**Name: -** Rohit Dilip Baruah

**ID NO: -** 21EL015

**Division: -** 04

**Year: -** 2023-24

**Subject: -** Digital System Design (3EL42)

**Branch: -** Electronics

**ASSIGNMENT 2**

[1] Design 4-bit Ripple Carry Adder with the help of 1-bit adder.

module full\_adder(

input a, input b, input cin, output sum, output carry

);

assign sum = a ^ b ^ cin;

assign carry = (a & b) | (cin & (a ^ b));

endmodule

module rca\_4bit( output [3:0] s, output cout, input [3:0] a, input [3:0] b, input cin

);

wire c0, c1, c2, c3; full\_adder fa0(s[0], c0, a[0], b[0], cin); full\_adder fa1(s[1], c1, a[1], b[1], c0); full\_adder fa2(s[2], c2, a[2], b[2], c1); full\_adder fa3(s[3], c3, a[3], b[3], c2); assign cout = c3;

endmodule

//Testbench//

module testbench;

reg [3:0] a; reg [3:0] b; reg cin; wire [3:0] s; wire cout;

rca\_4bit rca\_inst(

.s(s),

.cout(cout),

.a(a),

.b(b),

.cin(cin) written by 21EL015

);

initial begin

$dumpfile("testbench.vcd"); $dumpvars(0,testbench); end

initial begin

$display("a b cin | s cout");

$monitor("%b %b %b | %b %b", a, b, cin, s, cout);

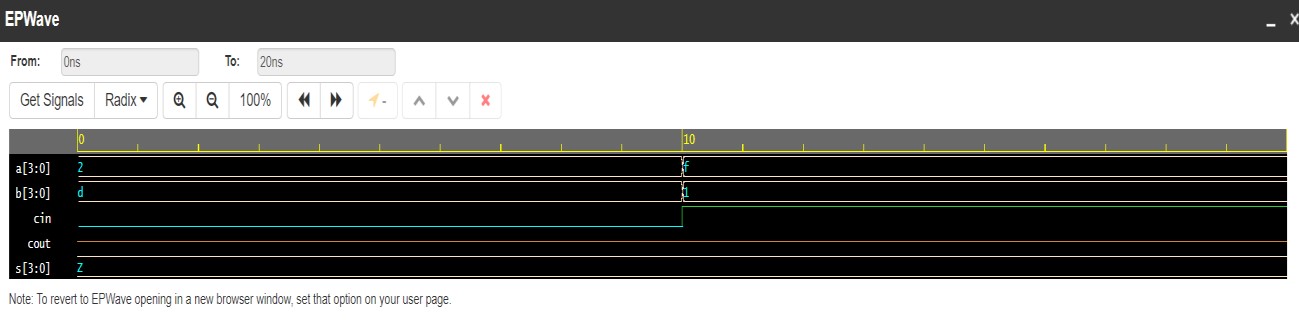
a = 4'b0010; b = 4'b1101; cin = 0;

#10;

a = 4'b1111; b = 4'b0001; cin = 1;

#10;

$finish; end endmodule



2] Design D-flipflop and reuse it to implement 4- bit Johnson Counter.

**Dataflow modelling:**

module Johnson\_Counter (

input wire clk, input wire reset,

output reg [3:0] counter\_output

);

reg [3:0] q\_temp;

always @(posedge clk or posedge reset) begin if (reset)

q\_temp <= 4'b0001; // Initial value else

q\_temp <= {q\_temp[2:0], q\_temp[3]};

end

assign counter\_output = q\_temp;

endmodule

//Testbench//

module testbench;

reg clk; reg reset; wire [3:0] counter\_output;

Johnson\_Counter dut (

.clk(clk),

.reset(reset),

.counter\_output(counter\_output)

);

initial begin

$dumpfile("testbench.vcd"); $dumpvars(0,testbench); end

always begin

#5 clk = ~clk; // Toggle the clock every 5 time units end written by 21EL015

initial begin clk = 0;

