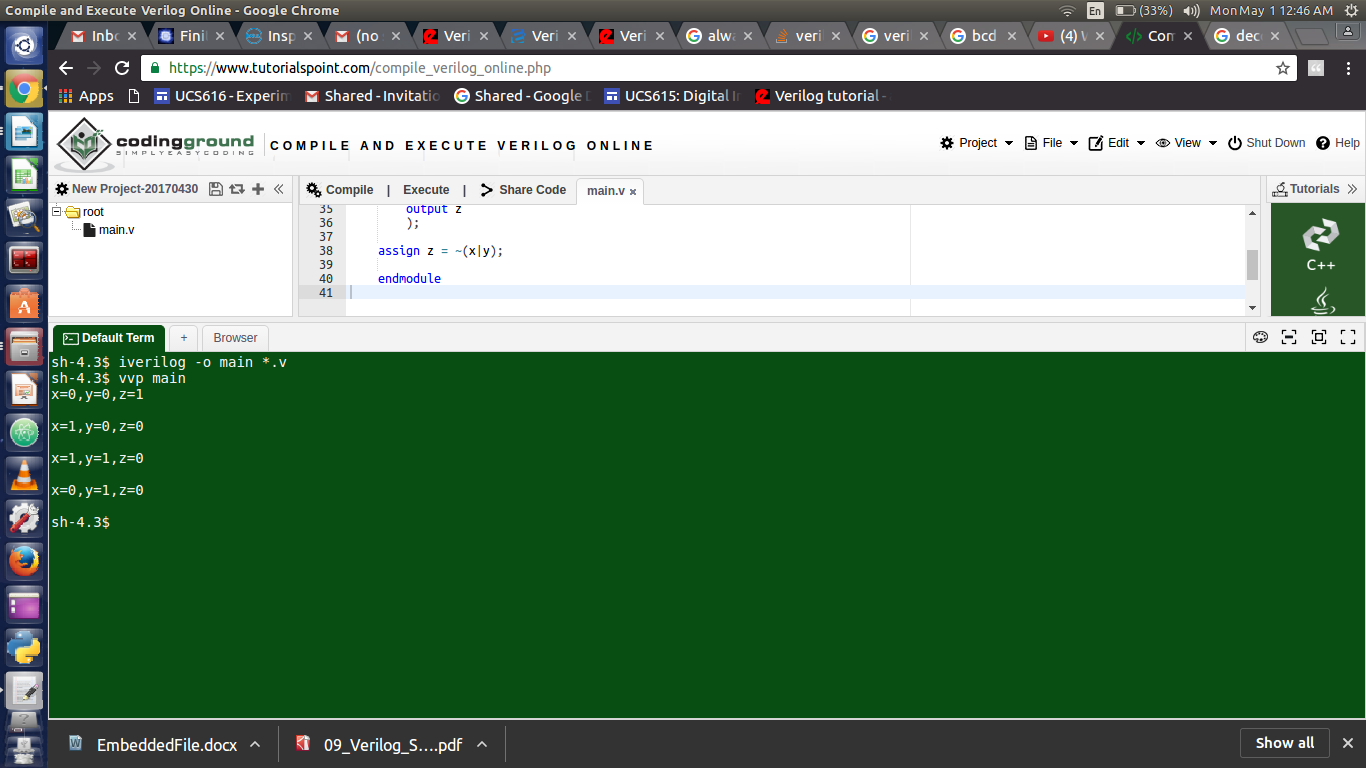
input y,

output z

);

assign z = ~(x|y);

endmodule



**(d)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(

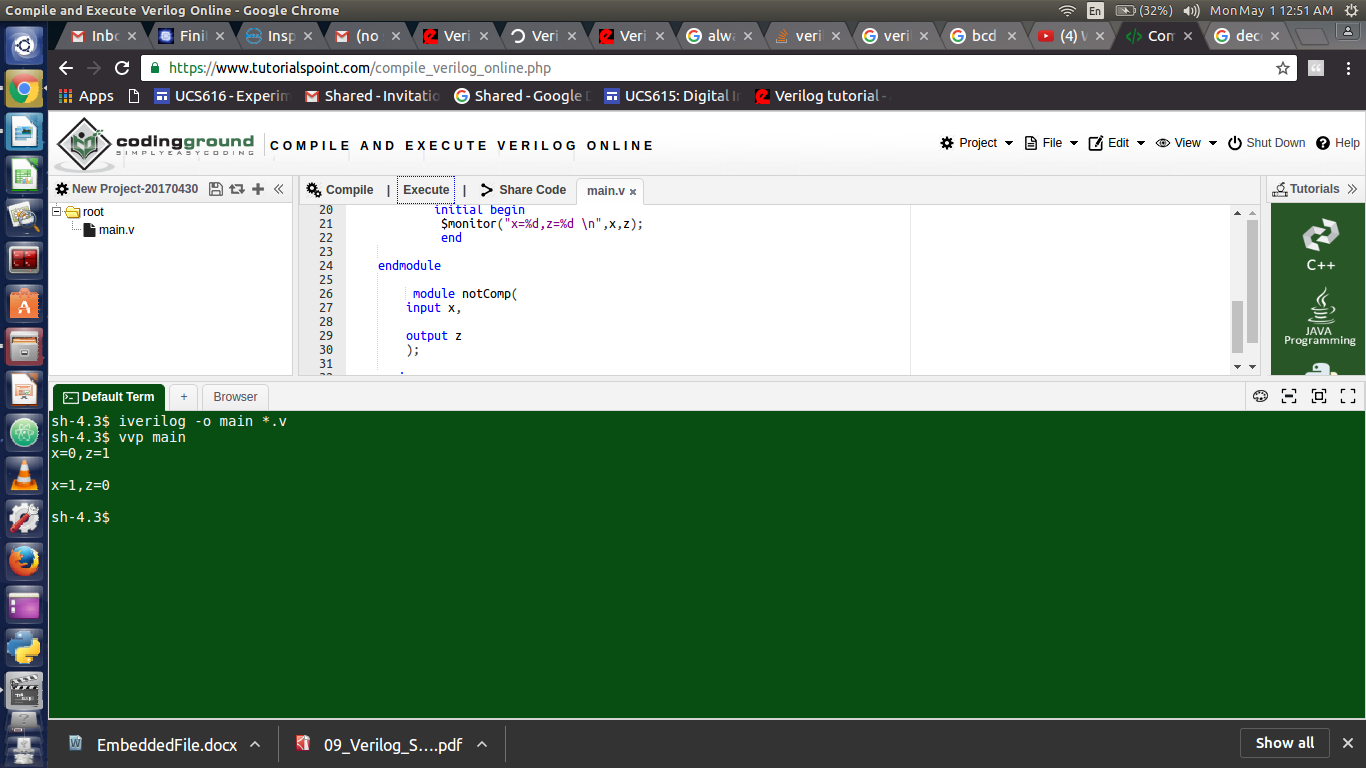
input x,

output z

);

assign z = ~x;

