**UCS704 EMBEDDED SYSTEMS DESIGN**

**SUBMITTED BY**

**NAME : BHAVIKA JINDAL**

**ROLL NUMEBR: 102103628**

**SUB GROUP: 4CO23**

**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).

**Code :**

module logic\_gates\_test;

// Declare inputs

reg a, b;

// Declare outputs for each gate

wire not\_a;

wire and\_ab, or\_ab, nand\_ab, nor\_ab, xor\_ab, xnor\_ab;

// Instantiate the gates

not U1 (not\_a, a);

and U2 (and\_ab, a, b);

or U3 (or\_ab, a, b);

nand U4 (nand\_ab, a, b);

nor U5 (nor\_ab, a, b);

xor U6 (xor\_ab, a, b);

xnor U7 (xnor\_ab, a, b);

// Initial block to generate input combinations and display results

initial begin

$display("A B | NOT AND OR NAND NOR XOR XNOR");

$display("---------------------------------");

// Loop through all combinations of a and b

a = 0; b = 0; #10 display\_values();

a = 0; b = 1; #10 display\_values();

a = 1; b = 0; #10 display\_values();

a = 1; b = 1; #10 display\_values();

$finish;

end

// Task to display the current values of inputs and outputs

task display\_values;

$display("%b %b | %b %b %b %b %b %b %b", a, b, not\_a, and\_ab, or\_ab, nand\_ab, nor\_ab, xor\_ab, xnor\_ab);

endtask

endmodule

**OUTPUT :**

A black screen with white text

Description automatically generated

**For NOT gate input A has been considered**

**Experiment 2 (Half Adder)**

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

**Code :**

module half\_adder (

input x, // First input

input y, // Second input

output S, // Sum output

output C // Carry output

);

// Implement the sum and carry using the given equations

assign S = (x | y) & (~x | ~y); // S = (x + y)(x' + y')

assign C = x & y; // C = xy

endmodule

The Test Bench Code to verify :

module half\_adder\_tb;

reg x, y; // Test inputs

wire S, C; // Outputs from the half adder

// Instantiate the half adder module

half\_adder uut (

.x(x),

.y(y),

.S(S),

.C(C)

);

// Test all possible combinations of x and y

initial begin

$display("x y | S C"); // Header for the output table

$display("-------------");

x = 0; y = 0; #10 $display("%b %b | %b %b", x, y, S, C);

x = 0; y = 1; #10 $display("%b %b | %b %b", x, y, S, C);

x = 1; y = 0; #10 $display("%b %b | %b %b", x, y, S, C);

x = 1; y = 1; #10 $display("%b %b | %b %b", x, y, S, C);

$finish;

end

endmodule

A screen shot of a computer code

Description automatically generated

**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

**Code :**

module full\_adder (

input x, // First input

input y, // Second input

input z, // Third input (carry-in)

output S, // Sum output

output C // Carry output

);

// Implement the sum and carry using the given equations

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

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

endmodule

**Test-Bench Code :**

module full\_adder\_tb;

reg x, y, z; // Test inputs

wire S, C; // Outputs from the full adder

// Instantiate the full adder module

full\_adder uut (

.x(x),

.y(y),

.z(z),

.S(S),

.C(C)

);

// Test all possible combinations of x, y, and z

initial begin

$display("x y z | S C"); // Header for the output table

$display("---------------");

x = 0; y = 0; z = 0; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

x = 0; y = 0; z = 1; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

x = 0; y = 1; z = 0; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

x = 0; y = 1; z = 1; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

x = 1; y = 0; z = 0; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

x = 1; y = 0; z = 1; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

x = 1; y = 1; z = 0; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

x = 1; y = 1; z = 1; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

$finish;

end

endmodule

A screen shot of a computer

Description automatically generated

**Experiment 4 (Half Subtractor)**

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

**Code :**

module half\_subtractor (

input x, // Minuend

input y, // Subtrahend

output D, // Difference output

output B // Borrow output

);

// Implement the difference and borrow using the given equations

assign D = (~x & y) | (x & ~y); // D = x'y + xy'

assign B = ~x & y; // B = x'y

endmodule

**TestBench Code**

module half\_subtractor\_tb;

reg x, y; // Test inputs

wire D, B; // Outputs from the half subtractor

// Instantiate the half subtractor module

half\_subtractor uut (

.x(x),

.y(y),

.D(D),

.B(B)

);

