# UCS704 EMBEDDED SYSTEMS DESIGN

**End-Semester Evaluation**

# Submitted by: Name: Manu Agarwal

**Roll no: 102103318**

# BE Final Year, COE Group: COE11

**Submitted to Dr. Anju Bala**

# 

**Experiment 1 (Truth Table and Logic Gates)**

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

**// using a module for my\_and**

module my\_and(input a,input b,output And,output Or,output Not,output Nand,output Nor,output Exor,output Exnor);

assign And = a&b; assign Or = a|b; assign Not = ~a; assign Nand = ~And; assign Nor = ~Or; assign Exor = (a^b); assign Exnor = ~Exor; endmodule

module test; reg a,b;

wire And,Or,Not,Nand,Nor,Exor,Exnor; my\_and

a1(.a(a),.b(b),.And(And),.Or(Or),.Not(Not),.Nand(Nand),.Nor(Nor),.Exor(Exor),.Exnor(Exno r));

initial begin

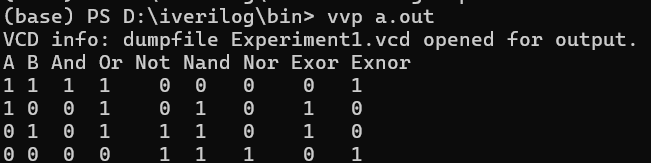
$dumpfile("Experiment1.vcd");

$dumpvars(0,test);

$display("A B And Or Not Nand Nor Exor Exnor");

$monitor("%b %b %b %b %b %b %b %b %b ",a,b,And,Or,Not,Nand,Nor,Exor,Exnor);

a = 1;

b = 1;

#10

a = 1;

b = 0;

#10

a = 0;

b = 1;

#10

a = 0;

b = 0;

#10

$finish; end endmodule

# Experiment 2 (Half Adder)

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

**// half\_adder using builtin gates**

module test; reg a,b;

wire sum, carry; xor x1(sum,a,b); and a1(carry,a,b);

initial begin

$dumpfile("Experiment2.vcd");

$dumpvars(0,test);

$monitor("a = %b, b = %b, sum = %b, carry = %b",a,b,sum,carry); a = 1;

b = 1;

#10

a = 1;

b = 0;

#10

a = 0;

b = 1;

#10

a = 0;

b = 0;

$finish; end endmodule

## // half\_adder using directly equations

module half\_adder(input a, input b, output sum, output carry); assign sum = a^b;

assign carry = a&b; endmodule

module test; reg a,b;

wire sum, carry;

half\_adder h(.a(a),.b(b),.sum(sum),.carry(carry));

initial begin

$dumpfile("Experiment2.vcd");

$dumpvars(0,test);

$monitor("a = %b, b = %b, sum = %b, carry = %b",a,b,sum,carry); #10

a = 1;

b = 1;

#10

a = 1;

b = 0;

#10

a = 0;

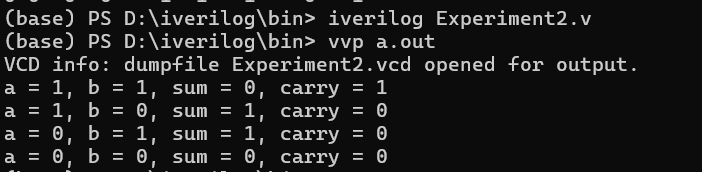
b = 1;

#10

a = 0;

b = 0;

$finish; end endmodule



# Experiment 3 (Full Adder)

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

**// full\_adder using half adder**

module half\_adder(input a,input b,output sum,output carry); assign sum = a^b;

assign carry = a&b; endmodule

module test; reg a,b,c;

wire sum, carry,e1,e2,e3; half\_adder h1(a,b,e1,e2); half\_adder h2(e1,c,sum,e3); or o1(carry,e2,e3);

initial begin

$dumpfile("Experiment3.vcd");

$dumpvars(0,test);