endmodule



**(e)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,

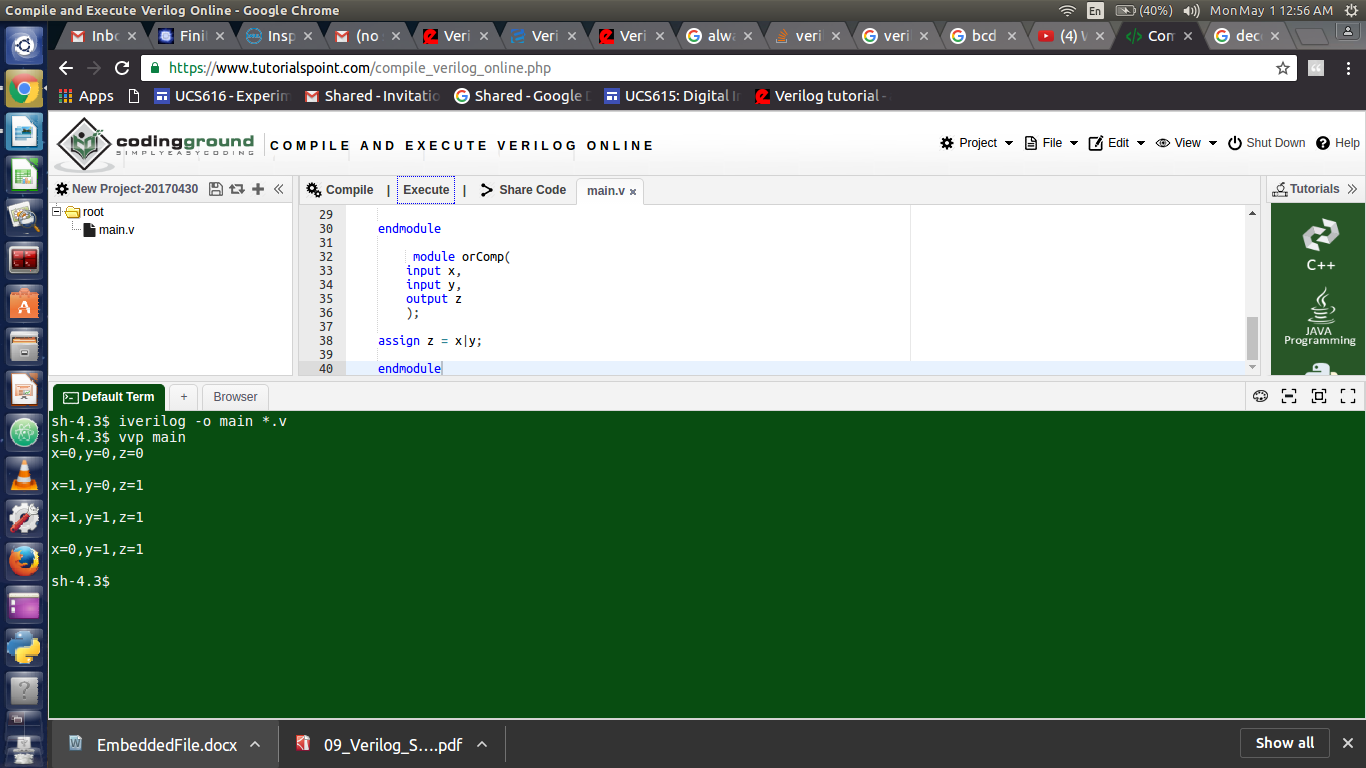
input y,

output z

);

assign z = x|y;

endmodule



**(f)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(

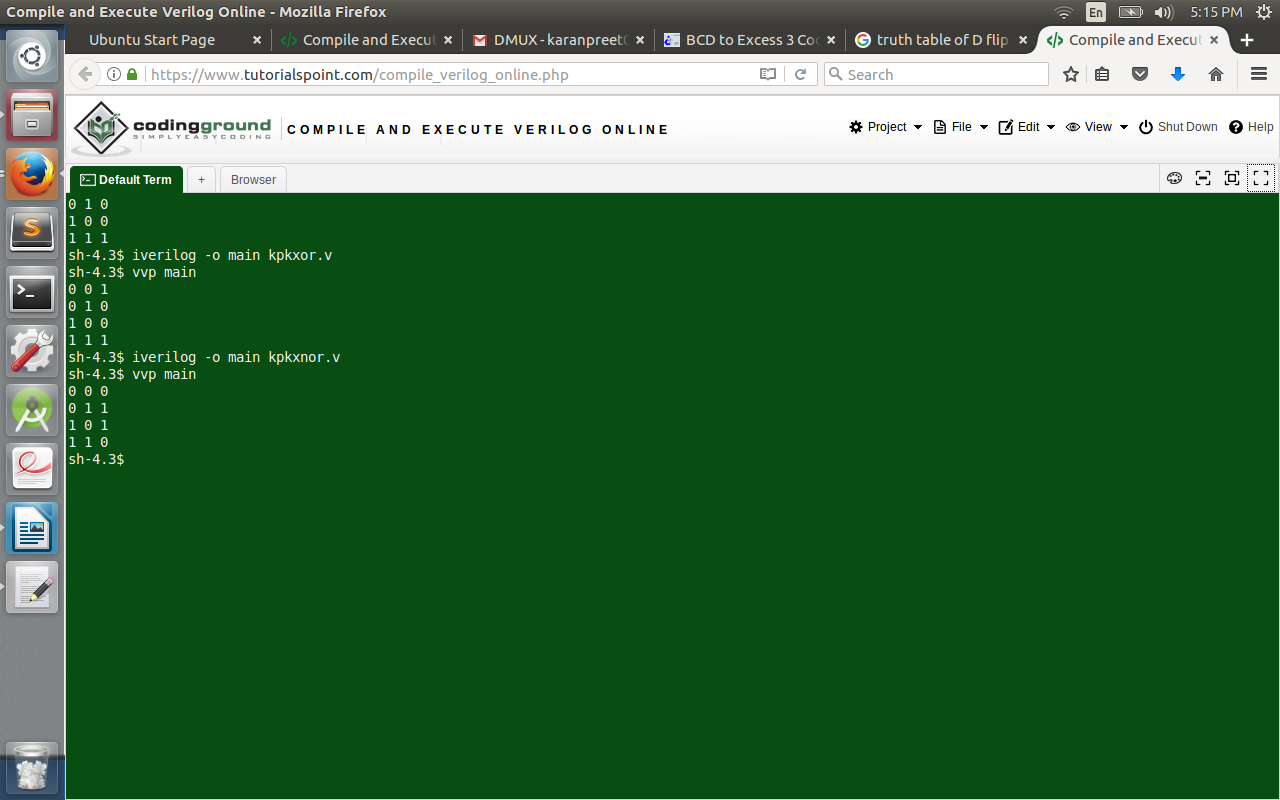
input x,

input y,

output z

);