reset = 1; // Start with reset active

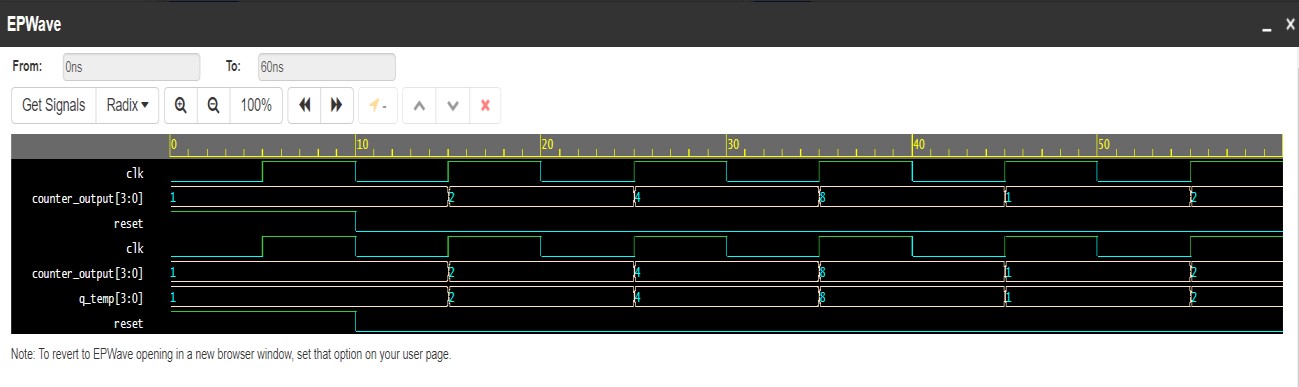
#10 reset = 0; // Deactivate reset after 10 time units

// Wait for a few clock cycles to observe the counter sequence

#50;

$finish; // End simulation end

endmodule



[3] Reuse 2:1 Mux code to implement 8:1 Mux.

module Mux2to1 (

input wire sel, input wire d0, input wire d1, output wire y

);

assign y = (sel == 1'b0) ? d0 : d1;

endmodule

module Mux8to1 (

input wire [2:0] sel, input wire [7:0] d, output wire y

);

wire [1:0] sel0, sel1; wire [3:0] d0, d1; wire [1:0] d2;

// First stage of 2:1 muxes

Mux2to1 mux0 (

.sel(sel[0]),

.d0(d[0]),

.d1(d[1]),

.y(d0[0])

);

Mux2to1 mux1 (

.sel(sel[0]),

.d0(d[2]),

.d1(d[3]),

.y(d0[1])

);

Mux2to1 mux2 (

.sel(sel[0]),

.d0(d[4]),

.d1(d[5]),

.y(d0[2])

Mux2to1 mux3 (

.sel(sel[0]),

.d0(d[6]),

.d1(d[7]),

.y(d0[3])

);

// Second stage of 2:1 muxes

Mux2to1 mux4 (

.sel(sel[1]),

.d0(d0[0]),

.d1(d0[1]),

.y(d1[0])

);

Mux2to1 mux5 (

.sel(sel[1]),

.d0(d0[2]),

.d1(d0[3]),

.y(d1[1])

Mux2to1 mux6 (

.sel(sel[1]),

.d0(d0[0]),

.d1(d0[1]),

.y(d1[2])

);

Mux2to1 mux7 (

.sel(sel[1]),

.d0(d0[2]),

.d1(d0[3]),

.y(d1[3])

);

// Third stage of 2:1 muxes

Mux2to1 mux8 (

.sel(sel[2]),

.d0(d1[0]),

.d1(d1[1]),

.y(y)

endmodule

//Testbench//

module testbench();

reg [2:0] sel; reg [7:0] d; wire y;

Mux8to1 dut (

.sel(sel),

.d(d),

.y(y)

);

initial begin

$dumpfile("testbench.vcd"); $dumpvars(0,testbench); end

initial begin // Initialize inputs

sel = 3'b000; // Select input 0 (000 in binary) d = 8'b10101010; // Arbitrary data input

// Wait for a few time units before changing inputs

#10;

// Test with different selection values sel = 3'b001; // Select input 1 (001 in binary)

#10;

sel = 3'b010; // Select input 2 (010 in binary)

