**UCS704**

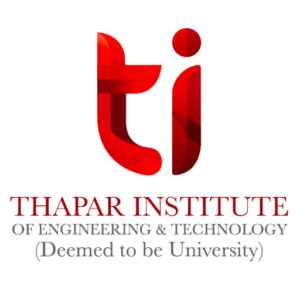
**EMBEDDED SYSTEMS DESIGN**

**Submitted By:**

**Divesh Goel (102103147)**

**COE-5**

**BE-4th Year**



**Thapar institute of Engineering and Technology, Patiala**

**June - December 2024**

**INDEX**

|  |  |  |
| --- | --- | --- |
| S no. | Experiment Name | Page No. |
| 1 | To study and verify the truth table of various logic gates (NOT, AND, OR, NAND, NOR, EX-OR, & EX-NOR). | 3-4 |
| 2 | To design and verify a half adder using S= (x+y)(x’+y’) C= xy | 4-5 |
| 3 | To design and verify a full adder using S = x’y’z+x’yz’+xy’z’+xyz C=xy+xz+yz | 5-6 |
| 4 | To design and verify a half subtractor using D = x’y +xy’ B=x’y | 6-7 |
| 5 | Design a BCD to Excess 3 code converter using combinational circuits. | 7-8 |
| 6 | To design and implement a 4:1 multiplexer | 8-9 |
| 7 | To design and implement a 1:4 demultiplexer. | 9-10 |
| 8 | To design and verify a 2:4 decoder. | 10-11 |
| 9 | To design and implement a 4:2 encoder. | 11-12 |
| 10 | To design and verify the operation of D flip-flops using logic gates. | 12-13 |
| 11 | To design and verify the operation of JK flip-flops using logic gates. | 13-14 |
| 12 | To verify the operation of an asynchronous counter. | 14-15 |

**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).

module logic\_gates;

reg a, b;

wire not\_a, and\_ab, or\_ab, nand\_ab, nor\_ab, xor\_ab, xnor\_ab;

not (not\_a, a);

and (and\_ab, a, b);

or (or\_ab, a, b);

nand (nand\_ab, a, b);

nor (nor\_ab, a, b);

xor (xor\_ab, a, b);

xnor (xnor\_ab, a, b);

initial begin

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

a = 0; b = 0; #1 $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);

a = 0; b = 1; #1 $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);

a = 1; b = 0; #1 $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);

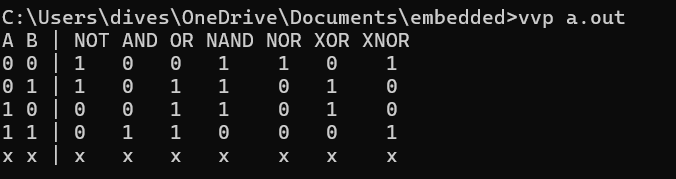
a = 1; b = 1; #1 $display("%b %b | %b %b %b %b %b %b %b", a, b, not\_a, and\_ab, or\_ab, nand\_ab, nor\_ab, xor\_ab, xnor\_ab);

a = 1'bx; b = 1'bx; #1 $display("x x | %b %b %b %b %b %b %b", not\_a, and\_ab, or\_ab, nand\_ab, nor\_ab, xor\_ab, xnor\_ab);

end

endmodule

**Console Output:**



**Experiment 2 (Half Adder)**

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

module half\_adder;

reg x, y;

wire S, C;

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

assign C = x & y;

initial begin

$display("X Y | S C");

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

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

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

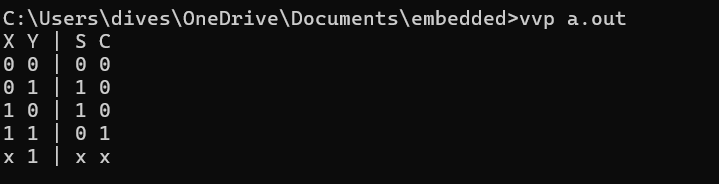
x = 1; y = 1; #1 $display("%b %b | %b %b", x, y, S, C);