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

 endmodule

**(g)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,

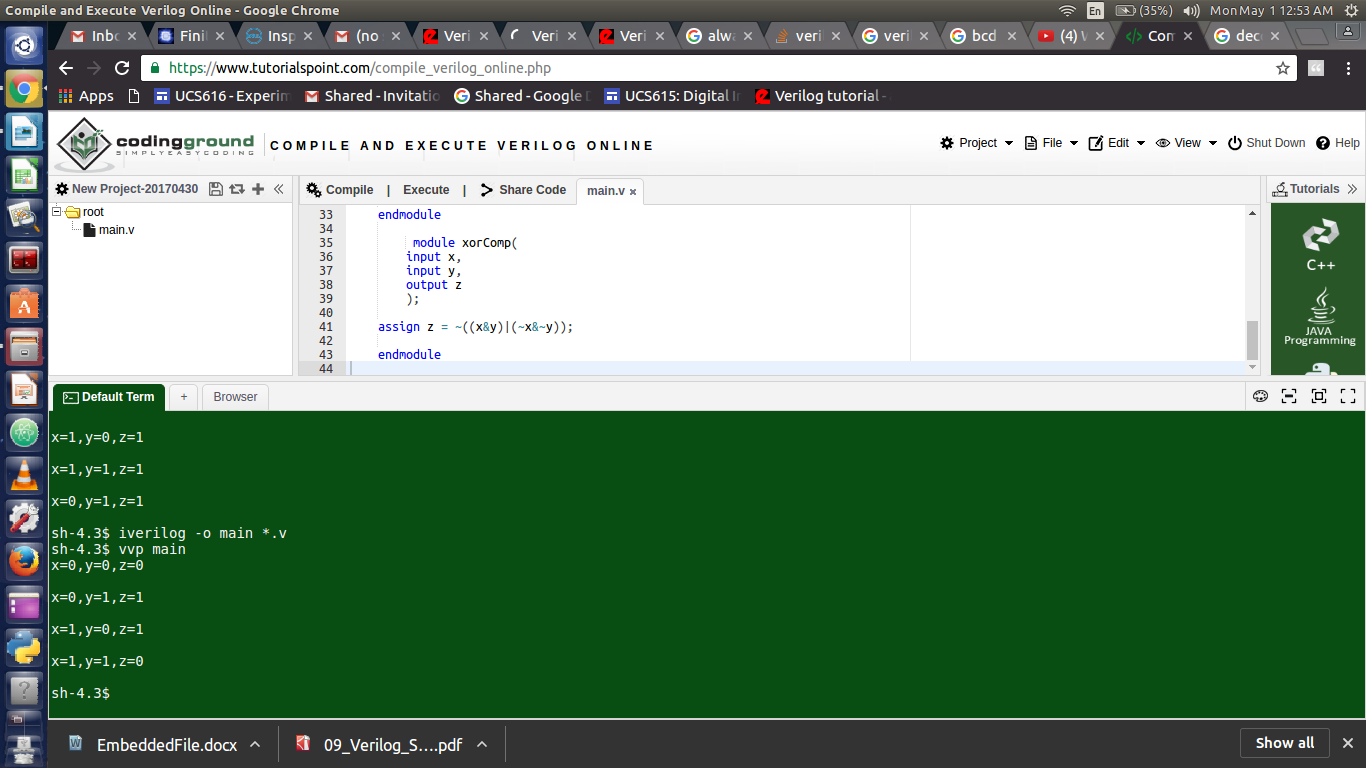
input y,

output z

);

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

endmodule



**2.) 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

/\*

x y S C

0 0 0 0

0 1 1 0

1 0 1 0

1 1 0 1

\*/

module halfAdderComp(

input x,

input y,

output S,

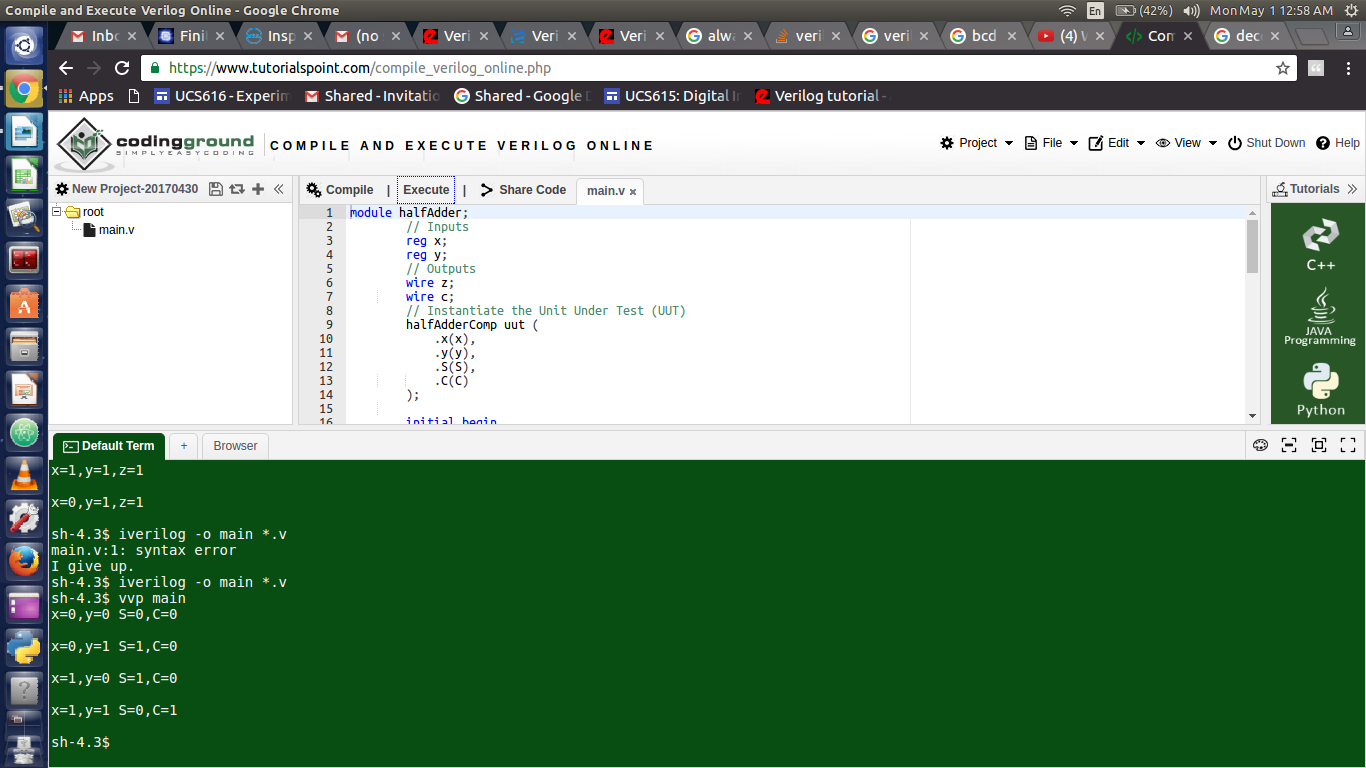
output C

);

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

assign C = x&y;

endmodule



**3.) Full Adder**

`timescale 1ns / 1ps

module fullAdder;

// Inputs

reg x;

reg y;

reg Ci;

// Outputs

wire S;

wire Cout;

// Instantiate the Unit Under Test (UUT)

fullAdderComp uut (

.x(x),

.y(y),

.Ci(Ci),

.S(S),

.Cout(Cout)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

Ci=0;

#20 Ci=1;

#20 y = 1;

Ci=0;

#20 Ci=1;

#20 y=0;

x=1;

Ci=0;

#20 Ci=1;

#20 x = 1;

y=1;

Ci=0;

#20 Ci=1;

//#20 x = 1;

//#40;

end

initial begin

$display("Full Adder\n");

$display("X Y Ci\tS Cout");

$monitor("%d %d %d \t%d %d \n",x,y,Ci,S,Cout);