$monitor("a = %b, b = %b, c = %b, sum = %b, carry = %b",a,b,c,sum,carry); a = 1;

b = 1;

c = 1;

#10

a = 1;

b = 1;

c = 0;

#10

a = 1;

b = 0;

c = 1;

#10

a = 1;

b = 0;

c = 0;

#10

a = 0;

b = 1;

c = 1;

#10

a = 0;

b = 1;

c = 0;

#10

a = 0;

b = 0;

c = 1;

#10

a = 0;

b = 0;

c = 0;

$finish; end endmodule

## // full\_adder using gates directly

module test; reg a,b,c;

wire e1,e2,e3,e4,e5; wire sum, carry; xor x1(e1,a,b);

xor x2(sum,e1,c);

and a1(e2,a,b);

and a2(e3,a,c);

and a3(e4,b,c);

or o1(e5,e3,e4); or o2(carry,e5,e2);

initial begin

$dumpfile("Experiment3.vcd");

$dumpvars(0,test);

$monitor("a = %b, b = %b, c = %b, sum = %b, carry = %b",a,b,c,sum,carry); a = 1;

b = 1;

c = 1;

#10

a = 1;

b = 1;

c = 0;

#10

a = 1;

b = 0;

c = 1;

#10

a = 1;

b = 0;

c = 0;

#10

a = 0;

b = 1;

c = 1;

#10

a = 0;

b = 1;

c = 0;

#10

a = 0;

b = 0;

c = 1;

#10

a = 0;

b = 0;

c = 0;

$finish; end endmodule

## // full\_adder using directly equations

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

assign carry = a&b | a&c | b&c; endmodule

module test; reg a,b,c;

wire sum, carry;

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

initial begin

$dumpfile("Experiment3.vcd");

$dumpvars(0,test);

$monitor("a = %b, b = %b, c = %b, sum = %b, carry = %b",a,b,c,sum,carry); a = 1;

b = 1;

c = 1;

#10

a = 1;

b = 1;

c = 0;

#10

a = 1;

b = 0;

c = 1;

#10

a = 1;

b = 0;

c = 0;

#10

a = 0;

b = 1;

c = 1;

#10

a = 0;

b = 1;

c = 0;

#10

a = 0;

b = 0;

c = 1;

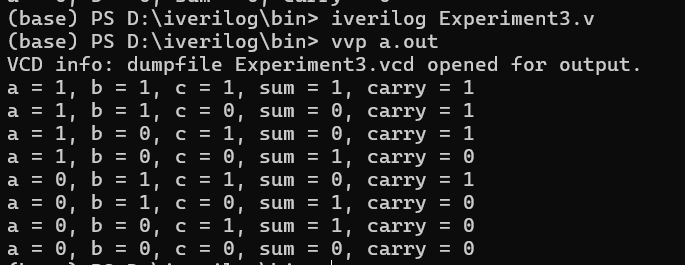
#10

a = 0;

b = 0;

c = 0;

$finish; end endmodule



# Experiment 4 (Half Subtractor)

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

**// full\_subtractor using two half subtractor**

module half\_subtractor(input a, input b, output diff, output borrow); assign diff = (a^b);

assign borrow = ~a&b; endmodule

// while using half\_subtractor or half\_adder for full\_adder and sub\_tractor always use carry and //borrow as two instance and then take or of these.

module test; reg a,b,c;

wire diff,borrow,e1,e2,e3,e4; half\_subtractor h1(a,b,e1,e2); half\_subtractor h2(e1,c,diff,e3); or o1(borrow,e3,e2);

initial begin

$dumpfile("Experiment4.vcd");

$dumpvars(0,test);

$monitor("a = %b, b = %b, c = %b, diff = %b, borrow = %b",a,b,c,diff,borrow); a = 1;

b = 1;

c = 1;

#10

a = 1;

b = 1;

c = 0;

#10

a = 1;

b = 0;

c = 1;

#10

a = 1;

b = 0;

c = 0;