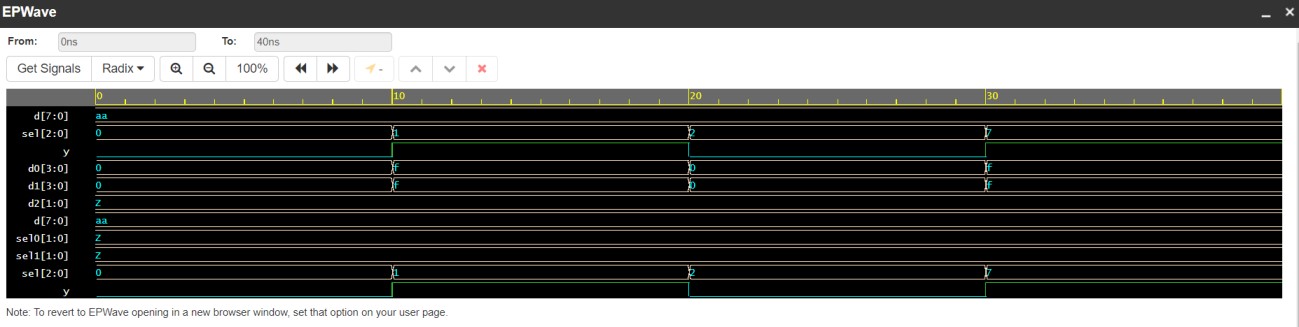
#10;

sel = 3'b111; // Select input 7 (111 in binary)

#10;

$finish; // End simulation end

endmodule



[4] Design a Full Subtractor with Gate Level Modeling Style.(use primitive gates)

module full\_subtractor\_gate\_level(

input x, input y, input z, output difference, output borrow

);

xor\_21 x1(x,y,k1); xor\_21 x2(k1,z,difference);

and\_21 a1(~x,y,k2); and\_21 a2(~x,z,k3); and\_21 a3(y,z,k4); or\_21 o1(k2,k3,k5); or\_21 o2(k5,k4,borrow);

endmodule

//Testbench// written by 21EL015

module xor\_21 ( input wire a, input wire b, output wire y

);

assign y = a ^ b;

endmodule

module and\_21 ( input wire a, input wire b, output wire y

);

assign y = a & b;

endmodule

module or\_21 ( input wire a, input wire b, output wire y

);

assign y = a | b;

endmodule

module testbench;

reg x, y, z; wire difference, borrow;

full\_subtractor\_gate\_level uut(

.x(x),

.y(y),

.z(z),

.difference(difference),

.borrow(borrow)

);

initial begin

$dumpfile("testbench.vcd"); $dumpvars(0, testbench);

// Initialize inputs x = 0; y = 0; z = 0;

// Wait for a few time units before changing inputs

#10;

// Change inputs and observe outputs x = 1; y = 0; z = 0;

#10;

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

#10;

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

#10;

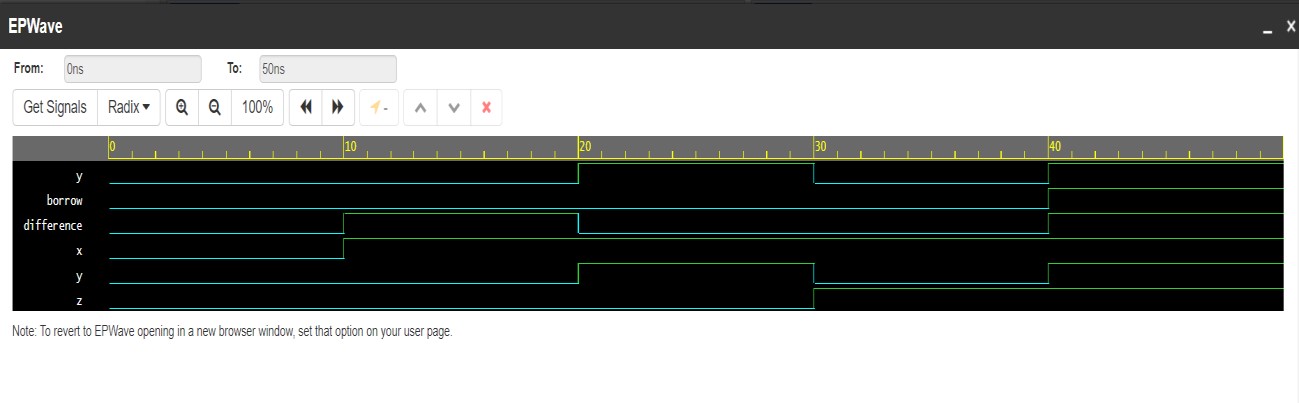
x = 1; y = 1; z = 1;

#10;

// End simulation

$finish; end

endmodule



[5] Design a 2X4 decoder using gate level modelling.

module decoder\_24\_gate\_level(

input x1, input x2, output y1, output y2, output y3, output y4

);

and a1(y1,~x1,~x2); and a2(y2,~x1,x2); and a3(y3,x1,~x2); and a4(y4,x1,x2);

endmodule

## //Testbench//

module decoder\_24\_tb;

reg x1,x2; wire y1,y2,y3,y4;

initial begin

$monitor(" %t | x1 = %b | x2 = %b | y1 = %b | y2 = %b | y3 = %b | y4

= %b ",$time,x1,x2,y1,y2,y3,y4);

end

decoder\_24\_gate\_level uut(x1,x2,y1,y2,y3,y4); initial begin

#000 x1=0; x2=0;