end

endmodule

module fullAdderComp(

input x,

input y,

input Ci,

output S,

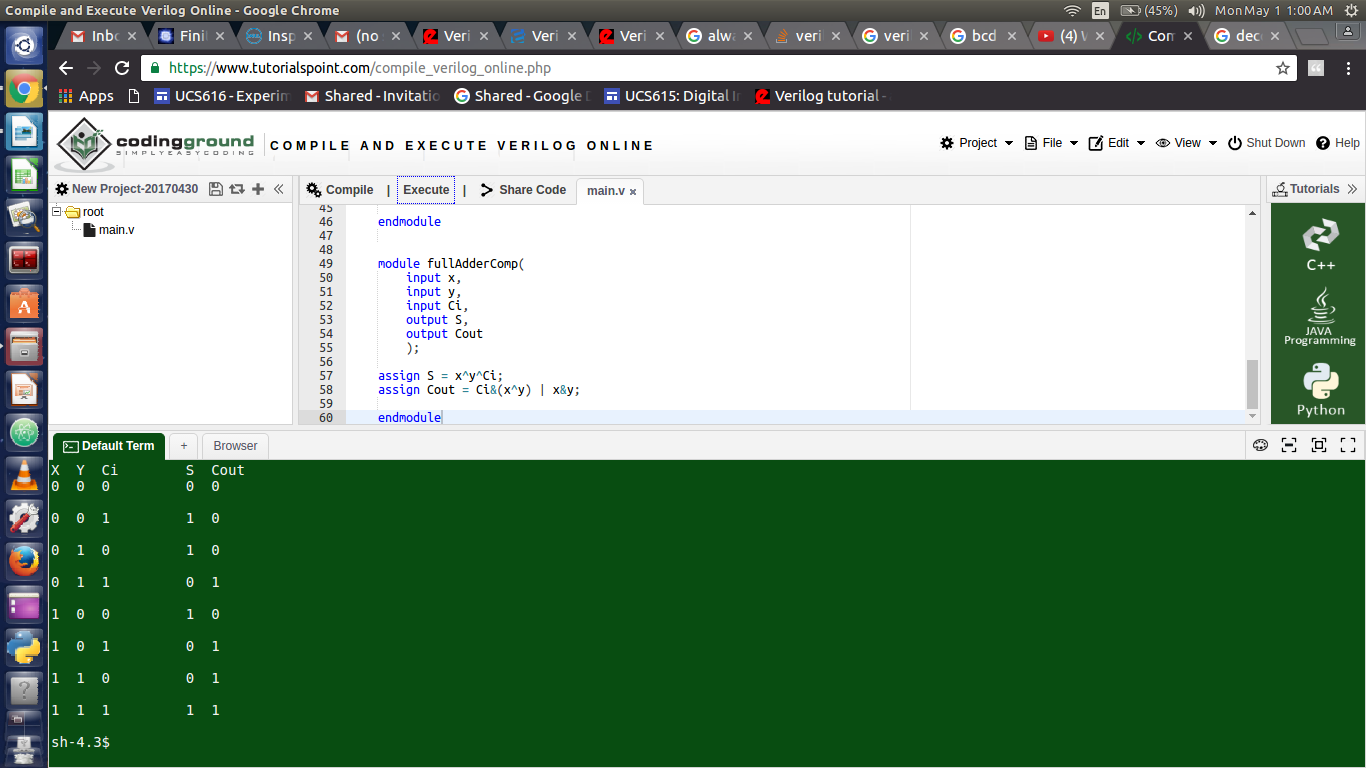
output Cout

);

assign S = x^y^Ci;

assign Cout = Ci&(x^y) | x&y;

endmodule



**4.) Half Subtractor**

`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

/\*

x y D B

0 0 0 0

0 1 1 1

1 0 1 0

1 1 0 0

\*/

module halfSubComp(

input x,

input y,

output D,

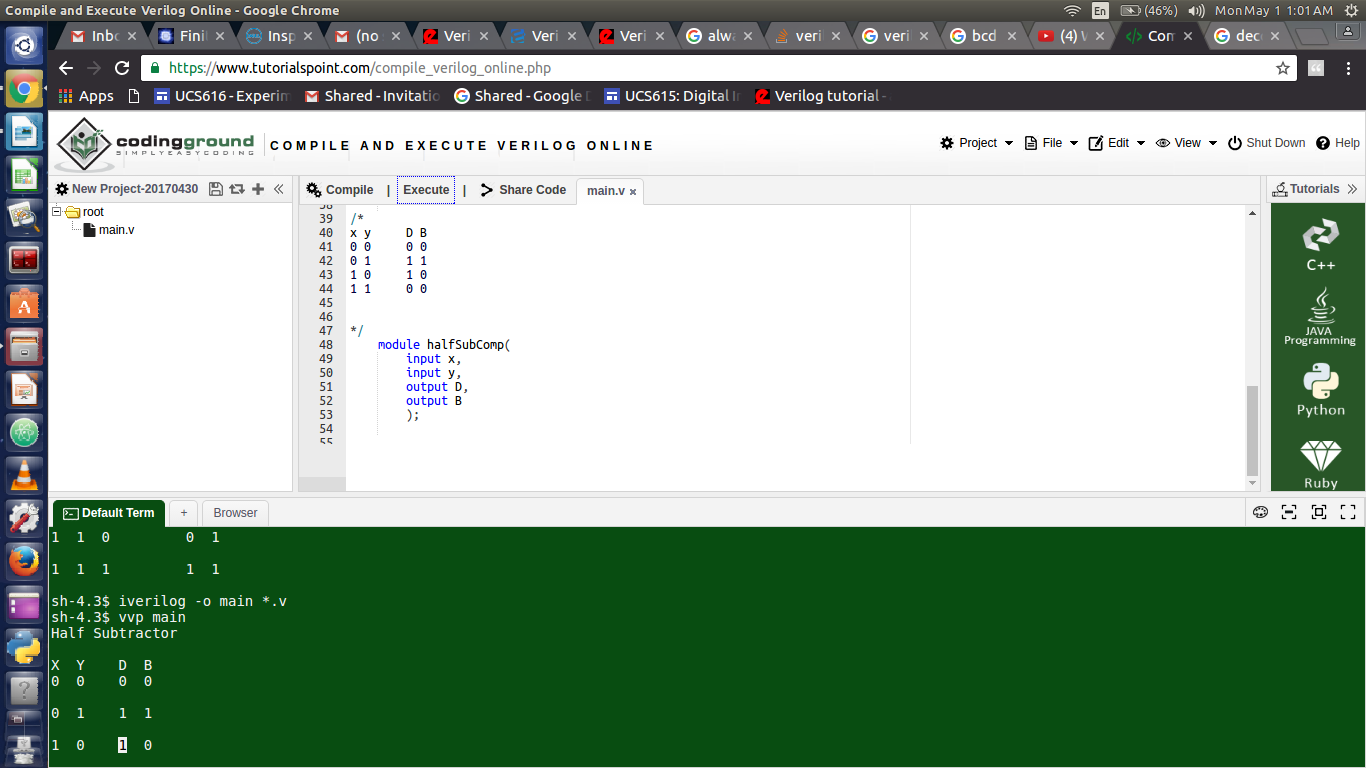
output B

);

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

assign B = ~x&y;

endmodule



**5.) Full Subtractor**

`timescale 1ns / 1ps

module fullSub;

// Inputs

reg x;

reg y;

reg Bi;

// Outputs

wire D;

wire Bout;

// Instantiate the Unit Under Test (UUT)

fullSubComp uut (

.x(x),

.y(y),

.Bi(Bi),

.D(D),

.Bout(Bout)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

Bi=0;

#20 Bi=1;

#20 y = 1;

Bi=0;

#20 Bi=1;

#20 y=0;

x=1;

Bi=0;

#20 Bi=1;

#20 x = 1;

y=1;

Bi=0;

#20 Bi=1;

//#20 x = 1;

//#40;

end

initial begin

$display("Full Subtractor\n");

$display("X Y Bi\tD Bout");

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

end

endmodule

/\*

x y Bi D Bout

0 0 0 0 0

0 0 1 1 1

0 1 0 1 1

0 1 1 0 1

1 0 0 1 0

1 0 1 0 0

1 1 0 0 0

1 1 1 1 1

\*/

// D=X-Y-Bi

//D--- ~x~yz + ~xy~z + x~y~z + xyz

// ~x(y^z) +x~(y^z)

// x^(y^z)