#10

a = 0;

b = 1;

c = 1;

#10

a = 0;

b = 1;

c = 0;

#10

a = 0;

b = 0;

c = 1;

#10

a = 0;

b = 0;

c = 0;

$finish; end endmodule

## // full\_subtractor using directly equations

module full\_subtractor(input a, input b, input c, output diff, output borrow); assign diff = (a^b)^c;

assign borrow = c&(~(a^b)) | ~a&b; endmodule

module test; reg a,b,c;

wire diff,borrow;

full\_subtractor h1(.a(a),.b(b),.c(c),.diff(diff),.borrow(borrow));

initial begin

$dumpfile("Experiment4.vcd");

$dumpvars(0,test);

$monitor("a = %b, b = %b, c = %b, diff = %b, borrow = %b",a,b,c,diff,borrow); a = 1;

b = 1;

c = 1;

#10

a = 1;

b = 1;

c = 0;

#10

a = 1;

b = 0;

c = 1;

#10

a = 1;

b = 0;

c = 0;

#10

a = 0;

b = 1;

c = 1;

#10

a = 0;

b = 1;

c = 0;

#10

a = 0;

b = 0;

c = 1;

#10

a = 0;

b = 0;

c = 0;

$finish; end endmodule

## // half\_subtractor using directly equations

module half\_subtractor(input a, input b, output diff, output borrow); assign diff = a^b;

assign borrow = ~a&b; endmodule

module test; reg a,b;

wire diff,borrow;

half\_subtractor h1(.a(a),.b(b),.diff(diff),.borrow(borrow));

initial begin

$dumpfile("Experiment4.vcd");

$dumpvars(0,test);

$monitor("a = %b, b = %b, diff = %b, borrow = %b",a,b,diff,borrow); a = 1;

b = 1;

#10

a = 1;

b = 0;

#10

a = 0;

b = 1;

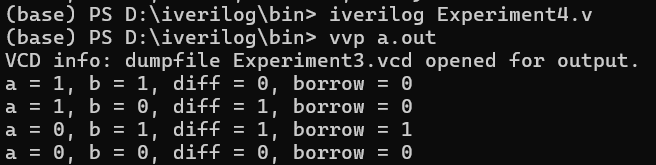
#10

a = 0;

b = 0;

$finish; end endmodule

\*/



# Experiment 5 (Number Converter)

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

**// BCD to excess 3 convertor**

module BCD\_Excess(input a, input b, input c, input d, output o1, output o2, output o3, output o4);

assign o1 = a | (b&d) | (c&b);

assign o2 = (~b&d) | (~b&c) | (~c&b)&(~d); assign o3 = (~c&(~d)) | (c&d);

assign o4 = (~c&(~d)) | (~d&c); endmodule

module test; reg a,b,c,d;

wire o1,o2,o3,o4;

BCD\_Excess b1(a,b,c,d,o1,o2,o3,o4); initial begin

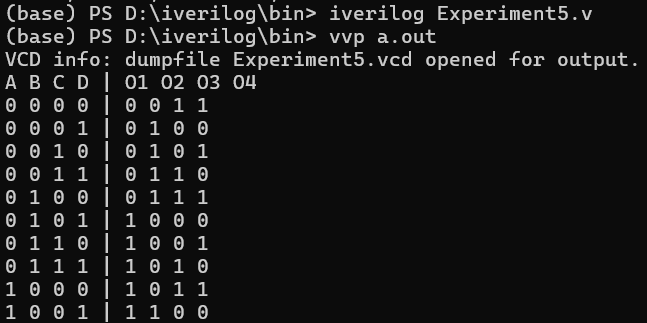
$dumpfile("Experiment5.vcd");

$dumpvars(0,test);

$display("A B C D | O1 O2 O3 O4");