// Test all possible combinations of x and y

initial begin

$display("x y | D B"); // Header for the output table

$display("-------------");

x = 0; y = 0; #10 $display("%b %b | %b %b", x, y, D, B);

x = 0; y = 1; #10 $display("%b %b | %b %b", x, y, D, B);

x = 1; y = 0; #10 $display("%b %b | %b %b", x, y, D, B);

x = 1; y = 1; #10 $display("%b %b | %b %b", x, y, D, B);

$finish;

end

endmodule

**A screen shot of a computer

Description automatically generated**

**Experiment 5 (Number Converter)**

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

**Code :**

module bcd\_to\_excess3 (

input A, B, C, D, // BCD input

output W, X, Y, Z // Excess-3 output

);

// Implement the Excess-3 outputs using the derived Boolean expressions

assign W = A | (B & (C | D)); // W = A + B(C + D)

assign X = (~B & D) | (~B & C) | (B & ~D); // X = B'D + B'C + BD'

assign Y = C ^ D; // Y = C XOR D

assign Z = ~D; // Z = D'

endmodule

**Test Bench Code**

module bcd\_to\_excess3\_tb;

reg A, B, C, D; // BCD inputs

wire W, X, Y, Z; // Excess-3 outputs

// Instantiate the converter module

bcd\_to\_excess3 uut (

.A(A),

.B(B),

.C(C),

.D(D),

.W(W),

.X(X),

.Y(Y),

.Z(Z)

);

// Apply test cases

initial begin

$display(" A B C D | W X Y Z ");

$display("--------------------------");

A = 0; B = 0; C = 0; D = 0; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 0

A = 0; B = 0; C = 0; D = 1; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 1

A = 0; B = 0; C = 1; D = 0; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 2

A = 0; B = 0; C = 1; D = 1; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 3

A = 0; B = 1; C = 0; D = 0; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 4

A = 0; B = 1; C = 0; D = 1; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 5

A = 0; B = 1; C = 1; D = 0; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 6

A = 0; B = 1; C = 1; D = 1; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 7

A = 1; B = 0; C = 0; D = 0; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 8

A = 1; B = 0; C = 0; D = 1; #10 $display(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z); // 9

$finish;

end

endmodule

**A screen shot of a computer

Description automatically generated**

**Experiment 6 (Multiplexer)**

To design and implement a 4:1 multiplexer

**Code :**

module mux\_4\_to\_1 (

input I0, I1, I2, I3, // Data inputs

input S0, S1, // Selection lines

output Y // Output

);

// Implement the 4:1 MUX logic

assign Y = (~S1 & ~S0 & I0) | (~S1 & S0 & I1) | (S1 & ~S0 & I2) | (S1 & S0 & I3);

endmodule

**Test-Bench Code**

module mux\_4\_to\_1\_tb;

reg I0, I1, I2, I3; // Data inputs

reg S0, S1; // Selection lines

wire Y; // Output from the MUX

// Instantiate the 4:1 MUX

mux\_4\_to\_1 uut (

.I0(I0),

.I1(I1),

.I2(I2),

.I3(I3),

.S0(S0),

.S1(S1),

.Y(Y)

);

// Test all combinations

initial begin

$display("S1 S0 | I0 I1 I2 I3 | Y ");

$display("-----------------------");

I0 = 1; I1 = 0; I2 = 1; I3 = 0;

S1 = 0; S0 = 0; #10 $display(" %b %b | %b %b %b %b | %b", S1, S0, I0, I1, I2, I3, Y);

S1 = 0; S0 = 1; #10 $display(" %b %b | %b %b %b %b | %b", S1, S0, I0, I1, I2, I3, Y);

S1 = 1; S0 = 0; #10 $display(" %b %b | %b %b %b %b | %b", S1, S0, I0, I1, I2, I3, Y);

S1 = 1; S0 = 1; #10 $display(" %b %b | %b %b %b %b | %b", S1, S0, I0, I1, I2, I3, Y);

$finish;

end

endmodule

**A computer screen shot of a black screen

Description automatically generated**

**Experiment 7 (Demultiplexer)**

To design and implement a 1:4 demultiplexer.