// ~X(Y~Z + ~YZ) + YZ

//

module fullSubComp(

input x,

input y,

input Bi,

output D,

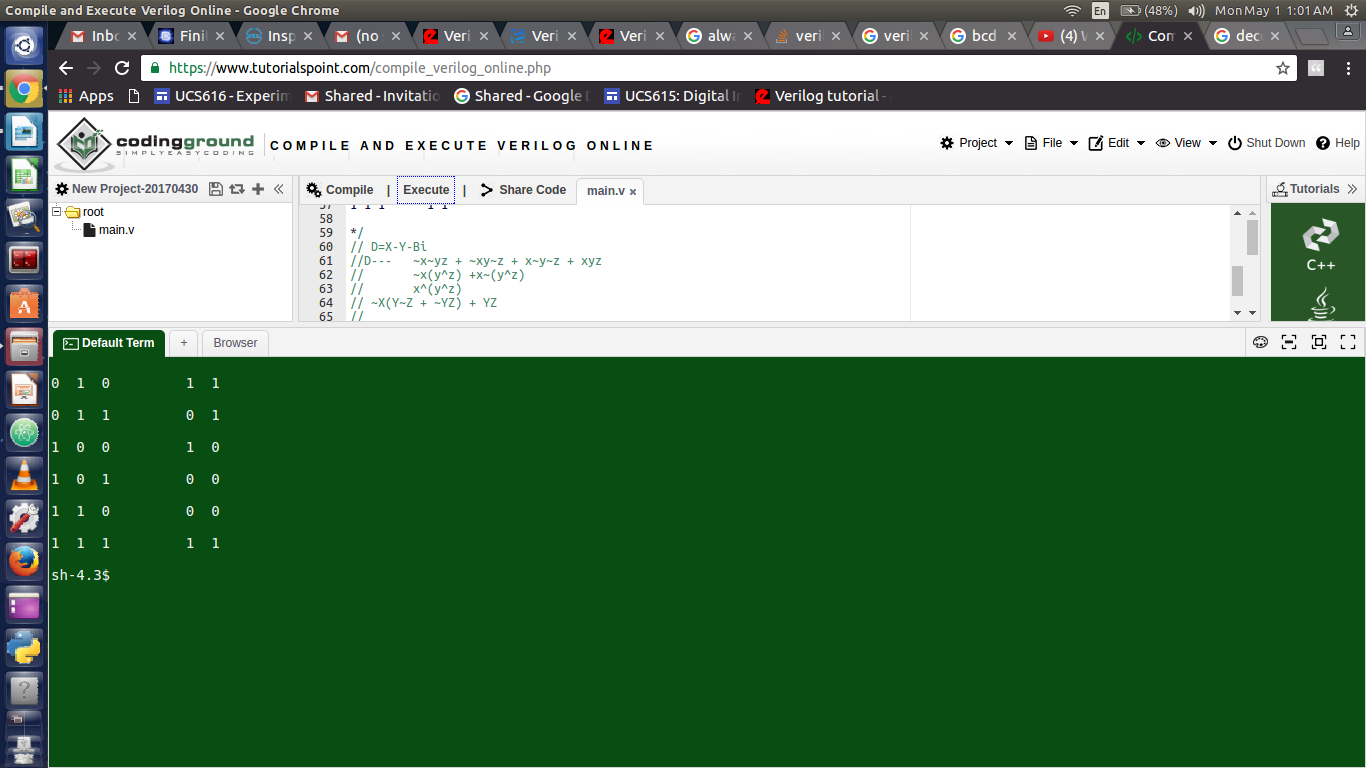
output Bout

);

assign D = x^y^Bi;

assign Bout = (~x)&((y&(~Bi))|((~y)&Bi))|(y&Bi);

endmodule



**6.) Number Converter**

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;

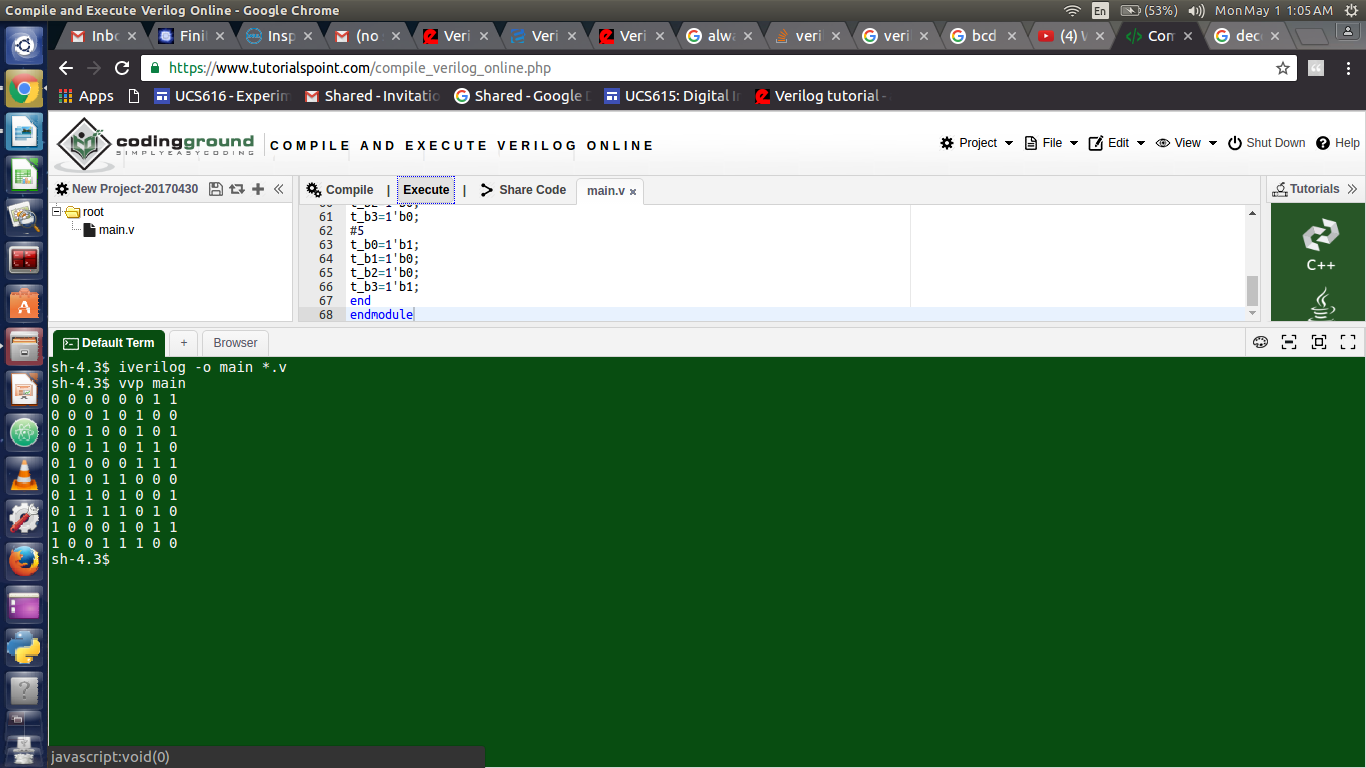
t\_b1=1'b0;

t\_b2=1'b0;

t\_b3=1'b1;

end

endmodule



**7.) 4\*1 Mux**

`timescale 1ns / 1ps

module MUX41Comp(

input wire i1,

input i2,

input i3,

input i4,

input wire s1,

input s2,

output o

);

assign o=((~s1)&(~s2)&i1)|((~s1)&s2&i2)|(s1&(~s2)&i3)|(s1&s2&i4);

// if (s1) begin

// assign o = i1;

// end

endmodule

module MUX41;

// Inputs

reg s1;

reg s2;

reg i1;

reg i2;

reg i3;

reg i4;

// Outputs

wire o;

// Instantiate the Unit Under Test (UUT)

MUX41Comp uut (

.s1(s1),

.s2(s2),

.i1(i1),

.i2(i2),

.i3(i3),

.i4(i4),

.o(o)

);

initial begin

// Initialize Inputs

i1=1;

i2=0;

i3=0;

i4=1;

s1=0;

s2=0;

#20 s2=1;