$monitor("%b %b %b %b | %b %b %b %b",a,b,c,d,o1,o2,o3,o4); a = 0;b = 0;c = 0; d = 0; #10;

a = 0;b = 0;c = 0; d = 1; #10; a = 0;b = 0;c = 1; d = 0; #10; a = 0;b = 0;c = 1; d = 1; #10; a = 0;b = 1;c = 0; d = 0; #10; a = 0;b = 1;c = 0; d = 1; #10; a = 0;b = 1;c = 1; d = 0; #10; a = 0;b = 1;c = 1; d = 1; #10; a = 1;b = 0;c = 0; d = 0; #10; a = 1;b = 0;c = 0; d = 1; #10;

$finish; end endmodule

# Experiment 6 (Multiplexer)

## To design and implement a 4:1 multiplexer

**// 4:1 mutliplexer**

module mutliplexer(input a, input b, input c, input d, input s0, input s1, output o); assign o = (~s0&~s1&a) | (s0&~s1&b) |(~s0&s1&c) |(s0&s1&d);

endmodule module test;

reg a,b,c,d,s0,s1,e; wire o;

mutliplexer m1(a,b,c,d,s0,s1,o);

initial begin

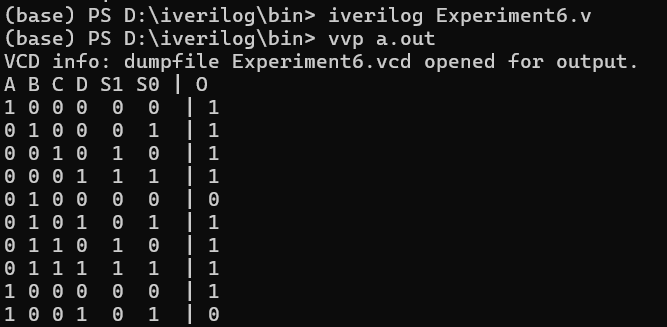
$dumpfile("Experiment6.vcd");

$dumpvars(0,test);

$display("A B C D S1 S0 | O");

$monitor("%b %b %b %b %b %b | %b ",a,b,c,d,s1,s0,o);

a = 1;b = 0;c = 0; d = 0; s1 = 0; s0 = 0; #10; a = 0;b = 1;c = 0; d = 0; s1 = 0; s0 = 1; #10; a = 0;b = 0;c = 1; d = 0; s1 = 1; s0 = 0; #10; a = 0;b = 0;c = 0; d = 1; s1 = 1; s0 = 1; #10; a = 0;b = 1;c = 0; d = 0; s1 = 0; s0 = 0; #10; a = 0;b = 1;c = 0; d = 1; s1 = 0; s0 = 1; #10; a = 0;b = 1;c = 1; d = 0; s1 = 1; s0 = 0; #10; a = 0;b = 1;c = 1; d = 1; s1 = 1; s0 = 1; #10; a = 1;b = 0;c = 0; d = 0; s1 = 0; s0 = 0; #10; a = 1;b = 0;c = 0; d = 1; s1 = 0; s0 = 1; #10;

$finish; end endmodule

# Experiment 7 (Demultiplexer)

## To design and implement a 1:4 demultiplexer.

**// demultiplexer**

module demultiplexer(output a, output b, output c, output d, input s0, input s1, input o,input e);

assign a = (~s0&~s1&o&e); assign b = (s0&~s1&o&e); assign c = (~s0&s1&o&e); assign d = (s0&s1&o&e); endmodule

// e is enable and o is input module test;

wire a,b,c,d;

reg s0,s1,o,e;

demultiplexer d1(a,b,c,d,s0,s1,o,e);

initial begin

$dumpfile("Experiment7.vcd");

$dumpvars(0,test);

$display("E S1 S0 | A B C D ");

$monitor("%b %b %b | %b %b %b %b ",e,s1,s0,a,b,c,d); o = 1;

e = 1;