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

end

endmodule

**Console Output:**



**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

module full\_adder;

reg x, y, z;

wire S, C;

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

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

initial begin

$display("X Y Z | S C");

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

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

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

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

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

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

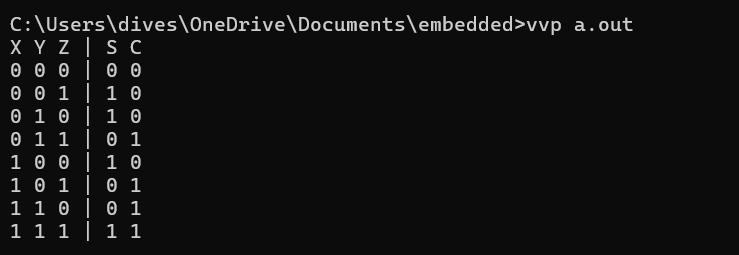
x = 1; y = 1; z = 0; #1 $display("%b %b %b | %b %b", x, y, z, S, C);

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

end

endmodule

**Console Output:**



**Experiment 4 (Half Subtractor)**

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

module half\_subtractor;

reg x, y;

wire D, B;

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

assign B = ~x & y;

initial begin

$display("X Y | D B");

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

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

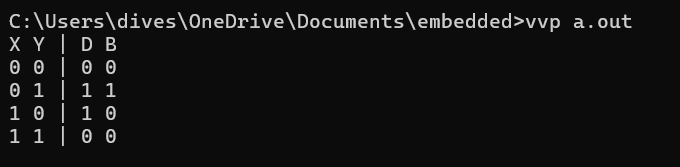
x = 1; y = 0; #1 $display("%b %b | %b %b", x, y, D, B);

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

end

endmodule

**Console Output:**



**Experiment 5 (Number Converter)**

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

module bcd\_to\_excess3;

reg [3:0] bcd;

wire [3:0] excess3;

assign excess3 = (bcd <= 4'b1001) ? (bcd + 4'b0011) : 4'bxxxx;

initial begin

$display("BCD | Excess-3");

for (bcd = 4'b0000; bcd <= 4'b1111; bcd = bcd + 1) begin

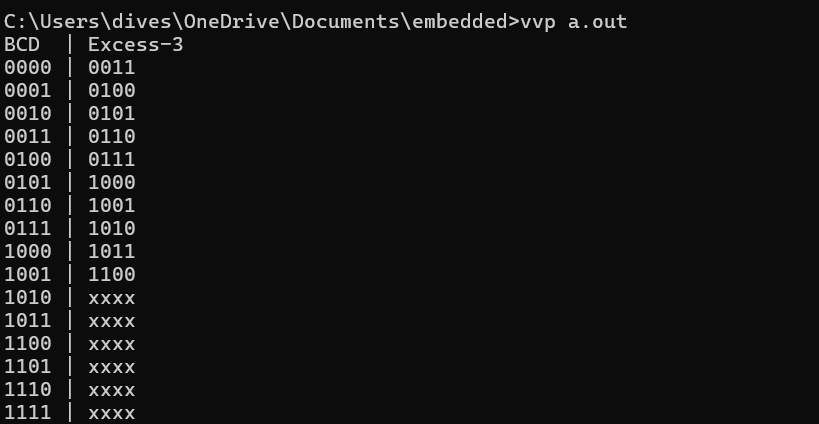
#1 $display("%b | %b", bcd, excess3);

end

end

endmodule

**Console Output:**



**Experiment 6 (Multiplexer)** To design and implement a 4:1 multiplexer

module mux\_4to1;

reg [3:0] in;

reg [1:0] sel;

wire out;

assign out = (sel == 2'b00) ? in[0] :

(sel == 2'b01) ? in[1] :