#20 s1=1;

s2=0;

#20 s2=1;

end

initial begin

$display("4\*1 MUX\n");

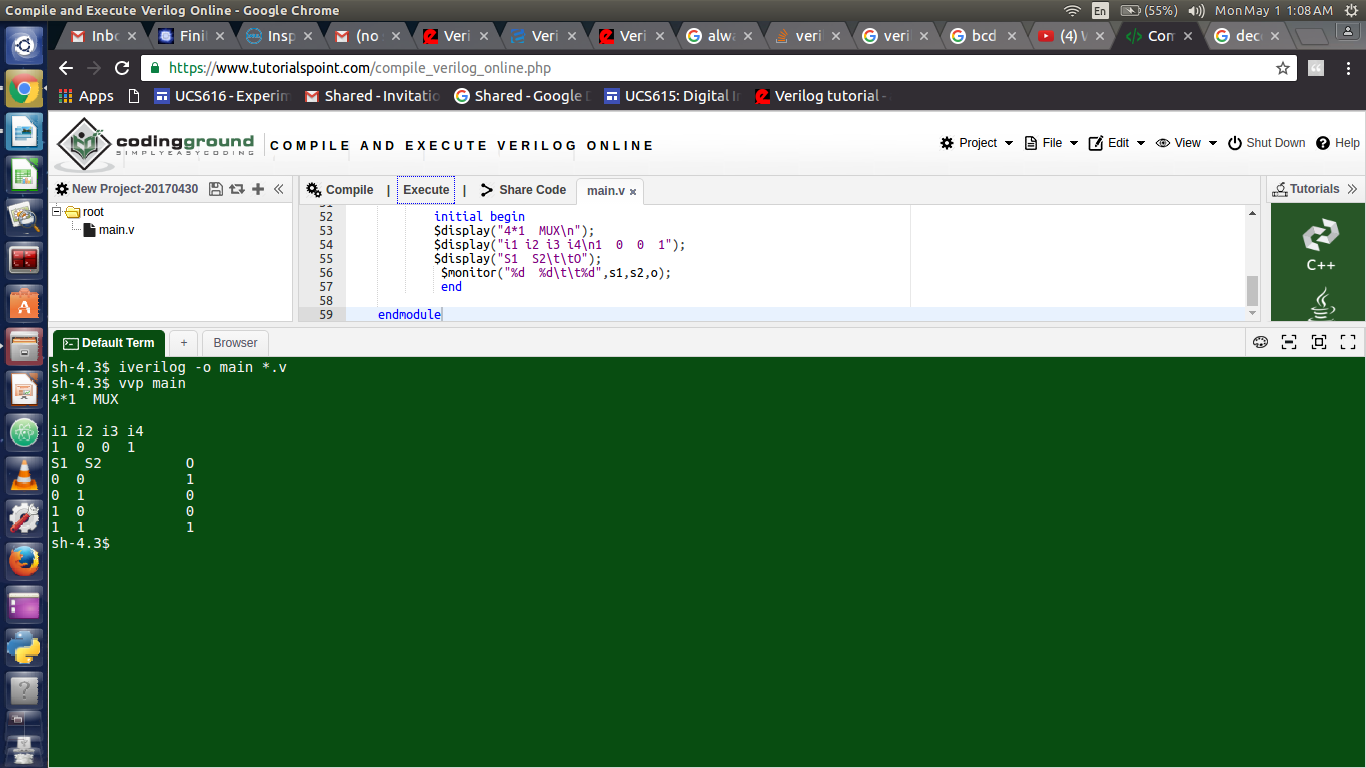
$display("i1 i2 i3 i4\n1 0 0 1");

$display("S1 S2\t\tO");

$monitor("%d %d\t\t%d",s1,s2,o);

end

endmodule



**8.) 1\*4 Demux**

module DEMUX14;

// Inputs

reg i;

reg s1;

reg s2;

// Outputs

wire o1;

wire o2;

wire o3;

wire o4;

DEMUX14Comp Demux (

.i(i),

.s1(s1),

.s2(s2),

.o1(o1),

.o2(o2),

.o3(o3),

.o4(o4)

);

initial begin

i=0;

s1=0;

s2=0;

#20 i=1;

#20 i=0;

s2=1;

#20 i=1;

#20 i=0;

s1=1;

s2=0;

#20 i=1;

#20 i=0;

s2=1;

#20 i=1;

end

initial begin

$display("1\*4 DEMUX\n");

$display("I S1 S2\t\tO1 O2 O3 O4");

$monitor("%d %d %d\t\t%d %d %d %d",i,s1,s2,o1,o2,o3,o4);

end

endmodule

module DEMUX14Comp(

input i,

input s1,

input s2,

output reg o1,

output reg o2,

output reg o3,

output reg o4

);

// assign o1=i&((~s1)&(~s2));

// assign o2=i&((~s1)&s2);

//assign o3=i&(s1&(~s2));

//assign o4=i&(s1&s2);

always @(i) begin

if((~s1)&~s2)

begin

o1=i;

o2=0;

o3=0;

o4=0;

end

else if(~s1&s2)

begin

o1=0;

o2=i;

o3=0;

o4=0;

end

else if(s1&~s2)

begin

o1=0;

o2=0;

o3=i;

o4=0;

end

else if(s1&s2)

begin

o1=0;

o2=0;

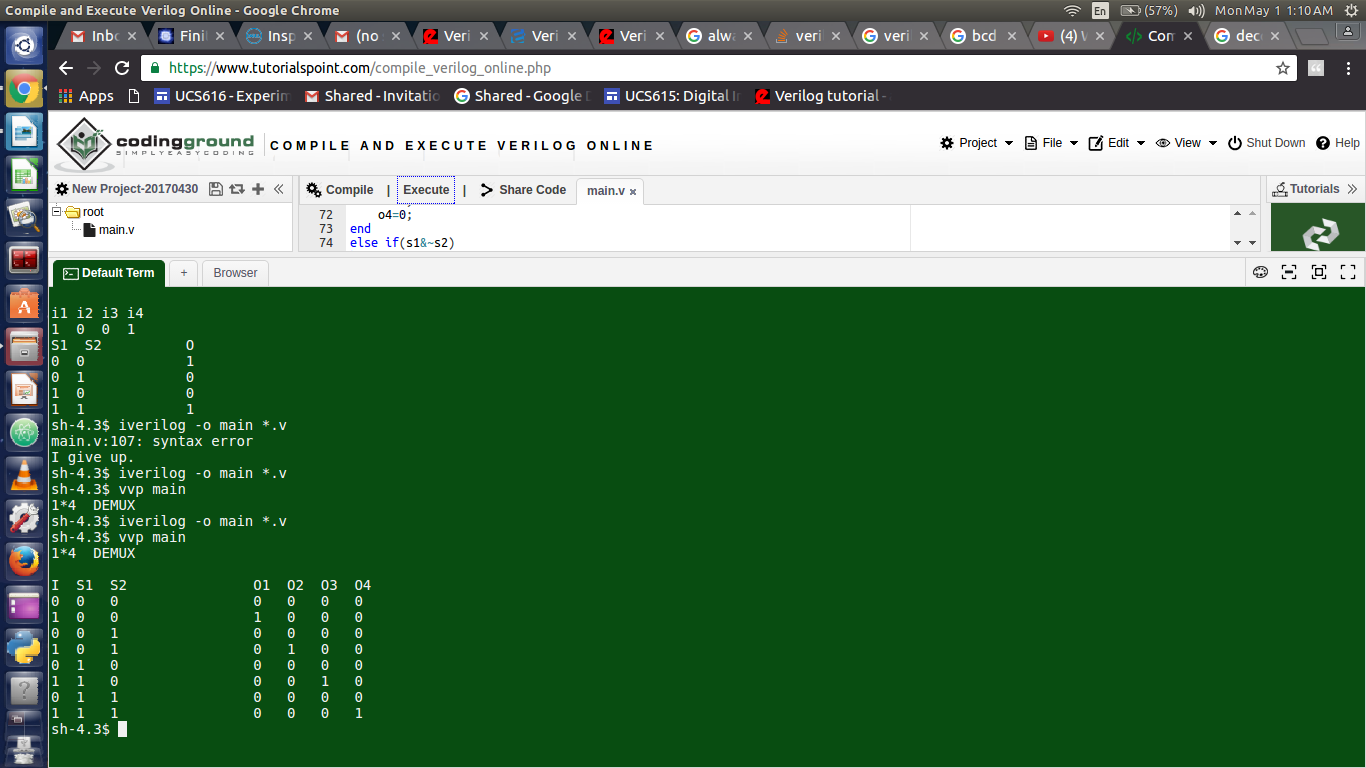
o3=0;

o4=i;

end

end

endmodule



**9.) Encoder**

module Encoder42;