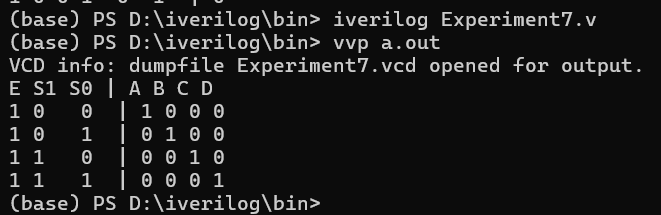
s1 = 0; s0 = 0; #10;

s1 = 0; s0 = 1; #10;

s1 = 1; s0 = 0; #10;

s1 = 1; s0 = 1; #10;

$finish; end endmodule



# Experiment 8 (Decoder)

## To design and verify a 2:4 decoder.

**// encoder**

module decoder(input a, input b, input c, input d, output s0, output s1); assign s0 = b|d;

assign s1 = c|d; endmodule

module test; reg a,b,c,d; wire s0,s1;

decoder d1(a,b,c,d,s0,s1);

initial begin

$dumpfile("Experiment8.vcd");

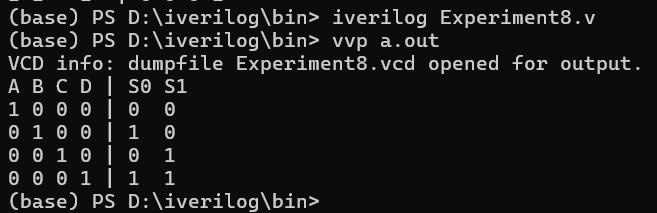
$dumpvars(0,test);

$display("A B C D | S0 S1");

$monitor("%b %b %b %b | %b %b ",a,b,c,d,s0,s1); a = 1;b = 0;c = 0; d = 0; #10;

a = 0;b = 1;c = 0; d = 0; #10; a = 0;b = 0;c = 1; d = 0; #10; a = 0;b = 0;c = 0; d = 1; #10;

$finish; end endmodule



# Experiment 9 (Encoder)

## To design and implement a 4:2 encoder.

**// decoder**

module encoder(output a, output b, output c, output d, input s0, input s1); assign a = (~s0&~s1);

assign b = (s0&~s1); assign c = (~s0&s1); assign d = (s0&s1); endmodule

module test; wire a,b,c,d; reg s0,s1,o;

encoder d1(a,b,c,d,s0,s1);

initial begin

$dumpfile("Experiment7.vcd");

$dumpvars(0,test);

$display("S1 S0 | A B C D ");

$monitor("%b %b | %b %b %b %b ",s1,s0,a,b,c,d); o = 1;

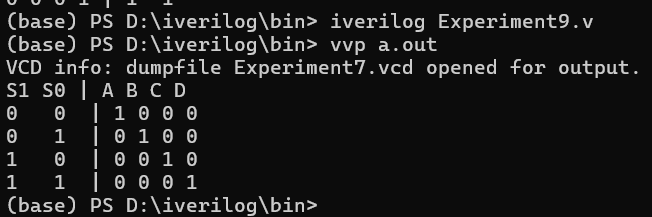
s1 = 0; s0 = 0; #10;

s1 = 0; s0 = 1; #10;

s1 = 1; s0 = 0; #10;

s1 = 1; s0 = 1; #10;

$finish; end endmodule



**Experiment 10 (Flip-Flops )**

**To design and verify the operation of D flip-flops using logic gates.**

module d\_flip\_flop (

input D, // Data input

input CLK, // Clock input

output reg Q, // Output

output Qn // Complemented output

);

assign Qn = ~Q; // Complementary output

always @(posedge CLK) begin

Q <= D; // Transfer D to Q on clock's rising edge

end

endmodule

module d\_flip\_flop\_tb;

reg D; // Data input

reg CLK; // Clock input

wire Q, Qn; // Outputs

// Instantiate the D Flip-Flop

d\_flip\_flop uut (

.D(D),

.CLK(CLK),

.Q(Q),

.Qn(Qn)

);