(sel == 2'b10) ? in[2] :

(sel == 2'b11) ? in[3] : 1'bx;

initial begin

$display("Sel | Inputs | Out");

in = 4'b1010;

for (sel = 0; sel < 4; sel = sel + 1) begin

#1 $display("%b | %b | %b", sel, in, out);

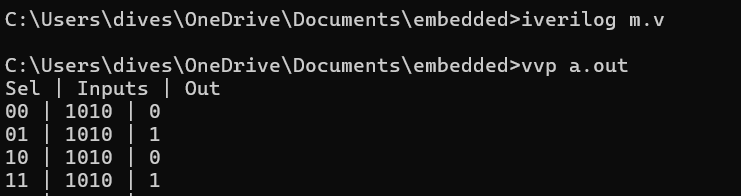
end

sel = 2'bx; #1 $display("x | %b | %b", in, out);

end

endmodule

**Console Output:**



**Experiment 7 (Demultiplexer)** To design and implement a 1:4 demultiplexer.

module demux\_1to4;

reg in;

reg [1:0] sel;

wire [3:0] out;

assign out[0] = (sel == 2'b00) ? in : 1'b0;

assign out[1] = (sel == 2'b01) ? in : 1'b0;

assign out[2] = (sel == 2'b10) ? in : 1'b0;

assign out[3] = (sel == 2'b11) ? in : 1'b0;

initial begin

$display("Sel In | Out");

in = 1;

for (sel = 0; sel < 4; sel = sel + 1) begin

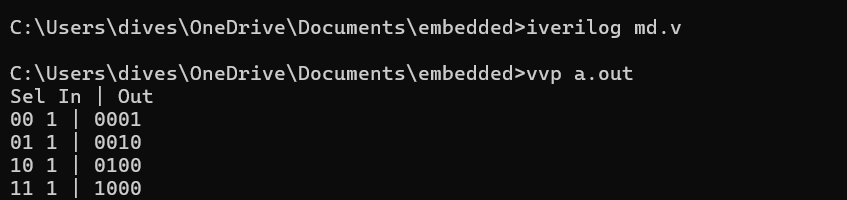
#1 $display("%b %b | %b", sel, in, out);

end

end

endmodule

**Console Output:**



**Experiment 8 (Decoder)** To design and verify a 2:4 decoder.

module decoder\_2to4;

reg [1:0] in;

wire [3:0] out;

assign out[0] = ~in[1] & ~in[0];

assign out[1] = ~in[1] & in[0];

assign out[2] = in[1] & ~in[0];

assign out[3] = in[1] & in[0];

initial begin

$display("In | Out");

for (in = 0; in < 4; in = in + 1) begin

#1 $display("%b | %b", in, out);

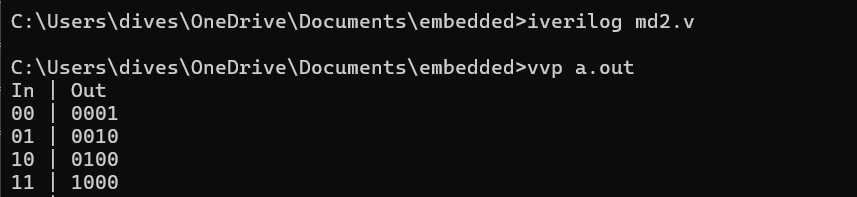
end

#1 $finish;

end

endmodule

**Console Output:**



**Experiment 9 (Encoder)** To design and implement a 4:2 encoder.

module encoder\_4to2;

reg [3:0] in;

reg [1:0] out;

always @(\*) begin

case (in)

4'b0001: out = 2'b00;

4'b0010: out = 2'b01;

4'b0100: out = 2'b10;

4'b1000: out = 2'b11;

default: out = 2'bxx;

endcase

end

initial begin

$display("In | Out");

in = 4'b0001; #1 $display("%b | %b", in, out);

in = 4'b0010; #1 $display("%b | %b", in, out);

in = 4'b0100; #1 $display("%b | %b", in, out);