// Inputs

reg i1;

reg i2;

reg i3;

reg i4;

// Outputs

wire o1;

wire o2;

Encoder42Comp enc (

.i1(i1),

.i2(i2),

.i3(i3),

.i4(i4),

.o1(o1),

.o2(o2)

);

initial begin

i1=0;

i2=0;

i3=0;

i4=0;

#20 i1=1;

#20 i1=0;

i2=1;

#20 i3=1;

i2=0;

#20 i3=0;

i4=1;

end

initial begin

$display("4\*2 Encoder\n");

$display("I4 I3 I2 I1\t\tO2 O1");

$monitor("%d %d %d %d\t\t%d %d",i4,i3,i2,i1,o2,o1);

end

endmodule

module Encoder42Comp(

input i1,

input i2,

input i3,

input i4,

output o1,

output o2

);

assign o1=(~i1&i2&~i3&~i4)|(~i1&~i2&~i3&i4);

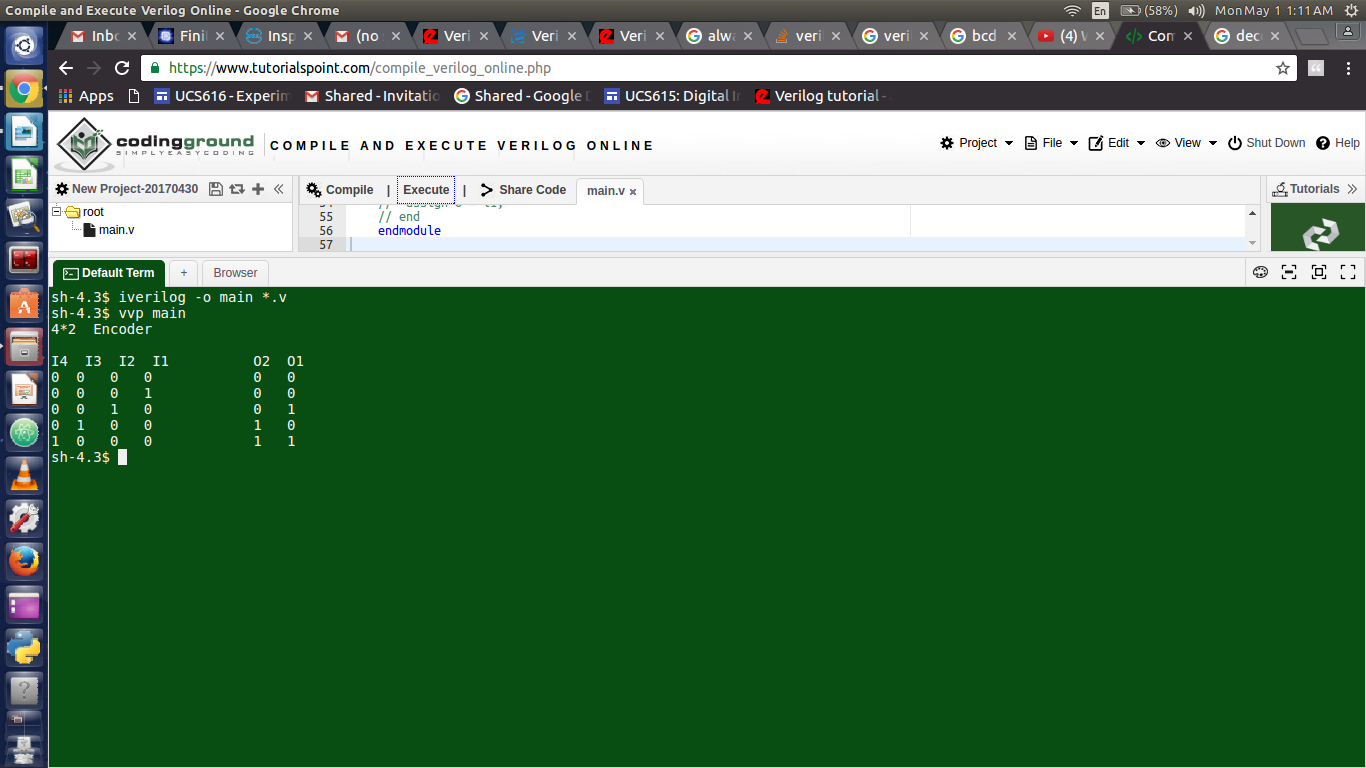
assign o2=(~i1&~i2&i3&~i4)|(~i1&~i2&~i3&i4);

// if (s1) begin

// assign o = i1;

// end

endmodule



**10.) Decoder**

module Decoder24;

// Inputs

reg i0;

reg i1;

reg e;

// Outputs

wire o0;

wire o1;

wire o2;

wire o3;

Decoder24Comp dec (

.i0(i0),

.i1(i1),

.e(e),

.o0(o0),

.o1(o1),

.o2(o2),

.o3(o3)

);

initial begin

i0=0;

i1=0;

e=1;

#20 i0=1;

#20 i0=0;

i1=1;

#20 i0=1;

end

initial begin

$display("2\*4 Decoder\n");

$display("E I1 I0\t\tO0 O1 O2 O3");

$monitor("%d %d %d\t\t%d %d %d %d",e,i1,i0,o0,o1,o2,o3);

end

endmodule

//iverilog -o Decoder24 Decoder24.v Decoder24Comp.v

//vvp Decoder24

module Decoder24Comp(

input i0,

input i1,

input e,

output o0,

output o1,

output o2,

output o3

);