// Clock generation

initial begin

CLK = 0;

forever #5 CLK = ~CLK; // Toggle clock every 5 time units

end

// Test sequence

initial begin

$display("Time | D | CLK | Q | Qn");

$monitor("%4d | %b | %b | %b | %b", $time, D, CLK, Q, Qn);

D = 0; #10;

D = 1; #10;

D = 0; #10;

D = 1; #10;

$finish;

end

endmodule

**A screen shot of a computer

Description automatically generated**

**Experiment 11 (Flip-Flops )**

**To design and verify the operation of JK flip-flops using logic gates.**

module jk\_flip\_flop (

input J, // J input

input K, // K input

input CLK, // Clock input

output reg Q, // Output

output Qn // Complemented output

);

assign Qn = ~Q; // Complementary output

always @(posedge CLK) begin

if (J == 0 && K == 0) begin

Q <= Q; // No change

end

else if (J == 0 && K == 1) begin

Q <= 0; // Reset

end

else if (J == 1 && K == 0) begin

Q <= 1; // Set

end

else if (J == 1 && K == 1) begin

Q <= ~Q; // Toggle

end

end

endmodule

module jk\_flip\_flop\_tb;

reg J; // J input

reg K; // K input

reg CLK; // Clock input

wire Q, Qn; // Outputs

// Instantiate the JK Flip-Flop

jk\_flip\_flop uut (

.J(J),

.K(K),

.CLK(CLK),

.Q(Q),

.Qn(Qn)

);

// Clock generation

initial begin

CLK = 0;

forever #5 CLK = ~CLK; // Toggle clock every 5 time units

end

// Test sequence

initial begin

$display("Time | J | K | CLK | Q | Qn");

$monitor("%4d | %b | %b | %b | %b | %b", $time, J, K, CLK, Q, Qn);

// Initialize inputs

J = 0; K = 0; #10;

J = 0; K = 1; #10;

J = 1; K = 0; #10;

J = 1; K = 1; #10;

J = 0; K = 0; #10;

J = 1; K = 1; #10;

$finish;

end

endmodule

A screen shot of a computer

Description automatically generated

**Experiment 12 (Counter)**

**To verify the operation of asynchronous counter.**

module t\_flip\_flop (

input T, // Toggle input

input CLK, // Clock input

output reg Q // Output Q

);

always @(posedge CLK) begin

if (T)

Q <= ~Q; // Toggle output if T is high

end

endmodule

module async\_counter (

input CLK, // Clock input

output [3:0] Q // 4-bit counter output

);

wire Q1, Q2, Q3; // Intermediate flip-flop outputs

// Instantiate T flip-flops for each bit of the counter

t\_flip\_flop T1 (.T(1), .CLK(CLK), .Q(Q[0])); // First flip-flop toggles on every clock

t\_flip\_flop T2 (.T(1), .CLK(Q[0]), .Q(Q1)); // Second flip-flop toggles on Q0

t\_flip\_flop T3 (.T(1), .CLK(Q1), .Q(Q2)); // Third flip-flop toggles on Q1

t\_flip\_flop T4 (.T(1), .CLK(Q2), .Q(Q3)); // Fourth flip-flop toggles on Q2

assign Q = {Q3, Q2, Q1, Q[0]}; // Concatenate the outputs of the flip-flops to form the 4-bit counter

endmodule

module async\_counter\_tb;

reg CLK; // Clock input

wire [3:0] Q; // 4-bit counter output

// Instantiate the 4-bit asynchronous counter

async\_counter uut (

.CLK(CLK),

.Q(Q)

);

// Clock generation

initial begin

CLK = 0;

forever #5 CLK = ~CLK; // Toggle clock every 5 time units

end

// Test sequence

initial begin

$display("Time | CLK | Q");

$monitor("%4d | %b | %b", $time, CLK, Q);

// Simulate for some time to observe the counter's behavior

#100;

$finish;

end

endmodule

A screen shot of a computer

Description automatically generated