**Code :**

module demux\_1\_to\_4 (

input D, // Data input

input S0, S1, // Selection lines

output Y0, Y1, Y2, Y3 // Outputs

);

// Implement the 1:4 DEMUX logic

assign Y0 = ~S1 & ~S0 & D;

assign Y1 = ~S1 & S0 & D;

assign Y2 = S1 & ~S0 & D;

assign Y3 = S1 & S0 & D;

endmodule

**Test Bench Code**

module demux\_1\_to\_4\_tb;

reg D; // Data input

reg S0, S1; // Selection lines

wire Y0, Y1, Y2, Y3; // Outputs

// Instantiate the 1:4 DEMUX

demux\_1\_to\_4 uut (

.D(D),

.S0(S0),

.S1(S1),

.Y0(Y0),

.Y1(Y1),

.Y2(Y2),

.Y3(Y3)

);

// Test all combinations

initial begin

$display("S1 S0 | D | Y0 Y1 Y2 Y3");

$display("-----------------------");

D = 1;

S1 = 0; S0 = 0; #10 $display(" %b %b | %b | %b %b %b %b", S1, S0, D, Y0, Y1, Y2, Y3);

S1 = 0; S0 = 1; #10 $display(" %b %b | %b | %b %b %b %b", S1, S0, D, Y0, Y1, Y2, Y3);

S1 = 1; S0 = 0; #10 $display(" %b %b | %b | %b %b %b %b", S1, S0, D, Y0, Y1, Y2, Y3);

S1 = 1; S0 = 1; #10 $display(" %b %b | %b | %b %b %b %b", S1, S0, D, Y0, Y1, Y2, Y3);

$finish;

end

endmodule

A screen shot of a computer

Description automatically generated

**Experiment 8 (Decoder)**

To design and verify a 2:4 decoder.

**Code :**

module decoder\_2\_to\_4 (

input A, B, // Inputs

output Y0, Y1, Y2, Y3 // Outputs

);

// Implement the 2:4 Decoder logic

assign Y0 = ~A & ~B;

assign Y1 = ~A & B;

assign Y2 = A & ~B;

assign Y3 = A & B;

endmodule

**Test Bench Code :**

module decoder\_2\_to\_4\_tb;

reg A, B; // Inputs

wire Y0, Y1, Y2, Y3; // Outputs

// Instantiate the 2:4 Decoder

decoder\_2\_to\_4 uut (

.A(A),

.B(B),

.Y0(Y0),

.Y1(Y1),

.Y2(Y2),

.Y3(Y3)

);

// Test all combinations

initial begin

$display("A B | Y0 Y1 Y2 Y3");

$display("-----------------");

A = 0; B = 0; #10 $display("%b %b | %b %b %b %b", A, B, Y0, Y1, Y2, Y3);

A = 0; B = 1; #10 $display("%b %b | %b %b %b %b", A, B, Y0, Y1, Y2, Y3);

A = 1; B = 0; #10 $display("%b %b | %b %b %b %b", A, B, Y0, Y1, Y2, Y3);

A = 1; B = 1; #10 $display("%b %b | %b %b %b %b", A, B, Y0, Y1, Y2, Y3);

$finish;

end

endmodule

A screen shot of a computer

Description automatically generated

**Experiment 9 (Encoder)**

To design and implement a 4:2 encoder.

**Code :**

module encoder\_4\_to\_2 (

input I0, I1, I2, I3, // Inputs

output Y0, Y1 // Outputs

);

// Implement the 4:2 Encoder logic

assign Y0 = I1 | I3;

assign Y1 = I2 | I3;

endmodule

**Test Bench Code**

module encoder\_4\_to\_2\_tb;

reg I0, I1, I2, I3; // Inputs

wire Y0, Y1; // Outputs

// Instantiate the 4:2 Encoder

encoder\_4\_to\_2 uut (

.I0(I0),

.I1(I1),

.I2(I2),

.I3(I3),

.Y0(Y0),

.Y1(Y1)

);

// Test all input combinations with only one input high

initial begin

$display("I3 I2 I1 I0 | Y1 Y0");

$display("-------------------");

I0 = 1; I1 = 0; I2 = 0; I3 = 0; #10 $display("%b %b %b %b | %b %b", I3, I2, I1, I0, Y1, Y0);

I0 = 0; I1 = 1; I2 = 0; I3 = 0; #10 $display("%b %b %b %b | %b %b", I3, I2, I1, I0, Y1, Y0);

I0 = 0; I1 = 0; I2 = 1; I3 = 0; #10 $display("%b %b %b %b | %b %b", I3, I2, I1, I0, Y1, Y0);

I0 = 0; I1 = 0; I2 = 0; I3 = 1; #10 $display("%b %b %b %b | %b %b", I3, I2, I1, I0, Y1, Y0);

$finish;

end

endmodule

A screen shot of a computer code

Description automatically generated

**Experiment 10 (Flip-Flops )**

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

**Code :**

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

**Test Bench Code**

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.

**Code :**

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

**Test Bench Code**

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.

**Code :**

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

**Test Bench Code**

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