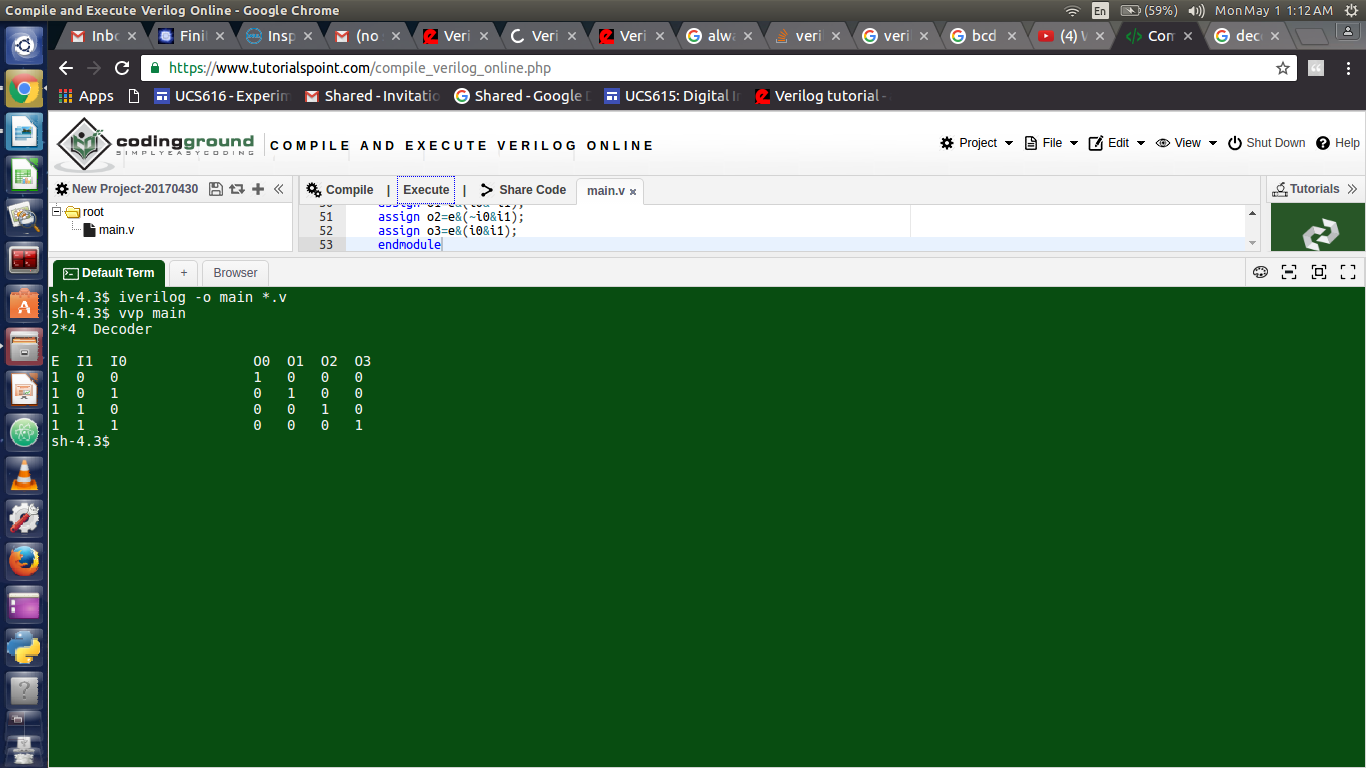
assign o0=e&(~i0&~i1);

assign o1=e&(i0&~i1);

assign o2=e&(~i0&i1);

assign o3=e&(i0&i1);

endmodule



**11.) Flip Flops**

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

input d,clk;

output q,qn;

reg q,qn;

dffComp ff(q,qn,clk,d);

initial begin q=0; qn=1; end

always @(posedge clk)

begin

q = d;

qn = !d;

end

endmodule

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

input q,qn;

output clk,d;

reg clk,d;

initial

begin

clk=0;

d=0; #9 d=1; #1 d=0; #1 d=1; #2 d=0; #1 d=1; #12 d=0;

#1 d=1; #2 d=0; #1 d=1; #1 d=0; #1 d=1; #1 d=0; # 7 d=1;

#8 $finish;

end

always

begin

#4 clk=!clk;

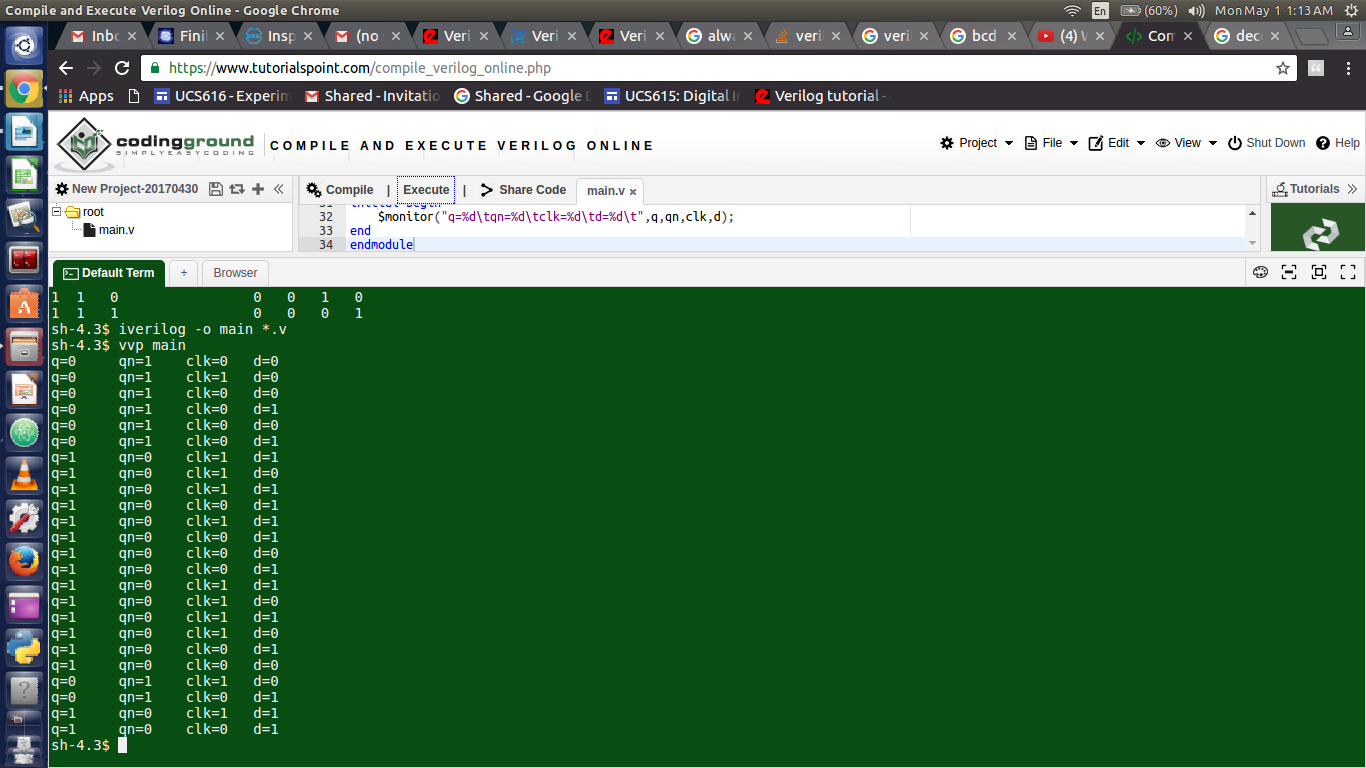
end

initial begin

$monitor("q=%d\tqn=%d\tclk=%d\td=%d\t",q,qn,clk,d);

end

endmodule



**12.) Counter**

`include "cntr4bits.v"

module cntr4bit();

reg clock, reset, enable;

wire [3:0] counter\_out;

initial begin

$display ("time\t clk reset enable counter");

$monitor ("%g\t %b %b %b %b",

$time, clock, reset, enable, counter\_out);

clock = 1;

reset = 0;

enable = 0;

#5 reset = 1;

#10 reset = 0;

#10 enable = 1;

#100 enable = 0;

#30 enable=1;

#50 $finish;

end

always begin

#5 clock = ~clock;

end

cntr4bits cntr (

clock,

reset,

enable,

counter\_out

);

Endmodule

module cntr4bits (clock,reset,enable,counter\_out);

input clock ;

input reset ;

input enable ;

output [3:0] counter\_out ;

wire clock ;

wire reset ;

wire enable ;

reg [3:0] counter\_out ;

always @ (posedge clock)

begin :Counter

if (reset == 1) begin

counter\_out = #1 4'b0000;

end

else if (enable == 1) begin

counter\_out = #1 counter\_out + 1;

end

end

endmodule