#100 x1=0; x2=1;

#100 x1=1; x2=0;

#100 x1=1; x2=1;

#100 $finish;

end

initial begin

$dumpfile("dump.vcd");

$dumpvars(0);

end

endmodule

[6] Design a 4x1 mux using operators. (use data flow)

module Mux4x1 (

input wire [3:0] data, input wire [1:0] sel, output wire out

);

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

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

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

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

Endmodule

//Testbench//

module testbench\_mux4x1;

reg [3:0] data; reg [1:0] sel; wire out;

Mux4x1 uut (

.data(data),

.sel(sel),

.out(out)

);

initial begin

$dumpfile("testbench\_mux4x1.vcd");

$dumpvars(0, testbench\_mux4x1);

// Initialize inputs

data = 4'b0000; sel = 2'b00;

// Wait for a few time units before changing inputs

#10;

// Test with different selection values sel = 2'b01;

#10;

sel = 2'b10;

#10;

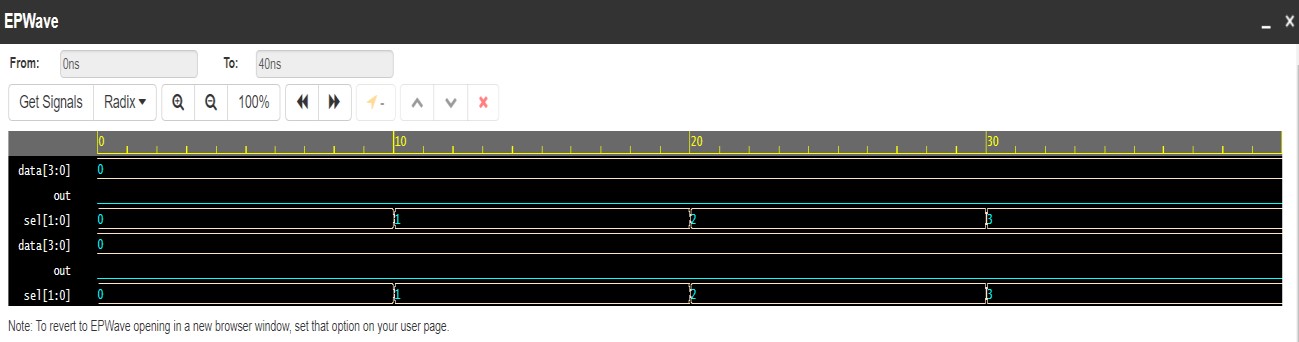
sel = 2'b11;

#10;

// End simulation

$finish; end

endmodule



[7] Design a Full adder using half adder.

module Half\_Adder ( input wire A, input wire B, output wire Sum, output wire Carry

);

assign Sum = A ^ B; assign Carry = A & B;

endmodule

module Full\_Adder ( input wire A, input wire B, input wire Cin, output wire Sum, output wire Cout

);

wire HA\_Sum1, HA\_Carry1, HA\_Carry2; written by 21EL015

Half\_Adder HA1 (.A(A), .B(B), .Sum(HA\_Sum1), .Carry(HA\_Carry1));

Half\_Adder HA2 (.A(HA\_Sum1), .B(Cin), .Sum(Sum), .Carry(HA\_Carry2));

assign Cout = HA\_Carry1 | HA\_Carry2;

endmodule

## //Testbench//

module testbench;

reg A, B, Cin; wire Sum, Cout;

Full\_Adder uut (

.A(A),

.B(B),

.Cin(Cin),

.Sum(Sum),

.Cout(Cout)

);

initial begin

$dumpfile("testbench.vcd");

$dumpvars(0, testbench);

// Test cases

A = 0; B = 0; Cin = 0;

#10;

A = 0; B = 0; Cin = 1;

#10;

A = 0; B = 1; Cin = 0;

#10;

A = 0; B = 1; Cin = 1;

#10;

A = 1; B = 0; Cin = 0;

#10;

A = 1; B = 0; Cin = 1;

#10;

A = 1; B = 1; Cin = 0;

#10;

A = 1; B = 1; Cin = 1;

#10;

$finish; end

endmodule