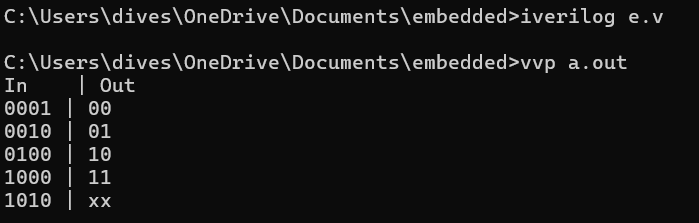
in = 4'b1000; #1 $display("%b | %b", in, out);

in = 4'b1010; #1 $display("%b | %b", in, out);

end

endmodule

**Console Output:**



**Experiment 10 (Flip-Flops)** To design and verify the operation of D flip-flops using logic gates.

module d\_flipflop;

reg D, clk;

reg Q;

always @(posedge clk) Q <= D;

initial begin

clk = 0;

D = 0;

#1 clk = 1; #1 $display("D=%b clk=%b | Q=%b", D, clk, Q);

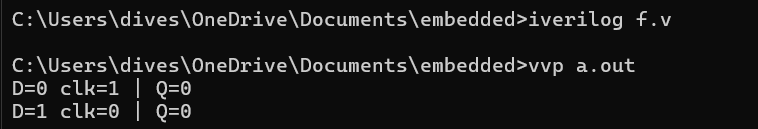
D = 1;

#1 clk = 0; #1 $display("D=%b clk=%b | Q=%b", D, clk, Q);

end

endmodule

**Console Output:**



**Experiment 11 (Flip-Flops )** To design and verify the operation of JK flip-flops using logic gates.

module jk\_flipflop;

reg J, K, clk;

wire Q;

reg Q\_int;

always @(posedge clk) begin

if (J & ~K) Q\_int <= 1;

else if (~J & K) Q\_int <= 0;

else if (J & K) Q\_int <= ~Q\_int;

end

assign Q = Q\_int;

initial begin

clk = 0;

Q\_int = 0;

$display("J K clk | Q");

J = 0; K = 0; #1 clk = 1; #1 $display("%b %b %b | %b", J, K, clk, Q);

J = 0; K = 1; #1 clk = 0; #1 clk = 1; #1 $display("%b %b %b | %b", J, K, clk, Q);

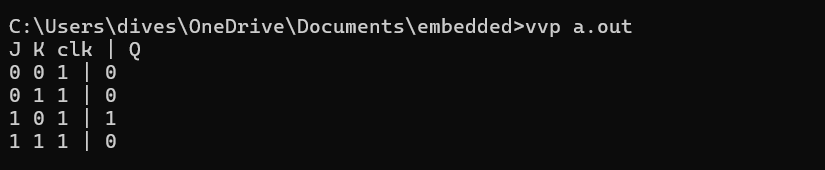
J = 1; K = 0; #1 clk = 0; #1 clk = 1; #1 $display("%b %b %b | %b", J, K, clk, Q);

J = 1; K = 1; #1 clk = 0; #1 clk = 1; #1 $display("%b %b %b | %b", J, K, clk, Q);

end

endmodule

**Console Output:**



**Experiment 12 (Counter)**

To verify the operation of an asynchronous counter.

module async\_counter(input clk, output reg [3:0] Q);

initial begin

Q = 4'b0000;

end

always @(posedge clk) begin

Q <= Q + 1;

end

endmodule

module async\_counter\_tb;

reg clk;

wire [3:0] Q;

async\_counter uut (.clk(clk), .Q(Q));

initial begin

clk = 0;

$display("Count");

forever begin

#5 clk = ~clk;

end

end

initial begin

forever begin

#6 $display("%b", Q);

end

end

endmodule

**Console Output:**

